Tóth, Gy. F., Johanyák, Zs. Cs.: Teaching software engineering – Experiences and new Approaches, XIX Didmattech 2006, September 6—7, Komarno, Slovakia ,
ISBN 978-80-89234-23-3 , pp. 261-265.

TEACHING SOFTWARE ENGINEERING – EXPERIENCES AND NEW APPROACHES

TÓTH György Ferenc, HU – JOHANYÁK Zsolt Csaba, HU

Resume International statistics show that more than a half of the software projects fail before any user could try the product. The general opinion is that the fiasco can be traced back to the lack of proper design work. It is a sign that made us to meditate on our teaching and curriculum developing practice. This paper intends to present our experience and a new strategy aiming a better training of our students for real-world software engineering tasks.

Keywords: software engineering curriculum

1 Introduction

According to international statistics more than a half of the software projects fail before any user could try the product. A widely held view is that the fiasco rises from the shortcomings or even from the lack of the design. Our opinion is that the roots of the problem partly can be traced back to the attitude of the specialists trained in the higher education. What do they consider important during their work? What do they emphasize in the software development process? Certainly the coding, the topic that was the most examined skill during their training. They consider coding the only important matter and they neglect all that either have been not treated as important during their studies or was not part of their curriculum.

As a result of the review of the curriculum of several higher education institutes that are involved in teaching programming and software engineering it can be summarized that generally the training of programming is organized as follows. The starting point is algorithms, followed by mastering a programming language in two or three semesters attained in the end to the grounding and implementations of the object-oriented attitude. Software engineering knowledge is transferred in the end mostly in courses with few contact hours. However, nowadays software development is a managed business process, where the language of the implementation fairly often is a matter of details and a significant part of the problem can be solved in the first phase by applying prefabricated software components.

The rest of this paper is organized as follows: section 2 gives a short overview of the training of application development in Kecskemét College by presenting the objectives and aims, the syllabus and the recommended positions in the curriculum of studies of the course-units. The experiences and results are emphasized, as well. Section 3 outlines a new approach that has been developed based on the analysis of the assessments of students and feedback received from the labour market.

2 Teaching software engineering in the information engineering branch

The students of Kecskemét College attain the basic programming skills in the information engineering branch in three semesters. In course of the unit *Problem Classes, Algorithms* placed in the first semester, they go deeply into the concepts of algorithms and they get acquainted with the related description methods. The basics of

C programming language are taught during the second semester. This unit is called *Programming I.* and students get two laboratory and two lecture contact hours a week. The evaluation of their work is continuous. In the next semester comes the course unit *Programming II.* that also contains two hours laboratory and two hours lecture per week. It continues the training of the C language extended with non object-oriented elements of C++. Relevant topics are data structures, file input-output operations, functions, dynamic memory management, etc. The students have to take an exam at the end of the course. The unit *Programming Paradigms and Techniques* comes in the next semester. Its syllabus comprehends the basics of Object-Oriented Programming (OOP) using the C++ programming language. The number of the weekly contact hours is the same. The assessment form is continuous. The students are suggested to register for the *Comprehensive Examination* in programming only after carrying out successfully these three programming courses. However, it is not a compulsory precondition. Surprisingly, in each examination session some students can be found trying their knowledge and skills before the end of the triple programming course and some of them are lucky. It should be mentioned that the students are trained in database management parallel with the programming courses in the second and third semesters.

The unit *Programming Paradigms and Techniques* is followed by the compulsory *Visual Programming* class. The students get acquainted in its frames with the visual application development through the use of a high level development tool (Visual Studio 2005 Professional), which supports Rapid Application Development techniques. They also learn a new object-oriented language called C#. The contact hours are two lectures and two laboratories, as usual. In parallel with the VP appears the subject *Software engineering* in form of two lecture hours a week. This is the first time the students familiarize themselves with the different models and methodologies, CASE tools, as well as concepts and construction practice of UML diagrams.

During the development the syllabus for the course units we have endeavoured to compile knowledge and materials in such way to obtain an optimal proportion between the general (not becoming obsolete) and the up to date (well applicable in the practice) information. The languages C and C++ have been chosen for the founding training considering that the basic software and the operating systems have been developed mostly in these languages to the present day and it does not seem to be programming language that could take over this task. The .Net programming has been selected as a topic for the visual application development. This decision was determined by the demand of our students as well by the availability of a high level Integrated Development Environment. The Visual Studio was available and could be checked out freely by our students before 2005 in the frames of an MSDNAA subscription and after 2005 owing to the project Clean-Software [1] (previously called Campus) financed by the Hungarian Government. The *Java Based Development* certainly is part of our training palette. This introductory course-unit is compulsory for the students who select the specialisation in *Network and Web Technologies* and it is freely selectable for the other students of the information engineering branch with two lecture and two laboratory contact hours a week.

