Journal of Software Engineering and Applications, **2013, 6, 543-553**
http://dx.doi.org/10.4236/jsea.2013.610065 Published Online October 2013 (http://www.scirp.org/journal/jsea)

Influence of Software Modeling and Design on Domain-Specific Abstract Thinking: Student's Perspective

Zakarya A. Alzamil

Software Engineering Department, King Saud University, Riyadh, Saudi Arabia.
Email: zakarya@ksu.edu.sa

Received August 5th, 2013; revised September 4th, 2013; accepted September 12th, 2013

ABSTRACT

Software engineering has been taught at many institutions as individual course for many years. Recently, many higher education institutions offer a BSc degree in Software Engineering. Software engineers are required, especially at the small enterprises, to play many roles, and sometimes simultaneously. Beside the technical and managerial skills, software engineers should have additional intellectual skills such as domain-specific abstract thinking. Therefore, software engineering curriculum should help the students to build and improve their skills to meet the labor market needs. This study aims to explore the perceptions of software engineering students on the influence of learning software modeling and design on their domain-specific abstract thinking. Also, we explore the role of the course project in improving their domain-specific abstract thinking. The study results have shown that, most of the surveyed students believe that learning and practicing modeling and design concepts contribute to their ability to think abstractly on specific domain. However, this finding is influenced by the students' lack of the comprehension of some modeling and design aspects (e.g., generalization). We believe that, such aspects should be introduced to the students at early levels of software engineering curriculum, which certainly will improve their ability to think abstractly on specific domain.

Keywords: Domain-Specific Abstract Thinking; Software Modeling; Software Design; Teaching Modeling and Design; Software Engineering Education

1. Introduction

Software engineering disciplines have been taught for many years as individual courses within the curriculum of computing areas such as computer science and information system, which caused the lack of software engineering graduates. Recently, many universities have established an undergraduate (BSc) and/or graduate programs (MSc) in software engineering, among which is the King Saud University (KSU) in Riyadh, Saudi Arabia that has founded BSc and MSc programs. The BSc in software engineering at KSU was approved in June 2007, and the first batch of students have started the program on the second semester of the academic year 2008/2009.

Software engineering community has recognized the importance of software engineering education and has organized several conferences and workshops that are specialized on the software engineering education. Examples of such conferences are the Conference on Software Engineering Education and Training (CSEET), education track of the International Conference on Software Engineering (ICSE), and Frontiers in Education conference (FIE). There are a number of software engineering education's studies that have been published in the literature. These studies can be categorized into many fields, such as investigating new teaching methods, integrating ethics into computing, approaches to software engineering course project, and enhancements to the software engineering curricula. Professional organizations such as ACM and IEEE have developed a guidelines handbook for the BSc program in software engineering [1].

Software engineering educators face many challenges in delivering the software engineering knowledge and skills that are needed in the labor market. Unfortunately, most of the researches in the area of software engineering concentrate on the practical and managemental issues, and few have investigated how to teach these aspects.

The joint IEEE/ACM software engineering 2004 curricula [1] emphasizes the importance of integrating theory and practice, so that students can recognize the importance of abstraction and modeling, and their influence

on understanding the domain knowledge beyond the computing discipline. Although the software engineering 2004 curricula provide important information for the instructors to consider for modeling and design courses as well as the course project, most of the instructors need more guidance on how to teach the software engineering disciplines. Moreover, most of the international organizations' standards, such as ABET and SWEBOK, did not provide practical guidelines for teaching software engineering disciplines and the capstone project, e.g. [1-3]. Teaching software modeling and design is a major challenge to software engineering educators because of the complexity of software modeling and design concepts when compared to other aspects of software engineering. Also, many software modeling and design techniques and approaches are evolving, and at the same time, not enough practical and only small examples are available for classroom use to illustrate these techniques and approaches [4].

Among the basic design concepts that are taught to the software engineering's students are abstraction, separation of concerns, and modularity [5]. Abstraction can be defined as the process of forgetting information so that things that are different can be treated as if they were the same [6]. The software engineering's students were taught two basic rules in abstraction: look at details and abstract up to concepts, and/or choose concepts, then add detailed substructures to move down. Based on this premise, abstraction focuses on two aspects: removing unnecessary details and the process of generalizing concepts and finding patterns [7]. Abstraction is very important and has been considered a "key skill" in computing, that a software engineer should master [8]. A key point that enhances the abstraction in domain analysis is separation of concerns. Separation of concerns means dividing the problem under analysis into independent parts, such as separating system components from their connectors. Modularity is a general system concept, typically defined as a continuum describing the degree to which system's components may be separated and recombined [9]. So it means that the system is divided into functional units (modules) that make up a larger application representing the whole system.

