Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2011, Article ID 920873, 12 pages
doi:10.1155/2011/920873

Research Article

Using Game Development to Teach Software Architecture

Alf Inge Wang and Bian Wu

Norwegian University of Science and Technology, Sem Sælandsv. 7–9, 7491 Trondheim, Norway

Correspondence should be addressed to Alf Inge Wang, alfw@idi.ntnu.no

Received 12 August 2010; Revised 14 April 2011; Accepted 2 May 2011

Academic Editor: Jihad El-Sana

This paper describes a case study of how a game project using the XNA Game Studio from Microsoft was implemented in a software architecture course. In this project, university students have to construct and design a type of software architecture, evaluate the architecture, implement an application based on the architecture, and test this implementation. In previous years, the domain of the software architecture project has been a robot controller for navigating a maze. *Robot controller* was chosen as the domain for the project, as there exist several papers and descriptions on reference architectures for managing mobile robots. This paper describes the changes we had to make to introduce an XNA game development project to the software architecture course, and our experiences from running a software architecture project focusing on game development and XNA. The experiences described in this paper are based on feedback from the course staff, the project reports of the students, and a mandatory course evaluation. The evaluation shows among other things that the majority of the students preferred the game project to the robot project, that XNA was considered to be suitable platform for a software architecture project, that the students found it useful to learn XNA and C#, and that some students were carried away when developing the game in the software architecture project.

1. Introduction

Games have been used in education for many years mainly focusing on teaching children in an interesting and motivating way. Research shows that integrating games within children's classroom can be beneficial for academic achievement, motivation, and classroom dynamics [1]. Teaching methods based on educational games are not only attractive to schoolchildren, but can also be beneficial for university students [2]. Research on game concepts and game development used in higher education is not unique, for example [3–5], but we believe there is an untapped potential that needs to be explored. By introducing games in higher education lecturers can access teaching aids that promote active students, provide alternative teaching methods to improve variation, enable social learning through multiplayer learning games, and motivate students to work harder on projects and exercises.

Games can mainly be integrated in higher education in three ways. *First,* traditional exercises can be replaced by games motivating the students to put extra effort in doing the exercises, and giving the course staff an opportunity to monitor how the students work with the exercises in real-time [6, 7]. *Second*, games can be used within a traditional classroom lecture to improve the participation and motivation of the students through knowledge-based multiplayer games played by the students and the teacher [8, 9]. *Third*, game development projects can be used in computer science (CS) or software engineering (SE) courses to learn specific CS or SE skills [10, 11]. This paper focuses on the latter, where a game development project was introduced in a course to teach CS and/or SE skills. The motivation for bringing game development into a CS or SE course is to utilize the students' fascination for games and game development to stimulate the students to put extra effort in the course project. Many students dream of making their own games, and game development projects allow the students to use their creativity in contrast to, for example developing a more traditional web-based application. Game technologies and game user interfaces are now being more commonly used in serious applications [12–14], and the market for serious games is growing. This makes it important for students to learn how to develop games even the students do not target to work in the game industry.

In this paper, we describe a case study of how a game project was integrated with a software architecture course. From the perspective of a game developer, knowledge and skills about how to develop appropriate software architectures are becoming increasingly important. As games are growing bigger and becoming more complex, well-designed software architectures are needed to cope with variations in hardware configurations, functional modifications, and network real-time constraints [15]. From the perspective of a software architect, games are interesting due to the inherent characteristics of the domain including real-time constraints, changing and varying functionality, and user-friendliness. In addition, games are interesting from the perspective of a software architect, as there exist no real functional requirements that stem from the users. Typical user requirements for games are that the game should be fun to play, it should have enough variety, and it should be engaging.

The case study presented in this paper describes how a software architecture course was adapted to include a game development project. The paper describes the parts of the course and syllabus that had to be changed to make game development a natural part of the course, and how XNA was used as a game development platform in the course. Further, we present an evaluation of how the game development project was perceived by the students and the course staff compared to the robot project. The data of this evaluation is based on the students' responses to the final course evaluation, the feedback from the students during the project, and the student project reports.

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 describes the software architecture course. Section 4 describes how the course was changed to adapt to the game project. Section 5 presents experiences we learned from running a game development project along with the robot development project in a software architecture course, and Section 6 concludes the paper.

2. Related Work

This paper describes experiences from introducing an XNA game development project in a software architecture course. The main benefits from using XNA to teach software architecture is that the students get more motivated during the software development project. As far as we know, there are only few papers (presented here) that describe usage of XNA to teach CS or SE, and only few papers that contain case studies of games used in CS and SE education (also described here). In this section, we will also briefly describe alternative game development frameworks to XNA that can be used in CS and SE education.

Youngblood describes how XNA game segments can be used to engage students in advanced CS education [16]. Game segments are developed solution packs providing the full code for a segment of a game with a clear element left for a student to implement. The paper describes how XNA was used in an artificial intelligence course where the students were asked to implement a chat bot, motion planning, adversarial search, neural networks, and flocking. Finally, the paper describes seven design principles for using game segments in CS education based on lessons learned. The approach described by Youngblood could also be used in a software architecture course, where the students can put together parts of the game (game segments) based on their designed architecture. However, this approach is very limiting as the architectural freedom will be very restricted and the students will not get the chance to design their own software architecture of their own game.

El-Nasr and Smith describe how modifying or modding existing games can be used to learn CS, mathematics, physics, and ascetic principles [10]. The paper describes how modding of the WarCraft III engine was used to teach high school students a class on game design and programming. Further, they describe experiences from teaching university students a more advanced class on game design and programming using the Unreal Tournament 2003 engine. Finally, they present observations from student projects that involve modding of game engines. Although the paper claims to teach students other things than pure game design and programming, the focus is on game development in contrast to CS or SE. Modding existing games is not very useful in a software architecture course, as the focus of the course is the structure of software components and not game content nor game engine scripts.

Sweedyk and Keller describe how they have introduced game development in an introductory SE course [17]. The students learn principles, practices, and patterns in software development and design through three projects. In the *first* project, the students develop a campus life 2D arcade game over four weeks with the educational focus on gaining familiarity with UML tools, learn and use a variety of development tools and gain understanding of game architecture and the game loop. In the *second* project, the students should build a one-hole miniature golf game over five weeks with the educational focus on learning and practicing evolutionary design, prototyping and refactoring, usage of UML design tools, usage of work management tools, and design and implementation of a test plan. In the *third* and final project, the students can develop a game of their own choice over five weeks with educational focus on reinforcing the practices and principles learned in two previous projects, learn to apply design patterns, and practice management of complex software projects. The students' response to this SE course has according to the authors been extremely positive. They argue that game projects allow them to better achieve the learning objectives in the SE course. Their main concern is related to gender, as women are less motivated to learn SE through game development projects. The main difference with Sweedyk and Keller's approach and ours is that they have introduced three projects instead of one, and the SE focus is different. For our purpose, more than one project would take away the focus on the software architectural educational goals and miss the opportunity to follow the evolution of the software architecture through a complete development cycle.

