X

Extensive Evaluation of Using a Game Project in a Software Architecture Course

A.I. WANG
Norwegian University of Science and Technology

This paper describes an extensive evaluation of introducing a game project to 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. In 2008, the students on the software architecture course could choose between the two domains: Khepera robot simulation in Java and XNA game development in C#. Independent of the domain chosen, the students had to go through the same phases, produce the same documents based on the same templates, and follow exactly the same process. This paper describes an evaluation where we wanted to investigate if a game development project could successfully be used to teach software architecture. Specifically in the evaluation, the effect of the choice of COTS (Commercial Off-The-Shelf) and domain is compared in relation to popularity of the project type, how the students perceive the project, the complexity of the software architectures produced, the effort put into the project, and the grades achieved for the project and the written examination. The main conclusion is that game development projects can successfully be used to teach software architecture. Further, the results of the evaluation show among other things that students that chose the Game project produced software architecture with higher complexity, and put more effort into the project than the Robot project students. No significant statistical differences were found in final grades awarded to the Game project students vs. Robot project students. However, the Game project students obtained a higher grade in their project than in the written examination whereas the Robot project students scored higher in the written examination than in their project. Finally compared to the Robot project students, those that chose the Game project had fewer problems with COTS hindering the architecture design and introducing technical challenges.

Categories and Subject Descriptors: K.3.2 [**Computer and Information Science Education**], D.2.11 [**Software Architectures**], K.8 [**Personal Computing**] – Games.

General Terms: Software Engineering Education, Evaluation, Game Development.

Additional Key Words and Phrases: Game Development, XNA, Robot simulation.

ACM File Format:
Wang, A. I. 2010. Extensive evaluation of using a game Project in a software architecture course. ACM Trans on Computing Education, X, X, Article X (Januay 2011), 29 pages, DOI =

1. INTRODUCTION

Games in education have become increasingly popular in recent years, especially for children and have proven to be beneficial for academic achievement, motivation and classroom dynamics [Rosas et al., 2003]. Teaching methods based on educational games are not only attractive to schoolchildren, but can also be beneficial for university students [Sharples, 2000]. Research on games concepts and game development used in higher education is not unique, e.g. [Baker et al., 2003] [Natvig et al., 2004] [Navarro&Hoek, 2004], but there is an untapped potential that needs to be explored. By introducing games in higher education professors can access teaching aids that promote more activity among students, provide alternative teaching methods to improve variation, enable social learning through multiplayer learning games, and motivate students to work harder on

Author's address: A.I.Wang, Dept. of Computer and Information Science, Norwegian University of Science and Technology, Trondheim, Norway. Email: alfw@idi.ntnu.no

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 [Foss&Eikaas, 2006] [Sindre et al., 2009]. *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 [Wang et al., 2007] [Wang et al., 2008]. *Third*, game development projects can be used in computer science (CS) or software engineering (SE) courses to learn specific CS or SE skills [El-Nasr&Smith, 2006] [Wu&Wang, 2009]. This paper focuses on an evaluation of the latter, where a game development project was introduced to a software architecture course. The motivation for bringing game development into a CS or SE course is to exploit the students' fascination for games and game development to stimulate them to work more with course material through the project. Many students dream of making their own games, and game development projects stimulate the creativity of the students. In addition, game technologies and game user interfaces are now more commonly used in serious applications [Homes, 2005] [Sliney et al., 2008] [Mili, et al., 2008] [Ahn, 2006], and development of serious games is on the rise. This makes it more important for students to learn how to develop games and utilize game technology.

From a game developer's perspective, knowledge and skills about how to develop appropriate software architectures are becoming more important [Caltagirone et al., 2002] [Anderson et al., 2008]. Well-designed software architectures are needed, as games are growing in size and becoming more complex [Blow, 2004]. From a software architect's perspective, games are interesting due to the inherent characteristics of the domain including real-time graphics and network constraints, variation in hardware configurations, changing functionality, and user-friendliness. Games are also interesting from a software architect's perspective, as there are 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 [Callele et al., 2008].

This paper describes an evaluation of introducing a game project in a software architecture course to find an answer to the research question: *"Are game development projects suited for teaching software architecture?"* The evaluation is a comparison of how students who chose a Game project perform vs. students who chose a Robot project. The students all go through the same phases and produce all the same documents based on templates that are independent of the chosen domain. The evaluation will also look at the students' perception of the project, and the popularity of the two domains related to demographics. The evaluation is based on data from a project survey, the project deliverables from the students and other accessible course information.

The rest of the paper is organized as follows. Section 2 describes the software architecture course. Section 3 presents the research questions and research method. Section 4 gives the results of the evaluation. Section 5 discusses the results and addresses the validity of the evaluation. Section 6 describes related work, and Section 7 concludes the paper.