Abstract thinking is defined in [10] as the ability to use concepts and to make and understand generalizations, such as the properties or pattern shared by a variety of specific items or events. In [11] the abstract thinking is defined as the final, most complex stage in the development of cognitive thinking, in which thought is characterized by adaptability, flexibility, and the use of concepts and generalizations. Usually, abstract thinking is compared with concrete thinking which can be defined as predominance of actual objects and events and the absence of concepts and generalizations [10], in which thinking about objects, ideas, or events is within their attributes and relationships. In contrast, abstract thinking is conceptualized or generalized thinking that understands each concept in multiple meanings, in which objects, ideas, or events are separated from their attributes and relationships to think "outside of the box" to come up with creative solutions.

In this paper we present the perspective of a set of software engineering's students on the influence of learning software modeling and design on their domain-specific abstract thinking. Software modeling and design in this research is used within the context of system's software architecture. In this context software design is an activity that creates part of a system's architecture in which a set of design decisions that are related to the system's structure, its primary components and their interactions, system's behavior, non-functional properties, etc., are made. In addition, software design considers stakeholder issues, decision about use of COTS component, architecture styles, and deployment issues. The software modeling is an artifact that captures some or all of the design decisions that comprise a system's architecture using specific notation. The software model should depict the major element of the architectural design such as components, connectors, interfaces, and configurations [5].

The aim of this study is to understand the impact of learning software modeling and design in improving the students' ability to think abstractly within specific problem domain, e.g. developing a specific software system.

The proposed study was conducted by studying the perception of 107 software engineering's students at King Saud University, representing 4 batches (levels). The following section presents the motivation of this work, next section describes the related work, then the study's questions and research methodology are illustrated, after that the study results are discussed, and finally the conclusions are presented in the last section.

2. Motivations

Most of the computing curriculums start with introductory programming courses that are introduced to the students using a traditional approach. These courses are taught to students using a so-called late-object approach e.g., [12,13], in which the students learn the basic elements of a programming language, and then they start thinking of the system as real world objects. We have observed that, when teaching software modeling and design, students tend to think about the details of the system components such as the source code and the content of the system's components, rather than thinking at a high level of abstraction.

Ideally, we assumed that, abstract thinking should contribute to the students' skills in modeling and designing software systems. On the other hand, the assumption is that, learning the modeling and design may improve the students' ability to think abstractly about the domain problem. Software modeling and design introduces the students to the conceptual understanding of domain problem, in which the domain concepts are generalized and separated from their attributes, properties and relationships shared by different objects or events. In addition, abstract thinking may improve the students' systems thinking by viewing the domain problem as parts of an overall system. Also, it has been reported that, using analogy in teaching can help the students to build a conceptual bridge between what they knew and what they are about to learn [14], which may lead to improve their conceptual understanding of the domain problem. Our observation of using analogy of physical buildings with software systems has shown that, students were able to think of the software system as blueprint of the system big picture as they do when thinking about the physical building's blueprint.

It has been observed that, in order for students to be competent and skilled in certain subject, they should practice what they have learned in the classroom within a real or semi-real world environment. One of the teaching approaches that expose the students to the real world environment is the course project. Course project is a course work assignment during an academic semester requiring the students to work as a team to develop a software system solution for a specific domain problem. Course project is very important tool that provides the students with the practical modeling and design skills that may lead to abstract thinking toward creating solutions for real world problems. There have been many research studies that had shown the benefits of employing the course project to gain the occupational skills needed for software development, e.g., [15-20]. Although, some studies [16] measured the time that students spend on each phase of the project and have shown that most of the time has been consumed in the meetings, students' involvement in brainstorming with project team's members during the early stages of the system development is a good technique that provides them with the opportunity to improve their way of thinking including domain-specific abstract thinking. We believe that, course project as a tool of teaching practical software modeling and design impacts the students' learning and comprehension of modeling and design and as a result will contribute to abstract thinking. Therefore, we would like to understand the students' perspective on the influence that software modeling and design may have on their domain-specific abstract thinking in terms of learning modeling and design in general and course project as a tool in specific.

3. Related Work

Software engineering education, relatively, is a new research track, in which, teaching software modeling and design has been recognized recently. In [4], a finite-state model has been adapted for classroom teaching of software modeling. A research study has been performed to determine whether there is a link between the abstraction skills of students and their success in object-oriented modeling [7]. This study has shown that students with high scores in an abstraction test achieve better results in an object-oriented analysis and design module than those with lower scores. In [21], the authors emphasis on the importance of the abstraction and information hiding as fundamental and essential principles in software development, and should be taught in the early computer science courses such as computer programming I and computer programming II courses, so that educators can use different techniques to provide the students with the tools they need to develop their abstract thinking. The authors support their claim with two examples one for computer programming I and another for computer programming II. An iterative teaching approach to teaching object-oriented programming has been presented in [22] using modeling languages, in particular UML, to support abstract thinking to understand the basic object-oriented concepts. This approach applies constant cycling between design and code, *i.e.*, high and low abstraction levels, by visualizing the concepts, discussing their relation to the abstraction, generating and changing the implementation (code), and iterating the process. A visualization model has been proposed in [23] that can be combined in teaching process to build a reasonable model for abstract concepts' teaching to improve teaching. The results of surveying 200 software engineering's students showed that, students can use and manage knowledge easier and can improve the learning efficiency.