K. Calypool and M. Calypool describe another SE course where a game development project was used to engage the students and make the course more fun [18]. In this course, the students worked with one game project where the students had to go through all the phases in a software development process. The preliminary results of comparing the game-based SE course with a traditional SE course showed that the game version had higher enrollment, resulted in average higher grades, a higher distribution of A grades, and had a lower number of dropouts. The feedback from the students was also very positive. The approach described in this paper is very similar to our approach. The main difference is that in our course the students carry out the various phases in a software process from a software architecture perspective focusing on quality attributes, software architecture design, and software architecture evaluation.

Volk describes how a game engineering course was integrated into a CS curriculum [19] motivated by the fact that game development projects are getting more and more complex and have to deal with complex CS and SE issues. The experiences from running this course showed that it was a good idea handle the game engineering course more in a form of a real project, that the students were very engaged in the course and the project, that the lack of multidisciplinary teams did not hinder the projects, that the transition from preproduction to production was difficult (extracting the requirements), and that some student teams were overambitious for what they wanted to achieve in their project. In our software architecture course, we experienced some of the same issues as described in this paper, namely difficult extraction of requirements and overambitious teams.

Linhoff and Settle describe a game development course where the XNA platform was used to allow the students gain experience in all aspects of console game creation [20]. The course focuses on creating of fonts, icons, 3D models, camera and object animation paths, skeletal animations, sounds, scripts, and other supporting content to the XBOX 360 game platform. In addition, the students are required to edit the source code of a game to change variables, and copy-and-paste code. The student response to the course was positive. The results also showed that students with programming background did better in the class. The students did not learn any CS or SE skills.

Zhu et al. describe how games can be introduced in SE courses to teach typical SE skills [21]. The paper describes how the two games SimSE and MO-SEProcess were used to give students an opportunity to practice SE through simulations to learn the complex cause and effect relationships underlying the process of SE. MO-SEProcess is a multiplayer online SE process game based on the SimSE in 3D implemented in Second Life. In this game, the players should collaborate with other developers to develop a system by giving out tasks and following up tasks. Although the models and simulations in SimSE are much more extensive than the ones in MO-SEProcess, the usage of Second Life bring some advantages such as better support for group sharing and collaboration, and the possibility to create interactive learning experiences that would be hard to duplicate in real life. This approach is very different from ours and does not fit with our educational goals.

Rankin et al. describe a study on how game design project impact on students' interest in CS [22]. In a Computer Science Survey course, the students are given the task to apply SE principles in the context of game design. The pre and post survey results reveals that game design project can have both a positive and a negative impact on students' attitudes about enrollment in a game design course, pursuit of a CS degree, further development of programming skills and enrollment in additional CS courses.

Leutenegger and Edgington argue that the course assignment and example content is more important than whether a introductory programming course should focus on procedural versus object-oriented approach [23]. Their paper describes an introductory programming course focusing on game programming. The results showed that the students improved their understanding basic programming concepts, and the students were satisfied with the course.

Coller and Scott describe an interesting approach for teaching mechanical engineering through game programming [24]. In a numerical methods course, the students are asked to program the behavior of a car in the Torcs open racing car simulator. The students must use numerical methods to program acceleration, steering, gearshifts, and breaking. A comparison with a traditional version of the course showed that for the game-based course the students on average spent roughly twice as much time on the course, and that the students achieved deeper learning as the students were more interested, more engaged, and invested more in learning the material.

We have found the XNA was a perfect fit for our game project as it provides a high-level API, the framework is mature and well supported, and the students are motivated by the fact that XNA makes it easy to develop for XBOX 360. There are also other alternative game frameworks that can be used. The *Labyrinth* [25] is implemented in Java and is a flexible and easy-to-use computer game framework. The framework enables instructors to expose students to very specific aspects of CS courses. The framework is a finished game in the Pac-Man genre, highly modular, and it lets the students change different aspects of the game. The *JIG (Java Instructional Gaming)* project [26] has the aims to build a Java Instructional Game Engine suitable for a wide variety of students at all levels in the curriculum, to create a set of educational resources to support the use of the game engine at small, resource-limited, schools, and to develop a community of educators that use and help improve these resources. The *DXFramework* [27] is a game engine written in C++ targeted specifically for 2D games to be used in game programming education. The *SAGE* [28] game engine is also written in C++ and is targeted for game programming educational use focusing on 3D games. *GEDI* [29] game engine is another alternative for 2D games in C++ designed with game programming educational use in mind. For business teaching, *Arena3D* [30] is a game visualization environment with animated 3D representations of the work environments, simulation of patients queuing at the front desk, and interacts with the staff. IBM has also produced a business game called *INNOV8* [31], which is "an interactive, 3D business simulator designed to teach the fundamentals of business process management and bridge the gap in understanding between business leaders and IT teams in an organization".

Of the related work described in this section, the work by Kajal and Calypool is closest to the work described in this paper. The main difference with our approach is that we focus on software architecture methods and processes and not only software engineering topics in general. The students' responses to our course are very similar to the studies described in this section, characterized by higher motivation, higher enrollment, and more effort spent on the course.

3. Software Architecture Course

The software architecture course is a postgraduate course offered to CS and SE students (not mandatory) at the Norwegian University of Science and Technology (NTNU). The course is taught every spring, its workload is 25% of one semester, and about 70–80 students attend the course every spring. The students in the course are mostly of Norwegian students (about 80%), but there are also 20% foreign students mostly from EU countries. There are about 10% female students. The textbook used in this course is the "Software Architecture in Practice, Second Edition," by Clements et al. [32]. Additional papers are used to cover topics that are not sufficiently covered by the book such as design patterns, software architecture documentation standards, view models, and postmortem analysis [33–37].

The education goal of the course is: "*the students should be able to define and explain central concepts in software architecture literature, and be able to use and describe design/architectural patterns, methods to design software architectures, methods/techniques to achieve software qualities, methods to document software architecture and methods to evaluate software architecture.*"

The course is taught in three main ways:

(1) ordinary lectures given in English,

(2) invited guest-lectures from the software industry,

(3) a software development project with emphasis on software architecture.

The software architecture course at NTNU (course code TDT4240) is taught in a different way than at most other universities, as the students also have to implement their designed architecture in a project. The motivation for doing so is to make the students understand the relationship between the architecture and the implementation, and to be able to perform a real evaluation of whether the architecture and the resulting implementation fulfill the quality requirements specified for the application. The architecture project in the course has similarities with projects in software engineering courses, but everything in the project is carried out from a software architecture perspective. Throughout the project, the students have to use software architecture techniques, methods, and tools to succeed according to the specified project requirements and the document templates. The development process in the project will also be affected by the focus on software architecture, as the development view of the architecture will specify how the teams should be organized and how they should work. The main disadvantage of this approach is that the students get less time dedicated to do the architectural design, as they have to spend time on the implementation. The main advantage is that the students are learning software architecture through doing a whole project where they can see the results of their architectural design as a product.

