PADERBORN UNIVERSITY
*The University for the Information Society*

Faculty for Computer Science, Electrical Engineering and Mathematics

# A Value-Centered Software Engineering Approach for Unique and Novel Software-Based Solutions

## Aligning Design Thinking with a Coopetition-Based Evolutionary Software Development

Björn Senft

Dissertation submitted in partial fulfillment of the requirements for the degree of *Doktor der Naturwissenschaften (Dr. rer. nat.)*

*Supervisor* Prof. Dr. Gregor Engels

March, 2021

**Björn Senft**

*A Value-Centered Software Engineering Approach for Unique and Novel Software-Based Solutions*

Doctoral Dissertation, March, 2021

Supervisor: Prof. Dr. Gregor Engels

**Paderborn University**

*Research Group Databases and Information Systems*

Department of Computer Science

Faculty for Computer Science, Electrical Engineering and Mathematics

Zukunftsmeile 2

D-33102 Paderborn

# Abstract

The development of unique and novel software-based solutions poses the challenge that we are very bad oracles when it comes to the correct prediction of value. Various studies show that we are only successfull in predicting value in 10% to 33% of the cases. One reason for this is that all constraints and dependencies can only become apparent in actual use. Furthermore, the initial evaluation of solutions is complicated by the fact that for a more profound evaluation of an innovation, a tangible prototype is usually required from the respondents. One possible approach to solve this problem is *Design Thinking*. However how *Design Thinking* is integrated with software development is still open. This thesis addresses this problem by introducing an approach called Insight-centric Design and Development (ICeDD). ICeDD integrates *Design Thinking* as a mixture of *Front-End Technique* and *Integrated Development Philosophy*. This is due to the fact that ICeDD recognizes that the best learning results can only be achieved by experimenting in parallel with prototypes that can already be used as good as possible under real conditions (i.e. software prototypes) and on the other hand that software development is generally expensive and takes significantly more time than e.g. simple paper prototypes. Hence, ICeDD guides through the different stages needed for using with software development from finding adequate design challenges over doing *Design Thinking* with non-software prototypes to create cheap and fast learning outcomes. Furthermore, ICeDD guides in preparing the outcomes of this stage to be used in software development and finally shows how to design the software development so that the 4Ps (people, process, product, and project) fit to the needs of developing and experimenting with several software alternatives at once to extent *Design Thinking* with more than one alternative also into software development. Furthermore, the feasibility of ICeDD is proven with the help of a case study.

# Zusammenfassung

Die Entwicklung einzigartiger und neuartiger software-basierter Lösungen birgt die Herausforderung, dass wir sehr schlechte Orakel sind was die korrekte Vorhersage von Wert betrifft. Verschiedene Studien belegen, dass wir nur in 10% bis 33% der Fälle erfolgreich den Wert vorhersagen. Dies liegt zum einen daran, dass alle Einschränkungen und Abhängigkeiten erst im tatsächlichen Gebrauch zum Vorschein treten können. Zudem wird die initiale Bewertung der Lösungen erschwert durch die Tatsache, dass für eine tiefgreifendere Bewertung einer Innovation in der Regel ein anfassbarer Prototyp von den Befragten benötigt wird. Ein möglicher Ansatz zur Lösung dieses Problem ist *Design Thinking*. Allerdings ist noch offen wie *Design Thinking* am Besten in die Software Entwicklung eingebettet werden sollte. Diese Arbeit addressiert dieses Problem durch die Einführung des Ansatzes Insight-centric Design and Development (ICeDD). ICeDD integriert *Design Thinking* als eine Mischung aus Front-End-Technik und Integrierte Entwicklungsphilosophie. Dies ist darauf begründet, dass ICeDD anerkennt, dass die besten Lernergebnisse nur durch paralleles Experimentieren mit Prototypen erzielt werden können, die bereits unter realen Bedingungen so gut wie möglich genutzt werden können (also Software-Prototypen) und zum anderen, dass Software-Entwicklung in der Regel teuer ist und deutlich mehr Zeit in Anspruch nimmt als z.B. einfache Papierprototypen. Daher führt ICeDD durch die verschiedenen Phasen, die für den Einsatz von *Design Thinking* in der Softwareentwicklung notwendig sind, von der Suche nach einer geeigneten Design Challenge über die Durchführung von *Design Thinking* mit Nicht-Software-Prototypen, um schnell und kostengünstig zu ersten Lernergebnissen zu kommen. Darüber hinaus führt ICeDD durch die Vorbereitung der Ergebnisse dieser Phase für die Nutzung in der Software-Entwicklung und zeigt schließlich, wie man die Software Entwicklung so gestaltet, dass die 4Ps (Menschen, Prozess, Produkt und Projekt) zu den Bedürfnissen der Entwicklung und dem gleichzeitigen Experimentieren mit mehreren Software-Alternativen auf einmal passen, damit *Design Thinking* mit mehr als einer Alternative auch in der Software-Entwicklung verwendet werden kann. Außerdem wird die Machbarkeit von ICeDD mit Hilfe einer Fallstudie belegt.

# Danksagung

Eine der wertvollsten Fähigkeiten der Menschheit ist es Wissen über Generationen hinweg weiterzugeben. Entsprechend fußen unsere Erfolge zu einem Großteil auf den Schultern derjenigen, die vor uns waren. Auch wenn eine Promotion eine eigenständige Arbeit ist, wäre sie ohne die Unterstützung anderer Menschen so nicht denkbar. Daher möchte ich mich an dieser Stelle bei den vielen Personen bedanken, die direkten oder indirekten Einfluss auf die Entstehung dieser Arbeit hatten.

Zuallererst möchte ich mich bei meinem Doktorvater Prof. Dr. Gregor Engels bedanken. Ohne seine Unterstützung und seine Offenheit gegenüber neuen Wegen im Rahmen interdisziplinärer Forschung hätte die Arbeit in dieser Form nicht entstehen können. Zudem wurde sie durch seine intensive wissenschaftliche Betreuung, Diskussionen und Denkanstöße maßgeblich geprägt. Durch seine gute Vernetzung mit der Industrie und den praxisnahen Projekten, die ich im SICP - Software Innovation Campus Paderborn durchführen durfte, hatte ich zudem die Möglichkeit meine Forschung mit den Erfordernissen der Industrie abzugleichen.

Ein besonderer Dank gebührt meiner Familie, insbesondere meinen Eltern Klaus und Hannelore sowie meinen beiden Onkel Gerhard und Wilfried Plass. Ohne ihre Ermunterung, Unterstützung und die Sicherheit, die sie mir über die Jahre gegeben haben, hätte ich diesen fachlichen Weg mit Studium und Promotion wahrscheinlich nie einschlagen können. Zudem wurde der Grundstein für mein Interesse an Informatik durch meinen Vater gelegt, der für meine Geschwister und mich sehr früh einen PC erworben hat, da er hierin für uns die besten Zukunftsmöglichkeiten gesehen hat.

Einen wesentlichen Einfluss hatten zudem die unzähligen Diskussionen mit meinen Kollegen aus den verschiedensten Disziplinen und der AG Engels, die zur Schärfung dieser Arbeit beigetragen haben und bei denen ich mich an dieser Stelle bedanken möchte. Hierbei sind zwei Personen besonders hervorzuheben. Zum einen mein Kollege Holger Fischer, der während meiner Promotion zu einem guten Freund geworden ist. Für die vielen Diskussionen besonders im Bereich der mensch-zentrierten Gestaltung, der Unterstützung und der gegenseitigen Motivation möchte ich ihm herzlich danken. Zum anderen ist Simon Oberthür an dieser Stelle hervorzuheben. Seine überaus konstruktive Diskussionskultur und Offenheit waren ein sehr wertvolles Element im Zusammenspiel mit dem sehr guten Auge meines Doktorvaters für Details und Herausforderungen, die mir bei dieser Arbeit sehr geholfen haben.

# Contents

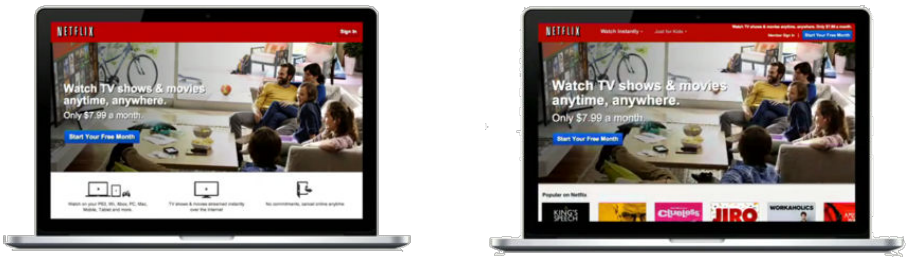
# Part I

Preliminaries & Foundations

# 1 Preliminaries

## 1.1 Introduction

Implementing unique and novel software-based solutions contains the great challenge that designers (e. g. UX or Requirements Engineer) can only build on existing knowledge to a limited extent. This limits the designer in accurately predicting what delivers value for the users. To illustrate how difficult it is to make accurate predictions, have a look at the following real–world example. The default experience for the *Netflix Frontpage* (see Fig. 1.1a) is a simple page with a *Sign In*–Button and a *Start Your Free Month*–Button offering three information: Their basic offering, the costs, and their promise to be able to use it anywhere.




**(a)** default experience **(b)** example prototype

**Fig. 1.1.:** Example for Qualitative Experiments regarding the Frontpage by Netflix [@BI16].

Our intuition tells us that we can convince more people to sign up and use the service if we give them more information about what shows and movies they can expect there. This is the way Netflix thought [@BI16] and a nice example of a unique and novel software-based solution on domain level in which we are especially interested in this thesis. Thus, they implemented a prototype where users could browse the library without logging in (see Figure 1.1b). They tested it against the default experience (see Figure 1.1a) and were surprised that the default experience still has a better conversion rate.

Their first guess was that their intuition is right, but their implementation was not good enough. They implemented four more prototypes that they tested against the default experience. The other four prototypes were inferior to it as well. However, Netflix learnt during their tests why their default experience is better than their supposed improvement (see [@BI16]).

Although it was only a moderate change, Netflix was unable to predict that it would deteriorate. This in turn coincides with the experience others have had with unique and novel ideas. Kohavi et al. [Koh+09] give an overview of the figures of some companies such as Microsoft, Netflix or Web Analytics. Between 66% and 90% of their implemented ideas fail to show value. This means that in most cases experts have suggested to implement a certain feature but had been wrong in estimating its value.

Current development approaches like *agile software development* (e.g. scrum) or *human–centered design* are only of limited help in this challenge if used on their own. As Norman & Verganti [NV14] argue, these approaches only fit incremental innovations as they optimize along a known solution path. But they do not try to understand the problem and find more suitable solution paths. Norman & Verganti use hill climbing to illustrate this (see Figure 1.2). For a set of design parameters we can achieve a certain product quality. *Agile software development* or *human–centered design* would start at a certain point (e. g. A) and manipulate the design parameter to reach a higher product quality (climb the hill). They would do it till they reach the local maxima (C in this case.). The challenge with this is, that they don't know that there is a another set of design parameters (B) with which they could achieve an even better product quality (D). Only if e. g. C is not satisfying, they would search for B to find a better set of design parameters. On the other hand, approaches like *Design Thinking* (see section 2.2 for more information) with its *diverging and converging thinking* for understanding underlying problems and suitable solutions (cf. Plattner et al. [PML10]) would start with several starting points and try to estimate the hill sizes by climbing them in parallel. Once a set of design parameters is found to be inferior, it would be dropped. This makes *Design Thinking* more suitable in finding a solution delivering a high product quality / value than *agile software development* or *human–centered design*. However, it is unclear how *Design Thinking* can be applied successfully in software development [LMW11].

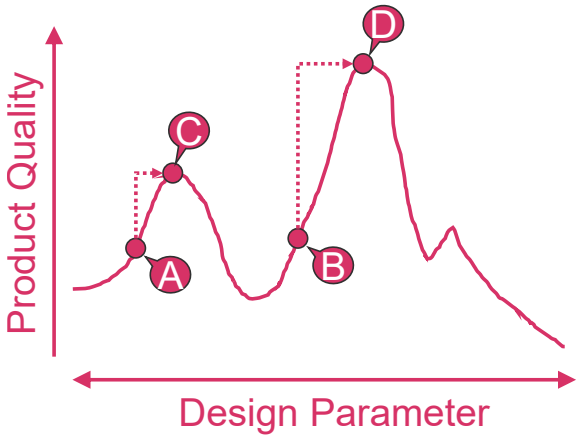



**Fig. 1.2.:** Hill Climbing representation with Design Parameters on the x-axis and Product Quality on the y-axis. Own representation based on [NV14]

Lindberg et al. [LMW11] see two ways in which *Design Thinking* can be applied to software development. On the one hand as *Front–End Technique* and on the other hand as *Integrated Development Philosophy*. In case of the *Front–End Technique*, *Design Thinking* would be placed as a phase prior to the development process. The output of the *Design Thinking* phase would be a single solution that would then be implemented as software. *Design Thinking* as *Integrated Development Philosophy* is implemented as a one–team approach. This means that all core members (e.g. software developer, designer, lead user) are involved throughout the development process.

As already mentioned, current approaches only fit incremental innovations. But in this thesis, the focus is on radical innovations. The challenge with radical innovations is that you can only define them as such ex post, because innovation implies that something is already adopted. Accordingly, we can only consider the first part of it, i.e. radicalism, for software development, as we cannot guarantee an adoption. This is one reason why there are approaches which make a difference between innovation and invention. One of such approaches is from Dahlin & Behrens [DB05] which we use to further break down radical innovation. Most important for us is their definition of technological radicalness which is actually quite similar to what we understand under radical innovation but mapped to technology. They state that a successful radical invention is defined by the following three criteria:

- Criterion 1. The invention must be novel: it needs to be dissimilar from prior inventions.
- Criterion 2. The invention must be unique: it needs to be dissimilar from current inventions.
- Criterion 3. The invention must be adopted: it needs to influence the content of future inventions.

Criterion 1 is a comparison with the past and Criterion 2 with the present. If both criterions apply, we have created a radical invention that potentially can become a successful radical invention. If all three criteria have been fulfilled an invention can be considered as a successful radical invention or in our words a radical innovation.

As can be seen in Criterion 3, the term successful radical invention implies that something must be adopted. But it can only be ex post determined whether something has been adopted and influenced the content of future inventions. This makes this criterion less applicable in a development approach as we can only integrate guesses on what terms the invention is adopted and therefore gets to be an innovation.

We will continue to use the definition of a 'successful radical invention' as a synonym for a radical innovation like Norman and Verganti [NV14] do. Since Criterion 3 does not allow us to guarantee that the development process will lead to radical innovation, we limit ourselves to the first two criteria for now. Hence, we are not talking about radical innovations or 'successful radical invention', but unique and novel software-based solutions.

If our software application is unique and novel, which is a prerequisite for radical innovations, we can neither rely on data from the past nor the present. Hence, constraints and interacting dependencies must be uncovered instead of analyzing or categorizing observations with existing knowledge. This in turn is only possible by probing or acting and observe the effects, which is what Netflix did as well in the *Sign–Up* example, described earlier. But how do we decide if in the current situation probing or acting and observing the effects is necessary or maybe analyzing is the best way to do it?

### 1.1.1 Decide on how to make sense of a situation

This is outlined by Kurtz and Snowden [KS03] in the *Cynefin* framework, which is developed to address "[...] the dynamics of situations, decisions, perspectives, conflicts, and changes in order to come to a consensus for decision–making under uncertainty". Instead of following a one size fits all approach [@LS16], it's advocating different decision–making approaches according to the context / domain. Very briefly, in *Cynefin* we have clockwise (see Figure 1.3) the domains *Chaotic*, *Complex*, *Complicated*, and *Obvious*. In the *Chaotic* domain, we know the least about the context and its constraints including interacting dependencies. The more we get to the *Obvious* domain, the more we know about the constraints and the better we can predict future states. Therefore, the closer we get to the *Obvious* domain, the better we can plan. The less we know, the more we must try out with different alternatives, to uncover and understand dependencies and constraints.

Unique and novel software-based solutions are related to the unordered domains *Chaotic* and *Complex* that are the domains of *novel* and *emergent* practices, whereas according to Kurtz and Snowden [KS03], incremental innovations are mainly located between the boundary of *Obvious* and *Complicated*. The high probability that unique and novel software-based solutions fall into these two domains also means that a mere selection of solutions is not enough, but that problem and solution understandings must be developed creatively.

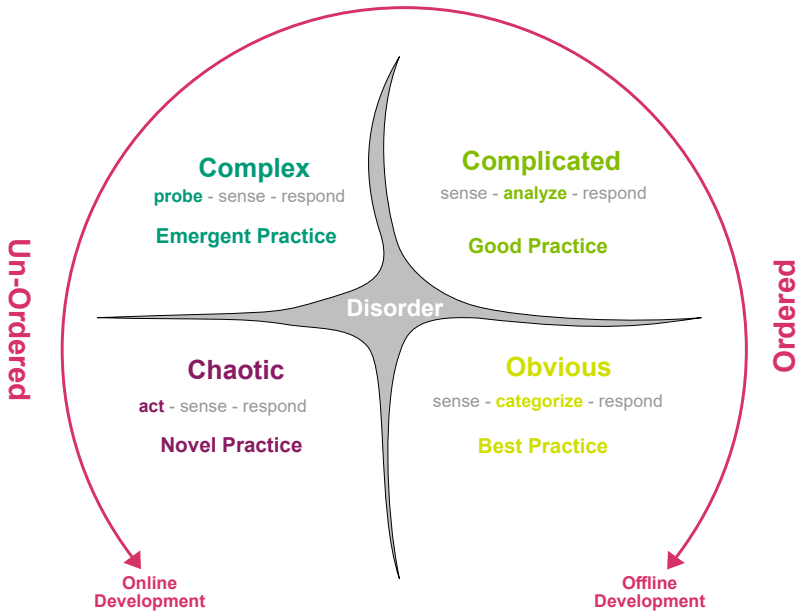



**Fig. 1.3.:** The Cynefin Framework and its five domains *Obvious, Complicated, Complex, Chaotic, and Disorder*. Own representation based on [KS03]

### 1.1.2 Preparing for Adoption

In addition to the first and second criteria (past and present perspective), the third criterion (future perspective) for radical innovations should also be considered in the approach. Even if an innovation cannot be predicted, it is important to know the attributes that influence the adoption of innovations (see section 2.1). According to Rogers [Rog10], innovation means that something is considered new by an individual or a group. It is irrelevant whether individuals or groups already exist who no longer regard it as new. Not even the time is important for this. It is only about the subjective perception of individuals or groups, whether something is regarded as innovation / new or not.

Innovations do not spread arbitrarily or abruptly but are subject to a certain lawfulness. The process that describes this is defined by Rogers as *Diffusion of Innovations* and consists of four main elements. An (1) innovation is communicated through certain (2) channels over (3) time between the members of a (4) social system. The first main element is particularly interesting, as it describes five attributes which have an influence on the product and the development process:

1. *Relative Advantage* is the degree to which an innovation is perceived as better than the idea to be replaced. At this point, it is not important whether the innovation provides objectively large advantages, but whether it is perceived as advantageous by the individual. The more advantageously an innovation is perceived, the faster it is accepted.

2. *Compatibility* is the degree to which an innovation is perceived to be compatible with the current value system, past experiences and needs of the adopters. If an innovation is incompatible, an adoption often requires a new value system, which is a relatively slow process. Therefore, compatible innovations are accepted faster than incompatible ones.

3. *Complexity* is the degree to which an innovation is perceived as difficult to understand and use. Innovations that are easier to understand spread more rapidly than innovations that require the adopter to learn new skills and knowledge.

4. *Trialability* is the degree to which an innovation is experimented with to some extent. An innovation that can be experimented with represents less uncertainty for the individual.

5. *Observability* is the degree to which the results of an innovation are visible to others. The easier it is for individuals to see the results of an innovation, the more likely they are to adopt them.

Regarding the development of unique and novel software-based solutions, this means that the more tangible they are, the better these innovations can be assessed. Depending on the compatibility, it could take a longer time till people accept and give positive feedback. The more incompatible it is, the longer it can be acceptable that people don't like it. Therefore, the goal must be to build tangible prototypes or software-based solutions as quickly as possible so that the relative value can be assessed at an early stage and with less bias.

### 1.1.3 Special Features of Software for Prototyping

As Boehm [Boe06a] points out in his summary of past software experiences, software development has always been in the continuum between *engineer software like you engineer hardware* and *software crafting*. The former means a process where everything is preplanned to ensure the quality before the first execution in the actual context of use. The second corresponds to a process of experimenting and working with rapid prototypes even in the actual context of use.

Software development methods oriented from the 1970's to the early 2000's more on the first part of the continuum. There were many reasons for this, such as contract design or infrastructure costs (e.g. testing, operation, distribution). The result, however, was an environment that is not beneficial for prototyping and experimenting and with that also not for acting and probing which we need for unique and novel software-based solutions. Why isn't such an environment beneficial for prototyping and experimenting?

Software created in these approaches usually are written in one technology and made to run on a system with shared libraries and fixed hardware. This leads to side effects if for example several versions of a shared library are required, or multiple applications require the same resource. Because multiple applications share a not isolated operating environment, changes can result in an unstable system. Therefore, changes are seldom made on such a system. In addition, a manual distribution, as is usual with such systems, leads to a higher effort and higher risk (cf. Knight Capital's bankruptcy due to incorrect deployment [@Sev14]).

Depending on the complexity of the already implemented code, it could be that a switch in technology becomes too expensive, because everything must be transferred at once. In addition, polyglotism is usually not possible in relation to technologies. This leads to the fact that despite more suitable concepts in other technologies, the concepts and constraints of the initially selected technology must be preferred.

Using sequential, phase–oriented software development, software is usually implemented with a point–based engineering approach (cf. Denning et al. [DGH08]) that results in a large overhead compared to set–based concurrent engineering as soon as changes must be communicated (see Ward et al. [War+95]). This overhead resulting from changes can make it seem unfeasible to integrate insights from experiments. In combination with figures about the relative cost of changing software (cf. Stecklein et al. [Ste+04]), it has also manifested the image that it is only possible to implement one solution at a time.

In summary, this inhibits prototyping and experiments as following:

- Risk to change a running system is high
- High effort to host several alternatives at the same time
- Integrating findings is associated with a high level of effort
- Technology decisions from the past limit the ability to make decisions in the future

Fortunately, this has changed since the advent of agile software development in the early 2000s. Agile software development put the testing of smaller executable software artifacts by customers in the foreground. Also, technologies and approaches like Cloud Computing [Arm+10], Containerization [Pah15], DevOps [SC17], and Microservices or Evolutionary IT Systems [DGH08] have a positive impact on the risk to change a system, the effort to host, and the effect of technology decisions from the past. Hence, several advancements in the recent past enable us to work more with prototyping and experimenting like it is the case in *software crafting* and reduce the need to do *engineer software like you engineer hardware* as Boehm states.

## 1.2 Research Question, Objectives, & Fitness Function

Developing unique and novel software-based solutions contains several challenges as already pointed out in the introduction. The starting point, which already results from the definition of unique and novel software-based solutions, is a very incomplete understanding. The best way to improve understanding in this situation is to act and probe (as Kurtz and Snowden [KS03] define it) in order to uncover constraints and interacting dependencies. But especially doing this with the medium software seems to be unfeasible. Nevertheless, it is essential to act and probe with software-based solutions in the actual context of use as can been seen in the Netflix Signup Example from the introduction. Would Netflix have used the alternative solution from the beginning and never have tested different alternatives, they would have been stuck with a bad design and could not explain to themselves why the front page works badly. This is underpinned by the results of Kohavi et al. [Koh+09] which can be summarized as we are very bad oracles regarding the real value of our proposed software. From these, we derive our **general research question**:

***How can unique and novel software-based solutions be developed that provide users with value in the actual context of use?***

### Objectives

In order to work on this general research question, a puristic computer science approach is not sufficient. The object of research is not limited to the internal structure of the software system, but also includes, through the software development process, the people who are involved in the development of the software, the context of use, and of course the process itself. This results in the necessity of an interdisciplinary approach, which is in line with the definition of software engineering by Boehm:

> *In this regard, I am adapting the [...] definition of* "engineering" *to define engineering as* "the application of science and mathematics by which the properties of software are made useful to people." *The phrase* "useful to people" *implies that the relevant sciences include the behavioral sciences, management sciences, and economics, as well as computer science.*
>
> **— Boehm [Boe06a]**

For this reason, we derive our first objective for working on the research question:

**Objective 1.** Identification of findings from other disciplines useful for the development of unique and novel software-based solutions.

As a second step, we can already set objectives for the design of a software development approach for unique and novel software-based solutions. Understanding and learning must be a keystone in such an approach, which results in several constraints.

First of all, the learning success is strongly limited by the use of only one possible solution. Due to the assumed uniqueness and novelty, we are in a situation where we cannot evaluate a product on the basis of an existing set of rules, but have to build that set ourselves. For that, dependencies and constraints have to be identified. In a not well understood real-world context this is only possible by manipulating design parameters of possible solutions to learn from the differences (in the sense of the Cynefin framework to act and probe). This is the only way to make statements that go beyond whether something is basically working or not.Therefore, our second objective is:

**Objective 2.** A possible approach for developing unique and novel software-based solutions must support multiple solutions simultaneously in order to learn from their difference about dependencies and constraints.

Furthermore, to achieve our goal of providing users with value in the actual context of use, it is not only important to compare alternatives with each other, but also how these are compared to each other. As Hoare [Hoa69] already stated in 1969: ”the most important property of a program is whether it accomplishes the intention of its user”. That the software actually accomplishes the intention can only be assured if used in the real world. Until then, everything about the software is just a thought experiment, a limited space of testing, where unknown or neglected dependencies can not come forward. Hence, it is essential for the learning outcome that alternative solutions can as well be tested in the real world (like in the Netflix Signup Example). To make this possible, among other things, a suitable distribution mechanism is required, which is why we set the following further objective:

**Objective 3.** Since the actual value can only be determined during operation, the structure and processes must support the simultaneous operation of several software solutions.

In a perfect world, the solution is limited only by the properties of the context of use and not by other factors such as technology. Of course, this is not always possible, but technological decisions should limit the solution space as little as possible in the long run. For unique and novel software-based solutions this is more important than ever. The basic assumption here is that the basis for fundamental decisions remains volatile. Therefore, in principle, it is

hardly possible to make technical decisions that will last over a longer period of time. If this is nevertheless done, this can mean an unjustified restriction of the solution space, resulting in our next objective:

**Objective 4.** As the understanding of the problem and solution space will change over time, technological decisions should be made in such a way that decisions in the future are as independent as possible of them.

Solutions can not only be limited by technological decisions but also by the development approach itself. Is the approach purely descriptive about a current state? Does it align new technological solutions to the current state? Or does it even improve the current state in conjunction with the new technological solutions? And if it improves the state, how much of the already existing state is kept and how much is created from scratch? For creating unique and novel software-based solutions that provide users with value in the actual context of use it is indispensable that novel tasks, structures and processes are introduced to the user (innovations on the domain level). From these we derive the following objective:

**Objective 5.** A corresponding development approach must be able to develop new ideas on how to shape the context of use.

The previous objectives referred to the possibility of using alternatives in software development. This is important to enable effective and efficient learning in the area of unique and novel software-based solutions. However, the learning process itself must also be geared to this situation. This results in our final objective:

**Objective 6.** Developing unique and novel software-based solutions implies that the development process itself is also a learning and understanding process. Therefore, a learning cycle that considers several alternatives must be defined in such a development.

## Fitness Function

To operationalise our objectives we are borrowing the concept of *Fitness Function* (FF) from evolutionary architectures (see Ford et al. [FPK17]). Ford et al. introduced this concept following the idea of FF in genetic algorithm. There it is used to define when an algorithm successfully evolved to have the desired properties. Evolutionary architectures change over time as well as genetic algorithms. They too try to find the best way to fulfill the desired properties by slowly approximating to the desired state. Therefore, it does not makes sense to define a simple true/false function, but we need a function that illustrates the progress.

With the objectives, a desired state for a development method for unique and novel software-based solutions has been defined. But still, it remains unclear how such a method is expressed in detail to make it work like desired. In fact, we face the same challenge in developing such a method, for which reason we want to develop such a method. Therefore, we can also assume that our understanding of problem and solution space regarding such methods will develop further over time. Which is why we also need to develop such methods in an evolutionary way and thus need something that helps us objectively evaluate how well we are approaching the desired state.

As a result, the challenges and objectives already mentioned have led us to define a FF guiding us in reaching our goal. The optimal state that we hereby define refers above all to the feasibility of such methods according to our previous knowledge. It can not be guaranteed that all required characteristics are already considered and also not that this leads to the best possible solution to develop unique and novel software-based solutions. This is the reason why this FF can be subject to change.

The FF itself consists of various characteristics for each of them a mathematical function is defined, which outputs a value between 0 and 5 on the basis of assessable conditions. We adjust the value range from 0 to 5 for each characteristic in order to make them comparable with each other. As mathematical functions we use only discrete functions and mainly two types of them. For the first type, the total is formed from the conditions that are fulfilled. This is used when conditions are equally justified and independent of each other. The second type is used for consecutive conditions. Here, a higher value is given depending on how many conditions are fulfilled. In total, we have defined the five characteristics *Alternatives*, *Operating Alternatives*, *Consequences of Technological Decisions*, *Focus on Novelty*, and *Learning Cycle*. These can be displayed as Radar Chart as shown in Figure 1.4.

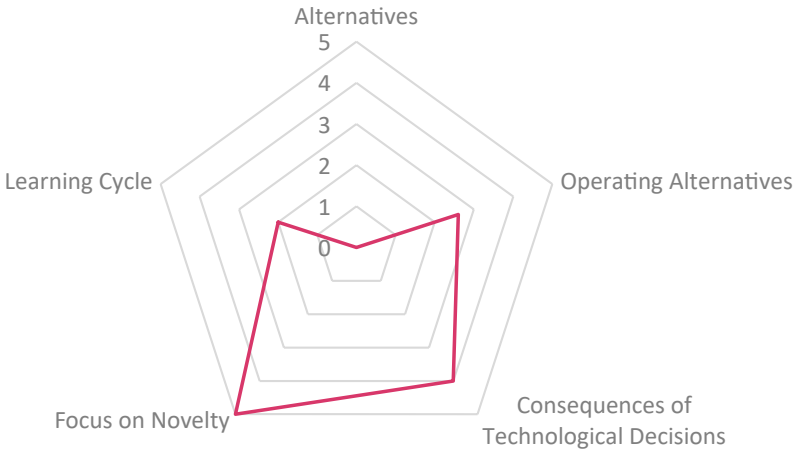



**Fig. 1.4.:** Example Radar Chart for our *Fitness Function* (FF).

### Alternatives

Our first characteristic is *Alternatives*, which we derived from Objective 2. We are using consecutive conditions for this characteristic. We refined the objective as such that we included the temporal aspect in addition to the requirement of having multiple solutions. In addition, we have introduced a distinction regarding the medium, because as mentioned before, software as medium in comparison to other media can be expensive and we need shorter learning cycles depending on the context.

| Value | Condition |
|---|---|
| 0 | Only one solution is supported at all times. |
| 1,25 | At least two solutions are supported, but not simultaneously. |
| 2,5 | At least two solutions are simultaneously supported, but created sequentially. |
| 3,75 | At least two solutions are simultaneously supported and created in parallel. |
| 5 | At least two solutions are simultaneously supported, created in parallel, and can be present as software as well as non-software. |

### Operating Alternatives

The second characteristic is *Operating Alternatives*, which is derived from Objective 3. For experiments in production it is important that a fallback mechanism is present to guarantee the smooth continuation of work in the event of an error. This also includes an automatic rollback of an update, as the Knight Capital case [@Sev14]) shows. Therefore, there is a need for automated deployment and configuration. This is as well needed to prepare and support numerous experiments with different alternatives. Which, in turn, require an assignment possibility to different user groups, which can be carried out online. Supporting these is a component based deployment as this enables us to timely rollback an update without taking down the whole system or waiting for a complete new instance of such system. As these conditions can help independently we are using the first type of mathematical function.

| Value | Condition |
|---|---|
| +1,25 | Online Fallback Mechanism. |
| +1,25 | Component based Deployment. |
| +1,25 | Automatic Deployment and Configuration. |
| +1,25 | Automatic User Specific Online Orchestration. |

### Consequences of Technological Decisions

*Consequences of Technological Decisions* is the third characteristic of our FF which is derived from Objective 4. The impact of decisions can be reduced by increasing the number of possible combinations and reducing dependencies. We can highly increase the number of possible combinations if we are able to use different programming languages (Polyglotism). Polyglotism is usually not possible in relation to technologies. This leads to the fact that despite more suitable concepts in other technologies, the concepts and constraints of the initially selected technology must be preferred. The use of System of Systems (cf. [DGH08; FPK17]) helps to enable polyglotism related to technologies nevertheless. Independence of UI and Model Layer allows a more rapid development of the UI as it can freely rearrange available data and do not have to refrain to constraints introduced to the model by other UI components. This situation can be improved even more, when parallel models can be used as they allow in the model layer as well to rearrange and introduce data with less interaction with other components.

| Value | Condition |
|---|---|
| +1 | Polyglotism |
| +1 | Independence of UI and Model Layer |
| +1 | Parallel Models |
| | System of Systems |
| +0,5 | Suite of Small Services |
| +0,5 | Bounded Contexts |
| +0,5 | Services run in its own Process |
| +0,5 | Independently Deployable Services |

### Focus on Novelty

Our fourth characteristic *Focus on Novelty* is derived from Objective 5. In [FS16], we introduced order picking as an example to illustrate the difference between the actual non-digital world, a digital copy, and unique and novel software-based solutions. As can been seen in Figure 1.5, the actual order picking process is running completely analog with a physical clipboard. A digital copy of that without introducing changes to the workflow and maybe just minor improvements is the use of a software on a tablet that mimics the physical clipboard including its workflow. Pick-by Vision with Augmented Reality is introducing a

complete new workflow to the picker as he is guided directly to the shelf via a funnel and doesn't have to search for it with a list. We use this distinction to determine the method in terms of its focus on the creation of unique and novel software-based solutions.

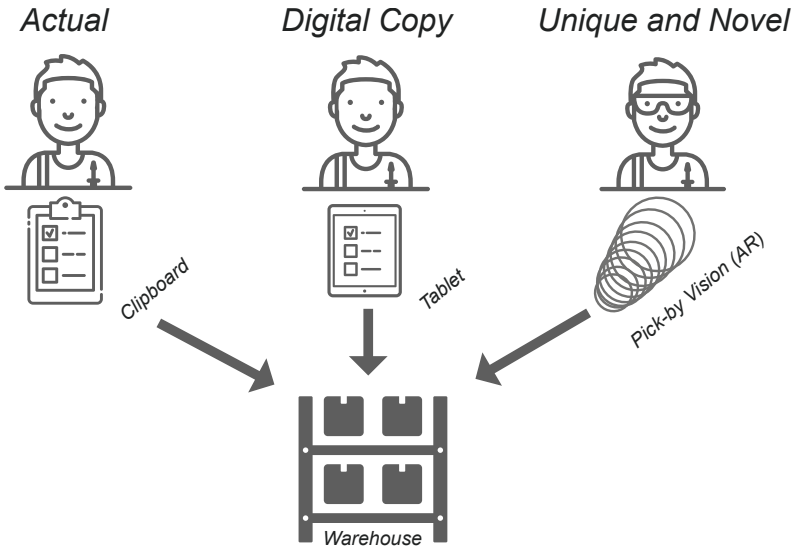



**Fig. 1.5.:** Example: Decision about the degree of novelty. Adapted from [FS16]

| Value | Condition |
|---|---|
| 0 | Method is only descriptive about current state. |
| $1,\overline{66}$ | Method describes current state and creates a digital copy. |
| $3,\overline{33}$ | Method describes current state, creates a digital copy and makes incremental improvements. |
| 5 | Method does not merely copy current state but creates novel solutions (this includes e.g. structure, tasks, or processes) |

### Learning Cycle

Derived from Objective 6 and Objective 2 is the last characteristic *Learning Cycle*. The main difference between the conditions is the explicitness of the learning cycle and the use of alternatives.

| Value | Condition |
|---|---|
| 0 | There is no learning cycle intended in the method. |
| 1,25 | A learning cycle is intended, but not explicitly defined. |

2,5 A learning cycle is intended, explicitly defined, but refers only to one solution at a time.

3,75 A learning cycle is intended, explicitly defined and refers to several alternatives at a time.

5 A learning cycle is intended, explicitly defined, refers to several alternatives at a time, and adapts it according to the context.

## 1.3 General Research Approach

> *Process/methodology PhDs can be very hard to make 'watertight' because they tend to become too voluminous and have too many inter-related aspects to be able to effectively justify. Although you may want to set the thesis within the context of a complete process, I would try to have some very clear aspects that you see as your 'contribution'. This will make it easier to defend.*
>
> **— Reviewer for the Doctoral Consortium of the British HCI 2016**

Developing and researching a software development approach for unique and novel software-based solutions is challenging for several reasons. People are involved, the software development domain itself is complex, and the target domain is not yet fully understood. Therefore, mathematical proof for this is only conditionally suitable. Working with controlled experiments is also difficult. The lack of knowledge leads to the fact that e.g. in the test design confounding variables are not properly considered or the operationalisation does not comply with what should actually be measured. In addition, controlled experiments (or A/B tests) with software development methods are very resource intensive as they have to be performed with several people over weeks or months in order to obtain reliable results. Is it even possible to do research in such a domain and if so how? How do we develop and research software development approaches for such a domain as unique and novel software-based solutions?

### 1.3.1 Research in General

To answer this question, we will start with Dodig-Crnkovic's discourse on scientific methods in computer science [Dod02]. She starts with a possible view on science which illustrates the plurality of different sciences. Logical reasoning is present in every kind of science and, together with Mathematics, represents the most abstract and exact sciences. They have the highest degree of certainty and are therefore usually the most desirable. Research subject are abstract objects like propositions or numbers which are mainly researched via deduction.

These two are indispensable for the natural sciences, while they are more important for physics than for chemistry or biology. As the natural sciences need them only as a tool they don't question the internal structure and therefore hide the deeper structure of Logic and Mathematics from the outside. While Logic and Mathematics are purely theoretical, natural sciences have as well empirical elements, meaning elements based on observation or experience. Reason behind it are the natural objects (e.g. physical bodies or living organisms) that are the research subject. Theoretical elements need to be aligned with how the natural objects behave. Therefore, the predominant method is not deduction but as Popper [Pop05] calls it the hypothetico-deductive method:

> *Hypothetico-deductive method, also called H-D method or H-D, procedure for the construction of a scientific theory that will account for results obtained through direct observation and experimentation and that will, through inference, predict further effects that can then be verified or disproved by empirical evidence derived from other experiments.*
>
> **— ENCYCLOPÆDIA BRITANNICA**[a]

[a] https://www.britannica.com/science/hypothetico-deductive-method

Natural objects have the advantage for researchers that they are subject to laws of nature that humans cannot enact or override at will. Social and cultural objects on the other hand may be subject to laws enacted by humans. These laws can therefore only occur temporarily or only for a certain subgroup. Therefore, an interpretive method is required for these objects, which interprets the data in context or makes sense of it. The theory and methodology for such interpretations is called hermeneutics.

Social sciences, who study humans as social beings (alone or in groups), rely therefore mostly on qualitative methods with the goal to understand and describe phenomenons. But they have as well quantitative aspects, primarily statistics. This is also the difference to humanities, which very rarely depend on any statistical method.

These interrelations between the different sciences have been illustrated by Dodig-Crnkovic in Figure 1.6. The inner regions are the prerequisites for the outer regions. Therefore, Logic is put in the most inner region as the core of all sciences. It is just one possible view for todays sciences, but shows quite clearly the specific areas of validity of the different sciences.

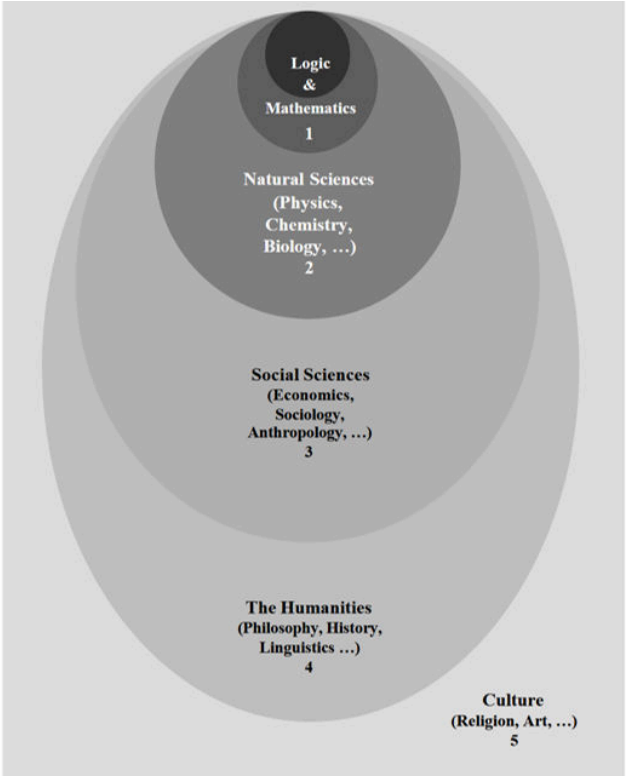



**Fig. 1.6.:** What is Science? One possible view from [Dod02]

Of course, there are not only hard-separated sciences, but more and more sciences that also look at cross-sectional/interdisciplinary topics and search their methods in very broad areas. Mechanical Engineering is an example for that. It is combining elements from physics, chemistry, economics, psychology, electrical engineering and computer science to advance with its research subject.

Although computer science is often seen with its roots in mathematical tradition especially related to algorithms and information structure, it has as well an empirical tradition (cf. Dodig-Crnkovic [Dod02]). The later one developed in Great Britain, whereas e. g. Germany and France had a stronger focus on the first one. Taking the software engineering definition from Boehm (cf. section 1.2 Objectives) or the statement of Hoare [Hoa69] (Developer of the Quicksort Algorithm) that 'the most important property of a program is whether it accomplishes the intention of its user' makes it clear that computer science is as well such an interdisciplinary science. The subjects of research in computer science lie in the validity

area of all sciences and therefore as computer scientists we need as well methods from the different sciences, especially empirical methods.

But as soon as we enter the domain of empiricism, we leave logic and mathematics with their fundamental truths and enter an area where it cannot be said with absolute certainty what is right or wrong. Nevertheless, empiricism has it's ways to assess it's research. The base for this are the three quality criterion objectivity, reliability, and validity which originate from quantitative research.

**Objectivity** is a general quality criterion of scientific investigations: Different researchers must arrive at the same results under the same (experimental) conditions (independent of the results from the experimental situation and the test managers) [HSE13; Ren+12].

This criterion is of particular interest for quantitative research. In qualitative research, however, this criterion is viewed critically, since data collection in qualitative research is geared to the social situation and therefore objectivity is hardly feasible or not considered desirable (cf. [HSE13]). Instead, the concept of inner comparability exists as an approximation for data collection, which is based on the assumption that objectivity cannot be achieved by the researchers behaving in the same way, but rather by their behaving context-specifically and thus creating the same inner situation. In case of data analysis the concept created to approximate objectivity in qualitative research is intersubjectivity, which expresses that a fact is equally recognizable and comprehensible for several viewers. Further criteria according to Mayring [May16] to increase objectivity in case of qualitative research are rule guidance and procedural documentation.

**Reliability** Reliability refers to the dependability and consistency of an investigation. An instrument is reliable if it delivers the same or similar results with a relatively constant behavior [HSE13; Ren+12].

For quantitative research, this usually means that studies can be repeated and that the results are the same or similar if the reliability is high (repetitive reliability). This concept is rather counterproductive for qualitative research, since it emphasizes the uniqueness of each (research) situation and partly assumes that, for example, the participants change in the course of the research process. Nevertheless, in qualitative research, interrater reliability, consensus among the researchers and procedural documentation lead not only to increased independence from the researcher, but also to a reduction in the susceptibility to errors. The very quality targeted by reliability.

**Validity** concerns the extent to which a study measures what it intended to measure [Ren+12; HSE13].

This quality criterion is important for quantitative as well as qualitative research. In general it can be broken down into internal and external validity. Internal validity concerns the degree a study was successful in eliminating potential confounding variables. This is something closely related to explanatory (respectively hypothesis testing) studies and therefore usually not of interest for qualitative studies.

External validity or Generalizability "refers to the extent to which a study's results apply to a wider range of people and settings than those actually studied. As a result, qualitative researchers have often employed the concept of 'transferability'; this concept does not imply that results can uncritically be generalized, but that they may apply more broadly, depending on differences in the nature and context of the situation to which they are transferred" [MR15]. As can been seen in this definition of external validity it can be distinguished between generalization of the results to the basic population and to other situations. Although a statistical generalization regarding the population is intended in quantitative research, qualitative research usually does not have this as goal and uses such a small sample that it is not possible at all. The generalization of the results to other situations on the other hand is firmly coded in the basic principles of qualitative research by researching the research subject in its natural habitat and not changing it actively. This is in contrast to quantitative research, which often can not rely on the natural setting or has to manipulate the research subject in order to eliminate potential confounding variables.

As already mentioned in the descriptions of these three quality criteria, it is usually not possible to have all criteria in their highest quality. Instead, it has to be decided which criteria should be prioritized higher and what is an acceptable quality according to the research situation. Helfferich [Hel11] provides an initial guidance of whether quantitative research methods or qualitative research methods should be used:

"Qualitative research methods justify their approach in contrast to quantitative methods with the special character of their subject matter: Qualitative research reconstructs meaning or subjective views [...]. Their research mission is understanding, working with linguistic utterances as 'symbolically pre-structured objects' or with written texts as their 'coagulated forms'. The object cannot be grasped through measurement, i.e. through the methodological approach of standardized research."

An example from our side for differentiating the need between quantitative and qualitative research is the conducting of a survey on how many companies perform agile software development. One could ask the companies whether they work with agile software development and offer only the answer yes/no. This can be used to make a quantified statement about

how many companies claim to work with agile software development. The problem here is that we do not know what the individual companies understand by this. So it can be that a company works very strongly with agile software development, but still answered with no. The reason for this can lie in a very detailed understanding and not reaching one's own standards. On the other hand, a company may have answered yes because it started programming randomly in a small project without any requirements. Understanding and elaborating these individual perspectives is the purpose of qualitative research.

### 1.3.2 Our Research Approach

With this knowledge about the basic features of scientific research methods, we come back to the question how we want to work on our research question "How can unique and novel software-based solutions be developed that provide users with value in the actual context of use?". First of all, if we examine the question, it is an explorative and not an explanatory or descriptive research question. We do not hypothesize that something is different from something else, want to give evidence that something works in a certain way or just trying to describe a state. We ask how something could work, we are trying to reconstruct meaning. This reconstructing meaning is exactly the area for which qualitative research is used.

Furthermore, we have already identified initial indicators that suggest that a solution to this question would have to produce alternatives and include them in an evaluation step. To provide evidence that this is indeed the case, we would need a functioning development approach for this and compare it with a traditional approach (e.g. Scrum). In our introduction and objectives we have already suggested that the probably best solution for this is to experiment with different alternatives to uncover dependencies and constraints (see section 1.1.1). This is based on the insights from the Cynefin Framework as well as the research methods of other disciplines to make sense of not well understood contexts. As already mentioned, however, such an approach did not seem possible so far (cf. section 1.1.3) and current approaches primarily aim at incremental innovation rather than reconstructing meaning using software (cf. [NV14]). Therefore, the first step must be to work out what is necessary in software development to be able to work like this. For this we assume, without proving it, that the findings from the Cynefin Framework and the other disciplines for reconstructing meaning can be transferred to software development (our first axiom). Once such an approach has been developed, work can be done to substantiate or disprove the transferability.

In the spirit of the British HCII reviewer to prevent the thesis from becoming too voluminous and having too many inter-related aspects, we focus in this work on identifying, understanding and describing phenomenons related to the development of unique and novel software-based solutions as well as providing evidence for the general feasibility of such an approach. We

are not interested in conclusively proving that an accordant approach is better suited than already existing modern software development methods, but rather that the feasibility is given and reasonable.

Since our focus is on developing such an approach, we assume on the basis of our first axiom that we still have to uncover dependencies and limitations. On the one hand, this means that quantitative studies cannot be carried out at an acceptable level regarding reliability and validity. On the other hand, purely theoretical considerations make little sense as they cannot yet be aligned with the research subject. Instead, the emphasis is on understanding and elaborating this context and therefore having a high external validity. This requires little to no obstruction of the research subject. One way to choose an appropriate research method for this is the *ABC framework* by Stol and Fitzgerald [SF18]. They use the variables *obtrusive research* and *universal contexts and systems* to divide research methods into the four quadrants *natural settings*, *contrived settings*, *neutral settings* and *non-empirical settings*. According to this classification, best suited for our requirements are methods from the quadrant of natural settings like field studies and field experiments. One such method that falls into this quadrant is *Action Research* [Lew46].

What is the difference of *Action Research* to other methods that makes it more suitable? To this end, we must first distinguish between two basic research directions. Stol and Fitzgerald [SF18] distinguish between *Knowledge-Seeking Research* and *Solution-Seeking Research*. The main difference is that the former attempts to describe a status quo and therefore has its focus on phenomenons and characteristics, while the latter tries to develop or improve solutions that can help to overcome challenges e. g. in the development of software systems and supporting processes. *Action Research* has manifested as critique to most empirical methods who only attempt to observe the world as it currently exists without trying to solve real-world problems. Instead, *Action Research* tries to ”solve a real-world problem while studying simultaneously the experience of solving this problem” [Eas+08]. It can therefore be classified as *Solution-Seeking Research*, the kind of research implied by our research question.

*Action Research* facilitates an approach that aims to intervene in the studied situations for the explicit purpose of improving the situation. It is aligned to a learning cycle where you try to incrementally and iteratively improve your problem and solution understanding. But therefore, the results created with this method are closely linked to the studied situation. This means that the transferability to other situations is only given to a limited amount. On the other side, the external validity is quite high with this method. The latter one is more important for us than the transferability because of the way we use *Action Research* as a foundation for further literature research trying to explain phenomena with existing scientific knowledge.

We assume that most phenomena affecting the development of unique and novel software-based solutions have already been described or explored in more detail. Hence, our first priority is not to describe the current status quo but to become aware of these phenomena in order to identify them in the corresponding disciplines. We use *Action Research* in an iterative and incremental way to approach our solution by introducing subgoals and learning from trying to achieve these the underlying constraints and dependencies. These constraints and dependencies are further elaborated with a literature research to identify possible requirements. Therefore, for this use case, transferability of the results on the *Action Research* stage is not necessary as it is the input for a literature research identifying already well-documented evidences. In concrete terms, our main instantiation of *Action Research* is an inquiry-based learning approach with a student project group[1].

In the master's programme in computer science at the Paderborn Universtiy, a practical project for the students, in the following called project group, is scheduled. It has a volume of 20 ECTS (30 ECTS till September 2017) and is spread over two semesters. In accordance with the framework guidelines, teamwork and organisation will be tested in practice in a group of up to 16 students. The students will get to know an extensive development process in their own experience in a team and will be supported in their personality development. In terms of content, they are introduced to current research topics that typically originate from the organiser's field of interest. In our case, this is the software development process. However, it is difficult to explore this process without a surrounding framework, especially by students who do not yet have the experience knowledge of an expert [Ben84].

Our project group was therefore mainly embedded in the context of the *History in Paderborn App* (HiP-App), which is primarily about the development of an app that makes the history of Paderborn accessible in an appealing way. This helps the students to focus on a concrete goal and to gain experience in the development of software systems in such environments.

In addition, they should learn how to build a formative research process for the research of the software development process and the product (inspired by the ideas of *Action Research* [Lew46], *Grounded Theory* [Str+96], and Design Thinking [PML10]). This results in the project group considering the three research subjects product, software development process and research process.

Since the examination regulations do not provide for a continuous course, the project group is designed in such a way that a new one can be started each semester and two project groups can formally overlap. This gave us an average of 16 students per semester, divided into two teams. One team consists half of students who have already completed one semester and half of new students. Thus, the project group renews itself every semester and the students have the possibility to practice the knowledge transfer, but also to critically question

[1] This has been published and discussed on the conference *Forschendes Lernen - The wider view, 2017* [SOF18]

their own procedures. To support inquiry-based learning by the students, we primarily use retrospective meetings, student discussions, task forces, steering meetings and design thinking workshops.

The retrospective meetings or sprint reviews are a method from Scrum (Schwaber & Beedle 2002). In Scrum, as in our case, we have subdivided the development activity into fixed time blocks, so-called sprints. After each sprint (usually one week) the teams analyse the positive and negative measures and discuss their improvement. The results are recorded in meeting notes in a wiki system and thus serve as a formative evaluation of the software development process. In addition, the current challenges are regularly discussed with the organizers in a steering meeting across teams.

Furthermore, individual discussions take place each semester between the students and the organizers. These are based on appraisal interviews [WH10]. In the first discussion at the end of the semester, goals for the next semester are set together with the student, the familiarization in the project group is discussed and impulses for own reflection and further development are set. This is deliberately done at the end of the first semester so that the students have a more realistic picture of what can be achieved in the project group. In the second discussion, the students reflect on what they have learned and the changes they have experienced since the beginning of the project group.

With the help of the task force, impulses are given for professional discussion in various areas such as requirements analysis, architecture or quality assurance. This takes place in close cooperation between the students and the organizers and serves to ensure that, in addition to product development, subject-specific research is not lost sight of.

In conjunction with students of cultural studies, we also conducted design thinking workshops. In this way, students learn to work together on an interdisciplinary basis and to question their own views and approaches. In this way they also learn a research process that iteratively develops, evaluates and improves products and ideas.

With this setting of a project group doing inquiry-based learning, we iteratively approached our goal of a software development approach for unique and novel software-based solutions term by term. We started the project group in December 2014 and finished the last round in September 2019. During this period we had a total of 71 students who participated in the project group. But more importantly, we tried out each term new aspects and learned from them (for a full list see Table 8.1 in section 8.1).

In supplement to the group work, a seminar was part of the project group. At the start of the project group this seminar was scheduled right at the beginning but changed in the following semesters. Then, each student that freshly joined the project group had around half a semester to get accustomed to the project group. Afterwards the student had to

work out a seminar topic in coordination with the supervisors and the other team members that should reflect and overcome current challenges in the project group (for a full list see Table 8.2 in section 8.1).

These illustrates quite good the broad topics in the project group that arise because of the setting. It was not just the pure focus on developing a research tool, but developing a product and learn from that about product development itself but also how to research a software development process via *Action Research*. This undermines as well that this setting has a high external validity as problems from the pure product development arose as well.

Besides our *Action Research* cycle based on insights we gained from the project group and then further elaborating on them with a literature research, we had two additional projects we used to gain insights and as a case study regarding the feasibility of chosen aspects. These projects are a *Firefighter Training System* [SFS14] and the centre for music edition and media (*ZenMEM*) [Mei+16b]. Furthermore we switched the project group from the HiP-App to the OWL.Culture-Portal in October 2018 as part of the reboot. For further information about how we did the case studies and our gained insights from the project group have a look at *chapter 8*.

In response to our initial question as to whether it is at all possible to conduct research in this domain, we can clearly answer yes. But as our discourse on scientific methods has shown, most of the methods are not applicable to our situation. We have not sufficient knowledge yet, so it is not guaranteed that theoretical considerations are already aligned with the research subject. Quantitative methods have the problem that for a high reliability and validity, they need sufficient knowledge as well. Therefore, qualitative methods are particularly suitable as we try to understand the context. For the reason of understanding, our main quality criterion is external validity. Which is why we decided for an *Action Research* inspired research approach with a inquiry-based learning project group. This ensures high external validity, allowing dependencies and limitations to manifest themselves and enabling us to become aware of such phenomena. But as we assume that most phenomena have already been well-documented in other contexts, we don't need a strong transferability as we are using the inputs from the project group for a literature research to reveal well-documented evidences for the phenomena. Furthermore, by using this research approach iteratively, we can incrementally develop and evaluate our approach in a case style manner in respect to feasibility. For this we also use other projects.

## 1.4 Overview of Publications

As already described in the research approach, this work was not done in a waterfall-like research style, but in an iterative style based on action research. The seed for this research was work [Klo+11; Böc+11; Mei+16b; FS16; FSS17] in the area of innovative software systems which lead to the questioning if current approaches are actually sufficient for such systems. In [SFS14] we explored firefighter training as a mostly analogous area for possible digitalization potentials which lead to insights about the adoption of innovations and with that about the importance of tangible prototypes early in the requirements phase. Furthermore, this is the foundation for our application case study regarding the adoption of innovation theories in software development (presented in section 8.2) and one of our main assumptions for ICeDD that we also need to experiment with different software alternatives and not just non-software prototypes.

Based on these findings, we have developed initial approaches [SO16; Mei+16a; Mei+16b; Fis+18] to make the problem of tacit knowledge, which occurs frequently in unique and novel software-based solutions, manageable by focussing more on the full learning cycle instead of just creating software like it is the case in agile software development. In addition to these approaches, we have also experimented with *Design Thinking* [Gre+16] which lead in conjunction with the approaches to our solution concept [Sen+19] which we present in section 3.1.

As the initiation of *Design Thinking* and therefore our approach proved to be challenging, we have developed with on-site feature requests and feature requests from systematic analysis two approaches to support with this task. On-site feature requests [Sen+18] is basically an assistance system for users to enable them to use requirements engineering methods to create more informative feature requests which can later on be used by the value designer to create design challenges. Feature requests from systematic analysis [BMS20; Mei+16a] is based on grounded theory and has its goal in refining information based on an iterative process conducted by the value designer.

Furthermore, in [BBS16], we have initially discussed the need for need for evolutionary software systems to be able to cope with change, especially in case of unique and novel software-based solutions that tend to change quite rapid in contrast to already well-established systems.

We presented our research approach based on action research in conjunction with a continuously running student project group in [SOF18] and put it up for discussion.

## 1.5 Overview of Thesis Structure

In Figure 1.7, we give an overview of the thesis structure. The thesis is separated in essentially three parts, namely *Preliminaries & Foundations*, *Solution*, and *Evaluation & Epilog*.

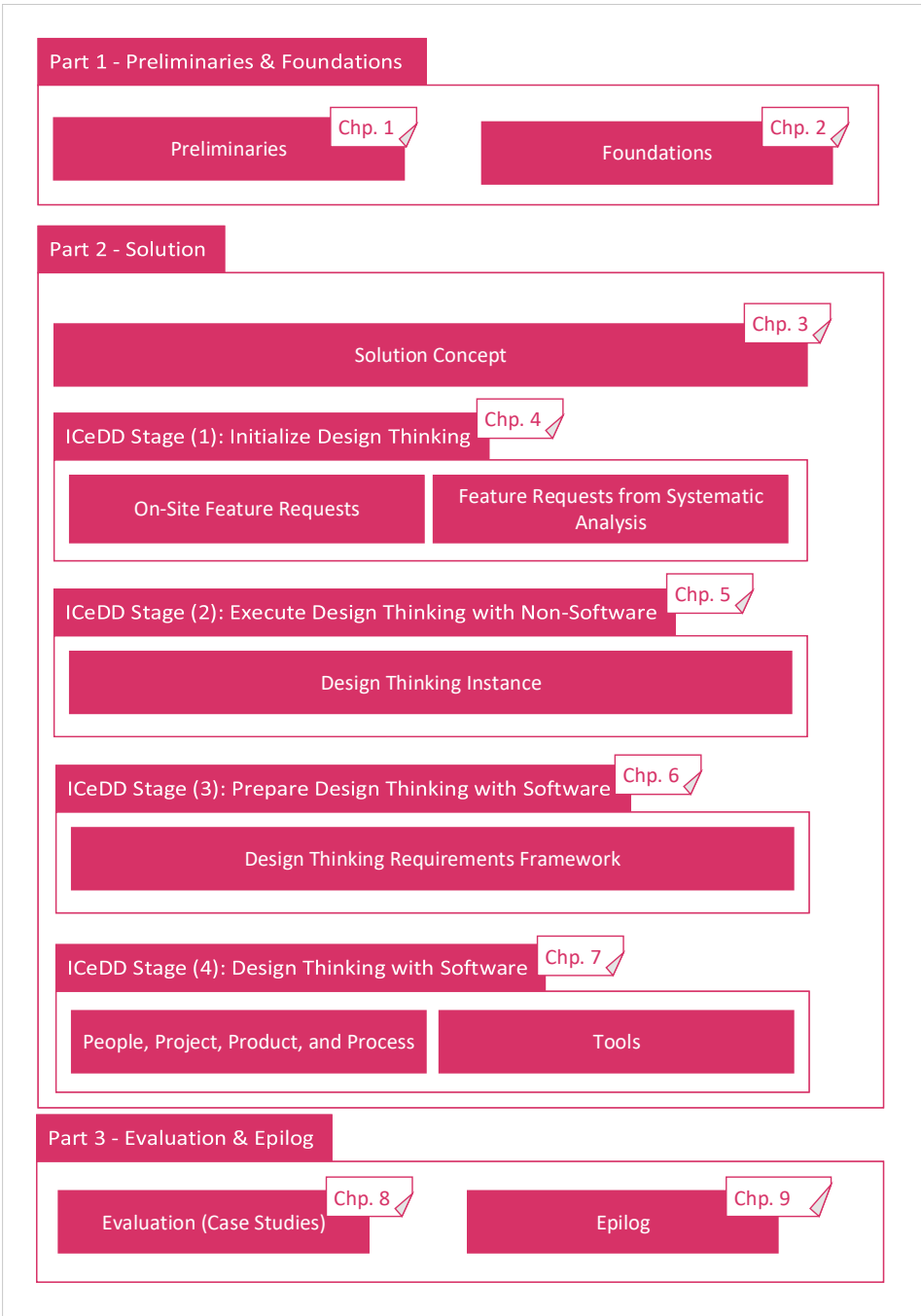



**Fig. 1.7.:** Overview of Thesis Structure

Chapter 1 contains the preliminaries with the introduction, the research question, research approach and publication overview.
In Chapter 2, we are presenting the foundations regarding our general solution concept which includes *Diffusions of Innovations* and *Design Thinking*. Further foundations that are not linked to the overall solution approach are presented if needed for the single stages in the *Solution* part.

The *Solution* part consists of the overall solution concept in Chapter 3, which is a brief description of our approach ICeDD, and a chapter for each stage of ICeDD except optimization as it is included only to highlight the transition to approaches optimized for incremental innovation.
Chapter 4, which is describing the initialization stage to find a good design challenge is further broken down into *On-Site Feature Requests* and *Feature Requests from Systematic Analysis*.
Chapter 5 explains how we used *Design Thinking* with non-software prototypes including how we adapted it to make it fit to software development in ICeDD. As we are producing mainly non-software artifacts like paper prototypes and not necessarily a complete requirements in this stage, we need a further refinement of the artifacts in this stage to be able to use them in software development.
This preparation stage is treated in Chapter 6.
The last chapter in this part is Chapter 7 which handles the integration of *Design Thinking* into the software development process. As it is an adaption of a software development process, we have used the 4Ps (people, process, product, and project) from the unified software development process to further break it down. Part of the 4Ps are also tools that are necessary to automate the process to make it feasible, which are presented for our approach in this chapter in the *Tools* section, which also concludes this chapter.

In the next part *Evaluation & Epilog* we begin with Chapter chapter 8 which includes the case study for our overall research approach in section 8.1, an application case study to evaluate if the assertions from innovations theories can be applied to software development in section 8.2, and finally our main case study in section 8.3 which give evidences that ICeDD is feasible.
In Chapter 9 we discuss the fulfillment of the fitness function regarding ICeDD and summarize our main contributions. Finally, we sketch future work by conducting further studies regarding efficiency and effectiveness of ICeDD, further refinement of the tools to make the approach more viable as well as the planned usage of ICeDD in a funded project phase of the *OWL.Culture-Platform*.

# 2 Foundations

In this chapter, we give an overview of foundations relevant to the understanding of the overall software development approach we have devised in this thesis. As already mentioned in the introduction, an understanding of what unique and novel software-based solutions mean for software development is essential. For this purpose, we present in section 2.1 the Diffusion of Innovations theory of Rogers [Rog10]. Roger's work, although already published in the 1960s, is one of the most cited works in this field and its basic assumptions are still used in today's textbooks on the subject, e.g. by Jürgen Hauschildt [Hau+16], one of the most renowned innovation researchers in Germany. Furthermore, we present in section 2.2 the basics of design thinking. As we have mentioned in the introduction, for the development of unique and novel software-based solutions it is essential to act and probe. Design thinking is a mean to do this systematically, but it is yet unknown how to apply it to software development.

## 2.1 Diffusion of Innovations

> *An innovation is an idea, practice, or object that is perceived as new by an individual or other unit of adoption. It matters little, so far as human behavior is concerned, whether or not an idea is "objectively" new as measured by the lapse of time since its first use or discovery. The perceived newness of the idea for the individual determines his or her reaction to it. If an idea seems new to the individual, it is an innovation.*
>
> **— Everett M. Rogers**
> [Rog10]

Innovation according to Rogers means that something is seen as novel by an individual or a group. It is irrelevant whether individuals or groups already exist who no longer regard it as novel. Not even the time is important for this. It is only about the subjective

perception of individuals or groups, whether something is regarded as innovation / novel or not. Innovations can be ideas, practices or objects.

The understanding of innovation is elementary for the correct method selection, documentation methods and interpretation of results in the context of the development of innovative systems. At the beginning of the development of innovative systems, such as the Internet, there may be no or other user needs and only with the development and diffusion do new needs and ideas emerge.

In addition, researchers can come across application partners or discussion/test persons who interpret innovations in completely different ways. This is shown by an example in the book *Diffusion of Innovations* by Rogers [Rog10], which at the same time is the basis for this section.

An example of this is a project of the Peruvian government to help the inhabitants of a small village to increase their health and life expectancy. An important element was to convince the inhabitants to boil their water before drinking it. The reason for this is due to the fact that the inhabitants use water sources that show signs of pollution and thus have an increased risk of waterborne diseases. Although this problem could be solved by a water treatment plant, it is too expensive for the village.

After a two-year water boiling campaign, during which, among other things, discussions with doctors clarified the situation, only eleven of the 200 families in the village were persuaded to boil their water before drinking. According to Rogers, three types can be identified in this example. First, the custom-oriented adopters who boil water because they are ill and because it is common in their culture to avoid extremely hot or cold medicine / goods as sick people. Water is considered extremely cold if it has not been boiled. If they were not ill, they would never drink boiled water, as they were taught from childhood that it was only for the sick.

The second group is made up of persuaded adopters, who feel that healthcare professionals are friendly, that the knowledge they bring with them is useful, and that they bring with them protection from uncertain dangers. As an example, Rogers cites a family who moved to the village from the plateau a generation ago. This gives the family an outsider status, and they don't have to worry about damaging their status.

The last group are the rejectors, which is the largest group in this example. They do not understand the theories behind bacteria and argue that something so small that it is impossible to see, cannot harm something as big as a human being. And there would be enough real dangers, like hunger or poverty, to worry about. [Rog10, pp. 1]

For the development of uniqe and novel software-based solutions, it can be assumed, among other things, that the diffusion process and the reliability of information sources must be taken into account. Therefore, these two aspects will be explained below.

## Diffusion

These novel ideas, practices or objects do not spread arbitrarily or abruptly, but are subject to a certain regularity. The process that describes this is defined by Rogers as diffusion and consists of four main elements. An (1) innovation is communicated through certain (2) channels over (3) time between the members of a (4) social system. According to Rogers, these four elements can be found in any study on diffusion research.

**1. Innovation** can be divided into 5 attributes that influence diffusion:

1. *Relative Advantage* is the degree to which an innovation is perceived as better than the idea to be replaced. At this point, it is not important whether the innovation provides objectively large advantages, but whether it is perceived as advantageous by the individual. The more advantageously an innovation is perceived, the faster it is accepted.

2. *Compatibility* is the degree to which an innovation is perceived to be compatible with the current value system, past experiences and adaptor needs. If an innovation is incompatible, an adaptation often requires a new value system, which to adapt is a relatively slow process. Therefore, compatible innovations are accepted faster than incompatible ones.

3. *Complexity* is the degree to which an innovation is perceived as difficult to understand and use. Innovations that are easier to understand spread faster than innovations that require the adopter to learn new skills and knowledge.

4. *Trialability* is the degree to which an innovation can be tried out. An innovation that can be tried out is less uncertain for the individual.

5. *Observability* is the degree to which the results of an innovation are visible to others. The easier it is for individuals to see the results of an innovation, the more likely they are to adapt them.