In [24], a Model-Driven Engineering (MDE) has been integrated into the software design course through the course project to observe how MDE helps the students to understand modeling concepts and the tools that support them. The results of this study have shown a positive impact on helping students to better understand how models can be used during software design. In [25], a guideline within the context of an Aspect-Oriented process support has been proposed for enabling an Integrated Development Environment (IDE) to promote abstract thinking. In this study lab activities have been conducted within Eclipse environment, in which thirty-three students studied the abstract thinking, and worked on real-life development tasks, where their development steps as

well as their visible thinking processes were logged by observers, and a reflection on the activity was collected. The authors concluded with that, concrete IDE support for abstract thinking is practical and suggest two kinds of such support, one is concerned with a positive feedback from the IDE in cases where abstraction is used, and the second with cases where the developer is encouraged to use abstract thinking. In [26], an approach has been presented to improve the student's engineering ability by shifting the engineering education from the simple presentation of knowledge toward computational thinking, in which the knowledge and the development are integrated in all courses teaching via training and practice. The presented approach focuses on extraction of fundamental discipline concept of engineering ability development, problem solving-centered organization of courses of software development tools, initiate the courses of software engineering as soon as possible, and continual training of abstract logical thinking for the purpose of software abstract thinking. An experience of teaching modeling at the high school level has been presented in [27], in which the abstract thinking processes involved in modeling are introduced to the students prior to teaching programming and embedded control. A UML modeling has been used to teach students to analyze various applications, systems and problem domains. The study tried to answer if the modeling should be introduced to the high school students or first year college. The study has shown that, students are learning to model and are finding abstractions for elements in the application domain rather than jumping into implementation too quickly. Also, the data modeling comes naturally, in which students were able to identify the classes, attributes, associations, and state machines involved in several simple systems. In addition students find abstraction is much easier to understand than implementation syntax. An experience of using abstraction to teach software engineering human aspects has been presented in [28]. The authors suggested introducing reflective and abstract thinking processes into courses that focus on improving students' analytical skills and problem-solving abilities.

Although some of the aforementioned studies have investigated the impacts of modeling and design on abstract thinking, many of them are conducted on a focus group of students attending specific courses that were established for such research. However, we would like to study the impact of regular software modeling and software design courses on students' ability to think abstractly on specific domain without any justification of such courses. Therefore, the purpose of this paper is to investigate the students' perception of the influence of learning software modeling and design on their domain-specific abstract thinking.

4. Study Questions

The purpose of this study is to understand the students' perspectives on the influence of learning software modeling and design on their domain-specific abstract thinking. As described earlier, hypothetically, learning software modeling and design as well as involving in a course project may improve the students' ability to think abstractly. Understanding the students' perception on such matter will help to validate such hypothesis. Therefore, we have identified two questions that help in understanding the students' viewpoint. The study questions are:

1) In the viewpoint of the students, to what extent learning software modeling and design enhances their domain-specific abstract thinking?

2) In the viewpoint of the students, to what extent the course project improves their domain-specific abstract thinking?

5. Research Methodology and Tool

In order to answer the study questions to understand the students' perceptions, we have developed a questionnaire as a study tool based on abstraction aspects that contribute to domain-specific abstract thinking; namely: generalizations (5 statements), separation of concerns (1 statement), and modularity (3 statements, including a shared statement with generalization), as well as analogy (1 statement). Also, we have injected one statement as an alarm statement to be used for filtering out the inaccurate responses (e.g., randomly filled questionnaire). In addition, 3 statements (with respect to abstraction) were added to serve the second question of the study. The questionnaire, see the Appendix, consists of 13 statements with three choices for each statement (agree, may be, and disagree), in which 9 statements are designed for answering the first question of the study, 3 statements for the second question, and 1 statement as an alarm statement. In this questionnaire, we have used a three-point scale instead of the famous five-point scale; because the respondents, in many cases, avoid using extreme responses such as "strongly agree" and "strongly disagree". In addition, sometimes during responses analysis, the agree responses ("strongly agree" and "agree") and disagree responses ("strongly disagree" and "disagree") of the five-point scale are combined into two categories of "accept" and "reject" respectively, for normalization purposes that leads to three-point scale. The questionnaire's language has been simplified by substituting some words with others for the sake of making it understandable to non-English speaking students.

The questionnaire has been used as a tool to understand the students' perspectives on the influence of the

software modeling and design in building their skills for domain-specific abstract thinking. Students' perception has been used because learning is very difficult to be measured by standard measurement techniques; therefore, understanding the students' opinion is the closest way in measuring the influence of learning software modeling and design on improving their domain-specific abstract thinking skill.

