A Computer-Based Approach for Software Engineering Teaching

Violeta Bozhikova and Nadezhda Ruskova

Abstract: *In this paper, we describe our experience with teaching software process development using computer-based approach. Problems in teaching Software Engineering discipline are in keeping students motivated and focused on general ideas instead of details. Our goals during the last years were to help students develop good software development habits, and to encourage them to see software engineering as a systematic and computer-aided discipline. A package for laboratory exercises and self study has been developed for this purpose and its model is described in the paper.*

Key words: *Computer- Based Teaching, educational software, Software Engineering course*

INTRODUCTION

In this paper, we describe our experience with teaching software process development using computer-based approach. Problems in teaching Software Engineering discipline consist of keeping students motivated and keeping them focused on general ideas instead of details.

Most of the typical undergraduate software engineering (SE) books [1-5] accentuate on the theory of software development (process, requirements, design, testing, and project management) but do not discuss enough the real software development practice (few pages in the books actually discuss use cases) or discuss practices that can be applied only to large projects. Methods for teams of less than ten developers are not enough discussed in the literature; yet these methods could help new engineers succeed with their first projects.

As a result of the dominating survey-style approach of the above textbooks, the students lose motivation and find them irksome. We can claim that students learn best when they learn something they can understand and really apply. Based on our more than twenty years practice in Computer Sciences and Engineering Department of the Technical University in Varna we are persuaded that one of the main goals during each software engineering teaching course consists of practice students how to use a couple of techniques and tools for real software projects development.

TEACHING SOFTWARE ENGINEERING – COMPUTER-BASED APPROCH

The teaching package

Our goals during the last two years of software engineering teaching were to help students develop good software development habits, and to encourage them to see software development as a systematic and computer-aided discipline. Developing and using a package of different tools during the labs we tried to ensure the students that the tools are very important, useful and effective in the real software development practice. We hope we have found an effective teaching approach that we would like to present in this paper.

Our software development teaching model (figure 1) during the last couple of years accentuated on the applied side of software engineering teaching, using a package of tools, instead of trying to instruct students in theory. An opened for later extension package for laboratory exercises and self study has been developed for this purpose. To the moment it consists of the following four tools:

- A tool for Software Project Management (PMS), presented in [6].

Project management software (PMS) is a term covering many types of software, including scheduling, cost control and budget management, resource allocation, collaboration software, communication, quality management and documentation or administration systems, which are used to deal with the complexity of large projects. PMS that we have developed can be determined as native web-based and multi-project

enabled PMS. Each user can work on more than one project and on more than one task. The projects have specific starting and completion dates that can be controlled by the project leader and the user himself.

- A tool (VLT) with lecture notes, exercises and tests in Microsoft.Net programming.

VLT is a case of web-based laboratory educational software that aims at helping the student study the relevant program language. Before doing the current laboratory exercises, the students could read the language materials. Choosing "Tests" pane the students could make the corresponding quiz and evaluate their learning progress (see figure 2). The teacher also can analyse the students' results, and get an overall view for the progress of the students. The architecture of the developed web-based tool provides instructors a possibility to easily create and manage different course materials. Nevertheless the tool doesn't impose limitation about the program language materials the students are suggested to use a .Net language.