2. DESCRIPTION OF THE SOFTWARE ARCHITECTURE COURSE

The software architecture course is for post-graduate CS and SE students at the Dept. of Computer and Information Science at the Norwegian University of Science and Technology (NTNU). The course workload is 25% of a semester, and about 70-80

students attend the course every spring. The students are mostly Norwegian (about 80%), but there are also 20% international students mostly from EU countries. About 10% of the students are female. The textbook used in this course is "Software Architecture in Practice, Second Edition", by Clements, Bass, and Kazman [2003]. Additional papers are used to cover topics that are not sufficiently covered by this book such as design patterns, software architecture documentation standards, view models, and post-mortem analysis [Coplien, 1998] [Perry&Wolf, 1992] [IEEE, 2000] [Kruchten, 1995] [Wang&Stålhane, 2005]. The learning outcomes from the course are that:

> *"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 mainly taught in three ways:

1) Ordinary lectures given in English
2) Invited guest-lectures from the software industry
3) A software development project that focuses on software architecture

2.1 An Unusual Approach

Programming has for many years been used for teaching students of all ages more then just programming. Papert [1980] inspired by Piaget's theories on assimilation (the process by which a person takes material into their mind from the environment…) [Piaget, 1969] demonstrated how children could learn mathematics for example through programming in Lego Mindstorm. Specialized programming languages and integrated development environments have been developed to teach students programming as well as other topics. Some examples are the Logo Turtle used for learning simple graphical principles [Papert, 1980], StarLogo programming language that was developed to teach students to do simulation of micro worlds (termites, traffic etc.) [Resnick, 1994], Alice that was designed to teach students object-orientation as well as building 3D applications [Pausch, 1995], and Scratch that was designed for rapid prototyping of media rich applications [Resnick et al., 2003]. Alice and Scratch have also been used in courses teaching students game development.

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 Dept. of Computer and Information Science in surveys conducted among ex-students 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.

2.2 Course Evaluation

In the software architecture course 30% of the grade awarded relates to the evaluation of the software architecture project all students have to do, while 70% is awarded for the results of a written examination. One grade per group is given for the project, while individual grades are given on the written examination. In special cases, students can get individual grades on the project if they have not contributed as much as the other group members. The goal of the project is for the students to apply the methods and theory examined in the course to design and fully document a type of software architecture, evaluate the architecture and the architectural approaches (tactics), implement an application according to the architecture, test the implementation related to the functional and quality requirements, and evaluate how the architectural choices affect the quality of the application. The course staff will evaluate the project according to *evaluation criteria* described in a document that is available to all students at the beginning of the project. The project is evaluated according to completeness of the architecture documentation as described in the IEEE 1471 standard [IEEE, 2000], a working implementation according to the defined requirements and the architecture, consistency between code and architecture, structured and readable documentation, structured and readable code, testable functional and quality requirements, architecture rationale, documentation according to given templates, and a clear description how the project uses COTS. The main emphasis when grading the projects is on the software architecture itself, but also on how the implementation reflects the architectural choices.

A team of four consisting of a professor and three PhD candidates rates the projects. The three PhD candidates rate one third of the projects while the professor rates all projects; so all projects are evaluated by two people.

2.3 The Software Architecture Project

The software architecture 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 quality requirements, describe the architectural drivers, and design and document the software architecture of the application in the project including several views and viewpoints, stakeholders, stakeholder concerns, architectural rationale.
4) *Architecture evaluation:* Use the Architecture Trade-off Analysis Method (ATAM) [Clements et al., 2003] [Kazman et al., 1998] [BinSubaih&Maddock,

2006] to evaluate the software architecture in regard to the specified quality requirements.

5) *Implementation:* Do a detailed design and implementation of the application based on the designed architecture and on the results from the ATAM evaluation. Test the application against 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 [Wang&Stålhane, 2005]. In this phase, the students will elicit and analyze the successes and problems they had during the project.

In the first two phases of the project, the students work on their own or in pairs. For phases 4-6, the students work in self-selected teams of four students. The students spend most time in the implementation phase (6 weeks), and they are also encouraged to 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 iterations.

In previous years, the goal of the project has been to develop a robot controller for the WSU Khepera robot simulator in Java [WSU, 2009] with emphasis on an assigned quality attribute such as availability, performance, modifiability or testability. The students were asked to program the robot controller to move a robot around in a maze, collect four balls and bring them to a light source in the maze. The robot controller was chosen for the software architecture project, as the problem of software architecture is well defined within this domain. Several examples of software architecture patterns or reference architectures for the robot controller domain are available such as Control loop [Lozano-Pérez, 1990], Elfes [Elfes, 1987], Task Control [Simmons, 1992], CODGER [Shafer et al., 1986], Subsumption [Toal et al., 1996], and NASREM [Lumia et al., 1990].

