Software Engineering Education Toolkit for Embedded Software Architecture Design Methodology Using Robotic Systems*

Dongsun Kim, Suntae Kim, Seokhwan Kim, and Sooyong Park
Department of Computer Science and Engineering, Sogang University,
Shinsoo-dong, Mapo-Gu, Seoul, Korea
darkrsw@sogang.ac.kr, jipsin08@sogang.ac.kr, goatkhan@gmail.com, sypark@sogang.ac.kr

Abstract

Recently, industries need more effective software engineering education for undergraduate students as software plays an increasingly important role in consumer products. Specifically, the manufacturing industry emphasizes overall experience with software development processes from requirements to implementation in embedded software development. This paper proposes an educational toolkit focusing on ***architecture design methodology for embedded software*** *and reports experience with teaching software engineering by using the toolkit. The toolkit has several tools that support methodology education. The toolkit consists of three perspectives: people, process, and technology. Each perspective represents a set of tools which can support educational activities. Particularly, the toolkit introduces LEGO MindStorms NXT as a robotic system to provide experiences with embedded software development, and visible and tangible course materials. We have conducted a case study based on the toolkit in undergraduate-level classes. The case study shows the toolkit can be successfully applied in undergraduate-level software engineering education.*

1. Introduction

Since software has a key role in not only enterprise systems but also consumer products including cellular phones, robotic systems, and televisions, the manufacturing industry increasingly realizes the importance of software engineering. Hence, companies are eager to hire engineers who are educated in software engineering. Also, they request more systematic and practical education programs for software engineers to educators in academies, especially in undergraduate-level education. In addition, as consumer products have more complex software systems, they need an educational program which teaches more specific knowledge which includes theoretical bases and practical experiences for embedded software development because embedded software development is very different from regular software development[7].

The current software engineering education, however, does not satisfy industrial needs. In addition to experiences on embedded software, the industry also requires actual experiences with the entire software life cycle including requirements analysis, architecture, detailed design, implementation, testing, and maintenance. However, teaching all those topics requires long time and high cost. Since this paper does not discuss a curriculum which covers two or three year courses design, we consider teaching architecture construction and implementation of embedded software. These activities can be covered by one semester course and can satisfy key requirements from industries. Hence, this paper focuses on teaching **architectural design methodology** for embedded software which covers requirements analysis, architecture, detailed design and implementation because those are a set of key activities in software development and most software engineering courses concentrate to teach these activities.

Teaching a software design method requires additional course materials. First, a term project must be introduced to lead the course because consistent and consecutive examples in the same context are needed to effectively teach a design method. The project must deal with quasi-realistic software development, especially about embedded software. The second necessary material is someone who can guide students not to seriously mislead the project or misunderstand the design method. This is important because consecutive activ-

*This paper was performed for the Intelligent Robotics Development Program, one of the 21st Century Frontier R&D Programs funded by the Ministry of Commerce, Industry and Energy of Korea.

1530-1362/08 $25.00 © 2008 IEEE
DOI 10.1109/APSEC.2008.58

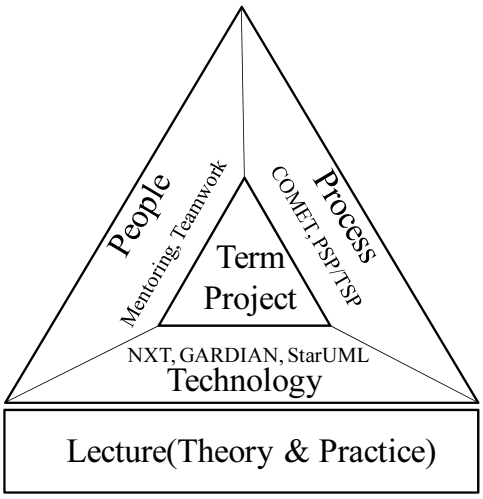

Figure 1. The Proposed Software Engineering Education Toolkit

ities in a design method are very relevant and wrong decisions in the early stage may lead to catastrophic results. Another one is teamwork in which students can discuss and debate their design decisions in software development. It is necessary to offer an opportunity to experience teamwork in software engineering courses because software development based on design methodology needs coordination of various stakeholders and cooperation of diverse software developers. These additional materials can support to teach a software design method.