**2. Communication Channels** are the paths through the messages from one individual is to reach another. The path through the channels of the mass media is more effective when it comes to imparting knowledge about innovations, whereas the path through interpersonal communication is particularly suitable for shaping or changing attitudes towards innovations. Consequently, this influences the decision to accept or reject an innovation. For most individuals it is true that they evaluate innovations on the basis of close colleagues and not on the basis of scientific knowledge. Therefore, these colleagues serve as role models so to speak.

One challenge in communication is the fact that most communication takes place between people who share certain characteristics, since such situations lead to more effective communication. In diffusion processes, on the other hand, the situation of having two dialogue partners with different characteristics is more common and with it an effective communication gets more difficult as they first need to understand what the other partner is actually referring to.

**3. Time** is involved in the innovation decision-making process, innovation and adaptation rate. The innovation decision process can be divided into the parts knowledge, opinion, decision, implementation and confirmation. In the first part an individual learns about the existence of an innovation and makes first experiences with it. In the second part, the individual forms a benevolent or not benevolent opinion about the innovation before it comes to the activities that lead him to decide for or against an innovation. Once the individual has opted for the innovation, it begins to use it and possibly re-evaluate it at a later point in time. Innovation indicates how early an individual is in the adoption of an innovation relative to other individuals.

The rate of adaptation indicates the speed at which an innovation is accepted by members of a social system.

**4. Social System** is defined as coherent units that together pursue a common goal. A social system is characterized by a structure that is defined by the systematic agreements of the units of a system and gives it stability and regularity in relation to individual behavior in the system. An important aspect of the social structure are norms that represent established patterns of action for members of the system.

Three types can be distinguished with regard to opinion forming and innovation decisions. In the case of the *optional* decision whether an innovation should be accepted or rejected, it is made independently of decisions of other members of the system. With the second type, the *collective* decision, it is made on the basis of a consensus of the members of a system. The last type of decision is the *authority*. Here decisions are based on the judgment of relatively few individuals who possess power, status or technical expertise in the system.

The degree is expressed by opinion leadership, which describes the extent to which a single individual can informally influence other individuals.[1]

The complexity of the factors influencing diffusion that have been presented so far means that individuals in a social system do not all accept an innovation at the same time. Rather, individuals accept the innovation in a temporal sequence that allows for subdivision into different adopter categories based on the time at which they first used a novel idea [Rog10, p. 267].

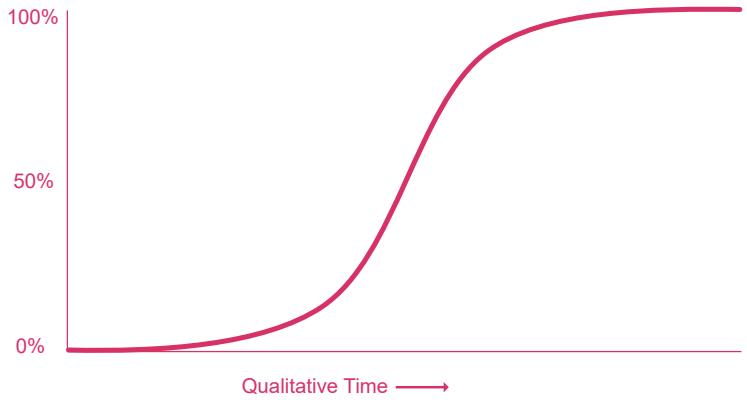



**Fig. 2.1.:** S-Curve illustrating the generalized adoption of an innovation.

A generalization for the distribution of adaptors that Rogers has set up, states that they tend to follow an s-curve, as shown in Figure 2.1, over time and approach a normal distribution [Rog10, p. 298]. How exactly this s-curve runs and really hits the 100% depends, however, on the innovation itself and external factors such as communication or the social system.

For validating possible unique and novel software-based solutions (like in the Netflix example in section 1.1), this s-curve means that for the interpretation of results and requirements an assessment must be made of how widespread an innovation is already in order to be able to assess the significance of the results.

If an innovation for a social system is relatively novel and not yet very widespread, then this may mean that, for example, the relative advantage or observability from the point of view of most individuals of the system under consideration is not yet given. In the case of surveys on the added value of innovation, most would answer rather negatively, because otherwise they would have accepted the innovation (if this were the only two reasons). From this it can be assumed that results must be critically questioned at the very beginning of the diffusion of an innovation in order, for example, not to reject an innovation that can provide added value on the basis of subjective perception.

[1] The statements in this part are based up to this point on [Rog10, pp. 1–37]

However, these information about the diffusion of innovations may not be sufficient for developing unique and novel software-based solutions. In order to save resources, for example, in the early phase only those members of the social system should be taken who are very innovative, since they are more open to the novel ideas and are later to function as role models / multipliers within the framework of a large field trial for the other individuals.

To this end, a categorisation must be introduced that classifies the members of the social system on the basis of their innovativeness. One method of doing this categorization took a dominant position in the early 1960s, namely categorization based on the s-curve [Rog10, p.272, p.297].

## Adopter/User Categorisation

In order to be able to carry out a categorization, it must be decided how many adopter categories are used, what proportion of the total social system is assigned to each category and according to which methodology this should be done [Rog10, p.279].

Ideally, the division into adopter categories should be complete, mutually exclusive and derived from a single classification criteria. Complete in this context means that all individuals of a considered social system can be assigned to a category. In addition, mutual exclusion prevents an individual from appearing in two different categories. [Rog10, p.280]

These requirements are met, among other things, by criteria that can be represented by continuous functions. In this case, for example, the output values of the function could be assigned to different categories according to a division of the value ranges. Innovativeness of an individual is a criterion that can be presented in this way.

This is due to the relativity of the comparison of one individual with the other individuals in the social system with respect to the time at which an innovation was accepted. As a result, it can be determined for each individual whether it is more or less innovative than others in the social system. The partitioning into discrete categories in such cases is only conceptual, similar to the classification of automobiles into small car class, lower middle class, middle class, upper middle class, upper class and luxury class. Such a classification simplifies the understanding and results in a loss of information. [Rog10, p. 280]

As a result of the approximation of the s-curve for the distribution of the adopters over time to a normal distribution, this can be represented as a density function of a normal distribution. The density function in this case would be innovativeness of an individual with the consequence that on the left side it is indicated how many individuals are particularly innovative and on the right side how many are not at all.

The normal distribution has some useful properties that help with classification. For example, the average $\overline{x}$ is always in the middle of the distribution and the width can be specified using the standard deviation $\sigma$. The latter is because in a normal distribution, the function is mirrored on the other side of the average. This also makes it possible to determine how many percent of the measured values are within a range defined by the standard deviation. The $\pm\sigma$ interval would contain about 68% of all measured values and the $\pm 2\sigma$ interval about 95% of all measured values.

Rogers used these two statistical properties, the mean value ($\overline{x}$) and the standard deviation ($\sigma$), to divide the normal distribution of the adaptors into five categories. This methodology is said to be the most widely used, if not the only one, in diffusion research today. It fulfils all three principles proposed by Rogers. The five categories are complete (with the exception of non-adopters), mutually exclusive, and derived on the basis of a single classification principle. [Rog10, pp. 280–282]

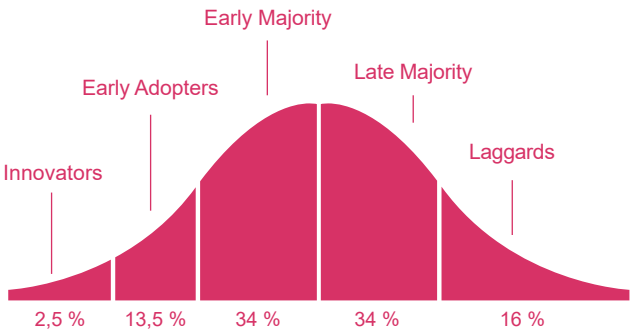



**Fig. 2.2.:** Density function of the normal distribution with drawing of the individual Adopter categories and their share in the total quantity.

Figure 2.2 shows the five categories *Innovators*, *Early Adopters*, *Early Majority*, *Late Majority* and *Laggards*. Terms like *Innovators* and *Early Adopters* are widely used and understood by the public [Rog10, p. 282].

*Innovators* represent 2.5% of the total amount and are in the interval $[-\infty, \overline{x} - 2\sigma]$. The *early adopters* with 13,5% is in the interval $[\overline{x} - 2\sigma, \overline{x} - \sigma]$. On the other hand, the *laggards* with 16% in the interval $[\overline{x} + \sigma, \infty]$ represent the asymmetric counterpart. In between, the largest groups are the *early majority* and *late majority* with 34% each in the interval $[\overline{x} - \sigma, \overline{x}]$ and $[\overline{x}, \overline{x} + \sigma]$.

That this categorization is not symmetrical in the sense that there are as many large categories to the left as to the right of the mean ($\overline{x}$) is due to the large differences within the groups *innovators* and *early adopters*, which are not found in the group of *laggards* [Rog10, p. 281].

In the following, the categories are presented as ideal types[2]. According to Rogers these are based on observations of reality and are designed to allow comparisons. Ideal types are not an average of all observations concerning an adopter category, but are based on abstractions of empirical investigations. Therefore, exceptions to the ideal types can be found. [Rog10, p.282]

**Innovators** have as their most prominent characteristic their *adventurousness*, which is almost an addiction and is based on their desire for the bold, daring and risky. In view of their interest in novel ideas they break out of the local circle of colleagues and have rather cosmopolitan social relations. Friendship with other innovators is common, regardless of the geographical distance between individuals. Innovators have to meet some preconditions. They should have sufficient resources to compensate for any losses caused by unprofitable innovations. They also need the ability to understand and apply complex technological knowledge. In addition, they must be able to deal with great uncertainties about innovation, because setbacks in innovation at an early stage are inevitable. Although innovators are not necessarily respected by other members of the social system, they play an important role in diffusion. By introducing innovations from outside the social system boundaries, it triggers novel ideas within the social system.

**Early Adopters** are better integrated into the social system than innovators, as they are less cosmopolitan, but more local. Early adopters have the highest degree of opinion leadership in the social system compared to the other groups and they know this, which is why they try to maintain a central position in the communication channels. They are consulted by other individuals when it comes to adopting innovations, as they are not too far from the average on this issue. They are *respected* by their colleagues and are regarded as the epitome of the successful and discreet use of novel ideas. Therefore, they serve as role models and help to reach critical mass. Early adopters reduce the uncertainties of others about a novel idea by adopting it and provide their colleagues with a subjective assessment of the adoption via interpersonal networks.

**Early Majority** adopts novel ideas before the average member of a social system and is one of the largest groups. Due to their central position in the distribution, they play an important role in the diffusion process. They provide the connections for the interpersonal networks of the social system. Despite their frequent interactions with colleagues, they seldom assume the position of opinion leadership in the social system. The duration of their innovation decisions is seen longer relative to the *innovators* and the *early adopters*; they

[2] Based on the descriptions of Rogers [Rog10, p.282ff]

*think* thoroughly. Their credo is ”Be neither the first to try something new, nor the last to set aside the old”.

**Late Majority** makes up exactly like the early majority 1/3 of the total population, with the individuals in this group accepting innovations only after the average member. They are sceptical about innovations and do not accept them until most others have done so. Their motivation to accept an innovation is mainly derived from two things. On the one hand the economic necessity and on the other hand the increasing pressure from their colleagues. Late majority individuals have relatively scarce resources. As a result, most uncertainties must be removed before they feel confident about accepting an innovation.

**Laggards** are the last in the social system to accept an innovation. They have almost no opinion leadership and represent the group with the strongest local orientation, sometimes even so far that they are almost isolated in the social networks of the social system. Individuals in this group often interact only with people who think similarly traditional as themselves. Their point of reference is the past and so their decisions are often based on what has been done before. Their innovation decision-making process takes a relatively long time and is lagging behind the known knowledge about innovation. They also tend to be suspicious of innovation. This is partly due to their precarious economic situation, which forces them to be extremely cautious when it comes to accepting innovations. Resistance to these can therefore be completely rational from the laggards point of view.

This classification into the five classes *Innovators*, *Early Adaptors*, *Early Majority*, *Late Majority*, and *Laggards* helps us to interpret results we got from validating unique and novel software-based solutions as well as to carefully select participants we do these validations with. To be able to choose in which category an individual possibly falls, Rogers has accumulated variables related to innovativeness after extensive literature research. He summarized these as generalizations under the three headings (1) Socioeconomic status, (2) Personal variables and (3) Communication behavior. For more details about the generalizations, you can look directly at [Rog10, p. 287 - 292].

In summary, the findings that can be drawn from diffusion research for the question of unique and novel software-based solutions are that users and discussion partners for innovations have to be carefully selected and the strategy with regard to method and evaluation adopted accordingly. This is essential for an acting and probing approach (as required for unique and novel software-based solutions, cf. section 1.1.1 and section 1.2) regarding domain related innovations as it requires regular validating of the proposed solutions with real users. In addition, requirements and statements can change during the diffusion process.

## 2.2 Design Thinking

Besides the innovation theory in general that we introduced in the previous section, understanding design thinking as a systematic way to act and probe as well as to create unique and novel solutions is essential for understanding the developed software development approach in this thesis. In the following, we will present the basics of design thinking as it is a well-researched process (see e. g. the 'Understanding Innovation' book series at SpringerLink[3]) for acting and probing that is also already in use in industry (e. g. at SAP[4] or IBM[5]).

Design Thinking describes first and foremost the way of thinking in the discipline of design. As Dorst [Dor11] points out, there is an "[...]incredibly diverse array of design practices[...]", with multiple perspectives and rich pictures which led to different modes of design thinking. Only by the demand of using this paradigm for dealing with problems in many professions, the urge rose to have a clear and definite knowledge about design thinking. However, this is problematic for the design research community, which is shy of oversimplifying its research object. Nevertheless, there are some key aspects that we will present here.

The core of design thinking "[...]is the need to create ideas and find solutions, which are as viable as possible for certain groups of users" (cf. [LMW11]). It is about finding innovations that work and for that it is crucial to align the different perspectives on a product. The very basic interdisciplinary view on this is the triad of desirability, feasibility, and viability (see Figure 2.3) like Weiss [Wei02] for example proposes. Desirability describes what motivates / delivers value for the users and therefore needs an understanding of how people interpret and interact with the things they encounter in the world. Feasibility is about the technology and how to use it, so that a meaningful product for the user is created. Viability means the alignment with the organization's strategic objectives and competitive positioning.

Lindberg, Meinel, and Wagner [LMW11] argue therefore that as design thinking tries "[...]to offer a very concrete solution to a complex problem that is socially highly ambiguous and hence neither easy nor certain to comprehend", its problems are close to wicked problems, blurred in character and not definitely definable. Dorst [Dor11] uses an analogy to the epistemological steps described by Peirce to explain what solving problems in design means. He puts up the equation "What (thing) + How (working principle) leads to Value (aspired)" to describe the human problem solving. If the "What" and "How" are known but not the "Value", we can predict the result or value, which is deduction. On the other hand if the "What" and "Value" are known and the "How" is missing, we can propose working principles,

[3] https://link.springer.com/bookseries/8802
[4] https://experience.sap.com/skillup/introduction-to-design-thinking/
[5] https://www.ibm.com/design/thinking/

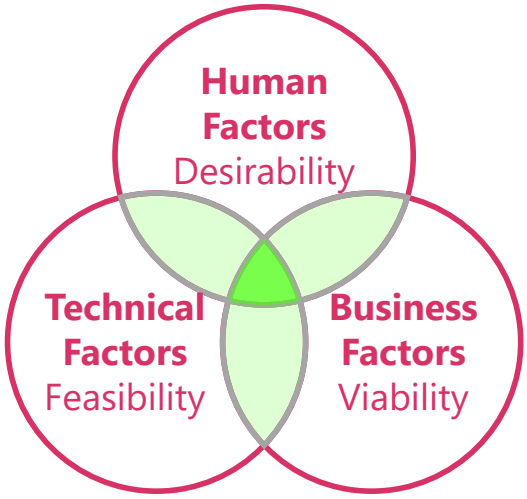



**Fig. 2.3.:** Innovation Triad based on Weiss [Wei02]

which is induction. This leads us to our problem solving for finding the "What", or in our terms the solution / product. If we know the "How" and the desired "Value", we can create a design that works within known working principles and for a scenario of known value creation. But design problems are more associated with a more open form of this, where neither the "What" nor the "How" are known.

In order to face this situation, in which neither the solution nor the working principles are known, there are two fundamental pairs of terms that are indispensible for *Design Thinking*: problem and solution space on the hand and diverging and converging thinking on the other. These two pairs are often illustrated as so-called *Design Thinking Double Diamond* (cf. [Nor13] or Figure 2.4 for a possible variant). The basic idea is that problem and solution space are iteratively explored and aligned with the help of diverging and converging thinking in each space.

According to Lindberg, Meinel, and Wagner [LMW11] this is a fundamental difference between *Design Thinking* and science. Science would primarily explore the solution while working with an initial problem, whereas *Design Thinking* sees both solution and problem as something to be explored. Furthermore, sciences analytical thinking takes a problem and divides it in smaller problems till a solution can be found, whereas *Design Thinking* creates new options before selecting the most suitable.

With the *Design Thinking Double Diamond* we have the phases of design thinking but not a process to run through it. Norman [Nor13] introduces for that the human-centered design process that consists of the four activities:

1. Observation
2. Idea generation (ideation)

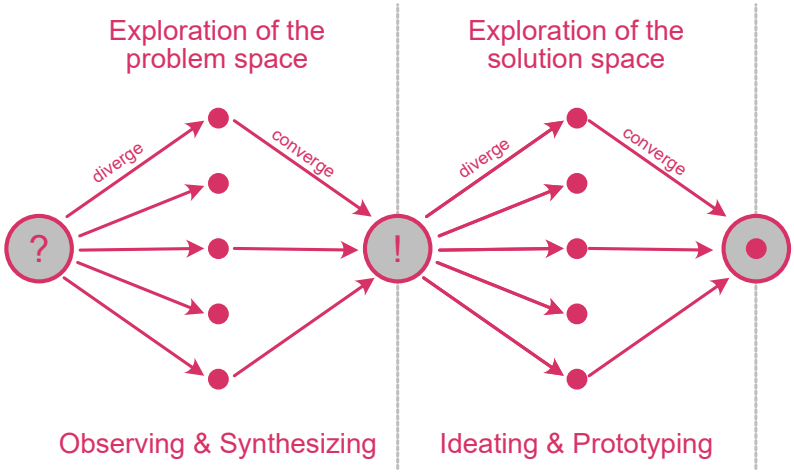



**Fig. 2.4.:** Problem and Solution Space in *Design Thinking* including diverging and converging Thinking. Own illustration based on Lindberg, Meinel, and Wagner [LMW11]

3. Prototyping
4. Testing

These four activities are iterated over and over in order to gain insights and getting closer to the desired solution. To emphasize that with each iteration progress is made, it is also called spiral model. This is the most basic form, which is often refined to five basic activities or stages visualized as the design thinking micro cycle (cf. Figure 2.5). The single stages as defined by [@DS19; @Doo+18] are *Empathize*, *Define*, *Ideate*, *Prototype*, and *Test*.

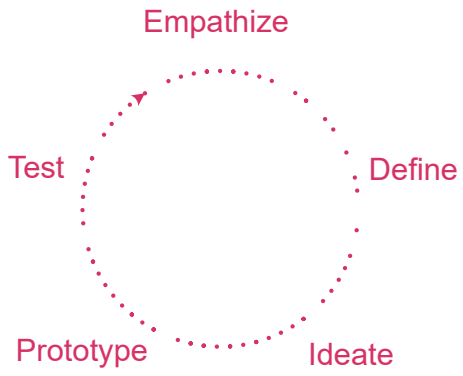



**Fig. 2.5.:** Design Thinking Micro Cycle

In the first stage *Empathize* the foundation of human-centered design will be laid. The goal is to understand the problems of the particular users as these are rarely the problems of the corresponding designers. Therefore, empathy is build by learning the users value through observation, engagement (e.g. contextual inquiry or interviews), and immersion (e.

g. empathy map). Only with this knowledge a solution can be developed that is focussed on the user needs.

The second stage is *Define* in which the findings of the *Empathize* stage are processed into actionable problem statements. These actionable problem statements can be for example *Point of View* (POV) (cf. Dam and Siang [@DS20]).

With the actionable problem statement, ideas to solve this problem can be generated. This is the third stage *Ideate*. The goal here is to explore a wide solution space to build a repository consisting of a large quantity and diversity of ideas.

This repository is the base to narrow down possible solutions to a few prototypes which are created in the fourth stage *Prototype*. The idea is to get ideas out of your head and into the real world so that people can experience and interact with them. Such kind of prototypes are the most successful as all assessed properties are physically present and people can experience them in the intended context. Just talking about ideas would incorporate the problem that the people need to interpolate missing information but not necessarily communicate the interpolation resulting in misunderstandings.

Experiencing and getting feedback based on the prototypes is the last stage *Test*. This stage is not about verifying that the prototype matches the specification but to validate if the intended problem can be solved with the prototype, to learn more about the user, to gather feedback, and to refine solutions.

Whereas Interaction-Design.org and the Stanford d.school [@DS19; @Doo+18] define these stages in this order, Plattner et al. as well as Uebernickel [PML10; Ueb+15] have switched the order. Their first step is *Define* followed by *Empathize*.

For the equation "What (thing) + How (working principle) leads to Value (aspired)", where "What" and "How" are unknown, we have shown how design thinking works to find them. What is missing, however, is the definition of "Value" to be created so that the equation can be resolved. Defining this, is the initial framing activity in design thinking, whereas framing stands "for the creation of a (novel) standpoint from which a problematic situation can be tackled" [Dor11]. The resulting standpoint is called design challenge. As you can imagine from this equation, the design challenge has a huge impact on the outcome of the design thinking as it frames the "What" and "How". Hence it "[...]should be approachable, understandable and actionable, and it should be clearly scoped—not too big or too small, not too vague or too simple" [IDE13].

# Part II

Solution

# 3 Solution Concept

In this chapter, we give an overview of our solution concept for developing unique and novel software-based solutions in section 3.1. The individual stages introduced in this concept are presented in the following chapters 4-7. An integral part of the solution concept is the assumption that the validation of a solution should be carried out using trialable and observable prototypes to gain a better understanding. This results from the assertions of the Diffusion of Innovations theory (see section 2.1) and Design Thinking (see section 2.2). Therefore, we conducted a case study to test the applicability of this assumption for software development prior to the development of this solution concept. We present this case study, which supports our assumption, in section 8.2.

## 3.1 Solution Concept: Insight-centric Design & Development (ICeDD)

From the previous section it becomes clear how important *Design Thinking* is for the development of unique and novel software-based solutions. Above all, diverging and converging thinking (cf. Transitions in Cynefin [KS03]) as well as working with prototypes (cf. Preparing for Adoption in section 2.1) are essential in order to find yet unknown interacting dependencies and constraints. However, once we enter the ordered domains (*Obvious* and *Complicated*) according to the Cynefin Framework, it is better to make decisions based on analysis or categorization than probing and acting (see section 1.1.1). Therefore, the solution should be limited to the transition from the unordered domains to the ordered domains in order to give priority to the established methods there.

Transition means a continuous improvement of the understanding, whereby at the beginning there is a very incomplete understanding. Therefore, less properties of the final product are needed at the beginning, but more probing with different cheap solutions is needed to efficiently improve our understanding. As a result, other media are more suitable than functionally integer software for use in the early phases. For example, paper prototypes can be produced much faster and cheaper if they are not to be all–inclusive or if interaction is less important. This is also pursued in set–based concurrent engineering in automotive engineering, where clay models instead of finished car bodies are the starting point [War+95].

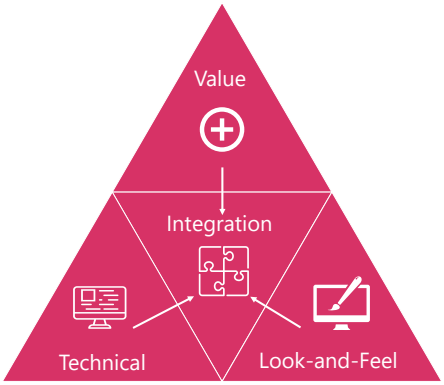



**Fig. 3.1.:** Prototype Levels. Own representation based on Houde and Hill [HH97]

Which general levels a prototype can have are described by Houde et al. [HH97] (cf. Fig. 3.1). The levels of prototypes they describe are *Value*, *Technical*, *Look & Feel*, and *Integration*. Houde et al. called *Value Role* in their paper, which translates to "what an artifact could do for a user". We find that a better description for this is *Value*, according to the idea of Value–Based Software Engineering [Boe06b]. In this context *Value* is not a financial term, but is meant as "relative worth, utility, or importance". The *Technical* or *Implementation* Level is for "answering technical questions about how a future artifact might actually be made to work". And the *Look & Feel* Level to "explore and demonstrate options for the concrete experience of an artifact". The last level is *Integration*, which can be an integration of properties of two or all three levels.

Unfortunately, Houde et al. [HH97] do not describe how to navigate through the levels. Therefore, we use the recommended user experience design process proposed by Mayhew [May12] (cf. Fig. 3.2). They suggest that the first thing to be determined is the utility as it is the prerequisite of a "great [...] user experience". This allows a goal–oriented development and minimizes the risks of changes on the functional or technical level (cf. Stecklein et al. [Ste+04] and point–based engineering [War+95] for the cost effects of changes on those levels.). It also coincides with the basic ideas of Value–Based Software Engineering, which sees value (e.g. utility) as guidance providing and a shortening of the consideration of value as the cause of most software project failures (cf. Boehm [Boe06b]). For this reason, our approach also starts at the value level, which includes usability and persuasiveness (cf. diffusion in section 2.1).

In contrast to the recommended user experience design process, we do not see the need that *Functional Integrity* must follow subsequently *Graphic Design*. Due to the greater adoption of the Model–View–Presenter (MVP) architecture pattern (cf. Potel [Pot96] and Fowler [@Fow06]) in software technologies (e.g. .NET or Angular) the presentation layer got more separated from the logic layer (e.g. in comparison to MVC). This allows a largely

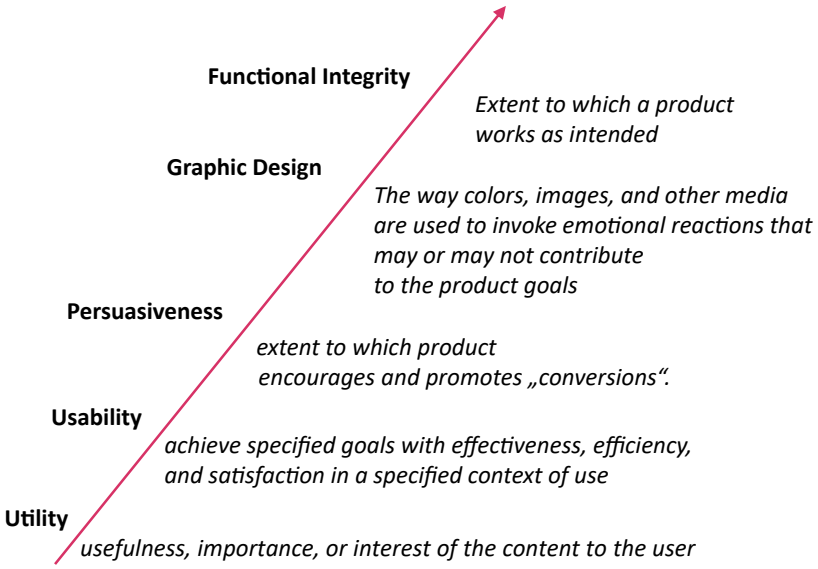



**Fig. 3.2.:** Recommended user experience design process. Own representation based on Mayhew [May12]

independent development of these levels (*Technical* and *Look & Feel* related to the prototype levels), which is why they can be developed in parallel.

As a result, for navigation through the prototype levels, *Value* must first be identified. In the next step, *Technical* and *Look & Feel* integrated with *Value* can be examined in parallel. Finally, integration is achieved across all three levels.

This leads us to our software development approach to handle the challenge of developing unique and novel software-based solutions which we call **Insight-centric Design and Development (ICeDD)** (cf. Figure 3.3).

*Insights* is intended to emphasize that this approach, like qualitative research, is mainly concerned with the reconstruction of meaning and the understanding of the problem and solution space. The focus is on creating unique and novel software-based solutions in terms of *Value* and not *Technical* or *Look & Feel*. Incremental improvements like in human–centered design (cf. Norman and Verganti [NV14]) and reliable measurement is outsourced to the (5) Optimization stage, which is the final stage. The other four stages in *ICeDD* are *(1) Initialize Design Thinking*, *(2) Execute Design Thinking with Non–Software*, *(3) Prepare Design Thinking with Software*, and *(4) Execute Design Thinking with Software*. We have divided *Design Thinking* into *(2) Execute Design Thinking with Non–Software* and *(4) Execute Design Thinking with Software* in order to take advantage of the non–software benefits, especially at the beginning when the problem and solution space is expanded by building many different solutions. In the following we will give a brief overview of these stages and further elaborate on them in the corresponding chapters 4-7.

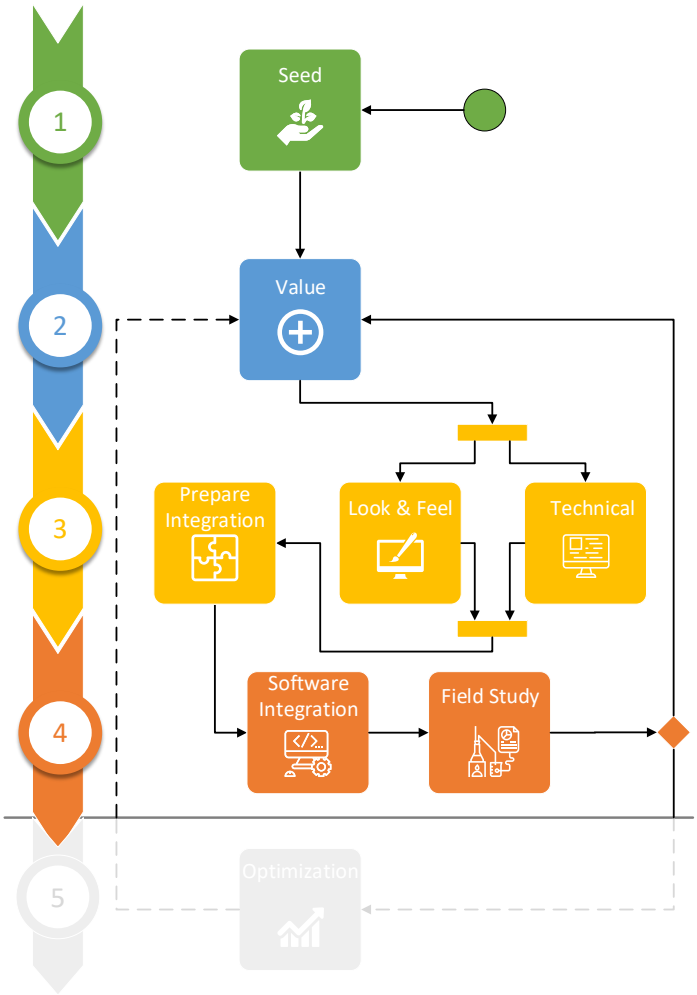



**Fig. 3.3.:** Integrated Process for *ICeDD* based on [KS03; May12; HH97; LMW11; JBR99]

**Stage (1): Initialize Design Thinking** The result of the Design Thinking process is highly dependent on an adequate design challenge. According to the Stanford d.school [@dsc16], the design challenge frames the process and should not constrain to one problem to solve nor leave it too broad which gives troubles in finding tangible problems. Ideally, it should include multiple characters, problems, and multiple needs of the characters, with the characters, problems, and needs in themselves being similar.

This is good to evaluate a design challenge in retrospect but helps only to a limited extent in finding such a challenge. To find appropriate design challenges and therefore initialize our approach, we propose the two possible paths *On–Site Feature Requests* and *Feature Requests from Systematic Analysis*.

*On–Site Feature Requests* is the idea of users stating asynchronously requests for improvements in a structured way. The necessity for this lies in Tacit Knowledge (cf. Gervasi et al. [Ger+13]) and the fact that certain knowledge is hard to recall without specific cues (cf. Gervasi et al. [Ger+13] and Benner [Ben84]). It is therefore important that users can define such requirements from their work context. In [Sen+18] we proposed a tool–guided elicitation process to empower users to do such requests in a structured way.

The other path of *Feature Requests from Systematic Analysis* is based on the analysis by an external person. Since our context does not allow a mere categorization or analysis (cf. section 1.1.1), a traditional systems analysis with the aim to examine an existing process in order to optimize it is not useful. Instead, we need a theory–generating approach that makes it possible to find unknown problems or solutions as well. In [Mei+16b; BMS20] we adopted *Grounded Theory* for software development to create such a theory generating approach.

With results from these two paths, a design challenge is created to start finding different solution designs and evaluate them with the help of *Design Thinking*.

**Stage (2): Execute Design Thinking with Non–Software** In this stage *Design Thinking* with the help of non–software prototypes and design challenges from the previous stages is carried out. The goal is to explore the problem and solution space with non–software prototypes to get a better understanding before software is used as a medium. Since *Design Thinking* is a methodology, it must be initiated according to the conditions (e.g. duration or stakeholders).

In our projects we have initiated *Design Thinking* as a one–day workshop format in which both developers and users participate. This workshop sensitizes the various stakeholders to each other and generates initial ideas. In a further step, these ideas are refined by the *Value Designer* (see Roles in paragraph 3.1) in coordination with the respective stakeholders.

The possible solutions should be reduced in this stage to at least two, but not more than five solutions. Result of this stage are non–software prototypes optimized on a *Value* Level and their documentation. Since the prototypes are more abstract than it is necessary for an implementation, they still must be prepared for implementation.

**Stage (3): Prepare Design Thinking with Software** Overall goal of this stage is to refine the prototypes on a *Technical* and *Look & Feel* Level and transfer them into requirements that can be used in a software development process (*Prepare Integration*). The task of the refinement on the *Technical* and *Look & Feel* Level is not to come up with novel solutions regarding these levels, but to align already existing solutions to the discovered value propositions from the previous stage. The *Value Designer* (see Roles in paragraph 3.1) supports the designer (*Look & Feel*) or the software developer (*Technical*) to ensure that the value is not lost.

The next step is to integrate all three levels (*Value*, *Technical*, and *Look & Feel*) on the requirements level (cf. Figure 3.3). The requirements should be specific and understandable but not include all underlying decisions to not overburden the developer. Nevertheless, it should be possible to understand the underlying incentives if necessary, in order to understand freedoms and make adjustments. Therefore, the requirements should be linked to the sources to allow traceability.

**Stage (4): Execute Design Thinking with Software** Using *Design Thinking* means evolution of the problem and solution understanding, experimenting with alternatives, and short learning cycles. Challenges in software development that arise from these have been listed in section 1.1.3. In order to overcome these, an adapted software development approach is required. To adapt such an approach, the 4P's (cf. Figure 3.4) described by Jacobson et al. [JBR99] are quite useful:

"The end result of a software *project* is a *product* that is shaped by many different types of *people* as it is developed. Guiding the efforts of the *people* involved in the project is a software development *process*, a template that explains the steps needed to complete the *project*. Typically, the *process* is automated by a *tool* or set of *tools*."

This means that it is not appropriate for a solution to consider only the process or only the product characteristics, since all the 4P's depend on each other. For example, product properties to enable incremental development may represent an overhead when a strictly sequential process is required (e.g. for legal reasons). Furthermore, tools can be needed to make a process practicable at all. Or people carry out processes differently because they

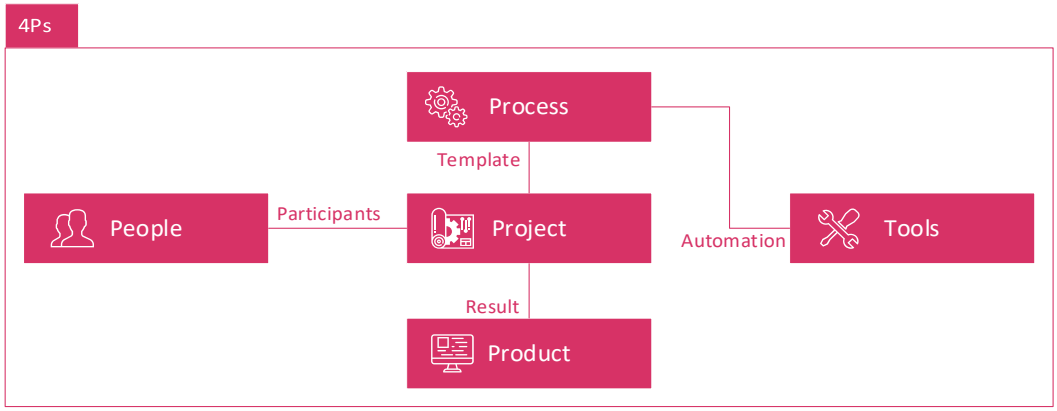



**Fig. 3.4.:** The 4P's People, Project, Product, Process, and Tools from the Unified Software Development Process [JBR99]. Own representation.

do not match with their mindset. Therefore, we describe in this section for all 4P's the characteristics, which makes *Design Thinking with Software* possible.

**People** It is important for the persons involved to understand that the artefacts created in this stage do not necessarily remain as they are and are partly discarded. Since at this stage the development of an understanding of the value comes first, people must be able to concentrate on the properties of the application that are necessary for the valuation of this application. A mindset with which the perfect application is to be developed immediately is not beneficial at this point. There should be a basic understanding of experimenting. Otherwise the same requirements apply as in agile software development.

**Product** As Denning et al. [DGH08] emphasize, traditional preplanned development focus on architectures that meet specifications from knowable and collectable requirements, do not need to change before the system is implemented, and can be intellectually grasped by individuals. This is not compatible with our requirements for *Design Thinking*, which is why we need evolutionary architectures instead.

Evolutionary architectures are designed for continuous adaptation through successive rapid changes or through competition between several systems [DGH08]. Ford et al. [FPK17] describe how evolutionary architectures can be achieved through appropriate coupling and allowing for incremental change. Patterns like Model–View–Presenter [Pot96] separate the presentation logic from the domain logic and therefore enable an independent development of the UI from the Backend. Event–driven architectures and Microservices allow more loosely coupled smaller components that can be polyglot regarding technology.

Event sourcing [@Fow05a] is particularly interesting in this context, as it enables parallel models, reconstruction of model states, and synergy effects, e.g. for the collection of interaction data. With parallel models the operation of different versions can be enabled on the one hand. On the other hand, data models can be adapted with less consideration of side effects. Model state reconstruction is useful for troubleshooting, preparing test environments, and data recovery.

In summary, evolutionary architectures help to provide fallback variants, increase reliability, develop individual components independently, and reduce the complexity of the individual components. By using synergies with event sourcing, it reduces the effort for data collection in experiments.

**Process** The requirement for this stage is the rapid implementation of software alternatives and conducting of field studies / experiments. In the previous stages an understanding of interacting dependencies, constraints, problem space, and solution space has been deepened. Therefore, the requirements at this stage should be stable to the extent that changes no longer occur in such a likeliness that cycles of several hours are useful and necessary. For this reason, these cycle lengths do not have to be supported in this stage and longer cycles can be considered, as in agile software development (cf. Terho et al. [Ter+17]).

DevOps including Continuous Deployment in combination with Agile Software Development is very suitable for this. According to Sharma et al. [SC17], DevOps' goal is to accelerate and increase the frequency with which production changes are made available in order to receive feedback from real users as early and frequently as possible. Consequently, these processes ensure the timely availability of alternatives for studies / experiments.

If you compare the Build–Measure–Learn Cycle from Lean Management (cf. Poppendieck and Poppendieck [PP03]) with the principles of the Agile Software Development (cf. Meyer [Mey14]) it stands out that Agile Software Development is focused on an acceleration of the Build step to increase the frequency of user feedback. The parts Measure and Learn and their expression are missing. However, these are indispensable in order to learn from their use.

To improve this situation, the implementation is embraced by a process for conducting studies / experiments. At this stage, field studies / experiments are to be used instead of controlled experiments. The advantage of field studies / experiments is the high external validity. We need this as we are still in a state where our mission is understanding and for that we need to reconstruct meaning or subjective perspectives from the usage in production. The disadvantage of field studies / experiments is the usually lower reliability compared to controlled experiments. At this point, however, controlled experiments would

not be appropriate, since they require a good knowledge of dependencies and confounding variables.

**Project** For each design challenge that ran through the third stage, a project is initialized. The aim is to understand which of the problems and solutions found lead to a value.

**Tools** In order to make studies / experimentation with software solutions more feasible, automation and assistance tools are needed. The use of evolutionary architectures, as described in product, in combination with containerization [Pah15], cloud architectures [Arm+10] and a build server for the continuous deployment pipeline enables independent automated deployment as well as roll–back of the individual alternatives, versions or components.

To make them available during a study / experiment a user specific online orchestration tool should be present. Its task is to orchestrate (e.g. by rerouting or feature toggles) the different alternatives, versions or components individually for each user. An opt–out for the user is essential so that he can continue to work productively with the system in the event of errors or malfunctions.

Since data (e.g. interaction data, surveys or free annotations) accumulate in experiments, a tool is required in which these can be stored, aggregated, and analyzed.

Finally an experiment management system is needed, that guides the users of *ICeDD* through the experiment design and controls the tools for orchestration and data accordingly.

**Stage (5): Optimization / Incremental Improvement** The main objective of *ICeDD* is to better understand the problem and solution space in cases of little or no knowledge. Once the fundamental interacting dependencies and constraints have been understood, it is possible, for example, to design a controlled experiment to learn in detail how the solution can be further optimized. Without this, it would not be possible to eliminate interfering factors and explain why the experiment has a valid operationalisation. Of course, other methods that are suitable for incremental innovation (cf. Norman and Verganti [NV14]) can also be used for optimization. This stage is listed here to emphasize that this is not a one size fits all approach. Accordingly, and because it can be achieved with existing methods, it is not the focus of this thesis and will not be discussed further below, unlike the other stages.

**Roles** In *ICeDD* we have the normal software development team as in Scrum as well as users. The difference is that instead of a product owner we have a role called *Value Designer*. Her main purpose is to ensure that intentions of the stakeholders are met, and value is delivered with the software application. In terms of Value Based Software Engineering [Boe06b] the *Value Designer* is responsible to identify all success critical stakeholders and to mediate between them. Therefore, she neither must be a domain expert nor a technology expert but needs considerable knowledge of both sides. She is the only role involved in all stages as well steps of *ICeDD*.

## 3.2 Related Work Regarding the Overall Solution Concept

In this section, we would like to situate our overall solution concept with regard to similarities and differences to related work. We will start with work that is closest to our concept, namely work that has also dealt with the integration of software development and design thinking. To identify them, we are using the chapter "Design thinking: A fruitful concept for it development?" from Lindberg, Meinel, and Wagner [LMW11] which is part of the "Understanding Innovation" book series that is mainly summarizing the fundamentals in design thinking. Hence, we assume that researchers working on the topic of integrating design thinking and software development will have to cite this paper to situate their own work into state of the art. Accordingly, we have used the cited by function of SpringerLink where the book series was published and Google Scholar to find papers that cited this very chapter and updated it in April 2020.

At SpringerLink, we got a total of 35 results and at Google Scholar a total of 121. In the first step, we reduced the results based on the title and abstract in terms of relevance for the integration of design thinking and software development. Correspondingly, 12 publications were left over at SpringerLink and 17 at Google Scholar (our own publications already excluded). From these publications, 11 were found both at Google Scholar and SpringerLink which leaves us with a total of 18 publications that are stating in the title and/or abstract that they handle the integration of design thinking and software development. 6 of these 18 publications turned out to be inappropriate on closer reading, as they did not go into detail about the integration of design thinking and software development.

7 publications [XAA15; Luc+17; Soh+19; ODr16; MV19; GSA16; Kow+14] were identified that integrate design thinking as a *Front–End Technique* into the software development process. This means that they stop with the milestone *x–is finished prototype* and use only one solution for the software development, which has the disadvantage that they cannot

experiment in the real world with software. To uncover certain dependencies or constraints like in the Netflix example (see section 1.1), at least two alternatives must be compared in an experiment in production. This is where our approach differs from the approaches mentioned in the publications, as we explicitly promote experimentation with several software alternatives.

Closest to our approach are the remaining 5 publications [DPU19; Dob+20; DP19; PA16; DP+17] by the two collaborating authors Dobrigkeit and de Paula. They separate their approach into the three phases *Design Thinking Phase*, *Initial Development Phase*, and *Development Phase*, whereas they state that the main difference "between the three phases is the ratio between Design Thinking, Lean Startup and development activities". The *Design Thinking Phase* has a similar concept to our *Stage (2): Execute Design Thinking with Non-Software*, but they presuppose the design challenge, which we develop explicitly in our *Stage (1): Initialize Design Thinking*. Their *Initial Development Phase* is actually a phase in which with mainly lean development a minimum viable product shall be developed by refining and testing the product vision from the previous phase with respect to desirability, technical feasibility and business viability. Therefore, they must have already reduced their ideas to a single solution idea or must do so in this phase. Testing will only take place in the next phase. Our approach differs from it again in the emphasis that several alternatives are to be tested with the software and we have explicitly defined how we want to transfer findings from Design Thinking into agile software requirements (cf. Meyer [Mey14, pp.119]) and thereby preserve findings in a comprehensible way. This is mainly due to the fact that, in contrast to Dobrigkeit and de Paula, we do not follow a process in which the same team shall participate equally in all tasks.

The final phase *Development* is for testing the *MVP* and gradually refining it into a product, whereas during the build you can have design thinking breakouts if necessary. Refining is quite similar to what we propose with *Stage (5): Optimization / Incremental Improvement* and testing is happening in our case already in *Stage (4): Execute Design Thinking with Software*. A fundamental difference is our understanding of the product. Whereas Dobrigkeit and de Pauly think of it more in the sense of a monolithic system, our understanding of it is a *System of Systems*. Accordingly, we see for our approach that for each new value proposition or feature it has to be passed through completely, from which a corresponding system for the system of systems is created or integrated into an existing system.

Apart from Design Thinking, there are also similarities between our approach and the idea of continuous experimentation. Continuous experimentation is an idea introduced by Olsson, Alahyari, and Bosch [OAB12], which is actually a combination of the ideas of continuous software engineering (see [FS14; Bos14]) and controlled online experiments (see [Koh+08]). Basically what they state is that you should continuously do experiments

and the possibility to do so is given to you by the efforts in continuous software engineering, especially with continuous deployment. Experiments are to be understood here as controlled experiments with a focus on quantitative data (cf. [OB14]). This is a fundamental difference to our approach where we focus on qualitative experiments. According to our approach, controlled experiments (sometimes also called A/B tests) would only be considered in *Stage (5): Optimization / Incremental Improvement*. Furthermore, the continuous experiments in this context are aimed at incremental innovations and not at novel and unique software-based solutions as we do. From the continuous software engineering we are using concepts such as continuous integration, testing and deployment. In the future it could be exciting to integrate further concepts from this area, such as the process model from Krusche [Kru16; Kru+14], to further increase the construct validity of the software development (the degree to which we implemented the software matches the intent of the ”specification”).

## 3.3 Summary

Developing unique and novel software-based solutions requires to validate different possible solution designs regularly with observable and triable prototypes in order to learn from them. One way to approach this systematically is Design Thinking, whereby the most sensible application in software development is still open. In our solution concept we present a variant in which Design Thinking is a hybrid variant between Front-End (only done before the development) and fully integrated approach (the same team works together the whole time). Furthermore, we describe how we can transition from prototypes reduced only on their value to software-based solutions that already contain enough background knowledge to further refine them with existing software development approaches. For this purpose, we have introduced stages that will be further elaborated in detail in the following chapters 4-7. The last stage, Optimization, is an exception to this, as it is only intended to highlight where the existing software development approaches, which focus on incremental innovation, can be applied.

As this is a solution concept and not the solution in detail, we will not start to apply our fitness function at this point and not discuss further in how far the solution fits our objectives. We have not defined enough details about our solution yet to decide for each characteristic of the fitness function to what extent it is met. Instead, we will do this later for each stage and summarize for the whole solution how far the fitness function is met in section 9.2.

Nevertheless, regarding the research question, we can already argue that this can be a way to develop unique and novel software-based solutions that provide users with value in the actual context of use. The reason for this lies in the usage of *Design Thinking*, which we

already found in the introduction (cf. section 1.1) to be very suitable for our needs. Design Thinking supports learning from multiple solutions simultaneously regarding dependencies and constraints and by further using that also with software, we ensure the usage of multiple solutions to learn from for the whole process (cf. Objective 2, Objective 3 and Objective 6). Objective 3 as well as Objective 4 is also supported by the integration of DevOps and evolutionary architectures. Design thinking is also attested to be very suitable for generating novel ideas, which is why Objective 5 is also very likely fulfilled. As already described, the extent to which these objectives are actually met will be analyzed in detail in the chapters for the individual stages.

# 4 ICeDD Stage (1): Initialize Design Thinking

In the previous chapter an overview of ICeDD as our development approach for unique and novel software-based solutions has been given. In this chapter, we will present the first stage of ICeDD, which is to initialize Design Thinking. First we discuss the requirements for this stage and give an overview of the stage in section 4.1. Essential for this stage are two paths that have the goal to create a design challenge. The first path *On-Site Feature Requests* will be presented in section 4.2, whereas the second path *Feature Requests from Systematic Analysis* will be presented in section 4.3. Our findings in this chapter are summarized and discussed in section 4.4.

## 4.1 Requirements & Overview

> *Every design process begins with a specific and intentional problem to address; this is called a design challenge. A challenge should be approachable, understandable and actionable, and it should be clearly scoped—not too big or too small, not too vague or too simple.*
>
> **— IDEO [IDE13]**

With the design challenge we define the frame within which we operate in our approach. If the design challenge is defined big and vague, it is more difficult to focus and dig deep. Therefore, more general results are produced that do not necessarily address a users problem. On the other hand, a very small and simple design challenge bears the problem that it doesn't allow for variations and explorations. In such cases we cannot be sure that that what we do is actually delivering a value.

Normally, a design challenge is about one of two types of problems (cf. [IDE13; @Roy+16; Sch17]). The first type of problem is something to create that is desired to exist but not yet be the case. The second problem type refers to already existing things that are either to be improved or changed. These problems can be reframed as "How might we ...?" questions as Schallmo [Sch17] and IDEO [IDE13] propose. The K12 Lab of the dschool at Stanford suggests on the other hand the following scaffolds:

- Redesign the _____(situation)_____ experience.
- Design a way for _____(specific group of people)_____ to better _____(situation)_____.
- How might we help _____(achieve some goal)_____.

Additionally to these scaffolds, the K12 Lab also provides resources to test design challenges for their applicability. Within this, for example, a test plan (The Challenge Generator) for the different phases of design thinking is included. Quite helpful is the decision matrix to decide if the design challenge is too broad or narrow. For example, if not multiple characters, multiple problems and multiple needs of the characters are included, the design challenge is too narrow. On the other hand if there are no similarities between characters, similarities between problems, or similarities between needs, the design challenge is too broad. These tests mainly help to validate a design challenge but not to come up with a good one.

IDEO [IDE13], Schallmo [Sch17], and Uebernickel et al. [Ueb+15] go a step further and describes a starting point for a design challenge. They suggest to list possible topics which results of "[..]all problems you've noticed or things you've wished for" and reframe them to "How might we?" questions in order to formulate design challenges. Their process for creating design challenges looks like this:

1. Deriving different topics including a short description
2. Discussion and assessment (potential as low, middle, or high) of the different topics
3. Choosing one topic

Even though Uebernickel et al. list concrete methods for this approach, all three sources remain very general in terms of implementation. They are particularly suitable for posthumously questioning the validity of the design challenges and possibly working on the formulation, but not for identifying the initial problems (improved / desired). For the identification of these initial problems we suggest the two different approaches *On-Site Feature Requests* and *Systematic Analysis* (cf. Figure 4.1). The main difference as already described in paragraph 3.1 is that the former is based on the idea that users are enabled to make feature requests as they are regularly working and the latter is based on an analyst called in to analyze the context.

But why do we need such a distinction? Our assumption from the introduction is that we are developing software systems that not only automate static well-known routines (digital copy) but also introduce unique and novel aspects (from the perspective of the application domain) into the software system. Pfeiffer and Suphan described the consequences of such goals:

> *In fact, sophisticated techniques of standardization and digitalization create new complexities and new areas of system-immanent unpredictability, not intentionally but nonetheless unavoidable. The ability to deal with those on an ad hoc and situational basis is a skill that comes from experience–and it does not fit into the standard routine/non-routine dichotomy.*
>
> **— Pfeiffer and Suphan [PS15]**

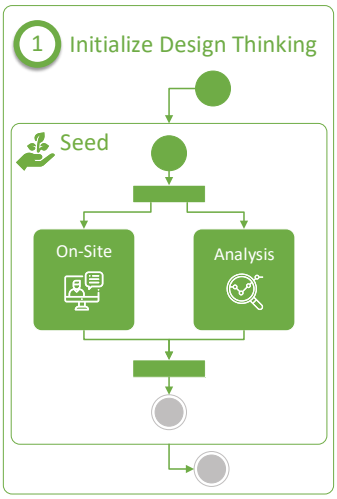



**Fig. 4.1.:** Initialize Design Thinking Process

As the value designer (see paragraph 3.1) often lacks specific production and process knowledge (cf. innovators in Pfeiffer and Suphan [PS15]), we usually need experts to give hints on what could potentially be interesting and to validate the proposed value. In contrast to other stages like novices, experts performance is not distinguished by the need of explicit rules, guidelines, or maxims but to her experience that allows her to connect her understanding of the situation to an appropriate action (cf. Dreyfus and Dreyfus [DD88] and Benner [Ben84]). This personal and contextual knowledge is also called tacit knowledge and "cannot easily be written down, formalised or aggregated. Since tacit knowledge cannot be expressed propositionally, exactly how particular work tasks are accomplished and any decision rules that may underlie any skilled performance cannot easily be provided" (Gervasi et al. [Ger+13]). Some tacit knowledge is hard to recall without cues (cf. Maiden and Rugg [MR96]) and in general a systematic process is needed to uncover tacit knowledge (cf. Gervasi et al. [Ger+13]). This is why we are proposing the two approaches of *On-Site Feature Requests* and *Systematic Analysis*.

That this is an ongoing problem in requirements engineering is indicated by recent studies like "Naming the pain in requirements engineering" from Fernandéz et al. [Fer+17]. They

designed a survey family on the status quo and problems in practical requirements engineering which they conducted with 228 companies from 10 different countries. The ten most cited requirements problems had been:

1. Incomplete and/or hidden requirements: 109 (48% named it)
2. Communication flaws between team and customer: 93 (41%)
3. Moving targets: 76 (33%)
4. Underspecified requirements: 76 (33%)
5. Time boxing/Not enough time in general: 72 (32%)
6. Communication flaws within the project team: 62 (27%)
7. Stakeholders with difficulties separating requirements from solution design: 56 (25%)
8. Insufficient support by customer: 45 (20%)
9. Inconsistent requirements: 44 (19%)
10. Weak access to customer needs and/or business information: 42 (18%)

The first 4 problems as well as problem 8 and 10 could be a direct result of tacit knowledge in requirements engineering leading to the described effects. Interestingly, *1. Incomplete and/or hidden requirements* and *3. Moving targets* were already among the most cited problems in the famous 1995 Standish Report [Gro+95]. Therefore, according to the current state of requirements engineering, it seems to make sense to place the two approaches *On-Site Feature Requests* and *Systematic Analysis* before Design Thinking in order to identify the Design Challenge. With this insight, the requirements for an approach to finding design challenges can be more precisely defined.

First of all, we focus on our *Fitness Function* (FF) from section 1.2. From this, the technology independent characteristics *Alternatives*, *Focus on Novelty*, and *Learning Cycle* are especially interesting for this stage. *Operating Alternatives* and *Consequences of Technological Decisions* are part of the technological implementation which is part of the later stages.

From the definition of a good design challenge it is immanent that the output of this stage is describing a multi-faceted problem for which the solution is not immediately apparent. With our goal of unique and novel software solutions we defined as well to stay in the *Chaotic* and *Complex* domain (cf. section 1.1.1). Hence, if the problem and solution space are already well known and solutions are immediately apparent or can be derived from analysis it would be more fitting to the *Complicated* or *Obvious* domain. But for these domains it would be better to directly skip to the fifth stage *Optimization* (cf. section 3.1). Therefore, our **first requirement** on this stage is that only results that include multiple characters, multiple problems and multiple needs of the characters as well as results for which solutions are not immediately apparent or can be easily derived by analysis are to be used for a design

challenge and the consecutive stage. All other results have to be either refined or can be used directly in the fifth stage.

Our default assumption is that we operate in the *Chaotic* and *Complex* domain, which means that problem and solution space are unknown to a high degree. This also implies that especially coming from *Chaotic* domain the only options are either by imposition or learning by diverging / converging or swarming to cross the boundaries (see [KS03]). The first would lead us to the *Obvious* domain, where solutions are immediately apparent. This would contradict our first requirement. Therefore, it is essential that we learn by means of divergence and convergence or swarming and thus already introduce learning cycles. For this implementation it is therefore necessary that a learning cycle explicitly exists that also integrates several alternative at the same time and thus is our **second requirement**.

Moreover, in order to achieve novelty, it is indispensable that the survey instruments do not only survey the current state but as well possible potentials (e.g. challenges or gaps), which is our **third requirement**. Otherwise, if we focus only on the well known problem and solution space, we would contradict the first requirement.

In addition to the requirements arising from the FF, we have already introduced the challenge regarding experts and the associated tacit knowledge earlier in this section. To further refine this for our requirements, we have a look at Müsseler [MR17, pp.607-609], who provides a more detailed description of experts based on psychological findings. He summarizes the key points as following:

- Experts encode problems more efficiently than novices and have better elaborated problem representations.
- Experts remember problem relevant information better. The better memory is due to broader and better organized knowledge, not to better basic cognitive capacities (e.g. wider memory span).
- Experts use different problem-solving strategies than novices.
- You become an expert through intensive practice.

Hence, expertism depends on learning. Which possible learning processes are assumed in science to become an expert, he also presents as following:

- *Storage of episodes*: Initially, problems are solved by a complex and step-by-step application of operators. At the same time, each individual problem solving episode is saved with its solution. The more often the same problem is solved, the higher the probability that the solution can be retrieved directly from memory.

- *Chunking of procedures*: If the same problem-solving operators have to be applied frequently one after the other, a new operator is formed which combines these operators. This allows problems that used to be solved in multiple steps to be solved in a single step.
- *Knowledge compilation*: Initially, problems with a certain factual knowledge (declarative knowledge) and general heuristics such as hill climbing are dealt with. If the same factual knowledge is repeatedly needed to achieve a particular sub-goal, a procedure is generated that allows the sub-goals to be achieved directly and much more quickly.

What these processes have in common is that by repeatedly solving the same problem, an optimized solution path in terms of usage of cognitive resources is created. So these depend on long-term memory. Besides the sensory register and the working memory, long-term memory is one of the three memories in Broadbent's memory model, which forms the basis of numerous multi-storage models [MR17, pp.403-434]. As a short explanation, the sensory register is an upstream processing in order to analyse many stimuli with limited processing resources in a less time-consuming way, e.g. to filter out the speaker's voice that one listens to and to "overhear" the others. The stimuli forwarded from this are transferred to the working memory, which is used for a greater processing depth (One can think of this analogous to a CPU cache). According to Miller, this can hold up to 7+/-2 Chunks[1] active, but only as long as they are constantly used and thus refreshed (see [Mil56; MR17]). Long-term memory differs from working memory mainly in the much greater amount of information that is available for a very long time.

The interesting thing of the long-term memory for our expert challenge is the way it is structured and working. It can be divided into a declarative (further subdivision into episodic knowledge and semantic knowledge) and a non-declarative system (further subdivision into perceptual knowledge and procedural knowledge). The declarative system stands for verbally reportable episodes and knowledge. In contrast to this stands the non-declarative, which existence is evidenced by a variety of situations in which experience shows an aftereffect even if it cannot be reported (e.g. try to describe how to tie your shoes or how to drive a bicycle.). Our challenge of tacit knowledge can therefore be traced back to this memory system.

Furthermore, Müsseler [MR17, pp.414-415] explains findings on context effects during learning and retrieval. With the learning of the "actually interesting" information, certain aspects of the learning context are always stored as well. For example, Godden and Baddeley [GB75] conducted an experiment with members of a diving school who learned a list of words on land and underwater. This was followed by a memory test that took place

[1] A grouping of single elements to larger meaningful units. For example, it is more difficult to keep the numbers 0,5,0,1,2,0,2,0 than the date 05.01.2020.

again on land or under water. The memory test showed that the number of freely reproduced words was high when the learning and testing phases took place in the same context. Each change of context between the learning and testing phases led to a deterioration of memory performance. However, such a drastic difference is not necessary as other experiments showed, where for example the mere variation of background color, font color and screen position led to similar effects.

From the structure and function of long-term memory, especially in relation to declarative and non-declarative systems and contextual effects, we derive our **fourth requirement**. The methods at this stage must at least partially query information in a context that is as close as possible to the user's work context in order to determine the presence of tacit knowledge and to activate as much knowledge as possible.

Sutcliffe and Sawyer [SS13] summarize the situation to deal with tacit knowledge from perspective of requirements engineering as an known unknown. From their "[...] experience and the evidence in previous surveys[...], interviews and workshops are more effective for tacit knowledge elicitation among the basic techniques since they approximate to natural conversations [...]". But they limit that all methods based on natural language communication, such as interviews, workshops and scenarios, carry the risk of ambiguous interpretation. This should also apply to observations, as user viewpoints are hidden and observer viewpoints may differ. Furthermore, they argue that "[...] the power of ethnography comes at the penalty of resources necessary for long-term observations, and the sampling to detect unknown unknowns is often a matter of luck."

For the reason of the sampling rate and our fourth requirement, we introduce in section 4.2 the concept of a tool-guided elicitation process to locate and scope problems and be applied by users independently from value designers for on-site feature requests. This is to ensure that the users can document opportunities as soon as they occur during their tasks and don't have to wait for a value designer with the risk of not remembering it as soon as she arrives. Such an approach has the challenge, however, that the users are not value designers and therefore do not have the necessary knowledge to set up requirements of the appropriate quality for software development or design thinking. Therefore, it is the goal to ensure a quality by tool support, which helps the value designer with the further decision making and which she can use as basis for further steps in the systematic analysis. Accordingly, the systematic analysis must fulfill all requirements as a refinement and alternative approach to on-site feature requests.

In the next sections we introduce our approaches to on-site feature requests (section 4.2) and systematic analysis (section 4.3).

## 4.2 On-Site Feature Requests

In this section we present our idea on how to make it possible for users to asynchronously give feedback on insights they gained during their work, problems they encounter, and visions they have for further developments. We assume that these users are experts who are not developers or requirements engineers. Furthermore, current solutions such as feature requests via e-mail or ticketing systems like Jira only take unstructured information. Therefore, we inevitable run into problems such as incomplete or hidden requirements and stakeholders with difficulties in separating requirements from known solution design [Fer+17]. Our solution for this challenge is to empower the users to participate independently and asynchronously in the requirements elicitation process via an assisting system that helps them to structure their thoughts so that they can straightaway suggest and describe experienced problems as well as new ideas and changes. To achieve this, we orient ourselves on methods already established in requirements engineering. The main contribution of this section, based on the paper [Sen+18] and the thesis [Pat17], is to discuss and evaluate the possibility of such a tool-guided elicitation process. For that we present

- the details of the idea of a tool guided elicitation process (see section 4.2.1),
- a classification of requirements elicitation techniques regarding their usability and usefulness to locate and scope problems and be applied by users independently from requirements engineers (see section 4.2.2),
- how we integrated selected elicitation techniques in our software prototype called Vision Backlog as a proof of concept that such a tool is technical feasible (see section 4.2.3),
- and the results of an initial study regarding the usability of the tool as well as the usefulness of the results (see section 4.2.4).

### 4.2.1 Towards a Tool-Guided Elicitation Process

Elicitation requires specific skills. Among all the stakeholders, only requirements engineers or value designer are familiar with those [Yoz14]. If stakeholders must depend on this role for elicitation all the time, the requirements engineer or value designer can become a bottleneck. Furthermore, in small organizations the requirements engineer or value designer is a relatively busy resource. Typically, naive stakeholders focus more on the suspected solution rather than the actual problem they are facing. Often, they are not even aware of the actual problem. But identifying the core of the stakeholder's problem is an almost necessary step to quality requirements [BR01] and therefore to a better understanding for a good design challenge. Additionally, notes, lists, or sketches which are mostly used to record needs

or expectations of stakeholders are not efficient ways, as they cannot naturally be tied to actual requirements [Yoz14]. Formal meetings are not feasible enough to thoroughly extract stakeholder's needs, expectations, etc. [Yoz14]. Stakeholders can get insights about their needs literally any time, especially when they are working on it. It could be hard during formal meetings to point out or recollect specific things [BR01; Bat+13; Mai+10].

Requirements elicitation is described as learning, uncovering, extracting, surfacing, or discovering needs of customers, users, and other potential stakeholders by Hickey and Davis [HD04]. In the requirements engineering process, elicitation is one activity besides analysis, triage, specification, and verification. Although most existing models show these activities in an ordered sequence, Hickey and Davis state that in reality, requirements activities are not performed sequentially, but iteratively and in parallel. This is an important insight for a tool-guided elicitation process, as it emphasizes that elicitation will never be unidirectional and must be conducted continuously.

Hence, a tool-guided elicitation process for users will never be a complete substitute for user researcher and requirements engineers as analysts. Instead, the activity of requirements elicitation will be shared among stakeholders and analysts with feedback mechanism between these two groups. A tool-guided process would not replace requirements analysts, but it will ease their work by reducing efforts and time spent on eliciting correct and complete requirements. Because of our use case, a tool-guided process must support and connect both stakeholders and analysts.

Just as much, the tool must support a learning process over the time like it is normal in approaches as Design Thinking, DevOps and Lean UX. Innovations are not adopted immediately by everyone at once but need time to prove their advantages. Usually they are adapted during the adoption process in a way that was not intended by the originators (see [Rog10]). Complex and chaotic problem domains need to be handled with a strategy that involves probing / acting as starting point to sense the effects and react accordingly to it (cf. [KS03]). Thus, we are safe to assume that the initial requests made with the tool need to be adapted and refined over the time.

To sum up, a tool-guided requirements elicitation process should capture the stakeholder's exact needs and expectations in the form of goals with associated context and always be available so that stakeholders can benefit from it at any time. The captured knowledge/information through this system shall help requirements engineers or value designers to extract tacit knowledge and to formulate more accurate and complete requirements. All the stakeholders shall actively participate in understanding the underlying problem to collectively reach an appropriate solution.

Besides the tool approach and its potential users, it must be defined which techniques should be incorporated. According to Maiden [Mai13], the primary function of requirements work is to locate and scope problems, then create and describe solutions. Hall [Hal13] once stated that the first rule of user research is to never ask anyone what they want. Nielsen [@Nie01] goes into the same direction and states that to design the best UX, pay attention to what users do, not what they say. Adding the 'I can't tell you what I want, but I'll know it when I see it' [Boe88] dilemma, it gets obvious that for user participation we should start with techniques that focus on locating and scoping problems instead of on creating and describing solutions. In the following, possible elicitation techniques are classified for a potential use in a tool-guided elicitation process.

### 4.2.2 Classification of Elicitation Techniques

Based on a literature research, we gathered requirements elicitation techniques from different disciplines like social science, design, usability engineering and requirements engineering (see Table 4.1 for the gathered techniques). These techniques need to be classified in accordance to their usefulness for our goals.

This classification can be done in many ways. One way is to classify them according to the means of communication they involve: conversational, observational, analytic, and synthetic [Zha07]. The conversational method is based on verbal communication between two or more people. Methods in this category are called verbal methods. The best example is interviews. The observational method is based on understanding problem domain by observing human activities. There are requirements which people cannot verbally articulate properly. Those are acquired through observational methods, for example protocol analysis. Analytic methods provide ways to explore the existing documentation of the product or knowledge and acquire requirements from a series of deductions which help analysis capture information about application domain, workflow and product features. Examples include card sorting. Synthetic methods systematically combine conversational, observational and analytical methods into a single method. They provide models to explore product features and interaction possibilities. An example is prototyping with storyboards. Although the techniques mentioned above make sense, these schemes are not much of a help considering our goals. The primary concern is that the stakeholders should focus on the problems they are facing and should not get distracted by solution or implementation details. Another challenge is that a technique should be representable in a software. Examining the literature that describes these techniques [McM04; ZC05], following classification criteria are established:

- **Locating and scoping problems:**
  Is the technique intended to locate and scope problem and not solution oriented?
- **Suitable for autarkic execution:**
  Can techniques be used individually, and it is not performed as a group activity?
- **Practicable for both stakeholder and analyst:**
  Can the technique be used by both stakeholder and analysts?
- **Representable in software**
  Can the technique be imitated as a software?

