Proceedings of the 7th International Conference on Applied Informatics
Eger, Hungary, January 28-31, 2007. Vol 2. pp. 63-70.

Reform of the software engineering teaching - demands and conception

György Ferenc Tóth[a], Zsolt Csaba Johanyák[b]

[a]Kecskemét College, GAMF Faculty, Institute of Information Technologies
e-mail:tgyferi@hetenynet.hu

[b]Kecskemét College, GAMF Faculty, Institute of Information Technologies
e-mail:johanyak.csaba@gamf.kefo.hu

Abstract

It is a common opinion that software is expensive and inflexible. Its update to the rapidly changing demands leads to inestimable expenses. Moreover the project itself or the improvements done lose validity by the time they are finished. It happens because the development companies try to save time and money by neglecting design and documentation. As a result they cannot use the most up-to-date, development environment integrated Adaptive Lifecycle Management (ALM) tools since the usage of these tools implies a thorough design.

The current software crisis can be traced back partly to the approach of our own-educated professionals. This paper presents our software development teaching related experiences and a newly worked-out strategy that fits more the real-life needs and practice of the software business.

Keywords: Software Engineering, teaching

MSC: 68N99

1. Introduction

A significant amount of software projects fail. They collapse very often like a house of cards after receiving feed-backs from the testers. It is a widely spread opinion that the failure comes from the deficiency or the total lack of software design. Applying only the "code and fix" approach, the changes made on the half-ready or ready software, shake the project to its grounds. Ad-hoc modifications without careful examination and a proper change management represent a huge amount of work and delay the project, which in many cases already failed its deadline. These problems also can cause the abortion of the product launch, because a competitor introduces earlier a similar product. Thus one of the main reasons of the lack of success is that the applications are not methodically developed.

According to the feedbacks in the economy the freshly graduated programmers do not dispose of the most up-to-date knowledge. Contrary, their solutions contain logical flaws and inconsistencies observable even for non-professionals. Moreover the methods and techniques of development they apply are decades behind the "state of the art" in software engineering. For example they apply often heterogeneous platforms (e.g. Linux and Windows operating systems, MySQL, Oracle, MS SQL database servers, PHP, Delphi, Java developing tools altogether) without in-deep knowledge, instead of using strictly one well-known platform or applying a platform-free solution.

Several times the applications are coded using tools offering only word processing capabilities without automatic generation of the source code or the data base. In lack of a proper version control system the modifications of the data structures and program are not followed by the actualization of the plan and documentation.

Another characteristically deficiency is the lack of financial and business planning. Several programmer groups work saying "it lasts until it lasts". The software produced on this way - as long as it is marketable - it has an extremely short life cycle and it is considered out of date even in the moment of its appearance on the market.

In our opinion 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 it is 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 the grounding 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. Recognizing the actual trends and requirements of the software industry we suggest a new approach in the software engineering education that takes into consideration the whole software life cycle and gives more attention to the design.

The rest of this paper is organized as follows: section 2 gives a short overview of the teaching 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 courses. The experiences and results are emphasized, as well. Section 3 outlines the 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 in information engineering at Kecskemét College attain the basic programming skills in course of three semesters. In frames of the course *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 in the second semester. This course is called *Programming I.* and here 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 *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 course *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 try 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 course *Programming Paradigms and Techniques* is followed by the compulsory *Visual Programming* class, where the students get acquainted 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 *Visual Programming* 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 of the course contents 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. Also the C++ programming language is proposed as a first language in [8] because of their multiparadigm nature. Furthermore, the knowledge of the C language is also a precondition of mastering other subjects like *Operating Systems* for example.

Next 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 [5] (previously called Campus) financed by the Hungarian Government. The *Java Based Development* certainly is part of our training palette. This introductory course is compulsory for the students who select the specialisation in *Network and Web Technologies* and it is freely selectable for the other students in information engineering. It has 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. In our experiences the results of the assessments at the end 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. Unfortunately less than 30% of the students pass the exam at the first attempt. Another problem is that according to the feedback from labour market the trained professionals often do not 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.