In order to support software engineering education that satisfies the above requirements and constraints, this paper proposes an educational toolkit which consists of five materials as shown in Figure 1. Basically, lectures are provided to the students as a foundation. On this foundation, three types of tools are provided to support practical activities of the students. The 'People' tools are mentoring which prevents the students from misunderstanding, and teamwork which provides experiences of cooperation. The 'Process' tools are COMET(Concurrent Object Modeling and architectural design mEThod)[3] as software design methodology for embedded software which is the main focus of the toolkit and PSP/TSP which provides experiences of systematic software development. The 'Technology' tools are LEGO MindStorms NXT[8] which provides experience with developing embedded software systems, GARDIAN(see Section 2.2) which helps students identify defects in design models, and StarUML which is a modeling tool. A term project organizes lectures and tools altogether by connecting theory and practice. Using this toolkit, we expect that instructors who have difficulties in software engineering education can teach their students more effectively and students can learn software engineering more actively.

The rest of the paper is organized as follows: Section 2 presents the key aspects of the toolkit design. Section 3 briefly describes a case in which we applied the toolkit to a course in the spring of 2007. Section 4 presents the lessons learned throughout developing and applying the toolkit. The paper concludes with overviews of future work.

2. Toolkit Design

The major perspectives of software development are people, technology and process[4]. These perspectives for software development can be naturally applied into software engineering education. Our toolkit for software engineering education accordingly consists of three major perspectives:

- *People* - Mentoring, Teamwork
- *Technology* - GARDIAN, StarUML, H/W System(LEGO MindStorms NXT)
- *Process* - COMET, PSP/TSP

The people perspective helps students avoid catastrophic decisions and learn how to cooperate with stakeholders. The technology perspective supports them to produce software models and implementations and provides development environments. The process perspective guide students through a series of recommended activities to successful software development. Also, the toolkit contains a term project. In the software engineering course driven by the toolkit, the project is the main driver which orchestrates the three perspectives. Through the project students can experience software development activities sequentially and systematically. Lectures which covers theory and practice of software engineering teach the content of software engineering activities and how to use the toolkit. The rest of this section illustrates the toolkit in detail and provides rationale why the toolkit is composed by those tools shown in Figure 1.

2.1. People

Mentoring[13] is one of the proven methods to effectively guide a less experienced person in a new field.

The main goal of mentoring in the course is to prevent the students from being confused during the project but without giving them solutions on the project. Previous courses[2, 1] in other universities adopted technical staffs or developers from the software industry. Those people can be effective because they guide the activities of design methodology the students follow and give experience from actual practices in industries. However, it is hard to find multiple numbers of volunteer mentors with expertise who can sacrifice their time for students. *Hiring* mentors is, also, not feasible because it costs more than the course can pay for classes. Hence, we decide that adopting experts from the software industry is not an option in this toolkit.

Another option is that graduate students support the (undergraduate) students in the course. Although they have not much knowledge and experience than experts from industries, they have theoretical knowledge more than basic software engineering knowledge and practical experience in software development from research projects, and can support undergraduate students with low cost. Hence, in this toolkit, graduate students who have software engineering knowledge take part in the course as a mentor.

Mentors are divided into two groups: several normal mentors and a super-mentor. A super-mentor is a mentor of mentors. The super-mentor is responsible for leading other normal mentors and establishing the direction of mentoring. The super-mentor educates the normal mentors several times for the course. Normal mentors are assigned into a team composed of undergraduate students who take the course, and guide the team. Normal mentors play a role who directly interacts with their assigned team, and responds to their questions on the project. By means of the mentoring system, undergraduate students resolve their questions on the project. Additionally, the mentors deliver diverse experiences in practical software development to them via the project.

Teamwork becomes one of the essential factors for developing the large-scale software. The project in software engineering education needs to be conducted by teams to meet the trend of software development. Each team consists of 4∼6 team members and they elect one of them as team leader. The role of the team leader is to orchestrate team members and communicate with their mentor. In addition to the team leader, there is a GARDIAN(see Section 2.2) tool operator. The operator have a role to evaluate the UML model of the team. The rest team members of the team take responsibilities based on work assignment the team leader does.