We follow the traditional and widespread programming-teaching curriculum in our training. However, we are not as effective as we wish. The experiences show that the results of the assessments of the programming and software-development courses are poorer than the results in mathematics, which subject is traditionally considered as one of the most difficult ones. Less than thirty percent of the students complete their courses successfully. Another problem is that according to the feedback of the labour market the trained professionals do not frequently follow that software development methodology that the employers expect based on the needs of the long term cost-effective development and production of standing values. Sometimes the approach of the young professionals can be considered hurry-scurry rather than systematic. Undoubtedly there are some elements in our curriculum that could be misleading for students being not enough engrossed in the subject. As a very simple example could be mentioned the instruction of some techniques (e.g. the use of global variable) without which starting teaching a programming language would be too complicated. However, later their use is forbidden and we try to break the students' habit of applying them.

In our observation it is determinant for the students the experience of meeting the program development for the first time. At this time they are still open to all techniques. They try in a natural way to understand, comprehend and learn the attitude they will be later supported by during their software development activity. Currently this determinant impression consists of an implementation practice where the instructor presents the specification briefly, which cannot be understandable or obscure for some students, and next students try to code the program rapidly without any design work or examination of the problem. Sometimes their approach can be traced back either to the small scale of the problem or to the fact that the problem seems to have a known solution. It frequently occurs that the students do not think ahead, they decide from line to line what mathematical equation to apply, what to organize inside or outside of an iteration, if selection statements are necessary or not. Studying the examples treated on programming laboratories and the solutions of the assignments made by the students it easily can be observed that they do not look like a source code developed systematically in the competitive sphere. We should admit that sometimes even the tutorial examples prepared by the teachers contain only the explications of the new functions and solution types. Due to the complete absence of documentation the students would not even be able to test the programs developed by their fellows. In the best possible case the name of the author appears in the source code but there is no description of the task and the objective of the software. Usually the subroutines are not preceded by the design and the annotation of the variables. One can not understand how the output comes from the input, which makes difficult to control or test the program or even the subroutines.

All the enumerated problems can be traced back to the fact that we teach a programming language and not software engineering. We should recognize that our unsaid, latent primary objective was to make the students understand the language elements and to make them exercise the methods being typical of the language. Moreover the instructors also prepared their syllabus and examples based on this approach. Another difficulty arises as a consequence of the reforms regarding the number of the contact hours, namely the available time is hardly enough for teaching the basic elements of a programming language. Therefore the less conspicuous topics are omitted in the teaching practice, namely the design and testing that can run up to 80% of the life-time of a software project in the competitive sphere.

Based on the results and the feedback (e.g. [3]) we have drawn the conclusion that we should rebuild the training of programming and software development in our college. The emphasized topics are changed in the new approach and besides the teaching methodology is also modified in order to reach a better fitting to the logical steps of the software development.

The implementation phase of the software development is not other than the conversion of the system specification into a system that can be run [2]. Thus application development can not exist without system specification, although we also make specifications in the traditional programming teaching. We call it problem definition or problem specification and we use our mother tongue (a human language) for its description. However, this approach has its serious limitations. The specifications easily can be misunderstood; they depend on the way of thinking and conception of the person who is defining them. One can not speak about a standard. Thus students are inclined to think that the system-design is an obscure description that is full with subjective concepts and based on it they can code anything they feel right or anything that they can understand from it. Therefore it is not a surprising fact that most of our students do not read the problem specification even during their exams. They begin the implementation after reading a few sentences from the specification. Practically there is no endeavour to establish the structure of the software first outgoing from the description and next to develop the program based on it. In all the cases the students appreciate their software as perfect if it conforms to the requirement specification although none of the traces of the originally documented and required specification can be identified in it. On the other hand the software development is going on in the competitive sector conform to a very strict system-design, not to mention the coding conventions, which most of the students face for the first time at their workplace.

In our opinion the traditional programming education has another serious drawback. It does not prepare the students for team-work, for the fact that in the software industry the applications are not developed by individuals and in arbitrary mode, but several people work on a project. In practice the preparation of standard system-designs, their understanding and the implementation of the programs based on them will be overall important. Unfortunately most of our students have no idea about the software development in teams, because our training system is based on the individual performance, only this skill is assessed during the studies. In the competitive sphere nowadays the one-person projects have no chance against the software projects developed systematically in team-work by larger software companies. Exactly for that reason we have decided to modify the teaching of programming in our college with an experimental character. We want to train our students not only to be able to develop software as isolated individuals but also to be good team-players.

3 New approaches in the teaching of the application development

We prepare a change in our approach regarding to programming and software development training. We want to superimpose system design and its interpretation to the implementation and we want to subordinate individual work to team-work in order to provide our students with more competitive knowledge, experience and skills. Besides an important objective for us from the very first moment is the development of object-oriented perspective. In our opinion we should not start the programming teaching in the traditional style, because the methodology used is considered nowadays outdated, so it should not be followed if we continue the training of software development on object-oriented foundations later.