The TDT4240 software architecture course has been rated as one of the most useful and practical courses offered at the Deptartment of Computer and Information Science in surveys conducted among exstudents now working in the IT industry. The course staff has also seen the benefits of making the students implement the architecture, as the students have to be aware of the developing costs of fancy and complicated architectural designs.

30% of the grade awarded to the software architecture course relate to the evaluation of the software architecture project all students have to do, while 70% is awarded for the results of a written examination. The goal of the project is for the students to apply the methods and theory in the course to design and fully document a software architecture, to evaluate the architecture and the architectural approaches (tactics), to implement an application according to the architecture, to test the implementation related to the functional and quality requirements, and to evaluate how the architectural choices affected the quality of the application. The main emphasis when grading the projects is on the quality of the software architecture itself, but the implementation should also reflect the architecture and the architectural choices.

The project consists of the following phases.

(1) *Commercial Off-the-Shelf (COTS).* Learn the development platform/framework to be used in the project by developing some simple test applications.

(2) *Design Pattern.* Learn how to utilize design patterns by making changes in two architectural variants of an existing system designed with and without design patterns.

(3) *Requirements and Architecture.* Describe the functional and the quality requirements, describe the architectural drivers, and design and document the software architecture of the application in the project including several view points and views, stakeholders, stakeholder concerns, architectural rationale, and so forth.

(4) *Architecture Evaluation.* Use the Architecture Tradeoff Analysis Method (ATAM) [32, 38, 39] to evaluate the software architecture in regards to the quality requirements.

(5) *Implementation.* Do detailed design and implement the application based on the designed architecture and based on the results from the evaluation. Test the application against both functional and quality requirements specified in phase 3, evaluate how well the architecture helped to meet the requirements,

and evaluate the relationship between the software architecture and the implementation.

(6) *Project Evaluation.* Evaluate the project using a Post-Mortem Analysis (PMA) method [34]. In this phase, the students will elicit and analyze the successes and problems during the project.

In the two first phases of the project, the students work on their own or in pairs. For the phases 3–6, the students work in self-composed teams of four students. The students spend most time in the implementation phase (6 weeks), and they are also encouraged start the implementation in earlier phases to test their architectural choices (incremental development). During the implementation phase, the students continually extend, refine, and evolve the software architecture through several increments.

In previous years, the goal of the project has been to develop a robot controller for a robot simulator in Java with emphasis on an assigned quality attribute such as availability, performance, modifiability, or testability. The functional aim of this project was to develop a robot controller that moves a robot in a maze collecting balls and bringing them to a light source. Robot controller was chosen as a case for the software architecture project, as the problem of software architecture is well defined within this domain. For the robot controller domain there exist several examples of software architecture patterns or reference architectures that can be applied, such as Control loop [40], Elfes [41], Task Control [42], CODGER [43], Subsumption [44], and NASREM [45].

4. How the Course Was Changed?

This section presents the changes we made to the course to integrate an XNA game development project with the software architecture course.

4.1. Course Preparations. Half a year ago we integrated the game development project with the software architecture course, we initiated a master research project, named XQUEST, to explore how XNA could be used and integrated with the course. The goal of this project was to answer the following questions.

(Q1) How well is the XNA framework suited for teaching students software architecture?

(Q2) What resources must be in place to quickly get up to speed developing games using the XNA framework?

(Q3) How should XNA be introduced to the students?

The first question (Q1) was decomposed into three sub-questions. *First*, the XQUEST project investigated which software/game components were required to allow the students to stay focused on the software architecture during the their project. This work resulted in an implementation of a game library named XQUEST framework [46] to provide a high-level sprite animation framework, a game object management framework, and some additional helper classes (audio, input, text out, and texture store) on top of XNA to ease the development. *Second*, the XQUEST project investigated how difficult it was for the students that only knew Java to learn the C# programming language. They found that it took about three days to learn the most essential features of C# for a postgraduate student with average Java skills. *Third*, the XQUEST project investigated what limitations or restrictions that should be put on a game development project in a software architecture course. The conclusion was to limit the projects to 2D games, and only to focus on the two quality attributes modifiability and testability. 2D games were preferred to 3D games, as the students should not spend too much time on 3D graphics and focus on the structure of the software. We also considered the quality attributes performance and usability for the project. *Performance* was dropped because the XNA framework handles most of the performance issues and it is hard to make architectural design that actually will affect this quality attribute. Further, *usability* was dropped because this quality attribute is rather hard to measure without extensive usability tests (not within the scope of the software architecture course).

The necessary resources to quickly develop games in XNA (Q2) was found to be C# and XNA tutorials, XNA examples, XNA documentation, libraries of graphical art (sprites, tiles, etc.), a high-level API on top of XNA, and making course staff available that could answer specific XNA or C# questions. Although XNA provides a high-level API, the XQUEST framework was found necessary to provide an even higher API to help the students get going faster.

The conclusion of final question (Q3) was that XNA should be exposed to the students through a mixture of lectures, an XNA resource webpage and continues technical support through the semester. It was found to be very important to give an introductory lecture in XNA to learn the tools, environments, and the core concepts of XNA, and give an overview of the differences between Java and C#.

4.2. Changes to the Syllabus. It was rather difficult to change the syllabus of the software architecture course to include more the literature about software architecture in games. Good books and papers that give an in-depth insight into game architectures and game architecture patterns are to our knowledge non-existent. There are several papers that describe architectures of specific games such as [47, 48] or books that give a brief overview of game architecture [49, 50], but none that looks at the typical abstractions (architectural patterns) you can observe in game software development. The syllabus ended up with including some chapter from the book "Game Architecture and Design" [50] to describe the initial steps of creating a game architecture, and two self-composed sets of slides on (1) *software architecture and games* and (2) *architectural patterns and games*. The *former* was a one hour lecture on motivation software architecture design in games [15], architectural drivers within game development [51], challenges related to software architecture in games [52], and the main components of game architectures [53]. The *latter* was a one-hour lecture describing architectural patterns that are common and useful for games, such as model-view controller, pipe-and-filter, layered architecture, and hierarchical task trees.

4.3. Changes of the Project. The course staff decided to let the student teams themselves choose between the robot and the game project. This meant that the main structure of the project had to remain the same, and that we had to make two variants of the project. For the robot project the students had fixed requirements, while for the game project the students should define their own requirements (design their own game). However, the documents to be delivered were the same for both types of projects based on the same templates, and the development process was also to be the same.