Teamwork is not a physical tool which can support development activities and a person who can directly help students. Rather than a physical tool, it is a virtual tool which can support effective execution of design activities based on collaboration of team members. To teach teamwork in the course, lecturers should teach work breakdown, assignment, and planning which are not specifically explained in design methodology but those are crucial knowledge for effective software development with design methodology.

2.2. Technology

LEGO MindStorms NXT[8] is adopted in order to tackle the invisibility of software and stimulate the students' interest in software engineering. LEGO MindStorms NXT is a programmable robotics kit released by LEGO in July 2006. The kit consists of 519 technic pieces, three servo motors, four sensors (ultrasonic, sound, touch, light), seven wires, a USB cable, and the NXT brick (a control unit).

The reasons why NXT is chosen as a hardware environment are low cost and ease of use. There are several alternatives for the hardware environment such LEGO MindStroms RCX[9], Sony AIBO[15], and virtual robot simulators (e.g. Karel[12]). In terms of cost, AIBO, RCX, and NXT have reasonable price (from \$300∼\$600), but Sony declared they discontinue the sales of AIBO and LEGO is focusing on supporting NXT rather than RCX because RCX is a previous version of NXT and NXT is the latest version of LEGO robot platforms. Robot simulators are extremely cheaper than other hardware-based platforms, but they are not tangible and provide only limited characteristics of embedded systems development.

On the other hand, NXT is not expensive (around \$300 per package), has various functionalities (rotational motors, several sensors). Also, it is easy to control its actuators and sensors because basically NXT provides a bundle control program based on LabView[10] which is a simple and visual-based programming environment and there are several third party control drivers in various programming languages such as Java, C++, and C# which support easy control for actuators and sensors. Another advantage of NXT is that it can be assembled in various form. Since it provides 519 technic pieces, sensors, and actuators, students can make thousands of robot models based on NXT. This fact is very important because it can prevent cheat by memory dumps of previous courses using the same toolkit. Based on these advantages, the toolkit adopts NXT as a hardware platform.

Selecting an appropriate UML modeling tool is one of the essential factors for a successful project. Likewise, providing an appropriate UML modeling tool to

students in the course is crucial. Requirements for the modeling tool are low price (or freeware), UML profile support, Korean language support, and easy documentation. Several commercial modeling tools[14, 17] support various and custom UML profiles, multi-language (especially asian languages), and rich documentation, but those are too expensive to be adopted in university courses. There are educational versions of the commercial tools which is less expensive than general versions, but those are still expensive for universities to pay charge every year. There are non-commercial modeling tools[11, 18] which are freeware, but some of them cannot support custom UML profiles, Korean language, or easy documentation. Hence, StarUML[16] is selected as a modeling tool in this toolkit since its manipulation is easy and it is also freeware. StarUML provides an approach profile which can define the steps and diagrams of COMET and support Korean language.

COMET[3] provides design guidelines to identify and refine objects of the software system by using various UML diagrams and specifies rules in modeling software systems. Those rules represents how objects and devices have to interact with each other. For example, a message between a sensor device and a controller should be passed from the sensor device to the controller. The toolkit provides a rule checker called GARDIAN which can automatically detect syntactic and semantic anomalies in design models. GARDIAN is implemented based on XML Metadata Interchange (XMI) to decipher model files generated by StarUML. GARDIAN helps the students to check whether their model observes the guidelines of COMET. GARDIAN supports rule syntax for the experts of the design method to specify the knowledge on the design guidelines, and the rule engine to evaluate designers' model according to the pre-described knowledge as depicted in Figure 2. It generates the guideline violation reports which describe which model elements violated the design guideline and the reason why it was violated. Students can modify their model based on the guideline violation report, and decrease the number of defects in it.