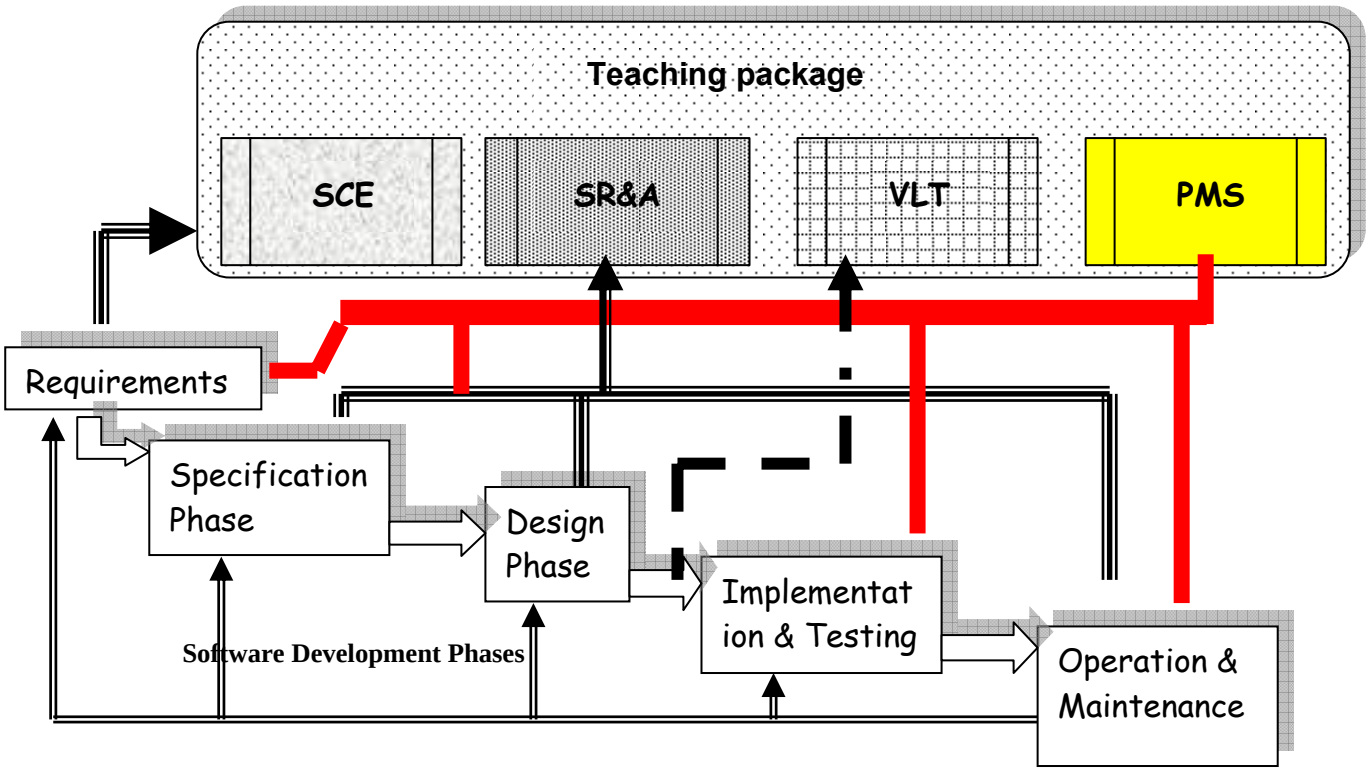

Fig.1: Computer-based approach for software engineering teaching

- A tool for Software Cost Estimation (SCE).

The tool (see figure 3) is based on the widely used Basic COCOMO and COCOMO II models [1-5] for software cost estimation. Software estimation is the part of project planning devoted to estimating the size, effort, time, people required etc. Software estimation process gives the information needed to develop a software project's schedule, budget and assignment of personnel and resources. Estimating product size is the basis of software estimation. The following three measures are generally used for software size estimation in the developed tool: Source Lines of Code (SLOC), Function Points and Object Points. Using size as input, estimates of effort applied, development time and people required can be made. Below the basic COCOMO equations are presented:

Effort Applied = $a_b(KLOC)^{b}_{b}$ [man-months], where KLOC=SLOC / 1000.

Development Time = $c_b(\text{Effort Applied})^{d}_{b}$ [months]

People required = Effort Applied / Development Time [count]

The coefficients a_b, b_b, c_b and d_b depend of the mode of the project (organic, semi-

detached or embedded) and are 2.4÷3.6 for a_b, 1.05÷1.20 for b_b, 2.5 for c_b and 0.38÷0.32 for d_b. According to Boehm in organic mode, relatively small software teams develop software in a highly familiar, in-house environment. The semidetached mode of software development represents an intermediate stage between the organic and embedded modes. In embedded-mode the software teams develop software respecting a complex of strongly constraints, such as hardware, software platform, etc.

- A tool for Software Re-structuring and Analysis (SR&A) is integrated in the package [7].