Direct answers to locating and scoping problems and suitable for autarkic execution can be found in the above-mentioned literature. Considering how much in-depth knowledge is required to use a specific elicitation technique, whether it can be used by stakeholders is indicated by practicable for both stakeholder and analyst, keeping in mind that stakeholders must not learn anything new. Representable in software classifies techniques as per their ability to be imitated as software. There already exist software which implement certain techniques [@Big18; Coo94; CRC07; ACO16]. Other techniques are included in our prototype and the usage is described in section 4.2.3. Techniques which fulfill a specific criterion are marked in Table 4.1.

| Technique | locating and scoping problems | suitable for autarkic execution | practicable for both stake-holder and analyst | represen-table in software |
|---|---|---|---|---|
| Interview [Coo94] | ✓ | ✓ | ✓ | ✓ |
| Questionnaire [ZC05; FF94] | ✓ | ✓ | ✓ | ✓ |
| Task analysis [ZC05; Wil+89; CK96] | ✓ | ✓ | ✓ | ✓ |
| Domain analysis [Pri90] | ✓ | ✓ | ✓ | ✓ |
| Observation [ZC05] | ✓ | ✓ | | |
| Protocol analysis [ZC05; GL93] | ✓ | ✓ | | |
| Prototyping [SS97] | | | ✓ | |
| Brainstorming [ZC05; Osb63] | | | ✓ | ✓ |
| Card sorting [ZC05] | | ✓ | ✓ | ✓ |
| Joint Application Development [WS95] | | | ✓ | |
| Scenarios [ZC05] | | ✓ | | |
| Viewpoints [ZC05] | | ✓ | | |
| SWOT analysis [PW98] | ✓ | ✓ | ✓ | ✓ |
| Theory of change [TC12] | ✓ | ✓ | ✓ | ✓ |
| Problem definition [KJ12; @Nes17] | ✓ | ✓ | ✓ | ✓ |
| Repertory grids [Kel03; FBB04] | | ✓ | ✓ | ✓ |
| Laddering [VZ09; Hin65; Gut82] | | ✓ | ✓ | ✓ |
| Literature review [Gre08] | ✓ | ✓ | | |
| Persona [CRC07; @Nes17] | ✓ | ✓ | ✓ | ✓ |

**Tab. 4.1.:** Classification criterion for elicitation techniques

### 4.2.3 Vision Backlog – A Prototype for a Tool-Guided Elicitation Process

Based on this classification, the elicitation techniques listed in Table 4.2 have been selected to be implemented in our prototype that we call *Vision Backlog*. The selected elicitation techniques intend to gather diverse information at different stages of the elicitation process. These techniques are practiced in different formats with different surrounding environment settings.

| No | Technique | Purpose |
|---|---|---|
| 1 | Interview | Structured stakeholder interviews are used to gather information about their personal attributes, their goals, business drivers behind them, underlying contexts and the domain |
| 2 | Questionnaire | Similar to the interview |
| 3 | Domain analysis | Used along with interview to gather domain specific information like vocabulary, specific terms used during execution of a specific process or a task |
| 4 | Task analysis | Used to gather information about the tasks stakeholders perform, subtasks and concrete steps, with contextual information like specific skills required |
| 5 | Problem definition | Used along with questionnaire to gather information about stakeholder's problems with current processes or tasks along with possible alternatives |
| 6 | Theory of change | Used along with interview to make stakeholder think about their high-level goals with tasks those help them achieve the goal, and potential risks etc. |
| 7 | SWOT analysis | Used to gather information about possible improvements, challenges and alternatives to the tasks stakeholders performs |
| 8 | Persona | Provide all the project stakeholders with a common understanding of their target user |

**Tab. 4.2.:** Selected elicitation techniques and their purpose in *Vision Backlog*

To be usable in *Vision Backlog*, we analyzed these techniques for common factors and found one in questioning. Based on this insight, we created in total 34 questions corresponding to the purpose of the different selected techniques. The questions are intended to ask information about stakeholder goals, activities, aptitudes, attitudes, and skills. These questions have been integrated in the stakeholder view of Vision Backlog. The stakeholder view is intended for stakeholders (like users) to be able to enter data about the task they perform, and more contextual data like why they perform this task, what tools or knowledge they require to perform it, and if there are any alternatives to this task. Along with these information, they provide personal information like education they have had, job designation, and skills they

possess. That is why the stakeholder view mainly separates in the two areas tasks and user profile. The user profile area (see Figure 4.2) is presented on the very first login with a help pop-up that explains the purpose of the application and the different functionalities including their importance. In this area the stakeholders create a personal profile. The questions we ask are based on Cooper's suggestions for Personas [CRC07] and generalized categorization for the adopter categories on how innovations are adopted by different groups of people from Rogers [Rog10].

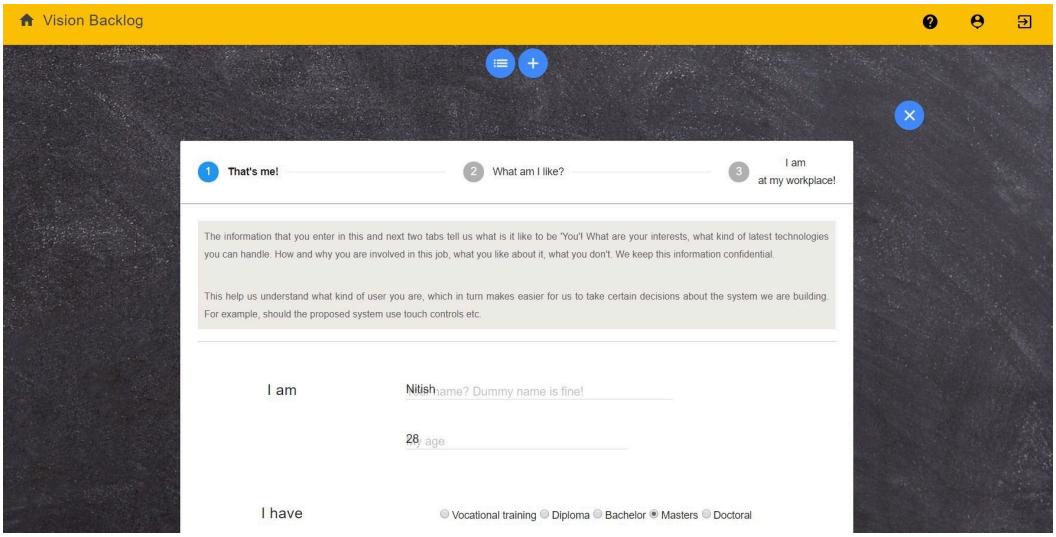



**Fig. 4.2.:** User Profile Creation

In the task area (see Figure 4.3), the stakeholder can create tasks as they perform them as part of their job. They can provide details about how frequently and how important the task is, the reason behind performing this task, and any improvements she can think of. This area is divided into four steps *Quick intro*, *Think about it*, *Supporting details*, and *Finishing details*. Within each step a description is given what this step is about. It is also possible to suspend doing the steps and continue them afterwards. Within the step *Quick intro* the user shall give an overview of the task, she wants to tell the analyst more about. The questions in this step are about a short description of the task, frequency, importance, his role, and an overview of the steps necessary in this task. In the next step *Think about it* a short rationale for this task shall be given before getting to the *Supporting details* step. In this step information regarding the context and the impacts of this task is given. In the last step *Finishing details* additional information about other involved people are given as well as possible factors for improvements.

Besides the stakeholder view with the user profile and task area, we implemented an analyst view (see Figure 4.4). This view is dedicated to the usability / requirements engineer to explore and discover inputs from the stakeholders. It has various filter and sorting options in this view. Additionally, it can structure the task descriptions from the stakeholders into

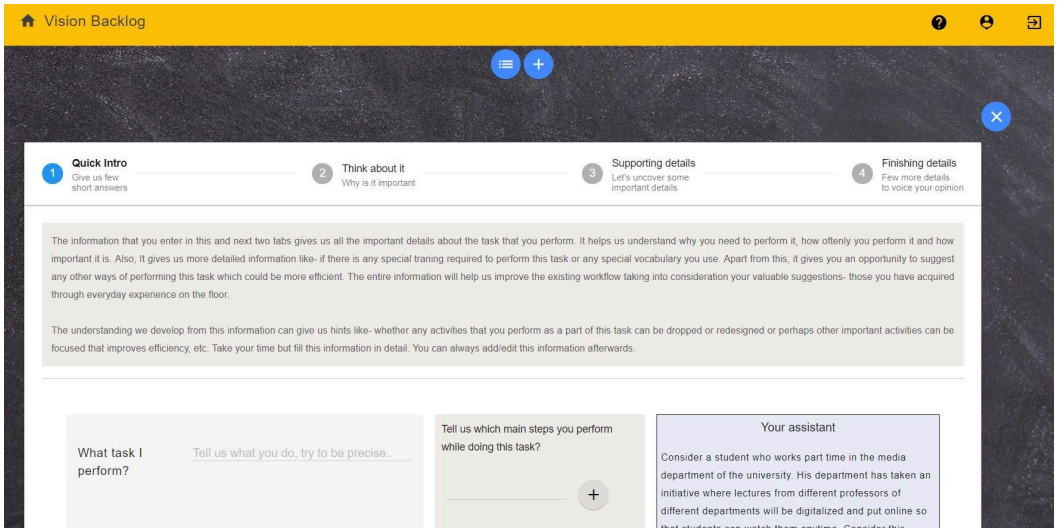



**Fig. 4.3.:** *Vision Backlog* – Stakeholder View: Task Creation

features for the further development. Besides these functionalities for sorting the created task descriptions, this view also includes a dashboard presenting the users with their attributes for creating personas as well as an overview of the system.

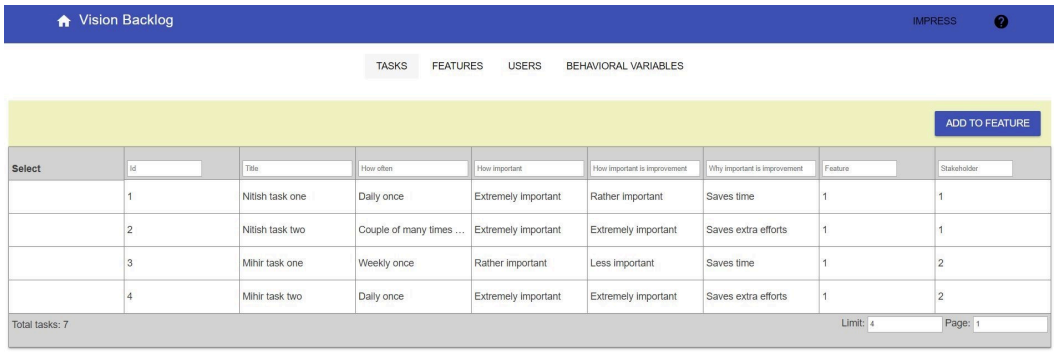



**Fig. 4.4.:** *Vision Backlog* – Analyst View: Task List Screen

### 4.2.4 Evaluation

We evaluated *Vision Backlog* with an initial two levels study consisting of a usability test and an expert review on the content quality. These two evaluations have been conducted non-consecutively. The entered data from the usability tests have been used for the expert reviews on the content quality.

The goal of the usability test was to show that stakeholders can understand the tasks presented in Vision Backlog and are able to use the tool with the selected elicitation techniques. We have been using the single evaluation from AttrakDiff [HBK03] to survey the usability

and attractiveness of our tool with a total of five participants. The usability test itself was done remotely and asynchronously. The summed-up results of the survey are presented in Figure 4.5. The meaning of the terms used in the diagram are as followed:

- **Pragmatic Quality (PQ)** indicates how successful users are in achieving their goals with the product,
- **Hedonic Quality- stimulation (HQ-S)** indicates to what extent the product supports human needs of developing something new in terms of novel, interesting and stimulating functions, contents, interaction and presentation styles,
- **Hedonic Quality- identity (HQ-I)** indicates to what extent the product allows user to identify with it and,
- **Attractiveness (ATT)** is the global value of the product to which Hedonic and pragmatic qualities contribute equally

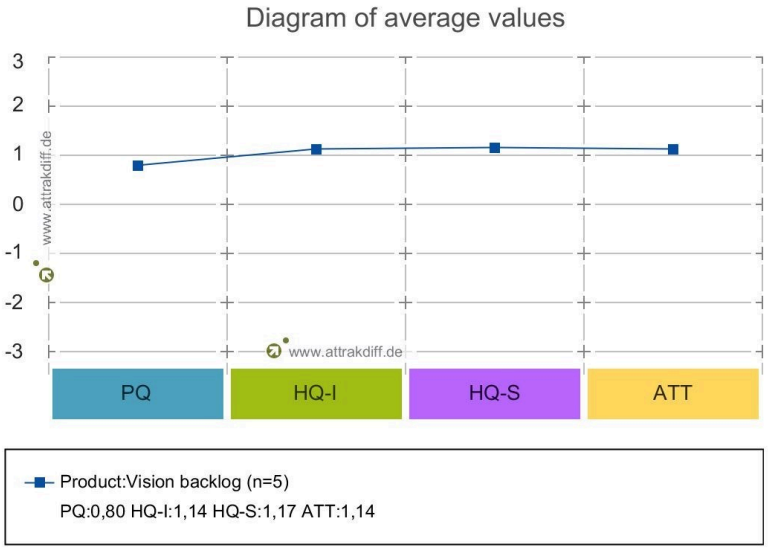



**Fig. 4.5.:** Usability evaluation results

The results indicate that Vision Backlog is rated positive, but the confidence level for PQ (1,62) and HQ (0,95) are quite huge, meaning that the participants evaluated the application differently. Overall, the participants have been successful in achieving their goals and rated the application attractive, but there is still room for improvement.

The goal of the expert review was to assess if the yielded results are useful for the further process, which is why all participants had a background in computer science and experiences in requirements elicitation. In total, we had three participants for the expert review on the content quality. They used the analyst's view and had to answer five questions afterwards in an online form with a four-point scale for the answers (see Table 4.3). Overall, the participants rated that the utilized techniques have been explained properly and how they

are related to requirements elicitation. They strongly agreed that the analytics provided by *Vision Backlog* were helpful but showed a differentiated picture regarding the representation of the analytics.

| | Strongly Agree | Agree | Disagree | Strongly Disagree |
|---|---|---|---|---|
| The concepts *Vision-analytics* uses i.e. Persona are explained properly | 66.7% | 33.3% | 0% | 0% |
| The application explains how the concepts used can be used for eliciting requirements | 66.7% | 33.3% | 0% | 0% |
| The semantic segregation of the data is helpful | 66.7% | 33.3% | 0% | 0% |
| The analytics provided are helpful | 100% | 0% | 0% | 0% |
| The structure used i.e. lists to represent the analytics are helpful | 33.3% | 33.3% | 33.3% | 0% |

**Tab. 4.3.:** Usefulness evaluation results

This marks the end of this section on on-site feature requests. As we mentioned in the beginning, this can only be a supplement to a systematic analysis, which we will present in the following section.

## 4.3 Feature Requests from Systematic Analysis

The systematic analysis at this point aims at exploring the problem space in order to develop a suitable design challenge for the design thinking process, which in turn leads to an improved understanding of the problem space as well as an understanding of the solution space and the connection between both spaces. Its goal is not to come up with specific functional requirements or qualitative requirements for already developing the solution but to understand the general conditions (cf. Pohl [Poh07] on the three types of requirements) and to gain an initial idea of possible features.

As we have mentioned in the previous section, Sutcliffe and Sawyer [SS13] propose for this stage techniques that approximate to natural conversations. In fact, they propose qualitative techniques without calling it like this. For this stage they are best suited as they try to reconstruct meaning or subjective views (see section 1.3), which is exactly what we need. Therefore, qualitative methods are first of all fundamentally suitable as main method, whether e.g. from requirements engineering, usability engineering or the social sciences.

But there is a restriction to it. Either alone or in combination with others, they must be able to develop new concepts in discussion with the empirical material and not rely solely on already known concepts for their implementation. Otherwise, there is a high probability that we will only move in the complicated or obvious domain (see section 1.1.1) with solutions immediately apparent or that can be easily derived by analysis. This in contrast would contradict our first requirement and wouldn't lead to a good design challenge.

One approach which aims at the development of new concepts, more specifically at the development of theories, is Grounded Theory. Grounded Theory also meets the second requirement through an explicit learning cycle. The fulfillment of the third requirement is not explicitly excluded by Grounded Theory, but must be ensured by the implementation of it.

### 4.3.1 Grounded Theory

Strübing [Str04] describes Grounded Theory as a practice which operates with a permanent iteration of the epistemological steps induction, abduction, and deduction, continuously developing, testing and modifying theories (cf. Figure 4.6). These epistemological steps are described by Peirce as following:

> *It [induction] never can originate any idea whatever. No more can deduction. All the ideas of science come to it by abduction. Abduction consists in studying facts and devising a theory to explain them.[...] Abduction is the process of forming an explanatory hypothesis. It is the only logical operation which introduces any new idea; for induction does nothing but determine a value, and deduction merely evolves the necessary consequences of a pure hypothesis. Deduction proves that something must be; Induction shows that something actually is operative; Abduction merely suggests that something may be. Its only justification is that from its suggestion deduction can draw a prediction which can be tested by induction, and that, if we are ever to learn anything or to understand phenomena at all, it must be by abduction that this is to be brought about.[...]The first starting of a hypothesis and the entertaining of it, whether as a simple interrogation or with any degree of confidence, is an inferential step which I propose to call abduction.*
>
> **— Peirce**
> As cited in [AND94]

Muller and Kogan [MK12] describe Grounded Theory as "a set of practices for exploring a new domain, or a domain without an organizing theory[...]. The practices are strongly grounded in the data and the theory is said to emerge from the data." Furthermore, they describe research with it as something that "[...]usually begins with a broad very shallow set of unorganized information[...]. Over time, through a series of disciplined procedures[...], the information becomes more narrowly focused, and understood in greater depth[...]."

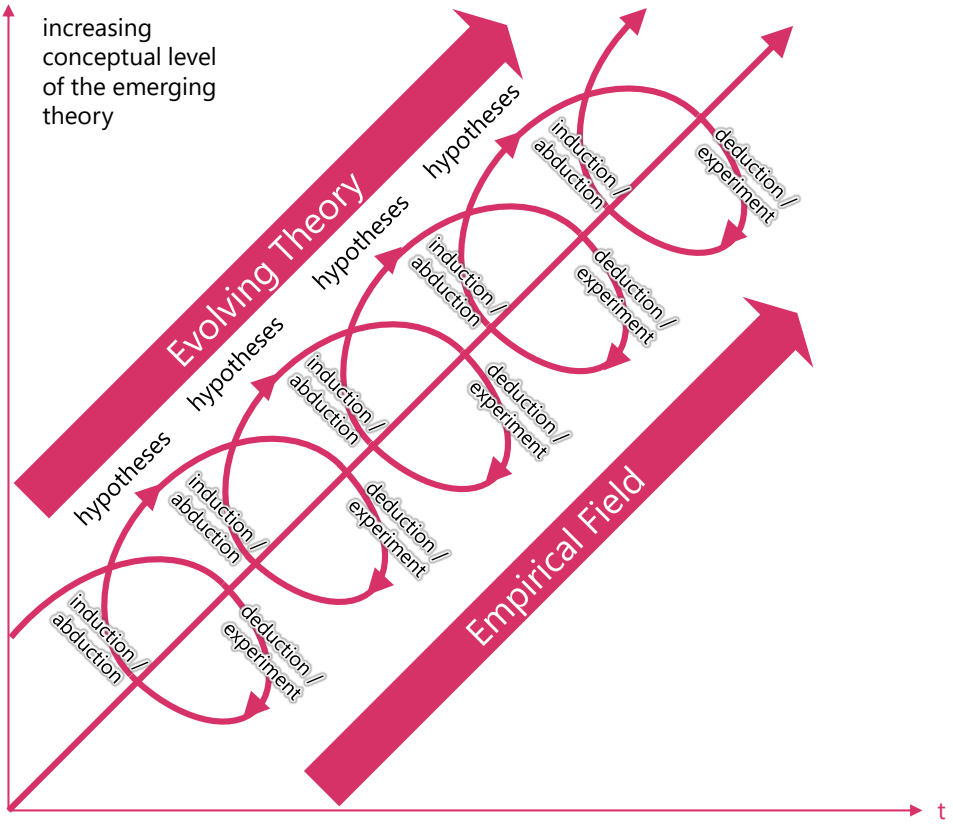



**Fig. 4.6.:** Schematic process model of grounded theory based on Strübing [Str04]

In other words, it is an agile process in which, in contrast to agile software development, not a product but a theory is developed and to a certain degree validated step by step. As Strauss and Corbin [SC94] point out, the means used for this purpose overlap with those of other qualitative research methods. Thus, quantitative data can be used as well as combinations of qualitative and quantitative techniques of analysis. The main difference between Grounded Theory and other qualitative methods is the emphasis upon theory development.

The advantages and disadvantages of using grounded theory are worked out by Muller and Kogan [MK12]. They highlight that Grounded Theory is useful to explore new domains or a domain without a dominant theory, for constructing a theory of this new domain, and to avoid a premature conclusion about the domain. However it should not be useful for testing a hypothesis, or for trying to prove or disprove a theory.

For the understanding of the core tasks in grounded theory, it is important to remember that qualitative research's goal is to reconstruct meaning, as mentioned in section 1.3. A central task of qualitative data analysis for this purpose is therefore to create an interpretative gateway to the data material obtained, i.e. to make sense of the data. In Grounded Theory, this process is called coding and is distinguished into two possible variants (see Strübing [Str04]). The first variant focuses on coding using predefined categories based on the research question and a subsequent analysis. This idea equals, for example, with the qualitative content analysis (cf. Gläser, Laudel, and Grit [GL09] or Mayring [May00]). Whereas in the case of the second variant, which is aimed by Grounded Theory, it is assumed that a theoretical framing does not yet exist and in this case coding is not understood as the subsumption of qualitative data among existing concepts, but as the process of the development of these very concepts in discussion with the empirical material. This implies a continuous analytic comparison with the data to successively develop these concepts.

Strauss and Glaser [Str04] have developed a three-stage coding model consisting of open coding, axial coding, and selective coding to achieve this. Open coding is the writing of simple descriptive labels (see Muller and Kogan [MK12]) to break up the data by analytically preparing individual phenomena (see Strübing [Str04]). In the next stage, axial coding, relationships among the codes are searched for to create a phenomenal context model (see Strübing [Str04] or Muller and Kogan [MK12]). The stage of selective coding serves to integrate the theoretical concepts developed to date in relation to these few core categories (see Strübing [Str04]), i.e. the axial codes that are more important than others will be determined (see Muller and Kogan [MK12]).

Although Grounded Theory originated from sociology [SC94] it is as well widely used in software engineering. Muller and Kogan [MK12] give an overview for this, ranging from the developers' orientation to new projects, software testing and quality processes, to developers' views on user interface design issues and team cohesion. Summed up, Grounded Theory has been used to explore the needs of scholars in media studies and for an examination of software engineering methods.

Grounded Theory can therefore be used particularly well in this stage to initially build up a valid knowledge base and understanding and to translate this into a design challenge. Furthermore, the approach of Grounded Theory can be extended into product development with the help of Design Thinking to more deeply validate the theories and insights gained from the social-scientific approach through implementation in products and testing them in practice. This results in an ideal collaboration between Grounded Theory and Design Thinking.

### 4.3.2 Our Grounded Theory Instance

Since Grounded Theory in its basic form is a methodology rather than a specific method, it offers many freedoms to shape the research process. The exact design can therefore be formulated according to the research perspective and the research phenomenon [MM11].

In [Mei+16a], we present an instantiation of grounded theory that was developed in close collaboration between media researcher and computer scientists for the project centre for music edition and media (ZenMEM)[2]. A schematic visualization of this grounded theory instance is presented in Figure 4.7.

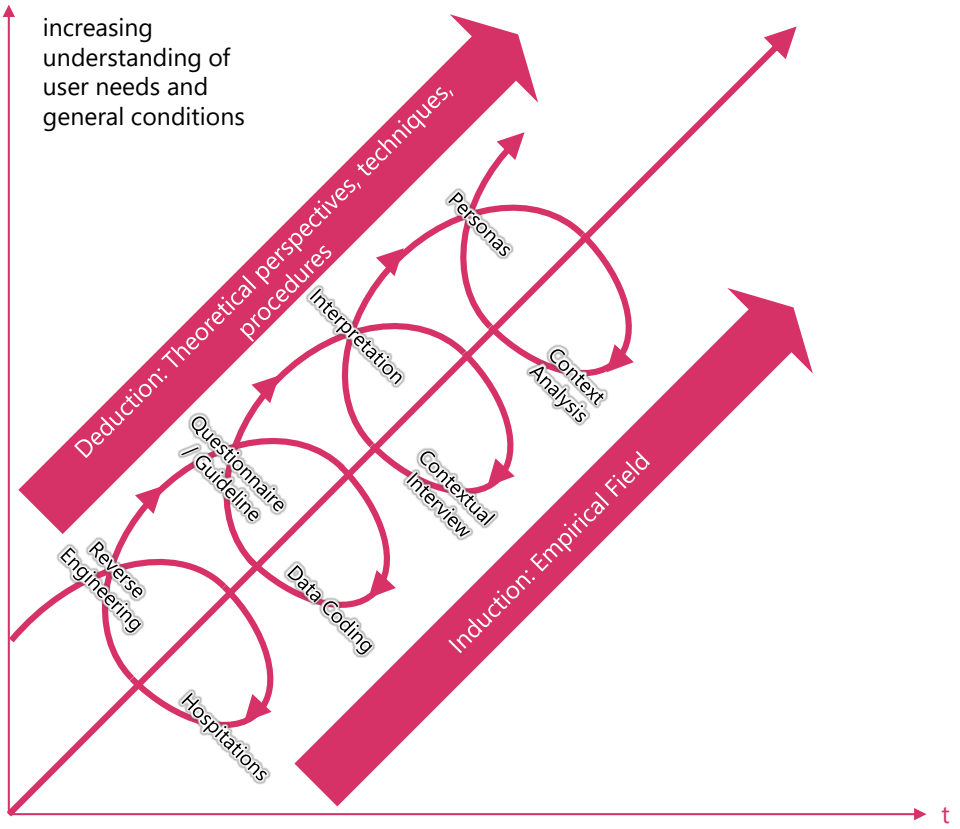



**Fig. 4.7.:** Schematic process model of the interdisciplinary research process with grounded theory (following Strübing [Str04])

With ZenMEM, we didn't start on a completely greenfield site, but with Edirom[3] we had a digital tool for creating digital music editions that already supported rudimentary functions such marking measures and with MEI[4] an XML schema for encoding music editions. Nevertheless, there were still many manual and non-digitized activities (cf. Rittmeier, Engels, and

[2] `https://www.zenmem.de`
[3] `https://www.edirom.de/edirom-projekt/software/`
[4] `https://music-encoding.org/`

Teetz [RET19] on identifying digitalization potentials in business processes) in this area. For this reason, we started grounded theory with a reverse engineering phase of the existing digital tools regarding the offered functionalities and to identify first structural weaknesses of the used data model. The resulting insights offer first clues for question dimensions for a quantitative questionnaire and a qualitative guideline.

Furthermore, the insights from a musicological hospitation were used, where the different approaches of the scientific work of music editors were explained, in particular with the Edirom software. A quantitative questionnaire and a qualitative guideline was then developed with this and with technical literature. The quantitative survey focused on the overall population with its characteristics, in particular its software and technology affinity, and usage of Edirom. Since the quantitative method is used to generate overviews of many users, it was supplemented by a qualitative method in order to go into depth and investigate, for example, implicit knowledge, unconscious routines or habitualized working methods.

The quantitative questionnaire was realized as online survey as the target group is a technically special and geographically strongly distributed group of persons. It is subdivided into four major thematic complexes: "How does scientific work and editorial activity work?", "How is Edirom used in scientific work?", "How are the participants involved in external scientific communication?", and "What sociodemographic and musical background do the participants have and what is their technology affinity?". Within the questionnaire, mainly closed questions were used, as these are characterised by a better comparability of the answers.

As instrument for the qualitative guideline, the narrative guideline interview with a problem-centered fraction was selected in order to address implicit knowledge, work routines, expertise and certainties. The advantage of this decision is that it allows the greatest possible freedom with regard to the formulation of questions, strategies of questioning and the order of the questions as well as for the narration of the editors (cf. [Hel11]). In total, interviews were conducted with eight editors that lasted between 90 and 180 minutes and were then evaluated with a variant of Strauss and Corbin's coding, as suggested by Przyborski and Wohlrab-Sahr [PW09]. On the one hand, these provided concrete hints for optimizing the software, but also context information on the working conditions, routines and experiences of the editors. In addition, they provided very good insights into the change processes of knowledge work, knowledge management and the knowledge acquired as such.

Furthermore, the qualitative interviews have revealed that not only editors are involved in creating music editions, but also assistants who digitize and marking measures on the selected sources and then specify the concordances. This process could therefore not be investigated by the interviews and therefore required further empirical investigations. Since these activities are very standardized processes, the assistants were observed and questioned

in their work with the help of a contextual interview (cf. [BHB04]), which is a common method in usability engineering.

In the next step, the users are described with this knowledge. User roles or personas are suitable for this. User roles according to Constantine and Lockwood [CL99] in their simplest form are lists of characteristics such as needs, interests, expectations, behavior and responsibilities. In contrast, personas (cf. [CRC07]) are not stereotypes, but archetypes based on empirical data. Thus, in contrast to user roles, they can be used as an abstraction of real persons, which is why we use personas to model users. The data from the previous investigations are excellently suited for this type formation.

With the data and knowledge gained in this way, design challenges can be construed in a well-founded manner that meet the requirements as defined in section 4.1.

## 4.4 Summary and Discussion

In this chapter we have presented the first stage of ICeDD, our approach to develop unique and novel software-based solutions. The main goal of this stage is to identify a fitting design challenge as it sets the scope for all further Design Thinking activities and thus the first essential step for the execution of Design Thinking in the context of software development. Even though the design challenge is so important, the existing literature hardly addresses how to identify a good design challenge, but rather how to validate it posthumously. For this reason we present two ways to identify this design challenge for the further development of unique and novel software-based solutions.

The first way is through on-site feature requests (section 4.2). Literature still presents tacit knowledge as an important challenge in requirements elicitation. In a follow-up investigation, we created a link between tacit knowledge and findings on expert knowledge in psychology. From these findings we concluded that we need an assistance system that enables the user to document findings regarding issues and possible solutions in a structured way while performing his normal tasks as knowledge is often only activated in the context it is needed. Accordingly, we developed a prototype (section 4.2.3) of such an assistance system and evaluated it (section 4.2.4) by means of a usability test for end users and an expert review for the usage of results by requirements analysts.

As on-site feature requests cannot guarantee that the users are already documenting everything like it is needed for a good design challenge, it may need to be supplemented by a systematic analysis. The systematic analysis (section 4.3) also functions as a starting point that can be triggered without specific on-site feature requests from the users. Furthermore,

goal of the systematic analysis is not to document functional requirements for the digital copy of something existing, but to understand the general conditions and come up with theories on how the situation can be improved with unique and novel software-based solutions. Accordingly, we need a theory-generating procedure at this point, which we have found with Grounded Theory (section 4.3.1) and have transferred it interdisciplinary into a usable instance in the ZenMEM project (section 4.3.2).

Overall, we have met the requirements we set ourselves for this stage (section 4.1). The first requirement is actually something that comes at the end of this stage and is dependent on the domain context. Our second requirement is regarding the learning cycle, which is also a characteristic in our FF. By using Grounded Theory in the systematic analysis and having the systematic analysis also as a refinement instrument for on-site feature requests, we have an explicitly defined learning cycle, as Grounded Theory itself emphasizes the iterative learning from data. Moreover, Grounded Theory is a methodology that can and should be adapted to the current context. Even though we have presented one possible variant as a concrete instance, Grounded Theory can also be used in another development project in different variant. Furthermore, Grounded Theory is not limited to one possible solution, but also allows for the highlighting of different approaches, e.g. through the selective coding. Accordingly, in this stage we fulfil the highest form of the characteristic *Learning Cycle* of the FF, since we intended a learning cycle that is explicitly defined, refers to several alternatives and can be adapted according to the context. Furthermore, we fulfill the *Alternatives* characteristic of the FF at least partially, because at least two solutions can be supported simultaneously in parallel with grounded theory, but we cannot yet guarantee that they are implementable as software.

The third requirement is referring to novelty and we have defined that it is indispensable that the survey instruments do not only survey the current state but as well possible potentials (e.g. challenges or gaps). Grounded Theory is originally intended to describe existing states as well as possible by iteratively improving the understanding of the state as a theory generating procedure based on empirical data. However, it does not have to be limited to this and can also be used to iteratively develop and improve new ideas, e.g. by means of expert interviews. These can be incremental innovations, but also completely new solutions. However, the limit here is the restricted significance without an observable and triable prototype (see section 2.1 and section 8.2). Nevertheless, with this we can assume that Grounded Theory does not merely copy the current state but also is able to creates novel solutions, which is completely fulfilling the characteristic *Focus on Novelty* of the FF.

The results regarding the FF are presented visually in Figure 4.8. We haven't looked at the other characteristics of the FF in this stage and therefore don't fulfill them as they are technology related and we don't yet look at this in this stage. Besides the first requirement

and the requirements deriving from the fitness function, we had one other requirement, namely that methods at this stage must at least partially query information in a context that is as close as possible to the user's work context in order to determine the presence of tacit knowledge and to activate as much knowledge as possible. The on-site feature request path allows us to exactly do this, but we can also adapt the grounded theory for example by using contextual inquiry [BH99] to achieve this goal.

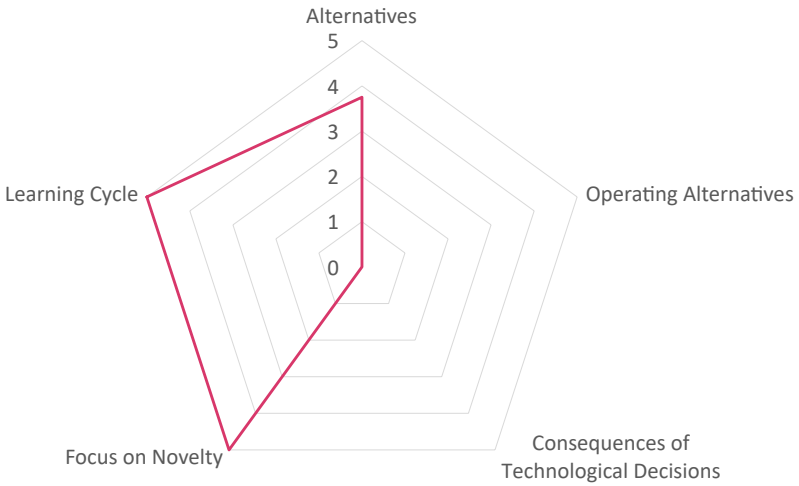



**Fig. 4.8.:** Radar Chart for Stage 1 regarding our *Fitness Function* (FF).

The most important findings in this chapter are the requirements for methods to derive design challenges to develop unique and novel software-based solutions. The two paths on-site feature requests and systematic analysis should be there to ensure that as much knowledge as possible is gathered for potential solutions. However, the manifestations that we have presented for these two paths represent only one possible variant. Neither the presented tool for the on-site feature requests nor Grounded Theory with our instance of it must be used. However, in our context both have proven to be very advantageous. For generalizations, though, further extensive studies have to be done in this area, since we have mainly focused on the feasibility.

# 5 ICeDD Stage (2): Execute Design Thinking with Non-Software

In this chapter we are presenting the second stage of our approach ICeDD to develop unique and novel software-based solutions. Within the first stage (see chapter 4), we have identified a possible design challenge that we will use in this stage to conduct design thinking in order to learn from prototypes more about the problem and solution space. Design Thinking in this stage refers mainly to the identification and validation of value by means of prototypes that do not use software as a medium. The reasons for this and other requirements are described in section 5.1. As Design Thinking is a methodology like Grounded Theory, we need to instantiate it for our purposes. In section 5.2, we are presenting our instantiation of Design Thinking. The findings leading to this instantiation from conducting several Design Thinking runs are presented and discussed in section 5.3. Finally, this chapter and the second stage are summarized and discussed in section 5.4.

## 5.1 Requirements & Overview

As can be seen e.g. from the recommended user experience design process [May12] or the basic idea of value-based software engineering [Boe06b], value should be identified first. In the previous stage, we identified a design challenge with the underlying constraints that will form the basis for further steps to identify and validate the value in this stage. The restriction of the systematic analysis in the previous stage is the validation capability, since it doesn't produce trialable and observable objects as required for a better understanding of unique and novel software-based solutions (cf. paragraph 2.1). Furthermore, it is mainly limited to existing solutions, which means that not all effects that may occur with a unique and novel software-based solution can necessarily be observed. For this reason, the aim of this stage is to further deepen and validate the existing understanding of the problem space with the help of trialable and observable objects, as well as to identify possible solutions, and to align problem and solution space in order to further identify and validate the value.

In our case, the trialable and observable objects are prototypes of possible solutions for the intended problem. Furthermore, we restrict the properties that these prototypes should have as we are still on the initial identification and validation of the value. As a short reminder, value means in our case something that is of relative worth, utility, or importance for a user. Hence, the prototypes in this stage need mainly the properties related to value (cf. prototype levels in section 3.1). All additional properties from the technical or look-and-feel level can or would be unnecessary overhead for the validation level of this stage[1]. This leads to our **first requirement** for this stage, which is to have trialable and observable prototypes that have only the complexity needed to validate value at this stage.

For the next requirements we will refer again to the single characteristics of our FF (see section 1.2). First of all, we can exclude the characteristics *Operating Alternatives* and *Consequences of Technological Decisions* for this stage. *Operating Alternatives* is related to operating software alternatives in production and for *Consequences of Technological Decisions* we would have to make lasting decisions regarding technologies in this stage. But both would imply that we introduce unnecessary complexity as right now only properties of the value level are important and the technology level is subsequent. On the other hand, the characteristics *Learning Cycle*, *Focus on Novelty*, and *Alternatives* must be considered at this stage.

We need to test with alternatives to improve our understanding. Only by testing with alternatives we can derive from the differences and the assessment of these what in detail is beneficial or not. Also, as our goal is to develop unique and novel software-based solutions and the preceding systematic analysis couldn't have been tested yet in practice, we assume that our understanding of problem and solution space will still change radically after the tests. Hence, we will need several iterations to consolidate our understanding. That is why our **second requirement** is a learning cycle that is also explicitly defined and refers to several alternatives at a time.

As the learning cycle needs alternatives to be able to increase understanding even more than the systematic analysis, we need to support at least two solutions simultaneously in this stage. Furthermore as we assume that our understanding will change radically in this stage, we need multiple iterations to consolidate our understanding which in conjunction with our first requirement means that we should focus in this stage on non-software as a medium (cf. section 1.1.3 or section 3.1). In addition, we should have the option to create the solutions in parallel to increase the speed we can iterate through the learning cycle even

[1] For example, one could introduce filtering according to distance from the user position as an added value for an event portal. This could be done e.g. by the distance in length (e.g. km) or by the travel time by car, bicycle or public transport. To validate the value at this stage, it is unnecessary to introduce algorithms for routing and distance calculation; it is sufficient to demonstrate the idea in a prototype using static examples without any application logic at all.

further. Hence, our **third requirement** is to support at least two solutions simultaneously, that can be created in parallel.

Within the first stage, we have ensured that the design challenge includes multiple characters, multiple problems, and multiple needs of the characters as well as problems for which solutions are not immediately apparent or can be easily derived by analysis. This is so, so that we can stay open minded, explore further the problem and solution space as well as to find alternative solutions that are maybe better suited. The goal is not to stick to already established solution paths, this is why our **fourth requirement** is to create novel (e.g. in the sense of structure, tasks, or processes) alternative solutions in this stage in contrast to already established solution paths.

Design Thinking (cf. section 2.2) is particularly suitable for meeting the requirements of this stage. The diverging and converging thinking over the problem and solution space (see Figure 2.4 in section 2.2) ensures to create novel solutions (fourth requirement). Combined with the micro cycle (see Figure 2.5 in section 2.2), we have an explicitly defined learning cycle that intentionally includes different alternatives (second requirement). Furthermore, the focus of Design Thinking is to learn with prototypes as trialable and observable objects (first requirement) and as well to create with the diverging and converging thinking at least two solutions simultaneously in parallel (third requirement).

As Design Thinking is more like a methodology, there is not the one Design Thinking process. Therefore, a concrete instantiation of Design Thinking is needed. In the next section we thus describe in detail the developed instantiation of Design Thinking for our approach.

## 5.2 Our Design Thinking Instance

In section 2.2 we have presented design thinking in general. The most important concepts include the alignment of different perspectives (see Figure 2.3), converging and diverging thinking in the problem and solution space (see Figure 2.4) and the design thinking micro cycle (see Figure 2.5). For the micro cycle, we have presented that there are different versions and orders. To instantiate our design thinking instance, we have to choose the version we want to use. We decided to stick for that with the order and definition of the Stanford d.school[2] as we think it is more open minded to start with trying to understand the users before defining the problem and therefore restricting ourselves. Furthermore this allows us to reuse resources offered by the Stanford d.school to conduct Design Thinking.

[2]For a more detailed look into how literature defines the different stages have a look at [@Wal12].

Our design thinking instance is mainly based on the *Design Thinking Playbook* by Tran [@Tra16] and has some minor adjustments based on the insights we gained during conduction of several design thinking runs. Accordingly, in the following we present Design Thinking after the *Design Thinking Playbook* and explain our changes to it. First of all, it consists of the five phases *Initiate*, *Discover*, *Create*, *Learn*, *Post-Work* (see Figure 5.1), whereas the phases *Discover*, *Create*, and *Learn* represent the design thinking micro cycle and are intended to be iterable.

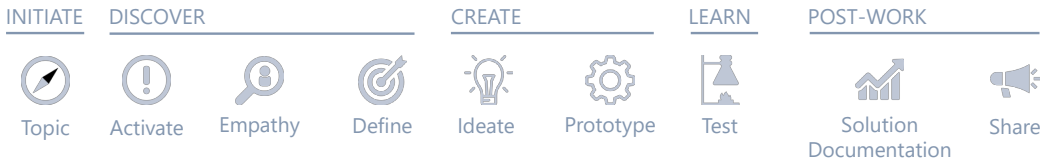



**Fig. 5.1.:** Design Thinking Process based on Design Thinking Playbook by Tran [@Tra16].

In contrast to the *Design Thinking Playbook*, which was intended as a 6 week sprint, our design thinking can either be conducted as design sprint (like in section 8.3) or as condensed workshop format, as we found that a workshop can be more suited in our context (see section 5.3). The main difference between design sprint and workshop format is the duration. Google for example introduced design sprints in their context that lasts five days (see [KZK16]). In our context we define design thinking workshops as design thinking instances that lasts no longer than two days and everything else as design sprint.

To explain our instance we will use a design thinking workshop we conducted in the context of education and training in industry 4.0. The starting point is the *Initiate* phase that we usually begin with an impulse presentation (see [Sen20d, slides 1–19]) to introduce the participants to the design challenge. After a short introduction to design thinking and the most important rules of it (see [Sen20d, slides 20–32]), the design challenge is presented explicitly (*Topic*) as the workshop goal, which is available for everyone printed out to be read during the entire workshop (see [Sen20c]). This initiation is the first difference in the process to the *Design Thinking Playbook* as they didn't intend to give an impulse presentation to give the participants a glance about what the design challenge would be, nor did they intend to explain Design Thinking in general. In our case, we consider both to be necessary, as our participants are usually a diverse mix of people and do not necessarily already have an understanding of what design thinking is all about, and some of them have not yet been working in the domain covered by the design challenge.

After this, the design thinking micro cycle starts with the *Discover* phase. As the first step (see [Sen20d, slides 35–36]), the participants context knowledge regarding the design challenge is activated (*Activation*), which includes the affected stakeholders, gaps, values, and basic conditions. It is carried out as individual work in which the participants should

write down their knowledge within a certain time frame (in the workshop about 10 minutes) without paying attention to correctness. This is like in the *Design Thinking Playbook*, but we will use in the following the memory model of humans to further explain why we kept it and think it is useful. Müsseler [MR17, p.419] explains as a widespread concept of semantic memory that it is working like a hierarchical organized semantic network. This means, for example, that there is a node "water" with a link to the nodes "boat" and "watermelon" but no direct link between "boat" and "watermelon". There are three key concepts for working with this model. First, a node gets a higher activation if a concept is mentally processed that is linked with the node. Secondly, the activation of one node increases the activation of linked nodes. And thirdly, in phases of inactivity, the activation of all nodes decrease. Hence, if the participants start with an activation step, the activation of nodes related to the design challenge should be increased and therefore more knowledge should be available in the next steps. In addition to the memory, we also use this step to put the participants' initial assumptions on paper (see [Sen20j, working sheet 1]), so that in the next steps contradictions become clearer compared to the other participants.

The next step is *Empathy* in which the participants should interview stakeholders and check their assumptions from the first step. In our workshop format the participants interview each other in pairs (see [Sen20d, slides 37–39]). To help them, they are told to pay attention to "interesting statements", "problems", "opportunities", "interpretations", "ideas", and "insights". Subsequently, they should note this down on sticky notes accordingly. Due to the time available, these are the only aids for the participants in the workshop format for conducting the interviews. In the case of a sprint, the participants receive further information on how to conduct qualitative interviews (see [Sen20a] or section 8.3.3.2). Furthermore, in the sprint they already work together in the subgroup, which remains until the end of the sprint.

In the next step *Define* (see [Sen20d, slides 40–42]), the participants work together in subgroups (up to 8 people) to first formulate POVs and then problem definitions. For the POVs, they are introduced to use the sticky notes from their interviews and write them in the POV template of working sheet 2 (see [Sen20f]) as needs and not solutions. For the problem definition they can thus draw on the collected points of views from the entire subgroup and formulate at least five problems on worksheet 3 using the criteria "only one user group", "a real problem", "no solution in question form", "invites to different solutions", "inspiring and motivational" from Tran [@Tra16]. Furthermore, the participants shall assess the problem definitions and in case of the workshop decide on one (which can also be a combination) to further work with.

The step *Define* finished the first phase *Discover* and with that as well the exploring of the problem space. With the next phase *Create*, the solution space is going to be explored.

For this, the first step *Ideate* is there to create as many ideas possible on how to solve the problem and be reduced to the most promising one with diverging and converging thinking. Accordingly, in the case of the workshop we have divided this step into four substeps (see [Sen20d, slides 43–49]).

As the first substep, the participants have the task to create as many "crazy" ideas as possible. To encourage them to do so, they first got an impulse with science fiction ideas that got reality and Clarke's three laws. This impulse is as well different to the *Design Thinking Playbook* as we wanted to explicitly encourage the participants to think outside of the box. Secondly, they were put under time pressure to write down 50 ideas in ten minutes on sticky notes, so that they are not able to overthink their ideas and already assess them regarding feasibility. However, this excludes group work, so this substep is done individually.

In order to better discuss these ideas in the subgroup, in the next substep five ideas are substantiated using worksheet 4 (see [Sen20f]). These are then shared, thematically ordered, and prioritized within the subgroup in substep three. As last substep, with the help of working sheet 5 (see [Sen20f]) a common idea is created in the subgroup to create a prototype for. The common idea can be one idea selected from these substantiated ideas or a combination of different ideas.

With the common idea, the next step is to create a prototype (see [Sen20d, slides 50–55]) which shall focus on the value level with regard to its properties as to our first requirement for this stage. To support the creation of prototypes, the participants get a testing card (see [Sen20f, working sheet 6]) to fill out in advance. On the testing card they are encouraged to think about what they want to test with the prototype, how they will do it, how they will measure it, and when they are right. It has to be kept in mind, that the purpose of the prototypes in this stage are not to examine the technical feasibility but are just a means to transport and validate an idea.

The validation of the idea brings us to the next phase *Learn* with the step *Test*. In the original process this phase included as well *Share* but we chose to put it at the end, because it is a good conclusion in our workshop format and after the intensive workshop the participants usually do not have so many comments in this step. For the *Test* step, the participants actually get two working sheets. The first working sheet is the learning card (see [Sen20f, working sheet 7]) which actually reflects the testing card and is for observing how the prototype is used by the people outside of the group. The second working sheet is the feedback grid (see [Sen20f, working sheet 8]). This is the for the people trying out the prototype to structure their feedback. It is important not only to criticize the negative aspects, but also to point out what you liked best, what should be improved and what was not understood. Only through this more detailed feedback it is possible to better understand the problem and solution space

and to adapt the ideas accordingly. To reflect and use the feedback promptly, part of this step is also the adaptation of the prototype or the idea.

The *Post-Work* phase in the original process by Tran [@Tra16] included in this phase a step called solution rollout, which is mainly a roadmap on how to create the product. In our case this stage is still far from a comprehensive understanding of the final product. This will only be developed in the next stages of the approach. Accordingly, it is essential that the findings from this are recorded accordingly for the subsequent stages. On the one hand, this includes the working sheets of the individual participants / groups including the sticky notes, on the other hand also the final prototypes of this stage. For the prototypes, we created a poster (see [Sen20e]) to summarize the core aspects of the prototype to not have just the prototype itself.

In the last step *Share* this stage will be finished by sharing the final results with the other groups. For the workshop format we choose a format where the groups will go from one prototype to the next and the group of the prototype will present it. A trying out of the prototypes would take too much time for the amount of participants usually involved in this stage.

In this section we have presented a design thinking instance based on the *Design Thinking Playbook* by Tran [@Tra16]. Its main purpose is to explore problem and solution space on the value level. Therefore, the steps in this instance are restricted to the value level. This includes that the prototypes have mainly properties that are required to validate ideas on the value level, which is why mostly wireframes, physicial prototypes made out of paper or lego, videos and theatre, story telling metaphor or customer journeys (see [Sen20a, slides 60–68]) are used in this stage.

## 5.3 Findings

The presented Design Thinking instance in the previous section is largely based on Tran's *Design Thinking Playbook* [@Tra16] from Stanford d.school (K12 Lab Network). As part of the Hasso Plattner Institute, Stanford d.school is one of the driving forces behind Design Thinking research and as such is also heavily involved in a book series on Design Thinking at Springer Link[3]. Hence, they are well informed about the state of research on Design Thinking and how to best conduct it. Furthermore, this book series gives a great overview on evaluations of Design Thinking effectiveness and efficiency in general, but also in particular for software development. Thus, in this section we will not focus on the evaluation of Design

[3] `https://link.springer.com/bookseries/8802`

Thinking in general or its applicability in software development, but rather present our findings from the conduction of our Design Thinking instance on several occasions.

In total, we have conducted seven Design Thinking workshops (one to two days) and two Design Thinking sprints (accompanying the semester) based on our instance between 2015 and 2019[4]. For each run, we had 17 to 60 participants (learning 4.0) with one run consisting only of computer scientists and the other runs of computer scientists and at least one other area but a maximum of four other areas.

For our workshop format we have experienced seven hours as the lowest limit to go through the Design Thinking process and get useful results (see Table 5.1 for an example schedule). Although with the Wallet Project [@Bot16] there is a Design Thinking workshop, which is designed for 90 minutes + debriefing, it is mainly aimed at giving an impression of Design Thinking and not on developing a real product in the given time frame. For working on a more complex topic even the seven hours make the impression on the participants that the workshop is tightly packed. Some parts like *Crazy Ideas* are intentionally kept very short to increase the pressure and force the participants to focus. In the case of *Crazy Ideas*, this means that the time constraint forces them to stop thinking about the feasibility of their ideas and just write them down without reflection. In general, however, the times have been chosen in such a way that there is enough time to exchange thoughts, but not so much that large parts of the group become idle.

The workshop format is especially beneficial to create awareness (in the project and stakeholders) of the different point of views, be it from stakeholders of the same or a different area. It is as well quite beneficial to fundamentally align the different perspectives. Due to the tight time frame, however, the ideas cannot be prepared in such a way that they are already suitable for technical implementation. Therefore, especially after the workshop, the prototypes have to be refurbished further until they can actually be realized as software.

On the contrary, in case of a sprint, the time frame could allow for more elaborated prototypes. However, the challenge here is that the risk that parts of the group become idle is higher than in the workshop and this can lead to tension in the group. Firstly, we have observed that there may be a discrepancy between the desired activities and those of Design Thinking (cf. section 8.3.4.1). For a small period of time, this is still acceptable for the participants, but dissatisfaction increases with the length and intensity of the period. Secondly, when working on tasks, it may happen that one part of the group is better suited to do them and

[4] see e.g. https://www.sicp.de/nachricht/news/design-thinking-heiliger-liborius/
https://www.sicp.de/nachricht/news/design-thinking-grenzen-ueberwinden/
https://www.sicp.de/nachricht/news/kultur-digitalisieren-sicp-ist-mitinitiator-beim-infotag-owlkultur-portal/
https://www.sicp.de/en/nachricht/news/lernen-40-wie-sieht-die-berufliche-bildung-zukuenftig-aus/

| Time | Step |
|---|---|
| 09:00 – 09:30 | Introduction / Topic / Impulse (cf. [Sen20d, slide 1–34]) |
| 09:30 – 09:40 | Activate (cf. [Sen20d, slide 35–36]) |
| 09:40 – 10:00 | Empathy / Interviews (cf. [Sen20d, slide 37–38]) |
| 10:00 – 10:15 | Empathy / Interview Results (cf. [Sen20d, slide 39]) |
| 10:15 – 10:30 | Define / Point of Views (cf. [Sen20d, slide 40–41]) |
| 10:30 – 11:00 | Define / Problem Definition (cf. [Sen20d, slide 42]) |
| 11:00 – 11:10 | Ideate / Crazy Ideas (cf. [Sen20d, slide 43–46]) |
| 11:10 – 11:25 | Ideate / Substantiate Ideas (cf. [Sen20d, slide 47]) |
| 11:25 – 11:45 | Ideate / Share Ideas (cf. [Sen20d, slide 48]) |
| 11:45 – 12:00 | Ideate / Common Idea (cf. [Sen20d, slide 49]) |
| 12:00 – 13:00 | Lunch Break |
| 13:00 – 14:00 | Prototype (cf. [Sen20d, slide 50-55]) |
| 14:00 – 15:00 | Test (cf. [Sen20d, slide 56–57]) |
| 15:00 – 15:30 | Solution Documentation (cf. [Sen20d, slide 58–59]) |
| 15:30 | Share (cf. [Sen20d, slide 60]) |

**Tab. 5.1.:** Example Time Schedule for a Design Thinking Workshop

thus works on them more than other parts of the group. The longer this task is worked on, the more present this imbalance becomes. In addition, there is a danger that not all participants can allocate the same amount of time for Design Thinking over longer periods of time, which also creates an imbalance. Overall, these effects can lead to resignation and idling and a negative effect on the outcome of the Design Thinking run.

Sometimes we could observe that some groups / participants had problems to understand what they should do or to focus on the task. In the feedback discussions with the students from the project group (see section 8.1 and section 8.3) we were told that they often understood Design Thinking and the meaning behind the individual steps only after the second run. In addition, some of the participants lacked the experience to build prototypes creatively and quickly. For these reasons, it is recommended that each group be assigned a "supervisor" who already has experience with Design Thinking and who will help the group to focus, to intervene quickly when uncertainties arise, and to give impulses on how to build prototypes, for example. This is only possible to a limited extent when supervising several groups.

## 5.4 Summary and Discussion

In this chapter we have presented the second stage of our approach ICeDD, whose primary goal is the identification and validation of value and thus also the further exploration of the problem and solution space. To ensure that we are doing this, we have gathered requirements in section 5.1 which turned to Design Thinking into a good fit. As Design Thinking, similar

to Grounded Theory, is a methodology, it needs to be instantiated for the context. This instantiation for our context is described in section 5.2, which is mainly based on the *Design Thinking Playbook* with further refinements from our side. Additionally, we have summarized our findings from conducting Design Thinking in our context in section 5.3.

By working with several groups per Design Thinking run it was always possible to create in parallel at least two different solutions simultaneously (third requirement). Furthermore, the participants were able to detach themselves from already established solutions in the process and thus produced solutions in general that were novel in the sense of structure, tasks, or processes (fourth requirement). The creation of trialable and observable prototypes was not successful in all cases, as some prototypes could only be understood through explanations. But in general the particpants were able to create such prototypes (first requirement). Especially in case of the workshop format, the participants created prototypes with non-software that were focussed on the necessary complexity to validate the value. In our instance but as well in Design Thinking in general a learning process is hard encoded in the process. That this worked in our instance is substantiated by the changes made by the participants after each learning phase. Moreover, the individual work phases and the exchange with the other groups ensured that the learning process referred to several alternatives at a time (second requirement). How this is reflected in our FF is presented in Figure 5.2

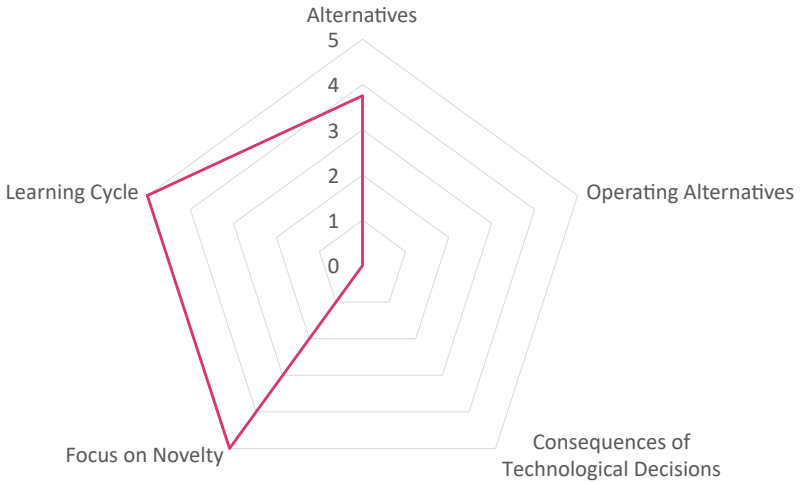



**Fig. 5.2.:** Radar Chart for Stage 2 regarding our *Fitness Function* (FF).

# 6 ICeDD Stage (3): Prepare Design Thinking with Software

Lindberg, Meinel, and Wagner [LMW11] argue that there is a risk of a fundamental disruption of the knowledge flow between design thinking and subsequent development stages. In this chapter, we are presenting our process to prepare Design Thinking with software including its requirements (see section 6.1) that shall support the knowledge flow, but also allowing us to use Design Thinking in a hybrid approach like mentioned in chapter 3. In addition to a refinement of the Technical and Look-and-Feel level as well as the integration of value, technical, and look-and-feel, this stage is mainly about the transformation of the findings from the Design Thinking from the previous stage into agile software specification / documentation. As this transformation with the specification / documentation is novel, we developed a framework to do so, which we present in section 6.2 including its feasibility evaluation in section 6.2.4. Our findings in this chapter are summarized and discussed in section 6.3

## 6.1 Requirements & Overview

In the first stage of our approach (see section 4.1), we have identified a possible value proposition described as design challenge. This design challenge was taken in the second stage (see section 5.1) to initialize a design thinking instance to further elaborate and validate the problem and solution space regarding the value. For this purpose, among other things, prototypes were produced which were optimized in their properties to learn about the value proposition. To achieve this, these prototypes are 'very experimental and consist of any material that allows achieving information about the ideas behind the [value proposition] (and not so much about its technical specifications)' [LMW11]. This is a fundamental difference to normal software prototypes, which are generally made of the same material as the final product and are meant to be iterated into the final solution as soon as possible [LMW11]. Furthermore, in the second stage, requirements were identified with design thinking, which in their form (e.g. interview transcripts or *Point of View* (POV)) do not yet correspond to the otherwise usual technical specifications (e.g. atomic requirements shell (see Figure 6.1 for an

example) or user stories) in software development. For these reasons, Lindberg, Meinel and Wagner [LMW11] see 'the risk of a fundamental disruption of the knowledge flow between [...] design thinking and subsequent development stages due to dissimilar communication media used in design thinking and IT development' if used as a front-end technique and not with an integrated design thinking strategy (cf. section 1.1).

| | | | | | |
|---|---|---|---|---|---|
| **Requirement #:** | **A12** | **Requirement Type:** | **Functional** | **Event / use case #:** | |
| **Description:** | The user interface must be operable via various input options. | | | | |
| **Rationale:** | In addition to the classic input via mouse and keyboard, the execution view also includes input via touch. Setting up the individual devices for an exercise is most likely done using a Tablet PC. In addition, the user interface for post-processing must be operable via another form of interaction. | | | | |
| **Source:** | Björn Senft | | | | |
| **Fit Criterion:** | The user interface can be operated via mouse + keyboard as well as via touch without unwanted interactions. | | | | |
| **Satisfaction:** | | | **Dissatisfaction:** | | |
| **Dependencies:** | | | **Conflicts :** | | |
| **Supporting Material:** | | | | | |
| **History:** | Created on 04.02.2013 | | | | |

**Fig. 6.1.:** Example of the atomic requirements shell used in the firefighter training system (see section 8.2).

In our case, we have decided for a mixture of front-end technique and integrated design thinking strategy. This means that we do not consider design thinking to be completely finished after the previous stage, but continue it in this and the next stage. However, we limit the continuation to the extent that we do not use a one team approach where all team members work together on the same task at any time. Instead, individual team members can work more independently in their own domains (e.g. UI Design, Software Architecture, Algorithms, or Business) to deliver the proposed value as intended with concurrent set-based engineering (cf. [War+95]). An exception is the value designer (see paragraph 3.1), who is responsible to mediate between the success-critical stakeholders and has to ensure that the value is considered and delivered (cf. paragraph 3.1 and [Boe06b]). This is why she has to be included in each stage and step. It follows from this, however, that there is a certain risk of disruption of the knowledge flow as seen by Lindberg, Meinel and Wagner [LMW11]. Even though, the risk should be significantly lower than in the case of pure front-end design thinking, since the value designer is continuously involved and all success-critical stakeholders should have been made aware of each other at least in the previous stage.

Accordingly, we have developed a process for this stage that is illustrated in Figure 6.2. It consists of the steps *Transform*, *Look & Feel*, *Technical*, and *Prepare Integration*. The first step is actually the transformation of the design thinking results to be used in software development. The further steps are based on the technical and look-and-feel level of prototypes

with the subsequent integration of these levels with value as introduced in paragraph 3.1 in accordance with the recommended ux design process by Mayhew [May12].

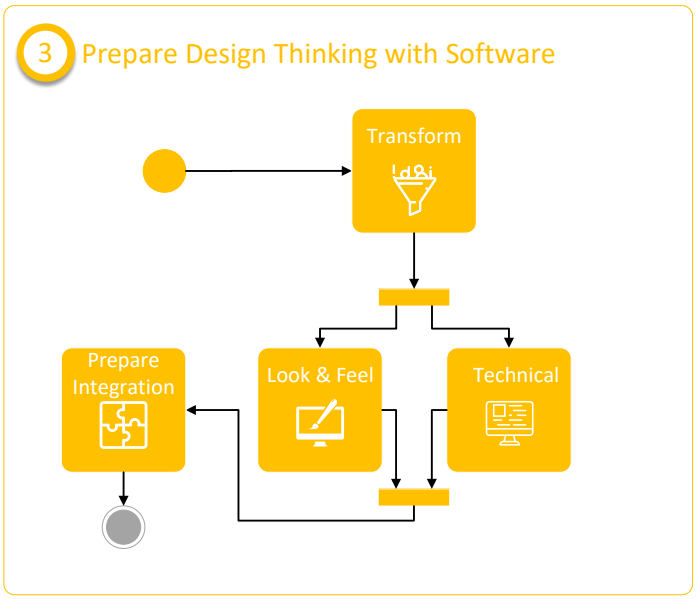



**Fig. 6.2.:** Prepare Design Thinking with Software Process Overview.

The first step in this stage to transform design thinking results into agile software requirements (cf. Meyer [Mey14, pp.119]) is essential to support the knowledge flow but also to have documentation for the future to understand decisions and what parts can be changed. There are various things to consider for the transformation of the design thinking results into agile software requirements. First of all, we can look at the objectives and fitness functions for our approach (see section 1.2) again to determine the general requirements for this stage.

Our most important requirement is embedded in our research question that we want to provide value (in the sense of Boehm [Boe06b]) for the user in the actual context of use. Accordingly, the preceding stages are designed to identify and initially validate such a value. Thus, the most essential element that we can draw from the previous stages is value. This also means that all further software requirements are derived from this value. So, our **first requirement** at this stage is:

**Requirement 6.1.1.** The value of the unique and novel software-based solution must be explicitly defined.

Furthermore, we assume that understanding of the problem and solution space remain subject to change. Accordingly, the software requirements are not yet final and may change. To avoid that the software requirement specification have to be constantly rewritten in its

entirety, the **second requirement** is the establishment of a requirements hierarchy with the basic value at the root and further refinements the further down you go. With this concept, sub hierarchies can easily be removed or changed in the software requirements specification. It also ensures that technical requirements cannot be defined detached from the value and therefore it supports the traceability of decisions as well as the understanding of what can be changed.

**Requirement 6.1.2.** In this stage, the software requirements are first defined in a requirements hierarchy with the basic value at the root and further refinements the further down you go.

The traceability of decisions related to value brings us to a further requirement, the traceability of decisions based on insights from the previous stages. Besides the value proposition, insights are made in the previous stages that further explain or limit certain aspects. These can be included in e. g. interview, test, or observation documentation. To not overload the developers with information and increase the clutter, it is desirable not to write these information in its entirety directly into the requirements specification. However, it can happen that developers have to use this information to clarify requirements or feasibility of a solution idea. Therefore, the **third requirement** is:

**Requirement 6.1.3.** As part of the software requirements specification the references to the information from the previous stage are maintained.

Moreover, as already described several times, we can only be sure about the validity of an idea when it has been used in the actual context of use and has proven itself there (cf. e.g. the Netflix example section 1.1, objective 2 and 3 section 1.2, or the application case study of diffusions of innovations section 8.2). It is therefore not enough to test just with non-software prototypes, but different (see objective 2) prototypes must be implemented as actual software and tested in the actual context of use (see objective 3). Consequently, our **fourth requirement** for the transformation is:

**Requirement 6.1.4.** For each value at least two solution alternatives are represented in software requirements and in best case can be created in parallel (see fitness function *alternatives*).

These aforementioned requirements refer to the transformation of design thinking results into software requirements, a further step is the elaboration of the technical level. As described in the previous stage (see section 5.3) the prototypes have been refurbished to a point that a rough idea on how to implement them on a technical level is given. However, this does

not mean that they can actually be implemented as they were only tested regarding their value (or desirability, see section 2.2) and not their feasibility. Therefore, at this stage, more specific solutions must be developed at the technical level in terms of value and tested using technical prototypes. This also includes the development of a micro architecture and the consideration of the macro-architecture.

To explain the decision regarding architecture, we have to first explain the difference between micro and macro-architecture. The macro-architecture represents the general idea of the system and its fundamental architecture decisions, e.g. on structures, components, data stores or architecture styles, whereas the micro architecture refers to the architecture within a specific component and its detailed design (cf. [HW95; Ger+16; KSW13]). There might be cross-functional values which are related to all components, but values identified from the previous stages will usually relate to specific components and not to the overall system as cross-functional values are met by means of more specific values (e. g. transportation can be achieved by bike, public transport, car and so on which have their individual characteristics and can even be combined). Furthermore, in objective 4 (see section 1.2) we have defined that the influence of technological decisions on future decisions should be minimized which includes that decisions on how to deliver one value should interfere with the decision on how to deliver another value as little as possible. Hence, technological decisions for a value should primarily done on the micro architecture level.

Nevertheless, it cannot be guaranteed that a micro architecture is completely independent of the macro-architecture, so the macro-architecture (we further elaborate how a macro-architecture for our system can be designed in section 7.1) has to be considered but not designed in this stage. From this, we derive our **fifth requirement**:

**Requirement 6.1.5.** In this stage, on the technical level, the value should be reflected primarily in the micro architecture and take into account the macro-architecture.

First technical prototypes, to evaluate the technical feasibility are also part of this step. To achieve this, standard procedures in software development for the implementation of technical prototypes or proof of concepts can be used. The exact technical design, however, should be left to the team, as usual in agile software development.

In addition to the technical level, the look and feel level is to be worked on in parallel (see Figure 6.2). The look and feel level is described by Houde and Hill [HH97] as the concrete experience of an artifact, what it would be like to look at and interact with, in short how the interface to the user is designed (interaction and perception wise (e.g. visually)). For prototype creation on this level, the standard processes in design, such as those described by Mayhew [May12], can be used to elaborate them according to the value idea.