To evaluate and grade the software architecture project, we posted some project evaluation criteria in the beginning of the semester that stated *how* the project should be documented, *what* should be *documented*, *what* should be *delivered* (such as documents, source code, complied code, etc.), *completeness* of robot controller or game, and an implementation that *reflects* the architecture. The main difference between the game and the robot versions of the evaluation criteria was how the implementation was to be evaluated. For the XNA projects, we required the game to have a certain level of complexity (at least five classes organized in a structure), the game should be easy to install and run. For a top grade (A), the game should be impressive in some way (fun, nice, creative, or original). For the robot controller, the implementation should similarly have a certain level of complexity, but it had to adhere to the given functional requirements. For a top grade (A), the robot should be able to solve the task efficiently.

Another thing we had to change was the quality attributes the various teams should focus on during the project. The teams that chose the robot projects were assigned to focus on safety of the robot (not get stuck in the maze), modifiability (easiness of changing the robot controller software), and testability (easiness of testing the robot software). For the game projects, we ended up with modifiability (easiness of changing the game software) and testability (easiness of testing the game software).

The main change of the project assignments was to add XNA game variant of the COTS introexercise (phase 1, see Section 2). The COTS intro exercise for the robot controller asked the students to do simple navigation and make to robot pick up balls. In the XNA game variant of this exercise, the students were asked to perform the following four tasks.

(1) Draw a helicopter sprite on the screen and make it move around on its own (computer controlled).

(2) Move around the helicopter sprite from previous task using the keyboard or a game controller, change the size of the sprite, rotate the sprite, and write the position of the sprite on the screen.

(3) Animate the helicopter sprite using several frames and do sprite collision with other sprites.

(4) Create the classical Pong game (2D from-above tennis game).

4.4. Changes of the Staff and the Schedule. The main change to staffing was that two last year master students were hired to give technical support for student during the project (both robot and XNA). The main tasks of the technical support staff were to give lectures on the COTS, to be available for technical questions on email, to be available two hours a week in a lecture halls for questions, and to evaluate the implementation of the final project delivery (testing the games and the robots).

The main changes that were made to the course schedule were:

(i) Changing the motivation of the software architecture project to also include the game project. An extra bonus for the teams that chose the game project was that they could register for the Norwegian Game Awards competition [54]. This is an open national game developer competition for the all universities and colleges in Norway.

(ii) Added an extra two-hour COTS introduction lecture to give an introduction to the robot simulator, C#, and XNA.

(iii) Adding an extra two-hour technical support lecture on COTS every week (both for robot and XNA).

(iv) Changing a one-hour lecture on architectural patterns to also include architectural patterns on games.

(v) Added a one-hour lecture on software architecture in video games.

(vi) Changing the project workshop where selected teams presented their work to give room to show more demos (mostly games and some demos of robots).

5. Experiences and Results

This section presents experiences and results from running the course. The experiences presented here are collected from course staff interviews and notes, final course evaluation, the project reports, student feedback by email, and feedback during lectures. The students doing game development projects used version 2.0 of XNA Game Studio (the most recent version at that time).

5.1. Staff Experiences. In the first weeks of the semester, we were faced with a problem introduced by allowing students to choose between a robot and a game project. In previous years, the students did not have to make any decisions (e.g., forming teams, etc.) regarding the project before week 7, as this was the start of the main project (phase 3, see Section 3). By introducing two variants of the project, the students had to choose in week 3 if they were going to do the robot or the game project (before they had formed the teams) due to the two variants of the COTS exercise. As a result, some students ended up doing an exercise on the robot and later did the game project and vice versa.

The course staff was exited to see the distribution the number of student that chose the robot versus the game project. When we introduced the project to the students in the beginning of the semester, we admitted that this was the first time running a game project in the software

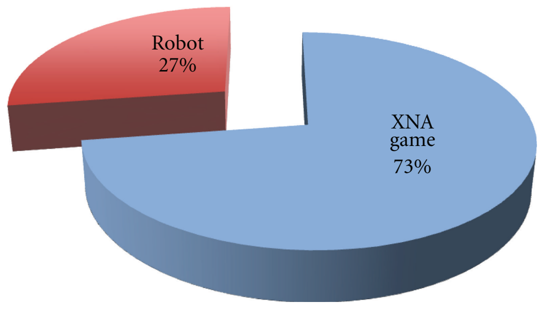

Figure 1: Distribution of project selection.

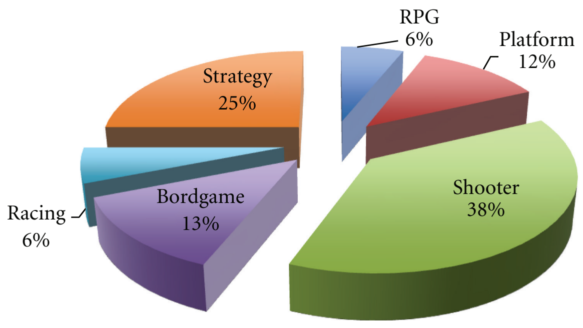

Figure 2: Distribution of game genres in student projects.

architecture course, and that the robot version of the project was better supported through previous experience, examples, the literature, and software architecture patterns. The result was that 6 teams chose the robot project while 16 teams chose the game project (see the distribution in Figure 1). The percentage of teams choosing the game project was much higher than we expected (almost 3 out of 4). The results show that students are attracted to games, and it indicates that games can be a motivation for choosing a course or for putting extra effort into projects.

During the semester, the students receive feedback on their part-deliveries from the course staff. The most notably difference between the part-deliveries made by robot, and game project teams were found in phase 3 of the project (Requirements and Architecture, see Section 3). For many game project teams, it was hard to create proper requirements documentation. This was not unexpected, as these teams first had to specify some gameplay element and then translate these into functional requirements. The course staff suspected that it also would be harder to specify the software architecture in the game projects due to less available literature and architectural patterns. This was, however, not the case. For the final delivery of the project, there was no noticeable difference in the quality of documentation, requirements, design, architecture, and implementation between the two variants (robot versus game). The implementation of some teams (both robot and game) suffered for being too ambitious resulting in unfinished implementations. For teams implementing a robot controller, the main challenge was to implement an intelligent maze navigator. For teams implementing a game, the main challenge was to implement advanced game logic.

