Practice in Software Engineering course: "what and how to study"

Yuriy Solyanik, Maryna Vladymyrova,
Iryna Zarets'ka, and Grygoriy Zholtkevych,

V.N. Karazin Kharkiv National University,
4, Svobody Sqr., Kharkiv, 61077, Ukraine
u.solyanik@gmail.com, vladymyrova@gmail.com,
zar@univer.kharkov.ua, zholtkevych@univer.kharkov.ua

Abstract. The goal of this paper is to share the experience of V. N. Karazin Kharkiv National University in Software Engineering training, especially in organizing students' practical work so that they gain competences required by modern software industry based on world standards.

As software development became now comprehensive industry there is strong demand for highly qualified specialists all over the world. As any industry it is based on standards for products as well as for processes. So any university graduate planning to work in this industry should know these standards and be able to work with them including tailoring to the concrete situation. It is especially true for the graduates majoring in Computer Science (CS). Usually these knowledge and skills are taught in the course of Software Engineering (SE) which is adjourned to the senior years when all the fundamental concepts of CS and technological skills have been already gained. Unfortunately all manuals on SE studies including Computing Curricula present only core topics to discuss and learn but no hints on how to organize practice with visible results of students' growing as specialists just ready for industrial work. It is up to the University to decide the ways to develop such competences in graduates. Our university developed its own approach to provide necessary competencies so the goal of this paper is to share our views and experience with other universities as well as to have feedback as to advantages and disadvantages of our approach.

There are several definitions of SE. Let us take this one: "Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into use" [1]. Another one tells that SE encompasses knowledge, methods and tools for defining software requirements and performing software design, software construction, software testing and maintenance tasks. Anyway as any engineering discipline SE is regulated by number of normative documents and standards. Among world leading standardization organizations in this field are ISO, IEC, IEEE, ESA. Their standards concerning SE cover all the parts of this discipline. It is just natural to build the subject of SE on the foundation of the normative documents and standards. Studying the majority of standards

in the course of SE we base our practice mostly on the ESA standards [2] as they usually include all the information from similar standards of other organizations but are more verified as they deal with critical software. According to these standards we divide Software Life Cycle (SLC) onto six phases which are User Requirements phase (UR), Software Requirements phase (SR), Architectural Design phase (AD), Detailed Design phase (DD), Transfer phase (TR) and Operation and Maintenance phase (OM). It is essential to teach SE in such a way that students could go through each phase of SLC performing its tasks and preparing corresponding artifacts and documents strictly in compliance with the standards. Knowing as many standards as time allows and being able to apply them practically will make it possible for students to perform agile tailoring of them depending on company policy and concrete type of project when working in industry.

The whole process of studying looks like this. According to SWEBOK [3] SE discipline consists of two big areas which are Software product engineering and Software management. In fact we have these parts in two different courses but they go in parallel with common ideas and projects to work on. Both courses are taught to the graduate students (fifth year) so the main goal of both of them is to systemize and generalize all the knowledge and skills gained by students before via series of conceptual lectures made by students themselves in form of presentation and practical work of real industrial level. So we organize semester long business game on SE (both parts) with students working in teams on real projects and playing different roles during the process of development. In fact they all go through being business analysts, system analysts, system architects, quality assurance personnel, team leaders, project managers and technical writers. Moreover they usually work on projects that are needed by University subdivisions or have real customers and timescales with several students of younger ages (usually fourth and third years of study) to their subordination making exploratory and experimental prototyping, coding, unit and integration testing, etc., which can be estimated for them as course work or even bachelor project. This heightens the responsibility of graduate students not only for the projects to be done in time but for undergraduates to have good marks under their leadership not to mention their own marks on the SE subjects. All graduate students are divided into teams consisting usually of 3 to 4 graduate students (depending on the project scope) plus 2 to 3 undergraduates. Each team works on a separate project but reviews the project of its peer team. The process of peer reviewing is not less important than working on their own project as it allows students to see mistakes and blunders as well as successful features more clearly. All the steps, activities and solutions are thoroughly documented and reviewed which at the end gives the full picture of students' progress and results.

The topics of conceptual lectures in SE are presented in the table. In fact all the key areas and units are well presented in SWEBOK [3], PMBOK [4] and Computing Curricula [5, 6] so we only packed them into topics and added some modern technological aspects [7]. In fact we use a lot of standards and literature in this course, it will take several pages to present all of them, so we put only

those we directly refer to in this paper into the list of literature below.

