Software requirements engineering training: problematic questions

Andrii M. Striuk[1], Serhiy O. Semerikov[1,2,3,4], Hanna M. Shalatska[1] and Vladyslav P. Holiver[5]

[1]*Kryvyi Rih National University, 11 Vitalii Matusevych Str., Kryvyi Rih, 50027, Ukraine*

[2]*Kryvyi Rih State Pedagogical University, 54 Gagarin Ave., Kryvyi Rih, 50086, Ukraine*

[3]*Institute of Information Technologies and Learning Tools of the NAES of Ukraine, 9 M. Berlynskoho Str., Kyiv, 04060, Ukraine*

[4]*University of Educational Management, 52A Sichovykh Striltsiv Str., Kyiv, 04053, Ukraine*

[5]*Customertimes Ukraine, 15b Leiptsyzka Str., Kyiv, 01015, Ukraine*

Abstract

The key problems of training Requirement Engineering and the following ways to overcome the contradiction between the crucial role of Requirement Engineering in industrial software development and insufficient motivation to master it in the process of Software Engineering specialists professional training were identified based on a systematic research analysis on the formation of the ability of future software engineers to identify, classify and formulate software requirements: use of activity and constructivist approaches, game teaching methods in the process of modeling requirements; active involvement of stakeholders in identifying, formulating and verifying requirements at the beginning of the project and evaluating its results at the end; application of mobile technologies for training of geographically distributed work with requirements; implementation of interdisciplinary cross-cutting Software Engineering projects; involvement of students in real projects; stimulating the creation of interdisciplinary and age-old student project teams.

Keywords

software requirements, software engineering training, software engineer competencies

1. Introduction

The first course in Software Engineering was developed under the guidance of Friedrich Ludwig Bauer [1, 2], it contained only a brief overview of the process of determining the requirements for the software product such as functions, user needs and operating environment requirements.

CS&SE@SW 2021: 4th Workshop for Young Scientists in Computer Science & Software Engineering, December 18, 2021, Kryvyi Rih, Ukraine

andrey.n.stryuk@gmail.com (A. M. Striuk); semerikov@gmail.com (S. O. Semerikov); shalatska@i.ua (H. M. Shalatska); holivervlad@gmail.com (V. P. Holiver)

http://mpz.knu.edu.ua/pro-kafedru/vikladachi/224-andrii-striuk (A. M. Striuk); https://kdpu.edu.ua/semerikov (S. O. Semerikov); https://scholar.google.com.ua/citations?user=vmRCFM8AAAAJ (H. M. Shalatska); https://www.linkedin.com/in/holiver-qa/ (V. P. Holiver)

0000-0001-9240-1976 (A. M. Striuk); 0000-0003-0789-0272 (S. O. Semerikov); 0000-0002-1231-8847 (H. M. Shalatska); 0000-0002-8276-5992 (V. P. Holiver)

Just as 50 years ago, defining software system requirements is the first step in development which largely ensures its success.

Recommendations for the development of curricula for Software Engineering bachelors define the competence to find compromises, the essence of which is to reconcile conflicting project goals, find acceptable trade-offs for cost, time, knowledge, existing systems and organizations: "Students should engage in exercises that expose them to conflicting and changing requirements. ... Curriculum units should address these issues, with the aim of ensuring high-quality functional and nonfunctional requirements and a feasible software design." [3, p. 21].

Requirements engineering is the process of identifying, formalizing and documenting requirements, that occurs during communication with a customer and other stakeholders who are not typically proficient in software engineering techniques. The identification of requirements demand from the Software Engineering specialist to apply the following general professional competencies [4]:

- ability to think abstractly, analyze and synthesize;
- ability to apply knowledge in practical situations;
- ability to communicate orally and in writing;
- ability to search, process and analyze information from various sources;
- ability to work in a team;
- ability to act socially responsible and conscious.

The formation and development of these competencies takes place in teaching of disciplines that do not belong to the professionally oriented, which reduces the attention of students to their study. At the same time, software requirements engineering is perceived as an unimportant course, which is not directly related to the activities that novice students associate with Software Engineering, especially with the creation of software code.