For the final step in this stage, all three levels have to be integrated. It has to be ensured that the technical level and the look and feel level align well with the value level as well with each other. Hence, in this step the value designer, designer (look and feel), and software developer (technical) evaluate together the results of the previous steps, discuss the alignment, and make adjustments where necessary. If insights are made in this step that require fundamental changes, it is possible to go back to the respective previous step or stage. Hence, our **sixth requirement** is:

**Requirement 6.1.6.** The final results of the integration shall be documented within the structure resulting from the second requirement.

In summary, this stage consists of four steps, of which two steps (*Look & Feel* and *Technical*) can be performed using existing standard methods, which is why we will not go into them in detail in the following. Especially for the step *Transform* we have developed a corresponding framework, which we present in the next section. This framework also allows the integration of the final results of the *Prepare Integration* step into a software requirements hierarchy and thus also serves to fulfill the last requirement.

## 6.2 Design Thinking Requirements Framework (DTRF)

In this section, we present our framework to transform design thinking results into agile software requirements, which was developed in conjunction with a master thesis [Kle19]. Its focus is not on the general process integration of design thinking and agile software development, but on the specification of requirements for agile software development based on the outputs of the design thinking activities. Therefore, it mainly consists of templates similar to the atomic requirements shell (as example see Figure 6.1 on page 98) by Robertson and Robertson [RR06] to capture and structure information. The templates we have developed are called *Capture Cards*. They can either be filled out and arranged pen and paper based, or digitally. For the digital version we created a page template in the wiki system Confluence, in which the *Capture Cards* filled out pen and paper based will be entered later as well. This later step is the last step of a three step process we have devised to transform the information from design thinking to agile software requirements (see Figure 6.3). In the following we will explain this process, before we go into detail about *Capture Cards* and last but not least compare our approach to other existing approaches.

### 6.2.1 Transformation Process

The basic idea behind the Transformation Process is a step-by-step formalization of the design thinking results. Starting point is the collection of information gathered with design thinking. As design thinking is intended to be done iteratively, being open-minded, incorporating diverging / converging thinking, learning with the help of prototypes, and exploring problem and solution space, the resulting information is not yet highly structured and often associated with a certain artifact created during design thinking. If we try to transform this large amount of unstructured information into a highly formalized structure in one step, this could lead to an overload. We have therefore opted for a three-step process (see Figure 6.3).

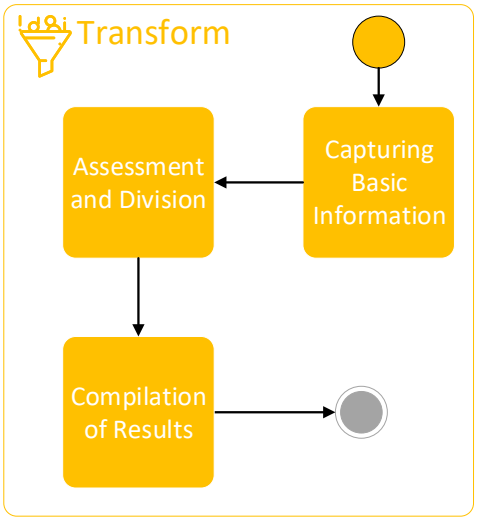



**Fig. 6.3.:** Process to Transform Design Thinking Results into Agile Software Requirements.

The first step is *Capturing Basic Information*, which is optimized to capture information in small unrelated chunks. Like in qualitative content analysis (cf. [May00; GL09]), the first step is to synthesize and extract the information into small chunks from the design thinking results. In the next step, the information is categorized with the help of a broad set of categories. This is where the pen and paper based *Capture Cards* come into effect. We are using these to capture the small information chunks and assign them already a broad category (see Table 6.1). Furthermore, dynamic categories in the form of initial objectives or alternatives are created, in addition to the predefined categories from the *Capture Cards*. These need not yet be the final objectives, but all *Capture Cards* should already refer to such an objective. Expected output of this step are multiple *Capture Cards* for each category.

In the next step *Assessment and Division*, the objectives are assessed regarding their complexity to implement and if necessary split into new objectives. This is a three-tier step, where in the first tier it is discussed with all participating team members how the objectives

are intended to be reached, how much resources are needed, and how complete the objective already is, which is actually the same as assigning Story Points in Agile Development (see [Mey14]).

In the second tier, this complexity assessment is used to further split or merge objectives to get actionable objectives. The goal is, that for a sub-objective, manageable User Stories can be created that are not too big to be implemented in one sprint. Furthermore, the lowest sub-objective shouldn't be as big that more than three months are needed to be implemented as it contradicts the idea of bounded contexts and the flexibility to evolve the system by re-implementing a service (cf. [Eva04; DGH08])

The third tiers goal is to prioritize the objectives according to their value. As Boehm [Boe06b] states, traditional software development treats requirements value-neutral. This is a problem if not everything can be implemented with the given resources or features that deliver less value are implemented before other features. For this reason, the objectives shall be prioritized according to the value (that can be found in the Values and Needs *Capture Cards*). A simple prioritization scale that can be used is e.g. high, medium and low.

The last step in our transformation process is to digitalize and aggregate the results. For that, we are using the mentioned digital *Capture Cards*. Each Objective will get a separate wiki page where the other categories related the Objective are collected. Furthermore Hyperlinks to additional material and for the hierarchy are introduced and maintained. From these digital *Capture Cards* Epics with the corresponding User Stories can be created, which link to the Digital Capture Card itself for additional information.

### 6.2.2 Capture Cards

There are a total of seven *Capture Cards*, each presenting one category (see Table 6.1). The categories are *Background*, *Objective*, *Needs*, *Values*, *Material*, *Hierarchy*, and *Misc*. *Background* and *Material* serve above all to make it traceable (cf. Requirement 6.1.3) where certain requirements come from, whereas *Objective* describes concisely the desired functionality derived from the *Needs* and *Values* (cf. Requirement 6.1.2 and Requirement 6.1.1). The purpose of *Hierarchy* is to enable us to have alternatives as well as refining objectives to more actionable sub-objectives (cf. Requirement 6.1.4). *Misc* is used for additional specific information that cannot generally be mapped into the other categories. In general, the pen and paper based *Capture Cards* maintain their link to the objective by writing out it explicitly at the end of the card, whereas the digital *Capture Cards* for one objective are collected on a single wiki page.

| Category Name | Category Summary | Category Fields |
|---|---|---|
| Background | Context information about conducted design thinking activity. | date, topic, participants, motivation |
| Objective | Functionality desired by a user described as a concise goal. | (first priority estimation) |
| Needs | Information about user needs derived from design thinking results. | Point of Views |
| Values | Relative worth, utility, or importance of the functionality defined in *Objective*. | User Values, Business Values |
| Material | Recordings of more tangible results of design thinking as well as *Technical* and *Look-and-Feel*. | None |
| Hierarchy | Relations between objectives described by parent-child-relation or alternative objectives. | Alternative Objectives, Super-Objective, Sub-Objectives |
| Misc | Any additional information that does not fit into the above categories. | Additional |

**Tab. 6.1.:** Category Overview.

In the following, we will further explain the different categories and give examples for filled out *Capture Cards*.

### Backgroud

| **Background** | |
|---|---|
| **Date** | 08.10.2018 – 19.05.2019 |
| **Topic** | Event portal |
| **Participants** | Björn, Simon, PG W18/19 |
| **Motivation** | Making cultural events in a decentrally organized region visible to interested parties. |
| "A website that bundles the cultural events of the OWL region and makes them accessible to interested parties while maintaining the local identity." | |

**Tab. 6.2.:** Capture Card: Background

The idea behind the Background capture card is to give context information about the circumstances the underlying information have been gathered, especially during the design thinking activities. Hence, the most critical part is the motivation for the activities, which in case for the design thinking activities is the underlying design challenge. Furthermore, the topic is a further restriction for what the activities have been conducted. Date and Participants

give an overview of the actuality of the data and the persons involved. Especially the involved persons are crucial to easily understand the reliability of background as well as whom to ask for further clarification.

### Objective

| **Objective** |
| --- |
| "A website that bundles the cultural events of the OWL region and makes them accessible to interested parties while maintaining the local identity." |

**Tab. 6.3.:** Capture Card: Objective

Objective is the essential part connecting all *Capture Cards* and the bridge between values and functionality. It describes the functionality desired by users concisely and is the first time to put functionality instead of values in the foreground. By defining the functionality, it further refines how a value shall be delivered and clarifies the goal for the developers even more. The granularity highly depends on the results from the design challenge and can be as broad as in this example (see Table 6.3) or be so specific that it cannot be further broken down into sub-objectives. The latter case is rather a result of further refinements in the Technical step of this stage or of the Optimization stage. Results of the design thinking from the previous level should be at a level where this specific granularity is not possible.

### Needs

The needs are actually the collected *Point of View* (POV)s from the previous stage (see section 5.2) that are important for an objective.

| Needs |
| --- |
| **Point of Views**<br>• A **new resident** with yet a small social circle in this region **needs** to know what she can do **but** the main information source is mostly the social circle and information regarding cultural events in OWL on the internet is quite scattered.«Ref#1»<br>• An **international student** who is new to this region **needs** an onboarding to this region and its events **because** she doesn't know what OWL stands for, how this region is connected and what places are still accessible by public transit from her location.«Ref#2»<br>• A **visitor** who is only for a short time in the region **needs** to find out what she can do spontaneously on an evening **because** her meeting was shorter and she doesn't want to spend the full evening isolated in the hotel room.«Ref#3» |
| "A website that bundles the cultural events of the OWL region and makes them accessible to interested parties while maintaining the local identity." |

**Tab. 6.4.:** Capture Card: Needs

### Values

We have defined value as 'relative worth, utility, or importance'. From the Needs we can derive Values for a user that give a more convenient presentation of what shall be delivered than the full description of user, needs, and additional insights as it is the case in Needs. Additionally, several needs from different user categories may be subsumed under one value if they overlap. Besides user value, we present here as well the business value, but due to transparency of this potentially contradicting values, we separate them. Nevertheless, the business value is essential as it introduces the viability dimension of a proposed user value.

| Values |
| --- |
| **User values**<br>• Finding events in OWL without the need of a social circle.<br>  – Building a social circle based on common interests in certain events.<br>  – Challenge own social circle to new things.<br>• Finding events in OWL without knowledge about the geographical and transit structure.<br>• Spontaneously find events that are reachable within an acceptable travel time.<br>**Business values**<br>• Make cultural events in OWL visible.«Ref#5»<br>• To perceive the cultural region OWL more strongly as a whole.«Ref#5»<br>• Reach more target groups.«Ref#5»<br>• Overcoming regional borders.«Ref#5»<br>• Strengthen the cultural profile and identity of the region.«Ref#5» |
| "A website that bundles the cultural events of the OWL region and makes them accessible to interested parties while maintaining the local identity." |

**Tab. 6.5.:** Capture Card: Values

### Material

| Material |
| --- |
| • "Machbarkeitsstudie" 2018<br>• Interview Results, Project Group OWL.Cultur-Platform<br>• Website: The 7 Factors that Influence User Experience<br>• Website: Usability: A part of the User Experience |
| "A website that bundles the cultural events of the OWL region and makes them accessible to interested parties while maintaining the local identity." |

**Tab. 6.6.:** Capture Card: Material

Material is the recordings of more tangible results of design thinking (e.g. photos of demonstrators, interview results) as well as *Technical* (e.g. technical prototypes) and *Look-and-Feel* (e.g. visual mock-ups). It is to support the understanding of reasons behind design decisions as well as to transport design decisions better. In our example (see Table 6.6), we are referencing a more vague objective (see Table 6.3), which is why we haven't used

visual mock-ups or prototype screenshots in the Material but referenced the interview results, feasibility study and some more general guidelines. In sub-objectives, like *"In the 75%-100% initial visible content to the user, transport the core idea of the portal as well as the core functionality with a search field and only present events after scrolling down."* for the Landing Page for such a website, more tangible and functionality related material, especially from the *Technical* and *Look-and-Feel* Level would be included.

### Hierarchy

This capture card is to maintain the relations between different objectives and alternatives. In our example (see Table 6.3) the objective is already the super-objective, which is why only sub-objectives are set (see Table 6.7). For the corresponding sub-objective, this objective (Table 6.3) would be set as super-objective. In each case (super-objective and sub-objective) only the first degree will be considered to reduce redundancy as well as cluttering. If there is an alternative objective, like for the landing page, all alternatives to this objective would be referenced under Alternatives.

| **Hierarchy** |
| --- |
| **Alternatives**<br>**Super-Objective**<br>**Sub-Objectives**<br>• LandingPageEvents: "In the 75%-100% initial visible content to the user, transport the core idea of the portal as well as the core functionality with a search field and only present events after scrolling down."<br>• LandingPageOnboarding: "In the 75%-100% initial visible content to the user, transport the core idea of the portal as well as the core functionality with a search field and only present description of further features after scrolling down."<br>• Profile: "Let the user create an account with basic information to save interesting events."<br>• Event Details: "Present the details for an event that are interesting for a user."<br>• Event Overview: "Present a set of events and give the user the option to further refine their search based on filters and browse through the results." |
| "A website that bundles the cultural events of the OWL region and makes them accessible to interested parties while maintaining the local identity." |

**Tab. 6.7.:** Capture Card: Hierarchy

#### Misc

Misc is to capture any additional information that does not fit into the other categories. This can include Comments about the given information as well as Questions or additional remarks. Possible additional requirements (e.g. on hard- or software) or additional constraints can be documented here as well.

| **Misc** |
| --- |
| **Additional Information**<br>• **Comment**: Although the OWL.Kultur-Portal is intended to help overcome regional borders, it is important to maintain local identity, which is why it should refer back to local platforms whenever possible (e.g. hyperlinks in the details).<br>• **Question**: Are we missing the "normal" user as user group? |
| "A website that bundles the cultural events of the OWL region and makes them accessible to interested parties while maintaining the local identity." |

**Tab. 6.8.:** Capture Card: Misc

### 6.2.3 Related Work

If we take our requirements for this stage, the three main parts for this step *Transform* with our design thinking requirements transformation framework as instantiation are requirements definition / documentation, traceability, and hierarchy. According to this we explain in the following how our solution differs from others beginning with requirements definition.

As already introduced in the introduction, our *Capture Cards* are inspired by the atomic requirements shell by Robertson and Robertson [RR06]. The requirements shell is intended to map low-level requirements, with its rationale, and a dedicated fit criterion that is measurable. It also integrates elements for tracing requirements such as source respectively originator and supporting materials. However, source is only intended to briefly and concisely name the person(s) who wrote/stated this requirement. It is not intended to give additional contextual information about the emergence of these requirements as it is done with background in our case. Furthermore, although you can link dependencies within the requirements shell, it is not intended to create a requirements hierarchy. The dependencies are more related to blocking or restricting requirements. Lastly, the requirements shell does not enforce you to explicitly state anything about the needs and values.

The requirements shell is a good example of a requirements template originated in a time waterfall models were predominent. De-facto standard for requirements definition in the

world of agile software development are user stories, which describe 'scenarios that represent user interactions with a system' (cf. [Mey14, p. 10]). Meyer [Mey14, pp. 119-121] describe user stories as the description of a fine-grain functionality of the system as seen by its users which usually follows a standard style consisting of the triple *category of user*, *goal*, *benefit*. Within this way of describing units of system functionality from the perspective of users lies its strength and weaknesses in agile software development. First and foremost, this type of requirements definition has the advantage of getting the team to think the system more from the user's point of view and less from the point of view on how to improve the existing code. However, this also brings disadvantages such as a more difficult estimation of the implementation effort, since the concrete implementation is not dealt with and thus it is not immediately obvious what effects a user story has on the technical foundation. In agile development, however, this is partly desired, since this decision shall be left to the team as a body of experts. The difference to our approach is that user stories are already strongly linked to the functionalities of the system to be implemented. *Values* and *Needs* on the other side describe goals from a more system independent level and are therefore less threatened by change. However, these two can be used wonderfully as a basis to formulate appropriate user stories in the further course. Furthermore, user stories are missing links to supporting material or documentation of how and why certain user stories arose. This makes it extremely difficult to change them and the system, as there is no traceability for decisions, as it is the case with our framework. User Stories in their standard form of *category of user*, *goal*, and *benefit* also do not support the representation of a hierarchy or alternative system designs.

Requirements traceability refers to the 'ability to describe and follow the life of a requirement, in both a forwards and backwards direction' [GF94]. In addition, a distinction is made between pre-requirements specification traceability and post-requirements specification traceability, where requirements specification is understood as the step of creating user stories or requirements with the requirements shell in terms of concrete system functionality. In this sense our framework falls under the pre-requirements specification traceability, whereas most research is concerned with post-requirements specification traceability (cf. [RM17; GF94]). In general, however, we do not use an elaborate metamodel or tools to establish traceability, but use simple text-based references or hyperlinks. According to Pohl [Poh07, pp.515], this is one of the most basic forms of representation in requirements traceability. We decided for this to have the flexibility to include all kind of different artifacts.

As for the hierarchy, software development is usually done under the premise to create one solution design and not different alternatives (cf. point based engineering, Ward et al. [War+95], Denning et al. [DGH08], and section 1.1.3). Furthermore, especially in lean design or agile software development, great attention is paid to reduce 'waste'. According to Meyer [Mey14], in software this is anything not delivered to the customer. Further

hierarchies and alternatives can be seen as 'waste' in this definition as they are not shipped to the customer respective getting the default user experience. Nevertheless, there are approaches in software development to feature such hierarchies and even mapping different alternatives. These are mainly goal models, with their most prominent representatives AND/OR trees, i*, and KAOS (cf. [Poh07, p.103]), and feature models (cf. [Bat05]). They are all dedicated modelling languages with their own syntax and besides i* are built as tree structures which usually include special annotations for the links to mark them as e. g. *and*, *or*, or *xor* links. Traceability as we needed for our case is not included in these models as well as additional descriptions like we have in *Background*, *Values*, and *Needs*. Moreover, we have so far relied primarily on text and hyperlink-based documentation in order to reduce media disruption and maintain flexibility. Using such models with extra tools would mean additional media disruption and restriction in linking to supporting materials. This is why we explicitly do not use these as main component to model our requirements hierarchy / alternatives but stick to plain text references and hyperlinks without additional checks. Nevertheless, it could be interesting to include these models as additional documentation or to extend hyperlinks (in the corresponding system) to represent these models in the future without additional media disruption and losing flexibility.

### 6.2.4 Feasibility Study

An initial feasibility study has been conducted to assess the suitability of this framework to transform design thinking results into software requirements. For that, our main goal of this study was to determine how software developers use and assess this framework, which is why we had a more detailed look in how the framework was understood, how well pre-requirements specification traceability was supported, how the handling of alternatives was implemented, and to what extent the framework has helped to clarify *Needs* and *Values* as motivators for the definition of requirements. To do this, we are using an a/b setup with observation, questionnaires, and analysis of the created artifacts as evaluation instruments. For the a/b setup we rely for one half of the participants on our developed framework and let the other half create agile software requirements (epics and user stories) directly with the help of the 'product requirements document' template embedded in confluence[1].

In order to use this framework, participants must first go through design thinking to have appropriate artifacts that they can transform into agile software requirements. We therefore embedded this study in our case study OWL.Culture-Platform (see section 8.3) directly after the conduction of design thinking with non-software. Accordingly, the participants are students (see section 8.3.1 for further information) who already received an introduction to

[1] `https://www.atlassian.com/software/confluence/templates/product-requirements-document`

SCRUM (including the concept of epics and user stories) as one agile software development method (see section 8.3.2,lego4scrum Workshop). Furthermore, they have already been working in six teams of three people prior to this study. We continue to use these teams and let three teams use our developed framework, whereas the other three teams shall directly create agile software requirements.

For the conduction of the feasibility study, the teams working with the framework get an introduction into the framework. The next step is for all teams to fill out a questionnaire to determine their current knowledge and experience regarding agile software requirements. After that, the teams working without the framework start directly to create epics and user stories without getting further hints or being influenced by the investigator. The teams using the framework are going through the three phases (*Capturing Basic Information*, *Revision and Refinement*, and *Compilation of results*) of the framework and have as last step the creation of epics and user stories based on the digitized *Capture Cards* and the 'product requirements document' template in confluence. After that, they have to fill out the second part of the questionnaire which captures the participants perception of the process irregardless of the framework. In case of the teams using the framework, follow up questions regarding the framework itself, the artifacts, and the process regarding its usability are asked. In the other case, the participants have to describe the approach they used and what major challenges they faced. As a last step, the single artifacts are analysed.

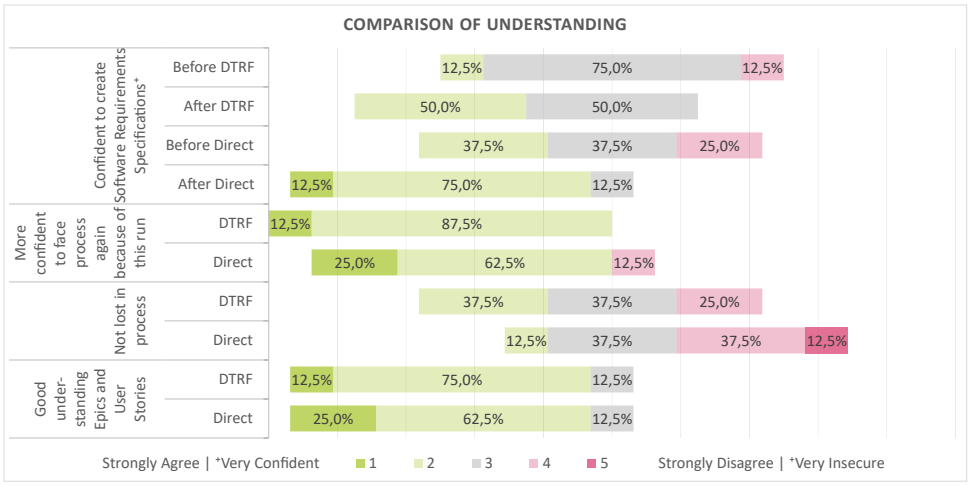



**Fig. 6.4.:** Design Thinking Requirements Framework (DTRF) Evaluation Results Questionnaire: Understanding. {Before, After} DTRF=Questionnaire for DTRF teams {before, after} they used DTRF and created agile software requirements. {Before, After} Direct = Questionnaire for teams {before, after} they directly created agile software requirements.

The first results we want to discuss from our evaluation is how DTRF changed the participants understanding of agile requirements specification. In Figure 6.4 we present the main results on this topic. In general, creating software requirements specifications (SRS) during the evaluation made the participants more confident to create them again. However,

the Direct participants stated more often that they are confident with creating SRS than DTRF participants. We believe that this result can be explained by the fact that the Direct participants immediately and significantly longer dealt with SRS than the DTRF participants in our evaluation. Although, 75% of all the participants stated that they are familiar with Epics and User Stories, our observations during the evaluation suggests otherwise, which is why there could be a training effect. In the questionnaire after the creation, the participants of both DTRF and Direct teams (strongly) agreed that they have a good understanding of these terms. Interestingly, the participants using DTRF stated more likely that they are more confident to face the process again and to always knew what to do next during the whole requirements specification process.

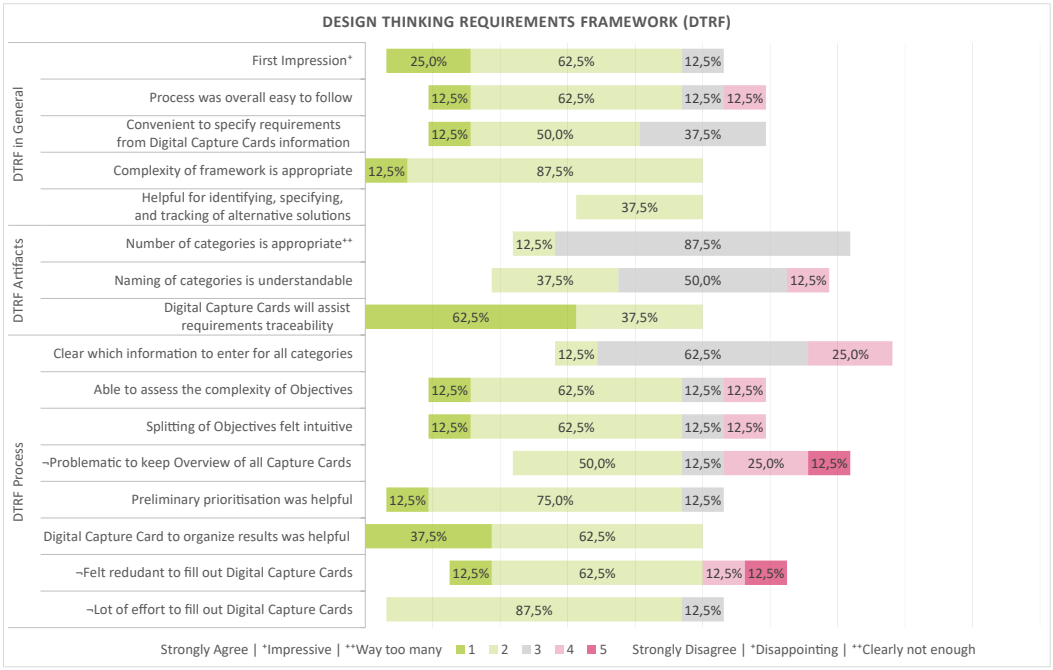



**Fig. 6.5.:** Design Thinking Requirements Framework (DTRF) Evaluation Results Questionnaire: Framework.

The next results are regarding the perceived usability of DTRF, which are presented in Figure 6.5. Of the eight participants who used DTRF, seven expressed that they had a positive or impressive first impression of DTRF, while one rated it neutrally. Overall the vast majority (>62,5%) stated that the process was overall easy to follow, the complexity of the framework was appropriate, and that it was convenient to specify requirements from Digital Capture Card Information. From the three teams using DTRF in this evaluation, only one team found alternatives, but they stated that they agree that DTRF is 'helpful for identifying, specifying, and tracking of alternative solutions'. As only one team could answer this question, it sums up to 37,5% to highlight that not all participants answered this question contrary to the other questions.

For the DTRF artifacts, all participants of the DTRF teams stated that they (strongly) agree that the digital *Capture Cards* will assist requirements traceability. Also the participants declared except one that the number of categories is just right. The understandable naming of the categories was answered affirmatively by only 37.5% of the participants, while 50% were neutral. From the comments to this question, it can be assumed that some participants had problems to distinguish why Values and Needs are separated and the difference between Background, Material, and Misc. Our observations were that the participants expressed significant issues at first regarding the understanding of certain categories. This could possibly be due to unclear definitions or the difficulty for participants to convert their mindset from a purely functional requirements specification to a value-based one.

The latter is supported by the fact that even if Values and Needs were actually be worked on with DT in the previous stage of our approach, our observations show that the participants found it difficult to identify them directly and spent a correspondingly large amount of time identifying them in the evaluation. Our analysis of the DTRF artifacts showed that all three teams have been using user teams, but only as a category without further describing it. In addition, Needs and Values are described from the point of view of a user without using functionalities, but already removed additional information about the user and insights like they did previously in the interview results (see section 8.3.4.2). Furthermore, the evaluation of the main case study on the OWL.Culture-Platform shows that even in the previous stages, the participants already had difficulties with the formulation of POVs from the user's perspective (cf. section 8.3.4.2 and section 8.3.5).

Regarding the DTRF process, a majority of more than 62,5% expressed a neutral opinion whether it was clear which information to enter for all categories (see also previous paragraph). 75% of the participants (strongly) agree that they were able to assess the complexity of the objectives and splitting them felt intuitive. One participant commented that it was not clear to what degree the splitting should be done and another that she was unsure about including technical aspects. 37,5% found it problematic to keep overview of all *Capture Cards* in contrast to 50% not finding it difficult. This could indicate that the physical usage of *Capture Cards* is not beneficial, especially as all teams have used only three to eight objectives (including the sub-objectives). Besides one participant, all participants agree that the preliminary prioritization was helpful and all participants agreed on that the usage of digital *Capture Cards* to organize results was helpful. Two participants felt it redundant to fill out the digital *Capture Cards* after filling out the *Capture Cards* (paper based) and only one participant felt that it takes a lot of effort to fill out the digital *Capture Cards*.

Overall, the participants were extremely positive about the use of DTRF and confirmed its usefulness. To what extent the usefulness is actually given, we checked by analyzing the

artifacts of the DTRF teams in comparison with the Direct teams and comparing them with our observation notes.

The Direct teams, that directly created epics and user stories, were functionality focussed with the mention of the word user without further defining it. 8 of 16 epics they created were described without a single reference (neither to the previous design thinking results nor additional material like scientific studies). 3 epics referenced interview results generally and 5 have included references into their user stories. From our observations, these teams revealed issues to identify the actual stakeholders. Furthermore, they had problems to asses the final complexity of objectives resulting in too small objectives for proper User Stories. Only one of the teams actually linked material supporting their decisions.

Regarding the DTRF teams, only one used users groups, but only as a category without further describing it. As already mentioned, they all described Needs and Values from a user's point of view without using functionalities. In addition, they all linked material supporting their decisions, but without mentioning the exact position. Based on the digital *Capture Cards*, they created Epics and User Stories, whereas the Objective was used in all cases to name the epic. Due to time constraints during the evaluation, not for all objectives User Stories have been created. All User Stories in the Epics are linked to the corresponding sub-objectives, whereas the link text was the sub-objective. Goals and Background of the Epics have been described without using functionalities.

Comparing the artifacts created by DTRF teams and Direct teams, the DTRF teams were actually less focussed on functionality and created a hierarchy based on the Values and Needs. The Direct teams were more focussed on functionalities, which was reflected as well in the way they created their hierarchies with Epics and User Stories directly. In this sense, DTRF showed that it can help that the value is explicitly defined and is on top of a requirements hierarchy with further refinements the further down you go. Although the defined Values and Needs by DTRF teams do not suffice the wished quality of them, it is still significantly better than the quality of the Direct teams. It also helped to better split objectives into actionable sub-objectives in comparison to directly creating the Epics and User Stories with functionalities in mind. Also, the traceability is more given with the artifacts created by the DTRF teams compared to the Direct teams, although the used references could be more specific. Overall, the evaluation shows evidence that DTRF works and can meet the requirements placed on it. A big challenge, however, seems to be the change of the mindset to value instead of function oriented thinking.

## 6.3 Summary and Discussion

In this chapter we have presented the third stage of our approach ICeDD, whose primary goal is the transformation and refinement of Design Thinking results from the previous stage to be able to continue it in software development. For that, we have introduced a process to achieve this in section 6.1. This process consists of the steps Transform, Technical, Look-and-Feel, and Prepare Integration, whereas the last three steps are the refinement and can be done with already existing methods which is why they are only briefly introduced. For Transform, on the other hand, it was an open question on how to ensure the knowledge flow from Design Thinking (cf. [LMW11]), especially without overburdening the developers with information. Furthermore, agile software specifications / documentation usually do not consider alternative solution designs but stick to only one solution at a time, but the alternative solution designs are needed for the next stage. Therefore, we proposed a solution for the Transform step, which we present in section 6.2.

We call this solution for the Transform step *Design Thinking Requirements Framework (DTRF)* and it consists of a process (section 6.2.1) in which we incrementally prepare the information from Design Thinking for software development. One key element in this process are the so called *Capture Cards* (section 6.2.2), which are templates to structure and relate the Design Thinking results from the previous stage. This proposed solution was then evaluated regarding its usability and feasibility in section 6.2.4 Result of this feasibility study was that in our case the participants were very positive about the use of *DTRF* and confirmed its usefulness. An analysis of the artifacts revealed that *DTRF* helped the participants to stay more focussed on the Values and Needs and less on functionality, as well as to create a hierarchy and split objectives, and to make it more traceable. But it also became clear that the change of the mindset to value instead of function oriented thinking is still a big challenge as the quality was not yet as good as wished.

In relation to our defined fitness functions (see section 1.2), we fulfill *Alternatives* to the fullest with the **fourth requirement**. *Operating Alternatives* (Component based Deployment) and *Consequences of Technological Decisions* (Bounded Context) are supported with the **fifth requirement**, but most of it will be covered in the next stage (see section 7.1) as this stage is not about implementing software but preparing it. As this stage is about transforming results and preparing the implementation with the current knowledge (cf. section 3.1, we are not focused in this approach on unique and novel technical or look and feel solutions), the fitness function *Focus on Novelty* is not of concern for this stage. For the last fitness function *Learning Cycle*, the condition "*A learning cycle is intended, explicitly defined and refers to several alternatives at a time.*" is fulfilled as we have defined it with the *Prepare Integration* step and our fourth requirement to represented at least two solution alternatives in software requirements. How this is reflected in our FF is presented in Figure 6.6

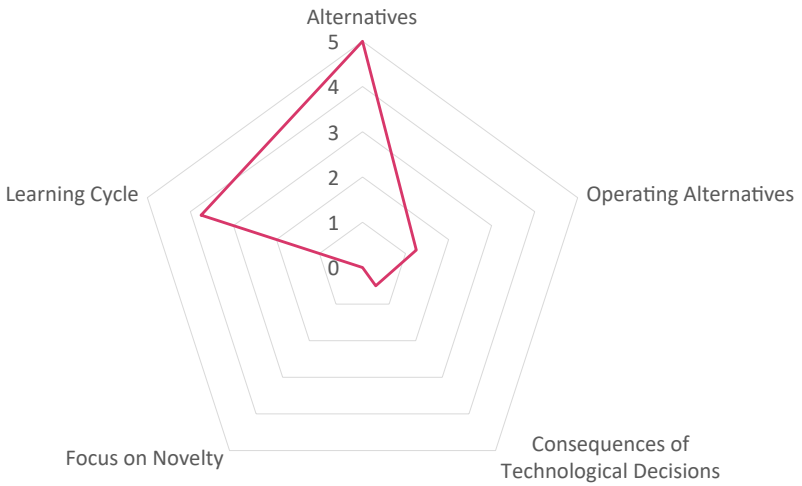



**Fig. 6.6.:** Radar Chart for Stage 3 regarding our *Fitness Function* (FF).

Although, we have shown that in our context our process was useful and that it helped to keep findings regarding Value, further research is needed. The main objective of the evaluation of *DTRF* was to show that it is feasible, but not yet how well suited it is. For the latter, further evaluations with more participants and stricter comparison conditions must be carried out in order to obtain reliable and generalizable results. It is also not yet clear whether the granularity we have used in our process and specification/documentation is the most optimal. We have only taken very rudimentary information from Design Thinking into the specification/documentation and references to the Design Thinking artifacts are relatively imprecise. If, for example, qualitative content analysis was used for the Design Thinking artifacts, accurate references could be made that could be traced back to the source. However, this is a significant additional effort that must pay off in the short, medium or long term.

# 7 ICeDD Stage (4): Execute Design Thinking with Software

In the first stage (chapter 4) we identified a design challenge that serves as a basis for Design Thinking with non-software in the next stage (chapter 5). We started with non-software as a medium as it is usually more efficient to build prototypes with it as it would be with software (cf. Stecklein et al. [Ste+04]). Nevertheless, it is essential to try out different alternatives also with software in real-world contexts (cf. section 1.1 or section 2.1), which is done in this stage as a continuation of Design Thinking. To be able to do so, we have refined and transformed the Design Thinking results with non-software in the third stage (chapter 6) into an agile software specification / documentation. In section 7.1, we are giving requirements and an overview of this stage which is based on the 4Ps. Accordingly, in section 7.2 we are presenting the usage of each P in our approach. The first P process (section 7.2.1) is adapting an experimentation process for qualitative experiments to be done within software development. In section 7.2.2, we are discussing the necessary product features, especially the needed macro-architecture, to support experimentation in this stage. Finally, in section 7.2.3 the prerequisites of the people and in section 7.2.4 the shaping as a project are discussed. An important point for the feasibility of this stage is viability. To substantiate this, we have developed tools which we present in section 7.3. Finally, we are summarizing this chapter and discuss the results in section 7.4.

## 7.1 Requirements & Overview

The basic idea of this stage is to continue Design Thinking by trying out different alternatives in a productive environment to learn from that. The main reason for that is the assumption that only if the software-based solutions are used in the real world can it be guaranteed and evaluated whether they fulfil the intention of the user and thus keep their value (relative worth, utility, importance) proposition (see section 1.1). This is supported by research in the area of innovation (cf. section 2.1), whose applicability to software we have investigated in a first application case study (see section 8.2).

Of our objectives for the approach, Objective 2 (Support multiple solutions simultaneously), Objective 3 (Simultaneous operation of several software solutions), Objective 4 (Independence of technological decisions) and Objective 6 (Explicitly defined learning cycle) are particularly interesting at this stage as they are directly related to the goal of this stage of having different software alternatives to learn from. Objective 5 (Novelty) is not in the focus of this stage, as this stage is only about validating and learning from the software alternatives regarding value. This is why, among the fitness functions as operationalised objectives (see section 1.2), the functions *Alternatives*, *Operating Alternatives*, *Consequences of Technological Decisions*, and *Learning Cycle* with their inherent requirements are particularly relevant.

Consequently, in this section we describe how we can try out different software alternatives in order to learn from them to what extent they provide value and which characteristics this may be due to. This includes that the software development has to be aligned accordingly to enable this. Which is why it is necessary to look at all parts of the software development. To do so, we are using the basic structure in software development introduced by Jacobson, Booch, and Rumbach [JBR99] as *The Four Ps: People, Project, Product, and Process*:

"The end result of a software *project* is a *product* that is shaped by many different types of *people* as it is developed. Guiding the efforts of the *people* involved in the project is a software development *process*, a template that explains the steps needed to complete the *project*. Typically, the *process* is automated by a *tool* or set of *tools*."

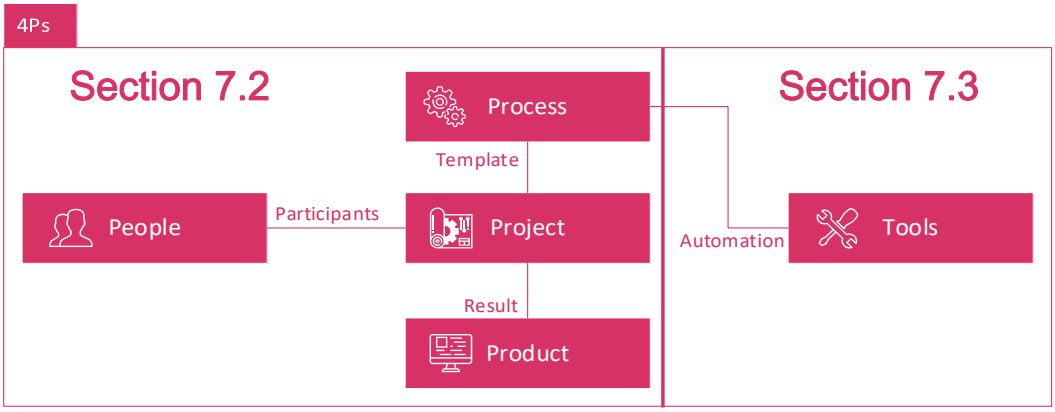



**Fig. 7.1.:** The 4P's People, Project, Product, Process, and Tools from the Unified Software Development Process [JBR99]. Own representation.

Accordingly, the following sections are structured based on *The Four P's* to discuss the implications of experimenting with different software alternatives in detail in them (cf. Figure 7.1). Most crucial in the sense of effectivity are *People*, *Project*, *Product*, and *Process* as they define how we will develop the software. In contrast, *Tools* is most crucial in the

sense of efficiency and with that also the viability. This is why we bundled *People*, *Project*, *Product*, and *Process* in section 7.2 and *Tools* in section 7.3.

Furthermore, this learning with the help of comparing alternatives in usage is a well-known process in various fields including computer science, but not necessarily that explicit in software development. It is called experimentation, for which we use the definition of Hussy, Schreier, and Echterhoff [HSE13, pp.120–145] to explain it in more detail. This is because, unlike, for example, Thomke [Tho03], Stol and Fitzgerald [SF18], or Kohavi et al. [Koh+08], it is not purely narrative, but defines concrete characteristics that distinguish experiments.

An experiment is the systematic observation of dependent variables while manipulating independent variables, whereby there must be a temporal order between independent and dependent variables and at least two variations of one independent variable are needed. In our case, the dependent variable is the value we want to deliver, which may include an corresponding operationalisation. The independent variable is represented by the variations in the different software alternatives. This separation of independent and dependent variables including the temporal order is the foundation for experiments.

Furthermore, there is not one single type of experiment, but different types. To distinguish them, two questions are asked. Are the participants randomly assigned? And is it conducted in a laboratory setting in which confounding variables are tried to be controlled / eliminated as good as possible? If it is conducted in a laboratory setting and the participants are randomly assigned, it is called a (laboratory) experiment or controlled experiment. If the participants are not randomly assigned, it is called a quasi-experiment. On the other hand, if the experiment is done in the field exercising less control over confounding variables, it is called a field experiment when the participants are randomly assigned. Otherwise it is a field study.

With this experiment classification we can further pin down what we need in our case. In this stage, it is the first time that we use the value propositions implemented in software in the real world and not under laboratory conditions. The reason we are doing this is, that all effects can only come to light in real usage, as they are not consciously or unconsciously masked by artificial situations. So we are still trying to understand rather than proving if, how, and why value is delivered. Therefore, experiments at this stage cannot and should not yet take place under laboratory conditions, as not all confounding variables can be known. Hence, the experiments are done in the field, which only makes field experiments and field studies appropriate. Accordingly, our **first requirement** for this stage is in addition to those resulting from the objectives and fitness functions mentioned above:

**Requirement 7.1.1.** Experiments done in this stage have to be conducted as field study or field experiment.

Furthermore, as we are still trying to understand rather than prove, qualitative methods are more important to be used in the experiments than quantitative ones (cf. section 1.3 or [Hel11; MR15]). However, experiments are usually regarded as a quantitative method (see e.g. [HSE13]), as they are often linked to laboratory experiments that try to test a hypothesis by operationalising the variables and controlling the experimental environment. Furthermore, because of that, the quality criteria typical for quantitative methods like repeatability are applied to quantitative experiments (see section 1.3). This would actually contradict our goal. But experiments do not necessarily have to be mainly quantitative, but can also be qualitative in which case they are called qualitative experiments. Usually, qualitative experiments are conducted ”naturally” / authentic and do not have repeatability as condition (cf. [Bur10]).

Kleining [Kle91] defines the qualitative experiment as "the intervention, according to scientific rules, in a [...] given situation to investigate its structure. It is the explorative, heuristic form of the experiment." (Translation by the author). Furthermore, the basic rules for qualitative research (cf. section 1.3) apply as well for qualitative experiments, which includes *openness of the researcher*, *openness of the research object*, *maximum structural variation of the research object*, and *analysis for similarities* (cf. [Bur10; Kle91]). This matches actually our goals for experiments in this stage, which is why our **second requirement** is:

**Requirement 7.1.2.** Experiments in this stage have to be conducted as qualitative experiments.

## 7.2 People, Project, Product, and Process

The main issue with current software development processes regarding experimentation or learning from different software alternatives is that they are optimized solely on the implementation (cf. paragraph 3.1). But in order to learn from software alternatives by e.g. experimentation, it has to be defined what you want to learn from them, what properties have to be implemented to be able to do so, how you will generate the data, and how you will interpret them (cf. [Ado06; RB04]). This is why it is important to first define the process to ensure these things and derive from that the necessary properties regarding the product, people, and finally the project.

### 7.2.1 Process

In the previous section, we have already defined that the best possible way to learn from software alternatives is the usage of qualitative experiments. We will use the process to

conduct qualitative experiments as scaffold for our process and combine it with software development to achieve our main goal for this stage. On a high level, the process to conduct experiments do not differ much from other general learning processes like Build-Measure-Learn (see [PP03]) or Observe-Orient-Decide-Act (cf.[Ado06]). Thomke [Tho03], for example, defines the high-level experimentation process as:

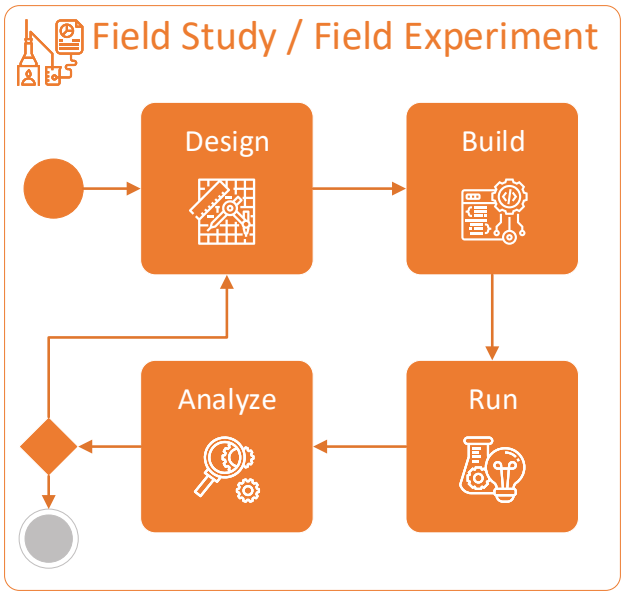



**Fig. 7.2.:** Process Overview for this stage based on general experimentation process by Thomke [Tho03].

1. **Design**: One conceives of or designs an experiment.
2. **Build**: One builds the software needed to conduct that experiment.
3. **Run**: One runs the experiment.
4. **Analyze**: One analyzes the result.

We will use this simple process of Thomke, that can be run iteratively, and refine it into the process we need in this stage (see Figure 7.2). To derive the necessary steps for the first activity **Design** (see Figure 7.3), we will primarily use the features of an experiment described by Hussy, Schreier, and Echterhoff in their book on research methods [HSE13].

The first thing that has to be set in the *Design* of an experiment is the *research question*. For quantitative experiments, this would usually be a falsifiable hypothesis that should be verified. In the case of qualitative experiments, however, no hypotheses are to be verified, but sense and structure are to be explored. Therefore, hypotheses as research questions make less sense here, but rather more research questions that ask about the structure of a research

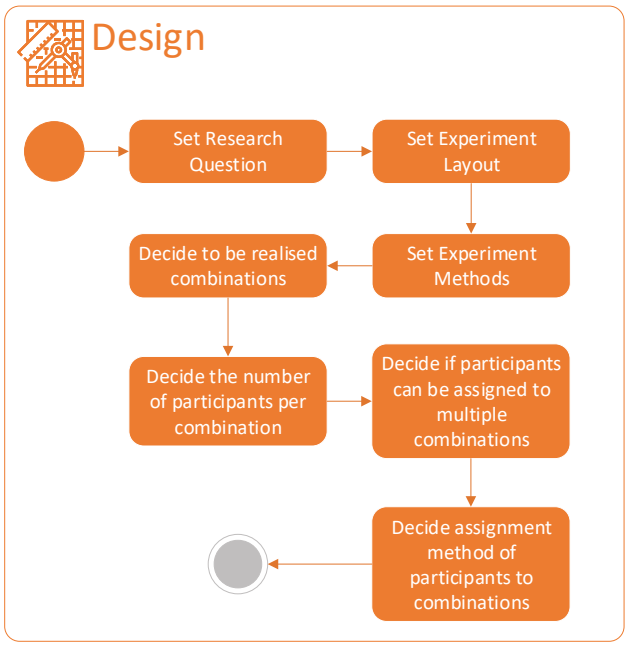



**Fig. 7.3.:** Design activity for field studies / experiments in this stage.

object. In our case, we propose to set the research question as how to deliver the value we identified in the previous stage.

With the research question set, the next step is to *decide for an experiment layout*. An experiment layout is actually an overview of the independent variables used in the experiments with their (possible) variations. This can be unifactorial, if only one independent variable is used, or multifactorial, if several independent variables are used. The experiment layouts purpose is to set which variations are used in the experiment. In case of a multifactorial layout, this indicates which variations of an independent variable are combined with variations of another independent variable in the experiment.

To create such an experiment layout, first the independent variables must be defined. If we are looking only at an objective without further sub-objectives (see section 6.2.2), we will use an unifactorial experiment layout with the objective as independent variable. Otherwise, we have to use a multifactorial experiment layout where the sub-objectives become the respective independent variables. Furthermore, the variations for the independent variables must be defined[1]. For that, we use the software alternatives defined in the digital capture cards for the respective (sub-)objective.

[1]To increase the learning output, the variations can be set systematically. Burkart [Bur10] lists for this the techniques *separation - segmentation*, *combination*, *reduction*, *adjustment - intensification*, *substitution*, and *transformation*. Accordingly, it may make sense to adapt the existing variations in line with these techniques.

The third step in this activity is to *define the experiment methods*. This is about how data in the experiments are actually generated. In a quantitative experiment, it could be that for example eye tracking is used in conjunction with a questionnaire. For our qualitative experiment at this stage, the main methods shall be qualitative methods, which can be supplemented by quantitative methods (see section 1.3).

In software development, one method that gets quite close to what we are trying to do in our qualitative experiment is *Usability Testing* (see [DF12]). It has its origins in Usability Engineering and is one of the most widely used usability evaluation methods (cf. for example [@Bes10]). According to Dumas and Fox [DF12], the basic characteristic of usability testing is

- the focus on usability
- and end users or potential end users as participants,
- who perform tasks with a product or prototype,
- usually while thinking aloud.

This combination of end users working with a product or prototype while thinking aloud is what makes *Usability Testing* quite compelling for this stage. First of all, with this method, the software alternatives are used by real users and not just e.g. reviewed by an expert. Secondly, by having the users stating their thoughts while using the software alternative, we gather qualitative data about what the user expected, what she is trying to achieve next, what was surprising for her, what she did not understand and so on - in short it helps us to further explore sense and structure of the prototype regarding the proposed value.

Furthermore, we can scale up the usability test over several iterations. In the beginning, we can use moderated usability tests conducted on-site or in a laboratory with both participant and investigator physically present. This has the advantage that more details can be observed and the usability test can be conducted more interactively and therefore giving the investigator more possibilities to inquire and challenge her understanding. But this setting is quite resource intensive as both participant and investigator have to be physically present at the same place and time. To overcome the place restriction, moderated / synchronous remote usability tests can be used, which has the disadvantage that the investigator can only observe and interact with the participant via the technical equipment. For overcoming the time restriction as well, the usability test can be conducted as unmoderated / asynchronous remote usability test with the additional disadvantage that the investigator cannot intervene during the test. Besides being able to get feedback from more participants by conducting remote usability tests, they allow to integrate participants who would be difficult to reach otherwise because of the distance as well.

Thus, in a first iteration, insights can be generated with a small number of participants on site. Through further iterations, more and more participants can be integrated remotely, thus generating more generalizable insights step by step. However, this cannot be extended arbitrarily, as there is a limit to the number of participants that can be involved, due to the necessary manual evaluation of the qualitative data and with that the justifiable experimentation costs in relation to the knowledge gained.

With the research question, experimentation layout, and the experimentation method, it can be *decided which variations are tested together (combination) per experiment run*. Among other things, it plays a role here whether the variations have an influence on each other. It may be that one wants to minimize this influence or even cause it in order to observe the correlations.

With the decision about the to be realized combinations, the *number of participants per combination has to be set* in accordance to the experimentation methods. In addition, it has to be *decided if one participant can do multiple combinations or can only be assigned to one combination*.

The last step is to *decide how the participants are assigned to the combinations*. On the level of the experimentation design this can be done randomly or systematically, which is, as mentioned earlier, the difference between field experiment and field study. The systematic approach is particularly suitable if, for example, the learning output is to be increased by using extreme user groups. Randomized assignment makes sense if a higher degree of generalisability is desired or if the aim is to bring to light potentially unthought-of participant introduced effects. This last step concludes the *Design* activity.

**Build** (see Figure 7.4) is the next activity. It actually consists of the three steps *Implementation*, *Data Preparation*, and *Infrastructure Preparation* that can be done in parallel. For the *Implementation of software alternatives*, we have already prepared epics and user stories in the previous stage for agile software development, which makes sense in this stage as we are expecting changing requirements and want to already learn from smaller increments. Hence, we will use agile software development to implement our software alternatives. As agile software development is more like a compendium of ideas or a methodology, we still have to choose a specific method to use. For that we will use SCRUM as the most dominant agile software development method, which has its focus more on the organizational aspects of how to precisely develop software iteratively [Mey14]. In our use of SCRUM, the role of the product owner is taken over by the value designer and the role of the SCRUM master by a person from the development team. Otherwise, we will stick to the process and practices as described by Meyer [Mey14] for implementing the software alternatives for our qualitative experiment.

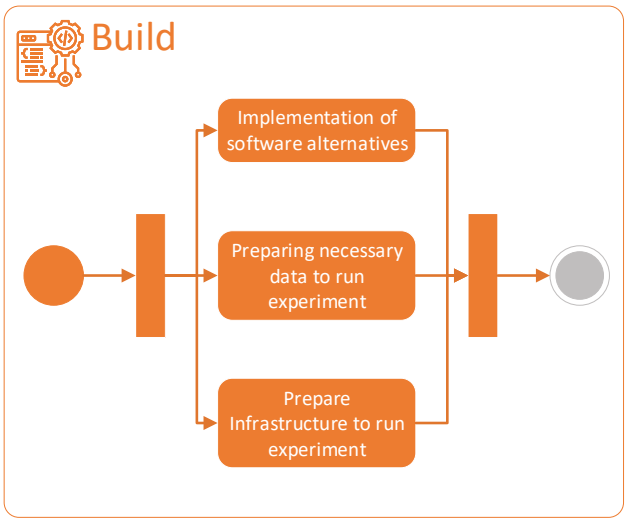



**Fig. 7.4.:** Build activity for field studies / experiments in this stage.

The second step is the *preparation of data necessary to run the experiment*. Usually, software needs additional data to perform certain calculations or to provide functionality. For example, to show events on a map, the software needs events with their geo position and map data (e.g. OpenStreetMap) for the area under consideration. It could be that for this, sample data has to be created / derived from real world data or already existing data has to be prepared for the use in the software. The later could also include that existing data is prepared for a protected use in the experiment, so that errors in the software do not affect the existing data. Data preparation can also mean that the data is prepared in a way that it can be bundled with the software to create a self-contained deployment (all things necessary to run the software is included in the deployment). This is particularly useful if it is intended that the participants should always work from the same starting point.

The last step is the *preparation of the infrastructure for the experiment*. This can include, among other things, the invitation of participants, scheduling of experiments, preparing the data collection, preparing the deployment, or setting up the technical assignment of participants to the according variations. For the invitation of participants, the decision about randomized or systematic assignment has to be considered for the selection of possible participants. In the case of synchronous experiments (investigator and participant simultaneously run through the experiment), appointments have to be found, while in the asynchronous case only the time period, in which it has to take place, has to be defined. For both cases, the scheduling of experiments in relation to other experiments is crucial. It should be prevented that other experiments running at the same time falsify the own experiment. Furthermore, if a ramp-up strategy (starting with a few participants and increasing the number over time, see also [Koh+08, p. 164])) is being considered, this has to be included as well in both the

scheduling and invitation. It is not uncommon, especially in qualitative research, to react to the first runs and adjust the subsequent runs accordingly (e.g. cf. [Hel11]). But with time, less adjustment is needed and more participants can be invited at once.

In addition to these tasks, the infrastructure has to be prepared to be able to collect the data for the experiment. This means that the different data, which can be unstructured, semi-structured or structured, can be stored and related to the experiment, especially to the run (including the variation and the participant). In particular for unstructured or semi-structured data such as videos, transcriptions or field notes, the setup of an appropriate file storage system may be necessary, whereas structured data, like eye tracking, may need special database systems. To be able to collect data, the software needs to be deployed somewhere and run. Hence, a deployment strategy has to be created and the necessary infrastructure be prepared for the deployment. The deployment could be for example into the cloud, on a physical device with internet connection like a smartphone, or an offline system like a machine controller.

Last but not least, the technical assignment of participants to the according variations has to be selected. This depends on the deployment / software architecture as well as the decision about the assignment of participants on experiment level. Kohavi et al. [Koh+08, pp. 163–170] give a good overview for web based applications on how to do this technically. They distinguish between the randomization algorithm and assignment method. The randomization algorithm could be for example the combination of a random number generator with caching of the assignment or hashing the user id and using the modular function to assign a participant to a variation. For the assignment method, traffic splitting, page rewriting on the server or branching in code on client-side could be used for example. However, this does not exclude other technical assignment methods.

With the necessary things built and prepared for the experiment, the next activity is the actual **Run** of the experiment (see Figure 7.5). For each run with a participant, the run has to be prepared, performed, and cleaned up. Preparation can include things like starting the software alternative, initializing it with the necessary data or starting the monitoring. Run is the actual conduction of the experiment, which includes the actual conduction as well as a possible introduction, pre- and post questionnaires / interviews, or data collection in general. Clean-up of the experiment can include reverting changes done during the experiment and should always include a reflection of it, to be able to adapt in the next runs to insights made in the current run.

The last activity in our process is **Analyze**. The first step in this activity is to *prepare the data for analysis / exploration*, which can include compiling, arranging, coding and processing, actually every step necessary to work with the data. With the data prepared, the *data analyzation and / or exploration* can be done for example with the help of qualitative

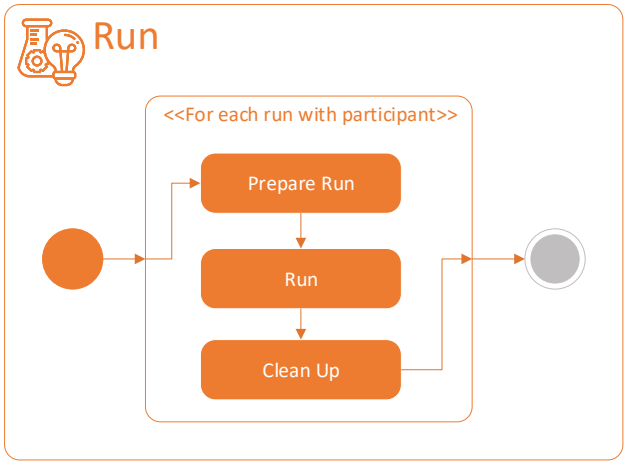



**Fig. 7.5.:** Run activity for field studies / experiments in this stage.

content analysis or any other mean appropriate for that. Finally, the *results of the analysis have to be interpreted* and *the next steps have to be derived*, e.g. moving on to the next stage *Optimization*, adjustments to the experiment, or jumping back to one of the previous stages. This concludes the process in this stage.

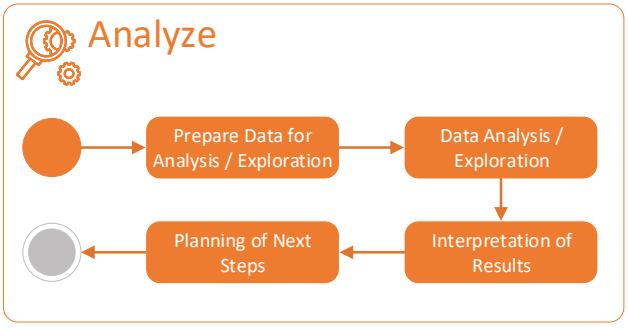



**Fig. 7.6.:** Analyze activity for field studies / experiments in this stage.

### 7.2.2 Product

The presented process for experimentation results in certain requirements for the product. First of all, it must be possible to operate different variations / software alternatives simultaneously. This includes that with the help of the alternatives the best fitting solutions should be found and a large part of the alternatives will be obsolete. Furthermore, changes over time are expected, but also short-term changes due to findings in an experiment run. These three points *changes over time*, *alternatives to experiment with*, and *finding the best*

*fitting solutions* are actually describing evolution, which is no unknown concept in software development. Denning [DGH08] discusses evolution on the level of software development and distinguishes between two ways of making software development evolutionary. The first is the successive release of a system, which is the familiar process of software product releases. Challenge with this way are for example the required very short release cycle for the full system, the inflexibility regarding technologies, or the configuration complexity to run multiple experiments. The other way is having many systems competing by mimicking natural evolution.

This way of having many systems that make up a composite system is called *System of Systems*. Damm and Vincentelli [DV15] define "*System of Systems* as a recursively defined entity where the top element is made of a set of [constituent systems]. Each [constituent system] may be itself a system of systems." Furthermore, they cite coopetition as a further characteristic of *System of Systems*, which means a mixture of cooperation and competition between the individual systems to achieve a higher goal. Klein and van Vlient [KV13] further define it as "an assemblage of components which individually may be regarded as systems, and which possesses the additional properties that the constituent systems are operationally independent, and are managerially independent". Under operationally independent they understand, that "each constituent system operates to achieve a useful purpose independent of its' participation in the system of systems". Managerial independence is defined by them as "each constituent system is managed and evolved, at least in part, to achieve its' own goals rather than the system of systems goals". This idea of *System of Systems*, where the constituent systems are operationally and managerially independent is actually the kind of macro-architecture (cf. section 6.1 and [HW95]) we are seeking for this stage.

#### 7.2.2.1. Macro-Architecture

In the following, we will discuss how such a *System of Systems* macro-architecture for qualitative experiments can be achieved. For this purpose, it must be answered how managerial and operational independence can be established, for which we are adapting existing patterns from other areas like scalability.

To establish **managerial independence**, the first thing we do is to look at the system from a domain perspective. In software development, domain-driven design was realized to actually incorporate a systematic and effective domain modelling (see [Eva04]). Part of it is the pattern *Bounded Context* (see Figure 7.7 for a schematic illustration), which "delimits the applicability of a particular model so that team members have a clear and shared understanding of what has to be consistent and how it relates to other contexts" [Eva04, p. 335 ff.]. By omitting the domain of other bounded contexts and only looking at your own, the

coupling, coordination, and complexity should be simplified (see [FPK17]). Furthermore, with a ”larger domain, it gets progressively harder to build a single unified model” as for example people start to use subtly different vocabularies [@Fow14]. Therefore, bounded contexts are well suited to break up a larger system into smaller parts that can be managed independently.

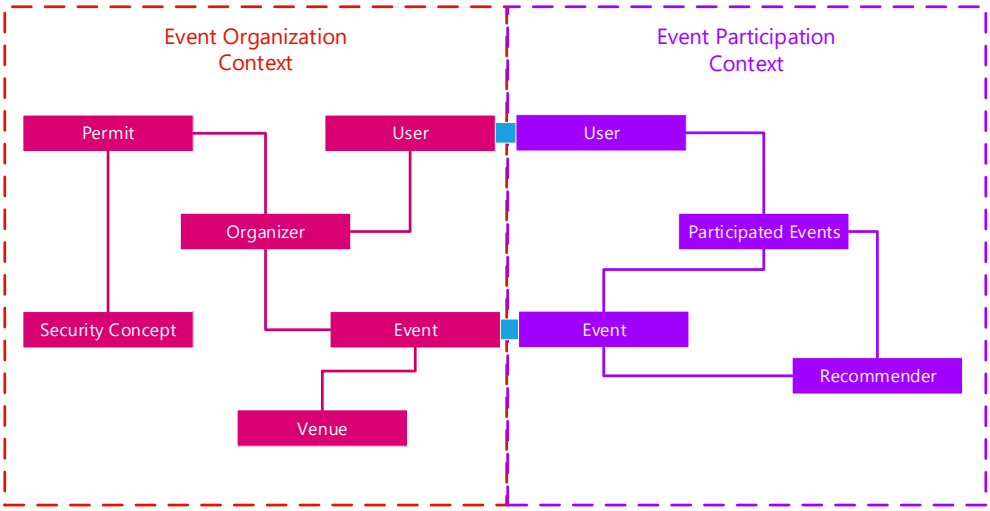
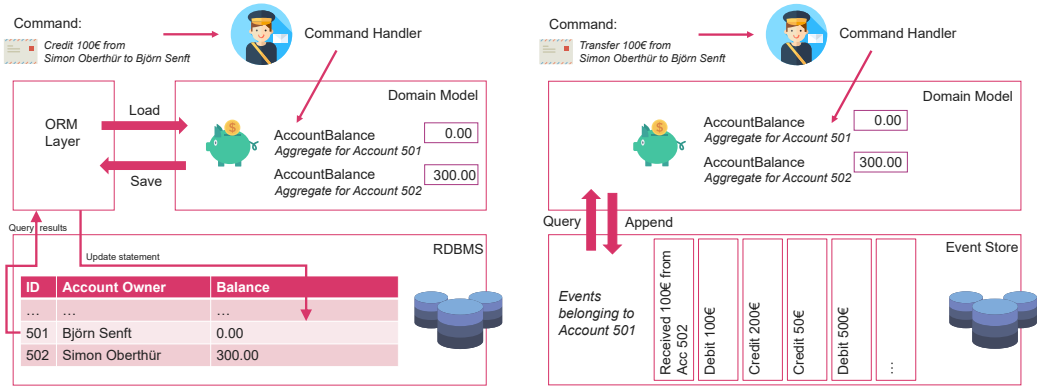



**Fig. 7.7.:** Schematic Illustration of two Bounded Contexts.

To partition bounded contexts, we can use our hierarchy from the previous stage (see section 6.2). If a subtree in this hierarchy becomes too complex, we can extract it into a new bounded context. We can also view our variations / software alternatives as bounded contexts. However, with this and generally for bounded contexts, it cannot be guaranteed that concepts are always located in only one bounded context. For example, in Figure 7.7 the concepts *User* and *Event* are concepts shared between *Event Organization Context* and *Event Participation Context*. To enable such a partition but still be managerially independent, we need parallel models.

A parallel model is actually an alternative representation of the state of a concept like *User* in *Event Organization Context* and *Event Participation Context*. To achieve such a parallel model, Fowler [@Fow05b] states that the best is if the system is designed with *Event Sourcing* (In this case, event means a technical event as in an event-driven architecture and not event in the sense of the OWL.Culture-Platform.). The idea behind *Event Sourcing* is to not persist the current state of the objects like in active records, but the events that lead to the current state (cf. [Bet+13; @Fow05a] or Figure 7.8 for an example). It is actually a quite common and intuitive pattern that is used in software development for example in code versioning systems like svn or transaction logs in sql. Best known examples are probably bank (saving) accounts where the truth lies in the individual deposits and withdrawals (transactions) and

the total value is only displayed as additional information. Lawyers use a similar procedure to edit contract texts; it is not the text itself that is edited, but a change list is maintained.




**(a)** Active Record **(b)** Event Sourcing

**Fig. 7.8.:** Example Comparison of Active Record and Event Sourcing.

And how does *Event Sourcing* help us to maintain parallel models? With *Event Sourcing* we are getting two features that are crucial for this. First of all, we get an *Event-Driven Architecture*, which helps on the one side to decouple the systems more as they omit the need to subscribe to a full domain model. On the other side, it makes the synchronization simpler as we do not have to compare two states and figure out the differences but get the changes directly. Secondly, we get an audit log, which helps us to identify the reason for an incorrect state and to repair it easily by introducing a retroactive event (An event undoing something that happened in the past). Furthermore, the audit log helps us to merge changes from two or more services and check their permissiveness. In this way we can achieve on the one hand that we can experiment with several alternatives simultaneously. On the other hand, existing *System of Systems* can operate the old and new version of a system simultaneously in this way until the complete *System of Systems* is converted to the new system. In this way, we reduce the potential costs for experimentation and technology decisions.

But of course, *Event Sourcing* has as well some drawbacks. In order to understand them, it is among other things important to understand which assumptions are made around *Event Sourcing*. First of all, Betts et al. [Bet+13] state as essential characteristics of events in *Event Sourcing* that *events happen in the past*, *events are immutable*, *events are one-way messages*, *events include parameters that provide additional information about the event*, and *events should describe the business intent*. Therefore, events must not be changed, which also applies to faulty events as well as to subsequent changes in the structure of an event. For faulty events, we can use the previously mentioned retroactive events. In the case of subsequent changes to the structure of an event, this means that special care must be taken with the initial design of the event and it should not include too much context. For example,

an EventChanged event in our OWL.Culture-Platform example would be a quite bad design as all attributes later on added to an event results in a new structure of the EventChanged event. It would be better to have smaller events like a EventVenueChanged event. This also results in challenges in data protection, for example if personal data is to be deleted. Therefore, it is not completely forbidden to change the events afterwards. However, it should remain the exception and not the normal way of working with the events (cf. [OSJ17] for examples on challenges and how to update events in *Event Sourcing*).