2.3. Process

Since COMET provides richer and more specific guidelines to design real-time, embedded and distributed applications than other general purpose software design methods and development process models such as waterfall model, spiral model, Open Unified Process, and Rational Unified Process this method is adopted in the toolkit. The goal of the project is to develop a robot software system which controls several sensors, and actuators in NXT hardware. The method

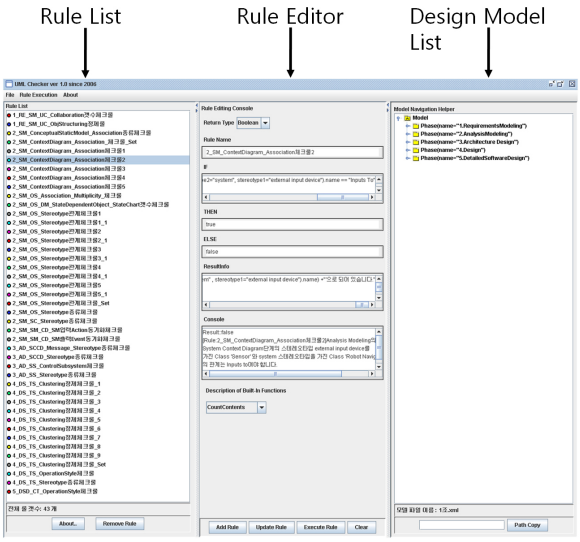

Figure 2. A Snapshot of GARDIAN

is partially modified after due consideration of the characteristics of the target software system. There exists diverse steps including use case modeling, analysis modeling, architectural design, task structuring, and detailed design in COMET. Among those steps the architectural design is skipped because the target system of the project[1] has one subsystem. The project more focuses on requirements engineering, analysis modeling, and task structuring instead of the architectural design.

In the use case modeling step, students elicit fundamental requirements of the software system in the embedded system. The analysis modeling step has two sub steps: static modeling and dynamic modeling. Static modeling focuses on the relationship of external I/O devices with the software system, and entities and objects in the system. Dynamic modeling focuses on state transitions of the system from the initial state to the final state and sequential interactions between objects identified in static modeling. In the task structuring step, students must identify tasks which are concurrent execution units in the system. In the detailed design step, students must revise and elaborate the previous models in order to implement actual embedded software.

Additionally, the data of PSP/TSP[5, 6] is compiled to measure the time they spent for the project. PSP and TSP is a quantified method aimed to the improvement of the quality and productivity of personal and

[1]The problem description used in the toolkit can be downloaded from 'http://seapp.sogang.ac.kr:8080/RPS_Problem_Statement.pdf'

team work. It requires students' extra-effort to collect the data. However, this technique is one of the most useful and easiest techniques for which they can measure the time they spent doing something. Based on the collected data, they can elaborate their previous work assignment and replan consecutive activities more efficiently. Thus, this can help students manage their time to execute design activities of COMET and support to achieve a successful project.

3. Case Study

This section describes a case study performed as an undergraduate course in the spring of 2007. The section focuses on the actual application of the proposed toolkit described in Section 2 and the effectiveness of the toolkit in the course. The course was thirteen weeks except for mid and final term exams. Each week had two classes and each class was an hour and half hour. Forty six students took the course and most of them were juniors or seniors who are majoring in computer science and engineering. The students organized teams of 4∼6 members before midterm exam. Eight teams were organized and each team elected one of the team members as a leader. The leader was responsible for communicating with a mentor, guiding team members, managing team schedules, and so on.

The lectures consisted of an introduction to software engineering, team organizing(1∼4 week), steps of COMET, RPS H/W assembly(by using LEGO MindStorms NXT), programming on NXT, PSP/TSP usage, GARDIAN framework usage and StarUML(5∼13 week), and presentations about the outcomes of students(14∼15 week). In weeks 5∼13, one class a week dealt with theoretical aspects of COMET, and another class dealt with practice time using COMET and the usage of tools and NXT. Classes through weeks 5∼13 were bases of the term project. The term project started just after the midterm exam.

The requirements description(problem statement) of the term project was provided to the students(it can be downloaded from 'http://seapp.sogang.ac.kr:8080/RPS_Problem _Statement.pdf'). The target system of the project was RPS(Robot Patrol System) which patrols between two locations. If an intruder is detected, then the system gives the alarm. When the intruder disappears or is removed, the system turns off the alarm and patrols again. The students assembled the hardware structure of RPS by using NXT blocks. To help assembly, we provided a movie clip which showed a assembly process.