In order to understand the influence of software modeling and design on the students' domain-specific abstract thinking, we have considered a sample of software engineering's students who finished the modeling and design courses at KSU, namely; SWE313 (Software Process and Modeling) and SWE321 (Software Design and Architecture). The students in these two courses learned and practiced different techniques of software modeling and design via lectures, laboratories, and teamwork project assignments with several practical examples that concentrate on different abstraction aspects. The targeted sample consists of 107 students representing 4 batches/groups that were admitted to the program (*i.e.*, their study level in the program, in which batch (1) is the most advanced group in the BSc program). We were able to get the feedback from 81 students, in which the questionnaire's statements were recited and explained to the students to assure that they understand them clearly to get a precise feedback. The data was collected and analyzed. **Table 1** displays the study sample properties.

In order to assure that the questionnaire statements correlate to the study questions, we have computed the correlation co-efficiency of the statements of the study questionnaire to the question that they belong to. In addition, we have computed the correlation co-efficiency of each study question to the total score of all study questions. **Table 2** displays the correlation co-efficiency of the study's questions. As can be seen, the study questions are correlated to the total scores of all questions with more than 0.20 which means that the statements of each question belong to that question with a good correlation.

In addition, we have performed a reliability analysis test to assure whether a group of statements that belong to a question measure that question. A Cronbach's alpha has been used to measures how well a set of items (statements) measures a single one-dimensional latent construct (question). When the value of Cronbach's alpha is low, it means that the statements measure multi-dimensional construct; however, when such value is high it means that the statements measure one-dimensional construct (the designated question), which indicate the reliability of the study questions. **Table 3** shows the value of Cronbach's alpha for the questionnaire's statements for all study questions. The value of Cronbach's alpha is high enough to indicate that the questionnaire's statements measure the study questions.

We have used other statistical test to understand the variations between the respondents. Analysis of variance (ANOVA) is a general method for studying sampled-data relationships [29]. The method enables the difference between two or more sample means to be analyzed, achieved by subdividing the total sum of squares. We have used the one way ANOVA test (*f* test) to study the relations between the study samples (4 batches/groups) in terms of their responses variations to identify whether such variation is statistically significant. In case we detect any variation between the study samples' responses, we use Scheffe test to identify the source of such variation. In the following section, we will discuss the results findings.

6. Result Discussion

In order to understand the students' perceptions on the influence of learning software modeling and design in building their domain-specific abstract thinking, we will answer the study questions by analyzing and discussing the students' responses to the questionnaire's statements that are related to the abstraction aspects (generalization, separation of concerns, and modularity). As described earlier, five statements are based on generalization, three on modularity (including the shared one with generalization), one on separation of concerns, one on analogy, and three on abstraction related to the course project. In the following subsections we discuss the students' responses related to the study's questions.

Table 1. Study sample properties.

Batch# (Group#)	*Frequency*	*Percent*	*Cumulative Percent*
Batch (1)	28	34.6	34.6
Batch (2)	8	9.9	44.4
Batch (3)	32	39.5	84.0
Batch (4)	13	16.0	100.0
Total	81	100.0	

Table 2. Correlation of the two questions to the total score.

Study Question	*Correlation Coefficient*	*Number of Statements*
Q1	0.992	9
Q2	0.917	3

Table 3. Cronbach's Alpha for the study's questions.

N of Items	*Cronbach's Alpha*
13	0.957

6.1. First Question

In the viewpoint of the students, to what extent learning software modeling and design enhances their domain-specific abstract thinking?

To answer this question, let us first discuss the responses on the questionnaire's statements that are related to generalization. Consider **Table 4**, as can be seen at statement S11, the majority of the students (64.2%) believe that learning modeling and design techniques improve their comprehension of different modeling and design concepts such as generalization and decomposition, and 28.4% believe it may help so. Also, when the students were asked about whether focusing on the concepts of the system under development would contribute in understanding the system big picture (statement S8), the majority of the students (55.6%) agree on such hypothesis whereas 40.7% believe it may do so. These results are supported by the responses of the students when asked whether thinking abstractly contributes to modeling the big picture of the problem domain, as can be seen at statement S7, 48.1% of the respondents agree on such premise whereas 39.5% think it may contribute to modeling the big picture of the problem domain. Additionally, **Table 4** shows the students' responses regarding using the simple-machine approach in describing the problem domain to simplify problem understanding (statement S1). Simple machine was described to the respondents as an abstraction of a potential system that will perform a required task. The result shows that 45.7% of the students believe that using simple machine helps them to understand the problem domain whereas 50.6% of them are not sure. As can be noticed, very few students disagree with the aforementioned hypotheses. Such result describes what the students believe as a first impression when learning and applying software modeling and design techniques.

Table 4. Students' responses to statements related to generalization.