The solution to this problem is possible by developing a practice-oriented methodology for training engineering requirements based on an activity approach aimed at overcoming *the contradiction between the decisive role of requirements engineering in the practice of large software projects industrial development and insufficient motivation to acquire essential knowledge in the process of software engineers professional training.*

2. Results

The guidelines for the development of undergraduate programs in Software Engineering ("Software Engineering 2014" [3]) state that the requirements reflect the real needs of users, customers and other stakeholders associated with the system being developed. Defining requirements includes identifying and analyzing the needs of stakeholders and creating an appropriate description of the desired behavior and qualities of the system, as well as relevant limitations and assumptions.

"Computing Curricula 2020" [5, p. 120] identifies the following competencies for defining software requirements:

1. Identify and document software requirements by applying a known requirements elicitation technique in work sessions with stakeholders, using facilitative skills, as a contributing member of a requirements team.
2. Analyze software requirements for consistency, completeness, and feasibility, and recommend improved requirements documentation, as a contributing member of a requirements team.
3. Specify software requirements using standard specification formats and languages that have been selected for the project and be able to describe the requirements in an understandable way to non-experts such as end-users, other stakeholders, or administrative managers, as a contributing member of a requirements team.
4. Verify and validate the requirements using standard techniques, including inspection, modeling, prototyping, and test case development, as a contributing member of a requirements team.
5. Follow process and product management procedures that have been identified for the project, as a contributing member of the requirements engineering team.

Sedelmaier and Landes [6] indicate that requirements engineering is a major component of Software Engineering, but it is difficult to train. In training, Software Engineering is usually limited to small "toy" projects that only partially reflect real tasks. There are several reasons for this.

1. At the initial stage of training in Software Engineering specialists, there is a *bias towards programming training*, rather than the identification, classification and formulation of software requirements. Typically, programming tasks are small and well-defined: students develop simple software components, often in small groups or individually, with assignments focused on a specific task that demonstrates, for example, the use of loops, arrays, or algorithms related to their processing. That is, software development tasks mainly focus on the technical aspects of programming and specific programming languages in an area that is familiar to students.
2. As a result, the teacher provides clearly defined and understandable requirements in this area without the use of unfamiliar terminology or unusual concepts, which gives students a misconception that they do not need to worry about the requirements because they *incorrectly summarize their initial programming experience in software development projects*: there is no such thing as vague requirements for stakeholders who use unfamiliar terminology because students have never encountered them. This information is often not readily available, and should be obtained from a number of relevant stakeholders, who are often difficult to identify and who do not cooperate. Students underestimate the importance of requirements engineering because its methods do not help to better solve programming problems. In addition, programming tasks are usually separate and unrelated to other tasks. Even if there is more than one possible way to solve the problem, the chosen approach will not have consequences for the following tasks: students do not need to compare the advantages and disadvantages of alternative solutions, as they will not suffer from the consequences of error, which means it doesn't matter if the requirements are wrong or not.

3. Often students cannot even imagine the problems in Software Engineering that arise due to errors in defining requirements, and do not trust teachers-practitioners, believing that they are exaggerating them. Methods for determining requirements seem boring and useless to them, because *students mostly don't know why they need them.*
4. In the later stages of Software Engineering training, even when the programming bias is shifted to Software Engineering, in particular requirements engineering, the situation remains problematic: due to time constraints *the complexity of real problems can hardly be reproduced in university education*, so students do not perceive interdependence between requirements, mistakenly believing that complexity scales linearly, while increasing the number of requirements exponentially increases the interdependence between them.

Given the limited training time for Software Engineering specialists, it is difficult to create conditions for students that reflect the real problems of requirements engineering: due to the lack of real customers, students cannot imagine the complexity and relationship between requirements within a large Software Engineering project.