Software Product Engineering		**Software Management**	
Topic	**Hours**	**Topic**	**Hours**
Taxonomy of standards in SE	1	State of the art in SE and Software Management. Software Project Management, Software Quality Management, Software Configuration Management. Four Ps: Project, Product, Process, Personnel	2
SLC concepts and models (including classical and modern ones like Agile, SCRUM, KANBAN, etc.)	2	Project goals. Project processes scheme. Six management processes. Business-planning of the project	1
User requirements elicitation methods, tools and artifacts	2	Organizational management.	2
Software requirements specification methods, tools and artifacts	2	Personnel and project environment management. General information about supporting processes. Communication management	2
System models, types and classification	2	Planning. Standards of planning. Different representations of the plan (Gant diagrams, network planning, etc.). Creating the plan. Diagrams analysis. Critical path, critical chain. Survey of tools SPMP	4
UML 2.x, history, development, usage	3	Infrastructure of planning: database of processes, base line of process stability, processes' assets. Methods of possibilities evaluation (PSP, TSP, CMM)	1
Formal methods of requirements specification	2	Software Management. Configuration. Standards, basic concepts, methods, tools SCMP	4
Quality of software, metrics of quality	2	Software size metrics. Methods of data collection. Software size evaluation	4

Software Product Engineering		**Software Management**	
Topic	**Hours**	**Topic**	**Hours**
Software architecture, architectural styles and patterns, POSA, MDA, SOA	4	Software cost evaluation. Standards, models (COCOMO,COCOMO2, SLIM, etc), tools	4
Architectural design processes, methods, tools and artifacts	2	Risks. Uncertainties. Standards. Risk management and control. Quantitative and qualitative risk analysis. Models, methods, tools. Risks planning	4
Verification and validation of software, methods, tools and artifacts	4	Quality Management. General principles. Product quality. Quality Assurance (QA) organization: organizational structure of the project. SQMP	4
Detailed design processes, methods, tools and artifacts	2	Software reliability evaluation	3
Transfer, operation and maintenance processes, methods, tools and artifacts	2	Project closing, analysis and summaries	1

Now about the practical work. We prepare a number of real projects with real customers and timescales (usually semester long to develop and further to continue coding, testing and maintaining by undergraduates from the corresponding team). After teams were formed and projects selected by teams they prepare vision documents in standard form while several first lectures acquaint them with the main concepts and activities to follow. Then they proceed with the projects going iteratively through each phase of SLC.

The first phase - UR one - takes usually 3 to 4 weeks which is longer than needed because at this time students are not yet well immersed into subject. They work directly with customers and domain experts, prepare questionnaires and make surveys to define project scope and boundaries, assess operational environment, determine users' roles, and elicit user requirements at most thoroughness and completeness. Sometimes exploratory prototyping is needed (usually made by undergraduate members of a team). Then it takes time to learn standards and prepare the draft version of the first full document in standard form which is URD (User Requirements Document) with all requirements identified and attributed and status sheet attached. This version is reviewed by peer team and all the discrepancies found are documented and discussed during the review meeting. Then several iterations with corrected URDs and updated status sheets follow until the final version is approved and signed by both parts. At the same time the acceptance test plan is being worked out and documented.

As to the management activities students define their projects goals from the management point of view and elicit main processes, find the appropriate structure of their team organization, distribute roles and write down the responsibilities of each role. They also get acquainted with the MS Project tool.

The second phase overlaps first one and usually begins when first URD draft is submitted for the review. Second phase usually takes the same time as first or a little longer as it supposes analysis of the requirements and logical model construction as well as system requirements specification. At this time students have already studied different system models and software quality metrics, which facilitates the process of a logical model construction and working with both functional and non-functional requirements in quantitative form to be properly verified. They use various CASE tools for model construction and specification. At the end of this phase the Software Requirements Document (SRD) is prepared with obligatory traceability matrix attached. Again iterative process of documented peer reviewing and making changes takes place, which ends with the approval and signing of SRD.