Statement	*Agree*	*Maybe*	*Disagree*	*Missing*
S11: Learning modeling and design techniques improves my comprehension of modeling and design concepts such as generalization, decomposition, abstraction, projection/views, and explicitness	64.2	28.4	7.4	0
S8: Focusing on the concepts of the system under development contributes in understanding the system big picture	55.6	40.7	2.5	1.2
S7: Thinking abstractly contributes to modeling the big picture of the problem domain	48.1	39.5	11.1	1.2
S1: Using simple-machine to describe the problem domain simplifies problem understanding	45.7	50.6	3.7	0
S6: Learning and applying modeling and design makes me focused on the big picture of the system without thinking in the details	34.6	54.3	11.1	0

However, such results are disturbed with the students' responses when asked whether modeling and design techniques make them focused on the big picture of the system without thinking in the details. As shown at statement S6, 34.6% of the students agree with such argument, and the majority of the students (54.3%) are not sure of such premise. We believe that, the reluctantly responses come from the fact that, most of the students tend to think in details when modeling and designing a software system, therefore, when asked about being focused on the big picture without thinking about the details, the answer may be rephrased as "well we have been taught to think about the details". It has been noticed that, most of the computing curriculums, including software engineering curricula, start with programming courses, in which the students learned to look at details to develop the design and then implement it using a selected programming language. Regardless of the suitability of such curriculum approach, it certainly, influences the way that students comprehend and practice generalization. We believe that, the students should learn and practice generalization and how to identify the levels between abstraction and details, in which they can think abstractly about the domain problem without going into very low level of details.

Although such responses may be interpreted differently, we believe that, such results can be interpreted as evidence of the advantage of using software modeling and design techniques in learning generalization to improve the students' ability to think abstractly for specific domain problem. As a natural response of the students when exposed to modeling and design techniques, they believed that such techniques may help them to think abstractly about specific domain; however, due to the lacks of enough practical and/or tutorials for software modeling and design, they are likely to look into the details.

The second aspect of abstraction is related to modularity, which is discussed through the responses of the statements shown in **Table 5**. But before discussing these responses, let us consider the responses to statement S11 in **Table 4**, as described earlier, 64.2% of the students

believe that learning modeling and design techniques improve their comprehension of different modeling and design concepts such as decomposition (*i.e.*, modularity), and 28.4% believe it may help so, which is considered as a positive response towards answering the study's first question. This response is supported by the responses to statement S2 shown in **Table 5**, which states whether using diagrams as modeling and design tool helps in modeling and designing the big picture of the system. As noticed, most of the respondents (86.4%) agree on the benefit of using diagrams as a tool for modeling and designing the big picture of the system. Also, **Table 5** shows the responses to statement S9 on whether the availability of many choices of modeling and design techniques contributes in improving their ability in thinking and comparing abstractly, in which 43.2% of the responses agree on such premise whereas 37% state it may contribute to thinking and comparing abstractly. Such result indicates that learning modeling and design techniques, most likely, contribute to improving the modularity skill, in which the students can divide the system into functional modules. We believe that, such result is achieved because modularity, unlike generalization, requires more level of details, in which several system components can be constructed to make up a larger application representing the whole system.

The third aspect of abstraction is separation of concerns, which is represented by statement S12 that is shown in **Table 6**. The software component was described to the respondents as an architectural element that captures a subset of the system's functionality and/or data. In addition, the software connector was described to the respondents as an architectural element that models and regulates the interactions among components. As can be seen, 43.2% of the students' responses agree on the hypothesis that learning modeling and design make them able to separate component from connectors, whereas 49.4% believe it may do so. Such result is encouraging towards the influence of the software modeling and design techniques on the abstraction in which almost half of the students believe such techniques improve their ability of separating different system's concerns.

Another aspect that has been used widely in modeling and design is analogy which has been taught to the students in the SWE321 course. We have designed one statement to understand the students' perception on the influence that analogy may have on domain-specific abstract thinking. Consider **Table 7**, which shows the students' responses to statement S13 on whether using analogy is a good tool to improve their domain-specific abstract thinking skill. As can be seen, 74.1% of the respondents agree on such premise, whereas 21% think it may do so.

We have studied the variations between the respondents' answers to the statements of this question with respect to the students' batch/group using ANOVA to identify whether such variation is statistically significant. **Table 8** shows the variations between the study samples responses to this question's statements in which *f* value and the significance is displayed. As can be noticed, there is no significant variations between the responses of the study sample to this question with respect to their batch/group *i.e.*, their study level.

The results of this part of the study show that, teaching and learning software modeling and design techniques such as analogy help the students to improve their domain-specific abstract thinking, however, such improve-

Table 5. Students' responses to statements related to modularity.

Statement	*Agree*	*Maybe*	*Disagree*	*Missing*
S2: Using diagrams as modeling and design tool helps in modeling and design the big picture of the system	86.4	12.3	1.2	0
S9: The availability of many choices of modeling and design techniques contributes in improving my ability in thinking and comparing abstractly	43.2	37.0	19.8	0

Table 6. Students' responses to the statement related to separation of concerns.

Statement	*Agree*	*Maybe*	*Disagree*	*Missing*
S12: Learning modeling and design makes me able to separate component from connectors	43.2	49.4	7.4	0

Table 7. Students' responses to the statement related to analogy.

Statement	*Agree*	*Maybe*	*Disagree*	*Missing*
S13: Using analogy is a good tool to improve my abstract thinking skill	74.1	21.0	3.7	1.2

Table 8. Variations of students' responses to statements related to the first question.