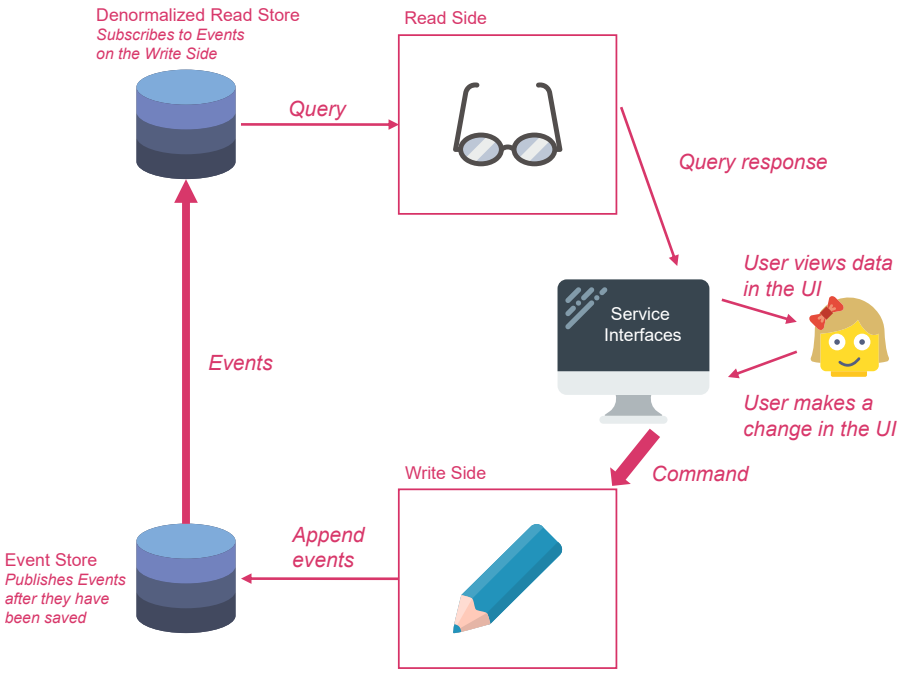



**Fig. 7.9.:** Event Sourcing and Command Query Responsibility Segregation Pattern Combined.

Furthermore, *Event Sourcing* introduces an especially big challenge regarding performance and querying. Depending on the number of events, it could take a long time to replay the events to load a domain object's state. Furthermore, it is quite difficult to search for example relational data within events. This is why it is recommended to combine *Event Sourcing* with the *Command Query Responsibility Segregation (CQRS)* pattern (see [Bet+13, p. 243]). *CQRS* describes in its core that you use a different model to update information than you use for reading information (cf. [Bet+13, p. 223 ff.] or [@Fow11]). In Figure 7.9, we have illustrated how *CQRS* works in conjunction with *Event Sourcing*. When a user wants to view data, a query on the read side is made, which gets her the corresponding data. If she makes changes in the UI, a command is created that is sent to the write side, which checks the commands permissiveness. The difference between a command and an event is that the event is always in the past and the command is an intent to change something. When the check is successful, an event is created and appended in the event store. An event store is actually the specialized database to store our events for the event sourcing. The read store

has subscribed to the event store and gets the corresponding new events, which it handles to update itself. Like that, we can use as read store e.g. a relational database like PostgreSQL, a document oriented databases like MongoDB, or different database types in parallel, but still persist our data in an event store.

Another important perspective on the managerial independence, besides the domain perspective, is the perspective on the different layers we usually have in a software. One quite common separation in layers is the three layered architecture *Presentation*, *Domain*, and *Data* (see Figure 7.10). *Presentation* is in the sense of the four layered architecture by Evans [Eva04] a combination of *User Interface* (Showing information to the user and interpreting her commands) and *Application Layer* (Defines the job the software has to do and directs it to the domain objects). *Domain* is the same as the *Domain Layer*, which means it is responsible for representing the concepts of the business, information about the business situation, and business rules. *Data* is just for accessing and storing the data and is a specialized part of the *Infrastructure Layer* in the four layered architecture, which shall in general provide generic technical capabilities. One concept, that is quite important here is the *Presentation Domain Separation* (see [@Fow03; Fow01]), as it allows to (more) independently develop the *Presentation* layer from the *Domain* layer.

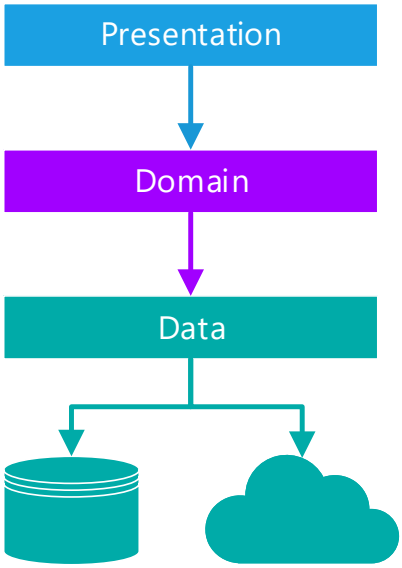



**Fig. 7.10.:** Presentation-Domain-Data-Layering. Own representation based on [@Fow15]

Take for example an online banking account. In this setting, it is important e.g. for an user to not lose money because of an unreliable system. Hence, the data integrity especially regarding managing the transactions (and depending on the system the current balance) is essential. However, new interactive forms of presentation or services that work on the data can be interesting and have a lower expectation regarding reliability as long as they do not affect data integrity. For example, having a virtual division into budgets (household, work, leisure) or a visual breakdown of regular debits. To realise these functionalities, the persisted data does not necessarily have to be touched, but for the presentation the data may have to be structured differently and aggregated with data from other services. This is why especially *Presentation Domain Separation* is so important for experimentation, since the *Presentation* layer often requires a different change cycle than the *Domain* layer. It is also what Sharma and Coyne [SC17, p. 4] mean when they are saying that *Systems of engagement* need to be capable of more rapid changes compared to *Systems of record*, which are only there to keep and present the data without much interaction.

To achieve *Presentation Domain Separation*, the *Model-View-ViewModel (MVVM)* (see Figure 7.11) pattern is quite useful as its main goal is to decouple *Presentation* from *Domain* as good as possible to enable designers to independently develop *Presentation* without the need of taking into consideration the actual model in *Domain*. *MVVM* is actually a combination of the patterns *Model View Presenter (MVP)* and *Presentation Model* (see [@Smi09]). The basic separation in *MVP* (see [Pot96; @Smi09]) is that what you see on the screen is the *View*, the data it displays is the *Model*, and the *Presenter* hooks the two together. It is no coincidence that this pattern is very close to *Model View Controller* as it has its origins in it [Pot96], but with the main differences of the required observer synchronization between *View* and *Presenter* and a one to one relation between *View* and *Presenter* [@Fow06]. The pattern *Presentation Model* on the other hand is simply ”pulling the state and behavior of the view out into a model class that is part of the presentation” [@Fow04]. Furthermore, it is intended that the *Presentation Model* takes over the coordination with the *Domain* layer as well as providing an interface for *Presentation*, so that decision making can be minimized in *Presentation*.

By putting these two patterns together, Smith [@Smi09] explains *MVVM* as having a *View* that binds to properties on a *ViewModel*, which, in turn, exposes data contained in model objects and other state specifics to the view. Changes from the view are delegated via *Commands* to the *ViewModel*, which updates the *Model*. If property values in the *ViewModel* change, those new values automatically propagate to the *View* via data binding. The *View* itself binds to the properties of the *ViewModel* simply by setting it as the data context of the *View*. Furthermore, the *ViewModel* does not need a reference to the *View* unlike in the case of *MVP*, The reason are the bindings that are used to propagate the properties and commands to delegate actions triggered in the *View* to the *ViewModel*. Although *MVVM*

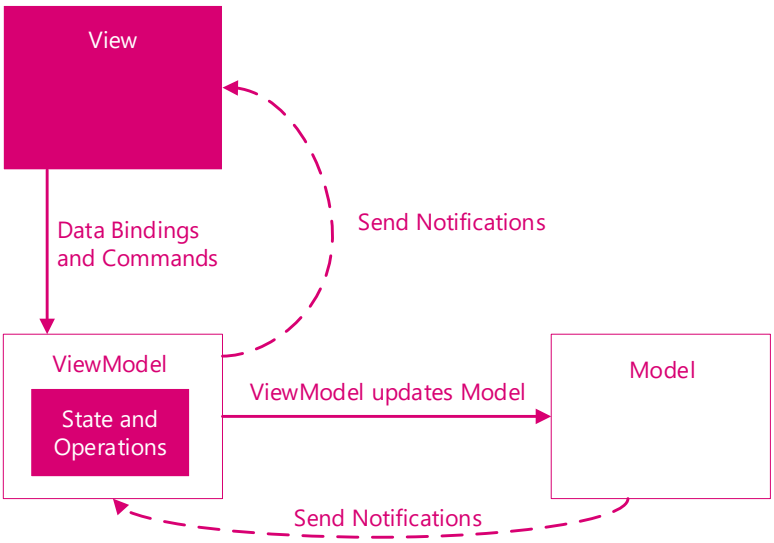



**Fig. 7.11.:** Model-View-ViewModel (MVVM) Pattern. Own representation based on [@Mic12]

was originally developed for the use in the *.NET* context, it is a general pattern that is used in other technologies as well like Angular[2] or in native Android applications[3].

Hence, with the help of the *MVVM* pattern we can achieve a managerial independence of *Presentation* and *Domain* layer, which helps us to use different technologies for both layers. In addition, it allows us to independently develop the *Presentation* layer, which allows shorter iterations and with that a more fitting developing situation, especially to experiment with alternatives on the different layers. If we wouldn't have this, we would need to use the same high requirements for every part of the system.

In summary for managerial independence in general as prerequisite for *System of Systems*, we can achieve it at domain level by using *bounded contexts* with *Event Sourcing* and *CQRS* for realization. On the software level with the different layers, we can use *Presentation Domain Separation* with the help of the *MVVM* pattern.

Besides the managerial independence, we also need **operational independence** to realize our *System of Systems* macro-architecture to support experimentation. For the constituent systems, that can be derived from the bounded contexts, this means that they can be deployed and run independently of other constituent systems. To achieve this, the first thing should be that each constituent system can run in their own process. In order to be able to run a constituent system, it has to be deployed, which makes an independent deployment as well a requirement for operational independence.

[2] https://angular.io/start/start-data
[3] `https://developer.android.com/topic/libraries/data-binding`

But what does independent deployment mean? For that, we can take as foundation a typical *Continuous Integration and Deployment Pipeline* (see Figure 7.12) as for example described by Shahin, Babar, and Zhu [SAZ17] or Ford, Parsons, and Kua [FPK17]. In the first step, the developer commits the code into the code versioning system, from where it is taken into the *Continuous Integration* pipeline. Within the *Continuous Integration* pipeline the code is built and tests like unit testing and static code analysis are done. If everything went successful, the build artifact is handed over to the *Continuous Deployment* pipeline, where it is deployed into a staging environment to run automated acceptance tests. If the acceptance tests are successful as well, the build artifact gets automatically deployed into the production environment. The difference between *Continuous Deployment* and *Continuous Delivery* is the needed manual approval in *Continuous Delivery* to deploy build artifacts to the production environment. If all these steps can be performed without relying on parts of the other constituent systems, one can consider it an independent deployment.

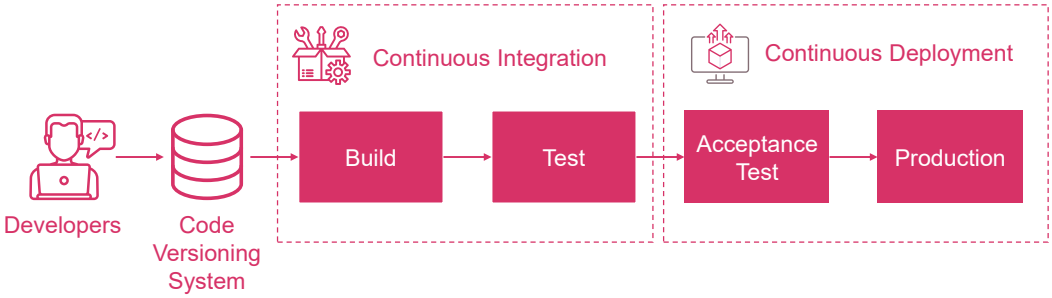



**Fig. 7.12.:** Continous Integration and Deployment Pipeline. Own representation based on [SAZ17].

In detail, this means that code of the individual constituent system needs to be separated to be able to build the constituent systems independently. Furthermore, build dependencies to other constituent systems must not be present or rather have to be minimized. The same applies to the tests on integration level, which can be carried along with the source code. Acceptance tests should be divided into one of two areas. On the one hand, acceptance tests that only refer to the individual constituent system and, on the other hand, acceptance tests that refer to the interaction in the *System of Systems*. In a *System of Systems* it will not be possible to avoid that acceptance tests also refer to the interaction of the originally independent constituent systems. However, for an acceptance test that fails in this case can not be assumed that the constituent system that triggered the acceptance test is to blame. With this separation it can be ensured that the constituent system, if considered separately, fulfills the requirements, even if it leads to an error in a deployment in a specific *System of Systems*. For the last step to be considered as a completely independent deployment, the deployment artifact must include everything needed to run the constituent system on any system (infrastructure). This can include frameworks, libraries, or even data. Overall, there may of course be nuances in terms of independence. For example, a deployment artifact does

not necessarily have to be executable itself, but can only provide the necessary references to libraries for cross plattforms or a special platform (see e.g. .NET Core Publishing[4]).

The last characteristic for operational independence that we consider is the communication between constituent systems. On the one hand, it must be ensured that the constituent systems are connected via common standardized interfaces such as *Representational State Transfer (REST)* (e.g. cf. [@Fow10] for a detailed explanation of *REST*). Furthermore, it must be possible to configure references to other constituent systems dynamically at runtime. By combining these two points, the constituent systems can be connected to each other at runtime, provided that they offer the required functions. Besides the common standardized interfaces, it is also essential for communication to use the *BASE (Basically Available, Soft state, Eventual consistency)* semantic instead of the *ACID (Atomicity, Consistency, Isolation, Durability)*. *BASE* was introduced by Pritchett [Pri08] for scaling databases and is intended to be optimistic and accepting that the database consistency will be in a state of flux. *ACID* in contrast is pessimistic and forces consistency at the end of every operation. For our communication using the *BASE* semantic means, that we acknowledge the fact that the other constituent system is available as much as possible, but without any kind of (consistency) guarantees (Basically Available). Furthermore, it means that we are in a soft state which has a probability to be in the correct state after some amount of time, but it is not guaranteed. The last point is that we will be eventually consistent with our expectations, if the system is functioning and we wait long enough after any given set of inputs. By incorporating the *BASE* semantic, we ensure that our constituent systems can handle communication errors and longer response times. This enables us to e.g. redirect a request from a failing instance of a constituent system to another. In best case, our *System of Systems* or constituent system is so designed that it can still serve its purpose even without having the other constituent system answering. Netflix for example is so designed that e.g. if the bookmarking service fails, the overall system is still capable of serving streams (cf. [Bas+19]). For our experiments, this means that we achieve with this a fallback mechanism, which is essential for experiments in the actual context of use as they allow less mature systems to be tested without interfering to much with the actual work (be it explicit test sessions in which no work can be done or failing systems that inhibit working).

In this part, we have shown how to achieve a *System of Systems* architecture, where the constituent systems are managerial and operational independent. For managerial independence, we are using *bounded context* to get a separation on domain level with the help of *Event Sourcing* and *CQRS*. A further separation on the level of the three layer architecture is introduced with *Presentation Domain Separation* and *MVVM* to realize this separation. Operational independence is achieved by having each system run in it owns process, being independetly deployable, and having a standardized communication based on the *BASE*

[4] `https://docs.microsoft.com/en-us/dotnet/core/deploying/`

semantic and configureable during runtime. This *System of Sytems* architecture is important for actually conducting our qualitative experiments (cf. section 1.1.3). In the following, we will give an example on how these patterns are instantiated with specific technologies in one of our projects.

#### 7.2.2.2. Implementation Example

We will use the OWL.Culture-Portal (see section 8.3) to explain how a technical implementation of the aforementioned patterns can be realized. Since the OWL.Culture-Portal is intended to be an event site, among other things, the use of web technologies for its realization is natural. However, this generally makes sense for *System of Systems* because as Denning [DGH08] points out, the www is one of the oldest and most mature examples of a functioning *System of Systems*.

Moreover, web technologies do not mean that physically separate units have to communicate with each other. You can also start web servers on your own computer and access them via the loopback network. For example, Edirom[5] has been implemented as a client application with web technologies, where an application server is started within an Eclipse environment and the Eclipse environment displays a full-screen web view pointing to the application server. For the user it feels like a native application. With the upcoming of *Progressive Web Apps (PWA)* [@RB15; @Osm15], this workaround with a specific wrapper application is not necessary anymore as applications are directly installable from the browser. Furthermore, app store provider like Microsoft are going to offer *PWAs* in their app stores and have implemented the possibility to add them manually or automated by a web crawler, if certain properties are fulfilled[6]. In addition, with WebAssembly[7], *PWAs* are capable of running high-level languages like C/C++/C#/Rust natively in the browser, which allows further polyglotism regarding language and technology.

In the following, we will present our selection of standard technologies. Although we want to allow polygotism in terms of technology and language, it makes sense to agree in a project on a set of standard technologies to be used if there is no serious reason against it. If we look back at the expert discussion in section 4.1, it is noticeable that the expert characteristics also apply to developers. This means that no developer can be an expert in every technology, and thus progress equally well and quickly in every technology. If we use a high variety of technologies, we make it more difficult for the developers to support each other and to move forward with the project, because they always have to learn again about the technology in

[5] https://github.com/Edirom/Edirom-Editor
[6] https://docs.microsoft.com/en-us/microsoft-edge/progressive-web-apps-edgehtml/microsoft-store
[7] https://webassembly.org/

question. Nevertheless, if it is about the use of an already well-implemented constituent system, where only minor changes have to be made and it is not a core component, or if the tool support of a technology is much better than the standard set and thus justifies the effort, the use of other technologies is definitely desired.

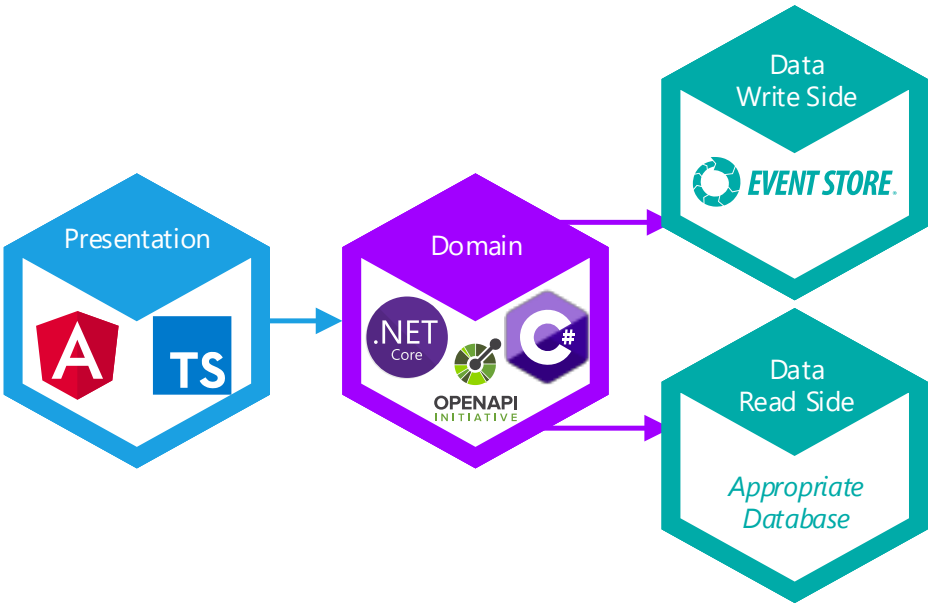



**Fig. 7.13.:** Implementation of Presentation-Domain-Data-Layering in OWL.Culture-Portal. The hexagon representation is based on the microservice representation by NGINX and is meant to emphasize that these are constituent systems.

With these general decisions, we can first address managerial independence by starting with *Presentation Domain Separation*. In Figure 7.13, we have illustrated the separation in the individual layers *Presentation*, *Domain*, and *Data* in the OWL.Culture-Portal and which basic technologies we use in which layer. Based on the *Bounded Contexts* we create separate constituent systems for each layer to allow an individual evolution as stated previously. For the *Presentation* layer we are using *Angular* with *TypeScript* as base technologies. *TypeScript* as programming language allows a better testability than javascript because of the static typing and is the default language for *Angular 2+*. *Angular* has the advantage that it natively implements the *MVVM* pattern and supports *PWA* out of the box. Furthermore, it structures the user interface into components which can be exported into web components. This helps us to load and unload components during runtime but also the to be created components to be used in other web frameworks and therefore making the decision about the technology less sticking. The communication with the consituent systems in the *Domain* layer is abstracted away from the user interface code by so called services in *Angular*, which

inject the necessary data into the view via dependency injection. This way, even the HTML code becomes more reuseable as it does not need to know about specific classes and angular code but can just bind to them like originally intended with *MVVM* in *WPF* for example. Unfortunately, it could be that additional *Angular* annotations are introduced to the html code to be working with *Angular* as framework. The connection of the service with the constituent system on the *Domain* level is realized via HTTP calls on a *REST* interface.

Therefore, constituent systems on the *Domain* level have to offer a *REST* interface. To achieve this, we are using *Web API* projects from *ASP.NET Core* in conjunction with *C#* and the *OpenAPI Specification*. *OpenAPI Specification*[8] is a standard for programming language-agnostic interface descriptions of *REST APIs*, which is backed by the Linux Foundation. Besides the standardized description of *APIs*, it allows the automatic technology related generation of interfaces. The decision for *ASP.NET Core* with *C#* was actually done in a previous project (see section 8.1) with the intent to harmonize the technologies for a cross plattform mobile development. At that point in time, there was only the choice between pure javascript frameworks for backend and frontend or *Xamarin* with *C#* as basis for cross platform development and *ASP.NET Core* for the backend. Because of the better testability, we decided to use *ASP.NET Core* with *C#*. Nevertheless, *ASP.NET Core* itself also offers some advantages[9] such as running on different plattforms (e.g. Windows, Linux, Mac), being ready for containerization (.NET Core is more modular and lightweight than .NET which makes contaienr images much smaller), the possibility to install applications side-by-side (.NET Core allows side-by-side installation of different .NET Core runtime versions on the same machine and with this different applications that require different .NET Core runtime versions), and having with *EF Core* a mature object relational mapper.

On the *Data* layer, we are implementing *CQRS* with *Event Sourcing* by using *Event Store*[10] on the *Write Side* and an appropriate database (like e.g. PostgreSQL, MongoDB, or Neo4j) on the *Read Side*. *Event Store* is an event store, which is a ”mechanism to store events and to return the stream of events associated with an aggregate instance so that events can be replayed to recreate the state of the aggregate” [Bet+13]. Using *Event Store* as an event store especially developed for event sourcing has some benefits. First of all, unlike a message broker, it is optimized not only for the distribution of events, but also for their persistence. For each class or instance of a consituent system, a message stream to persist and distribute the events can be created. It has already integrated the mechanisms for access control lists (ACL) and thus enables data protection. This is especially useful if the experiment data is confidential. Furthermore, it allows *temporal correlation queries* called *projections* to query event streams for events related to certain criterias and time, which is quite helpful for new

[8] `http://spec.openapis.org/oas/v3.0.3`
[9] `https://docs.microsoft.com/en-us/dotnet/standard/choosing-core-framework-server`
[10] `https://eventstore.com`

value propositions build ontop of the data of already existing value propositions. Finally, it has already been optimized for scalability and performance in the field of event sourcing. The *Domain Layer* is actually responsible for synchronising the write and read side.

In addition to the use of technologies to achieve managerial independence, we also give an example of how operational independence can be supported by technology. Ensuring that each constituent system runs in its own process can be achieved straightforward by putting them in a container, which encapsulates the code and all its dependencies so that it can run uniformly and consistently on any infrastructure[11]. As container engine Docker[12] or Podman can be used.

For the build and deployment preparation a continuous integration and deployment pipeline like for example integrated in GitLab, Jenkins, or Azure DevOps can be used. The actual deployment or orchestration of the container is usually done by a container orchestration engine like *Docker Compose*[13] or *Kubernetes*[14]. Using *Kubernetes* has the advantage that it is part of the *Cloud Native Computing Foundation* and regarded as the defacto standard in this area (cf. e.g. `https://github.com/containers/libpod#out-of-scope`). Furthermore, by using *Kubernetes YAML* as file format to describe the orchestration of a constituent system allows the deployment on different infrastructures. The infrastructures can be e.g. cloud based with *Kubernetes* itself, a local machine with *minikube*[15] or with podman[16] (which eliminates the need of kubernetes services), and with *K3S*[17] even IoT & Edge computing based. This also concludes our implementation example.

### 7.2.3 People

The process for qualitative experiments and the macro-architecture result in certain requirements for the people involved in this stage. Let us start with the creation of qualitative experiments. Setting up good qualitative experiments from scratch usually requires good knowledge about empirical research with people like it is the case in social sciences or in psychology. However, we cannot assume that this is a universal requirement for all people involved in this process. It is illusory to assume that, in addition to their expertise in their specialist areas, all developers can build up the necessary expertise to set up qualitative experiments independently (cf. Expert discussion in section 4.1). However, they should have a basic understanding of qualitative experiments and roughly understand which properties

[11] `https://www.ibm.com/cloud/learn/containerization`
[12] `https://www.docker.com/`
[13] `https://docs.docker.com/compose/`
[14] `https://kubernetes.io/`
[15] `https://github.com/kubernetes/minikube`
[16] `https://podman.io/`
[17] `https://k3s.io/`

are required in the variations / software alternatives and which are not necessary for the experiment. And of course, as they are intended to develop the variations / software alternatives with the help of SCRUM, they need basic knowledge of agile software development as well.

In order to be able to set up good qualitative experiments nevertheless, we use the value designer (see paragraph 3.1) and suggest a tool that supports the setup (see section 7.3.1). The person holding the role of value designer is primarily responsible for the design of the qualitative experiments. Through her role in the previous stages, she should have the best understanding of the value propositions, which is highly valuable for the design. It is therefore essential for her to have the necessary expertise to carry out independently qualitative experiments with people. Furthermore, she is the first contact person for the developers for questions regarding the experiment and necessary properties of the variations / software alternatives for the execution of the experiment. Accordingly, she must be able to make technical assessments. As the primary person responsible for the qualitative experiments, she has to schedule as well the experiments with the participants, which brings us to the next group of people involved in this process.

A qualitative experiment puts special requirments on the participants too. Within such an experiment it is not sufficient to just use the variation / software alternative. At the same time, the participants have to express (thinking aloud) as well as possible for example what is on their minds, what they expect, what bothers them, or what they liked. The better they are capable of expressing themselves, the better we can understand the situation. Of course a good investigator can intervene here, but only to a certain extent.

With regard to the macro-architecture, the developers are faced with a challenge. The widespread use of the selected patterns has only recently begun, especially in the field of cloud computing for scalability and performance (cf. e.g. [Bet+13]), but not yet for conducting experiments. Accordingly, developers need to become familiar with the corresponding patterns and understand how they relate to experiments.

### 7.2.4 Project

Jacobson, Booch, and Rumbaugh [JBR99] define *Project* as the ”[...] organizational element through which software development is managed” and has as outcome a released poduct. In our context, the outcome of a project would be the experiment results. Accordingly, for each value proposition identified in the previous stage we would set up a new project and thus a new experiment (cf. *Set Research Question* step in *Design* activity).

For the realization of the project, at least one value designer is needed, who can also be a developer. Furthermore, at least one developer is required. Regarding the participants, if we use the criteria for usability tests, we need at least five participants, who cannot be value designer or developer in this project. Furthermore, the required infrastructure such as a cloud computing infrastructure or the tools for automating process steps (see also section 7.3) must be set up for the project.

This concludes our presentation and discussion on how we can try out different software alternatives in order to learn from them to what extent they provide value and which characteristics this may be due to in software development with the help of *The Four Ps (Process, Product, People, and Project)*. In the next section, we will give an overview of how single steps in the process could be automatized with specific tools.

## 7.3 Tools

The Four Ps not only consist of *Process*, *Product*, *People*, and *Project*, but includes *Tools* to automate the *Process* as well. Jacobson, Booch, and Rumbaugh [JBR99, pp.22] state that the process is strongly influenced by tools as ”tools are good at automating repetitive tasks, keeping things structured, managing large amounts of information, and guiding you along a particular development path”. Tools are therefore integral to the process and are essential for the viability of it as they influence the resources needed to conduct a process as well as the effectivity. For this reason we are presenting and discussing tools in this section to give evidence for the viability of the process. In section 7.2.2.1, we already mentioned continuous integration and continuous deployment pipelines as a tool for automatizing the build and deployment process of a variation / software alternative. Looking at our process for qualitative experiments, we have the four major activities *Design*, *Build*, *Run*, and *Analyze* from which we can derive further tools for automating / supporting our process (see Figure 7.14).

Looking first at *Design*, it becomes clear that a tool support will be more related to guidance and structuring than automating as it doesn't include many repetitive tasks but mainly human decisions. We explore in section 7.3.1 (*Experiment Design System*) how such a constituent system that guides and structures the design process can look like. As it is setting up the experiment, we further use it to initialize a project (qualitative experiment) and to coordinate with the other constituent systems in our tool suite.

The next activity *Build* consists of the steps *Implementation of software alternatives*, *Preparing necessary data to run experiment*, and *Prepare Infrastructure to run experiment*. For the first step, we already introduced the continuous integration and continuous deployment

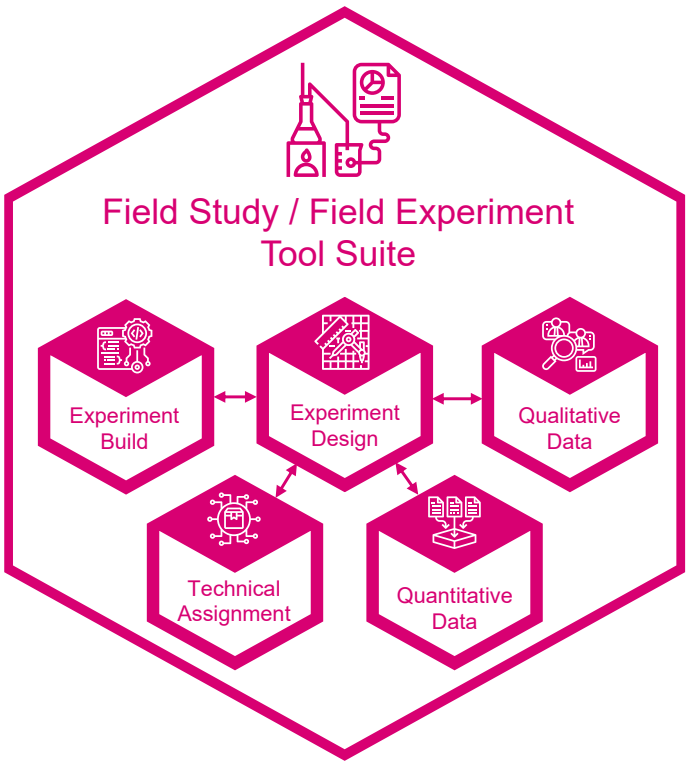



**Fig. 7.14.:** Overview of Tools for Automating the Field Study / Field Experimentation Process.

pipeline which depends strongly on the technologies used. In general, tools for conducting continuous software engineering (cf. [Bos14]) can be beneficial for this step. The next step *Preparing necessary data to run experiment* is quite depending on the implementation, which is why we can not further say anything about it. The preparation of the infrastructure is done in the *Experiment Design* constituent system as it setups the project and coordinates with the different constituent systems. Accordingly, the tools in the *Build Experiment* constituent system depend heavily on the technologies used, which is why we will not go into this in more detail at this point.

For the activity *Run*, we first have to enable the technical assignment, so that the respective participant receives the variations she is supposed to receive. We are using a *Technical Assignment* constituent system for this, which we further describe and evaluate in section 7.3.2. Furthermore, the qualitative and quantitative data has to be collected. For gathering qualitative data, especially with thinking aloud in a usability test, there are already great tools out in the field like Morae[18], Validately[19], lookback[20] or for doing surveys for example LimeSurvey[21]. This is why we are not going to look further into the *Qualitative Data* constituent

[18] `https://www.techsmith.com/tutorial-morae-current.html`
[19] `https://validately.com/`
[20] `https://lookback.io/`
[21] `https://www.limesurvey.org/`

system at this point either. In case of quantitative data, it is mainly about collecting the data, which is what we looked at and evaluated in section 7.3.3. The reason, why we looked further into the *Quantitative Data* constituent system, is the question if our event sourcing infrastructure can be used for that as well, to use synergies.

Our last activity is *Analyze*, which highly depends on the selected method and the data itself, which is why we have not introduced a separate constituent system for it, but include it into the *Quantitative Data* and *Qualitative Data* constituent systems.

In the following we give an overview of how the non-standard parts of the constituent systems *Experiment Design*, *Technical Assignment System*, and *Quantitative Data System* can be designed. The exemplary systems in these parts have been developed and evaluated in conjunction with the the master theses [Sch19] (*Experiment Design System*), [Abi19] (*Technical Assignment System*), and [Kra18] (Quantitative Data System) based on the previously introduced concepts for qualitative experiments and partly on the overview of Kohavi [Koh+08] for concepts in online controlled experiments. We present these exemplary systems in more detail at this point in order to provide evidence for the support possibility by tools and thus also the viability of the process.

### 7.3.1 Experiment Design System

In our discussion about the people (see section 7.2.3), we already mentioned the challenge of defining experiments as a normal software developer. Even though we have assigned the main responsibility for designing experiments to the value designer, who must have the appropriate background knowledge, we think it makes sense to have a system that supports the creation of experiments. First of all, it can help normal developers understanding the terms and basic structure of an experiment including the rationales behind it. Furthermore, a structured experiment design that is already digitally available can be further used in other systems to e.g. initialize them accordingly. We therefore developed a system called *FEXP (Feature Experimentation Platform)* that lets you define qualitative or quantitative experiments, which has a guided mode to support the understanding of the terms and guiding through the individual steps (see Figure 7.15). However, this is only a first prototype to develop an understanding of the usefulness of the guided mode and the feasibility of interactions with other services. In the following, we will present and discuss this prototype in order to provide evidence for the tool support possibility for the first activity *Design*.

In Figure 7.16, the details that are defined for an experiment can be seen. It is mainly the experiment name itself with a further description, which in our case is a link to get further details. Most importantly is the hypothesis, which is in case of a qualitative experiment the research question of how to deliver the value we identified (see section 7.2.1). Furthermore,

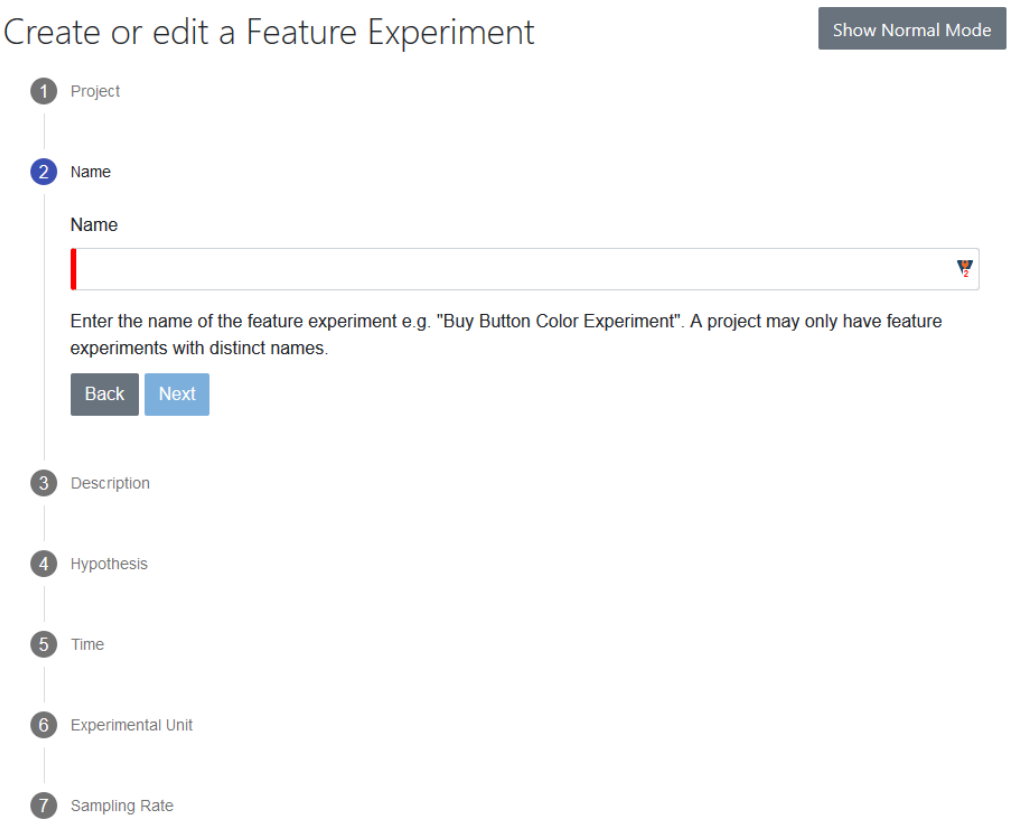



**Fig. 7.15.:** Guided Mode in our prototype *FEXP* to define experiments.

the time period in which the experiment should take place is defined, as well as the experimental unit and sampling rate. The experimental unit is describing the set of entities to be studied like for example all users. The sampling rate specifies how many entities from this set should participate in the experiment.

Furthermore the different variations (in this system called *Feature Variants*) can be defined and controlled if they shall be active in the experiment and which one of them is the control variation or default experience if new variations shall be compared to a design already in use. This definition is used to initialize a technical assignment system that uses feature toggles (see e.g. [@Hod17] for a description of feature toggles). As this was the first prototype tool we developed, we are not using the technical assignment system presented in section 7.3.2, but a simpler version that only supports feature toggles. Feature toggling means enabling or disabling code branches by usually using if else statements with a boolean variable. The system in this case can be asked if for a certain user the corresponding code branch shall be activated and returns a boolean value. Furthermore, boilerplate code for using the system is generated under *Usage* for the corresponding technologies (in this case for Java and TypeScript). This is one disadvantage of feature toggling, that you need native code to make

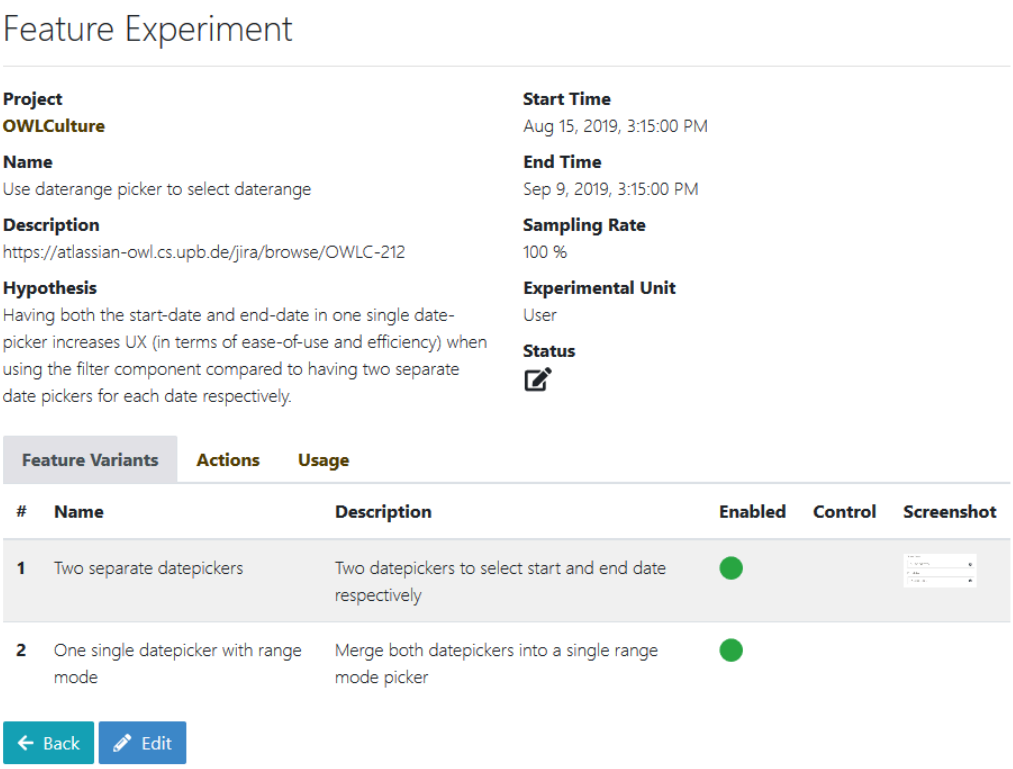



**Fig. 7.16.:** Details Overview of an experiment in our prototype *FEXP*.

it work and if there is not already something written in the technology you want to use, you have to write it yourself.

The quantitative measurement units for an experiment can be defined under *Actions*. Quantitative data that is then collected during an experiment can this way be linked to the defined measuring unit and the experiment itself.

For this system we are not considering qualitative data yet as it can be more vague what data will be collected and it highly depends on the method itself. Furthermore, we have not introduced a system supporting the selection of the right method. To complement the *Experiment Design System* with such a functionality, one could, for example, build on the proposal by Fischer, Strenge, and Nebe [FSN13] for a tool for selecting usability methods.

In order to get an overview of current and past experiments, this system also provides an overview of these experiments (see Figure 7.17). This concludes the definition features of this prototype, which queries the most essential characteristics of experiments according to our definition in section 7.2.1 and Kohavi et al. [Koh+08] overview for online controlled experiments.

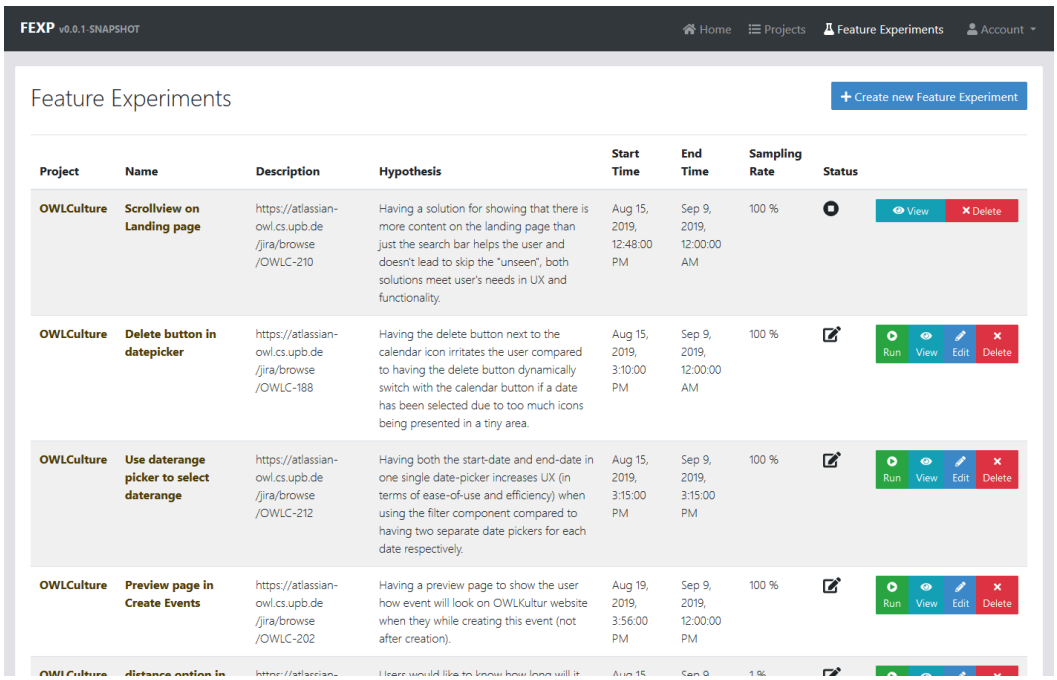



**Fig. 7.17.:** Running and finished experiments overview in our prototype *FEXP*.

For the evaluation regarding the feasibility of this tool, a usability test (as part of the master thesis) with a fictitious bookstore as case was carried out, as well as a case study within the OWL.Culture Portal case study to investigate its use in a real project (see section 8.3.4.2, Work product #9: *Experiment Definitions*). In both cases, the participants were the second group of students (10 in total) of the project group as described in section 8.3.4.

The usability test defines 18 tasks that the participants have to complete and covers all functionalities of the prototype. This includes the creation of an experiment (with and without the guided mode), introducing feature toggling in code, a simulated run (a simulator was written according to the Wizard of Oz method), ramp-up (increasing the sampling rate in an ongoing experiment), as well as the analysis of quantitative data regarding the variations using Kibana as an external data exploration tool. From these 18 tasks, 15 tasks could be solved by all participants. One participant could not define a control variation with two additional variations. Two participants had problems to define the code for feature toggling, which could be more a problem of the used programming language. Not a single participant was able to decide which feature variation differed statistically significant from the control variation. Overall, this shows that the participants were able to use this tool to solve the tasks intended with it and therefore is a first evidence for the viability of the process in this stage.

Since we have no alternative solutions for this prototype, classic A/B tests cannot be carried out to put usability into perspective. Therefore, we decided to use the *System Usability*

*Scale (SUS)* [Bro+96], which offers an easy way to quantify usability on a scale from 0 to 100. Unfortunately, the score on this scale is relative and could be in general lower for a certain product category like enterprise resource planning systems. To put it into relation for the totality of software products, Bangor, Kortum, and Miller [BKM09] developed an adjective rating scale for the system usability scale score to translate it into a descriptive score based on their database of conducted studies with system usability scale. Overall, we got an average *SUS* score of 82.73 for using the system without the guiding mode and a score of 86.36 for using it with the guiding mode. First of all, this shows a slight tendency that the guiding mode could be helpful in defining and using experiments. Furthermore, these scores can be translated into ”excellent” for the guiding mode and ”good” without it according to Bangor, Kortum and Miller [BKM09].

In the case study, the use of this prototpye for real qualitative experiments showed in summary that the participants were able to define usable experiments. For more details, look directly at section 8.3.4.2, Work product #9: *Experiment Definitions*.

In summary, we provided evidence that the use of an *Experiment Design System* can be useful. However, for the exact design of such a system further studies have to be made. Furthermore, we have currently only considered the interaction with other simple systems that have been specially developed for this purpose. Nevertheless, this seems to be feasible in principle and with that we provided first evidence for the viability of the process in this stage.

### 7.3.2 Technical Assignment System

An automated technical assignment has the benefit of being less error prone, an audit log to understand which variation was delivered to which user at what time, and possible automatic fallback e.g. in case of a failing instance or deployment (cf. [@Sev14]). For actually establishing such automated technical assignment, we first have to look at how our different variations / software alternatives are implemented. More precisely, how the separation of the variations was solved in terms of implementation. This can in principle be realized by a physical or logical separation. A physical separation in this context means that a variation is operationally independent from the other variations in an experiment. Accordingly, for a logical separation this means that the individual variations cannot be deployed and addressed independently of each other, so the separation must take place logically within the constituent system. A combination of physical and logical separation is also imaginable.

The advantage of physical separation is the independence of each variation from the other variations. Thus, different versions of frameworks or completely different frameworks can be used for each variation. In addition, the technical assignment can be performed independently

of the technology used to implement the variation as it can be done outside of the constituent system (Keep in mind that each constituent system shall have a standardized communication interface). The disadvantage of this approach is the high resource consumption due to the individual deployment of each variation, even in case of small variations. Furthermore, for a multifactorial experiment layout (see section 7.2.1) a separate system must be set up for each combination of independent variables (mostly sub-objectives in our case), which leads to code redundancy among other things.

On the other hand, the advantage of a logical separation is a low overhead for the implementation, because it is not necessary to set up a separate project, repository, pipeline, etc. for each variation. Furthermore, it is not necessary to deploy a complete constituent system for each variation, especially beneficial is it if there are only small differences. Likewise, multi-factorial experiment layouts can be better implemented, since all combinations are available in one system and there is no need to redirect requests; all that is needed is to enable the corresponding feature branches. Disadvantages are the larger size of the deployment artifact as every variation is shipped, unforeseen reciprocal effects between the variations (e.g. bugs that crash the constituent system), or a higher configuration effort within the code.

Back to the technical assignment, this means that we need two different ways to perform it. First, in the case of logical separation, the assignment is done within the code, which requires a system that returns what variations a given user should receive. In the simplest case, this is a query for each variation, returning a boolean whether or not to activate it. For the physical separation, we need a system that routes the requests to the correct variations. In Figure 7.18, we have illustrated these systems which we call *Orchestration Configuration* and *Orchestrator*.

The *Orchestration Configuration* system is responsible for managing the different assignment rules and to compute the assignment of variations to the user. It can be queried by the *Orchestrator* system or directly by the constituent system holding the variations. In case of the *Orchestrator* system, the *Orchestrator* system would take the information from the *Orchestration Configuration* system and decide how to route the request.

The *Orchestrator* system flexibly combines the physical separated systems to a composition. To be able to do this, they need to be in a network with the *Orchestrator*. Then the *Orchestrator* system can act on the edge of the network as an access point for all incoming requests and route them accordingly. In terms of network terminology, the *Orchestrator* system can be implemented as an edge router or reverse proxy. Simply put, the main difference between these two is that the edge router is stateless and the reverse proxy is stateful. Whereas the edge router just forwards the requests to the corresponding systems, the reverse proxy terminates the connection (not to be confused with closing a connection) from the outside

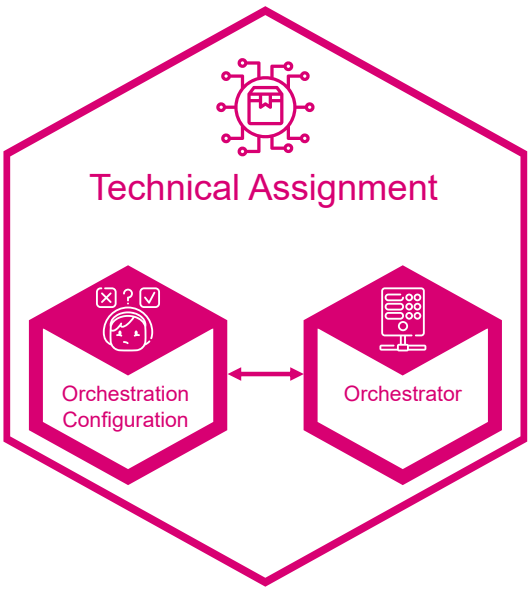



**Fig. 7.18.:** Constituent Systems in the Technical Assignment System.

and opens a new connection to the corresponding variation. This makes it easier for the reverse proxy to assign the same variation to a user over and over again, but also limits its performance compared to the edge router.

In [Abi19], we created a prototype of our technical assignment system with a custom implementation of a reverse proxy as *Orchestrator* system. As for the prototype for the *Experiment Design* system, we did as well a usability test with the participants from the second group of students (10 in total) of the project group as described in section 8.3.4. Additionally, a performance analysis was conducted to better understand the effects of further computations on the routing, especially regarding response time and throughput.

For the usability test, the participants had to work on seven tasks that included the setup of a project, selecting the appropriate technical assignment method (e.g. feature toggling or canary release (ramp-up strategy)), creating assignment rules, deactivating feature variations for a specific user, and deleting feature variations and projects. Although all participants were able to solve all tasks, some of them had problems understanding the terminology used in the system especially regarding the naming of the assignment methods. This is also reflected in the quantitative evaluation of the usability, for which we used the *System Usability Scale (SUS)* as for the *Experiment Design* system prototpye. In total we got a score of 77.88, which reflects to only ”good” according to Bangor, Kortum, and Miller [BKM09].

As for the performance test, we used a middleware based architecture to build up a pipeline in which we could add different middlewares like the routing based on IP, or the identification of users based on an id, and mirroring requests for dark launches. We tested the effects

of the middlewares individually and all together on the throughput and response time on three different setups (1 Node (4 CPU - 4000 Mib Memory), 2 Nodes (each 4 CPU - 4000 Mib Memory), and 2 Nodes (each 7 CPU - 7000 Mib Memory) with each one instance running on it) with the help of Apache JMeter. For the response time, having the system run on two nodes already decreased the time by a factor in the area of 3 to 8.5 for all settings. Increasing the CPU and Memory just had an effect of factor roughly 1 to 1.5. The effect on the throughput is roughly by a factor 2 for doubling the number of nodes and as well again for nearly doubling the resources. Having this big effect on the response time could indicate that our test setting (ramp-up 100 threads in 100 seconds, with each thread making one get and one post request) was bringing the single node on its limitations regarding simultaneous requests in contrast to the two node setups. Therefore, for the comparison of the middlewares we will mainly discuss the differences in the two node setups.

Looking at the response times, the response time for the individual middlewares increased by a factor of 1.5 to 2.5 and by a factor of 2.8 if all middlewares are combined. To set it in comparison, turning off the cache would decrease the response time by 1.9. For the throughput, the decrease for the individual middlewares is by factor 1.5 to 2 and factor 2.9 for all middlewares turned on. Turning the cache off would decrease the throughput by 2.1. To get a feeling for the absolute times, the response time for just the reverse proxy was in the second setup 4059 ms and 3074 ms in the third setup. Turning all middlewares on put the response time to 4736 ms for the second setup and 4632 ms for the third setup. Overall, the performance results are satisfactory and speak for a feasible use in productive environments.

In summary, we provided evidence that a *Technical Assignment* system is feasible and can be useful. The test users in the usability test were able to conduct all tasks intended with the system. Furthermore, the performance analysis showed that this constituent system has an acceptable effect on the response time and throughput. Hence, we delivered with this constituent system additional evidence for the viability of our process. Nevertheless, it was mainly a first prototype that still has potential for a better performance and usability. For further studies, we would recommend to evaluate existing solutions as well like the edge router traefik[22], which was not feasible to use for our purpose at the time of this evaluation.

### 7.3.3 Quantitative Data System

For the *Quantitative Data* system, we look at the data collection from the *Run* activity as well as data preparation and analysis / exploration steps from the *Analyze* activity in regard of quantitative data. The interpretation and planning of next steps involves mainly individual

[22] `https://containo.us/traefik/`

human decisions for which it is difficult to deliver guidance or automation. Accordingly, we propose additional constituent systems for the remaining three steps *Collection*, *Preparation*, and *Analysis* (see Figure 7.19).

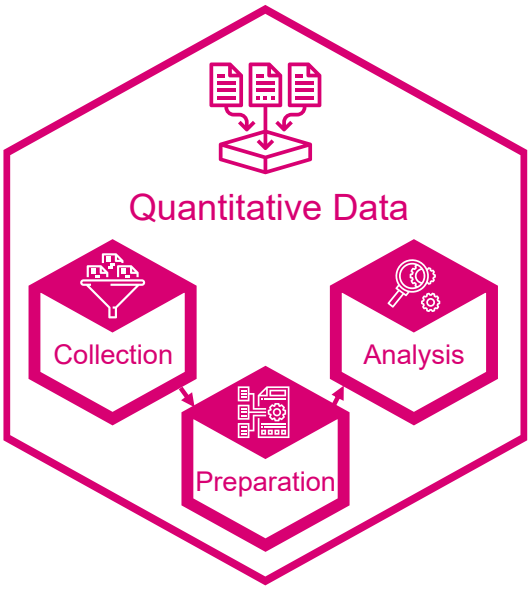



**Fig. 7.19.:** Constituent Systems in the Quantitative Data System.

Quantitative data can be counted, measured, and expressed using numbers, it is highly structured and usually can be broken down into smaller chunks. Some typical examples in the area of software development for quantitative data are survey results, measuring how many clicks a user needed to complete a task, or eye tracking. Of course there are many more potential quantitative data sources that can be used in software development, but we will stick to the mentioned examples to explain how they can be broken down into smaller chunks. A survey for example, can be broken down into events about the answering of the single questions. Measuring clicks is a typical example for the use of UI events to make statements about usability. Eye tracking is nothing more than the regular determination of the position of the eyes in relation to the screen and thus a continuous stream of events informing about the current position. Thus, we assume that quantitative data can usually be broken down into smaller events.

This property, that quantitative data can be represented as a sequence of smaller events is very advantageous for our *Collection* system. For our macro-architecture (see section 7.2.2.2) we have specified that we need event sourcing and thus an event store. Accordingly, we can use the event store, which we need for the macro-architecture anyway, to collect and store the quantitative data. This has the advantage in terms of resource consumption and implementation effort that we do not have to set up an extra system for data collection. A disadvantage is the fact that event stores are usually not optimized for the searchability

of events with complex queries, but for persistence. However, this disadvantage can be compensated for with the *Preparation* system.

The main goal of the *Preparation* system is the aggregation and indexing of the quantitative data as well as making it searchable for the analysis and exploration. The fact that this system is separate from the collection is actually beneficial as it often has high hardware requirements due to indexing and optimizations for searchability. In contrast to the event store, optimized systems that would not run on every target platform can also be used this way. Furthermore, this separation means that data can also be collected without a connection to the *Preparation* system. This data can then later be imported into the *Preparation* system either automatically if a connection can be made (e.g. via internet) or manually after prior checking. Especially the latter point should not be underestimated, since in certain contexts such as production plants experiments could be done, but the data must first be cleared by the respective company, which also has an interest in the system not being directly connected to the internet.

The *Analysis* system builds on top of the *Preparation* system. It mainly includes data visualization and exploration tools as well as statistic analysis tools. Furthermore, process mining or machine learning functionalities could be integrated to identify patterns that can be further investigated.

In [Kra18], we have implemented exemplarily a *Quantitative Data* system to investigate whether this concept is technically feasible and whether the performance is satisfactory. For the *Collection* system we have used the same event store as in our implementation example for the macro-architecture (see section 7.2.2.2) supplemented by a tool to export the events to the *Preparation* system. As *Preparation* system we decided for Elasticsearch[23] because of its http interface which includes as well a sophisticated search query dsl and the better integration with our possible data visualization and exploration tools. Other candidates had been Solr[24], Algolia[25], Spark[26], and a custom implementation. For this exemplary implementation, the *Analysis* system only consists of a data visualization and exploration tool as it is the most crucial part in data analysis and the further tools are more specific to the context of the quantitative data. Possible candidates for the data exploration and visualization were Kibana[27], Grafana[28], Redash[29], and a custom implementation. Even if the nuances between these individual tools are only very small, in the end we chose Kibana because of its better integration with Elasticsearch. With all these decisions, it must be

[23] `https://www.elastic.co/elasticsearch`
[24] `https://lucene.apache.org/solr/`
[25] `https://www.algolia.com/`
[26] `https://spark.apache.org/`
[27] `https://www.elastic.co/kibana`
[28] `https://grafana.com/`
[29] `https://redash.io/`

kept in mind that the system selection was made with the aim of achieving a quick proof of concept. Accordingly, a different set of systems may be more appropriate depending on the context.

For the performance analysis, it was important to us that this *System of Systems* can handle a realistic amount of events. Therefore, we have chosen a very successful experiment of Henze, Rukzio, and Boll [HRB11] as benchmark for our exemplary implementation. Henze, Rukzio, and Boll [HRB11] have developed a game as a mobile app to investigate the accuracy and error rate of touch input regarding target size and screen location. Over three months, they have collected 120,626,225 touch events from 91,731 installations, which translates to roughly 15 events per second on average. In our performance test, on a laptop with an Intel Core i7 2.7 Ghz and 16 GB memory, the event store was the bottleneck with a processing capacity of about 5,000 events per second. Elasticsearch was able to index 16,000 events per second and Kibana was able to render around 120 million documents in 1535ms, which includes the query to Elasticsearch. Looking at the system in its entirety, it is able to store, index, and visualize 150,000 events in 46.5 seconds. This performance is more than sufficient for handling what was generated during the experiment of Heinze, Rukzio, and Boll as well as for most cases. To put in comparison, Urban, Sreenivasan, and Kannan [@USK16] stated in 2016 that Netflix as a whole platform is handling 150,000 to 450,000 requests per second.

In summary, our idea of a *Quantitative Data* system is technically feasible and has a sufficient performance for most use cases. Even for higher demands, it could be researched to what extent a horizontal scaling could fulfil the demands. Although, we measured same strange behavior of event store regarding the performance which we could not completely explain and only eliminate by tweaking the configuration which could contradict this goal. This is also the last evidence we provide for the viability of our process.

## 7.4 Summary and Discussion

In this chapter, we have presented and discussed the last stage of our approach ICeDD. The main goal of this stage is to extend Design Thinking into software development. To be precisely, we wanted to achieve that the converge phase of Design Thinking didn't end with only one solution alternative in the previous stage but with at least two so that we can actually implement them in this stage as software and compare them in the actual context of use. As mentioned several times (e.g. in section 1.1), this use of software alternatives in the actual context of use is important to truly uncover all constraints and interacting dependencies. If the software alternatives are not tested in actual usage, it is

always restricted to the thought model of the researcher or certain knowledge of the user is not activated (cf. section 4.1). Furthermore, if we are not using at least two alternatives, we cannot compare them and conclude from the difference what is actually delivering value and how. Therefore, experimenting is of utmost importance for this stage.

In section 7.1 we discuss the general requirements for conducting experiments with software and what this means for the software development process. A key finding of this is the need to adapt the software development process to field studies / field experiments, which in the end affects the 4Ps (People, Process, Product, and Project) as introduced by Jacobson, Booch, and Rumbaugh [JBR99]. We use this structure of 4Ps in section 7.2 to discuss in detail how the software development process in this stage has to be adapted.

The starting point is the process in section 7.2.1 which we define based on the general steps *Design*, *Build*, *Run*, and *Analyze* for conducting experiments. Whereas *Design*, *Run*, and *Analyze* are more related to experiments in general, *Build* is actually an adaption of agile software development to implement different alternatives at once and preparing data as well as infrastructure for the experiments. We further derive the necessary product properties from this process in section 7.2.2 to not contradict the process but supporting it.

The main finding for the product is that we need an evolutionary software architecture that, in our can case, shall be realized as a *System of Systems* architecture. In our case, the *System of Systems* architecture has the advantage that it is based on independent subsystems, which can be technologically independent of each other. On the one hand, this allows us to let systems compete against each other, as is the case in nature with evolution, and on the other hand, technological decisions from the past limit us only to a limited extent in what we want to try out. To realize this, we need to distinguish between macro- and micro-architecture, whereas the macro-architecture describes how the systems interact with other systems and the micro-architecture is related to the internal architecture of a system. Accordingly, for the realization of a *System of Systems* architecture the macro-architecture is more important. We discuss its realization on the basis of existing patterns from other areas in section 7.2.2.1 and give an example for a concrete implementation in section 7.2.2.2.

Last but not least, we discuss the implications for the people in section 7.2.3 and for the project in section 7.2.4. The most important implication is that the value designer shall be primarily responsible for the design of the qualitative experiments and needs the related empirical knowledge to not overstress the developers.

To provide evidence for the viability of our process for this stage, we are discussing a potential tool suite in section 7.3 as ”tools are good at automating repetitive tasks, keeping things structured, managing large amounts of information, and guiding you along a particular development path” [JBR99, pp.22]. Furthermore, we discuss three tools *Experiment Design*

*System*, *Technical Assignment System*, and *Quantitative Data System* in detail and present our first evaluations of their usability and feasibility. In general, they proved to be feasible and with that provided evidence for the overall viability of the process.

In the requirements in section 7.1, we have defined that from our FF especially *Alternatives*, *Operating Alternatives*, *Consequences of Technological Decisions*, and *Learning Cycle* are relevant. We fully and completely fulfil *Alternatives* as we are actually creating at least two solutions simultaneously in parallel based on the requirements specification / documentation from the previous stage in software. Furthermore, we can operate these alternative independently as with the *System of Systems* macro-architecture we proposed, we are getting an component based deployment, can configure and deploy it automatically with a CI/CD pipeline and due to the *Technical Assignment System*, we have an online fallback mechanism as well as an automatic user specific online orchestration.

The *Consequences of Technological Decisions* are kept at a minimum as well because of our *System of Systems* macro-architecture. With this macro-architecture, we allow to freely use programming languages and other technologies on the micro-architecture level of a constituent system and achieve therefore Polyglotism. Furthermore, we introduced with MVVM a strong independence from UI and Model Layer. By using event sourcing in conjunction with CQRS, we allow parallel models and furthermore the decomposition into a suite of small services, that run their own processes, are independently deployable and have a bounded context.

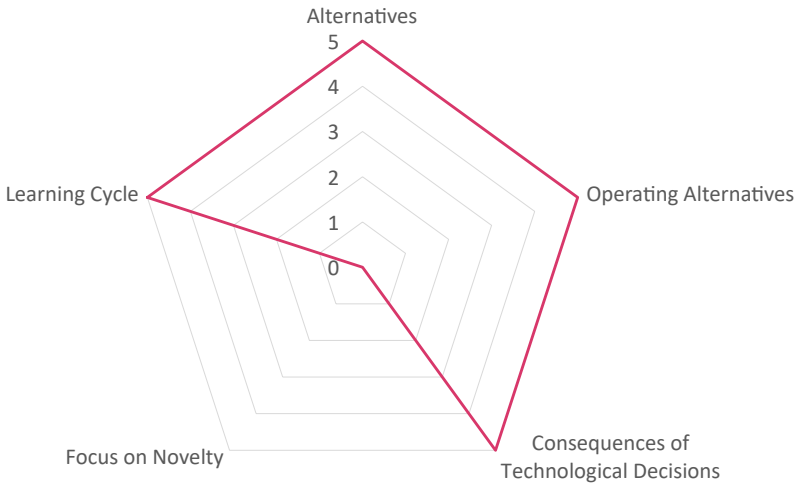



**Fig. 7.20.:** Radar Chart for Stage 4 regarding our *Fitness Function* (FF).

Last but not least, with the explicit definition of an experimentation process we get an explicitly defined learning cycle that refers to several alternatives at a time. As we are still allowing to select the best fitting methods in the *Design* activity, we further enable this

learning cycle to adapt it according to the context. Hence, we also fulfill this characteristic to the fullest. How this is reflected in our FF is presented in Figure 7.20.

The most important thing about this chapter was to show that experimenting with software is possible. In section 1.1.3, we pointed out possible issues that could inhibit experimentation with software. These have been the foundation for tackling these on a conceptual level in section 7.2. In order to show that our concept is not only feasible but also viable, we have presented and discussed tools in section 7.3. Overall, we have delivered evidence that the issues mentioned in section 1.1.3 can be challenged even if they are primarily on an argumentative level. Nevertheless, the concepts, especially for the macro-architecture, are largely based on already established and widely used patterns, but up to now not so specific for experimentation. Furthermore, the tools have been evaluated regarding their feasibility and usability. This is why this should still be a good foundation for further research. In further studies, especially the effectiveness and efficiency has to be researched in more detail as goal in this thesis is primarily to look at the feasibility. Hence, we can not be sure that this stage will deliver the intended results although having good arguments for that.

# Part III

Evaluation & Epilog

# 8 Evaluation

In this chapter we are summarizing the challenges and results from our general research approach in section 8.1. Furthermore, as we already mentioned in chapter 3, we conducted an application case study for the assumptions we made based on the assertions of the Diffusion of Innovations theory (see section 2.1) and Design Thinking (see section 2.2) which we present in section 8.2. Moreover, we are presenting and discussing a feasibility case study to give evidence on the feasibility of the last three stages as well as their ability to interact with another in section 8.3.

## 8.1 History in Paderborn App

This case study is actually a summary of challenges and results from our general research approach that we described in section 1.3. In the sense of the *Case Study Research Cycle* (see Figure 8.1) of Yin [Yin17] it is not a case study in which the individual phases *Plan, Design, Prepare, Collect, Analyze and Share* are run through once, but repeated times. What stays the same over all iteration is the main case with the *History in Paderborn App* (cf. [Gre+16]) and some evaluation instruments.

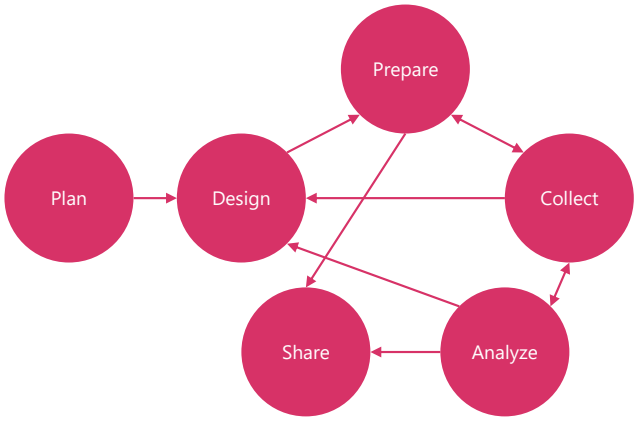



**Fig. 8.1.:** Case Study Research Cycle. Own Representation based on [Yin17]

As a short recap, the *History in Paderborn App* has the aim to make the history of Paderborn accessible in an appealing way via an App. The unique character of this is the vision of a multi-modal, spatial-bound communication of historical knowledge. For this we have

worked together with researchers from the humanities in the fields of history, art history, german medieval studies, and german linguistics. In other words, we could describe the *History in Paderborn App* as a new form of scientific communication with the general public. This makes this case very suitable for our approach as the *History in Paderborn App* represents a unique and novel software-based solution.