Using this tool the students can specify, document, analyze and re-structure the software structure. During the specification stage the students specify the software structure as a weighted oriented graph G=(X, U), where X (N=|X|) is the set of numbered nodes that models the set of components (classes, modules, files, packages, etc.) of the software system and U is the set of edges that represent the dependencies (e.g., procedural invocation, variable access, etc.) between the components of the software system. The weight w_i of each node $x_i \in X$, i=1…N could be interpreted as the number of the component's elements (for example as the number component's functions) or as the importance (rank) of the corresponding component. Next, the students choose the appropriate re-structuring analysis algorithm, based on Cluster Analysis method, Formal Concept Analysis (FCA) method or Program Slicing techniques. A set of heuristic algorithms corresponding to the cited analysis methods are available: a descent hill-climbing algorithm, a Tabu-Search algorithm, a FCA – based algorithm and a static slicing algorithm extracting components of the program (slices) that effect on the values of some variables (the slicing criterion). The last algorithm is based on well known Depth-first search (DFS) algorithm for traversing the graph G. The first three algorithms treat re-structuring as an optimization problem: each algorithm tries to find a "good partition" of the graph G, minimizing a goal function "k" (1) and satisfying the constraint W_0 (2) related with the weight W_d (W_d is the sum of the weights of all nodes in cluster X_d) of each cluster X_d, (d=1…M, $\cup X_d$=X) produced. Changing W_0, the students can change the number of the clusters and can guide the algorithm to converge to a different solution.

The tool provides visualization of the initial software structure and the re-structured solution (the final solution). In order to find a better solution this feature helps the students to make the appropriate changes to the initial software structure.

$$k = \frac{1}{2} \sum_{i=1..M} \sum_{j=1..M} k_{ij}, \forall\, i \neq j \qquad (1)$$

$$Wd = \sum_{x_i \in X_d} w_i <= W_o \qquad (2)$$

The goal function "k" (1) is sum of the number of edges between the clusters in a partition found. It is a measure of the quality of the partition. The "good partition" minimizes its value. "M" is the number of clusters in the given partition; k_{ij} is the number of edges between each couple of clusters.

The teaching experience

During the first five weeks of each usually fifteen weeks term the students created teams and started on learning the needed environment (Visual Studio.Net) and the programming .Net language using the integrated in the package tool (VLT) with lecture notes, exercises and tests (see figure 2).

During the next two weeks the students started on developing their term project in teams. They specified, approved and estimated the software project with the aid of

SR&A and SCE tools. The team leader shared the different tasks between the team members.

During the next six-seven weeks the team members worked on coding their term project. They demonstrated the realized .Net solution in the last week of the semester and presented the created specifications and user documentation. The team leader was responsible for the overall team work. Using PMS the team leader effectively managed the project works.

CONCLUSIONS AND FUTURE WORK

Our experience with teaching software process development using computer-based approach is commented in this paper. Problems in teaching Software Engineering are maintaining enough self-discipline and finding proper tool support.

The paper presents an effective computer-based approach for Software Engineering teaching. An extensible package of different tools has been developed for this teaching course, its model is described in figure 1 and our experience with its usage is briefly commented. Our observation is that the effect on the rate of student learning is positive. The number of students choosing term projects and graduation thesis, based on .Net Framework has increased nearly twice. It is evident that as a result of this approach the students will be able to work as part of a software development team when they graduate. In the last decade a lot of foreign software companies have entered our country that needs skilled professionals. The number of graduated students working in leading industrial companies and developing high quality code in Microsoft.NET programming environment increases significantly. Using the computer-based approach, the students obtain very quickly the following software engineering skills:

- Use Software cost estimation tools during the early software development phases (like SCE).
- Create and adjust the specifications using the software re-structuring tool (SR&A). The student learn that when the specifications change (as a result of the change of the requirements), the specifications must respond to the changes.
- The students learn just enough the graph theory to make describe the system's structure and to use efficiently the software re-structuring tool.
- The students must also provide user documentation. The created specifications can be used as documentation (for future students to understand the project).
- Create teams (not more than fifteen students) and work as a member of a team.
- Each team is assigned a project leader by the team itself. The team leader manages the overall software development work using PMS. The teams are free to use their own work style.
- The teams might develop software iteratively and incrementally using a new software language and environment (VLT). At the end of the iteration the teams might demonstrate their work.

Further work is needed in order to extend our teaching package. The package can be applied in other Software development courses as well. Our idea is to develop and integrate in the package a tool for software testing. Software Testing is a costly stage (around 50% of the labor expended to produce a working program) of the software life cycle that is usually performed by an independent group of testers after the functionality is developed (before the software is shipped to the customer). Test techniques (unit level test, integrated level tests and system level tests) are not limited to the process of

executing a program unit or program system with the intent of finding software bugs. Software testing techniques (black-box or | and white-box) for validating and verifying that the software unit or system meets all planned functional and user requirements could be implemented by the suggested testing tool.

