KEIL-SLAWIK, REINHARD

IMPARTING PRACTICAL SKILLS IN SOFTWARE ENGINEERING

KONZEPTION EINES SOFTWARETECHNIK - PRAKTIKUMS

1. INTRODUCTION

Software has become a key factor in systems development in view of increasing computerization in all fields. Consequently, universities are called upon to direct research and teaching efforts towards improving software engineering techniques.

Training software engineers requires setting problems of a realistic but flexible nature, which allow various problem dimensions to be dealt with : requirements analysis, user/developer communication, interface design, incremental system development, team organisation and project management. Different tools and techniques have to be used in different phases of the development process for different purposes.

Computer science students at the Technical University of Berlin have the option of participating in a software engineering project lasting two semesters. The participants are required to complete an introductory course beforehand. This three-semester cycle has not proved sufficiently flexible to accommodate the increasing number of students each semester.

A new course has therefore been devised by the author, enabling students to gain practical experience in software engineering within a single semester. Again, completion of the introductory course is compulsory.

Before giving a more detailed description of this course, we present a brief outline of our methodical approach STEPS (Software Technology for Evolutionary Participative Systems Development) [1]. The skills we aim to impart are discussed with reference to STEPS and to the overall teaching situation at the university.

2. PROCESS-ORIENTED SOFTWARE DEVELOPMENT

STEPS embodies a different view of software development from that advocated by conventional approaches [2]. While in conventional approaches software is regarded as an autonomous product, STEPS focusses on the processes of development and use of software.

According to our process-oriented approach, as described in [3], we regard software development in the wider context of systems development. It is thus seen as part of an ongoing organisational process designed to improve an organisation's ability to pursue specific goals. Hence, the organisational context, i.e. the usage context of the desired DP system, has to be taken into account. The adequacy of DP systems with respect to the users' work tasks is one of our basic quality criteria [4].

STEPS comprises a process-oriented view, human-centred quality criteria, a cyclic development model, gestalt-forming project techniques, and component methods for requirements analysis, dialogue specification and modular design. Depending on the actual problem setting, these components have to be arranged, modified, or even partly replaced by other components.

3. THE OVERALL CURRICULAR SITUATION

The introductory course which the students have to complete before attending the practical course does not aim to give a lexicographical overview of the wide variety of tools and techniques in use in software engineering. The main intention is rather to enable students to gain a substantial understanding of the process of software development with the help of our methodical approach. This is followed up by short practical exercises.

Practical skills can, however, only be acquired by working in a real project situation where at least one development cycle, from problem analysis to implementation and system evaluation, is run through. Thus, in the practical course, the students are required to develop a DP system in a team context, with all relevant technical aspects being touched upon:

- problem analysis
- interface specification
- modular design
- implementation
- use of tools and description techniques.

In addition to these technical aspects, the participants have to learn

- to discuss requirements with persons from outside the project
- to communicate on the basis of written documents
- to cooperate in different teams
- to document the work they have done.

In order to achieve this, it is necessary to define a problem which allows us to cope with all these aspects effectively within one semester. The problem must, then, relate to the students' own field of experience. This has been accomplished in three ways:

1. A students' accommodation office was selected as the problem to be dealt with, the main aim being to computerize the process of allocating requests for accommodation to suitable offers. Though most of the students are acquainted with this problem, it still raises a good many unfamiliar questions which require consideration in the development process.

2. The tools and techniques used are the same as those employed in the introductory course, with the exception of the programming language (now COBOL). However, the tools have been adapted to COBOL to make the programming task easier.

3. In each development phase throughout the course, the students arrive at partial solutions which have to be extended (in most cases), or modified, or adapted to actual needs.

Finally, a variety of solutions is possible, depending on the actual project course. Each solution must, however, be based on a contractual agreement with the so-called 'virtual user'.

4. PROJECT MANAGEMENT

The participants are divided up into four tutorials, each of which is headed by a student teacher (tutor) throughout the semester. A tutorial consists of up to 16 students, organised into four teams. Each tutorial is required to develop a system, the individual teams working on their own specific problems in the respective development phase.

The development phases can be roughly divided into: requirements analysis, functional specification, modular design, implementation, and evaluation.

The tutorials meet twice a week, the meetings being supervised by the tutor.

In order to promote communication among teams and to afford each participant an overall view of the project, we use a number of different techniques to organise the development process:

- Work review: Each team is required to present its work at the weekly meetings during the first two phases.
- Task exchange: Solutions worked upon by a team in one phase are further processed by another team in the next pahse, e.g. one team specifying and another implementing a module.
- Intercommunicative groups: Specific tasks related to the work being done in individual teams are carried out by ad hoc intercommunicative groups (comprising one member from each team). Such tasks include: the definition of the basic functionality, the overall modular design, the integration test, and the negotiations with the virtual user.
- Virtual user: This role is played by the course teacher, who is not involved in any of the tutorials and who acts as the client. His job is to accept the final product, i.e. the program text along with its documentation.

As may be surmised, the overall project plan is quite complicated. Additional difficulties may also arise during the course if there is a lack of team spirit among the groups, or if a team delivers its work later than scheduled, or if any hardware problems occur.

We have nevertheless succeeded in establishing a rigorous project plan, at the same time providing mechanisms to cope flexibly with the actual project situation by redefining contractual agreements with the virtual user. We therefore feel entitled to claim that the course participants are given a fairly accurate taste of a real-life project situation.