In our observation the experience of meeting the program development for the first time is determinant for the students. 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, whether 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.

Surprisingly we have recognized on several occasions that students memorize two or three programs before the examination. In course of the examination if there is a minimal coincidence between the actual assignment and one of these programs, they reproduce the memorized one without any minimal effort for the adaptation. Moreover, from time to time the supervisors observe that there are students who use photocopies of programs as cheat sheets in miniaturized form and folded into concertina.

The implementation phase in software development is not other than the conversion of the system specification into an application that can be run [1]. Thus application development cannot exist without system specification. We also made specifications in the traditional programming teaching. We called it problem definition or problem specification and we used 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 of 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.

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. [2]) we have drawn the conclusion that we should rebuild the teaching of software engineering at our college. The emphasized topic is 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. 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 (OO) 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 *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 content of this course 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. Next begins the OO based software design, where further class and object diagrams are prepared and grouped based on the 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 parallel 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 *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 set a time limit 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 systematically approach from the very beginning of the project returns quickly.

The students get acquainted with the OO Programming (OOP) in the frames of the course *Programming Paradigms and Techniques.* The C++ language is used for this purpose, and the team-work based practical training is also applicable. The grading 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 her/his own. The assignment consists of implementing a pre-designed application in a period of 90 minutes. The course Visual Programming follows the *Comprehensive Examination.* In its frames the students learn the visual application development through mastering the language C# and by using Visual Studio 2005 Professional. This subject touches how an Integrated Development Environment (IDE) 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.

4. Conclusions

The analysis of the reasons of current software crisis, the feedbacks received from the software business and our own experiences led us to the recognition that our teaching approach no more satisfies the demands of the software industry. After a deep examination of our curriculum and the typical deficiencies of the fresh graduated professionals we have found that main problematic points are the lack of proper design and documentation as well as the failing team work skills.

For this reason we decided to develop new concepts regarding to the software engineering curriculum based on the results of the investigations. Thus now we emphasize more than previously mastering of modeling, design and team-work skills in our objectives. Besides, we seek new student evaluation models and methods that better fit the project- and team-centered teaching approach, ensuring the most fairly individual grading. Our new experimental team based student work evaluation system, which takes into consideration peer ratings as well, will be introduced in the next academic year.

References

[1] Sommerville, I., Software Engineering, 7th Edition, Pearson Education, (2004)

[2] Szabolcsi, J., 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, 25-27. Aug (2004)

[3] Teleki, S., A practical approach to predictable software development performance in small to medium size software development organizations, *Engineering Management Conference*, 2004 IEEE/UT, 12-13 Aug. (2004), 70–72.

[4] Stevens, K. T., Experiences teaching software engineering for the first time, *Annual Joint Conference Integrating Technology into Computer Science Education*, Canterbury, United Kingdom, (2001), 77–80.

[5] Tisztaszoftver Program, http://www.tisztaszoftver.hu

[6] Jones, C., Patterns of Software Systems Failure and Success, International Thompson Computer Press, Boston, Mass., (1996)

[7] Sertic, H., Filjar, R., Pozgaj, Z., Efficient software development organisation based on unified process, *Electronics in Marine 46th International Symposium*, 16-18 June, Zadar, Croatia, (2004), 390–395.

[8] Porkoláb, Z., Zsók, V., Teaching Multiparadigm Programming Based on Object-Oriented Programming, *10th Workshop on Pedagogies and Tools for the Teaching and Learning of Object-Oriented Concepts*, TLOOC Workshop, ECOOP 2006, Nantes, 2006, avalable at: http://www.cs.umu.se/ jubo/Meetings/ECOOP06/Submissions/7-Porkolab.pdf

György Ferenc Tóth
Izsáki út 10, H-6000 Kecskemét, Hungary

Zsolt Csaba Johanyák
Izsáki út 10, H-6000 Kecskemét, Hungary