As to the management activities this phase is dedicated to planning. Students plan their work on the project using MS Project tool, identify tasks, define their sequence and duration of each of them, and determine types of relations between tasks. Then they assign real terms and resources for each task, e. g. form the project's schedule and construct the critical path using MS Project tool and developing their own program to build a critical path just to compare the results. It allows them to evaluate the real duration of the project. Students also consider different solutions to the problem of resource overloading. At this phase the SPMP in standard form is being prepared and submitted for the review.

The third phase is the most hard for students because it requires all their previous knowledge and skills, sometimes they are compelled to learn new technologies and make exploratory prototyping to solve architectural problems usually rising from non-functional requirements and to apply appropriate architectural styles and patterns. They have to construct a physical model with detailed description of each system component, process, software module and physical node of software deployment which is not only difficult to perform but time consuming as well even using advanced CASE tools (not to mention regular returns to the previous documents). At the end of this phase the standard ADD (Architectural Design Document) is submitted for the review which also takes several iterations before final approval.

As to the management activities this phase is dedicated to the Configuration Management. Within the frame of their project students create repository to store all the artifacts of the project, make checkouts, updates and commitments, add files and folders, assign tags and create alternative branches of development process using TortoiseCVS and TortoiseSVN tools. They also prepare Software Configuration Management Plan (SCMP) in standard form and submit it for the review. Another students' activity at this phase concerns with software cost estimation. They use Costar and USC-COCOMOII 1999.0 with CostXpert tools to

estimate the cost of software for the early design model and for post-architecture. For their projects students assign the values of Scale Drivers, Cost Drivers, Function Points or SLOC, Resource Costs and make necessary calculations and reports. The SPMP draft made at the first phase is now being improved with reference to the risk plan developed further.

The fourth phase heavily overlaps the third one as it is usually done by undergraduate members of the team and starts just as the critical architectural solutions have been made. It consists of coding, unit and integration testing with all documentation required by standards. At this time graduates begin writing PHD (Project History Document) as time goes to the end of semester. The main output document of this phase - DDD (Detailed Design Document) - takes too much time to prepare so graduates usually only supervise its preparation and as our experience shows never have time to verify it thoroughly. This is mainly the responsibility of undergraduates to finish coding and testing as well as transfer releases to customers and maintain them during the operation before provisional acceptance and after it. At this moment we try to have new younger students to learn this software to be able to maintain it.

As to the management activities this phase is dedicated to risk control and management. Students make the risk plan for their projects which includes all steps of working with risks e. g. risk identification, risk estimation, methods of responding to risk events and control of these responses. They use PertMaster Project Risk v7.6.0006 tool for this work. At this phase they also prepare Software Quality Management Plan (SQMP), specify and improve SCMP made at the previous phase and consider problems of management according to the plans. At the end of this phase the process of project closing is being considered, analyzed and the whole project is summarized.

Certainly the time limits and kinds of projects students develop do not allow them to go practically through all the management activities and artifacts considered in theoretical study, but even what is done gives students some management skills to be developed in their further work as no industry newcomer begins his or her career as a software project manager.

As a result our graduate students learn to work with requirements and specify them, make sound and grounded architectural solutions, carry out management activities and present all information in compliance with world standards. All that is being done in a team work with peer verification and audits made by teaching staff. As to the benefits for undergraduates they learn a lot from their senior mates which is good in itself, plus get ready for analytical work next year. Sure a lot of work is done by each member of a team with all those paper artifacts being meticulously prepared according to standard forms. A lot of them might seem excessive for real middle size industrial software company using agile methods and techniques. But what we think is: it is better to master the whole process and use only parts of it than to know only some parts when it is necessary to use the whole process.

References

1. Sommervile, Ian: Software Engineering. Pearson Education, New York and London (2001)
2. ESA Software Engineering Standards. ESA Board for Software Standardization and Control (1991)
3. Software Engineering Base of Knowledge (SWEBOK). Revision of IEEE (2010)
4. A Guide to the Project Management Body of Knowledge, 4th ed. PMBOK Guide (2008)
5. Computing Curricula 2001. Computer Science. Final Report. The Joint Task Force on Computing Curricula. IEEE & ACM (2001)
6. Computing Curricula 2005. The Overview Report. The Joint Task Force on Computing Curricula IEEE & ACM (2005)
7. Futrell, R. T., Shafer, D. F., Shafer, L. I.: Quality software Project Management. Addison - Wesley Publishing Company, New York (2002)