The educational approach for our software architecture course is to force the students to use the theory described in the textbook during the project by applying the methods and theoretical framework described. To make this work, the course schedule is heavy on theoretical presentations in the first part of the semester. At the same time, the students have to learn the COTS through exercises (phase 1 and 2). Phase 3 is really the start of the project, where the students will document the requirements and do the architectural design. Although the students at this stage should know the COTS and all the software architectural theory required to describe the requirements and to the design, we discovered that the students were lacking both knowledge of the COTS and the theory. This was true for both types of projects, and we did not discover any differences between robot and game teams. Based on feedback from the course staff and from another student team evaluating the project using ATAM, the software architectures improved significantly in terms of quality and quantity in the implementation phase of the project. The teams discovered problems with their architectural design mainly due to wrong assumptions about the COTS. Both XNA and Khepera put constrains on how to design the architecture, and the students discovered this through trial and error. The XNA teams struggled to make this work due to the complexity of the COTS, while for the Khepera simulator the main problem was lack of documentation. The students learned most during the implementation phase of the project, as they in this phase had to put everything together, reflect on their choices, make changes to make it work, and do the final documentation including updating documentation from previous phases. The course staff also noticed that the students worked a lot the last couple of weeks to be able to finish in time, and put everything together.

One noticeable difference for the course staff after introducing the game project was that the software architecture workshop, where a selected number of teams presented their work, was much more interesting and exciting. In previous years, these workshops have not been very interesting, since most all the students had worked with the same domain (robot). The game projects brought new life to the workshop, and it was very interesting to learn from creative game projects.

5.2. The Games Developed. In total, 16 different 2D games were developed. The type of games varied in several dimensions like number of players, game genre, network support, real-time versus turn-based games, and so forth. The distribution of the game genres implemented by the students is shown in Figure 2. From the figure, we can see that most students chose to implement a variant of a shooter game including a bee-shooter, space shooters, balloon-shooter, tank-shooter, and so forth. The other major game genre was the strategy games that included trading games, and turn-based worm clones.

The student projects also varied in support for multiplayer and network, and usage of the XQUEST-framework as

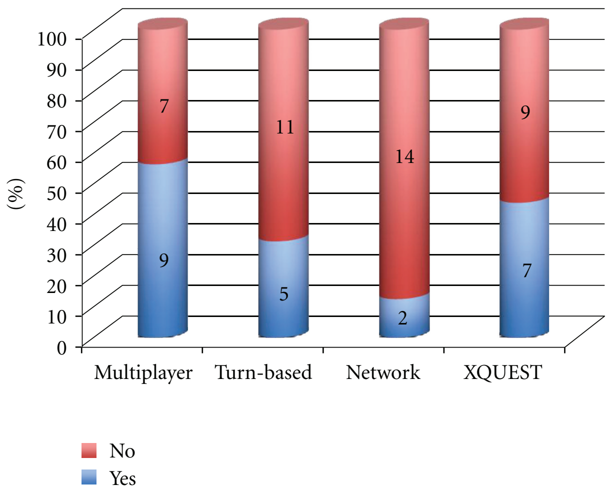

Figure 3: Distribution of game characteristics.

shown in Figure 3. More than 56% of the games developed supported multiplayer, 31% were turn-based, and only two games supported playing over network. About 44% of the games used the XQUEST framework that was developed for this course to simplify the development in XNA.

None of the games developed were groundbreaking in terms of gameplay or graphics, but several of the games had new twists in gameplay or graphics (like including the two most known buildings in the local city—Trondheim). The most novel game was a two-player split screen death-match shooting game, where two players were navigating in an environment that was hand-drawn using colored pencils. One of the levels in the game was actually architectural drawings of the implementation of the game itself. Figure 4 shows a screenshot of this game named BlueRose.

Some of the teams have continued to develop their games after the course ended.

If we look further into differences between how the robot teams and the game teams in terms of the implementation, we found that the projects varied in complexity and size. Although the APIs of XNA and Khepera framework is about at the same abstraction level, the game projects on average had more complex architectures. The architecture of game teams on average consisted of 12 classes compared to 9 for robot teams. We also noticed that the robot teams had a standard deviation of about 3 classes compared to 4 classes for game teams. We found the same tendency for lines of code where robot teams wrote in average 1800 lines of code (without comments), while game teams wrote 3400 (about 90% more lines of code). Another finding was that there was much more variation in number of lines code in game teams compared to robot teams. For robot teams, the most productive team wrote about 2500 lines of code (less than the average for game teams), and the least productive 850 lines of code. For game teams, however, the most productive team wrote about 12000 lines of code and the least productive about 800 lines of code. From analyzing the code, we found that the game teams that produced most lines of code really got carried away with programming the game with less attention to the software architecture. We also compared the final grade of students doing game projects versus students doing robot projects and did not find any significant difference in the final grade. However, we noticed a tendency that students from game teams got a better grade on the project compared to the final written examination, and the students from the robot teams the opposite. An extensive analysis of the differences between the two projects is described in [55].

5.3. Lessons Learned from the Students. This section describes experiences described in the students' lessons learned section of the teams' final reports.A striking difference between students that did a game versus students that did a robot project was how they experienced using the COTS. None of the robot students said anything positive about the Khepera framework. The students that did the game projects described XNA and C# to be easy to learn and work with, that the tools were user-friendly and helpful, that the XNA framework provided the most important functionality including the game loop, and that the game project was very interesting. The students also wrote that it was very valuable to learn XNA and C#, and that XNA and the XQUEST library let them focus on the logic of the video game thus saving a lot of time.

There were several comments both from robot and game teams about the negative experiences from using the chosen COTS. For the students working with the robot simulator, the main problems were related to random and unpredictable behavior of the robot, that the robot simulator performed differently on different PCs, that it was difficult to implement the designed architecture using the API, and that the implementation forced the students to think too much on AI issues instead of software architecture. The random and unpredictable behavior of the robot simulator is a built-in feature to simulate unpredictable sensors in the real worlds. This issue caused a lot of frustration among the students. The different performance of the robot simulator on different PCs is due to problems of real-time execution in Java and real-time performance on different virtual machines. The negative experiences from using XNA was insufficient audio support (only support uncompressed audio files), no support for network testing of two instances on the same machine, limitations of the provided network API in XNA, and that more knowledge of the XNA framework was required to do a good architectural design.

Another topic that was covered by many teams in the lessons learned was their experience with the software architecture domain. Both robot and game teams found that they had learned a lot about software architecture through the design and implementation of the software architecture. One game team said that especially the XQUEST put some major restrictions on the architecture as it was tightly coupled to XNA. This made it difficult to implement a layered architectural pattern. Their conclusion was that the team should have spent more time in the beginning discovering the architectural limitations of the COTS. Another XNA team found that the COTS enabled a proper balance between the game