In our conception students get acquainted with the theoretical and practical aspects as well as the description methods of algorithms in the frames of the course-unit *Problem Classes and Algorithms*. In line with it we introduce our new ideas in a subject called *Software Engineering*. Its contact hours will be two laboratories per week. The syllabus of this unit does not contain any coding and implementation parts yet. The students build Domain Models of well known subjects (e.g. the structure and the functioning of a school, a library, a kitchen, a confectionery, a newspaper stall, etc.). As a first step they prepare domain diagrams. The point of these diagrams is that in practice they are prepared by specialists, who are expert in that field of interest, helped by professionals in information engineering. Using these diagrams they can describe both the properties and the functioning of the objects belonging to the system that is unknown for the software developers. It is called business process modeling. These description tools and methods perfectly fit the needs of the inexperienced information engineering students in order to develop their OO view. Next begins the OO based software design, where further class and object diagrams should be prepared and grouped based on the previously elaborated domain diagrams.

The implementation in C++ starts in the second semester in course of the subject *Programming I.* At this time the students either design all problems in advance in an OO way or they receive the designs from the instructor. The applied UML diagrams can be rather simple ones because the students should be able to implement them at the given competence level. The students also learn data access and database management in line with *Programming I.* The relational data model also can be elaborated using OO design methods. For example the heading of an invoice and the items can be viewed as objects and their connections can appear in a relational database, too. This approach fortifies further their OO outlook and its applicability.

The course unit *Programming II.* is placed in the third semester. It introduces the file management and file input output operations, the memory management as well as the lists as main topics. The students develop applications built up from several source code files. Based on an OO system design it is possible to divide the task into several parts that can be worked out by different students. We could fix a term to exercise a little pressure on them similar to the life in the competitive sphere and to lead them into temptation of omitting the design and documentation phases that seem to the beginners to be useless. In case of long term projects the assignments can be hardened as follows. From time to time it could be worthy to reorganize the teams in order to simulate the real life workplace situations, where due to the standing fluctuation of labour it very often occurs that a project is not finished by the same people it was started by. It also might be feasible to change the projects between the teams. It could be the best evaluation form of the design and documentation of a software project. The team members would face immediately the additional work necessary due to the lack of proper documentation and design. Moreover they would recognize that the time spent on the systematical approach from the very beginning of the project returns quickly.

The students get acquainted with the OOP in the frames of the course unit *Programming Paradigms and Techniques*. The C++ language is used for this purpose, and the team-work based practical training is also applicable. The mark of the students will be determined by the evaluation of the assignment made by the team. The tasks should be distributed by the team members among themselves in a documented way, and they receive the system design from the teacher. We suggest the registration for the *Comprehensive Examination* in programming only after completing this subject successfully. During the exam each student works on their own. The assignment is the implementation of a pre-designed application in a period of 90 minutes. The course unit *Visual Programming* follows the Comprehensive examination. In its frames the students learn the visual application development through mastering the language C# using Visual Studio 2005 Professional. This subject touches how the integrated development environment supports the development of large scale software projects as well as the documentation. We plan the application of the Visual Studio Team System Server for the support of the team-work training.

Conclusions

In our conception in the education of the information engineering students one should break away from the traditional teaching and evaluation forms. The world of software development is in many respects beyond these techniques. As a consequence one should search for and adapt approaches that ensure a better correspondence of our students to the requirements of the competitive sphere. We prepare not only for a change in our view but we also must introduce new didactical methods in order to simulate the real processes in the software industry. The latter aims to support the selection and teaching of the right software development methodology. The new approach is in conflict in some points with the generally accepted student evaluation system. There are some open questions, so we still search for the optimal answers, for example how to asses the individual performance in case of a team-work. We have to evaluate each student with marks in a most possible fair way. We have developed a team-work evaluation system that will be introduced in the next academic year.

References

1. Tisztaszoftver Program, http://www.tisztaszoftver.hu
2. Ian Sommerville, Szoftverrendszerek fejlesztése, Budapest: PANEM, 2002, s.85. ISBN 963 545 311 6
3. Szabolcsi Judit: A standard C++ és az MFC-osztály alapú vizuális programozás oktatásának tapasztalatai Visual C++ 6.0-s fejlesztőkörnyezetben, Főiskolák Matematika, Fizika és Számítástechnika Oktatóinak XVIII. Országos Konferenciája, Nyíregyháza, 2004. augusztus 25-27.

Reviewed by: Steven Teleki, Director, Software Development, Webify Solutions, Inc., Austin, USA

Authors:
György Ferenc Tóth, Senior Lecturer,
Department of Information Technology, Kecskemét College,
H-6044 Kecskemét-Hetényegyháza, Belsőnyír 326/3., mob: +36 20 9123-914,
e-mail: tgyferi@hetenynet.hu

Zsolt Csaba Johanyák, Senior Lecturer,
Department of Information Technology, Kecskemét College,
H-6000 Kecskemét, Hungary, Izsáki út 10., tel. +36 76 516 413,
e-mail johanyak.csaba@gamf.kefo.hu
web: http://www.johanyak.hu