Ouhbi et al. [7] formulate recommendations for teachers on the formation of the ability to identify, classify and formulate software requirements:

1. Learn to identify the scope of the problem, avoid general and vague specifications. To do this, teachers should take into account the individual characteristics of students, which can be determined, in particular, by questionnaires. This gives teachers the opportunity to form teams that demonstrate the best results. Acuña et al. [8] have shown a significant positive correlation between personality extraversion factor and software product quality, including requirement satisfaction.
2. Teach to choose and use appropriate requirements engineering tools: students must know the capabilities of modern tools and be able to choose the best tool for the project according to its needs – isolation of requirements, analysis of requirements, specification of requirements, verification and confirmation of requirements, requirements management.
3. Facilitate requirements analysis and modeling activities in addition to requirements management and implement the concept of prototyping in learning: through prototyping, a working model of a software product can be created before the final product is implemented, and prototypes presented to stakeholders are very effective for clarifying the requirements.
4. Involve students in real projects to give them the opportunity to acquire sufficient knowledge and skills. It is also advisable to invite practitioners to present real projects and gained experience.
5. Develop the abilities, skills, and strategies needed to align engineering requirements with modern conditions of geographically distributed (global) software development [9].
6. Teach students an approach to solve problems, methods and tools of development. Students do not understand the importance of activities to determine the requirements and their impact on the success or failure of projects in the lecture-laboratory form of requirements engineering training. Teachers are encouraged to use alternative forms of learning, such as games.
7. Use mobile devices as learning tools in a mobile learning environment: students can share a virtual board, e-textbooks and exchange data through a network environment

to actively participate in course discussions. These devices help to intensify student's educational and cognitive activities [10].

Mich [11] points out the contradiction between the importance of requirements isolation activities for industrial projects and the lack of student motivation to learn requirements isolation due to their lack or little experience in the field and the corresponding disregard for business requirements, focusing on modeling requirements with a bias in detail instead of a preliminary global review of the project, analysis of requirements for the implementation of "toy" projects instead of industrial, non-involvement of stakeholders at the stages of problem statement and evaluation of student performance, etc.

To overcome the isolated contradictions, Mich [11] suggests using CASE tools to support modeling using UML and provides recommendations for reducing the risks of requirements modeling in a hurry due to the desire of students to start modeling requirements, even if business analysis and requirements detection is just beginning. The researcher developed a template for a student project aimed at:

- demonstration the role of computer systems in solving business problems or developing business strategies;
- integration of organizational issues into problem analysis to answer questions such as "Who should collaborate to collect the data needed to develop the system?";
- understanding the role of requirements analysis in the system development process, including contractual implications;
- detection and management of conflicting requirements;
- use of UML from the very first step of requirements modeling, also for business processes and subjects;
- documenting requirements as project specifications and their confirmation.

Mich [11] points out that thanks to the developed template, projects become more and more connected with real organizations or companies. Initially, companies participated little in the projects, through the initial representation of the company by its representative and the final presentation of the results by students. Later, more and more interviews were conducted with stakeholders representing real organizations.

Goswami and Walia [12] emphasize that requirements development is one of the earliest and most important phases of a software development lifecycle. This is a critical phase when program requirements are collected from a variety of stakeholders (both technical and non-technical) and described in natural language in an official document known as the software requirements specification. Due to the ambiguity, inaccuracy and uncertainty of the natural language, errors are often made during the development of the specification. Therefore, the focus should be on identifying and correcting inconsistencies in the early stages of the software development lifecycle to avoid unnecessary effort and cost of software processing in the later stages. To do this, researchers suggest using software assessment (inspection, survey) methods when qualified professionals review documentation, code, and other project artifacts to identify and report problems. Such inspection is a systematic method of detailed study of software artifacts. The study confirmed the benefits of verifying artifacts developed at different stages of software development (e.g. requirements, design, code, interfaces). The main stages of inspection are:

a) selection of qualified inspectors; b) individual examination to identify problems; c) team meeting to systematize problems and d) further actions to eliminate them.

Ozkaya et al. [13] offer two half-semester mini-courses for students majoring in architecture and construction.