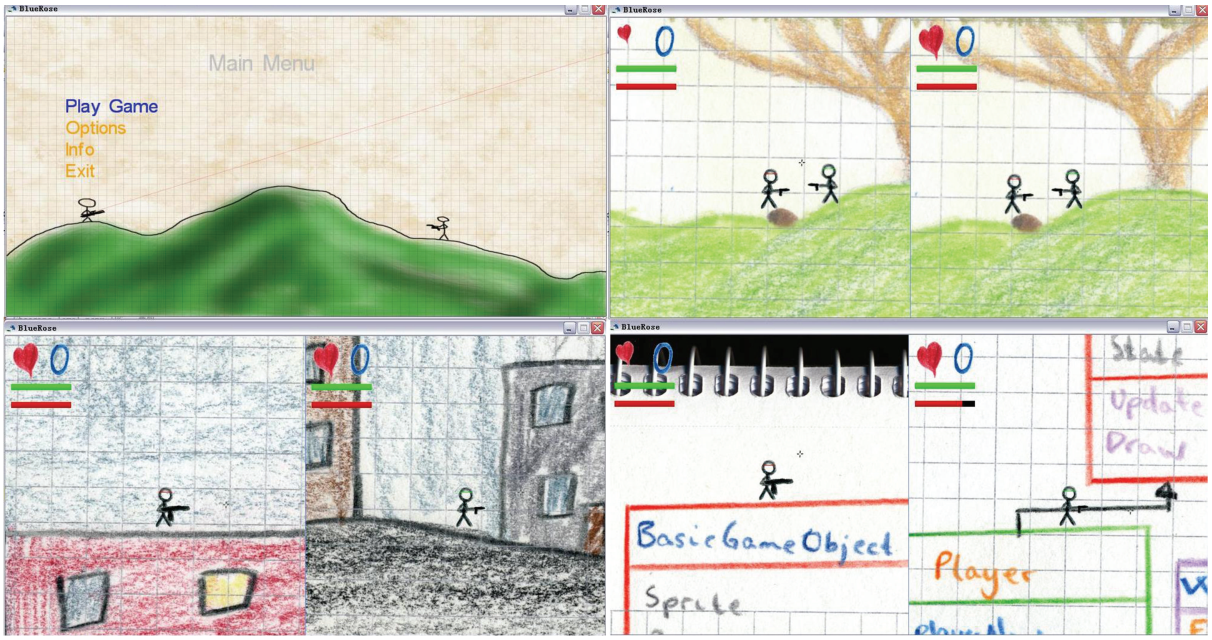

Figure 4: Screenshots from the BlueRose XNA game.

functionality and the software architecture, which resulted in a smooth implementation. Finally, an XNA team described that they did not do an attempt to separate game logic and graphics beyond what was done in XNA, and that this was a big mistake that cause a lot of problems later in the project. For the robot teams, one team said that they used an inappropriate amount of time on the implementation and that the software architecture was therefor, put in the background. One robot team discovered that having a well-planned architecture before starting to implement made it a lot easier to divide the work and make changes during the project. Another robot team explained that they in the beginning only had considered the top-level architecture without examining the architecture of the major modules, which caused a lot of problem. Finally, yet another a robot team admitted that they should had thought more about splitting different classes into packages, as they ended up with code that was hard to modify and manage.

The overall lessons from the students doing a robot project were a mixture of positive and negative issues. The robot simulator itself frustrated the students, and they had nothing positive to say about the COTS. Many students found the robot simulation domain to be fascinating, but they thought it was too difficult to implement the logic of the robot. However, the students had many positive comments about learning software architecture through such a project and designing a software architecture for a robot controller. They also mentioned that they had many reference architectural patterns they could use as a starting point. The hard part was implementing the architecture and the logic for the robot controller.

The overall lessons learned from the students doing an XNA game project were very positive about introducing a game project in a software architecture course. Some students felt that learning C# and XNA in addition to the syllabus was a bit too much, but generally most students said that to learn XNA and C# did not take much time. Some students said that the XNA architecture put major restrictions on their architecture. This is of course true, but this is also the case in most commercial software development projects, as they often use some kind of framework that the architecture must adhere to. The main challenges of using XNA in the software architecture project was to spend enough time learning the framework before designing the architecture, and doing the design and implementation. The identified issue of lacking support for other audio format than wav was resolved in XNA Game Studio 3.0. From the reports we could also see that our own XNA extension (XQUEST) limited the choices of architecture more than only using XNA. The main benefit of using XQUEST was a simpler interface to some of the most useful game functionality.

5.4. Student Evaluation Feedback. After completing the project, all students had to fill in a final course evaluation and write responses to three questions: what has been good about the course, what has been not so good, and what would you like to change to next year?

The responses regarding *what had been good* about the course can be categorized into main areas the project, learning, practical work, and group dynamics. Both students from robot and game teams stated that the project had been good, but students from game teams were overall happier with the project and described it to be cool, interesting, fun, and motivating. Also both categories of students described that they learned a lot from the project in that they got to try out the theory from the lectures in practice. They also gave concrete example of theory that they got to try out in the project such as architectural and design patterns and how the software architecture is represented in code. Many students from game teams also wrote that the project was a fun way of learning software architecture and that it was useful to learn about the interplay of game and architectural approaches. Regarding the practical work, students from game teams mentioned that it was really useful to learn C# as it is commonly used in industry and that it was easy to learn because of its similarities with Java. Both robot and game students gave positive comments about the fact that the course forced the students to do practical work. Finally, it was mentioned that it was useful to learn from other teams through the final workshop. The responses from the students taking the course were overall very positive. The feedback from game team students was generally more positive than the feedback from the robot controller projects. Typical positive feedback we received from students doing a game project was that they felt they learned a lot from the game project, that they liked the practical approach of the project and having to learn C#, and the interaction between the teams (both ATAM and the project workshop). The students doing a robot project were pleased with learning software architecture through practical work, and thought it was very interesting to learn about software architecture in general.

The responses regarding *what had been not so good* about the course mainly concerned the COTS. Both students from robot and game teams complained about the lack of technical support during the project and sufficient introductory lecture on the COTS in the beginning of the course. Further, both categories of students complained that the COTS took away focus from software architecture in the course. Few students on game teams complained that learning C# took so much time that they did not have enough time to study software architecture. Some other students on game teams said that the focus on the game itself keep them from focusing on the software architecture, and that the game domain limits the choice of architecture too much. Students on robot teams complained that the difficulty of implementing the robot controller took the focus away from architectural design, and that the workload of the project was way too high. The main negative feedback from students doing game projects focused on the lack of XNA technical support during the project, and that some student felt that there was too much focus on C#, XNA and games and too little on software architecture. The students doing a robot project also complained about not sufficient technical assistance, and that the robot simulator and the robot domain were very difficult to master.

On the final question in the course evaluation, *what would you have changed* for next year's course; we received various course improvement suggestions. Game team students suggested to allocate more time to develop the game, to make the project count 50% of the grade, to give a better C# introduction, to provide better technical support, and to put more restrictions on game-type to ensure that the teams choose games suited for the course. The robot team students suggested to either give better information on how to program the robot or drop the robot project all together, provide better technical support during the project, and split the project into several smaller exercises. One robot team student said that he rather would choose the game project if he could start all over again. The suggestions to improve the course were mainly according to the negative feedback namely to improve teaching and technical support related to the COTS (XNA and robot simulator), and to adjust the workload of the project.