The general design of this case study, including the participants and some evaluation instruments, haven been described in section 1.3 and in [SOF18]. As a small summary, the participants are students of the Master's program in Computer Science at the Paderborn University, who participate in a student project group over two semesters (30 ECTS till 2017, 20 ECTS afterwards). As general evaluation tools we use retrospective meetings, student discussions, task forces, steering meetings, *Design Thinking* workshops, and individual discussions between the students and the organizer that take place every semester.

The aim of this case study was to gradually develop our software development approach for unique and novel software solutions over several iterations in an environment as real as possible. Starting point for this was the status of software development methods in 2015 (cf. e.g. Lindgren and Münch [LM15]). From this, our approach was gradually developed, with challenges, limitations, and requirements emerging each iteration. Each iteration, a newly developed part of the approach was used and therefore each iteration new challenges, limitations, and requirements could emerge. This is a common approach in *Design Thinking*, *Grounded Theory*, or *Action Research*, for example, to develop and refine a theory (in our case how a software development approach for unique and novel solutions shall look like) iteratively. Hence, each term started a new iteration with a new goal (see Table 8.1).

| **Term** | **Goal** |
|---|---|
| Winter 2014/2015 | Beginning of the project group. How to do agile software development with scrum in a team? |
| Summer 2015 | First overlapping project group. How to transfer the knowledge of the existing students to the new ones? |
| Winter 2015/2016 | Increasing number of students (>10 Students). How to scale up to two teams? How to use a Continuous Integration Pipeline? |
| Summer 2016 | Challenges from framework usage over a longer period of time. How to harmonize technologies? |
| Winter 2016/2017 | Increasing complexity of implemented system. How to break down the system into microservices? |
| Summer 2017 | From release driven to everything is a protoype. How to do continuous delivery and incorporate quality assurance? |
| Winter 2017/2018 | Synchronization and dependency challenges of services. How to incorporate *Event Sourcing* and *CQRS* to make systems more independent? |

| | |
|---|---|
| Summer 2018 | Increasing complexity of operated services. How to operate and maintain a system of software systems? |

**Tab. 8.1.:** Project group main goals for each term with the corresponding research question

In addition to the main objectives of each semester, further topics were dealt with within the scope of a seminar. This brought to light not only new challenges but also solutions for these. An overview of the seminar topics is given in Table 8.2. The thing is that the topics are not necessarily aligned with the term goals. In some cases, further new findings have emerged on an old term goal or the previous solutions have not yet proved suitable. In other cases, the students showed vision and dealt with problems and solutions that were relevant for the project group but could not be implemented yet because other things were missing or not ready yet (e.g. continuous delivery in may 2015 that could not be fully implemented till summer 2017).

| **Term** | **Seminar Topics** |
|---|---|
| February 2015 | Knowledge Management |
| | Scrum & DevOps |
| | Software Testing (Testing and Agile Testing) |
| | AngularJS & Play!Framework |
| | Reactive Design |
| | Android Development |
| s May 2015 | Scaled Agile Development |
| | Logging / Analytics |
| | Location based services (e. g. indoor/outdoor navigation, beacon, GPS) |
| | DevOps / Continuous Delivery Pattern (e. g. Dark launches, Branching in Code) |
| | Augmented Reality |
| | Design Guidelines & Usability |
| | Requirements Engineering (Elicitation, Specification) |
| September 2015 | Containerization (Docker) |
| | Infrastructure as Code (Puppet and Chef) |
| | Resilience with DevOps |
| | Change Management |
| | Microservices |
| | Reactive Design |

| | |
|---|---|
| August 2016 | Usability Evaluation Process |
| | User Onboarding in Mobile Apps |
| | Animations |
| | Implementation of Augmented Reality in Xamarin |
| | Quality Assurance in Continuous Delivery |
| | Operating a system of systems (Docker Cloud) |
| | Architecture paradigms in Android and iOS |
| March 2017 | Realization of Quality Assurance environments for HiP-App (Mobile) |
| | Concepts for CMS Quality Assurance environments (Web) |
| | Concepts of different avatars based on GenderMag method |
| | Web technologies for visualizing and interacting with Geo-Data |
| | Indoor navigation utilizing BLE beacons |
| | Angular Universal & Augury |
| | Indexing and Searching with Elasticsearch |
| | User Onboarding in Web Applications |
| August 2017 | Event Sourcing & CQRS |
| | Recommender systems |
| | Machine learning techniques for Gamification |
| | Accessibility for users with special abilities for HiP-App |
| | Improvements of usability based on UI/UX |
| | Monitoring & Service Discovery |
| | Grow effective software development team |
| | Improving the exploration experience - new modes, quests, and reward systems |
| | Geofences for Gamifcation and Exploration |
| March 2018 | Building interactive voice interfaces: An overview |
| | Automatic Client Generation from a Swagger Specification |
| | Social Media in the HiP-App |
| | Test strategies for Microservices |
| | Progressive Web Applications |
| | Visual sight detection using the Google Cloud Vision API |
| | Tracking user engagement and app usage by means of Analytics service |
| | Centralized Logging and Monitoring |

| | |
|---|---|
| August 2018 | Security Threats and best practices in Angular to avoid them |
| | Navigational structure and resulting user interaction of a Web App |
| | Best Practice for Documentation |
| | API Gateway & Circuit Breaker Patterns |
| | Peer-to-Peer Services on Mobile Phone using the internet |
| | Stronger Web Authentication |
| | Assistive Technology for people with vision defects |
| | Real-time web functionality with SignalR |

**Tab. 8.2.:** Seminar topics

In sense of the three quality criterion objectivity, reliability, and validity, this case study has its focus strongly on validity and not very much on the reliability of the results in the sense of repeatability. The insights generated each iteration do not serve to be taken as general truth, but as a basis for a further literature research. We assume that in most cases research has already taken place. This can be exactly in the discipline of *Software Engineering* or in similar adjacent disciplines like psychology. This approach is in essence a systematic literature search based on the principles of *Action Research*. Therefore, we need a high validity for a successful implementation to ensure that the corresponding effects can come to light.

This case study is not intended to demonstrate the feasibility of the overall approach. The case study on the OWL.Culture-Platform serves that purpose (see section 8.3). The main purpose of this case study is to explain the basis on which certain concepts were integrated into our approach. For this purpose, we present in the following insights that resulted from the implementation of this case study to highlight how practical experiences influenced our approach.

We started the project with an agile software development approach with a team in winter term 2014/2015. As development method we decided for the widespread Scrum. One of the students took over the role of Scrum Master. The Product Owners were represented by the Humanities. The advisors of the project group had an advisory and light steering role. Mock-ups were created for the software to be developed and discussed with the product owners.

After using Scrum for several months, one thing stood out. The official Scrum Guide [@SS17] as well as other Sources like from Meyer [Mey14] hide very well the complexity and challenges behind the product owner role. The product owner has the responsibility to clearly express product backlog items, prioritize them, and optimizing the value of the work the development team performs. Although this sounds simple, we quickly got into a situation

where we had to rework and throw away more often than we accepted features, even though the product owners in this case were the actual customers who should have known best what functionality was needed. This made us think if something with the role or with our context was odd, which is why we did a literature research on people specific characteristics that could have an influence on this as well special characteristics that related to the context. Thanks to the author's psychological knowledge, the first one quickly led to the findings regarding experts and tacit knowledge (see section 7.1). The second has led to discussions about the properties of digital humanities (cf. [Bur+12] and with that to the questioning of whether agile software development is really that agile, which led to discussions about hybrid software development methods (cf. [Kuh+17]) as well as alternative agile definitions for software development (see [Ado06]). Especially the latter led to the rethinking of whether agility has to be considered depending on the context and thus to a search in project management with the discovery of the Cynefin framework (cf. section 1.1.1 or [KS03]) as a classification help. For the product owner, these findings meant above all, that in certain contexts he can only predict the value of the items in the product backlog to a limited extent (cf. [Koh+09]) and that this value can only be realistically determined by a realization in software.

It was precisely this necessity for implementation in software that was very unsatisfactory for all those involved. For example, a useful requirement for the *HiP-App* was that it should support communication for collaborative content creation. In order to realize this, it was agreed to implement a chat system that was also tested for usability during development. Only conversations with users in real usage revealed that they prefer other established communication channels. This made the chat system obsolete and also the immense energy that the developers put into it, since software development is comparatively expensive (cf. [Ste+04]). When most of the things you do turn out to be useless this is very demotivating. For this reason we researched if there are possibilities to obtain these findings in a better cost-benefit ratio and why usability engineering did not help here. This lead us to the insights from Frohlich and Sarvas [FS11] as well as Norman and Verganti [NV14] that Usability Engineering is mainly made for incremental rather than radical innovations. Further research into methods for radical innovations led to concurrent set-based engineering (cf. [War+95]) as foundation for lean development (cf. [Rie11]) and *Design Thinking* (see section 2.2) due to its proximity to existing usability engineering / user experience methods.

As stated in the work of Lindberg et al. [LMW11], it was not clear at the time how *Design Thinking* could be integrated best into the software development process. Therefore, we have tested several iterations with variants from a two-day workshop, a one-week workshop to a semester-accompanying (integrative, decoupled) workshop. The first two are more the mentioned *Front–End Technique* for integrating *Design Thinking* and the later one the *Integrated Development Philosophy* (see [LMW11]). The decoupled semester-accompanying

version is the one presented in the OWL.Cultur-Platform (see section 8.3 and section 5.2) which is the combination of *Front–End Technique* and *Integrated Development Philosophy*. The disadvantages of the Front-End Technique in our case were that we were able to build up initial knowledge in the combination of computer science students and humanities students, but did not come up with any concretely conceivable software products. A realization in software would be unthinkable in this short time period. Longer formats that were running integratively had the disadvantage that the participants brought different skills with them and were more involved in certain parts and less so in others. We observed the two-day workshop as adequate to avoid the conflicts arising from the different contributions. Nevertheless, this variant lacks the further learning through prototyping until it has become a usable software product, which is why we have added a phase to this two-day workshop in which the developers continue *Design Thinking* on their own and get feedback from outside on their prototypes at given times.

Putting these organizational ideas into practice, however, also leads to effects on software development, especially through the need for evolution and experimentation. After a year of working on the software, we found that it became difficult to modify it, although we used modularity within the software. Technological decisions that seemed right in the past suddenly prevent useful technological implementations in the present. It became increasingly difficult to adapt the database schema for the individual modules. The original framework did not really adhere to semantic versioning, and minor changes often meant that the entire application had to be adapted all at once in order to keep it executable and compilable. This resulted in significantly less resources being available for own experiments. Furthermore, CMS and mobile app, each of which was the responsibility of a separate team, have developed into independent entities, although they share many commonalities. We have used these experiences from the first year of software development as a basis for further literature research regarding evolvability and experiments with software. The two main results of this were Conway's Law and microservices.

Conway's Law actually states that "organizations which design systems [...] are constrained to produce designs which are copies of the communication structures of these organizations" [Con68]. Kwan, Cataldo, and Damian [CKD12] revisited it for software development including evidence from software development research and came to the conclusion that in software development as well the tasks and the communication structure have a bigger influence on the architecture than the proposed architecture design. This has encouraged us to harmonize technologies in order to be more flexible in the design of tasks, e.g. by simplifying the relocation of development capacities and the possibility, especially with regard to bounded contexts, to draw the system boundaries differently than along technologies. In addition, the experience we have gained with the technology that has been in use to date has been taken into account in the selection for harmonization. Accordingly, we

have added stability as an important criterion for us when selecting the technology for evolutionary systems. The more you can rely on the reliability of your foundation, the better you can conduct your own experiments without having to constantly react to changes in the foundation.

Microservices were a relatively new term around 2015 for which conferences, online talks, and podcasts were often the best sources for the newest information [Thö15]. At that time, Thönes [Thö15] defined the term microservice as ”a small application that can be deployed independently, scaled independently, and tested independently and that has a single responsibility”, which is actually what we were looking for. Dragoni, et al. [Dra+18] defined as basic principles for microservices *Bounded Context*, *Size*, and *Independency*, which can be summed up to small services with a loose coupling and high cohesion that combine related functionalities into a single business capability. Furthermore, they as well as Lewis and Fowler [@LF14] point out the importance of infrastructure automation including continuous integration and continuous delivery/deployment to develop microservices. Consequently, our further research in this area went in the direction of how to create bounded context in order to implement small services that can operate independently of each other and are supported by an appropriate infrastructure automation.

Regarding the bounded contexts, we first started by creating two bounded contexts for content creation and consumption in the app. As we had domain objects like user that were shared between these two bounded contexts and the consumption in the app had slightly different requirements (e.g. scalability or need for relational data) on the database than the content creation, we decided for a separation with CQRS. However, the use of a relational database directly with a more document-oriented one proved to be problematic with regard to synchronization. Which is why we further investigated into this challenge and found out about *Event Sourcing*, that we started to implement in August 2017. But we had difficulties with this because the participants had problems with the granularity of events and treated them rather as objects (e.g. EventChanged with all properties of an Event instead of e.g. EventLocationChanged). It has nevertheless proven to be very beneficial for parallel models.

With the Bounded Contexts also came the creation of smaller independently deployable services. While in the beginning these services could be maintained and operated manually, the need to monitor and configure them automatically quickly arose. Hence, the change from Docker alone to a Kubernetes infrastructure that unifies and simplifies configuration, monitoring and distribution was made in September 2018. At that time, the services were manually deployed to the specific virtual machine via docker and a virtual machine with nginx as reverse proxy and manual configuration served as the central access point to these

services. Not only was the manual configuration very error-prone, monitoring was much more complex to implement due to a lack of standardization.

Although much more has been learned in this process and more than enough empirical material can be discussed for further analysis, the aim of this section is primarily to give an impression of how the research process described in section 1.3 has influenced the procedure and the decision-making process. In the following we would like to discuss this research process, which is oriented towards external validity and qualitative instruments.

All in all, we found this kind of research process very helpful in which gaps are approached from the current state of the art with a practical project. Through the actual use of methods and patterns, the complexity behind supposedly simple formulations such as the task of the product owner becomes clear and tangible. In addition, the interaction of different areas such as requirements analysis, operation, or development can be observed at an early stage and be better understood through targeted intervention. However, the gradual development of a solution in this way requires a correspondingly long time, especially until one has reached the edge of the current state of research/technology. In this case from December 2014 to September 2018, i.e. around three and a half years. In addition, most of the interim results are the results of qualitative methods, which are of course highly vulnerable in terms of their generalizability and repeatability and can be correspondingly difficult to publish scientifically. Even in the sciences, from which many of these procedures come, their use is strongly discussed (cf. [Hel11, p.9], [MM10]). Accordingly, it may be more difficult to publish and discuss interim results in fields that have used little or no qualitative methods so far. Moreover, this type of research requires at the end a lot of additional effort (interviews, planning and implementation of the individual learning processes, clarification of the observed effects with the state of research, etc.) for a method developed compared to approaches in which methods based on theoretical consideration are developed. For that, the result of such a process is at the end a developed method with an arguably high external validity.

## 8.2 Application Case Study regarding Innovation Assumptions

An integral part of our approach is to prefer early testing with prototypes prior to specification and development of a final system. On the one hand, this is due to the fact that certain effects, if they were previously unknown or unconsidered, can only occur if the system is used or tested in practice (cf. section 5.2). On the other hand, because of the way we humans adopt innovations. According to Rogers (see section 2.1 or [Rog10]), in addition to things such as

relative advantage or compatibility, factors like complexity, testability, and observability are to be mentioned. The latter three factors can be influenced by prototypes and thus also the tendency of the participants to judge the solution more realistically and not to reject it directly due to uncertainties. This case study examines whether innovation theory can be applied to software development and whether prototypes tested in a realistic environment have an effect on the evaluation of a solution.

For this we are using the case of IT support in the practical training of fire brigades, which we also described in [SFS14]. At the time of the study, this was still largely unexplored, in contrast to IT support in the training of managers or the use of IT in the field (see [MK13]). In addition, it was characterized by a high number of manual activities and tactical procedures, which must be practiced intensively by the voluntary fire brigade and beginners in order to guarantee safe handling during operation. Hence, it was a good starting point for unique and novel software-based solutions.

### 8.2.1 Concept and Conduction

The case study itself is structured in three parts: requirements analysis, implementation of a prototype and a subsequent usability test in which the prototype is used in as realistic a setting as possible. The idea behind this structuring is to collect ideas for problems and solutions in the requirements analysis with classical tools and to discuss a promising idea with stakeholders at the concept level already at the end of this phase. This results in a picture of how this idea is accepted without a tangible prototype. The prototype is then implemented based on the requirements from the requirements analysis and handed over to the stakeholders in a usability test in a realistic setting for testing. The attitude of the stakeholders to the idea is collected as well in the usability test. From this it can then be deduced whether and what effect the test of a prototype in a realistic setting has on the evaluation of a solution by the stakeholders and whether the innovation theory can be applied to this setting.

The requirements analysis is divided into a document and system analysis, a user observation of an exercise and an analysis of the stakeholders. For the document and system analysis legal documents about the structure of the fire brigade system and the training, requirement documents for software systems in the fire brigade system, as well as similar systems on the market were analyzed. From this a meaningful subdivision in preparation, execution and debriefing of exercises has crystallized as well as a basic understanding of the structure and tasks in a fire brigade especially for training.

In the next step, a structured user observation of an exercise was carried out in order to gain insights into the practical training operations. The findings serve to identify problems as well

as potentials in practical training operations with regard to IT support. For the preparation of the structured user observation, findings from a preliminary survey of the Dortmund fire brigade were used in addition to the findings from the document and system analysis. This preliminary survey included information on the stakeholders to be observed, including their work tasks and interests/objectives, the structure of the environment and presence of technology (e.g. surveillance cameras), number and positioning of observers, and necessary documentation forms.

The scenario of the observed exercise was a fire caused by handicrafts in an apartment house in which missing persons are suspected. According to the training concept of the Dortmund Fire Department, each year there is a training focal point which must be completed by each fire station with an operation exercise. In the year of the observation it was fire fighting with the mentioned scenario. The Dortmund Fire Department has a special training facility for the training, which also includes a fire training house (see FigureFigure 8.2a). This fire training houses purpose is to simulate apartment house fires as realistic as possible, which is why it was designed to work in a controlled manner with real fire and smoke.

**(a)** Fire Training House **(b)** Control Desk

**Fig. 8.2.:** Fire Training House including Control Desk of the Dortmund Fire Department.

For the exercise several observers positioned themselves around the fire training house and close to important stakeholders like the squad leader and made notes. Due to the high heat and smoke it was not possible to enter the fire training house without breathing protection. However, in the control room (see FigureFigure 8.2b) there was the possibility to follow all radio traffic as well as the proceeding troops by video surveillance and the thermal imaging camera of the accompanying instructors.

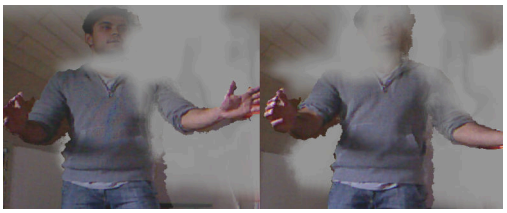
A lot of effort is put into carrying out such an exercise. Not only is a special building needed where the conditions are as realistic as possible, but several people are needed to observe the exercise and discuss the units' procedures in the debriefing session and point out possible mistakes. From the document analysis and for this reason we decided for an Augmented

Reality smoke simulation (see FigureFigure 8.3b) as IT support. The idea behind it is that AR allows to train under realistic conditions on site and to record the exercise accordingly (e.g. walking routes).

Next, we had expert interviews with firefighters from volunteer and professional brigades who are responsible for training at site level. A total of seven people were interviewed about the actual and desired state of the practical training. As part of the desired state, the concept of an augmented reality smoke simulation was presented orally and the interviewees were asked for their assessment. Subsequently, the interviews were transcribed, extracted, processed, and then evaluated with regard to the research question according to Gläser [GL09].

We used these results from the document and system analysis, the user observation and the expert interviews for the AR smoke simulation requirements and implemented a system accordingly. The core of this system consists of a smoke gas simulation (see FigureFigure 8.3b), which is faded in according to the depth information of a Kinect and the position in the room. To increase immersion, a custom Video-See-Through Mount (see FigureFigure 8.3c) was designed to restrict the participants' view only to the mount with a sheet (see FigureFigure 8.3a).

(a) Experimental Setting



(b) Augmented Reality Smoke Simulation

(c) Custom Video-See-Through Mount

**Fig. 8.3.:** Prototype Test for Room Smoked in Augmented Reality.

Furthermore, to make it more realistic not only the execution of the training with the augmented reality smoke simulation was implemented but also the other phases (see Figure 8.4). For the preparation, the task was to setup a positioning system based on wifi tracking (see Figure 8.4b) and to define the room that shall be smoked (see Figure 8.4c). The debriefing view includes a map with the visualization of the walking paths of the proceeding troop (see Figure 8.4d).


(a) Firefighter Training System

(b) Prepare Tracking

(c) Defining Smoke Border

(d) Training Debriefing

**Fig. 8.4.:** Prototype: Firefighter Training System - Fire Operation

This prototype was then tested in a usability test with five persons, whereas four persons had been members of voluntary fire brigades and one person of the Federal Agency for Technical Relief (THW). As already mentioned, the aim of the usability test was not to uncover concrete usability problems, but rather to make the techniques comprehensible and testable for the participants in a concrete scenario. Therefore, the participants have to go through all phases. After each phase, they had to answer a questionnaire regarding the assessment of the prototype and the INTUI questionnaire [UD10] for measuring the intuitiveness / immersion. In addition, a schematic representation of a better integrated variant of the demonstrator (see Figure 8.5) was shown after the implementation phase in order to serve both as a comparison to the early development stage of the prototype and as a stimulus for the imagination.

### 8.2.2 Results and Discussion

In the expert interviews, all interview participants expressed ideas for the desired state. Most participants responded with the idea of a (partially) automated recording of the exercise. In addition, the desire for a central platform for teaching content and professional exchange was expressed several times. There were also requests for animations and a promotion of

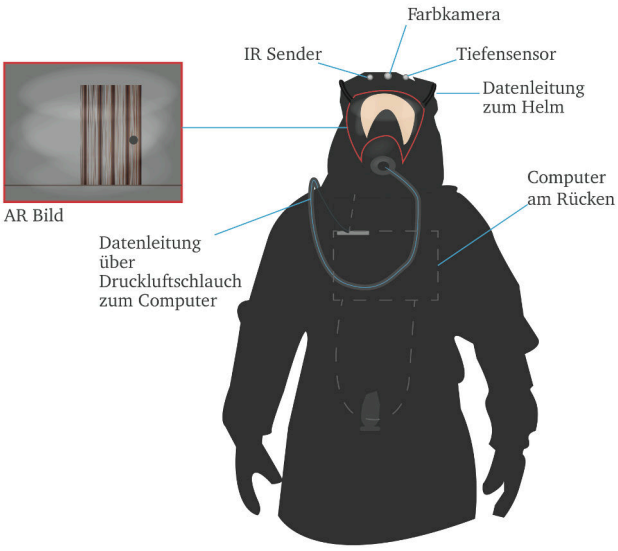



**Fig. 8.5.:** Schematic representation for a more realistic integration of the prototype into the protective clothing.

realism, and even augmented reality was once mentioned directly, although it was not yet known to the general public at that time. Another frequently mentioned wish is to have the same possibilities mobile as in a classroom. Almost all ideas were described from the user's point of view and did not yet aim at concrete technical solutions.

The direct addressing and discussion of Augmented Reality as a concrete solution, however, has proven to be difficult. Participants couldn't really imagine what this technology is about and what advantages and disadvantages it has. They struggled as well with picturing possible use cases. In summary, the participants had problems recognizing the relative advantage and classifying the implications of this technology, which is why they tended to oppose this technology.

In the usability test, the participants have on average more positive mentions than negative mentions in the individual phases (between 1-2 more mentions in an average of 4 mentions). For the augmented reality part it was on average two more positive mentions and the negatives were mainly limited to the representation of the smoke. The results of the INTUI questionnaire for this phase also suggest that the participants experienced a high level of immersion or intuitivity. The values for verbalization ability, magic experience, and ease are all above 5 on a scale of 1 to 7. Only the gut feeling is evaluated neutrally. This indicates that the participants were able to familiarize themselves well with the technology.

This is also supported by the fact that the participants noted further possible applications. It starts with concepts for normal operations and ends with suggestions for a more realistic training situation. Here, the effect of the layer formation of the smoke within a room fire was

called above all. In addition, a stronger inclusion of the other senses, above all the sense of hearing and touch, was wished for the best possible closeness to reality. Advantages are seen in the small preparation efforts, the missing dependence on the location, and the possibility to interrupt an exercise immediately.

Whereas in the expert interviews only one person was positive towards augmented reality, after trying out the prototypes in the usability test all participants had a positive attitude towards augmented reality. Hence, in our case study the test of a prototype in a realistic environment had an effect on the evaluation of a solution. We, therefore, regard the applicability of the innovation theory as given. This shows that it is advantageous for a more realistic assessment of solutions if they can be tested as early as possible under realistic conditions. Such an assumption is also supported by the Cynefin Framework (cf. section 1.1.1), Design Thinking (cf. section 2.2) and the theories of expert knowledge (cf. section 4.1). For our solution concept this means, that it is not only favorable to test with prototypes but it could also be advantageous

## 8.3 OWL.Culture-Platform

In this case study, the interaction of the individual stages (cf. Figure 8.6) is studied. Furthermore, it is investigated how developers with little or no previous knowledge of this kind of approach are able to work with it. For this purpose, the individual stages of the approach are passed through by a student project group (mainly stage three to five) within one year. In the following we will describe our case (section 8.3.1) as well as the general concept of the case study (section 8.3.2). Furthermore, we are presenting the evaluation instruments we are using for the case study including the expected results in section 8.3.3. As evaluation instruments we are using a questionnaire (section 8.3.3.1) and analysis of the individual work products created (section 8.3.3.2). The conduction of the case study including the results from each evaluation instrument are presented and discussed in section 8.3.4. In section 8.3.5 this section is summarized and discussed.

### 8.3.1 Context

Our case in this case study is the *OWL.Culture-Platform*. The goal of the *OWL.Culture-Platform* is to bundle the *Ostwestfalen-Lippe* (OWL) region's cultural offerings and make them more visible and usable in the future. OWL as a region consists of five counties and one county free city, each of which operates its own event site for cultural offerings. The resulting borders make cultural exchange beyond the city and county borders more difficult.


**Fig. 8.6.:** Solution Overview

Furthermore, existing systems such as events on Facebook are also strongly oriented towards administrative borders (e.g. city boundaries) and not towards accessibility (e. g. by public transport or car). Therefore, the *OWL.Culture-Platform* with its bundling and explorability across administrative borders represents a unique and novel software-based solution. This makes it extremely suitable to be used as a case for a case study on our approach.

The team that uses our developed approach in this case study is a student project group. As already introduced in section 1.3, a student project group is part of the master course computer science at the Paderborn University. In that, a group of 8-20 students work together over a period of a year on a research topic determined by the group organizer. The workload is 20 ECTS which corresponds to approximately 15 hours/week (1 ECTS~30 hours).

Goals of the project group according to the guidelines [@Dep18] are on a personal development level and on a research level. For the personal development, the students shall learn to work in a team, organizing a project, how to report progress and research findings as well as to experience an extensive development process in a team. In addition to the personal development, the project group shall contribute actively but not primarily to university research on current research topics and prepare for a master's thesis in this area.

How is this possibly affecting this case study? The team members are inexperienced regarding software development projects. This includes the development itself, working in a team as well as organizing such a project. Furthermore, the team members are not working full-time (40h/week) on the project but only part-time (~15h/week). Hence, they not only have to learn our approach but have to sharpen their skills regarding development, team work, and organizing during the project with less time than a normal employee would

have. This is a challenge because problems that may occur in this case study can only be partly attributed to the approach. It could be that the problems just occured because of the inexperience.

On the other hand, this inexperience could lead to team members learning the approach better. They do not have the expert knowledge that makes a certain approach look like no alternative (cf. Cognitive Flexibility [Dan10] and section 4.1). Also it could lead to a faster adaption due to less conflicts with existing value systems (cf. compatibility with value systems in section 2.1).

In general, we assess this situation positively for the case study. As already mentioned in section 1.3.2, this thesis focuses on the feasibility and not the performance compared to other approaches. We do not need to have a team perform at its very best with this approach to be able to compare it with other approaches. We just need a team to successfully implement the approach to be able to discuss its feasibility. Therefore, related to the possible limitations mentioned above, if the team members can successfully implement the approach although they have less time and more things to learn, then this should also be possible for experienced full-time employee. And if there is a negative effect because of the expert knowledge and the value system of experienced developers, it cannot occur in this setting, so we've eliminated a potential interfering variable.

With the goal, the case, and the participant in mind, we can go to the construction of the case study with the individual phases and tasks (section 8.3.2) before we discuss the evaluation instruments (section 8.3.3), and finally the conduction including the results (section 8.3.4).

### 8.3.2 Concept

The first stage (see Figure 8.6), *Initializing Design Thinking*, was already conducted separately from the other stages without the student project group. This stage was already entered in 2016, when initial ideas were analysed and formalized in the form of two project proposals, which were not approved. As a result, a further feasibility analysis was carried out, the integral part of which was a design thinking workshop in september 2017 (cf. [@Neg17]).

The design thinking workshop was conducted in an earlier version of the workshop format that we presented in section 5.2. Rather than developing immediately implementable solutions, the aim was to explore the problem and solution space and validate the ideas already collected. Accordingly, the target groups participating in the workshop were artists, decision-makers from culture, tourism and economy as well as people interested in culture. In total there were 61 participants from these groups. Outcome of the workshop were eleven concrete ideas of the eleven sub-groups in the form of handcrafted prototypes as well as 97 less

concrete ideas and a better understanding of the needs of the individual groups. These were analysed, processed and summarized in the so-called feasibility study *OWL.Culture-Platform* [@KNO18]. This feasibility study provides the basis for the design challenge (cf. section 2.2 and section 4.1) and thus the conclusion of the first phase.

As design challenge we decide for the following:

> How can we make Culture in OWL more attractive for Students and Young Professionals (with the help of Software)?
>
> – For finding and consuming culture
>
> – For creating culture / art
>
> – For organizing cultural events

We deliberately reduced the design challenge to students and young professionals for two reasons. First of all, we have better access to students as test subjects than to creative artists and decision-makers through our integration in the university context. This is reinforced by the student project group, which shall pass the individual stages, as they can integrate additional friends and acquaintances. Secondly, the first phase took place before the student project group was formed. As a result, there is a potential lack of understanding or empathy for the other target groups and the overall context. In order to compensate for this, we decided for a context with students and young professionals for the design challenge.

This leads us to the stages of our approach whose interaction we mainly examine in this case study. These are the stages *2. Execute Design Thinking with Non-Software*, *3. Prepare Design Thinking with Software*, and *4. Execute Design Thinking with Software*, which are genuine for our developed approach. Stage *5. Optimize* is the traditional incremental innovative software development that is not in the focus of this case study. The individual phases and tasks for passing through these three stages are presented in Table 8.3, while the individual stages are highlighted according to their color from Figure 8.6. Tasks that are not highlighted are not originally necessary for the implementation of the approach, but are introduced to teach the participants the necessary concepts at appropriate points. In the following we will explain the individual phases and tasks.

The first phase with the tasks 1 to 7 is the introductory phase. Goal of this phase is to introduce the team members to the case as well as to the mindset of the approach and some specific methods.

During *Onboarding* (task 1), the participants meet for the first time in this constellation. Until then, we assumed that they have no prior knowledge of the approach, its mindsets and the case to be worked on. Therefore, the first part of *Onboarding* is a brief introduction to

| # | Week | Task |
|---|---|---|
| 1 | 1 | Onboarding |
| 2 | 2 | 3 Ideas for the OWL.Culture-Platform |
| 3 | 3 | 1. Presentation Block |
| 4 | 4 | lego4scrum Workshop |
| 5 | 5 | 2. Presentation Block |
| 6 | 6 | Design Thinking Introduction |
| 7 | 7 | 3. Presentation Block + Introduction to Expert Interviews |
| 8 | 8 | Interview Guidelines |
| 9 | 9-10 | Present Results from Interviews |
| 10 | 11-12 | Pitch Ideas + Present Low Fidelity Prototypes |
| 11 | 12-15 | Create High Fidelity Prototypes |
| 12 | 15 | Internal Fair |
| 13 | 15-17 | Integrate Insight from Internal Fair for External Fair |
| 14 | 17 | External Fair |
| 15 | 17-18 | Preparation of the Results of the External Fair |
| 16 | 19-20 | Transforming Design Thinking Results into Agile Software Requirements |
| 17 | 20-23 | Determination of Minimum Viable Product |
| 18 | 24 | Presentation of Design Thinking Phase Results |
| 19 | 25-29 | High Fidelity Prototype of the Minimum Viable Product |
| 20 | 30-43 | Implementation of the Minimum Viable Product |
| 21 | 31 | Introduction to Feature Experimentation Platform |
| 22 | 40 | Introduction to Experiment Assignment Methods |
| 23 | 44-46 | Implementation and Preparation of Experiments |
| 24 | 46 | On-Site Experiments |
| 25 | 47 | Analysis of the Experiments and Preparation of the Final Presentation |
| 26 | 48 | Final Presentation |

**Tab. 8.3.:** OWL.Culture-Platform Schedule. ■ Stage 2: Execute Design Thinking with Non-Software, ■ Stage 3: Prepare Design Thinking with Software, ■ Stage 4: Execute Design Thinking with Software.

the case, the mindset/approach, the technologies to be used, and the organization in general (cf. [Sen20g]). This is followed by a team game (Egg Drop Challenge[1]) that has two goals. First, it is an icebreaker for the participants to get to know each other. Second, it shall teach the participants the lesson that cost of learning can seem large but put into perspective it is usually a good idea. The learning by testing in the field is an essential part in our approach that the experience of this game is meant to internalize. The third and last part of this task is an introduction to working culture, personality types and intercultural communication. This serves as preparation for teamwork, independent working and sensitization to other cultures. Especially the last part is important as the computer science masters programme at the Paderborn University has a high internationalization rate and it is the first time for many students to work with people from different cultural backgrounds and different working cultures.

Task 2 *3 Ideas for the OWL.Culture-Platform* corresponds to the activation step of the second stage *Execute Design Thinking with Non-Software*. The 3 ideas are actually our work product #1 that we will further analyze (see section 8.3.3.2). We deliberately set this task at the beginning, even though the remaining parts of this stage do not start until 5 weeks later. We want to give the participants the time to familiarize themselves with the case and let their ideas mature. The exact task for the participants is this:

> *To start you up with the OWL.Culture Platform, everyone shall send me at least three ideas what he/she would do to improve the situation in Ostwestfalen-Lippe (`https://en.wikipedia.org/wiki/Ostwestfalen-Lippe`). This can be about finding, organizing, advertising, and whatever you can think of related to culture (art, history, musicals, concerts, cinema, fairs, and so on). I anticipate Point of Views (see `https://www.interaction-design.org/literature/article/stage-2-in-the-design-thinking-process-define-the-problem-and-interpret-the-results`) for all ideas.*

The following phase with the tasks 3 to 7 serves to make the participants more familiar with the underlying concepts of the approach. The basic structure of this phase consists of a theory block followed by a practice block, which is intended to deepen and internalize the theory through practical experience. Within the theory block two sub-groups present a paper that they chose during the *Onboarding* event. For each presented paper two additional groups are selected as 'opposing groups'. They must prepare the paper and discuss with the presenters the strengths and weaknesses of the paper as well as its applicability. This is to ensure that the participants are already familiar with the content of the paper before the

[1] In the Egg Drop Challenge the participants shall protect a fresh egg from breaking when dropped approximately 4 metres onto concrete using only paper and adhesive tape. The material of the final version costs money as well as a broken egg in the test run. The participants are free to carry out a test run and take the risk.

presentation and can establish new connections or manifest existing knowledge during the presentation and discussion.

The first block of theory and practice is thematically based on the historical development of software development methods to the point of agile software development methodology and the significance of loops in these methods. The papers "A view of 20th and 21st century software engineering" by Boehm [Boe06a], "Understanding the relations between iterative cycles in software engineering" by Terho et al. [Ter+17] and "What lessons can the agile community learn from a maverick fighter pilot?" by Adolph [Ado06] will be presented for this purpose. This is followed by a lego4scrum Workshop[2] in the practical part in order to acquaint the participants with Scrum as an agile software development method. In our case the lego4scrum workshop is divided into a theoretical part and a practical part. The theoretical part (cf. [Sen20h][3]) serves as an additional introduction to the theories of Scrum to meet the university requirements. In the practical part, the participants have to "build a Lego town consisting of houses, cars, airplanes, roads and other objects". The single parts to build are written as user stories and are kept in a backlog. The building process is organized in several sprints with the corresponding scrum events like sprint planning or retrospective meeting.

As the next step, the participants learn in the following block the difference between point-based concurrent engineering (as the classical way in software engineering) and set-based concurrent engineering and how design thinking is related to that as well as to agile software development. For that, the papers "The second Toyota paradox: How delaying decisions can make better cars faster“ from Ward et al. [War+95] and "Design Thinking: A Fruitful Concept for IT Development?" from Lindberg, Meinel and Wagner [LMW11] will be presented. In the practical part, a theoretical introduction to design thinking was given (cf. [Sen20b]) and a short design thinking workshop conducted to familiarize the participants with design thinking. The workshop is based on the *Wallet-Project* from the Stanford d.school and is carried out with the help of the facilitator guide (cf. [Sen20i]) and the corresponding working sheets (cf. [Sen20j]).

The last block is to point out the limitations of agile software development regarding radical innovations and the need for field experiments. For this, the papers "Incremental and radical innovation: Design research vs. technology and meaning change" from Norman and Verganti [NV14] and "Online Experimentation at Microsoft" from Kohavi et al. [Koh+09] will be presented. This theoretical part is immediately followed by the practical implementation of the rest of the second stage *Execute Design Thinking with Non-Software*, which will be conducted in teams of three people.

[2] `https://www.lego4scrum.com/`

[3] Based on `https://www.mountaingoatsoftware.com/agile/scrum/resources/a-reusable-scrum-presentation`

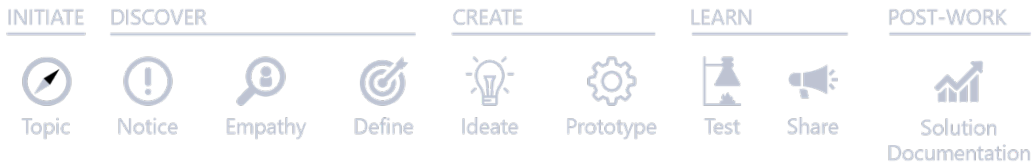



**Fig. 8.7.:** Design Thinking Process

As the participants already went through the activation, the next step in this stage is to gain empathy with the help of interviews (see Figure 8.7 and section 5.2). For this, a short introduction to interviews (cf. [Sen20a]) is given based on material from Helfferich [Hel11]. The task (8) of the teams is then to create an interview guideline (work product #2). Part of the interview guideline are the questions for the interview as well as the eighteen questions from Helfferich [Hel11] to prepare such interviews (Preparation, Conduction and Evaluation). After that, the teams conduct the interviews regarding their described target groups in the interview guideline with friends, relatives and students of the university in the ninth and tenth week. In the tenth week they present their results (task 09, slides are work product #3) in front of the other groups so that everyone can benefit from their insights.

Based on the insights they gained as part of their interviews and the results from the other teams, the next task (10) is to create ideas and build low-fidelity prototypes (work product #4) which will be pitched in week 12. For this, the prototype part of the design thinking theory presentation (cf. [Sen20b]) will be held again. This task corresponds to the creation phase of our design thinking workshop format. Goal is to define ideas (cf. *Create Phase* in Figure 8.7) with corresponding low-fidelity prototypes and already get feedback to them from the other teams before going to build more sophisticated prototypes which includes more than the to be delivered value.

The next task (11) for the teams is then for each one to create a high fidelity prototype (work product #5) (cf. [Sen20b]). These prototypes mark the completion of the create phase and will be evaluated in the learn phase via two fairs (cf. *Learn Phase* Figure 8.7). The concept of the fair is that each team exhibit their high-fidelity prototype and gather feedback (work product #6) with the help of the test and learning card (cf. [Sen20f], 5 Lern-Karte and 6 Test-Karte, translated to english) and the feedback grid (cf. [Sen20f], 07 Feedback-Bogen, translated to english). While on the internal fair (task 12) the other teams try out the prototypes, on the external fair (task 14) the target group are other students and it takes place on the main campus in front of the library (see Figure 8.8). There are two weeks between the two fairs to give the teams enough time to incorporate the feedback and experiences (e.g. on the implementation) into their prototypes and exhibition stand (task 13).

This stage is finished by task 15, *Preparation of the Results of the External Fair* which corresponds to the Post-Work Phase (cf. Figure 8.7). Within this task, the participants have


**Fig. 8.8.:** External Fair

to process the feedback from the *external fair* and incorporate it into their prototype before continuing with the next stage.

The next stage is to *Prepare Design Thinking with Software*. For this, the results of the previous stage have to be transformed into agile software requirements (task 16). In our case, this will be done via the *Design Thinking Requirements Framework (DTRF)* presented in section 6.2. Since we also use this opportunity to evaluate this framework, we will conduct an experiment at this point in which one half of the group performs the transformation using the framework and the other half of the group does it solely based on their knowledge of agile software development.

The result of the last task is a list of epics and user stories (work product #7) linked to specific design thinking artefacts. The challenge here is the natural ability of design thinking to generate partly overlapping and partly independent ideas. As a result, we do not get a single coherent product and there can be partial overlaps between the groups. Therefore, the epics and user stories from task 16 must be consolidated and prioritized in task 17. The prioritization has its focus on the indispensible features necessary for a minimum viable product as we are developing in a green field. Otherwise, we would not be able to experiment in as natural a setting as possible, but could only set up strictly isolated experiments.

This point is an important milestone in this case study for two reasons. On the one hand, it creates the prerequisites for actively turning to software development. On the other hand, this is the halftime of the case study. Therefore, a summarizing presentation of the previous results takes place at this point (task 18), which is designed in the form of several stations and is also open to external parties.

With the transformed and prioritized design thinking results, the potential *Value*, the things that have "relative worth, utility, or importance" to the users, has been gathered. According to our approach (cf. section 3.1), in the next step, the *Technical* level (”answering technical questions about how a future artifact might actually be made to work”) and the *Look &*

*Feel* level ("explore and demonstrate options for the concrete experience of an artifact") will be explored in parallel. This is done in task 19 where a high fidelity prototype of the minimum viable product will be developed as well as technical concepts evaluated. This time, the high fidelity prototype includes already technical functionalities that are partly mocked with the Wizard of Oz method (cf. [Kel84]). Foundation for the backend services will be a natural language processing pipeline for cultural events developed in a previous seminar at the Paderborn University.

With this high-fidelity prototype, which has designed, implemented and evaluated both *Look & Feel* and *Technical* concepts as well as integrated all three levels (*Value, Look & Feel, and Technical*), the actual implementation of the minimum viable product (task 20) can begin. This also marks the beginning of the fourth stage (*Execute Design Thinking with Software*). The implementation of the minimum viable product (work product #8) will be done as a *System of Systems* (cf. section 7.2.2) realized as *Docker* containers that are deployed on a *Kubernetes* cluster set-up by *Rancher*. For the continuos deployment pipeline, the integrated *GitLab CI/CD* pipeline will be used. As UI Framework Angular 7 with Typescript will be used. On the backend side, *.NET Core* with web-apis and C# will be used as standard implementation if other technologies are not preferred. The services on the backend side shall work with event sourcing and *CQRS* (see section 7.2.2.1) to enable the use of parallel models and achieve a higher evolvability. As event store for event sourcing, *Event Store*[4] will be used. The development method to develop the software is *Scrum*.

With the minimum viable product finished as the prerequisite to conduct experiments in a natural setting, the implementation and preparation of experiments can start. To prepare this, the participants were already familiarized with the definition of experiments (task 21) and the technical assignment of experiments (task 22) during the implementation of the minimum viable product. The familiarization with the definition of experiments is carried out in the form of an introduction to the *Feature Experimentation Platform* system by an usability test (see section 7.3.1), and the introduction to the technical assignment of experiments is carried out as an introduction to the *Technical Assignment* system with another usability test (see section 7.3.2).

The experiments wil be defined (work product #9) in the *Feature Experimentation Platform* tool and then the additional variants are implemented (work product #10) accordingly (task 23). Designed as field experiments, it is indispensable for them to collect qualitative data in order to understand and explain why certain situations arise. This requires above all the possibility to actively question the participants of the experiments. This is why these experiments will be carried out on site with stakeholders of the *OWL.Culture-Platform* (task 24).

[4] `https://eventstore.org`

The last task in this fourth stage is the analysis (work product #11) of theses experiments (task 25). Finally, all results of the tasks up to this point will be summarized in a presentation by the participants (task 26). The fifth stage *Optimization* is not part of the consideration of this case study. With the presence of the product and the learning from the experiments, the corresponding parts can be improved incrementally, as according to Norman and Verganti [NV14], using existing software development methods such as *Scrum* and therefore it should be possible to conduct the fifth stage from this point on.

### 8.3.3 Evaluation Instruments

In order to measure the interaction of the individual stages and to what extent inexperienced developers are able to work with our approach, we use two measuring instruments. This is on the one hand a standardized questionnaire and on the other hand it is the examination of the individual work products.

#### 8.3.3.1. Questionnaire

For the questionnaire we use the student course critique for project groups, which has been in use for several years and has proven itself accordingly. This is an excellent tool for the questions we have in this case study, as it asks for important factors for project groups such as difficulty, effort, affinity, group dynamics, advisor relationship and task adequacy. How exactly these factors are related to our case study is explained in the following using the individual parts of the student course critique.

The first part with the first three blocks (see Table 8.4) focus on the expectations of the participants and some general information. In *1. Questions about you* study time in the Master's program and the location of the bachelors degree are queried. Especially the latter is revealing to what extent the study contents of the Bachelor in Paderborn can be accepted as given. In addition with the study time it gives some indicator for potential conflicts because of unaware diversity[5]. In particular, a high degree of diversification is an argument in favour of better generalizable results. If the participants had studied exclusively in Paderborn, it could be argued that the results came from the studies in Paderborn and cannot be generalized.

The second block *General Questions* asks for general classifications of the project. It begins with the difficulty level. The best result here would be a 3, since the participants were not

[5]Paderborn's Master's program in Computer Science has a high degree of internationalization (2/3 foreign students in 2019). Especially in such practical projects, for the first time the differences in the cultures to which the participants have yet to adjust are clearly visible.

| Shortcode | Question | Answer Option |
|---|---|---|
| 1. Questions about you | | |
| Study Time | How many semesters have you already been studying in the Master's program? | ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ >=6 |
| Bachelor Location | Where have you earned your Bachelor's degree? | ☐ *in Paderborn* ☐ *somewhere else in Germany* ☐ *outside Germany* |
| 2. General Questions | | |
| PG Difficulty | How do you rate the PG's difficulty level? | too easy ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 too difficult |
| Time Spent | How much time did you spend on PG work per week? | ☐ *approximately 15h* ☐ *far less* ☐ *far more* |
| Project Field | Which discipline matches your project best? | ☐ *Software Engineering* ☐ *Algorithm Design* ☐ *Networks and Communication* ☐ *Computer Systems* ☐ *Intelligence and Data* |
| PG Research Type | Please rate the PG regarding the terms research-oriented and practice-oriented | research-oriented ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 practice-oriented |
| 3. Choice of the Project Group | | |
| Looking For Field | I was looking for a PG associated to | ☐ *Software Engineering* ☐ *Algorithm Design* ☐ *Networks and Communication* ☐ *Computer Systems* ☐ *Intelligence and Data* ☐ *I had no preference* |
| Looked For Type | I was looking for a PG with a focus on | research ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 practice |
| Preferred PG | I joined my preferred PG | ☐ *Yes* ☐ *No* ☐ *I had no preferred PG* |
| Attended Announcement | I attended the project groups announcement event | ☐ *Yes* ☐ *No* |
| Realistic Introduction | The event provided me a realistic impression of the PG | agree ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 disagree |
| Prior Expectations | Prior to the PG I had an accurate impression of how the PG will be like | agree ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 disagree |

**Tab. 8.4.:** Questionnaire Part 1: General Information and Expectations

overstrained by the tasks and the approach and considered them not as trivial. The latter in particular may indicate that the participants did not understand the tasks correctly, which is why they erroneously judge them to be too easy.

A student project group course has a workload of 20 ECTS spread over a year, which roughly translates to 600 hours or approximately 15h/week. Therefore, for the second question, the participants should answer *approximately 15 hours* in the best case. Otherwise, this indicates that the tasks were overstraining or were not handled with the intended thoroughness or that they were wrongly assessed in their scope by the authors.

The third question aims at the classification of the project group on the basis of the focus areas in the Paderborn Master's programme. Since our approach is focused on requirements engineering and system design, the best answer here is *Software Engineering*. However, we use different techniques from the other focus areas (cf. section 7.3). If therefore the other areas were chosen, this would indicate that they are too much in the focus and are no longer recognizably only a tool to achieve the goals of our approach.

Although this is a case study in which our approach is evaluated, the best result of the fourth question of this block regarding the type of research is that it is a practice-oriented project group. This would speak for a more natural setting in which software development is in the foreground and not academic application and research. On the one hand, this improves the generalizability of the results and on the other hand, it indicates that at least the participants did not perceive any confounding variables resulting from the test setting.

The third block of this first part is about the expectations the participants had prior joining the project group. This includes for which focus area and research type they were looking for as well as they have joined their preferred project group. The better the match here, the more likely it is that they are open to the ideas of our approach (see Diffusions of Innovations, Value System paragraph 2.1).

Furthermore, the influence of the announcement meeting on the expectations is asked. Each semester an announcement meeting is held to introduce the students to all the project groups starting that semester and the advisors are then available to answer questions. As this is not mandatory it is asked as well if the students took part in it. Furthermore, it was asked if the students had a an accurate impression of how the PG will be like before joining it.

The second part (see Table 8.5) focuses on the advisors[6], the team, and the adequacy of the tasks, i.e. the general conditions that determine the success of the case study regardless of the approach.

[6] Advisors is the term used in students project groups for the teaching staff organizing the project.

| Shortcode | Question | Answer Option |
|---|---|---|
| 4. Advisors | | |
| Easy Contact | The advisors where easy to get into contact with | agree ☐1 ☐2 ☐3 ☐4 ☐5 disagree |
| Competent Answers | The advisors competently answered questions | agree ☐1 ☐2 ☐3 ☐4 ☐5 disagree |
| Guidelines | The advisors should have given more guidelines | agree ☐1 ☐2 ☐3 ☐4 ☐5 disagree |
| Too little Managing | The advisors managing of the group was too little | agree ☐1 ☐2 ☐3 ☐4 ☐5 disagree |
| Too much Influence | The advisors exercised too much influence | agree ☐1 ☐2 ☐3 ☐4 ☐5 disagree |
| Sufficient Feedback | The advisors provided me with sufficient feedback | agree ☐1 ☐2 ☐3 ☐4 ☐5 disagree |
| Atmosphere | The atmosphere between group members and advisors was pleasant | agree ☐1 ☐2 ☐3 ☐4 ☐5 disagree |
| 5. Tasks and Team | | |
| Tasks Understandable | The tasks were understandable | agree ☐1 ☐2 ☐3 ☐4 ☐5 disagree |
| Satisfied with Responsibilities | I was satisfied with my responsibilities | agree ☐1 ☐2 ☐3 ☐4 ☐5 disagree |
| Satisfied with Workload | I was satisfied with my workload (regarding the amount of work I did and the work I got assigned) | agree ☐1 ☐2 ☐3 ☐4 ☐5 disagree |
| Well Internal Communication | The group's internal communication and selforganization worked well | agree ☐1 ☐2 ☐3 ☐4 ☐5 disagree |
| Pleasant Group Atmosphere | The atmosphere between group members was pleasant | agree ☐1 ☐2 ☐3 ☐4 ☐5 disagree |
| Sufficient Feedback from Group | The other group members provided me with sufficient feedback | agree ☐1 ☐2 ☐3 ☐4 ☐5 disagree |
| Satisfied with other Members | I was satisfied with the other members of the PG | agree ☐1 ☐2 ☐3 ☐4 ☐5 disagree |

**Tab. 8.5.:** Questionnaire Part 2: Advisors, Team, Task Adequacy

Within the fourth block (*Advisors*), the involvement of the investigator is captured from the point of view of the participants. The role as investigator and advisor in this case study is on the one hand to teach the participants the approach and to support them in case of problems. On the other hand, to grant freedoms and to limit the influence to the most necessary extent in order to limit confounding effects by the investigator participants' relationship.

Besides the basic things like *Easy Contact*, *Competent Answers*, *Sufficient Feedback*, and *Atmosphere*, especially the questions about *Guidelines*, *Too Little Managing* and *Too Much Influence* are interesting for this case study. These last three must be seen as a package in order to evaluate the influence of the advisor/investigator from the participant's perspective. In the manner in which these three questions are asked, they can be interpreted differently in the area three to five. On the one hand, it can mean that the participant regards three as optimal and five as the other negative. On the other hand it can mean that five fully contradicts the statement in the question and that this would thus be the optimal answer. Therefore, these three questions need to be considered together and it is positive for the case study if the participants consistently choose the range between three and five for all three questions.

The fifth block (*Tasks and Team*) is divided into the first three questions regarding task adequacy and the last four questions regarding team dynamics. An advantage for the case study in these cases is an answer behavior close to agree. This would argue for appropriate tasks and advantageous team dynamics that do not interfere with the case study.

The last part (see Table 8.6) consists of the three blocks *Tools*, *Conclusion*, and *Praise, Criticism and Suggestions*, whereas the last block only consists of a free-text-field for further feedback from the participants. In the block *Tools*, the provided tools as well as the self-organized tools are captured, as well as whether the provided tools are regarded as useful and whether requested tools were organized.

*Conclusion* captures the relation of the participants with the approach. The first question about the learning outcome also gives feedback of how well the participants are already used to this kind of approach. Generally speaking, answers on the right side (disagree) of the scale are more favorable for our approach as the other side (agree) would mean that it is not as novel and this raises the question if it is necessary. To verify that the approach is not only seen as novel but as well as useful this is captured in the fourth question of this block. Furthermore, it is asked if the *prior knowledge was sufficient* to participate in the project, i.e. if the given information was sufficient to work with the approach and if the information was helping to gain an understanding of the approach (*gained understanding*). The second last question asks if the participants are motivated to further work in this field and the last question is for a summed up assessment of the complete project group.

| Shortcode | Question | Answer Option |
|---|---|---|
| 6. Tools | | |
| Provided Tools | The following tools were provided by the advisors (e.g. mailing list, repositories, wiki, software, hardware, etc.) | Free-Text Field |
| Useful Tools | The tools provided by the advisors were useful | agree ☐1 ☐2 ☐3 ☐4 ☐5 disagree |
| Got Requested Tools | The advisors organized the tools requested by the group | agree ☐1 ☐2 ☐3 ☐4 ☐5 disagree |
| Self-Organized Tools | The following tools were organized by the group: | Free-Text Field |
| 7. Conclusion | | |
| Only learned Little | I already knew major portions of the PG's contents so I only learned little | agree ☐1 ☐2 ☐3 ☐4 ☐5 disagree |
| Prior Knowledge Sufficient | My prior knowledge was sufficient to effectively participate in the PG's work | agree ☐1 ☐2 ☐3 ☐4 ☐5 disagree |
| Gained Understanding | The PG provided me with deeper and sounder understanding of the relevant topics | agree ☐1 ☐2 ☐3 ☐4 ☐5 disagree |
| Gained Useful Experience | I gained useful experience | agree ☐1 ☐2 ☐3 ☐4 ☐5 disagree |
| Wants to Deepen Knowledge | I would like to deepen my knowledge in the field the group worked on | agree ☐1 ☐2 ☐3 ☐4 ☐5 disagree |
| Overall good Impression | My overall impression of the project group | good ☐1 ☐2 ☐3 ☐4 ☐5 poor |
| 8. Praise, Criticism and Suggestions | | |
| Further Feedback | Here you can give further feedback | Free-Text Field |

**Tab. 8.6.:** Questionnaire Part 3

#### 8.3.3.2. Work Products

Our second evaluation instrument besides the standardized questionnaire is the examination of the individual work products. Whereas the standardized questionnaire is mainly to evaluate if inexperienced developers are able to work with our approach, the examination of the individual work products is to assess the quality and the interaction between the different stages. Within the description of the different stages section 8.3.2 we described the created work products and numbered them accordingly (work product #XX). The full list of the work products created in this case study can be found in Table 8.7.

| # | Title |
|---|---|
| 1 | 3 Ideas for the OWL.Culture-Platform |
| 2 | Interview Guideline |
| 3 | Interview Results Summary |
| 4 | Low Fidelity Prototype |
| 5 | High Fidelity Prototype |
| 6 | Fair Feedback |
| 7 | Agile Software Requirements |
| 8 | Minimum Viable Product |
| 9 | Experiment Definitions |
| 10 | Experiment Implementation |
| 11 | Experiment Analysis |

**Tab. 8.7.:** Individual Work Products

**Work product #1:** *3 Ideas for the OWL.Culture-Platform* This work product is not a prerequisite for the further stages. However, it can be used to determine the extent to which the participants are already able to distinguish between problem and solution as well as user needs and technical needs as this work product should consist in underpinning *Point of View* (POV) and a corresponding technical solution idea according to the task. The POV should be a description of the user, her need, and insights that substantiate this need (see section 5.1). This makes it to an actionable and meaningful problem statement as we have a specific user with its need and the supporting information that helps us to decide to what extent our solution idea actually matches the need. It also allows us to think about different solutions, as it has the actual intention and does not directly focus on a technical solution. Therefore, the ideas are first analyzed with regard to the POV to what extent they are explicitly named, the proposed structure from *[**User**...(descriptive)] needs [**need**...(verb)] because [**insight**...(compelling)]* is used, or they exist in this order in another form.

In order to determine other forms as well to evaluate the quality of the POV, we first mark all occurrences that can be assigned to either *User*, *needs* or *insight*. In the second step we assess

these occurrences. For *User*, we categorize whether only the word user, an umbrella term for a specific user group, or an exact description of the user is used. *Needs* are categorized into if they try to solve the need of the corresponding user or another and if it is an action or a description. If a need is not written as an action it is hiding the underlying task of the user and therefore important information of what she is actually trying to achieve. Finally, for *insight* we categorize to what extent the insights represent challenges, gaps, limitations or deeper needs related to the corresponding *need*.

After the analysis of the POV, as a basis for all solution ideas, the separation of problem and solution space is analyzed. For this purpose, the separation of problem and solution space is categorized in an initial categorization with regard to obviousness (explicitly named, comprehensible due to text structuring, and not present or implicitly present). In a second step, it is analyzed whether a direct connection is made between the solution and the problem and whether the solution and the problem space are actually cleanly separated.

It follows from this that a high-quality idea follows a clear separation between problem and solution space and clearly represents the connection between the two. In addition, at least one user group must be explicitly described in the problem space within the POV as well as a need in the form of an action of this user group and the associated insights, which are derived from challenges, gaps, limitations or deeper needs related to the corresponding *need*. This is essential to develop and validate different solution alternatives. If the participants have already mastered this, they have the best prerequisites for carrying out the approach.

**Work product #2:** *Interview Guideline* This is the result of the preparation for the *Empathy Design Thinking* step that has as goal to getting to the bottom of latent user needs, explore surprising, develop empathy through interviews, observations, and personal experience (see [Sen20b]). Furthermore it is for joint sense-making, unite perspectives and gain insights out of them through analytical condensing and focussing on one user.

The *Interview Guideline* consists of the question set for the interviews and the 18 decision steps by Helfferich, whereas the decisions serve to to "determine the object of research, its theoretical location, the form of the interview and the evaluation strategies" [Hel11, pp. 168-171]:

1. Decision for a (precise) research object
2. Choosing a target group and narrowing down the sample
3. Decision for an interview form
4. Decision for an evaluation strategy
5. Decision and clarifications related to interview behavior

6. Decision related to the strangeness/familiarity of the actors
7. Decisions to profile the professional role in advance
8. Decisions to profile the professional role in the interview
9. Decisions on the design of spatial aspects
10. Decisions for the behavior of the interviewees in difficult interview situations
11. Decisions regarding the design of instruments
12. Decisions for recruitment channels
13. Decisions on dealing with ethical aspects
14. Decisions on the scheduling of execution
15. Decisions on personnel implementation
16. Decisions related to research documentation
17. Decisions related to the continuation of interview qualification into the evaluation phase
18. Decisions on the use of other instruments

The *Decision for a (precise) research object* is the most important decision in this work product. It is the basis for the further decisions, especially for the interview form, the specific questions, and the target group. According to Helfferich the research object consists of a content-related facet (e.g. event organization, event consumption, software development) and a theoretical-methodological facet (What should be presented as a result? e.g. patterns of interpretation, subjective theories, structures of meaning, coping patterns). Thus, we first check whether both parts are present. If this is the case, we look at the quality of the respective facet.

For the content-related facet a thorough description of the context is important, since also here as in requirements engineering it is valid that if context and relevant context aspects are not considered, this leads directly to errors (cf. [Poh07, p.55]). Therefore, this facet is categorized according to whether the context is described with more than one sentence and analyzed how it is described.

The theoretical-methodological facet is limited by the assignment of the interviews to the *Empathy Design Thinking* phase, as already described above. This way, the facet is analyzed to see if it can capture latent user needs, explore surprising, and develop empathy (e.g. through subjective theories, coping patterns, patterns of interpretation, structures of meaning). It would not be good if this facet was limited only to the mere collection of information and content, as this would be contrary to qualitative techniques (see [Hel11, p. 168]) and thus also to the reconstruction of meaning as well as the understanding of the problem which is the foundation for gaining empathy.

*Choosing a target group and narrowing down the sample* is important in the sense that it determines the generalizability of the results as well as ensuring that the desired objectives of the research object can be achieved. To this end, the decision should precisely define the group to be interviewed (including an exhaustive description of the group), how many interviewee are necessary, and a discussion of the extent to which this is relevant to the research object. Accordingly, at this point it is analysed to what extent the group to be investigated is described, the number of participants and their characteristics are defined and a connection to the research object is established.

*Decision for an interview form* depends on the previous decisions, as the interview forms (see [Sen20a] for interview forms presented to the participants) are specialized for certain applications. Therefore, it will be checked at this point to what extent the selected interview forms are related to the previous decisions.

In order to prevent the results of the interviews leading to stereotypes based on the assumptions and inclinations of the interviewers, it is essential to define an evaluation strategy that guarantees the objectivity of the results. Hence, the *Decision for an evaluation strategy* is analyzed regarding the intermediate steps introduced to get to the final results. The less intermediate steps are present the more difficult it gets to comprehend how the interviewer got to the results. For example, omitting the recording and transcription of the spoken word leads to a situation in which only what the interviewer considers to be important during the interview is recorded and thus important sentences of the interviewee can be lost for the objective analysis.

According to Helfferich [Hel11, p. 169], these four decisions make up the bulk of the decision-making work. The other decisions relate to individual aspects, but these are largely no longer open. These individual aspects include interview behavior, the relationship between interviewer and interviewee, concrete preparations and further decisions on implementation details. We will therefore analyse these individual aspects in relation to the previous four main decisions.

In addition to these decisions, this work product also naturally consists of the question set respective the concrete interview questions and the interview structure. As we have learned in section 4.1, knowledge is not always available on an ad-hoc basis, but requires an activation either by working in that context or by imaging to be in this context (the more triggers, the better we can imagine it and the more related knowledge is activated). Therefore, we will first analyze to what extent the participants are activated with respect to the context. Will the actual questions be started immediately? Is there an activation phase? How is this designed?

The next part of the analysis of the questions deals with their design. Are these openly designed and encourage the interviewee to report in detail about himself or are they rather closed and ask for information or serve only to confirm the ideas of the interviewer? As already described above, it is essential for the development of empathy to understand how users act and from which motives. A pure factual information that someone is using a certain website is therefore less helpful than a description from which can be derived what goals the person is trying to achieve and what deeper needs (e.g. motives or emotions) are behind it.

Furthermore, the dig deep mentioned in design thinking is crucial for building empathy. This means that the understanding of the statements made has to be verified by in-depth follow-up questions. Therefore, it is investigated to what extent such follow-up questions exist and are suitable to deepen or validate the understanding.

**Work product #3:** *Interview Results Summary* As mentioned in the concept section 8.3.2, the purpose of this work product is to share the results of one's own interviews with the other groups so that they can also benefit from them. Since the other groups were of course not part of the interview, the first question is whether enough context information is given to the conduction of the interviews so that the other groups can understand what happened and to what certain answers could be linked. Was the interview introduced? Are there any details about the number of participants, their background, and the length of the interviews? Were there any deviations from the 18 decisions?

The most important part here, however, are definitely the results presented. Since we are still in an understanding phase, it is important that the results are not only condensed on a factual basis, but also the corresponding individual stories. Only this makes it possible to derive further hypotheses on user needs. It is not the aim to confirm existing knowledge by a quantitative method, but to make sense and to generate new insights. For this reason, the results are checked to see whether they only reflect facts in a condensed form, tell underlying stories or do both.

**Work product #4:** *Low Fidelity Prototype* Goal of this work product is to define ideas (cf. *Create Phase* in Figure 8.7) with corresponding low-fidelity prototypes on a *Value* level (see Figure 8.9 and cf. section 3.1). Hence, the first prerequisite is the existence of related POV. The check of the POV takes place in the same way as in work product #1. Furthermore, in the best case, there is a POV's comprehensive problem definition, which also describes the user group and the context. If this is not available, our design challenge (see section 8.3.2) can still be assumed to be the fundamental problem.

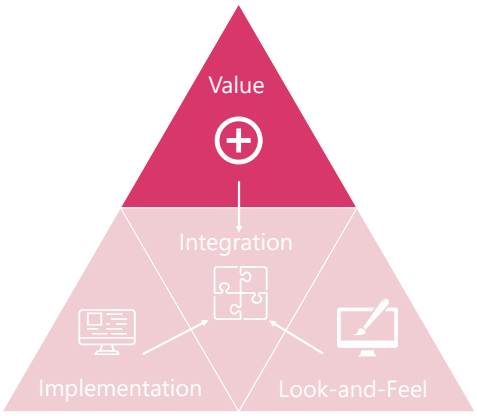



**Fig. 8.9.:** Prototype Levels: Value

In the next step, we evaluate the selected prototype techniques for the low fidelity prototype. Suitable prototyping techniques for this stage are concepts, digital wireframes, analogous wireframes, physical prototypes, story-telling metaphor, or theatre and videos (cf. [Sen20b, slides 60–68]). However, this is not an exclusive list. For the quality of the implementation, we refer in the evaluation to the do's and don'ts for the prototypes from slides 59-78 in [Sen20b]. In this stage, the participants shall focus on understanding the problem and possible solutions, install information and structure, make it tangible and observe how others use the prototype. Don'ts for this stage are prototypes that are presented as a finished product, have complexity that is not necessary for understanding the value, and prototypes that need to be explained.

Finally, it is investigated to what extent a connection between POV, idea and prototype has been established.

**Work product #5:** *High Fidelity Prototype* The goal of this work product is to have a prototype that feels already like a real product but hasn't implemented the necessary functionality. It is to have an artefact with a high immersion that users can try out and doesn't need as many resources as software to be implemented. This is based on the idea to try out as early as possible and as close as possible to reality the solutions (cf. section 8.2). Therefore, this prototype is about value, look & feel, and integration of these two (see Figure 8.10). This includes that the participants shall include the insights from value and look & feel as well as interactions, but leave out functionality (cf. [Sen20b, slides 72–77]).

As first step we analyse the foundation of the high-fidelity prototypes. What kind of tools are used to create the prototype? In which way do they limit them to only include value and look & feel or encourage to include more? What is the amount of boilerplate stuff that you have to do to achieve the things necessary for this prototype?

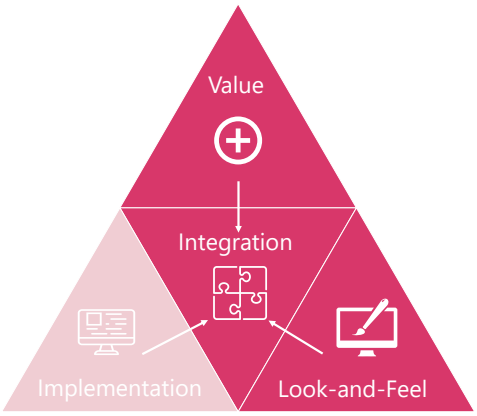



**Fig. 8.10.:** Prototype Levels: Value, Look & Feel, Integration

In the next step we have a closer look at the high-fidelity prototypes itself. Do they incorporate only value related concepts and look & feel or have they already added functionality? If they added functionality was it already implemented or done like in the Wizard of Oz method (see [Kel84]) to mock this functionality? Is the prototype based on the previous low-fidelity prototype? What is the immersion level? Are transitions obviously done manually or not fluent? Are they using a consistent design language? Is the information presented complete and related to real world data? Did the participants design the prototype as throw-away prototype or as reusable prototype that shall be evolved to a product? If it is designed as reusable prototype, what parts are designed to be reusable?