The purpose of the first mini-course "Software Requirement Modeling" is to review the methods of modeling software requirements and to demonstrate with the help of modeling requirements strategies for solving problems in software development. Requirements modeling helps engineers (not only software engineers) to better understand the task, reveals the essence of the relationships that characterize it. Researchers emphasize that this is a significantly different approach than considering the functionality of the final product. For specific engineering tasks, such as building a geometric representation of information models, designing communication programs for mobile devices or building data models for different stages in design, the engineer must not only identify the requirements for the software being developed, but also understand the requirements of the engineering industry for which it is being developed. In other words, customer expectations and needs must be systematically modeled for better design of software solutions.

The content of the first mini-course is purposefully focused on the development of software for automated architectural and engineering design. It offers a study of several methods for identifying requirements and ways to obtain basic information that will help in software development. The application of the discussed methods of determining the requirements for the software project should take place with the participation of stakeholders: some real stakeholders, usually with customers. Upon completion of the first mini-course, students acquire the ability to develop a prototype of the designed system and specification of software requirements.

The program results of the proposed course are:

- ability to correctly use basic terminology for identifying and specifying software requirements;
- ability to choose different data collection methods to identify software requirements;
- ability to distinguish types of requirements and classify relevant information;
- ability to document requirements in various forms;
- ability to use information about software requirements to improve design;
- ability to evaluate different methods of identifying requirements and develop strategies for selecting the most appropriate [13, p. 5].

The purpose of the second mini-course "Software Requirement Application" is to teach the application of certain requirements to solve practical problems. Researchers point out that it is not enough to just define software requirements, typically in design, requirements relate to the context in which the problem is solved. The designer produces various drawings, notes and diagrams as part of the solution aimed to meet these requirements. The step of transforming requirements into a project is crucial, and mistakes made in processing of requirements during the design become a source of many software projects failure.

The program results of the second mini-course are:

- ability to build charts, in particular UML, to represent requirements;

- ability to use needs management in the software development life cycle;
- ability to choose software development method suitable for managing specified requirements;
- ability to apply methods of validation, verification and tracking of requirements;
- ability to develop strategies for transferring information about needs to high-level design [13, pp. 6-7].

The program results of the "Software Modeling" course, developed by Sedelmaier and Landes [6], are:

- deepening the understanding of the term "requirements" and their role in software development;
- ability to use specification of functional and non-functional requirements methods and determination of their priorities;
- understanding the role of communication with other parties involved in the development of requirements;
- understanding the role of business processes as a source of requirements;
- ability to jointly apply appropriate methods and designations to determine the requirements for software product example;
- ability to use methods to assess the complexity and cost of software systems.

The objectives of the "Software Modeling" course are:

1) forming student understanding of the role and importance of requirements for their future careers – students must know the problems of requirements development, recognize their importance and difficulties in their formation; students must be able to extract requirements from future users, model business processes and create documents with requirements;
2) improvement of specific communication skills demanded for requirements development – students must be able to meet with customers for identify requirements that they did not invent themselves, but formulated in collaboration with a real customer, learn to ask customers about information that can be the basis for requirements, document requirements, assign roles, etc.;
3) strengthening self-reflection, self-organization and responsibility of students as a basis for development of appropriate competence.

The learning environment for such course is proposed by Sedelmaier and Landes [6] on the basis of a constructivist approach, in which teachers act as trainers, creating conditions for students to gain individual learning experience. Without determining the special ability to identify, classify and formulate software requirements, the researchers highlight following 4 groups of competencies necessary for the formation of such ability [6]:

1) problem awareness: knowledge of the subject area, ability to abstract;
2) context sensitivity: ability to moderate and present, meet with customers, integrate into a team, empathy, endurance;
3) personal competencies: methods of work, self-organization, role distribution, time management, personal involvement, purposefulness, ability to self-reflection;

4) creativity, variety of techniques.

The main components of the approach proposed in [6] are the broad, active student participation in learning and a realistic, integrated environment, which includes writing a document with requirements for a complex project and extracting requirements from real clients, as they play a dual role in addition to the source of requirements, they also act as external communication experts.