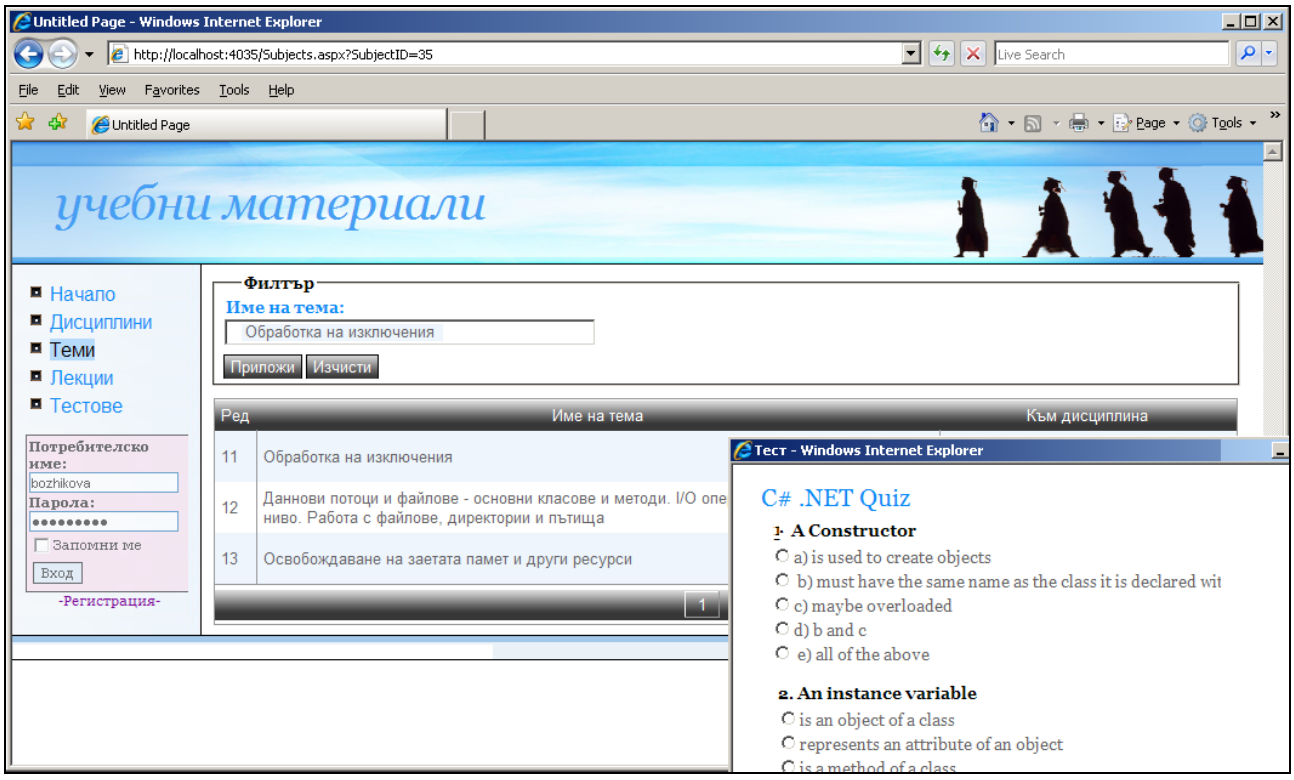

Figure 2: VLT – a tool with lecture notes, exercises and tests in Visual.Net programming