**Work product #6:** *Fair Feedback* This work product summarizes the feedback the participants got on their prototypes. As with all survey instruments, in order to collect this feedback, it is important to first define which hypothesis is being investigated, which questions shall be answered and what the fail/pass conditions are. This is the only way to observe in a targeted manner. In order to achieve this, the participants have been given the test and learning cards as an obligation for the internal fair (see [Sen20f, Working Sheets 6 and 7]). These cards are based on the testing and learning card from the *Design Thinking Playbook* [@Tra16] and consist in their basic structure of *Step 1: Hypothesis - We believe that ...*, *Step 2: Test - To verify that, we will ...*, *Step 3: Metric - And measure ...*, and *Step 4: Criteria - We are right if ...* respectively *Step 1: Hypothesis - We believed that ...*, *Step 2: Observation - We observed ...*, *Step 3: Learnings and Insights - From that we learned that ...*, and *Step 4: Decisions and Actions - Therefore, we will ...*. Since this work product is only the collection of the feedback, the test card is not necessarily part of this work product. However, the completed learning card should at least appear in the feedback on the internal fair, at best as well in the feedback for the external fair.

In addition, we have the situation that our knowledge of the problem space at this level is not yet secured (cf. section 1.1.1), which is why it is necessary not only to ask for potential challenges, but also what works or has been particularly valued and what has not been understood. To support the participants in giving and gathering such feedback, we have handed them out the feedback grid (see [Sen20f, Working Sheet 8]) from Tran [@Tra16] which actually asks for *Things I like most*, *Things that can be improved*, *Things I don't understand*, and *New ideas to consider*. These points should therefore appear in the documented feedback from both fairs.

**Work product #7:** *Agile Software Requirements* As mentioned in section 8.3.2, the evaluation for the *Design Thinking Requirements Framework (DTRF)* has been done in conjunction with a master thesis and is presented in section 6.2. The additional part of this work product is the derivation of the minimum viable product out of the results from the *DTRF*. This has to be a synthesis of the different group results with a sustained connection to the *DTRF* results. Therefore, at this point it will be examined to what extent the results of different groups have been combined and how the connection to the actual results will be maintained.

**Work product #8:** *Minimum Viable Product* Even if no alternatives should yet be implemented in this work product, it is already possible to analyze the extent to which the system can be changed and the participants can implement the technical concepts for evolutionary systems (cf. section 7.2.2). The database and domain logic is provided by a *natural language processing* (nlp) pipeline, which has already been developed in another project preceding this case study. This nlp pipeline (see Figure 8.11) consists of a crawler (for events in OWL), an enricher (enhance, refine, and expand raw data from crawler), a data store (structured data model and API gateway), and a recommender (recommending events). It was developed with the idea of event sourcing in mind and thus uses a message-driven architecture. The participants will adapt individual parts of this pipeline, but above all they will develop the web frontend.

Even if the pipeline was already developed before this case study, we analyse it first, as the participants are dependent on it and its changeability. The analysis takes place primarily with regard to the implementation of event sourcing and CQRS, since these two patterns are the key for us to use parallel models and thus for the independent changeability of individual components as well as the simultaneous operation of different variants. Problems should arise during the change attempts by the participants of the case study.

Further analysis will focus on the implementation of the frontend. Among other things, the implementation of the Model View Presenter or Model View ViewModel pattern is considered, which is to guarantee the detached development of the frontend from the backend.

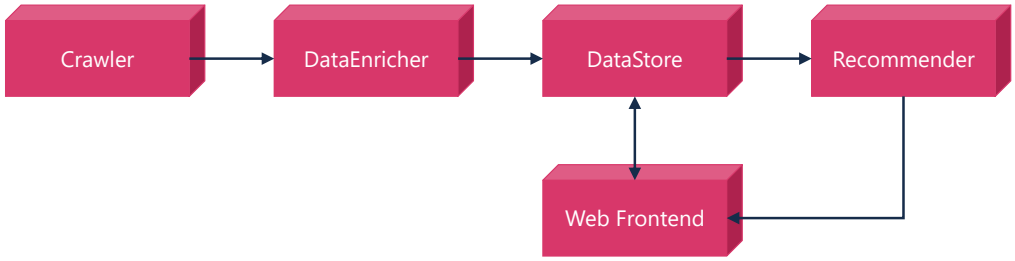



**Fig. 8.11.:** nlp-Pipeline developed preceding this case study.

In addition to the technical implementation, we record the extent to which the participants were blocked during the development of frontend components by changes to the backend and the extent to which backend properties were successfully changed subsequently for the frontend or had to be changed. This is the connection between frontend and backend.

For the frontend itself, certain technical properties must also be met, so that it can be easily adapted in the future and allows the simultaneous operation of different alternatives. This means that the frontend should also support the replacement and online loading of components. One way in which this can be realized is through web components. Since we have not specified how the participants are to implement this, we consider the extent to which components can be exchanged and loaded online.

**Work product #9:** *Experiment Definitions* The minimum requirements for an experiment definition are a hypothesis, an evaluation criterion (dependent variable), treatments, test participants and experiment conditions (cf. section 7.2.1 and [Ren+12]). The experiment conditions and the test participants are already predefined to a certain degree by the advisor / investigator. The participants should carry out the experiments on site with the steering committee of the OWL.Culture-Platform. This results in the participants having a time window of 120 minutes for their experiments and a large meeting room in which all experiments with the members of the steering committee are carried out simultaneously. The members of the steering committee (maximum 20 persons) are representatives of the local authorities in the field of culture and other cultural stakeholders. Hence, the participants have to define at least a hypothesis, evaluation criterion and treatments.

Thus, we check first whether all these three parts are defined and possibly also further parts. Subsequently, we analyse the design of the individual parts and their interaction. Due to the current stage in the approach with the associated uncertainties, we expect above all definitions here that are based on qualitative techniques and are closer to a usability test than to a controlled experiment. Further, we expect the hypothesis to have a low granularity. This

should be expressed in the treatments, here the different software components, to the extent that these do not only cover one level in a software component hierarchy, but several.

**Work product #10:** *Experiment Implementation* To this work product we include the preparatory measures for the execution of the experiment as well as the actual implementation. Which evaluation strategy shall be used and how is the collection of data organized? How is the experiment structured from the greeting to the introduction, the guidance through the experiment and the farewell? Which media should be used for data recording and how were they prepared to assure that they work in the experiment? How is additional support required for the experiments realized?

**Work product #11:** *Experiment Analysis* Last but not least, the experiment analysis is analysed. What was the evaluation strategy? How many intermediary steps have been documented? Is it obvious which statements are based on which experiments? Did they gathered additional insights? Is there a summary for each hypothesis with the variants, results / action to take, and additional insights?

This marks the last work product to be examined. In the next section we come to the conduction of the case study as well as their results.

### 8.3.4 Conduction and results

This case study was conducted by the thesis author as advisor and investigator. A total of 18 computer science students participated in this case study as part of the project group of the master's degree course in computer science in Paderborn. These 18 students are divided into two groups. On the one hand a group, which started already in the semester before and carried out the quality assurance and archiving of the HiP-App in this one, thus the optimization phase. This group did not come into contact with the approach or design thinking in the first semester. The second group consisting of the remaining 10 students started the project group with the case study.

Accordingly, both groups are at the same level of unknowingness about the approach. However, this also means that some of the participants drop out after the first half of the case study. This, however, is not further tragic for the case study, since after the first half, i.e. one semester, a new stage begins in which work is again carried out as a whole group and not in subgroups (cf. section 8.3.2). Thus, we conducted the questionnaire at two points (at the end of each half) of the case study in order to receive feedback from all participants as well as for all parts of the case study.

#### 8.3.4.1. Questionnaire results

For the first survey after the first half of the case study, 17 of the 18 participants filled out the questionnaire, whereas for the second half all remaining 10 participants filled it out (see Table 8.8). 23,5% (20%) earned their Bachelor's degree in Paderborn and 11,8% (20%) somewhere else in Germany. The majority of 64,7% (60%) earned their degree outside Germany. Hence, we have a high degree of diversification, which is an argument in favour of better generalizable results.

Participants of the PG who filled out questionnaires: 17 (10)
- Where have you earned your Bachelor's degree?
  - 23,5% (20%) in Paderborn
  - 11,8% (20%) somewhere else in Germany
  - 64,7% (60%) outside Germany
- How much time did you spend on PG work per week?
  - Approximately 15h 94,1% (100%)
  - far less 5,9% (0%)
  - far more 0% (0%)
- Which discipline matches your project best?
  - Software Engineering 100% (100%)
  - Algorithm Design 0% (0%)
  - Networks and Communication 0% (0%)
  - Computer Systems 0% (0%)
  - Intelligence and Data 0% (0%)
- I was looking for a PG associated to
  - Software Engineering 94,1% (100%)
  - Algorithm Design 0% (0%)
  - Networks and Communication 0% (0%)
  - Computer Systems 0% (0%)
  - Intelligence and Data 0% (0%)
  - I had no preference 5,9% (0%)
- I joined my preferred PG
  - Yes 94,1% (100%)
  - No 0% (0%)
  - I had no preferred PG 5,9% (0%)

**Tab. 8.8.:** Participants Overview from Questionnaire. XX (YY), XX = 1. Survey, YY = 2. Survey

All participants, except one who has no preference, have joined their preferred project group. In addition, the participants consider the project group as part of the *Software Engineering* focus area, which suggests that the techniques from the other focus areas we use have not come to the foreground too much and that everyone with a preference joined their preferred focus area (see Table 8.8).

In general, the participants spent about 15 hours a week on the project, which, as already mentioned, is the best possible result for us, as it corresponds exactly to the desired workload. Around 3/4 of the participants answered for the difficult level with three (neither too easy nor too difficult) and around 1/4 with four (difficult), which is as well a good result as it indicates that the level of difficulty was generally reasonable, well accepted and only slightly more challenging for some (see Figure 8.12).

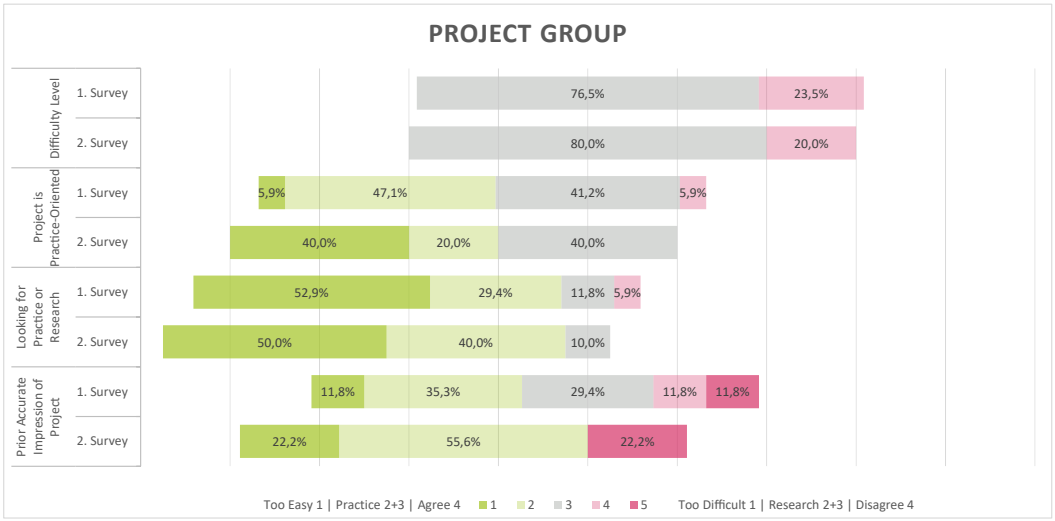



**Fig. 8.12.:** Evaluation Chart Summary General

Around 80% to 90% of the participants looked for a practice-oriented project group, which they have mainly found according to their answers as around 50% to 60% consider this project as research oriented (see Figure 8.12). Around 40% do not determine whether the project is more practice-oriented or research oriented. One thing that stands out is the increased number of responses in the first survey, which assess the project as less practice-oriented than in the second. This may be due to the attitude of some participants that the requirements work in the first half is considered less practice-oriented than the actual implementation work, as can be assumed from the *Further Feedback* section and discussion of the results with the participants. Overall, the results speak for a more natural setting mimicking a software development project, which is positive for generalizability and also for the research methodology that seem to not disturb the project too much.

Furthermore, a relative majority (47.1%) in the first survey and an absolute majority (77.8%) in the second indicate that they had a more or less accurate impression of how the project will be like. We have no conclusive explanation for the fact that in the second survey significantly more participants spoke out in favour of an accurate impression of how the project will be like. This could be due to the fact that the first group started with the idea of History in Paderborn App (cf. section 8.1) and did not know that it will be changed to the

OWL.Culture-Platform. Or it could be because in the second half the participants rated the development work as what they expected.

With the overview of the participants we can turn to the advisors as the next part of the questionnaire. In Figure 8.13 we have visualized the results for this part. There's a special thing here. For all shortcodes preceded by the ¬ character, we display the results in reverse order. So 1 is switched with 5, 2 with 4 and a 3 remains a 3. We do this where we have classified a 5 as a good answer (see section 8.3.3.1). This makes it easier to read the diagrams, as now all positive rashes for the case study are on the left side.

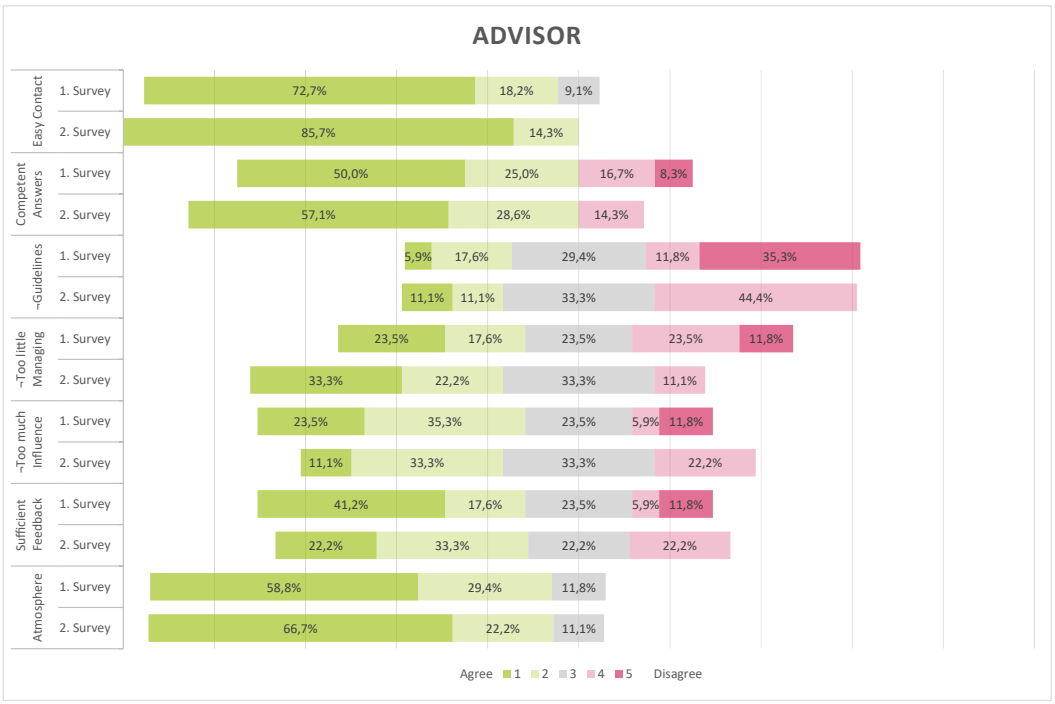



**Fig. 8.13.:** Evaluation Chart Summary Advisor

A clear absolute majority indicated that the advisors were both easy to contact and gave competent answers. A relative majority of the participants think that the advisors could have given more guidelines, whereas in the first survey even more stated it as a 5 and not a 4 (with regard to ¬). This indicates that the participants lacked orientation, especially in the first half. In the discussion of the results, participants made it clear that they would have liked to know better what tasks are to be performed. However, especially in the first half, there was a plan that was broken down into tasks / milestones on a weekly basis and presented during the onboarding (see [Sen20g]). Therefore, we would conclude that it was due to miscommunication or tasks unfamiliar to the participants (cf section 8.1, where the participants put on record that they understood and appreciated design thinking and its steps only in the second run).

For ¬*Too little managing* and ¬*Too much influence* we will also first consider the values 4 and 5, because as mentioned in section 8.3.3.1, we classify the values 1 to 3 as positive (keep ¬ in mind). In both cases the participants were more negative in the first survey. The exact reasons for this are unknown, but it may have been due to the larger group size. However, these negative responses were limited to 11.1% to 35.3%, and by far 64.7% to 88.9% of the respondents saw the influence and the managing of the advisors positive.

As stated in section 8.3.3.1, the three questions ¬*Guidelines, ¬Too Little Managing, and ¬Too Much Influence* have to be seen as package in regard to interpretation of the influence of the advisor/investigator from the participant's perspective. For all three, the answers in the majority of both surveys are in the range 1 to 3, which suggests that the influence of the advisors/investigators on the project was good in the sense of implementing the approach, but it was not yet optimal. Since this case study does not focus on comparison with other approaches, but on feasibility, we interpret the overall result as positive for it. Positive was as well the atmosphere between the participants and the advisors as well as the feedback the participants got from the advisors.

The participants answered that the following tools were used: Jira, Confluence, GitLab, Slack, E-Mail List, OpenStack, Rancher, Auth0, Indigo. All of them were mainly managed by the participants and, except for Indigo, provided by the advisors. The tools provided by the advisors were considered useful (82.4% respective 90%).

The next part with *Tasks and Teams* (see Figure 8.14) is the last part that looks at the conditions of this case study. The tasks were understandable. The fact that 76,5% of the participants think so in the first survey and 90% in the second as well as the assessment that the requirements engineering tasks in the first half were not seen as practical underlines our theory that the results of ¬*Guidelines* are related to the unfamiliarity of the tasks for the participants. Furthermore the participants were satisfied with the workload (76.5% respective 80%) as well with the responsibilities (76.4% respective 90%).

In regard to the team dynamics it was stated by the participants that the group's internal communication and self organization worked well (82.4%), the atmosphere between group members was pleasant (100% respective 90%), the other group members provided sufficient feedback (76.5% respective 100%), and that they are generally satisfied with the other members (82.4% respective 80%).

In summary, neither the advisors nor the tools, tasks, or team dynamics should have a negative effect on the case study.

With the consideration of the conditions of the case study, we can now come to the relation of the participants with the approach (see Figure 8.15). ¬*Only learned Little* was agreed by a relative majority in the first survey (47.1%) and an absolute majority (60%) in the

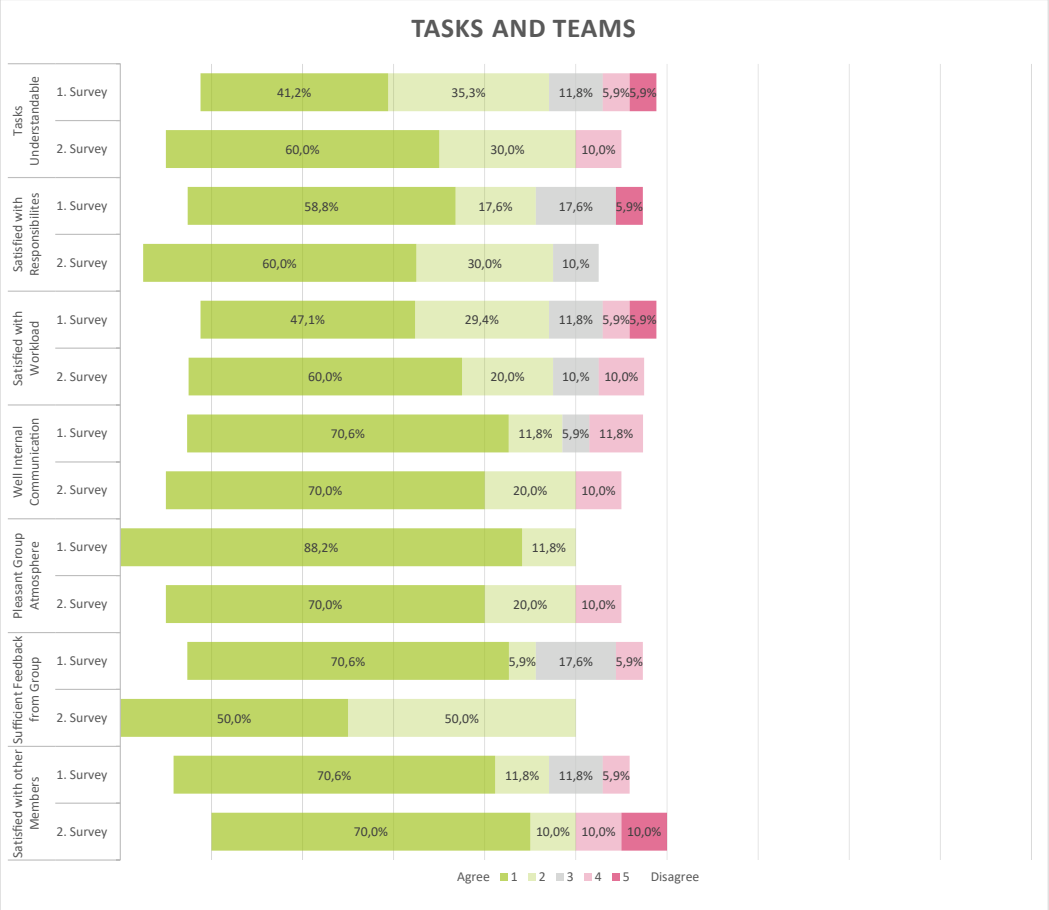



**Fig. 8.14.:** Evaluation Chart Summary Tasks and Teams

second survey, which indicates that this kind of approach was novel to the majority of the participants and is positive as discussed in section 8.3.3.1. The approach and the experience gained through carrying it out are also seen as useful (76.4% respective 90%).

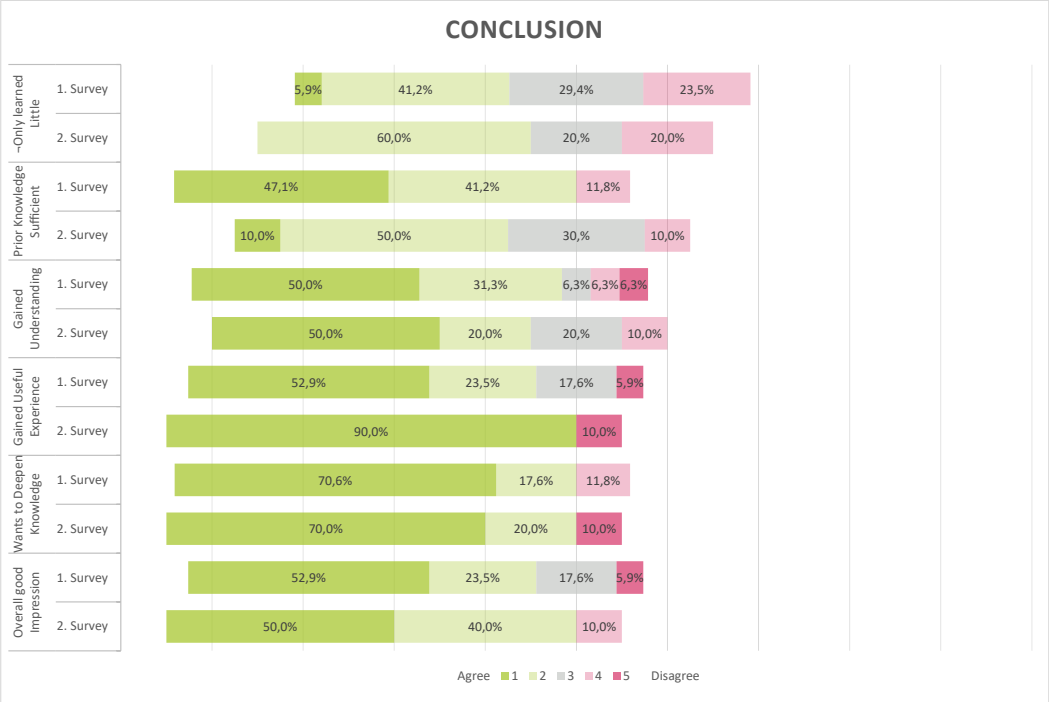



**Fig. 8.15.:** Evaluation Chart Summary Conclusion

Furthermore 88.3% respective 60% agreed on that the prior knowledge was sufficient to participate in the project. This means that the given information was sufficient to work with the approach. In addition, these were sufficient to give the participants the impression that they had developed a deeper and sounder understanding (81.3% respective 70%).

The overall impression of the project was good (76.4% respective 90%) and the participants want to deepen their knowledge in this field (88.2% respective 90%). In summary, it can be assumed that the approach was new for the participants, could be learnt with the information given, proved useful to them and was seen as positive.

#### 8.3.4.2. Work Product Evaluation Results

As the questionnaire results indicated, inexperienced developers are able to work with our approach with the given information. With this in mind, we now focus on the quality of the results and the interaction of the individual stages with the help of the evaluation of the individual work products.

**Work product #1:** *3 Ideas for the OWL.Culture-Platform* As already stated, this work product is not a prerequisite for the further stages and is mainly analyzed to determine the extent to which the participants are already able to distinguish between problem and solution as well as user needs and technical needs. The analysis was performed as described in section 8.3.3.2 for this purpose.

The first part of the analysis examines the POVs as a basic mean of describing user needs detached from technical solutions. From the 18 participants, two participant marked POVs explicitly. 12 participants (including the two marking them explicitly) used the proposed structure *[**User**...(descriptive)] needs [**need**...(verb)] because [**insight**...(compelling)]*, whereas only 9 used it continuously. Within the ideas of four participants, it was not possible to recognize each of the three properties *User*, *Need*, and *Insight*. Six participants only used the word *user*, another six participants used an umbrella term for a specific user group, and another five participants used a description for their users. One participant failed to include user at all. For four participants it was not possible to extract something that could count as a need. The rest, except two, try to solve the need of the corresponding user or another. Of these 14 trying to solve the need of the corresponding user or another, seven use an action as need, whereas five use a description. In addition to the 12 participants who used the proposed structure, insights could be identified for another participant. From these 13 participants with insights, eleven used them consistently to describe challenges, gaps, limitations or deeper needs.

In summary, the three properties *User*, *Need*, and *Insight* can only be found in the ideas of the participants who have used the proposed structure. This means that each of the these three properties are consistently found only with nine of the 18 participants. Furthermore, the quality of the POVs vary greatly. In regard to users, only five participants described their users in detail. From the 14 of 18 participants describing a need corresponding to a user, only seven participants described the need as action. Only 13 of the participants described supporting insights, eleven of which consistently used them for challenges, gaps, limitations or deeper needs. Even if this status is not bad in respect of previous knowledge, it is not sufficient to be able to work reasonably with the approach. Without properly defined *User*, *Need*, and *Insight* it is not possible to understand as developer the intention of the user and constraints as well as opportunities for possibly different solutions. Accordingly, this makes experimenting more difficult and thus better understanding of the problem.

The next part of the analysis is the separation of problem and solution space review. Only one participant explicitly named its parts in problem and solution and for additional two participants it was comprehensible due to text structuring. For the remaining 15 participants this was not present or implicitly present. More precisely, four of these 15 participants described only solutions and 11 participants only problems. Furthermore, no one made an

explicit connection between problem and solution space and no one made a clear separation as parts of the solution or the problem could be found vice versa if both were present. This is a rather disastrous result.

**Work product #2:** *Interview Guideline* In the previous work product we were able to analyse to what extent the participants are able to distinguish between user needs and technical needs. In this part it is primarily about analyzing to what extent the participants are able to capture user needs. As described in the concept (see section 8.3.2), the 18 participants in this phase work in six subgroups of three persons each. Therefore, we have six interview guidelines to examine, starting with the 18 decisions.

Regarding the first decision about the research object, all six groups consider only the content-related facet. Therefore, at this point only the content-related facet can be analyzed and not the theoretical-methodological facet. All six groups do not further describe the context, but only name it in one sentence, which concludes a further analysis on this point.

For the next decision about the target group, all six groups described their target group more or less only with umbrella terms. Three groups additionally defined the age range they are interested in. Furthermore three groups defined the number of interviewee they require and not a single group made a link to the research object.

Regarding the interview form, three groups decided for a guideline interview, two groups for a problem-centered interview, and one group for a scenic interview. Not a single group made a link to their preceding decisions. With regard to the groups with a problem-centered form of interview, it is questionable whether this fits the current situation. For this, an exact problem with a previous level of knowledge is assumed. However, this is not guaranteed at this stage.

For the evaluation strategies, two groups introduced intermediate steps whereas four groups only described what should be evaluated as end results. The two groups that have described intermediate steps have decided for a purely qualitative study. For the other four groups, two have also decided for this and the other two want to carry out an additional quantitative study.

The remaining decisions relate mainly to general aspects and not to the previous main decisions. On the positive side, all groups have decided that, in addition to an interviewer, at least one transcriber should always take part in the interview. However, the planned time per interview is negative. Three groups plan only 5-10 minutes, two don't give any information and one group plans 15-45 minutes. In our experience, at least 15 minutes including activation are necessary to build and validate a deeper understanding.

In summary, the 18 decisions of the groups can not ensure that they are getting to the bottom of latent user needs, explore surprising, and develop empathy. The quality is simply not high enough for this. In many cases, the links to previous decisions and more precise descriptions are missing. On the other hand, the quality should still be sufficient to get these answers.

That leaves us with the question set. Three of the six groups actually introduced an activation phase, whereas the scenic group realized it via storytelling in a slideshow and the other two via structure that begins with general questions and build up to more specific ones to the research object towards the end. Although all groups focus on getting factual information, the questions have the potential to build a deeper understanding of the interviewees. However, this depends primarily on the interviewees and their situation-specific follow-up questioning, even though all groups have defined follow-up questions. The problem with these, however, is that they are only very superficial and only go one level lower.

The analysis of the interview guidelines shows that all groups are prepared for the interviews and have the potential to build empathy. However, the quality does not correspond to one that would ensure that latent user needs, surprising, and information regarding empathy are actually captured. Much depends on the individual skill for in-depth follow-up questions.

**Work product #3:** *Interview Results Summary* These are the results of the interviews of the individual groups, in order to inform the others about exactly these so that they can also profit from them. To start with how helpful they were, we will look at the context information about the conduction of the interviews first.

From six groups, four groups presented their results on the basis of the questions, resulting in what the interview asked. The group with the Scenic Interview presented the results of the interview before, so that the content of the interview is also clear for this group. This leaves one group that did not describe how the interview was designed.

Furthermore, five of the six groups have broken down information about the interviewees. All of these five groups indicated the number of interviewees, which was between 8 and 12 with an average of 10.6. In the case of four groups, it can be seen from the description that they were local and international students. In addition, three of these groups also indicate which subject areas they come from or that they come from different fields. One group only gave information that they were students and another group did not give any information about the participants.

Just three groups gave information to the duration of the interviews with an average of 8 minutes, an average of 10 minutes, and 10-15 minutes. One group deviated from their planed interview duration by having 5 minutes longer interviews. In general, however, no group has mentioned deviations from the interview guidelines.

With the analysis of the context information, we now come to the results themselves. Actually all six groups present their results in a condensed fact form. Only one group also provides background stories of the interviewees.

In summary, it can be said that the groups in this work product tend to present results immediately condensed as facts. By immediately removing the life realities of the individual interviewees, important information is also removed for further decisions. This way, the problem space is severely restricted at a very early stage, which can have a negative effect on further solution finding. Nevertheless, five of the six groups have created transcripts in addition to the presentation in which the individual life realities are still preserved. Therefore, the negative consequences are likely to be limited.

**Work product #4:** *Low Fidelity Prototype* Five of six groups are using underlying POVs, whereas the other group uses a scenario description / user stories. Of these five groups, four have used the proposed sentence structure, while one group uses a table divided into user, need, and insight. Three groups that used the proposed sentence structure did not consistently define the insights for every POV. They as well only used the term *user*. The other two groups used an umbrella term. The one group that did not use POVs have at least a user description included and describes the needs of the user as actions. For the rest of the groups, three use their needs solely or mainly as description and two groups describe their needs as action. The quality and usage of the POVs correlate with that of work product #1. In addition to the POVs, two teams used user stories to describe their idea. Only one group had a comprehensive problem definition that included user group and context, but had no POV.

As prototyping techniques all groups used wireframes, whereas four groups used the digital variant and two groups used the analogous variant. The extent to which they adhered to the first do (focus on understanding the problem and possible solutions) is difficult to evaluate for most groups due to the quality of the POVs and the lack of problem definitions. However, what can be stated here is that all groups have focused solely on the information architecture and navigation and have not on other parts not necessary at this point. Furthermore all low-fidelity prototypes are tangible and can be used either by clicking on it for most digital wireframes or by changing to new screens or adding sticky notes. Not a single group presented their prototypes as finished product (e.g. by incorporating functionality or focussing on the look) and all prototypes are self-explaining.

The last part of the analysis to what extent a connection between POV, idea, and prototype has been established, has shown that no group made this direct connection. However, all POVs can be found in the wireframes.

In summary, it can be said about this work product that the participants were consistently able to focus on the essential parts of the prototype without adding unnecessary parts. Accordingly, the first prototypes can be tested on the value level at this point at a lower cost than software prototypes. The problems with the uncleanly defined POVs are reflected in the quality of the problem definition and thus also in the prototypes of this stage. Without clearly defined POVs and problem definitions, it is difficult to develop appropriate test scenarios for how the solution will presumably be used as well as to focus on what is actually needed in the solution.

**Work product #5:** *High Fidelity Prototype* Three groups used dedicated mock-up / visual prototyping tools for their high-fidelity prototypes whereas the other three groups decided to use HTML / CSS web frameworks with backend functionality. The dedicated mock-up / visual prototyping tools used are Adobe XD, Balsamiq, and Indigo Design. These are characterized by the fact that each individual change is represented by a new screen, similar to the wireframes. In addition, they allow for more extensive representations than just the wireframe representation. Usually each of these tools offers a navigation between the screens with the help of hitboxes. For example, hitboxes can be defined for areas of a screen that perform an action during activation, e.g. the display of another screen. Screens are usually used with fixed widths and heights, so they are not responsive. However, this has the advantage that the additional complexity and the boilerplate code for the responsiveness are omitted. Special features of the tools can be e.g. automatic animations, reusable components, components with states, generated replication of components, or also the export and use in different end devices like a PC with a web-browser or as app in a smartphone. Such tools are optimized to create a visual representation of the prototype, which feels like a real product but has no functionality yet, fast and resource-saving. Accordingly, they limit themselves to the *value* and *look & feel* level and allow interaction but no further functionality or domain logic. These tools usually require no to only a bit boilerplate as you can directly draw your interface or import your graphics.

In contrast to these tools are dedicated HTML / CSS web frameworks with backend functionality. These are used for prototypes if you want to reuse code and evolve the prototype. The advantage of this approach is the reusability of components. Some web frameworks, such as django[7] are also optimized for rapid prototyping. In this case, rapid prototyping does not necessarily mean that it is faster than the tools presented above, but that it is faster and easier to set up first prototypes compared to other web frameworks, since they typically generate a lot of boilerplate code automatically.

[7] `https://www.djangoproject.com/`

In our current phase, however, such HTML / CSS web frameworks are not advantageous. In order to be able to play out the advantage of reusability, many components are required which will also be used in other prototypes in this form in the future. However, these prototypes are about building understanding and trying out different possible solutions, which is why most likely many components will not find their way into the final product or will find their way in a strongly changed form (cf. [Koh+09] and section 1.1). Additionaly, they limit the solution space by the technical feasibility. If they are rapid prototyping variants, only the prefabricated components can be used; if they are normal frameworks, the solution space that can be tried out is limited solely by the implementation effort. These tools still require a lot more boilerplate than the tools presented above, e.g. for correct layouting or just for setting up the project. Furthermore, they do not encourage to concentrate only on *value* and *look & feel*, but also to implement functionalities.

With the tools analysed, we can come now to the prototypes itself. The three groups that used dedicated mock-up / visual prototyping tools have and could only incorporate *value* related concepts and *look & feel*, but not add functionality. In contrast to that, all three groups that used the dedicated web frameworks already started to add functionality that was not mocked with a Wizard of Oz kind method but implemented in a rudimentary way. For all six prototypes it holds true that they are based on the previous low-fidelity prototype of the respective group.

All six groups reached a high immersion level regarding their prototypes. They used one design language consistently throughout the entire prototype. Furthermore, no prototype required manual intervention to interact with it and the transitions are running smoothly. The information presented in each prototype was complete and related to real world data. All prototypes can be used outside the tool, e.g. in a web browser. Moreover, in addition to the PC-optimized prototypes, two groups (dedicated mock-up / visual prototyping tools) have also designed a prototype as a mobile version, which can run directly on mobile devices.

Of course, the layout, information structure, and visual language can be reused for all prototypes. But in case of the groups using the web frameworks, the visual elements as well as the functionality has been designed to be reused in further prototypes and have been as well propagated like this by these groups. Hence, the three groups designing their prototypes with dedicated mock-up / visual prototyping tools have designed it as throw-away prototypes whereas the other three groups designed it as reusable prototype that shall be evolved to a product.

In summary, all the groups have managed to build a prototype that feels already like a real product, whereas one half didn't implement any functionalities and the other half implemented the functionalities in a rudimentary way. In terms of the amount of ideas implemented, the groups don't differ significantly, but two groups that used the dedicated

prototype tools managed to work out visualization and interaction concepts for a mobile variant in addition to the desktop variant.

**Work product #6:** *Fair Feedback* Only one of the groups included the learning card in their documentation and that only for the internal fair. The same group used the given structure of the feedback grid within their documentation of the internal fair feedback. Furthermore, for the internal fair feedback two additional groups used a subset of the feedback grid with *Things I like most* and *Things that can be improved*. This same structure has been used by two groups for the external fair, whereas one group additionaly documented ideas. The rest was unstructured feedback mainly regarding improvements / bugs and occasionally parts that are not understood. It has not been indicated that the groups have no further data for the other two parts of the feedback grid.

The groups have rather rudimentarily written down their results in this work product. A systematic, target-oriented observation can still be read out for the internal trade fair in an isolated case, but no longer for the external one. The focus of most of them was above all on the improvements and partly on the things that are already good. All in all, this work product in this form does not allow much learning, which contributes to a better understanding of the needs and challenges of the users.

**Work product #7:** *Agile Software Requirements* As the results from the *Transformation Framework* evaluation show, the participants using the framework had been successfull in keeping the design thinking results, derive from them agile software requirements, and sustain the connection between both. The derivation of the minimum viable product was also successful. In the Epics, which overlapped in several groups, the findings from the corresponding groups were used (e.g. by including the screenshots of the different prototypes) and the connection to the results from design thinking / *Transformation Framework* was maintained by including links to the corresponding wiki pages und references.

**Work product #8:** *Minimum Viable Product* The first thing we analyse is the nlp pipeline as foundation for the development of the minimum viable product in this case study. Its components were developed in different technologies communicating together over HTTP REST-APIs defined with OpenAPI. To be specific about the technologies, the crawler and the data store were implemented in C# with ASP.NET Core, the data enricher was implemented in Java, and the recommender in Python. All of these components have been packed individually into a container (Docker) and are deployed via a CI/CD pipeline from GitLab to a Rancher controlled Kubernetes Cluster (see Figure 8.16). The discovery of the components in the cluster takes place via the kubernetes own DNS system and service

definitions created for it. They are accessible through Traefik via normal URLs defined as Ingresses to the outside world . Communication between the components is event-driven. However, the observer pattern is not used, but each component is initialized with the end points of the components required for this component. This cannot be changed during runtime in the component, but by using the service definition in the kubernetes cluster the routing to the corresponding instance can be controlled during runtime. In this way, the components of the backend are technology-independent and can be exchanged without other components noticing if the API stays the same.

Even if event-driven communication has been used, it does not automatically mean that it is particularly suitable for event sourcing or beneficial for parallel models. In this case, there are the three main events *CulturalEventCreated*, *CulturalEventDeleted*, and *CulturalEventUpdated*. The challenge with these is that they act like a central database schema. All properties that are used by the single components have to be represented in these events as all components use them. Therefore, all adjustments to the schema must also be made in all components. This in turn is counterproductive for parallel models and an independent development of individual components. In this case study this has been shown several times, among other things, when it came to changing a date format that in the end had to be changed in all components under the increased effort of the different technologies. It would have been better if the events had been defined on a more fine granular level, for example *EventDateChanged* and *EventLocationChanged*. Like this, each component would only have to listen to the events required for them and could introduce new events without considering all other components.

In the next step, we will examine the frontend and the interaction between it and the nlp pipeline as the backend. First of all, the frontend was realized with Angular (Angular 7.x) that uses an architecture consiting of *Templates*, *Components*, and *Services*. The *Templates*, which translate to *View* in the MVP or MVVM pattern, actually consists of HTML Templates that can use every HTML framework and include data via property bindings on the component (see Figure 8.17). Changes to these properties are propagated via the event binding. The *Component* includes logic specific to the views and get data injected as well as non-view specific functionalities by *Services*. *Components* act as *Presenters* or *ViewModels* as we are using bindings. The *Services* on the other hand are 'encompassing any value, function, or feature that an app needs'[8]. In this sense, a *Service* is more than the *Model*, but in this case it is used as well for getting and pushing data to the backend. For example, the participants implemented different interchangeable service to either mock data or getting it from the backend. Hence, with the use of Angular a clear implementation of the MVP respectively MVVM pattern is given.

[8] `https://angular.io/guide/architecture-services`
[9] `https://angular.io/guide/architecture-components`

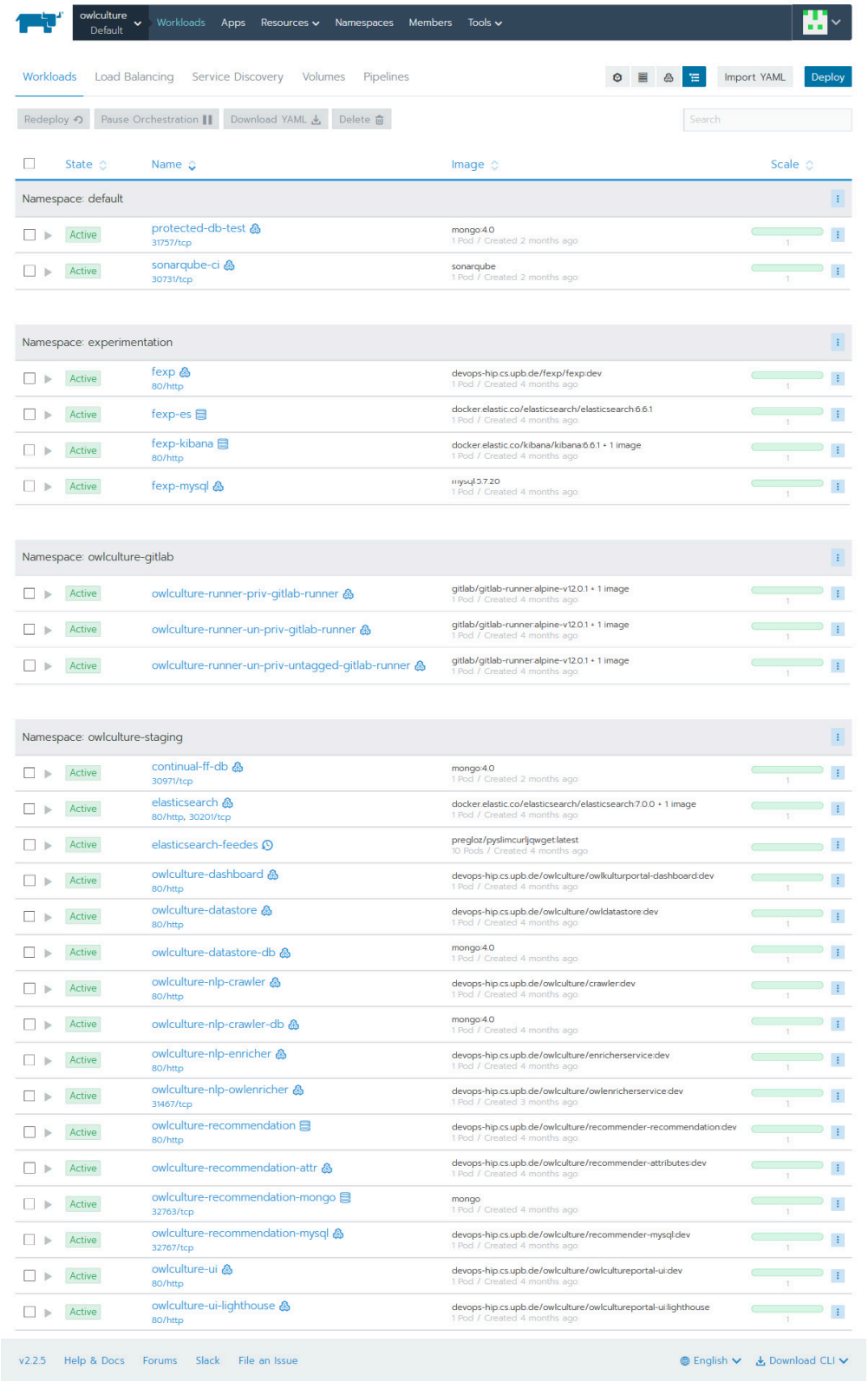



**Fig. 8.16.:** Minimum Viable Product in Rancher controlled Kubernetes Cluster

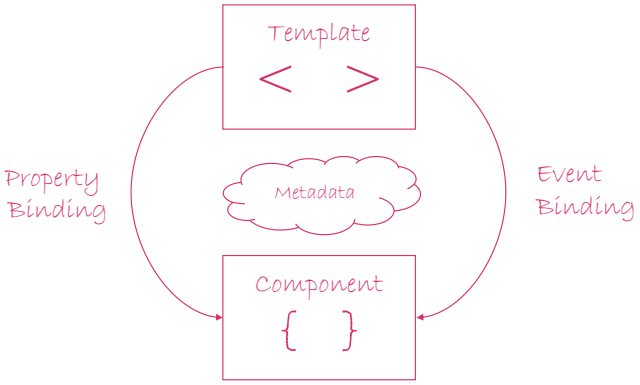



**Fig. 8.17.:** Relation between Template and Component in Angular. Own representation based on Angular Docs[9].

This has several advantages for us regarding the detached development of the frontend from the backend, the adaption, and the simultaneous operation of alternatives. In case of the detached development, the *View* doesn't need to consider the data structure of the backend as it can be restructured for it by either the *Service*/*Model* or the *Component*/*ViewModel*. That this is working is showing the use of different *Services* by the participants to either mock a data source or using the backend directly. This enabled them to start developing and trying out the *View* before all changes have been made to the backend. Furthermore, the way Angular implements the *Components* allows you to use them as web components or simply load web components and fill them with data from the *Services*. With this, we can load components online and as well mix different web technologies. Hence, with the replacement and online loading of web components we achieve easy adaption in the future as well as the simultaneous operation of different alternatives.

In summary, the analysis of this work product shows that evolutionary software development is feasible in this case study. This worked particularly well in the front end area, probably due to the forced implementation by Angular. At the level of the backend, however, problems arose with the correct implementation of the event sourcing pattern as it was already with the previous project groups (cf. section 8.1). This restricted the evolutionary capability of the system as well as the simultaneous operation of different alternatives.

**Work product #9:** *Experiment Definitions* The participants defined in total 11 experiments in *FEXP* (see section 7.3.1) and Jira (as it allows more information than in *FEXP*). An example of such an experiment is the one about the *Scrollview on Landing page*. The underlying challenge is that "*Users don't recognize that landing page is scrollable*" with the two possible treatments (variants) (see Figure 8.18) to have the imageview of the landing page set to 75% instead of 100% or to add a scroll arrow at the bottom. The hypothesis of the experiment is "*Having a solution for showing that there is more content on the landing page than just the*

*search bar helps the user and doesn't lead to skip the "unseen", both solutions meet user's needs in UX and functionality.*". Whether the experiment participants recognized further content and how they perceived the functionality is what is to be tested. To do this, a set of quantitative and qualitative questions are asked. Like this, all experiments are defined. Hence, all experiments have a hypothesis, treatments, and evaluation criteria defined.

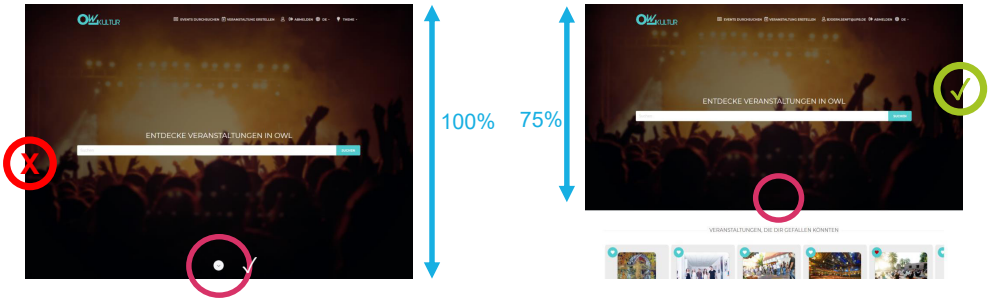



**Fig. 8.18.:** Example Experiment: Scrollview on Landing page

From the eleven experiments, ten experiments defined their hypothesis in a cause effect relationship format, which is good. One hypothesis is not really a hypothesis as it only describes an action. All eleven hypotheses are on a granularity level that their treatments (two per each experiment) only cover one level in a software component hierarchy or just one small software component. This means that the hypotheses have already a high granularity which is desired in the optimization stage of this approach but not in this since the questions still relate primarily to the meaningfulness of the intended value. Nevertheless, there is a chance for further information through the choice of a quantitative and qualitative survey to capture the evaluation criteria. All parts defined in the single experiments relate to each other or are fitted together. Summed up, the participants were able to define different experiments although they are not yet on the desired granularity level.

**Work product #10:** *Experiment Implementation* To conduct these experiments, they have been wrapped up in a story that leads to single tasks which will trigger a single experiment. The tasks itself are described from a user perspective and not in a way like "*now press this button*". The story is divided into two parts that can be used independently of each other. In the first part, half of the treatments are tested and the other half in the second part. Depending on the length of the experiments, both treatments can be tested this way without repeating the tasks.

Furthermore, the story with the tasks is handed out to the experiment participants on paper. This paper also includes the questions on the individual experiments at the corresponding points. A consent relating to the personal data and a general questionnaire about the experiment participant is as well included. The general introduction and farewell was carried out

by the advisor / investigator for all test participants simultaneously and supplemented by the participants for the individual experiment participants.

For data recording the participants are using paper questionnaire as well as screen and audio recording. For the screen and audio recording ten Laptops have been prepared with corresponding capturing software as well as with an offline version of the to be tested software in case the internet connection breaks. The functionality of the software and recording capability was tested by the participants for all laptops the day before. In addition, the experiment participants were instructed to think aloud. Besides this, no additional notes have been made.

The assignment of the treatments is done via feature branching in code like it already uses *FEXP*. However, this is done manually and does not use *FEXP*, because *FEXP* currently only assigns treatments randomly according to user ID and cannot be influenced from outside.

In general, the participants conducted a well performed usability test with different alternatives. They have issued tasks that lead to certain functions being used or highlighting the fact that these functions cannot be found. A user-centered task description was used. With Thinking-Aloud, Screen Recording, Audio Recording and Questionnaires they used the very common instruments of a usability test. This will also allow them to dig deeper into problems that occured and gather additional feedback in the analysis.

To implement the different experiments with their treatments the participants logged in total 173.5 hours. On average implementing one experiment took 15.75 hours, whereas the median is 4.85 hours.

**Work product #11:** *Experiment Analysis* The strategy for the experiment analysis consists primarily of analyzing the questions. An analysis of the screen and audio recordings is hardly or not at all included. In total the analysis consists of two steps, where in the first step the answers were collected in a table. To be precise, there are two tables. In the first table the answers are combined to one half of the treatments and in the second half to the other half. If the answers are not free text answers, all statements can be assigned pseudonomized to the test subjects. The second step involves the aggregation of the results into the two treatments. The result of this step is a document with an overview of the experiments, the associated treatments, the results, and corresponding suggestions.

Whereas the analysis of the questionnaire was well performed, the participants missed the opportunity to gather or document additional insights via the screen and audio recordings. Due to the thinking aloud, it is very likely that additional information can be found here that is not requested by the questionnaires, even though a total of 168 questions were worked out.

### 8.3.5 Summary and Discussion

In this section we have presented our case study with the OWL.Culture-Platform as case to investigate the feasibility of the overall approach. For this purpose, we have used a student project group (participants) that has gone through the various stages of the approach over a period of one year. Since we did not assume any prior knowledge of the approach among the participants, additional elements were carried out in order to bring the approach and its background closer to the students. In order to assess the feasibility of the approach, this case study relied primarily on questionnaires and the assessment of the work products of each task as evaluation instruments.

The questionnaire serves on the one hand to record the participants' view of the approach and on the other hand to measure potential negative influencing factors related to the case study. For this purpose, among other things, the topics project affinity, influence of the advisors / investigators, tasks, team, tools, degree of difficulty, and impression of the approach were asked. The participants needed only the estimated workload, the level of difficulty was generally reasonable, well accepted, and only slightly more challenging for some. Overall, the results speak for a more natural setting mimicking a software development project, which is positive for generalizability and also for the research methodology that seem to not disturb the project too much. The advisors / investigators role with regard to influence and managing was seen positive, but it was not yet optimal as participants lacked orientation, especially in the first half of the case study. Furthermore, the participants got the tools they needed and considered the tools provided by the advisors useful. Tasks were understandable and appropriate in scope. With regard to the team dynamics, the participants also expressed themselves positively. Accordingly, in this case study, no excessive confounding effects by the supervisors, the tasks, or the participants are to be expected.

The participants were mainly positive about the approach. They said that they considered the approach to be new to them. They also think that they have gained a deeper understanding of the approach and that the gained experiences are useful. Overall the participants state that they have a good impression of the approach and want to continue to work in this area.

The questionnaire results show that, from the participants' point of view, the case study was carried out well and the tension between supervisor and investigator was satisfactorily resolved. In addition, the participants understood the approach, found what they had learnt to be useful, assessed the overall approach positively, and wanted to deepen their knowledge in this area. This speaks for the feasibility of the approach from the participants' point of view.

Whether the feasibility at the content level is also given was checked by the evaluation of the individual work products. A total of 11 work products from the various stages of the

approach were used for this purpose. In general, this evaluation has shown that the approach is also feasible at the content level and that the interaction between the individual stages works.

However, it has also revealed some weaknesses in the approach. About half of the participants had difficulty formulating requirements from the user's point of view. The main problems were the weak separation of problem and solution space, the description of users, and the inability to describe users actions and not structural requirements. As a result, the solution space is restricted at a very early stage and underlying problems are not comprehensibly documented. Apparently, some developers tend to specify the solution as early as possible and to avoid potential additional work (cf. point based engineering, Denning et al. [War+95] and section 1.1.3). This was also evident in the high-fidelity prototypes, where two groups tried to implement functionalities with web frameworks in order to ensure reusability and reduce potential additional work, but could not have these effects in the corresponding stage.

That the developers try this is not bad per se. They may be experts in more technical areas of software development and accordingly their quality is to think exactly in these areas. Accordingly, the involvement of such persons should be carefully considered. Two possible ways would be less involvement in the first three stages or more steering by appropriately trained personnel.

In the case of less involvement, these persons could, for example, only participate in a dedicated design thinking workshop like our instance (see section 5.2) in order to be sensitised to the context of use. Further requirements work could be done by the value designer and only when the value is set (e.g. technical prototypes in stage 3), these technical persons start to work on it.

If these people are to work continuously, steering by appropriately trained personnel is important. The value designer should have a strong empirical und user centered background and needs to continuously monitor the approach. She must ensure that these people focus theirselves on the user and deliver high quality results although it is not their area. This includes as well guidance regarding interview conduction, or prototype creation, experiments, and empirical evaluations.

Another challenge was the implementation of the technical requirements resulting from the experiments and evolutionary systems. In principle, these were implemented, but not necessarily in a good quality. Especially the implementation of event sourcing proved to be problematic. The participants oriented themselves far too much on active records, in which a state is persisted and the state is modelled accordingly. They tried to do the same with the events in event sourcing instead of getting to a much finer granularity. Without

already having extensive experience in the implementation of certain architectural patterns, the implementation will always be problematic. One solution could be an architecture expert who reviews each pull request accordingly. However, this may not be feasible in terms of time. Another promising solution could be automated checking at IDE level or in the continuous delivery pipeline with corresponding feedback. For example, the introduction of Lighthouse[10] with its score system in the continous delivery pipeline has led to participants reducing the technical debt of the frontend and introducing fewer new ones or eliminating them quickly. Accordingly, we need to evaluate how such automated checks could look for our system architecture and introduce them into the continuous delivery pipeline, e.g. as a Fitness Function (cf. [FPK17]).

[10] `https://developers.google.com/web/tools/lighthouse/`

# 9 Epilog

In this chapter, we are summarizing the overall results of this thesis in section 9.1. Furthermore, we discuss the results in section 9.2 before giving an outlook on future work in section 9.3.

## 9.1 Summary

In chapter 1, we have introduced you to the challenges of developing unique and novel software-based solutions, especially at the domain level. The main issue with it is the fact that we start on a greenfield regarding problem and solution space and therefore cannot be certain about the predictions we make regarding the value and proposed software-based solutions (cf. Kohavi et al. [Koh+09]). Furthermore, existing software development approaches like SCRUM seem to not fit anymore as they are optimized for incremental innovations and not unique and novel software-based solutions (cf. Norman and Verganti [NV14]). Kurtz and Snowden [KS03] have introduced a sense-making framework that helps to understand why existing software development methods are not that helpful. In short, the development of unique and novel software-based solutions means that we are acting in the chaotic and complex space in which we have to uncover the constraints and interacting dependencies by trying out different alternatives to learn from their difference. On top of that, the diffusion process of innovations make it difficult to assess potential solutions during development, which is why we have introduced the basic characteristics in section 2.1. Design thinking with its diverging and converging thinking, the separation of problem and solution space, as well as the focus on learning with the help of tangible prototypes, seems to be best suited to meet this challenge, which is why it is introduced in section 2.2.

However, the integration of design thinking with software development is unclear as Lindberg, Meinel and Wagner [LMW11] state. Is it best to use it as a front-end technique (which means that design thinking is finished before any code is written) or use it as a fully integrated development philosophy? We decided for a mixture between front-end technique and fully integrated development philosophy for our approach we call Insight-Centric Design and Development (ICeDD) and present its concept in section 3.1. One reason for this is the cost of creating software prototypes as they e.g. always need boilerplate code to be setup, restrict your ideas by the framework, or need more than general knowledge to be created.

For example, paper or lego prototypes are usually quite cheap to make for prototypes with a reduced feature set and can be created by almost everyone as we usually learnt the necessary skills for that already. This way, we can better include other stakeholders in the process and learn from their knowledge. Otherwise, they would need to idle in certain phases which can be frustrating. This brings us to the other main reason. Most stakeholders involved in this process, including the developers, are experts in certain fields, which is why they tend to prefer certain tasks over others. A software developer can be good a solving technical issues and this is what she really likes, but e. g. has little knowledge and interest in interviews and therefore doesn't want to do them as main part of her work. In order to take account of this situation, we do not force all participants to be equally involved throughout the process, but include them accordingly in the different stages. The only role equally involved is the value designer as the advocate for the value proposition. In Figure 9.1, we present ICeDD in an illustrative way. We start on a greenfield and explore the problem and solution space initially with the help of grounded theory with the result of a design challenge. The design challenge is the starting point for design thinking with non-software as medium and has the goal of further exploring the problem and solution space with the help of tangible non-software prototypes. The results and prototypes from this are taken into the stage design thinking with software as a medium to test at least two alternatives with field experiments to learn from their difference in as natural a context as possible. With finishing this stage, enough knowledge is created to switch to the optimization stage in which we can focus on incremental innovations through e.g. agile software development or online controlled experiments.

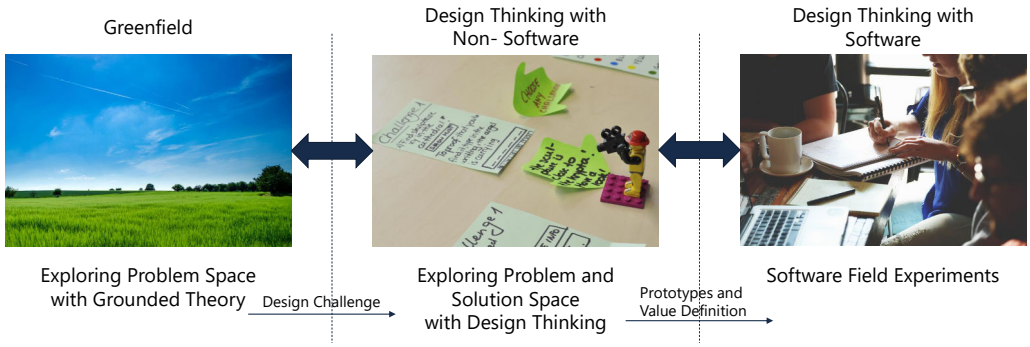



**Fig. 9.1.:** Schematic representation of our approach.

Besides an overview of our approach, we also go into details of the single stages, which are illustrated in Figure 9.2. They are *ICeDD Stage (1): Initialize Design Thinking*, *ICeDD Stage (2): Execute Design Thinking with Non-Software*, *ICeDD Stage (3): Prepare Design Thinking with Software*, *ICeDD Stage (4): Execute Design Thinking with Software*, and *ICeDD Stage (5): Optimization*, whereas the last stage is not discussed in detail as it is there to emphasize how this approach is connected to existing software development methods.

The first stage is presented and discussed in chapter 4. It became necessary because design thinking requires an appropriate design challenge, but the literature mainly deals with how to evaluate the quality of a design challenge ex post and not on how to derive a good one for software-based solutions. Based on our findings on expert knowledge, we developed an approach based on the grounded theory methodology as a theory generating approach (see section 4.3). Furthermore, we have supplemented this approach with on-site feature requests (see section 4.2) as expert knowledge is often only activated in their needed context and this way we want to ensure that we can also get the cases that have been maybe forgotten or not popped up during the systematic analysis with an analyst. Although grounded theory is a theory-generating method, its limitations lie in its validability, since it is designed primarily for observational and interview techniques rather than learning through concrete prototypes (cf. section 2.1 and section 1.1.1). Therefore, this level alone is not enough and we primarily take our design challenge from the insights we have gained in this level.


**Fig. 9.2.:** Solution Overview

The design challenge is the starting point for the next stage *ICeDD Stage (2): Execute Design Thinking with Non-Software*, which we present and discuss in chapter 5. In its essence, this stage is actually normal design thinking. However, since design thinking is also a methodology and needs to be adapted to our context, we have developed our instance based on the K12 Labs proposal for the implementation of design thinking, which we present in section 5.2. A main difference in our instance is that it is mainly designed as a 1-2 day workshop rather than a 6 week design sprint. Our goal was not to develop with it a fully finished product but create awareness and empathy for the different stakeholders as well as explore the problem and solution space further to find appropriate value propositions. This is mainly because we have observed (cf. section 5.3 for our findings in conducting design thinking) that for all our stakeholders it was appropriate to focus one or two days

on this stage, but difficult to engage in it for a longer period. The results of this stage are very experimental prototypes and additional insights, which are documented in various artifacts.

To make these prototypes and findings usable for software development we introduced the third stage *ICeDD Stage (3): Prepare Design Thinking with Software*, which we present and discuss in chapter 6. For this we have to do two things in particular. First, we need to transform the experimental prototypes and insights into an agile software specification / documentation. To do this, we must, among other things, structure them and make it possible to represent alternatives. For this purpose we have developed a transformation framework (DTRF) (see section 6.2) and gave evidence for its feasibility with a usability test (see section 6.2.4). Furthermore, in terms of the prototype level of Houde and Hill [HH97] and the recommended ux design process by Mayhew [May12], the Technical and the Look & Feel level must be populated and all levels must be integrated. This also completes this stage and provides the basis for implementing concrete software solutions.

The implementation of different software solutions or rather experimenting with different software solutions is the main part of the fourth stage *ICeDD Stage (4): Execute Design Thinking with Software*, which we present and discuss in chapter 7. As we mentioned in the introduction (see section 1.1), before we have tested something in reality we cannot be sure that we have considered all constraints and dependencies. That is why it is so important to test our software-based solutions in reality with at least two alternatives to build a better understanding of the problem and solution space based on the differences. Accordingly, we have to experiment in this stage, more precisely to conduct qualitative experiments in the form of field studies / field experiments. This means that we have to change the software development process to experimenting with at least two solution alternatives. A change in the software development process does not only mean a change in the process itself, but in the sense of 4P (Jacobson, Booch, and Rumbaugh [JBR99, pp.15]) involves the product, the people, and the project as well. Accordingly, we have divided section 7.2 into these same 4Ps, which are interrelated. The starting point is the process, as it is essential for this stage and restricts the further Ps. Based on the general experimental process *Design*, *Build*, *Run*, and *Analyze* we have refined and adapted the process for software development / qualitative experiments with software (see section 7.2.1). The experimentation itself requires certain product properties, which we have defined in section 7.2.2 in the form of a macro-architecture. With experimenting we have a situation similar to natural evolution, where different alternatives compete against each other and furthermore we have to assume at this stage that our knowledge of problem and solution space has not yet consolidated to the extent that there are no more changes. Accordingly, we need an evolutionary architecture that allows testing with different alternatives and also allows changes over time. We have found this with the *System of Systems* architecture. For the concrete realization, we have

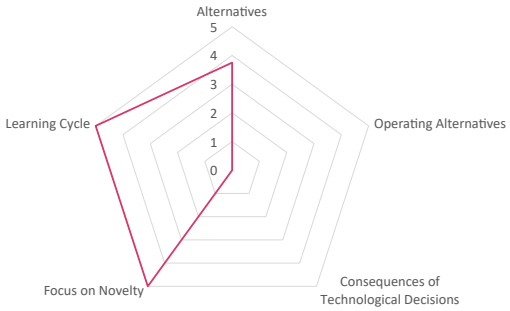
adapted established patterns from other areas of software engineering such as scalability. In addition, we present the implications for the people in section 7.2.3 and for the project in section 7.2.4. Besides the adaptation of these 4Ps, a very important point is the viability of the process. In order to provide evidence for the viability, we present and discuss tools in section 7.3 as they "[...] are good at automating repetitive tasks, keeping things structured, managing large amounts of information, and guiding you along a particular development path" [JBR99, pp.22] and therefore influence the resources and effectivity of a process. We have developed tools for process parts that had not yet tools and provided evidence for their feasibility with the help of usability tests and the implementation itself.

As mentioned at the beginning, these 4 stages are in the focus of this thesis. The 5th stage only serves as a highlighting for the connection to existing methods for incremental innovations like Scrum or online controlled experiments. To investigate the interaction of the most important stages in this approach, we have conducted a feasibility case study, which we present and discuss in section 8.3. We use the OWL.Culture-Platform as a case and have a first prototype of it developed by a student project group in computer science, which consists of up to 18 students and runs for a year. The first stage has already been completed for the OWL.Culture-Platform prior to the case study, which is why the students complete only the further stages 2-4. For the evaluation, we used surveys to measure the usability of the approach from the perspective of the participants, but also to identify possible confounding factors. Furthermore, we analyzed the individual artifacts created in each stage in order to assess the quality of the work results and the interaction of each stage. The surveys showed that the participants perceived the approach as feasible and also evaluated it positively. Furthermore, the analysis of the artifacts has shown that the approach is feasible and that the interaction between the different stages works. However, it was also found that most of the participants had great difficulties to formulate requirements from a user perspective and to maintain a separation between problem and solution. This concludes our summary and in the following we discuss the results of this thesis.