3. Conclusions

1. An overview of the sources on the research problem show the commonality of problems in teaching requirement engineering of future Software Engineering specialists:
    - student lack of understanding of the importance of requirement engineering due to insufficient experience in working with real customers;
    - insufficient attention to the process of identifying interrelated requirements in communication with stakeholders who do not interact with each other;
    - bias in learning languages and means of modeling requirements, in which their detailing precedes detection;
    - high clarity of requirements for training projects with Software Engineering in comparison with real ones;
    - nonlinear dependence of the software project complexity on the growth of the number of requirements.
2. The ways to overcome these problems in teaching requirement engineering for training Software Engineering specialists:
    - use of activity and constructivist approaches, game teaching methods in the process of modeling requirements;
    - active involvement of stakeholders in identifying, formulating and verifying requirements at the beginning of the project and evaluating its results at the end;
    - application of mobile technologies for training of geographically distributed work with requirements;
    - implementation of interdisciplinary cross-cutting Software Engineering projects;
    - involvement of students in real projects;
    - stimulating the creation of interdisciplinary and age-old student project teams.

References

[1] A. Striuk, "Advanced course on software engineering" as the first model for training of software engineers, Journal of Information Technologies in Education (ITE) (2019) 48–67. URL: http://ite.kspu.edu/index.php/ite/article/view/732. doi:10.14308/ite000702.

[2] A. Striuk, S. Semerikov, The dawn of software engineering education, CEUR Workshop Proceedings 2546 (2019) 35–57.

[3] Software Engineering 2014, Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, A Volume of the Computing Curricula Series, 2015. URL: https://www.acm.org/binaries/content/assets/education/se2014.pdf.

[4] S. Semerikov, A. Striuk, L. Striuk, M. Striuk, H. Shalatska, Sustainability in Software Engineering Education: a case of general professional competencies, E3S Web of Conferences 166 (2020) 10036. doi:10.1051/e3sconf/202016610036.

[5] CC2020 Task Force, Computing Curricula 2020: Paradigms for Global Computing Education, A Computing Curricula Series Report, 2020. URL: https://www.acm.org/binaries/content/assets/education/curricula-recommendations/cc2020.pdf.

[6] Y. Sedelmaier, D. Landes, A multi-level didactical approach to build up competencies in requirements engineering, CEUR Workshop Proceedings 1217 (2014) 26–34.

[7] S. Ouhbi, A. Idri, J. Fernández-Alemán, A. Toval, Requirements engineering education: a systematic mapping study, Requirements Engineering 20 (2015) 119–138. doi:10.1007/s00766-013-0192-5.

[8] S. T. Acuña, M. Gómez, N. Juristo, How do personality, team processes and task characteristics relate to job satisfaction and software quality?, Information and Software Technology 51 (2009) 627–639. doi:10.1016/j.infsof.2008.08.006.

[9] D. Damian, A. Hadwin, B. Al-Ani, Instructional Design and Assessment Strategies for Teaching Global Software Development: A Framework, Association for Computing Machinery, New York, NY, USA, 2006, p. 685–690. URL: https://doi.org/10.1145/1134285.1134391.

[10] V. Tkachuk, S. Semerikov, Y. Yechkalo, S. Khotskina, V. Soloviev, Selection of mobile ICT for learning informatics of future professionals in engineering pedagogy, CEUR Workshop Proceedings 2732 (2020) 1058–1068. URL: http://ceur-ws.org/Vol-2732/20201058.pdf.

[11] L. Mich, Teaching requirements analysis: A student project framework to bridge the gap between business analysis and software engineering, CEUR Workshop Proceedings 1217 (2014) 20–25.

[12] A. Goswami, G. Walia, Teaching software requirements inspections to software engineering students through practical training and reflection, Computers in Education Journal 16 (2016) 2–10.

[13] I. Ozkaya, Ömer Akin, J. E. Tomayko, Teaching to Think in Software Terms: An Interdisciplinary Graduate Software Requirement Engineering Course for AEC Students, 2005, pp. 1–10. URL: https://ascelibrary.org/doi/abs/10.1061/40794%28179%298. doi:10.1061/40794(179)8.