Source of Variance	*Sum of Squares*	*df*	*Mean Square*	*F Value*	*Sig.*
Between Groups	33.349	3	11.116	0.472	0.703
Within Groups	1813.491	77	23.552		
Total	1846.840	80			

ment suffers from the students' lack of a very important aspect of abstraction which is the generalization aspect as has been discussed earlier.

6.2. Second Question

In the viewpoint of the students, to what extent the course project improves their domain-specific abstract thinking?

It has been recognized that students should be exposed to the real world problems, so that, they can practice what they have learned in a real world environment. Course project has been considered to be one of the most important means in which the students can practice, improve, and test their skills. As described earlier, course project is a course work assignment during an academic semester requiring the students to work as a team to develop a software system solution for a specific domain problem. In this part of study we investigate whether the course project improves the students' domain-specific abstract thinking. We have designed three statements, within the study questionnaire, that are related to the influence that the course project may have on improving the students' domain-specific abstract thinking. In the following paragraphs we will discuss the responses to these statements.

Consider **Table 9**, which shows the students' responses to statements related to course project. The responses of the students to statement S4 on whether the course project improves the way they think about a problem domain, show that, the majority of the students (61.7%) agree on such hypothesis whereas 30.9% believe it may do so. Also, the students' responses to statement S3 on whether dealing with real world problem in the course project improves their conceptual thinking about the problem solution, most of the students (77.8%) agree on such premise, whereas 21% think it may do so. In addition, when the students were asked, whether a teamwork and brainstorming with their colleagues make them think abstractly about the system (statement S5), the majority of the students (72.8%) agree with such argument, whereas 24.7% think it may do so.

We have studied the variations between the respondents' answers to this question with respect to the students' batch/group using ANOVA to identify whether such variation is statistically significant. **Table 10** shows the variations between the study samples responses to this question's statements in which *f* value and the significance is displayed. As can be seen, there is a signifycant variation at 0.05 between the respondents' answers to the statements of this question with respect to their batch/group. In addition, we have performed Scheffe test to identify the source of such variation and found that there is a significant variation at 0.05 between respondents with respect to their batch/group to the favor of batch (1), which is the most advanced group in the BSc program. Such result is understandable because the first batch had experienced more courses' projects than later batches due to their advancement in the study program, which may contribute to their domain-specific abstract thinking improvement.

Table 9. Students' responses to the statements related to second question.

Statement	*Agree*	*Maybe*	*Disagree*	*Missing*
S4: The course project improves the way I think about a problem domain	61.7	30.9	6.2	1.2
S3: Dealing with real world problem in the course project improves my conceptual thinking about the problem solution	77.8	21.0	0	1.2
S5: Teamwork and brainstorming with my colleagues makes me think abstractly about the system	72.8	24.7	2.5	0

Table 10. Variations of students' responses to statements related to the second question.

Source of Variance	*Sum of Squares*	*df*	*Mean Square*	*F Value*	*Sig.*
Between Groups	18.104	3	6.035	2.751	.048
Within Groups	168.884	77	2.193		
Total	186.988	80			

Although such results show that, most of the students believe that the course project may improve their domain-specific abstract thinking, we believe that utilizing the course project to improve the students' domain-specific abstract thinking depends on the nature of the domain problems that the students work on as well as the activities that associated with the course project such as teamwork, brainstorming, reviewing one another work, and so on.

As mentioned earlier, we have inserted one statement (statement S10) as an alarm statement, which states an opposite meaning of statement S9, and has been used to filter out the falsely filled questionnaire. We have examined all responses looking for any randomly filled questionnaire to filter it out as false response, and find nothing.

7. Research Findings

We can summarize the perceptions of the surveyed students regarding the influence of the software modeling and design on their domain-specific abstract thinking in

the following points:

1) Students' believe that learning modeling and design techniques improve their comprehension of different modeling and design concepts such as generalization and decomposition.

2) Although, focusing on the concepts of the system under development would contribute in understanding the system big picture, most of the students tend to think in details when modeling and designing a software system which influences the way they comprehend and practice generalization.

3) Students find that, using diagrams as a tool for modeling and designing the system under development helps them in comprehending the system big picture.

4) Students think that, the availability of many choices of modeling and design techniques contributes in improving their ability in thinking and comparing abstractly for specific domain.

5) Students find that the software modeling and design techniques improve their ability in separating different system's component from their connectors.

6) Students find that, using analogy is a good tool to improve their domain-specific abstract thinking skill.

7) We have found that the course project improves the way they think about a problem domain, which may lead to improving their conceptual thinking about the problem solution. Additionally, the teamwork and brainstorming with colleagues may help the students to look at the domain problem differently, which may contribute to their problem-domain abstract thinking.