6. Conclusion

In this paper, we have described how we changed a software architecture course to include a game development project. The main motivation for introducing such a project was to motivate the students to put extra effort into the project and motivating for higher course enrollment. Some parts of the syllabus were changed to include game development as a natural part of the software architecture course. A challenge we discovered was to find the appropriate literature on design of software architecture for the game domain, which we are still looking for. It is not very hard to motivate for why game developers can benefit from learning more about software architectures as games are becoming increasingly more complex (especially massively multiplayer online games). From a software architecture perspective, games are interesting since they introduce relevant challenges such as dealing with continues changes of functional requirements (modifiability), and hard real-time requirements both for hardware and network.

Our experience from running a game development project in a software architecture course is very positive. The course staff noticed an increasing interest and motivation for the project in the course. From the course evaluation, we also notice that students choosing the game project were more positive towards the project compared to those who chose the robot project. Robot team students complained more about the project while game team students generally expressed that the project was fun and engaging. Game development projects are also very positive for the group dynamics, as other that CS and SE skills are required (e.g., creative and artistic skills). The main negative effect of introducing a game development project was that some teams focus more on developing the game than on the software architecture of the game. This effect was not a major issue, as most teams did a good job of designing the architecture and then implementing it. There will always be some students that do not like to do a project on games. When we looked at the demographics to see if there were any various in choosing game projects, we only found minor variations between male (73%) and female (71%). Actually, the difference was larger between Norwegians (74%) and foreign students (70%). One challenge for some students was that they had to learn C#. Most students did not think this issue was negative thing, as to know C# is useful for later in the career and it is not very different from Java. Another challenge using XNA as a development platform was that it only runs on the Microsoft Windows platform. This is a major problem as more and more students have laptops running Mac OS X and Linux. To compensate for this problem, we provided a computer lab where 10 PCs running Microsoft Windows with XNA Game developer studio 2.0 installed. Unfortunately, these PCs did not have proper graphics cards, making game development slow and tedious. To compensate for this problem in the future, we might offer game projects on other platforms such as Android and iPhone. Apart from the lack of support for other operating systems, we were very pleased with using XNA as a game developer platform. The high-level APIs in XNA makes it possible to be productive with little effort. Also XNA is flexible in terms of what games can be implemented and how the architecture can be designed. For the students, the opportunity to develop XBOX 360 games is very tempting. Only few of the teams tried to run their games on the XBOX 360 mainly due to time pressure. In XNA Game Studio 4.0, it is also possible to develop for Windows Phone, extending the target platform even more. This can give more variety of what kind of projects the students can develop in future projects.

Acknowledgments

The author would like to thank Jan-Erik Strøm and Trond Blomholm Kvamme for implementing XQUEST and for their inputs to this paper. They e would also like to thank Richard Taylor at the Institute for Software Research (ISR) at University of California, Irvine (UCI) for providing a stimulating research environment and for hosting a visiting researcher from Norway. The Leiv Eriksson mobility program offered by the Research Council of Norway has sponsored this work.

References

[1] R. Rosas, M. Nussbaum, P. Cumsille et al., "Beyond Nintendo: design and assessment of educational video games for first and second grade students," *Computers and Education*, vol. 40, no. 1, pp. 71–94, 2003.

[2] M. Sharples, "The design of personal mobile technologies for lifelong learning," *Computers and Education*, vol. 34, no. 3-4, pp. 177–193, 2000.

[3] A. Baker, E. O. Navarro, and A. Van Der Hoek, "Problems and programmers: an educational software engineering card game," in *Proceedings of the 25th International Conference on Software Engineering (ICSE '03)*, pp. 614–619, Irvine, Calif, USA, May 2003.

[4] L. Natvig, S. Line, and A. Djupdal, "Age of computers: an innovative combination of history and computer game elements for teaching computer fundamentals," in *Proceedings of the 34th Annual Frontiers in Education: Expanding Educational Opportunities Through Partnerships and Distance Learning*, pp. F-1–F-6, Trondheim, Norway, October 2004.

[5] A. O. Navarro and A. Hoek, "SimSE: an educational simulation game for teaching the software engineering process," in *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '04)*, p. 233, ACM Press, New York, NY, USA, 2004.

[6] G. Sindre, L. Natvig, and M. Jahre, "Experimental validation of the learning effect for a pedagogical game on computer fundamentals," *IEEE Transactions on Education*, vol. 52, no. 1, pp. 10–18, 2009.

[7] B. A. Foss and T. I. Eikaas, "Game play in engineering education—concept and experimental results," *International Journal of Engineering Education*, vol. 22, no. 5, pp. 1043–1052, 2006.

[8] A. I. Wang, O. K. Mørch-Storstein, and T. Øfsdahl, "Lecture quiz—a mobile game concept for lectures," in *Proceedings of the 11th IASTED International Conference on Software Engineering and Application (SEA '07)*, November, 2007.

[9] A. I. Wang, T. Øfsdahl, and O. K. Mørch-Storstein, "An evaluation of a mobile game concept for lectures," in *Proceedings of the 21st Conference on Software Engineering Education and Training, (CSEET '08)*, pp. 197–204, April 2008.

[10] M. S. El-Nasr and B. K. Smith, "Learning through game modding," *Computers in Entertainment*, vol. 4, no. 1, pp. 45–64, 2006.

[11] B. Wu and A. I. Wang, "An evaluation of using a game development framework in higher education," in *Proceedings of the 22nd Conference on Software Engineering Education and Training, (CSEET '09)*, pp. 41–44, Hyderabad, India, February 2009.

[12] A. Sliney and D. Murphy, "JDoc: a serious game for medical learning," in *Proceedings of the 1st International Conference on Advances in Computer-Human Interaction, (ACHI '08)*, pp. 131–136, February 2008.

[13] F. Mili, J. Barr, M. Harris, and L. Pittiglio, "Nursing training: 3D game with learning objectives," in *Proceedings of the 1st International Conference on Advances in Computer-Human Interaction, (ACHI '08*, pp. 236–242, Rochester, NY, USA, February 2008.

[14] L. V. Ahn, "Games with a purpose," *IEEE Computer Magazine*, vol. 39, no. 6, pp. 92–94, 2006.

[15] J. Blow, "Game development: harder than you think," *ACM Queue*, vol. 1, no. 10, pp. 28–37, 2004.

[16] G. M. Youngblood, "Using XNA-GSE game segments to engage students in advanced computer science education," in *Proceedings of the 2nd Annual Microsoft Academic Days Conference on Game Development*, February, 2007.

[17] E. Sweedyk and R. M. Keller, "Fun and games: a new software engineering course," *ACM SIGCSE Bulletin*, vol. 37, no. 3, pp. 138–142, 2005.