The students modeled the software system of RPS using COMET. They used StarUML as a modeling tool. For each activity, they drew diagrams and wrote documents. They used GARDIAN to check out their models whether the models observed the guidelines of COMET. They submitted the outcomes at the end of every activity. Based on the outcomes(diagrams and documents), they implemented the software system of RPS. Communication between NXTs and desktop PCs was established by Bluetooth. We provided device drivers to control sensors and actuators of NXT through Bluetooth communication in C# language. By using these drivers, they could easily implement the software system reflecting models they documented. The final deliverables they produced were diagrams(including a use case diagram, a conceptual static model, a system context class diagram, an object structure, a state diagram, a collaboration diagram, a subsystem collaboration diagram, a subsystem concurrent collaboration diagram, a task structure, and a task clustering diagram), documents describing diagrams, and source code files of RPS.

Mentoring supported teams to perform the term project correctly. Mentors were graduate students majoring in software engineering at Sogang University. One mentor advised one team. The roles of mentors are listed as follows: 1. Maintain students enthusiasim throughout the course, 2. Infuse desires into all team members to participate actively in their project, 3. Help students quench their curiosity or solve difficult parts in the progression of the project. However, do not bring up a solution, 4. Act as a guide for them to solve problems themselves, 5. Give advice for troubles or worries. Mentors had project meetings for a minimum of once a week with their teams. During mentor meeting(of mentors and the super-mentor) every week, mentors reported the situations of their team to the super-mentor. Through mentor meetings future plans were also discussed.

The students recorded their effort for each step in the project by using the given PSP/TSP sheet. Every team member recorded his/her efforts on a personal sheet. The team leader aggregated the personal sheets of the team and recorded them in a team sheet, and reported it to the super-mentor. Some teams exploited the results to anticipate the efforts of the next stage but some other teams did not at all.

The project was evaluated based on criteria of 70% of their outcomes and 30% of their presentation. Every team presented their diagrams, documents, and their system. Each team had fifteen minutes for the presentation. Also all mentors participated in the presentations and gave comments about the overall evaluation of the team. After the presentations we selected three

outstanding teams in order to encourage participation.

4. Lessons Learned

After the course described in Section 3, we had interviews with the students to verify the effectiveness of the toolkit in the course. The interviews consisted of questions about the effectiveness of the lectures, the term project, the tools, and overall opinions of the course. The interviews were conducted for each team around thirty minutes. Before the interviews, we gave them survey sheets and let them discuss the effectiveness of the course for a week and record their opinions on the sheets. The sheet has twelve questions and we asked them to record scores based on five-level Likert scale (1: Strongly disagree, 2: Disagree, 3: Neither agree nor disagree, 4: Agree, 5: Strongly agree). The result of the survey sheets is shown in Table 1 and 2[2]. Then, we had a discussion with them based on the sheet. The rest of this section qualitatively discusses the effectiveness of the course and toolkit based on quantitative results of the sheet

Question	Average
1. How well do you understand Use Case Modeling?	3.38
2. How well can you apply Use Case Modeling in practice?	4.13
3. How well do you understand Static Modeling?	3.63
4. How well can you apply Static Modeling in practice?	4.00
5. How well do you understand Dynamic Modeling?	4.00
6. How well can you apply Dynamic Modeling in practice?	4.00
7. How well do you understand Task Structuring?	3.25
8. How well can you apply Task Structuring in practice?	3.25

Table 1. Survey results #1

The survey sheet on lectures comprised questions about both the lectures on software engineering theories (Table 1) and those of practice (Table 2). The students said that the lectures about theories, especially about COMET, were great because those showed the overall software development process at a glance. In other words, they didn't have a chance to develop complete software systems that included activities from requirements analysis to maintenance but only had a chance to implement small pieces of programs that contain functions of specific technologies such as artificial intelligence and operating systems. Hence, the lectures on methodology were impressive and gave ideas for software development.

We asked about how effective COMET is for the project. The students said it was good but some problems for each step must be addressed. They discussed that the use cases modeling helped them know the outline and boundary of the system, and set the start line of the project based on scenarios. However, some of them said that there was not enough explanation of use cases and actors so they had some difficulties in modeling use cases and actors. As shown in the average score of question #1 in Table 1, the students thought use case modeling was hard to understand but, positively, the average score of question #2 indicated they thought they could apply what they had learned.