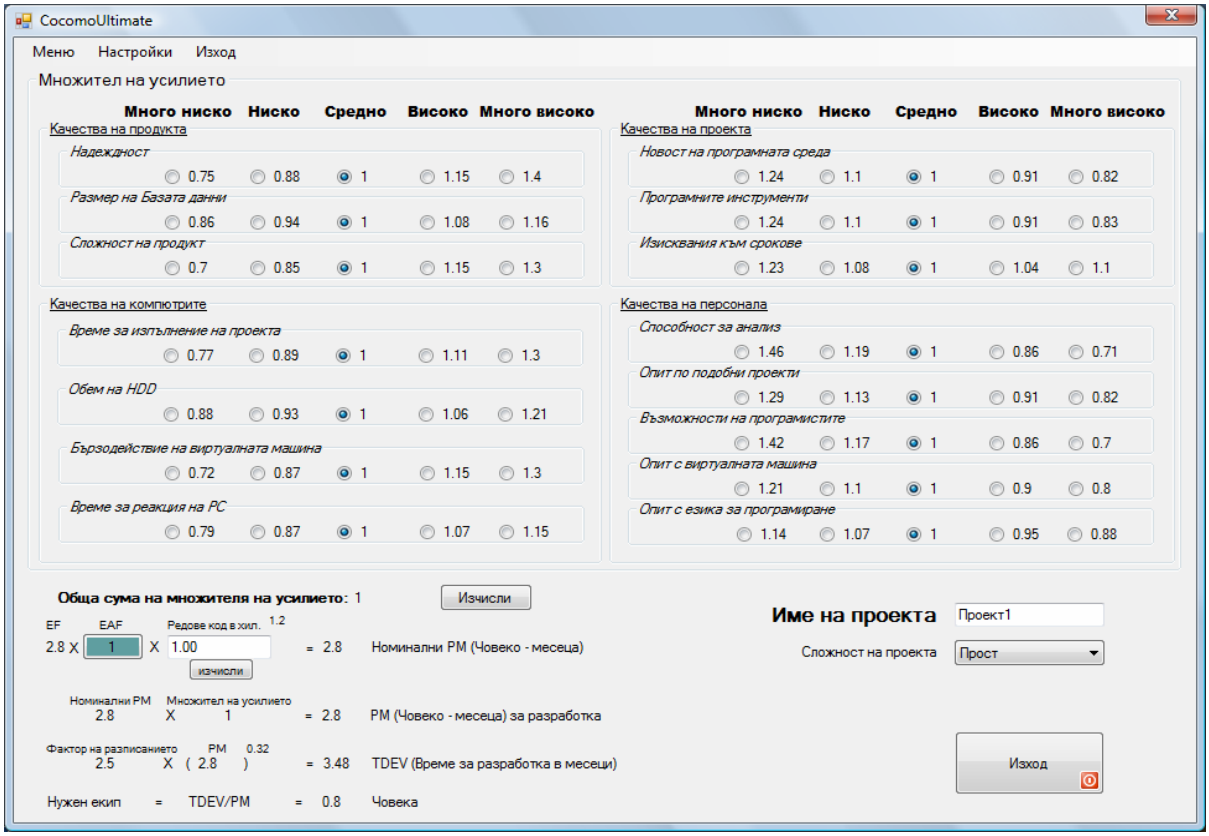

Figure 3: SCE - tool for Software Cost Estimation (the main form)

REFERENCES

[1] Eric J. Braude, Software Engineering: An Object-Oriented Perspective. John Wiley 2001.

[2] Leszek A. Maciaszek and Bruc Lee Liong, Practical Software Engineering: A Case Study Approach, Addison-Wesley 2005.

[3] Bernd Bruegge and Allen H. Dutoit, Object-Oriented Software Engineering. Prentice Hall, 2000.

[4] Shari Lawrence Pfleeger and Joanne M. Atlee, Software Engineering: Theory and Practice, Third Edition, Prentice Hall 2006.

[5] Roger S. Pressman, Software Engineering: A Practitioner's Approach, Sixth Edition. McGraw-Hill, 2005.

[6] Violeta Bozhikova, Mariana Stoeva, Krasimir Tsonev, A practical approach for software project management, CompSysTech Conference 2009, Russe (to appear)

[7] V.Bozhikova, M. Stoeva, A. Antonov, V. Nikolov, Software Re-structuring (An architecture-Based Tool), ICSOFT 2008, Third Int'l Conference on Software and data Technologies, pp.269-273, Porto, Portugal, 2008.

ABOUT THE AUTHORS

Assist. Prof. Violeta Bozhikova, PhD, Department of Computer Sciences and Engineering, Technical University of Varna, Phone: (+359) 898 279 887, E-mail: vbojikova2000@yahoo.com.

Assoc. Prof. Nadezhda Ruskova PhD, Department of Computer Sciences and Engineering, Technical University of Varna, Phone: (+359)886 770 125, E-mail: nada.ruskova@abv.bg.