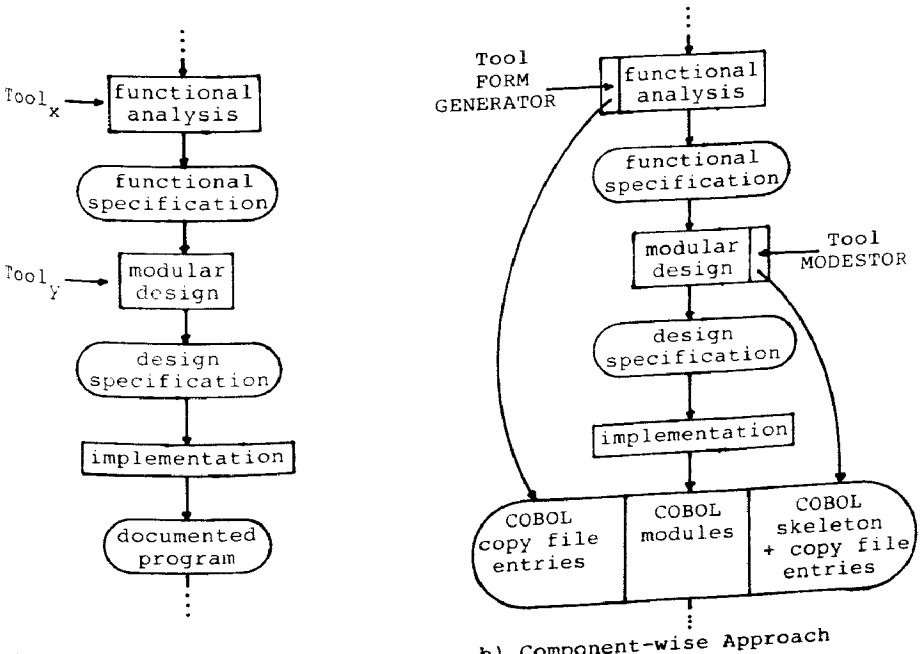
5. TOOLS AND TECHNIQUES

The rigorous project plan allows no time for experiments or evolutionary development techniques like prototyping or incremental software development, as might be suggested by the philosophy of STEPS.

Tools and techniques have to be applied in a straightforward manner.

We have therefore adapted our tool to the principle of component-wise development. By using this principle, we aim to overcome the problems inherent in the conventional phase model strategy, such as found in software engineering textbooks [5]. A phase comprises a set of related activities, whereby the result produced is related to the whole system. Tools used in one phase aim at supporting the activities designed to produce that result. Since in each case the result of the previous phase is taken as input for the next phase, so as to produce a new result providing the necessary input for the subsequent phase, we call this strategy **transformational** (see Figure 1a).

In contrast, we propose a **component-wise** strategy. By this, we mean that partial solutions, i.e. formalizations being done in one phase and covering only part of the desired result, should be directly implemented by means of suitable tools without being transformed in intermediate phases as part of a complete result (see Figure 1b).



a) Transformational Approach

b) Component-wise Approach

Figure 1. Development Strategies

During the functional analysis phase, for instance, students have to specify forms by means of a form generator in such a way that the forms generated can be demonstrated to the virtual user. We have adapted the form generator by writing a small program that takes the internal data structures produced by the form generator as input and produces COBOL code to process forms by means of the COBOL copy mechanism. Another tool used in the modular design phase (MODESTOR) supports the activity of specifying the system architecture by means of defining object modules, type modules, and the associated module interfaces. MODESTOR is based on the specification language MODEST [6]. It not only checks the module specification for completeness and consistency, but also produces a COBOL program skeleton according to our COBOL programming conventions, and defines copy file entries for the specified type definitions.

The component-wise approach means, then, that the students are working on a sound basis with respect to the desired implementation right from the start of the project.

REFERENCES

[1] Floyd, C.: **STEPS- Eine Orientierung der Softwaretechnik auf sozialverträgliche Technikgestaltung**; to be published in: Report of the Fachbereich Informatik, Universität Dortmund; 1987

[2] Floyd, C.: **Outline of a Paradigm Change in Software Engineering**; in: Bjerkness, G., Ehn, P., Kyng, M. (eds.): Computers and Democracy. A Scandinavian Challenge; Gower Publishing; Hampshire; 1987

[3] Floyd, C., Keil, R.: **Adapting Software Devleopment for Systems Design with Users**; in: Brief, U., Ciborra, C., Schneider, L. (eds.): System Design For, With, and By the Users; North-Holland; Amsterdam, New York, Oxford; 1983

[4] Keil-Slawik, R.:**Supporting Participative Systems Development By Task-oriented Requirements Analysis**; Proceedings of the IFIP TC9/WG 9.1 Conference on "System Design for Human Development and Productivity: Participation and Beyond", Berlin, GDR, 12-15 May 1986; Supplement Volume; to be published by North-Holland; Amsterdam, New York, Oxford; 1987

[5] Kimm, R., Koch, W., Simonsmeier, T., Tontsch, F.: **Einführung in Software Engineering**; de Gruyter; Berlin, New York; 1979

[6] Pasch, J., Schmidt, G.: **MODEST - Eine modulare Entwurfstechnik und eine darauf abgestimmte Entwurfssprache**; in: Wippermann, H.-W. (Hrsg.): Software-Architektur und modulare Programmierung; Teubner; Stuttgart; 1986