The students said that static modeling supported the idea that they could have a more specific structural view of the system than use cases. However, some students said that it was hard to identify objects and system structures only based on requirements and use cases without actual executable code because they were used to implementing software directly by a programming source code. Also some teams complained that they didn't know the inter-relationship between use cases and static models. In spite of complaints, they gave more positive scores than the scores of use case modeling because static modeling was more familiar and was more close to coding.

In dynamic modeling, the students could understand overall behavior and data flows of the system. The average scores of question #5 and #6 reflect their positive opinions. However, they stated that the collaboration diagram of the system was complex and unreadable even though there were only a few objects in the diagrams. Some collaboration diagrams of some teams were very hard to traverse and modify because of complexity. This fact can be a reason for inventing a more efficient model of software behavior.

They said that task modeling helped them identify concurrent behavior and structures of the systems so that they could perform a detailed design more easily. On the other hand, a few teams had problems with task structuring. For example, they first implemented an executable partial code for the systems. Then, they observed the code and extracted tasks reversely. This was not the way the course was intended to go. The reason why question #7 and #8 had the lowest scores of other questions was the course did not have enough

[2]In each question, we represent only the average score of eight teams due to space limitation.

Question	Average
9. How effective does the use of LEGO MindStorms NXT encourage you in focusing on the course in terms of attractiveness, visibility, and tangibility?	4.25
10. How well does teamwork support the course?	4.00
11. How well do PSP/TSP sheets support your time management?	3.25
12. How effective do mentors support your project?	3.5

Table 2. Survey results #2

time to teach task structuring so they could not understand enough.

Students' overall evaluation for COMET was good, because they could experience software development from requirements to implementation(or maintenance for some teams). Various perspectives of software modeling prevented them from misdirecting the project in a wrong way such as spaghetti code which is not readable, not modifiable, and complicated. Also the process taught them to have insights into systematic software development. In spite of the positive opinions, there were problems such as we could not invest enough time to teach unfamiliar concepts as mentioned above and could not make more examples which could help them make better models. To improve, we need to organize the lectures more compactly and precisely to be able to teach within a semester. We also need to make more examples that provide various characteristics of embedded software.

The survey on the toolkit focused on how effective it was and what was the problem. The students stated LEGO MindStorms NXT was remarkable because it made the course possible to experience how embedded software development proceed and to see how the system that they developed interacted with H/W and operating environments. Specifically, it was impressive that software systems are not affected simply by computing resources or algorithms, but also by factors in real world such as a gradient, friction, and radio frequency and that the course showed how they can analyze and resolve it. Also it helped to understand the characteristics of embedded software such as timeliness, concurrency, liveness, and reactivity. Overall positive opinion of the students was shown in the result of question #9 in Table 2.

The students said teamwork was basically good as shown in question #10 in Table 2, but also there was a problem. They said they could learn how to communicate with team members, how to break down work, and how to integrate the results by teamwork. They said teamwork was helpful to learn how to solve complex problems in software development by cooperation and it would work in companies after graduation. However, it was hard for them to divide and distribute work and to integrate the results without any guidance because we just helped to organize teams and gave only a few guidelines for teamwork. They usually developed software alone so dividing and assigning work to an appropriate person was not easy for them.

Collecting data of PSP and TSP sheets helped the students measure and predict their effort themselves quantitatively. They were skeptical about collecting data and complained that recoding and tracking PSP/TSP data was a burden at the beginning. However, gradually, they made use of the collected data to anticipate their workload and divide it up into the next stages. Also, they requested more specific usage of collected data from us. Another complaint in collecting data was tool support. We just provided a recoding sheet made by Microsoft Excel so they had problems in collecting data precisely and sometimes they forgot time duration or recording itself. This problem was reflected in the score of question #11. Hence, we need to introduce tools for precise recording and teach specific usage of PSP/TSP.