[18] K. Claypool and M. Claypool, "Teaching software engineering through game design," in *Proceedings of the 10th Annual SIGCSE Conference on innovation and Technology in Computer Science Education (ITiCSE '05)*, pp. 123–127, Caparica, Portugal, June, 2005.

[19] D. Volk, "How to embed a game engineering course into a computer science curriculum," in *Proceedings of the Conference on Future Play: Research, Play, Share*, pp. 192–195, Toronto, Canada, November, 2008.

[20] J. Linhoff and A. Settle, "Teaching game programming using XNA," in *Proceedings of the 13th Annual Conference on innovation and Technology in Computer Science Education (ITiCSE '08)*, pp. 250–254, Madrid, Spain, June-July 2008.

[21] Q. Zhu, T. Wang, and S. Tan, "Adapting game technology to support software engineering process teaching," in *Proceedings of the 3rd International Conference on Natural Computation, (ICNC '07)*, pp. 777–780, Haikou, China, August 2007.

[22] Y. Rankin, A. Gooch, and B. Gooch, "The impact of game design on students' interest in CS," in *Proceedings of the 3rd International Conference on Natural Computation (ICNC '07)*, pp. 777–780, Miami, Fla, USA, February-March 2008.

[23] S. Leutenegger and J. Edgington, "A games first approach to teaching introductory programming," *SIGCSE Bulletin*, vol. 39, pp. 115–118, 2007.

[24] B. D. Coller and M. J. Scott, “Effectiveness of using a video game to teach a course in mechanical engineering,” *Computers and Education*, vol. 53, no. 3, pp. 900–912, 2009.

[25] J. Distasio and T. Way, “Inclusive computer science education using a ready-made computer game framework,” in *Proceedings of the 12th Annual Conference on Innovation and Technology in Computer Science Education*, pp. 116–120, June 2007.

[26] Washington State University Vancouver and University of Puget Sound, “2008 The Java Instructional Gaming Project,” June 2008, http://ai.vancouver.wsu.edu/jig/.

[27] C. Johnson and J. Voigt, “DXFramework,” June 2008, http://www.dxframework.org.

[28] I. Parberry, “SAGE: a simple academic game engine,” June 2008, http://larc.csci.unt.edu/sage.

[29] R. Coleman, S. Roebke, and L. Grayson, “GEDI: a game engine for teaching videogame design and programming,” *Journal of Computing Science in Colleges*, vol. 21, no. 2, pp. 72–82, 2005.

[30] Rockwell Automation Inc, “Arena Simulation Software,” June 2008, http://www.arenasimulation.com.

[31] IBM, “INNOV8—a BPM Simulator,” June 2008, http://www-304.ibm.com/jct03001c/software/solutions/soa/innov8.html.

[32] P. Clements, L. Bass, and R. Kazman, *Software Architecture in Practice* , Addison-Wesley, 2nd edition, 2003.

[33] J. O. Coplien, “Software design patterns: common questions and answers,” in *The Patterns Handbook: Techniques, Strategies, and Applications*, pp. 311–320, Cambridge University Press, New York, NY, USA, 1998.

[34] A. I. Wang and T. Stålhane, “Using post mortem analysis to evaluate software architecture student projects,” in *Proceedings of the 18th Conference on Software Engineering Education and Training (CSEET ’05)*, pp. 43–50, April 2005.

[35] D. P. Perry and A. L. Wolf, “Foundations for the study of software architecture,” *ACM Sigsoft Software Engineering Notes*, vol. 17, no. 4, pp. 40–52, 1992.

[36] IEEE, “IEEE recommended practice for architectural description of software-intensive systems,” *Software Engineering Standards Committee of the IEEE Computer Society*, 2000.

[37] P. B. Kruchten, “The 4+1 view model of architecture,” *IEEE Software*, vol. 12, no. 6, pp. 42–50, 1995.

[38] R. Kazman, M. Klein, R. Kazmani et al., ““The architecture tradeoff analysis method,” engineering of complex computer systems,” in *Proceedings of the Fourth IEEE International Conference on Engineering Complex Computer Systems (ICECCS ’98)*, vol. 0, 1998.

[39] A. BinSubaih and S. C. Maddock, ““Using ATAM to evaluate a game-based architecture”, workshop on architecture-centric evolution,” in *Proceedings of the 20th European Conference on Object-Oriented Programming (ECOOP ’06)*, Nantes, France, July 2006.

[40] T. Lozano-Pérez, *Autonomous Robot Vehicles*, Springer, New York, NY, USA, 1990.

[41] A. Elfes, “Sonar-based real-world mapping and navigation,” *IEEE Journal of Robotics and Automation*, vol. 3, no. 3, pp. 249–265, 1987.

[42] R. G. Simmons, “Concurrent planning and execution for autonomous robots,” *IEEE Control Systems Magazine*, vol. 12, no. 1, pp. 46–50, 1992.

[43] S. A. Shafer, A. Stentz, and C.E. Thorpe, “An architecture for sensor fusion in a mobile robot,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2002–2011, April 1986.

[44] D. Toal, C. Flanagan, C. Jones, and B. Strunz, “Subsumption architecture for the control of robots,” in *Proceedings of the 13th Irish Manufacturing Conference (IMC-13)*, pp. 703–711, 1996.

[45] R. Lumia, J. Fiala, and A. Wavering, “The NASREM robot control system and testbed,” *The International Journal of Robotics and Automation*, no. 5, pp. 20–26, 1990.

[46] A. I. Wang and B. Wu, “An application of game development framework in higher education,” *The International Journal of Computer Games Technology*, 2008.

[47] C. Vichoido, M. Estranda, and A. Sanchez, “A constructivist educational tool: software architecture for web-based video games,” in *Proceedings of the 4th Mexican International Conference on Computer Science (ENC ’03)*, Apizaco, Mexico, September 2003.

[48] J. Krikke, “Samurai romanesque, J2ME, and the battle for mobile cyberspace,” *IEEE Computer Graphics and Applications*, vol. 23, no. 1, pp. 16–23, 2003.

[49] S. Rabin, “Introduction to game development,” in *Course Technology Cengage Learning*, 2008.

[50] A. Rollings and D. Morris, *Game Architecture and Design—A New Edition*, New Riders, 2004.

[51] G. Booch, “Best practices in game development,” IBM Presentation, March 2007.

[52] A. Grossman, *Postmortems From Game Developer*, Focal Press, 2003.

[53] R. Darken, P. McDowell, and E. Johnson, “Projects in VR: the delta3D open source game engine,” *IEEE Computer Graphics and Applications*, vol. 25, no. 3, pp. 10–12, 2005.

[54] NGA, “Norwegian game awards 2011—home,” April 2011, http://www.gameawards.no.

[55] A. I. Wang, “Extensive evaluation of using a game project in a software architecture course,” *Computers & Education*, vol. 11, pp. 1–28, 2011.