Although such findings show that, learning and practicing software modeling and design concepts contribute to the student ability to think abstractly on specific domain, students should learn by practice some modeling and design aspects (e.g., generalization) at early levels of software engineering curriculum (e.g., programming and introductory software engineering courses), which certainly will improve their domain-specific abstract thinking. Although of the encouraging results of the influence of modeling and design on students' domain-specific abstract thinking, this study has been conducted on small software development projects during an academic semester. Therefore, more investigation is needed to better understand the influence of software modeling and design on improving domain-specific abstract thinking on larger domains within the real world software development.

8. Conclusions

In this paper, we have presented a study of the students' perception on the influence of learning software modeling and design on their domain-specific abstract thinking. This study is designed to answer whether the students think that learning software modeling and design enhances their domain-specific abstract thinking. In addition, we investigate the students' perspective on the role of the course project in improving their abstract thinking for specific domain. The study has been performed by surveying groups of software engineering's students who studied or are studying the software modeling and software design courses. A questionnaire has been developed to serve as the study tool to answer the study questions.

The results of the study have shown that, most of the surveyed students believed that learning and practicing software modeling and design techniques improve their domain-specific abstract thinking. However, such improvement is suffered by the students' lack of some basic skills related to modeling and design such as some abstraction aspects. Therefore, more emphasis has to be considered on teaching and practicing abstraction aspects such as generalization. Although students' perception on the influence of software modeling and design on improving their domain-specific abstract thinking are optimistic, many practical modeling and design aspects have to be embedded within the software engineering curriculum. Specifically, the course project and the course laboratory should offer the students the opportunities to observe practical aspects that improve their domain-specific abstract thinking.

In spite of the encouraging results of this study, more investigations are needed to better understand the influence of software modeling and design on abstract thinking of larger domains. In addition, we believe that, the perception of the software engineering practitioners on the influence of software modeling and design to improve their domain-specific abstract thinking is another dimension that may contribute not only to software modeling and design education, but to the software engineering education in general. Therefore, we plan to investigate the software engineering practitioners' perceptions on practicality of software design and architecture curriculum on their daily tasks, which may help us better understand the industry perspectives as well as closing the gap between the industry and academia.

REFERENCES

[1] IEEE/ACM, Software Engineering, "Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, a Volume of the Computing Curricula Series," 2004. http://sites.computer.org/ccse/

[2] ABET, "Criteria for Accrediting Computing Programs, Computing Accreditation Commission," 2004. http://www.abet.org

[3] A. Abran, J. Moore, P. Bourque, R. Dupuis and L. Tripp, "Guide to the Software Engineering Body of Knowledge (SWEBOK)," IEEE Computer Society Professional Prac-

tices Committee, 2004. http://www.computer.org

[4] S. Kundu, "Teaching Software Modeling and Design Based on the Science of Design and Science of Learning," *Proceedings of the IEEE International Conference on Frontiers in Education*: *Computer Science & Computer Engineering*, Las Vegas, 14-17 July 2008, pp. 434-438.

[5] R. Taylor, N. Medvidovic and E. Dashofy, "Software Architecture: Foundations, Theory, and Practice," John Wiley & Sons, Hoboken, 2010.

[6] B. Liskov and J. Guttag, "Program Development in Java: Abstraction, Specification, and Object-Oriented Design," Addison-Wesley, Boston, 2001.

[7] P. Roberts, "Abstract Thinking: A Predictor of Modeling Ability?" *Proceedings of the ACM/IEEE* 12*th International Conference on Model Driven Engineering Languages and Systems*: *Educators' Symposium*, Denver, 4-9 October 2009.

[8] J. Kramer, "Is Abstraction the Key to Computing," *Communications of the ACM*, Vol. 50, No. 4, 2007, pp. 36-42. http://dx.doi.org/10.1145/1232743.1232745

[9] M. Schilling, "Towards a General Modular Systems Theory and Its Application to Inter-Firm Product Modularity," *Academy of Management Review*, Vol. 25, 2000, pp. 312-334.

[10] "The American Heritage Medical Dictionary," Houghton Mifflin Company, Harcourt, 2007.

[11] "Mosby's Medical Dictionary," Elsevier, Amsterdam, 2009.

[12] P. Deitel and H. Deitel, "Java How to Program: Late Objects," 8th Edition, Prentice Hall, Upper Saddle River, 2009.

[13] C. Horstmann, "Big Java Late Objects," Wiley, Hoboken, 2012.

[14] S. Glynn, "Methods and Strategies: The Teaching-with-Analogies Model," *Science and Children*, Vol. 44, No. 8, 2007, pp. 52-55.

[15] Z. Alzamil, "Towards an Effective Software Engineering Course Project," *Proceedings of the ACM* 27*th International Conference on Software Engineering* (*ICSE*05), St. Louis, 15-21 May 2005, pp. 631-632.

[16] J. Tuyal and J. Garcia-Fanju, "Effort Measurement in Student Software Engineering Projects," *Proceedings of the* 30*th ASEE/IEEE Frontiers in Education Conference*, Kansas City, 18-21 October 2000, pp. F1A-3-F1A-6.

[17] T. Reichlmayr, "The Agile Approach in an Undergraduate Software Engineering Course Project," *Proceedings of the* 33*rd ASEE/IEEE Frontiers in Education Conference*, Boulder, 5-8 November 2003, pp. S2C13-S2C18.