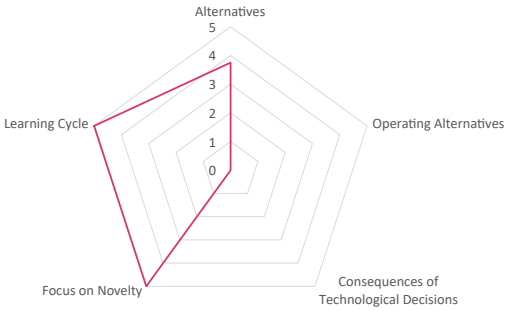
## 9.2 Discussion

For the discussion of this thesis, we will start with our objectives from section 1.2, more precisely with their operationalization as *Fitness Function* (FF). In the summary and discussion of each stage we have already discussed the fulfillment of the fitness function based on the individual stage. These are summarized in Figure 9.3. However, a discussion of the fitness function in relation to the overall approach is still missing.

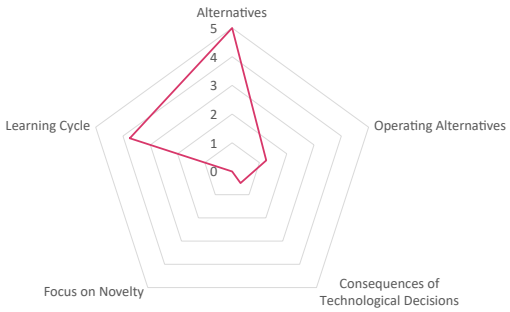
Our first characteristic of our FF is *Alternatives*. For this, we have defined that it is fully fulfilled if "at least two solutions are simultaneously supported, created in parallel, and



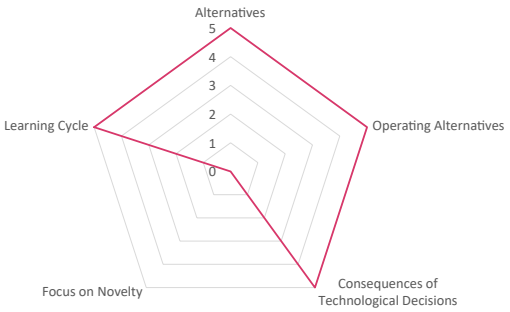

**(a)** ICeDD Stage (1): Initialize Design Thinking




**(b)** ICeDD Stage (2): Execute Design Thinking with Non-Software




**(c)** ICeDD Stage (3): Prepare Design Thinking with Software




**(d)** ICeDD Stage (4): Execute Design Thinking with Software

**Fig. 9.3.:** Summary of the FF results for the individual stages.

can be present as software as well as non-software". In our first stage, we have ensured that the corresponding design challenge at least two solutions can be found and created in parallel, but we cannot yet ensure at this stage that it can also be present as software, which is why we it only got 3,75. Furthermore, we create at least two solutions in the second stage, but since the focus here is still on value, it is not yet ensured that the solutions can actually be implemented as software. However, it is still ensured that at least two solution alternatives are available at the end. This changes only in the third stage, in which we transform the results from the second stage for software development and enrich them with the Technical and Look & Feel levels. In this way we achieve that "at least two solutions are simultaneously supported, created in parallel, and can be present as software as well as non-software". As we are using these in the fourth stage and develop actual software, this also holds true for that stage. Accordingly, the characteristic *Alternatives* is also fulfilled for the overall approach.

The next characteristic is *Operating Alternatives*, which is related to the implemented software. This is why stage 1-3 do not fulfill it as they are not concerned about the specific implementation yet, but stage 4 is. We have defined that it is fully fulfilled if we have an *Online Fallback Mechanism*, *Component based Deployment*, *Automatic Deployment and*

*Configuration*, and *Automatic User Specific Online Orchestration*. By using our *System of Systems* macro-architecture, we ensure that we get constituent systems that can be deployed and run individually and therefore get a component based deployment. Furthermore, by introducing a CI/CD pipeline the deployment and configuration of the constituent systems can also be done automatically as well as automatically reverting back to a previous version. In conjunction with the introduced *Technical Assignment System* (section 7.3.2) we can ensure an *Online Fallback Mechanism*, but also an *Automatic User Specific Online Orchestration* as we can individually control which user gets redirected to which constituent system but also with feature toggling which feature is activated. Correspondingly, this characteristic is also completely fulfilled for the overall approach, since only the fourth stage refers to concrete software and also fulfils this completely.

The third characteristic is *Consequences of Technological Decisions*, which is fully fulfilled if different programming languages and technologies (*Polyglotism*) can be used, the UI is independent of the Model layer, we can support parallel models, and have a System of Systems architecture (*Suite of Small Services*, *Bounded Contexts*, *Services run in its own Process*, *Independently Deployable Services*)). This is also only related to the fourth stage, which fulfills it completely. We have a *System of Systems* macro-architecture that uses *Bounded Contexts* to split services on a managerial level in a suite of small services or constituent systems and additionally have ensured with it that each constituent system or service runs in its own process and is independently deployable. Furthermore, by introducing event sourcing and CQRS we have enabled the creation of parallel models by utilizing the event-driven architecture obtained with event sourcing. This also allows us to use completely different technologies in the services as long as they adhere to the macro-architecture that mainly requires REST-APIs as a means for a unified communication and the usage of event sourcing. With the introduction of MVVM, we have also ensured the independence of UI and Model.

For the fourth characteristic *Focus on Novelty*, we have defined that it is fully fulfilled if the ”method does not merely copy current state but creates novel solutions (this includes e.g. structure, tasks, or processes). If we look at Figure 9.3, we see that the first two stages fulfill this, but not the third and fourth stage. In the first stage, we are using grounded theory as a theory generating approach and with that fulfill that we do not focus only on describing the current state but also creating new ideas on how the state could be changed. But as asking experts about new ideas without having a tangible prototype is not very promising (cf. section 2.1 for when an innovation is likely to be adopted), we additionally need design thinking in the next stage. With Design Thinking and its diverging and converging thinking we ensure that we do not stick to only the current state but also create novel solutions that are tested out with tangible prototypes. We are ending this stage with the converging phase of design thinking in which we still have at least two alternative solutions. Accordingly, the

next stages serve only for further convergence and not for divergence. That is why they do not fulfil the focus on novelty. But if we look at the overall approach, we fully meet the *Focus on Novelty* characteristic, because we do this already in the first two stages and then only converge.

Our last characteristic is *Learning Cycle* that is fulfilled if "a learning cycle is intended, explicitly defined, refers to several alternatives at a time, and adapts it according to the context". All stages beside the third stage have introduced an explicitly defined learning cycle that refers to several alternatives at a time and adapts it according to the context. The first stage has it with grounded theory, the second stage with design thinking, and the fourth stage with qualitative experiments. The third stage is not fully covering it as it has a learning cycle that uses several alternatives but is not adapting to its context. However, at this stage it is not necessary for the learning cycle to adapt to the context, since this stage is not primarily concerned with new insights, but with the transformation of existing ones. Accordingly, our overall approach also fully meets this characteristic. In Figure 9.4, we have illustrated all characteristics related to the overall approach in a radar chart.

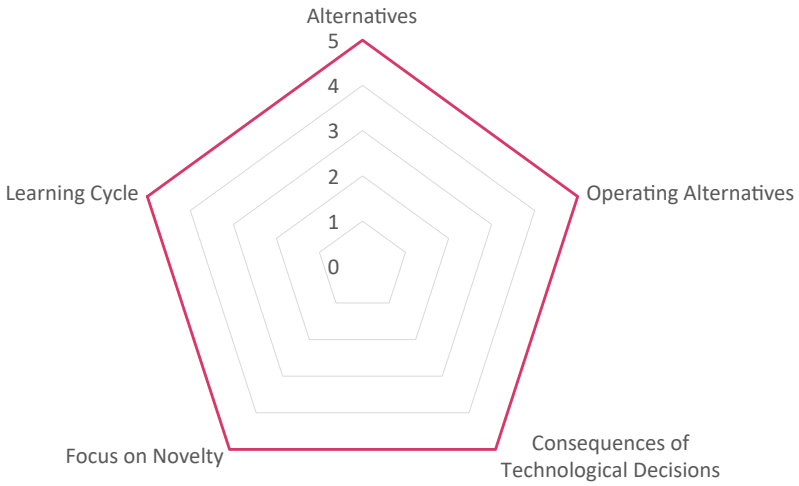



**Fig. 9.4.:** Radar Chart for our approach ICeDD regarding our FF.

With the FF now completed for the approach, we can next discuss our objectives. Objective 2 is represented by the two characteristics *Alternatives* and *Learning Cycle*, with *Alternatives* being derived entirely from Objective 2. In short, Objective 2 defines that we need to learn about dependencies and constraints from the difference of multiple solutions that are simultaneously supported. Accordingly, fulfilling these two characteristics means that we have also fulfilled Objective 2.

Objective 3 is about supporting the simultaneous operation of several software solutions. It is fully represented by the characteristic *Operating Alternatives*. Hence, as we have fully fulfilled *Operating Alternatives*, we have as well fulfilled this objective.

Objective 4 is there to ensure that technological decisions restrict us in the future as few as possible. This objective is fully covered by the characteristic *Consequences of Technological Decisions*, which we fulfill completely as well with our approach. Therefore, this objective is fulfilled as well.

Objective 5 is concerned with the development of new ideas on how to shape the context of use. The characteristic *Focus on Novelty* is only derived from this objective and no other characteristic is related to this objective. By completely fulfilling this characteristic as well, we also fulfill this objective.

Objective 6 is the last objective related to our FF and defines that the development process itself should be a learning and understanding process. From this objective we have only derived the characteristic *Learning Cycle*, which we already mentioned fulfill completely. Therefore, this objective is fulfilled as well.

Our last objective is Objective 1 which requires that findings from other disciplines useful for the development of unique and novel software-based solutions are identified. In this thesis, we have ensured it with the help of our research approach (see section 1.3.2). We have used *Action Research* as a foundation to uncover potential issues in the development process and take these to conduct a literature research also in other fields. Overall, this was an iterative and incremental research approach for developing our approach for developing unique and novel software-based solutions. In section 8.1 we have summarized challenges and results from this approach. This way, we have integrated results from economics with *Diffusions of Innovations* for innovations or the *Cynefin* framework to understand the basic properties we need for our approach. Furthermore, by identifying the challenge of tacit knowledge, we got to expert knowledge and psychology which strongly influenced the way we structured our approach and included certain aspects like the *On-Site Feature Requests*. For the creation of design challenges we have identified the need for a theory generating approach, which found in social sciences with grounded theory. Design Thinking as generalized approach on how designers are working got the foundation for our approach and was adapted accordingly for software development. Accordingly, we have also fulfilled the goal of integrating findings from other disciplines into our approach in order to obtain a meaningful approach.

Our research approach has helped us to achieve a high level of validity, even if this was at the expense of reliability. However, we were able to compensate for this with the help of the literature research and thus with well-proven findings. Furthermore, it has helped us to focus on points that are actually and not just theoretically important for the approach, since our research approach is based on problems that have arisen in actual development. However, one could argue that due to the university setting with the project group, the whole thing is not really transferable to software development companies.

Overall, in this thesis we have given with ICeDD a possible approach for developing unique and novel software-based solutions that is mainly based on well-proven findings from different disciplines including computer science. We have delivered evidence for the feasibility of this approach on an argumentation level in the single stages but also with our OWL.Culture-Platform case study in section 8.3. The feasibility was also what was in focus of this thesis. What is missing, is further evidence that ICeDD is actually better in developing unique and novel software-based solutions than other approaches. This brings us to the next part, future work.

## 9.3 Future Work

The focus of this thesis was to develop an approach for unique and novel software-based solutions and give evidence for its feasibility. Neither the efficiency nor the effectivity have yet been evaluated. In future studies this has to be done by doing for example a controlled experiment in which one group is using this approach (ICeDD) and another one an alternative approach like Scrum alone.

Furthermore, as the focus was on the feasibility, the single stages included only the necessary parts for showing the feasibility of the approach. Especially the tools are just first prototypes with rudimentary features that have potential for further improvements. Hence, in further studies it can be researched how the individual parts of this approach can be further improved.

As the OWL.Culture-Platform got a funding starting from 2020, we can actually try out this approach with a real company. Besides further looking into how to actually implement the macro-architecture correctly, we will further look into realizing qualitative experiments. In this thesis, we only conducted a rudimentary experiment at the end of the case study in section 8.3. For this project, we want to further improve our tool set for qualitative experiments, especially for conducting remote testing. It is planned to continuously do user studies in this three year project with actual users in the *Ostwestfalen-Lippe* (OWL) region. Furthermore, the goal is not to develop a research prototype, but a working product. Hence, we can experiment in a more realistic setting than we already had with our student project group.

# Bibliography


[Abi19] Zain Ul Abidin. "User Specific Online Feature Orchestration". MA thesis. 2019 (cit. on pp. 146, 152).

[Ado06] S. Adolph. "What lessons can the agile community learn from a maverick fighter pilot?" In: *AGILE 2006 (AGILE'06)*. July 2006 (cit. on pp. 122, 123, 168, 183).

[AND94] PEK VAN ANDEL. "Anatomy of the Unsought Finding. Serendipity: Orgin, History, Domains, Traditions, Appearances, Patterns and Programmability". In: *The British Journal for the Philosophy of Science* 45.2 (June 1994), pp. 631–648. eprint: `http://oup.prod.sis.lan/bjps/article-pdf/45/2/631/9745686/631.pdf` (cit. on p. 78).

[Arm+10] Michael Armbrust, Armando Fox, Rean Griffith, et al. "A View of Cloud Computing". In: *Commun. ACM* 53.4 (Apr. 2010), pp. 50–58 (cit. on pp. 9, 55).

[ACO16] N. L. Atukorala, C. K. Chang, and K. Oyama. "Situation-Oriented Requirements Elicitation". In: *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 1. June 2016, pp. 233–238 (cit. on p. 71).

[BKM09] Aaron Bangor, Philip Kortum, and James Miller. "Determining what individual SUS scores mean: Adding an adjective rating scale". In: *Journal of usability studies* 4.3 (2009), pp. 114–123 (cit. on pp. 150, 152).

[Bas+19] Ali Basiri, Lorin Hochstein, Nora Jones, and Haley Tucker. "Automating chaos experiments in production". In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE. 2019, pp. 31–40 (cit. on p. 138).

[Bat+13] A. Batool, Y. H. Motla, B. Hamid, et al. "Comparative study of traditional requirement engineering and Agile requirement engineering". In: *2013 15th International Conference on Advanced Communications Technology (ICACT)*. Jan. 2013, pp. 1006–1014 (cit. on p. 69).

[Bat05] Don Batory. "Feature models, grammars, and propositional formulas". In: *International Conference on Software Product Lines*. Springer. 2005, pp. 7–20 (cit. on p. 112).

[Ben84] Patricia Benner. "From novice to expert". In: *Menlo Park* (1984) (cit. on pp. 24, 51, 63).

[Bet+13] Dominic Betts, Julian Dominguez, Grigori Melnik, Fernando Simonazzi, and Mani Subramanian. *Exploring CQRS and Event Sourcing: A Journey into High Scalability, Availability, and Maintainability with Windows Azure*. 1st. Microsoft patterns & practices, 2013 (cit. on pp. 131–133, 141, 143).

[BH99] Hugh Beyer and Karen Holtzblatt. “Contextual Design”. In: *Interactions* 6.1 (Jan. 1999), pp. 32–42 (cit. on p. 85).

[BHB04] Hugh Beyer, Karen Holtzblatt, and Lisa Baker. “An Agile Customer-Centered Method: Rapid Contextual Design”. In: *Extreme Programming and Agile Methods - XP/Agile Universe 2004*. Ed. by Carmen Zannier, Hakan Erdogmus, and Lowell Lindstrom. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 50–59 (cit. on p. 83).

[Böc+11] Irina Böckelmann, Daniel Schenk, Thoralf Rößler, et al. *Physiologische Beanspruchungsreaktionen bei der Anwendung von kopfgetragenen AR-Displays*. 2011 (cit. on p. 27).

[Boe88] B. W. Boehm. “A spiral model of software development and enhancement”. In: *Computer* 21.5 (May 1988), pp. 61–72 (cit. on p. 70).

[Boe06a] Barry Boehm. “A view of 20th and 21st century software engineering”. In: *Proceedings of the 28th international conference on Software engineering*. ACM. 2006, pp. 12–29 (cit. on pp. 8, 10, 183).

[Boe06b] Barry W Boehm. “Value-based software engineering: Overview and agenda”. In: *Value-based software engineering*. Springer, 2006, pp. 3–14 (cit. on pp. 48, 56, 87, 98, 99, 104).

[BBS16] Benjamin W. Bohl, Axel Berndt, and Björn Senft. “Formate als Sackgassen: Handlungsempfehlungen”. In: *Konferenzabstracts der 3. Tagung des Verbands "Digital Humanities im deutschsprachigen Raum e. V."* Leipzig, Mar. 2016, pp. 103–107 (cit. on p. 27).

[Bos14] Jan Bosch. “Continuous Software Engineering: An Introduction”. In: *Continuous Software Engineering*. Ed. by Jan Bosch. Cham: Springer International Publishing, 2014, pp. 3–13 (cit. on pp. 57, 145).

[Bro+96] John Brooke et al. “SUS-A quick and dirty usability scale”. In: *Usability evaluation in industry* 189.194 (1996), pp. 4–7 (cit. on p. 150).

[BR01] Glenn J. Browne and Michael B. Rogich. “An Empirical Investigation of User Requirements Elicitation: Comparing the Effectiveness of Prompting Techniques”. In: *Journal of Management Information Systems* 17.4 (2001), pp. 223–249. eprint: `https://doi.org/10.1080/07421222.2001.11045665` (cit. on pp. 68, 69).

[Bur+12] Anne Burdick, Johanna Drucker, Peter Lunenfeld, Todd Presner, and Jeffrey Schnapp. *Digital_Humanities*. Mit Press, 2012 (cit. on p. 168).

[BMS20] Bianca Burgfeld-Meise, Dorothee M Meister, and Björn Senft. “Subjektorientierte Softwareentwicklung als medienpädagogische Herausforderung”. In: *MedienPädagogik: Zeitschrift für Theorie und Praxis der Medienbildung* 39 (2020), pp. 86–102 (cit. on pp. 27, 51).

[Bur10] Thomas Burkart. “Qualitatives Experiment”. In: *Handbuch Qualitative Forschung in der Psychologie*. Ed. by Günter Mey and Katja Mruck. Wiesbaden: VS Verlag für Sozialwissenschaften, 2010, pp. 252–262 (cit. on pp. 122, 124).

[CK96] P. Carlshamre and J. Karlsson. “A usability-oriented approach to requirements engineering”. In: *Proceedings of the Second International Conference on Requirements Engineering*. Apr. 1996, pp. 145–152 (cit. on p. 72).

[CKD12] M. Cataldo, I. Kwan, and D. Damian. “Conway’s Law Revisited: The Evidence for a Task-Based Perspective”. In: *IEEE Software* 29.01 (Jan. 2012), pp. 90–93 (cit. on p. 169).

[CL99] Larry L Constantine and Lucy AD Lockwood. *Software for use: a practical guide to the models and methods of usage-centered design*. Pearson Education, 1999 (cit. on p. 83).

[Con68] Melvin E Conway. “How do committees invent”. In: *Datamation* 14.4 (1968), pp. 28–31 (cit. on p. 169).

[Coo94] Nancy J Cooke. “Varieties of knowledge elicitation techniques”. In: *International Journal of Human-Computer Studies* 41.6 (1994), pp. 801–849 (cit. on pp. 71, 72).

[CRC07] Alan Cooper, Robert Reimann, and David Cronin. “Modeling Users: Personas and Goals”. In: *About Face 3: The Essentials of Interaction Design*. Wiley Publishing, 2007 (cit. on pp. 71, 72, 74, 83).

[DB05] Kristina B. Dahlin and Dean M. Behrens. “When is an invention really radical?: Defining and measuring technological radicalness”. In: *Research Policy* 34.5 (2005), pp. 717–737 (cit. on p. 5).

[DV15] Werner Damm and Alberto Sangiovanni Vincentelli. “A Conceptual Model of System of Systems”. In: *Proceedings of the Second International Workshop on the Swarm at the Edge of the Cloud*. SWEC ’15. Seattle, Washington: Association for Computing Machinery, 2015, pp. 19–27 (cit. on p. 130).

[Dan10] Erik Dane. “Reconsidering the Trade-off Between Expertise and Flexibility: a Cognitive Entrenchment Perspective”. In: *Academy of Management Review* 35.4 (2010), pp. 579–603. eprint: `https://doi.org/10.5465/amr.35.4.zok579` (cit. on p. 179).

[DGH08] Peter J. Denning, Chris Gunderson, and Rick Hayes-Roth. “The Profession of IT: Evolutionary System Development”. In: *Commun. ACM* 51.12 (Dec. 2008), pp. 29–31 (cit. on pp. 9, 15, 53, 104, 111, 130, 139).

[Dob+20] Franziska Dobrigkeit, Philipp Pajak, Danielly de Paula, and Matthias Uflacker. “DT@IT Toolbox: Design Thinking Tools to Support Everyday Software Development”. In: *Design Thinking Research : Investigating Design Team Performance*. Ed. by Christoph Meinel and Larry Leifer. Cham: Springer International Publishing, 2020, pp. 201–227 (cit. on p. 57).

[DP19] Franziska Dobrigkeit and Danielly de Paula. “Design Thinking in Practice: Understanding Manifestations of Design Thinking in Software Engineering”. In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2019. Tallinn, Estonia: Association for Computing Machinery, 2019, pp. 1059–1069 (cit. on p. 57).

[DP+17] Franziska Dobrigkeit, Danielly de Paula, et al. “The best of three worlds-the creation of INNODEV a software development approach that integrates design thinking, SCRUM and lean startup”. In: *DS 87-8 Proceedings of the 21st International Conference on Engineering Design (ICED 17) Vol 8: Human Behaviour in Design, Vancouver, Canada, 21-25.08. 2017*. 2017, pp. 319–328 (cit. on p. 57).

[DPU19] Franziska Dobrigkeit, Danielly de Paula, and Matthias Uflacker. “InnoDev: A Software Development Methodology Integrating Design Thinking, Scrum and Lean Startup”. In: *Design Thinking Research : Looking Further: Design Thinking Beyond Solution-Fixation*. Ed. by Christoph Meinel and Larry Leifer. Cham: Springer International Publishing, 2019, pp. 199–227 (cit. on p. 57).

[Dod02] Gordana Dodig-Crnkovic. “Scientific methods in computer science”. In: *Proceedings of the Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden, Skövde, Suecia*. 2002, pp. 126–130 (cit. on pp. 18, 19).

[Dor11] Kees Dorst. “The core of ‘design thinking’ and its application”. In: *Design Studies* 32.6 (2011). Interpreting Design Thinking, pp. 521–532 (cit. on pp. 40, 43).

[Dra+18] Nicola Dragoni, Ivan Lanese, Stephan Thordal Larsen, et al. “Microservices: How To Make Your Application Scale”. In: *Perspectives of System Informatics*. Ed. by Alexander K. Petrenko and Andrei Voronkov. Cham: Springer International Publishing, 2018, pp. 95–104 (cit. on p. 170).

[DD88] Hubert L. Dreyfus and Stuart E. Dreyfus. *Künstliche Intelligenz : von den Grenzen der Denkmaschine und dem Wert der Intuition*. Aus dem Engl. übers. 1988 (cit. on p. 63).

[DF12] Joseph C. Dumas and Jean E. Fox. “Usability Testing”. In: *Human Computer Interaction Handbook: Fundamentals, Evolving Technologies, and Emerging Applications*. Ed. by Julie A Jacko. CRC press, 2012, pp. 1221–1241 (cit. on p. 125).

[Eas+08] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. “Selecting Empirical Methods for Software Engineering Research”. In: *Guide to Advanced Empirical Software Engineering*. Ed. by Forrest Shull, Janice Singer, and Dag I. K. Sjøberg. London: Springer London, 2008, pp. 285–311 (cit. on p. 23).

[Eva04] Eric Evans. *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004 (cit. on pp. 104, 130, 134).

[Fer+17] D. Méndez Fernández, S. Wagner, M. Kalinowski, et al. “Naming the pain in requirements engineering”. In: *Empirical Software Engineering* 22.5 (Oct. 2017), pp. 2298–2338 (cit. on pp. 63, 68).

[FS16] Holger Fischer and Björn Senft. “Human-Centered Software Engineering as a Chance to Ensure Software Quality Within the Digitization of Human Workflows”. In: *Human-Centered and Error-Resilient Systems Development. Proceedings of the 6th International Conference on Human-Centered Software Engineering (HCSE)*. Vol. 9856. LNCS. Springer, 2016, pp. 30–41 (cit. on pp. 15, 16, 27).

[Fis+18] Holger Fischer, Björn Senft, Florian Rittmeier, and Stefan Sauer. “A Canvas Method to Foster Interdisciplinary Discussions on Digital Assistance Systems”. In: *Design, User Experience, and Usability: Theory and Practice. Proceedings of the 20th International Conference on Human-Computer Interaktion (HCI International 2018)*. Ed. by Aaron Marcus and Wentao Wang. LNCS, vol. 10918. Springer, 2018, pp. 711–724 (cit. on p. 27).

[FSS17] Holger Fischer, Björn Senft, and Katharina Stahl. “Akzeptierte Assistenzsysteme in der Arbeitswelt 4.0 durch systematisches Human-Centered Software Engineering”. In: *Wissenschafts- und Industrieforum 2017 - Intelligente Technische Systeme*. Ed. by Eric Bodden, Falko Dressler, Roman Dumitrescu, et al. Vol. 369. Paderborn: Verlagsschriftenreihe des Heinz Nixdorf Instituts, 2017, pp. 197–210 (cit. on p. 27).

[FSN13] Holger Gerhard Fischer, Benjamin Strenge, and Karsten Nebe. “Towards a Holistic Tool for the Selection and Validation of Usability Method Sets Supporting Human-Centered Design”. In: *Design, User Experience, and Usability. Design Philosophy, Methods, and Tools*. Vol. 8012. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 252–261 (cit. on p. 148).

[FS14] Brian Fitzgerald and Klaas-Jan Stol. “Continuous Software Engineering and Beyond: Trends and Challenges”. In: *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*. RCoSE 2014. Hyderabad, India: ACM, 2014, pp. 1–9 (cit. on p. 57).

[FF94] William Foddy and William H Foddy. *Constructing questions for interviews and questionnaires: Theory and practice in social research*. Cambridge university press, 1994 (cit. on p. 72).

[FPK17] Neal Ford, Rebecca Parsons, and Patrick Kua. *Building Evolutionary Architectures: Support Constant Change*. " O’Reilly Media, Inc.", 2017 (cit. on pp. 12, 15, 53, 131, 137, 223).

[Fow01] M. Fowler. “Separating user interface code”. In: *IEEE Software* 18.2 (2001), pp. 96–97 (cit. on p. 134).

[FBB04] Fay Fransella, Richard Bell, and Don Bannister. *A manual for repertory grid technique*. John Wiley & Sons, 2004 (cit. on p. 72).

[FS11] David M. Frohlich and Risto Sarvas. “HCI and Innovation”. In: *CHI ’11 Extended Abstracts on Human Factors in Computing Systems*. CHI EA ’11. Vancouver, BC, Canada: Association for Computing Machinery, 2011, pp. 713–728 (cit. on p. 168).

[Ger+16] Sebastian Gerdes, Stefanie Jasser, Matthias Riebisch, et al. “Towards the Essentials of Architecture Documentation for Avoiding Architecture Erosion”. In: *Proccedings of the 10th European Conference on Software Architecture Workshops*. ECSAW ’16. Copenhagen, Denmark: Association for Computing Machinery, 2016 (cit. on p. 101).

[Ger+13] V. Gervasi, R. Gacitua, M. Rouncefield, et al. “Unpacking Tacit Knowledge for Requirements Engineering”. In: *Managing Requirements Knowledge*. Ed. by Walid Maalej and Anil Kumar Thurimella. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 23–47 (cit. on pp. 51, 63).

[GL09] Jochen Gläser and Grit Laudel. *Experteninterviews und qualitative Inhaltsanalyse. 3., überarbeitete Auflage*. 2009 (cit. on pp. 80, 103, 174).

[GB75] Duncan R Godden and Alan D Baddeley. "Context-dependent memory in two natural environments: On land and underwater". In: *British Journal of psychology* 66.3 (1975), pp. 325–331 (cit. on p. 66).

[GL93] J. A. Goguen and C. Linde. "Techniques for requirements elicitation". In: *[1993] Proceedings of the IEEE International Symposium on Requirements Engineering*. Jan. 1993, pp. 152–164 (cit. on p. 72).

[GF94] O. C. Z. Gotel and C. W. Finkelstein. "An analysis of the requirements traceability problem". In: *Proceedings of IEEE International Conference on Requirements Engineering*. Apr. 1994, pp. 94–101 (cit. on p. 111).

[Gre08] Sue Greener. *Business research methods*. BookBoon, 2008 (cit. on p. 72).

[Gre+16] Markus Greulich, Nicola Karthaus, Simon Oberthür, et al. "Design Thinking als Methode in den Digital Humanities. Optionen interdisziplinären, forschenden Lehrens und Lernens". In: *Konferenzabstracts der 45. Jahrestagung der Deutschen Gesellschaft für Hochschuldidaktik*. 2016 (cit. on pp. 27, 163).

[Gro+95] Standish Group et al. "The Standish Group Report-CHAOS". In: *The Standish Group* (1995) (cit. on p. 64).

[GSA16] Kavitha Gurusamy, Narayanan Srinivasaraghavan, and Sisira Adikari. "An Integrated Framework for Design Thinking and Agile Methods for Digital Transformation". In: *Design, User Experience, and Usability: Design Thinking and Methods*. Ed. by Aaron Marcus. Cham: Springer International Publishing, 2016, pp. 34–42 (cit. on p. 56).

[Gut82] Jonathan Gutman. "A Means-End Chain Model Based on Consumer Categorization Processes". In: *Journal of Marketing* 46.2 (1982), pp. 60–72. eprint: `https://doi.org/10.1177/002224298204600207` (cit. on p. 72).

[Hal13] Erika Hall. *Just enough research*. A Book Apart New York, 2013 (cit. on p. 70).

[HBK03] Marc Hassenzahl, Michael Burmester, and Franz Koller. "AttrakDiff: Ein Fragebogen zur Messung wahrgenommener hedonischer und pragmatischer Qualität". In: *Mensch & Computer 2003: Interaktion in Bewegung*. Ed. by Gerd Szwillus and Jürgen Ziegler. Wiesbaden: Vieweg+Teubner Verlag, 2003, pp. 187–196 (cit. on p. 75).

[Hau+16] Jürgen Hauschildt, Sören Salomo, Carsten Schultz, and Alexander Kock. *Innovationsmanagement*. Vahlen, 2016 (cit. on p. 31).

[Hel11] Cornelia Helfferich. *Die Qualität qualitativer Daten : Manual für die Durchführung qualitativer Interviews [Elektronische Ressource]*. 2011 (cit. on pp. 21, 82, 122, 128, 171, 184, 194–196).

[HRB11] Niels Henze, Enrico Rukzio, and Susanne Boll. "100,000,000 Taps: Analysis and Improvement of Touch Performance in the Large". In: *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*. MobileHCI '11. Stockholm, Sweden: Association for Computing Machinery, 2011, pp. 133–142 (cit. on p. 156).

[HD04] ANN M. HICKEY and ALAN M. DAVIS. “A Unified Model of Requirements Elicitation”. In: *Journal of Management Information Systems* 20.4 (2004), pp. 65–84. eprint: https://doi.org/10.1080/07421222.2004.11045786 (cit. on p. 69).

[Hin65] Dennis Neil Hinkle. “The change of personal constructs from the viewpoint of a theory of construct implications”. PhD thesis. The Ohio State University, 1965 (cit. on p. 72).

[Hoa69] Charles Antony Richard Hoare. “An axiomatic basis for computer programming”. In: *Communications of the ACM* 12.10 (1969), pp. 576–580 (cit. on pp. 11, 19).

[HW95] Joseph E Hollingsworth and Bruce W Weide. “Micro-Architecture vs. Macro-Architecture”. In: *Proceedings of the Seventh Annual Workshop on Software Reuse*. Citeseer. 1995 (cit. on pp. 101, 130).

[HH97] Stephanie Houde and Charles Hill. “What do prototypes prototype?” In: *Handbook of human-computer interaction*. Elsevier, 1997, pp. 367–381 (cit. on pp. 48, 50, 101, 228).

[HSE13] Walter Hussy, Margrit Schreier, and Gerald Echterhoff. *Forschungsmethoden in Psychologie und Sozialwissenschaften*. 2. Aufl. Berlin Heidelberg New York: Springer-Verlag, 2013 (cit. on pp. 20, 21, 121–123).

[IDE13] IDEO LLC. *Design Thinking for Educators*. Online available under https://designthinkingforeducators.com/toolkit/ (visited on Jul. 10, 2019). IDEO-books, 2013 (cit. on pp. 43, 61, 62).

[JBR99] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The unified software development process*. Addison-Wesley Professional, 1999 (cit. on pp. 50, 52, 53, 120, 143, 144, 157, 228, 229).

[Kel84] J. F. Kelley. “An Iterative Design Methodology for User-friendly Natural Language Office Information Applications”. In: *ACM Trans. Inf. Syst.* 2.1 (Jan. 1984), pp. 26–41 (cit. on pp. 186, 199).

[Kel03] George Kelly. *The psychology of personal constructs: Volume two: Clinical diagnosis and psychotherapy*. Routledge, 2003 (cit. on p. 72).

[KJ12] Lucy Kimbell and Joe Julier. “The social design methods menu”. In: *perpetual beta* (2012) (cit. on p. 72).

[KV13] John Klein and Hans van Vliet. “A Systematic Review of System-of-Systems Architecture Research”. In: *Proceedings of the 9th International ACM Sigsoft Conference on Quality of Software Architectures*. QoSA ’13. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2013, pp. 13–22 (cit. on p. 130).

[Kle91] Gerhard Kleining. *Das qualitative experiment*. 1991 (cit. on p. 122).

[Kle19] Markus Klemens. “Developing a Conceptual Framework for Transforming Design Thinking Results into Agile Software Requirements with Preservation of Alternative Solutions”. MA thesis. 2019 (cit. on p. 102).

[Klo+11] Florian Klompmaker, Björn Senft, Karsten Nebe, Clemens Busch, and Detlev Willemsen. “User Centered Design Process of OSAMi-D - Developing User Interfaces for a Remote Ergometer Training Application”. In: *HEALTHINF 2011 - Proceedings of the International Conference on Health Informatics, Rome, Italy, 26-29 January, 2011*. 2011, pp. 268–273 (cit. on p. 27).

[KZK16] Jake Knapp, John Zeratsky, and Braden Kowitz. *Sprint: How to solve big problems and test new ideas in just five days*. Simon and Schuster, 2016 (cit. on p. 90).

[Koh+08] Ron Kohavi, Roger Longbotham, Dan Sommerfield, and Randal M. Henne. “Controlled experiments on the web: survey and practical guide”. In: *Data Mining and Knowledge Discovery* 18.1 (July 2008), pp. 140–181 (cit. on pp. 57, 121, 127, 128, 146, 148).

[Koh+09] Ronny Kohavi, Thomas Crook, Roger Longbotham, et al. “Online experimentation at Microsoft”. In: *Data Mining Case Studies* 11 (2009), p. 39 (cit. on pp. 4, 10, 168, 183, 214, 225).

[Kow+14] Thomas Kowark, Franziska Häger, Ralf Gehrer, and Jens Krüger. “A Research Plan for the Integration of Design Thinking with Large Scale Software Development Projects”. In: *Design Thinking Research: Building Innovation Eco-Systems*. Ed. by Larry Leifer, Hasso Plattner, and Christoph Meinel. Cham: Springer International Publishing, 2014, pp. 183–202 (cit. on p. 56).

[Kra18] Janis Krasemann. “Designing and Implementing a System for Passive User Feedback”. MA thesis. 2018 (cit. on pp. 146, 155).

[KSW13] Stephan Kraus, Guido Steinacker, and Oliver Wegner. “Teile und Herrsche–Kleine Systeme für große Architekturen”. In: *OBJEKTspektrum* 5 (2013), pp. 8–13 (cit. on p. 101).

[Kru16] Stephan Krusche. “Rugby - A Process Model for Continuous Software Engineering”. PhD thesis. Technical University Munich, Germany, 2016 (cit. on p. 58).

[Kru+14] Stephan Krusche, Lukas Alperowitz, Bernd Bruegge, and Martin O. Wagner. “Rugby: An Agile Process Model Based on Continuous Delivery”. In: *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*. RCoSE 2014. Hyderabad, India: ACM, 2014, pp. 42–50 (cit. on p. 58).

[Kuh+17] Marco Kuhrmann, Philipp Diebold, Jürgen Münch, et al. “Hybrid Software and System Development in Practice: Waterfall, Scrum, and Beyond”. In: *Proceedings of the 2017 International Conference on Software and System Process*. ICSSP 2017. Paris, France: Association for Computing Machinery, 2017, pp. 30–39 (cit. on p. 168).

[KS03] Cynthia F Kurtz and David J Snowden. “The new dynamics of strategy: Sense-making in a complex and complicated world”. In: *IBM systems journal* 42.3 (2003), pp. 462–483 (cit. on pp. 6, 7, 10, 47, 50, 65, 69, 168, 225).

[Lew46] Kurt Lewin. “Action research and minority problems”. In: *Journal of social issues* 2.4 (1946), pp. 34–46 (cit. on pp. 23, 24).

[LMW11] Tilmann Lindberg, Christoph Meinel, and Ralf Wagner. "Design Thinking: A Fruitful Concept for IT Development?" In: *Design Thinking: Understand – Improve – Apply*. Ed. by Christoph Meinel, Larry Leifer, and Hasso Plattner. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 3–18 (cit. on pp. 4, 5, 40–42, 50, 56, 97, 98, 117, 168, 183, 225).

[LM15] Eveliina Lindgren and Jürgen Münch. "Software Development as an Experiment System: A Qualitative Survey on the State of the Practice". In: *Agile Processes in Software Engineering and Extreme Programming*. Ed. by Casper Lassenius, Torgeir Dingsøyr, and Maria Paasivaara. Cham: Springer International Publishing, 2015, pp. 117–128 (cit. on p. 164).

[Luc+17] Percival Lucena, Alan Braz, Adilson Chicoria, and Leonardo Tizzei. "IBM Design Thinking Software Development Framework". In: *Agile Methods*. Ed. by Tiago Silva da Silva, Bernardo Estácio, Josiane Kroll, and Rafaela Mantovani Fontana. Cham: Springer International Publishing, 2017, pp. 98–109 (cit. on p. 56).

[Mai13] N. Maiden. "So, What Is Requirements Work?" In: *IEEE Software* 30.2 (Mar. 2013), pp. 14–15 (cit. on p. 70).

[Mai+10] N. Maiden, S. Jones, K. Karlsen, et al. "Requirements Engineering as Creative Problem Solving: A Research Agenda for Idea Finding". In: *2010 18th IEEE International Requirements Engineering Conference*. Sept. 2010, pp. 57–66 (cit. on p. 69).

[MR96] Neil AM Maiden and Gordon Rugg. "ACRE: selecting methods for requirements acquisition". In: *Software Engineering Journal* 11.3 (1996), pp. 183–192 (cit. on p. 63).

[MK13] Robin Marterer and Rainer Koch. "Möglichkeiten der IT-Unterstützung für die Planung, Steuerung, Protokollierung und Auswertung von Einsatzübungen der Behörden und Organisationen mit Sicherheitsaufgaben". In: *Tagungsband der Jahresfachtagung der Vereinigung zur Förderung des deutschen Brandschutzes (vfdb)*. Weimar, 2013 (cit. on p. 172).

[MR15] Joseph A. Maxwell and L. Earle Reybold. "Qualitative Research". In: *International Encyclopedia of the Social & Behavioral Sciences (Second Edition)*. Ed. by James D. Wright. Second Edition. Oxford: Elsevier, 2015, pp. 685–689 (cit. on pp. 21, 122).

[May12] Deborah J. Mayhew. "Usability + Persuasiveness + Graphic Design = eCommerce User Experience". In: *Handbook of human-computer interaction*. Elsevier, 2012, pp. 1181–1194 (cit. on pp. 48–50, 87, 99, 101, 228).

[May16] Philipp Mayring. *Einführung in die qualitative Sozialforschung*. ger. 6th ed. Beltz Verlagsgruppe, 2016 (cit. on p. 20).

[May00] Philipp Mayring. "Qualitative Content Analysis". In: *Forum Qualitative Sozialforschung / Forum Qualitative Social Research* 1.2 (2000) (cit. on pp. 80, 103).

[McM04] J. McManus. "A stakeholder perspective within software engineering projects". In: *2004 IEEE International Engineering Management Conference (IEEE Cat. No.04CH37574)*. Vol. 2. Oct. 2004, 880–884 Vol.2 (cit. on p. 70).

[MV19] Yalcinkaya Mehmet and Singh Vishal. “Exploring the use of Gestaltâ€™s principles in improving the visualization, user experience and comprehension of COBie data extension”. In: *Engineering, Construction and Architectural Management* 26.6 (Jan. 2019), pp. 1024–1046 (cit. on p. 56).

[Mei+16a] Bianca Meise, Yevgen Mexin, Franziska Schloots, Björn Senft, and Anastasia Wawilow. “Interdisziplinäre Forschung als Basis nachhaltiger Entscheidungsprozesse in der Softwareentwicklung. Dezentral, vernetzt, kollaborativ”. In: Konferenzband zur 1. interdisziplinäre Konferenz zur Zukunft der Wertschöpfung (2016). Ed. by Jens Wulfsberg, Tobias Redlich, and Manuel Moritz, p. 221 (cit. on pp. 27, 81).

[Mei+16b] Bianca Meise, Yevgen Mexin, Franziska Schloots, et al. “Von implizitem Wissen zu nachhaltigen Systemanforderungen”. In: *Tagungsband der Forschungsdaten in den Geisteswissenschaften (FORGE)*. 2016 (cit. on pp. 26, 27, 51).

[MM11] Günter Mey and Katja Mruck, eds. *Grounded Theory Reader*. VS Verlag für Sozialwissenschaften, 2011 (cit. on p. 81).

[Mey14] Bertrand Meyer. *Agile!: The Good, the Hype and the Ugly*. Springer Science & Business Media, 2014 (cit. on pp. 54, 57, 99, 104, 111, 126, 167).

[Mil56] George A Miller. “The magical number seven, plus or minus two: Some limits on our capacity for processing information.” In: *Psychological review* 63.2 (1956), p. 81 (cit. on p. 66).

[MM10] Katja Mruck and Günter Mey. “Einleitung”. In: *Handbuch Qualitative Forschung in der Psychologie*. Ed. by Günter Mey and Katja Mruck. Wiesbaden: VS Verlag für Sozialwissenschaften, 2010, pp. 11–32 (cit. on p. 171).

[MK12] Michael J Muller and Sandra Kogan. “Grounded Theory Method in Human-Computer Interaction and Computer Supported Cooperative Work”. In: *The Human-Computer Interaction Handboook. CRC Press, Boca Raton {ua}* (2012), pp. 1003–1023 (cit. on pp. 79, 80).

[MR17] Jochen Müsseler and Martina Rieger. *Allgemeine Psychologie*. 3. Aufl. Berlin, Heidelberg: Springer-Verlag, 2017 (cit. on pp. 65, 66, 91).

[NV14] Donald A Norman and Roberto Verganti. “Incremental and radical innovation: Design research vs. technology and meaning change”. In: *Design issues* 30.1 (2014), pp. 78–96 (cit. on pp. 4, 6, 22, 49, 55, 168, 183, 187, 225).

[Nor13] Donald A. Norman. “The Design of Everyday Things”. In: The MIT Press, 2013. Chap. Design Thinking, pp. 217–257 (cit. on p. 41).

[ODr16] Kieran O’Driscoll. “The agile data modelling & design thinking approach to information system requirements analysis”. In: *Journal of Decision Systems* 25.sup1 (2016), pp. 632–638. eprint: `https://doi.org/10.1080/12460125.2016.1189643` (cit. on p. 56).

[OAB12] H. H. Olsson, H. Alahyari, and J. Bosch. “Climbing the "Stairway to Heaven" – A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software”. In: *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*. 2012, pp. 392–399 (cit. on p. 57).

[OB14] Helena Holmström Olsson and Jan Bosch. “The HYPEX Model: From Opinions to Data-Driven Software Development”. In: *Continuous Software Engineering*. Ed. by Jan Bosch. Cham: Springer International Publishing, 2014, pp. 155–164 (cit. on p. 58).

[Osb63] Alex F. (Alex Faickney) Osborn. *Applied imagination : principles and procedures of creative problem solving*. English. 3d rev. ed. Includes bibliographical references. New York : Charles Scribner’s Sons, 1963 (cit. on p. 72).

[OSJ17] M. Overeem, M. Spoor, and S. Jansen. “The dark side of event sourcing: Managing data conversion”. In: *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2017, pp. 193–204 (cit. on p. 133).

[Pah15] C. Pahl. “Containerization and the PaaS Cloud”. In: *IEEE Cloud Computing* 2.3 (May 2015), pp. 24–31 (cit. on pp. 9, 55).

[Pat17] Nitish Patkar. “Vision Backlog”. MA thesis. 2017 (cit. on p. 68).

[PA16] Danielly F. O. de Paula and Cristiano C. Araújo. “Pet Empires: Combining Design Thinking, Lean Startup and Agile to Learn from Failure and Develop a Successful Game in an Undergraduate Environment”. In: *HCI International 2016 – Posters’ Extended Abstracts*. Ed. by Constantine Stephanidis. Cham: Springer International Publishing, 2016, pp. 30–34 (cit. on p. 57).

[PS15] Sabine Pfeiffer and Anne Suphan. “The labouring capacity index: Living labouring capacity and experience as resources on the road to industry 4.0”. In: *Retrieved January* 30 (2015), p. 2016 (cit. on p. 63).

[PW98] David W. Pickton and Sheila Wright. “What’s swot in strategic analysis?” In: *Strategic Change* 7.2 (1998), pp. 101–109. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/%28SICI%291099-1697%28199803/04%297%3A2%3C101%3A%3AAID-JSC332%3E3.0.CO%3B2-6` (cit. on p. 72).

[PML10] Hasso Plattner, Christoph Meinel, and Larry Leifer. *Design thinking: understand–improve–apply*. Springer Science & Business Media, 2010 (cit. on pp. 4, 24, 43).

[Poh07] Klaus Pohl. *Requirements Engineering*. 1st ed. Heidelberg: dpunkt.verlag GmbH, 2007 (cit. on pp. 77, 111, 112, 195).

[PP03] Mary Poppendieck and Tom Poppendieck. *Lean Software Development: An Agile Toolkit: An Agile Toolkit*. Addison-Wesley, 2003 (cit. on pp. 54, 123).

[Pop05] Karl Popper. *The logic of scientific discovery*. Routledge, 2005 (cit. on p. 18).

[Pot96] Mike Potel. *MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java.* Tech. rep. Taligent Inc, 1996 (cit. on pp. 48, 53, 135).

[Pri90] Rubén Prieto-Díaz. “Domain Analysis: An Introduction”. In: *SIGSOFT Softw. Eng. Notes* 15.2 (Apr. 1990), pp. 47–54 (cit. on p. 72).

[Pri08] Dan Pritchett. “BASE: An Acid Alternative”. In: *Queue* 6.3 (May 2008), pp. 48–55 (cit. on p. 138).

[PW09] Aglaja Przyborski and Monika Wohlrab-Sahr. *Qualitative Sozialforschung: Ein Arbeitsbuch*. Walter de Gruyter, 2009 (cit. on p. 82).

[RM17] P. Rempel and P. Mäder. “Preventing Defects: The Impact of Requirements Traceability Completeness on Software Quality”. In: *IEEE Transactions on Software Engineering* 43.8 (Aug. 2017), pp. 777–797 (cit. on p. 111).

[Ren+12] Karl-Heinz Renner, Timo Heydasch, Gerhard Ströhlein, Timo Heydasch, and Gerhard Strohlein. *Forschungsmethoden der Psychologie - Von der Fragestellung zur Präsentation*. 1. Aufl. 2012. Berlin Heidelberg New York: Springer-Verlag, 2012 (cit. on pp. 20, 21, 201).

[Rie11] Eric Ries. *The lean startup: How today’s entrepreneurs use continuous innovation to create radically successful businesses*. Crown Books, 2011 (cit. on p. 168).

[RET19] Florian Rittmeier, Gregor Engels, and Alexander Teetz. “Process Weakness Patterns for the Identification of Digitalization Potentials in Business Processes”. In: *Business Process Management Workshops*. Ed. by Florian Daniel, Quan Z. Sheng, and HamidEditors Motahari. Vol. 342. Lecture Notes in Business Information Processing. Springer International Publishing, 2019, pp. 531–542 (cit. on p. 82).

[RR06] Suzanne Robertson and James Robertson. *Mastering the Requirements Process*. 2. Edition. Addison-Wesley Professional, 2006 (cit. on pp. 102, 110).

[Rog10] Everett M Rogers. *Diffusion of innovations*. Simon and Schuster, 2010 (cit. on pp. 7, 31, 32, 35–39, 69, 74, 171).

[RB04] Robert Rousseau and Richard Breton. “The M-OODA: A model incorporating control functions and teamwork in the OODA loop”. In: *Proc. Command and Control Res, & Tech. Symp*. 2004 (cit. on p. 122).

[Sch17] Daniel R.A. Schallmo. *Design Thinking erfolgreich anwenden*. Springer Fachmedien Wiesbaden, 2017 (cit. on pp. 61, 62).

[Sch19] Peter Schick. “Design and Implementation of an Online Feature Experimentation Platform”. MA thesis. 2019 (cit. on p. 146).

[Sen+18] Björn Senft, Holger Fischer, Simon Oberthür, and Nitish Patkar. “Assist Users to Straightaway Suggest and Describe Experienced Problems”. In: *Design, User Experience, and Usability: Theory and Practice*. Ed. by Aaron Marcus and Wentao Wang. Cham: Springer International Publishing, 2018, pp. 758–770 (cit. on pp. 27, 51, 68).

[SFS14] Björn Senft, Holger Fischer, and Christian Sudbrock. “IT-Unterstützung im praktischen Ausbildungsbetrieb der Feuerwehr”. In: *Mensch & Computer 2014 - Workshopband*. De Gruyter Oldenbourg, 2014, pp. 111–116 (cit. on pp. 26, 27, 172).

[SO16] Björn Senft and Simon Oberthür. “Auf dem Weg zu einer experimentellen und evidenzbasierten Softwareentwicklung in den Digital Humanities”. In: *Konferenzabstracts der 3. Tagung des Verbands "Digital Humanities im deutschsprachigen Raum e. V."* 2016 (cit. on p. 27).

[SOF18] Björn Senft, Simon Oberthür, and Holger Fischer. “Forschendes Lernen in der Informatik - In praxisnaher Projektgruppe einen Softwareentwicklungsprozess erforschen”. In: *Schriften zur allgemeinen Hochschuldidaktik - Band 3. Tagungsband Forschendes Lernen - The wider view*. 2018 (cit. on pp. 24, 27, 164).

[Sen+19] Björn Senft, Florian Rittmeier, Holger Fischer, and Simon Oberthür. “A Value-Centered Approach for Unique and Novel Software Applications”. In: *Design, User Experience, and Usability. Practice and Case Studies*. 2019 (cit. on p. 27).

[Sen20a] Senft, Björn. *Design Thinking - Empathy Phase*. `https://uni-paderborn.sciebo.de/s/sxImLBpQD1DkRa8`. Accessed: 2020-02-03. 2020 (cit. on pp. 91, 93, 184, 196).

[Sen20b] Senft, Björn. *Design Thinking Theory*. `https://uni-paderborn.sciebo.de/s/6b0SdXy6R2e1rYO`. Accessed: 2020-02-03. 2020 (cit. on pp. 183, 184, 194, 198).

[Sen20c] Senft, Björn. *Design Thinking Workshop - Day's Goal*. `https://uni-paderborn.sciebo.de/s/dM2VdjX3CZJ5CSZ`. Accessed: 2020-02-03. 2020 (cit. on p. 90).

[Sen20d] Senft, Björn. *Design Thinking Workshop - Presentation*. `https://uni-paderborn.sciebo.de/s/Z3OL2EzfYUiWqgY`. Accessed: 2020-02-03. 2020 (cit. on pp. 90–92, 95).

[Sen20e] Senft, Björn. *Design Thinking Workshop - Solution Documentation*. `https://uni-paderborn.sciebo.de/s/PX9mmEdByEApyzr`. Accessed: 2020-02-03. 2020 (cit. on p. 93).

[Sen20f] Senft, Björn. *Design Thinking Workshop - Working Sheets*. `https://uni-paderborn.sciebo.de/s/pcwoJ8oshYirYfl`. Accessed: 2020-02-03. 2020 (cit. on pp. 91, 92, 184, 199, 200).

[Sen20g] Senft, Björn. *Onboarding Slides*. `https://uni-paderborn.sciebo.de/s/zy42PZz3bDpmzXl`. Accessed: 2020-02-03. 2020 (cit. on pp. 182, 205).

[Sen20h] Senft, Björn. *Scrum Introduction / Lego 4 Scrum Workshop*. `https://uni-paderborn.sciebo.de/s/4l7BHqNZ9IH55r2`. Accessed: 2020-02-03. 2020 (cit. on p. 183).

[Sen20i] Senft, Björn. *The Wallet Project - Facilitators Guide*. `https://uni-paderborn.sciebo.de/s/JU6MNQ9sqePvFXV`. Accessed: 2020-02-03. 2020 (cit. on p. 183).

[Sen20j] Senft, Björn. *The Wallet Project - Working Sheets*. `https://uni-paderborn.sciebo.de/s/5wtIxs7UBtzG9NH`. Accessed: 2020-02-03. 2020 (cit. on pp. 91, 183).

[SAZ17] M. Shahin, M. Ali Babar, and L. Zhu. “Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices”. In: *IEEE Access* 5 (2017), pp. 3909–3943 (cit. on p. 137).

[SC17] S Sharma and B Coyne. *DevOps for Dummies. 3rd Limited IBM edn*. 2017 (cit. on pp. 9, 54, 135).

[Soh+19] Osama Sohaib, Hiralkumari Solanki, Navkiran Dhaliwa, Walayat Hussain, and Muhammad Asif. “Integrating design thinking into extreme programming”. In: *Journal of Ambient Intelligence and Humanized Computing* 10.6 (June 2019), pp. 2485–2492 (cit. on p. 56).

[SS97] Ian Sommerville and Pete Sawyer. *Requirements Engineering: A Good Practice Guide*. 1st. New York, NY, USA: John Wiley & Sons, Inc., 1997 (cit. on p. 72).

[Ste+04] Jonette M Stecklein, Jim Dabney, Brandon Dick, et al. “Error cost escalation through the project life cycle”. In: (2004) (cit. on pp. 9, 48, 119, 168).

[SF18] Klaas-Jan Stol and Brian Fitzgerald. “The ABC of software engineering research”. In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 27.3 (2018), p. 11 (cit. on pp. 23, 121).

[SC94] Anselm Strauss and Juliet Corbin. “Grounded theory methodology”. In: *Handbook of qualitative research* 17 (1994), pp. 273–85 (cit. on pp. 79, 80).

[Str+96] Anselm L Strauss, Juliet M Corbin, Solveigh Niewiarra, and Heiner Legewie. *Grounded theory: Grundlagen qualitativer sozialforschung*. Beltz, Psychologie-Verlag-Union Weinheim, 1996 (cit. on p. 24).

[Str04] Jörg Strübing. *Grounded Theory*. VS Verlag für Sozialwissenschaften, 2004 (cit. on pp. 78–81).

[SS13] Alistair Sutcliffe and Pete Sawyer. “Requirements elicitation: Towards the unknown unknowns”. In: *2013 21st IEEE International Requirements Engineering Conference (RE)*. IEEE. 2013, pp. 92–104 (cit. on pp. 67, 77).

[TC12] Dana H Taplin and Heléne Clark. “Theory of change basics: A primer on theory of change”. In: *New York: Actknowledge* (2012) (cit. on p. 72).

[Ter+17] Henri Terho, Sampo Suonsyrjä, Kari Systä, and Tommi Mikkonen. “Understanding the relations between iterative cycles in software engineering”. In: *Proceedings of the 50th Hawaii International Conference on System Sciences*. 2017 (cit. on pp. 54, 183).

[Tho03] Stefan H Thomke. *Experimentation matters: unlocking the potential of new technologies for innovation*. Harvard Business Press, 2003 (cit. on pp. 121, 123).

[Thö15] J. Thönes. “Microservices”. In: *IEEE Software* 32.1 (2015), pp. 116–116 (cit. on p. 170).

[Ueb+15] Falk Uebernickel, Walter Brenner, Britta Pukall, Therese Naef, and Bernhard Schindlholzer. *Design Thinking: Das Handbuch*. Frankfurter Allgemeine Buch, 2015 (cit. on pp. 43, 62).

[UD10] Daniel Ullrich and Sarah Diefenbach. “INTUI. Exploring the facets of intuitive interaction”. In: *Mensch & Computer 2010: Interaktive Kulturen* (2010) (cit. on p. 175).

[VZ09] Vero Vanden Abeele and Bieke Zaman. *Laddering the User Experience!* eng. 2009 (cit. on p. 72).

[War+95] Allen Ward, Jeffrey K Liker, John J Cristiano, Durward K Sobek, et al. “The second Toyota paradox: How delaying decisions can make better cars faster”. In: *Sloan management review* 36 (1995), pp. 43–43 (cit. on pp. 9, 47, 48, 98, 111, 168, 183, 222).

[Wei02] Laura Weiss. “Developing tangible strategies”. In: *Design Management Journal (Former Series)* 13.1 (2002), pp. 33–38. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1948-7169.2002.tb00296.x` (cit. on pp. 40, 41).

[Wil+89] Michael Wilson et al. “Task models for knowledge elicitation”. In: *Knowledge Elicitation: principles, techniques and applications* (1989), pp. 197–220 (cit. on p. 72).

[WH10] Brigitte Winkler and Helmut Hofbauer. “Das Mitarbeitergespräch als Führungsinstrument”. In: *Handbuch für Führungskräfte und Personalverantwortliche* 4 (2010) (cit. on p. 25).

[WS95] Jane Wood and Denise Silver. *Joint Application Development (2Nd Ed.)* New York, NY, USA: John Wiley & Sons, Inc., 1995 (cit. on p. 72).

[XAA15] Bianca H. Ximenes, Isadora N. Alves, and Cristiano C. Araújo. “Software Project Management Combining Agile, Lean Startup and Design Thinking”. In: *Design, User Experience, and Usability: Design Discourse*. Ed. by Aaron Marcus. Cham: Springer International Publishing, 2015, pp. 356–367 (cit. on p. 56).

[Yin17] Robert K Yin. *Case study research and applications: Design and methods*. Sage publications, 2017 (cit. on p. 163).

[Yoz14] K. Yozgyur. “A proposal for a requirements elicitation tool to increase stakeholder involvement”. In: *2014 IEEE 5th International Conference on Software Engineering and Service Science*. June 2014, pp. 145–148 (cit. on pp. 68, 69).

[Zha07] Zheying Zhang. “Effective requirements development-a comparison of requirements elicitation techniques”. In: *Software Quality Management XV: Software Quality in the Knowledge Society, E. Berki, J. Nummenmaa, I. Sunley, M. Ross and G. Staples (Ed.) British Computer Society* (2007), pp. 225–240 (cit. on p. 70).

[ZC05] Didar Zowghi and Chad Coulin. “Requirements Elicitation: A Survey of Techniques, Approaches, and Tools”. In: *Engineering and Managing Software Requirements*. Ed. by Aybüke Aurum and Claes Wohlin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 19–46 (cit. on pp. 70, 72).

## Internet

[@Bes10] Martin Beschnitt. *Verbreitung und Relevanz von Usability-Methoden in der Praxis*. 2010. URL: `https://www.usabilityblog.de/verbreitung-und-relevanz-von-usability-methoden-in-der-praxis/` (visited on Apr. 2, 2020) (cit. on p. 125).

[@Big18] Garenne Bigby. *10 Card Sorting Tools for Surveying Information Architecture (IA)*. 2018. URL: https://dynomapper.com/blog/19-ux/428-card-sorting-tools (visited on Dec. 4, 2019) (cit. on p. 71).

[@BI16] Anna Blaylock and Navin Iyengar. *Art vs. Science: Using A/B Testing To Inform Your Designs (Netflix at Designers + Geeks)*. 2016. URL: https://www.youtube.com/watch?v=RHWVWiiW8DQ (visited on Apr. 20, 2018) (cit. on p. 3).

[@Bot16] Thomas Both. *The Wallet Project*. 2016. URL: https://dschool-old.stanford.edu/groups/designresources/wiki/4dbb2/the_wallet_project.html (visited on Jan. 14, 2020) (cit. on p. 94).

[@dsc16] Stanford d.school. *Create Design Challenges Guidelines*. 2016. URL: https://dschool-old.stanford.edu/groups/k12/wiki/613e8/Create_Design_Challenges_Guidelines.html (visited on Jan. 10, 2019) (cit. on p. 51).

[@DS19] Rikke Dam and Teo Siang. *5 Stages in the Design Thinking Process*. 2019. URL: https://www.interaction-design.org/literature/article/5-stages-in-the-design-thinking-process (visited on Sept. 18, 2019) (cit. on pp. 42, 43).

[@DS20] Rikke Dam and Teo Siang. *Stage 2 in the Design Thinking Process: Define the Problem and Interpret the Results*. 2020. URL: https://www.interaction-design.org/literature/article/stage-2-in-the-design-thinking-process-define-the-problem-and-interpret-the-results (visited on May 18, 2020) (cit. on p. 43).

[@Dep18] Paderborn University Department of Computer Science. *General Guidelines - Project Groups*. 2018. URL: https://cs.uni-paderborn.de/en/studies/study-elements/general-guidelines/project-groups/ (visited on Oct. 10, 2019) (cit. on p. 178).

[@Doo+18] Scott Doorley, Sarah Holcomb, Perry Klebahn, Kathryn Segovia, and Jeremy Utley. *Design Thinking Bootleg*. 2018. URL: https://dschool.stanford.edu/resources/design-thinking-bootleg (visited on Sept. 18, 2019) (cit. on pp. 42, 43).

[@Fow14] Martin Fowler. *Bounded Context*. 2014. URL: https://www.martinfowler.com/bliki/BoundedContext.html (visited on Apr. 9, 2020) (cit. on p. 131).

[@Fow11] Martin Fowler. *CQRS*. 2011. URL: https://martinfowler.com/bliki/CQRS.html (visited on Apr. 9, 2020) (cit. on p. 133).

[@Fow05a] Martin Fowler. *Event Sourcing*. 2005. URL: https://martinfowler.com/eaaDev/EventSourcing.html (visited on Jan. 10, 2020) (cit. on pp. 54, 131).

[@Fow06] Martin Fowler. *GUI Architectures*. 2006. URL: https://martinfowler.com/eaaDev/uiArchs.html (visited on Apr. 9, 2020) (cit. on pp. 48, 135).

[@Fow05b] Martin Fowler. *Parallel Model*. 2005. URL: https://martinfowler.com/eaaDev/ParallelModel.html (visited on Apr. 9, 2019) (cit. on p. 131).

[@Fow15] Martin Fowler. *Presentation Domain Data Layering*. 2015. URL: https://martinfowler.com/bliki/PresentationDomainDataLayering.html (visited on Apr. 7, 2020) (cit. on p. 134).

[@Fow03] Martin Fowler. *Presentation Domain Separation*. 2003. URL: https://martinfowler.com/bliki/PresentationDomainSeparation.html (visited on Apr. 7, 2020) (cit. on p. 134).

[@Fow04] Martin Fowler. *Presentation Model*. 2004. URL: https://www.martinfowler.com/eaaDev/PresentationModel.html (visited on Apr. 7, 2020) (cit. on p. 135).

[@Fow10] Martin Fowler. *Richardson Maturity Model*. 2010. URL: https://martinfowler.com/articles/richardsonMaturityModel.html (visited on Apr. 10, 2020) (cit. on p. 138).

[@Hod17] Pete Hodgson. *Feature Toggles (aka Feature Flags)*. 2017. URL: https://martinfowler.com/articles/feature-toggles.html (visited on Apr. 14, 2020) (cit. on p. 147).

[@KNO18] Nicola Karthaus, Antje Nöhren, and Simon Oberthür. *MACHBARKEITSSTUDIE DIGITALES OWL•KULTUR-PORTAL*. 2018. URL: https://www.ostwestfalen-lippe.de/images/stories/Machbarkeitsstudie_Kultur-Portal_web.pdf (visited on Oct. 2, 2019) (cit. on p. 180).

[@LF14] James Lewis and Martin Fowler. *Microservices - a definition of this new architectural term*. 2014. URL: https://martinfowler.com/articles/microservices.html (visited on Apr. 20, 2020) (cit. on p. 170).

[@LS16] Ben Linders and Dave Snowden. *Q&A with Dave Snowden on Leadership and Using Cynefin for Capturing Requirements*. 2016. URL: https://www.infoq.com/articles/dave-snowden-leadership-cynefin-requirements (visited on June 27, 2018) (cit. on p. 6).

[@Mic12] Microsoft. *The MVVM Pattern*. 2012. URL: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10) (visited on Apr. 8, 2020) (cit. on p. 136).

[@Neg17] Julia Negri. *Kultur digitalisieren: SICP ist Mitinitiator beim Infotag "OWL.Kultur-Portal"*. 2017. URL: https://www.sicp.de/nachricht/news/kultur-digitalisieren-sicp-ist-mitinitiator-beim-infotag-owlkultur-portal/ (visited on Oct. 2, 2019) (cit. on p. 179).

[@Nes17] Nesta. *Tools - Development Impact and You*. 2017. URL: http://diytoolkit.org/tools/ (visited on Dec. 3, 2019) (cit. on p. 72).

[@Nie01] Jakob Nielsen. *First Rule of Usability? Don't Listen to Users*. 2001. URL: https://www.nngroup.com/articles/first-rule-of-usability-dont-listen-to-users/ (visited on Dec. 4, 2019) (cit. on p. 70).

[@Osm15] Addy Osmani. *Getting Started with Progressive Web Apps*. 2015. URL: https://developers.google.com/web/updates/2015/12/getting-started-pwa (visited on Apr. 12, 2020) (cit. on p. 139).

[@Roy+16] Adam Royalty, Rich Crandall, Katie Krummeck, and Devon Young. *Create Design Challenges Guidelines*. 2016. URL: https://dschool-old.stanford.edu/groups/k12/wiki/613e8/Creating_Design_Challenges.html (visited on July 10, 2019) (cit. on p. 61).

[@RB15] Alex Russel and Frances Berriman. *Progressive Web Apps: Escaping Tabs Without Losing Our Soul*. 2015. URL: https://medium.com/@slightlylate/progressive-apps-escaping-tabs-without-losing-our-soul-3b93a8561955 (visited on Apr. 12, 2020) (cit. on p. 139).

[@SS17] Ken Schwaber and Jeff Sutherland. *The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game*. 2017. URL: https://www.scrumguides.org/scrum-guide.html (visited on Apr. 17, 2020) (cit. on p. 167).

[@Sev14] Doug Seven. *Knightmare: A DevOps Cautionary Tale*. 2014. URL: https://dougseven.com/2014/04/17/knightmare-a-devops-cautionary-tale/ (visited on Jan. 10, 2019) (cit. on pp. 9, 14, 150).

[@Smi09] Josh Smith. *Patterns - WPF Apps With The Model-View-ViewModel Design Pattern*. 2009. URL: https://docs.microsoft.com/en-us/archive/msdn-magazine/2009/february/patterns-wpf-apps-with-the-model-view-viewmodel-design-pattern (visited on Apr. 7, 2020) (cit. on p. 135).

[@Tra16] Norman Tran. *Design Thinking Playbook*. 2016. URL: https://dschool.stanford.edu/resources/design-thinking-playbook-from-design-tech-high-school (visited on July 10, 2019) (cit. on pp. 90, 91, 93, 199, 200).

[@USK16] Steve Urban, Rangarajan Sreenivasan, and Vineet Kannan. *It's All A/Bout Testing: The Netflix Experimentation Platform*. 2016. URL: https://medium.com/netflix-techblog/its-all-a-bout-testing-the-netflix-experimentation-platform-4e1ca458c15 (visited on Apr. 20, 2018) (cit. on p. 156).

[@Wal12] Gerd Waloszek. *Introduction to Design Thinking*. 2012. URL: https://experience.sap.com/skillup/introduction-to-design-thinking/ (visited on Sept. 18, 2019) (cit. on p. 89).

# List of Acronyms

**HiP-App** *History in Paderborn App*

**FF** *Fitness Function*

**POV** *Point of View*

**nlp** *natural language processing*

**OWL** *Ostwestfalen-Lippe*

# List of Figures

# List of Tables