Mentoring played a very important role for the students to perform the project. Even some students stated that they could not perform the project without their mentor. Although mentors did not give answers, they objectively evaluated the outcomes of the students and prevented them from misunderstanding theories and misleading the project. Because faults in the early stage of software development may propagate to the next stages, if the faults are found in th outcomes of the students, then the project may fail due to the faults. The failure might depress the students and this may lead to misunderstandings that they might think software engineering is not important or effective even though they misunderstood using theories properly. Fortunately, with mentors, they could have the right conception of software engineering.

On the other hand, mentoring had a set of problems such as the students complained about knowledge differences between mentors. This led to unfair help and it even might have an affect on students' grades. The reason why the situation happened was there was not enough communication between mentors, and between mentors and the supermentor. Another problem was mentors did not spend equal time with their team. Hence, they didn't give many high scores to the men-

toring as shown in question #12 in Table 2. To provide better mentoring, we need to improve and enhance the education of mentors, to have more frequent mentor meetings, and to enforce more time slots for mentor meetings in the course schedule.

Applying the StarUML tool and the GARDIAN positively supported this project process. StarUML as open software can be downloaded and installed without cost. Thus, it was an appropriate choice for undergraduate education. GARDIAN has a set of syntactical and semantic rules for checking common mistakes the student can make. Thus, GARDIAN contributed to improving the quality of their models without being time consuming. Some students who used GARDIAN said they could not make high quality models without GARDIAN because COMET has a lot of semantic and syntactical rules compared with general UML models.

5. Conclusions

This paper has listed the requirements of software engineering education from the industry. Based on the requirements, this paper has presented the software engineering education toolkit which focuses on teaching architecture design methodology for embedded software. The described toolkit consists of lectures, a term project, and three types of tools. Those tools support students to learn software design activities in various perspectives. We have conducted a case study to evaluate the toolkit in the context of an undergraduate class. The case study has shown the effectiveness of the toolkit in terms of industrial requirements within time and resource constraints. At the same time, from the case study we have recognized a number of issues which need improvement. Our future work will span issues such as improving mentor education, organizing lecture schedules more efficiently, expanding to other programming languages, and emphasizing use of PSP/TSP sheets.

References

[1] D. Garlan, D. P. Gluch, J. E. Tomayko, Agents of change: Educating software engineering leaders, IEEE Computer 30 (11) (1997) 59–65.

[2] N. E. Gibbs, The sei education program: the challenge of teaching future software engineers, Commun. ACM 32 (5) (1989) 594–605.

[3] H. Gomaa, Designing Concurrent, Distributed, and Real-Time Applications with UML, Addison-Wesley, 2000.

[4] T. B. Hilburn, Software engineering education: A modest proposal, IEEE Software 14 (6) (1997) 44–48.

[5] W. S. Humphrey, Introduction to the Team Software Process, Addison-Wesley, 2000.

[6] W. S. Humphrey, PSP: A Self-Improvement Process for Software Engineers, Addison-Wesley, 2005.

[7] E. A. Lee, Embedded software, Advances in Computers 56 (2002) 56–97.

[8] Lego MindStorms NXT, http://mindstorms.lego.com/ (2007).

[9] Lego MindStorms RCX, http://www.lego.com/eng/education/mindstorms/home.asp?pagename=rcx (2008).

[10] NI LabView, http://www.ni.com/labview/ (2008).

[11] Papyrus UML, http://www.papyrusuml.org/scripts/home/publigen/content/templates/show.asp?l=en&p=55&vticker=alleza&itemid=3 (2008).

[12] R. E. Pattis, Karel the Robot: A Gentle Introduction to the Art of Programming, John Wiley & Sons, Inc., New York, 1994.

[13] R. Ramaswamy, Mentoring object-oriented projects, IEEE Software 18 (3) (2001) 36–40.

[14] Rational Software Architect, http://www-306.ibm.com/software/awdtools/architect/swarchitect/ (2008).

[15] Sony AIBO, http://support.sony-europe.com/aibo/index.asp (2006).

[16] StarUML, http://staruml.sourceforge.net/en/ (2007).

[17] Together, http://www.borland.com/us/products/together/index.html (2008).

[18] Violet UML Editor, http://alexdp.free.fr/violetumleditor/page.php?id=en:tour (2008).