[18] J. Hayes, "Energizing Software Engineering Education through Real-World Projects as Experimental Studies," *Proceedings of the IEEE* 15*th Conference on Software Engineering Education and Training*, (*CSEET*.02), Covington, 25-27 February 2002, pp. 192-206.

[19] P. Robillard, "Teaching Software Engineering through a Project-Oriented Course," *Proceedings of the IEEE* 9*th Conference on Software Engineering Education* (*CSEE*), Daytona Beach, 22-24 April 1996, p. 85.

[20] S. Ludi and J. Collofello, "An Analysis of the Gap between the Knowledge and Skills Learned in Academic Software Engineering Course Projects and Those Required in Real Projects," *Proceedings of the* 31*st ASEE/IEEE Frontiers in Education Conference*, Reno, 10-13 October 2001, pp. T2D-8-T2D-11.

[21] P. Bucci, T. Long and B. Weide, "Do We Really Teach Abstraction?" *Proceedings of the ACM* 32*nd SIGCSE Technical Symposium on Computer Science Education*, Charlotte, 21-25 February 2001, pp. 26-30.

[22] I. Hadar and E. Hadar, "Iterative Cycle for Teaching Object Oriented Concepts: From Abstract Thinking to Specific Language Implementation," *Proceedings of the* 10*th Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts*, Nantes, 3-7 July 2006.

[23] H. Wei and L. Yao, "Research in Visualization Teaching Method of Program Design," *Proceedings of the IEEE* 2*nd International Conference on Communication Systems, Networks and Applications*, Hong Kong, 29 June-1 July 2010, pp. 180-183.

[24] P. Clarke, Y. Wu, A. Allen and T. King, "Experiences of Teaching Model-Driven Engineering in a Software Design Course," *Proceedings of the ACM/IEEE* 12*th International Conference on Model Driven Engineering Languages and Systems: Educators' Symposium*, Denver, 4-9 October 2009.

[25] O. Mishali, Y. Dubinsky and I. Maman, "Towards IDE Support for Abstract Thinking," *Proceedings of the ACM* 2*nd International Workshop on the Role of Abstraction in Software Engineering*, Leipzig, 11 May 2008, pp. 9-13. http://dx.doi.org/10.1145/1370164.1370167

[26] D. Zhenrong, H. Wenming, D. Rongsheng and W. Peizhi, "Exploration of Ability Development of Engineering and Computational Thinking Skills in Software Engineering Majors," *Proceedings of the IEEE* 4*th International Conference on Computer Science & Education*, Nanning, 25-28 July 2009, pp. 1665-1668.

[27] C. Starrett, "Teaching UML Modeling before Programming at the High School Level," *Proceedings of the IEEE* 7*th International Conference on Advanced Learning Technologies*, Niigata, 18-20 July 2007, pp. 713-714.

[28] O. Hazzan and J. Tomayko, "Reflection and Abstraction in Learning Software Engineering's Human Aspects," *IEEE Computer Magazine*, No. 6, 2005, pp. 39-45.

[29] G. Clarke and D. Cooke, "A Basic Course in Statistics," Arnold, London, 1998.

Appendix

Study Questionnaire

Dear SwE Student

This is a questionnaire to take your opinion in the knowledge and skills that you have learned in the software design and software modeling courses to improve your domain-specific abstract thinking. This questionnaire is part of an academic study for the research purposes, and doesn't aim to any trade purposes. The goal of this study is to evaluate whether the approaches that have been applied to your classroom, laboratory, as well as the course project when studying these two courses have provided you with the knowledge and skills needed to improve your domain-specific abstract thinking. Your opinion is very important in this study, so please be precise when selecting the answer in the following questions. I thank you so much for the time that you have given to fill this questionnaire, and appreciate your cooperation in supporting the scientific research.

My best wishes to you in a delight future.

Researcher

No	Statement
S1	Using simple-machine to describe the problem domain simplifies problem understanding
S2	Using diagrams as modeling and design tool helps in modeling and designing the big picture of the system
S3	Dealing with real world problem in the course project improves my conceptual thinking about the problem solution
S4	The course project improves the way I think about a problem domain
S5	Teamwork and brainstorming with my colleagues makes me think abstractly about the system.
S6	Learning and applying modeling and design makes me focused on the big picture of the system without thinking in the details
S7	Thinking abstractly contributes to modeling the big picture of the problem domain.
S8	Focusing on the concepts of the system under development contributes in understanding the system big picture.
S9	The availability of many modeling and design techniques contributes in improving my ability in thinking and comparing abstractly.
S10	The availability of many modeling and design techniques confuses me in thinking and comparing abstractly.
S11	Learning modeling and design techniques improves my comprehension of modeling and design concepts such as generalization, decomposition, abstraction, projection/views, and explicitness.
S12	Learning modeling and design makes me able to separate component from connectors.
S13	Using analogy is a good tool to improve my abstract thinking skill