In 2008, the students were allowed to choose between a robot controller project and a game development project. The process, the deliverables and the evaluation of the project were the same for both types of projects – only the domain was different. In the Game project, the students were asked to develop a game using the Microsoft XNA framework [Microsoft, 2009a] and C# [Microsoft, 2009b]. All our students have good skills and knowledge in Java, but very few knew C#. The students were allowed to decide what type of game they wanted to develop themselves, but a certain level of complexity (more than a specified number of classes) was required. Unlike the robot domain, there was little appropriate literature on software architecture and software architectural patterns for games. There are some papers and presentations that describe the architectures of specific games [Vichoido et al., 2003] [Krikke, 2003] [Booch, 2007] [Grossman, 2003] [Darken et al., 2005], and books that give a brief overview of game architectures [Rabin, 2008] [Rollings&Morris, 2004], but no literature that gives an in-depth study of the typical abstractions one can observe in game software development. The most recurring architectural patterns described in books and papers are the model-view controller, pipe-and-filter, layered and hierarchical task trees.

The evaluation presented in this paper is within the context of a software architecture course, but the results presented should also be applicable to other courses as well, as the robot and game domains are commonly being used to teach various topics in software engineering and computer science. The robot domain has been used to teach various CS

topics such as expressions, loops, finite state machines, data structures, threading, fuzzy logic, and embedded systems [Linder et al., 2001] [Delden and Zhong, 2008], and artificial intelligence and real world vs. virtual world interfaces [Pfeifer, 1997] [Flowers and Gossett, 2002] [Imberman, 2004]. Similarly the game domain has been used to teach object-oriented programming [Chen and Cheng, 2007], object-oriented software engineering [Ryoo, 2008], human-computer interaction [Shiray et al., 2009], software design and software process [El-Nasr and Smith, 2006], artificial intelligence [Sung, 2009], algorithms [Faltin, 1999], design patterns and architecture [Gestwicki, 2007] [Nguyen and Wong, 2002], and computer graphics [Sung et al., 2007]. The two domains (robot and games) have even been combined to motivate learners to become interested in programming, science technology, engineering and math [Lahey et al., 2008].

3. RESEARCH QUESTIONS AND RESEARCH APPROACH

The goal of the evaluation presented in this paper was to investigate if there were any differences in how students perceived the project, and how they performed in the project and the course related to their choice of project domain (Robot vs. Game). The Robot project represents our benchmark of a successful project in teaching students software architecture. The Robot project was chosen as a benchmark based on experience from five previous years of successfully teaching students the practices, skills and techniques in software architecture in a practical way [Wang&Stålhane, 2005]. This meant that if the game project performed at the same level as the Robot project, the Game project was well suited for teaching software architecture.

The comparison of the Robot and Game projects should help to discover the differences and reveal the positive and negative effects of introducing a Game project. However, the evaluation cannot be defined as a controlled experiment. The research method used is based on the Goal, Question Metrics (GQM) approach [Basili et al., 1995] where we first define a research goal (conceptual level), then define a set of research questions (operational level), and finally describe a set of metrics to answer the defined research questions (quantitative level). In our case, the metrics used to give answers to the research questions are a mixture of quantitative and qualitative data.

3.1 Research Goal and Research Questions

The research goal of this study was defined as the following using the GQM template:

> The purpose of this study was to *evaluate* the effect of *using a game development project* from the point of view of *a student* in the context of *a software architecture course*.

The following research questions (RQs) were defined by decomposing the research goal above:

- RQ1: Are game projects popular among the students in a software architecture course, and are there any specific groups that favor game projects?
- RQ2: Are there any differences in how the students perceive the project for students choosing a Robot project vs. students choosing a Game project?
- RQ3: Are there any differences in the software architectures designed by students doing a Robot project vs. students doing a Game project?
- RQ4: Are there any differences in the effort put into the project by students doing a Robot project vs. students doing a Game project?

- RQ5: Are there any differences in the performance of students doing a Robot project vs. students doing a Game project?

3.2 Data Sources and Metrics

Table I shows the data sources, the metrics and how the data are compared with respect to the five research questions given in Section 3.1. Note that the qualitative data are mainly used as a supplement to the quantitative data.

Table I. Data Sources, Metrics and Comparison Method

RQs	Data sources	Metrics	Comparison method
RQ1	Course Data	Numeric data: *[project selection data, demographic classification of students]*	Percentwise distribution chart.
RQ2	Project Survey	5-level Likert scale: *[Strongly agree (1) - Agree (2) - Neutral (3) - Disagree (4) - Strongly disagree (5)]*	Kruskal-Wallis Test. Percentwise distribution chart.
RQ3	Project Reports	Numeric data: *[Number of classes/modules, Number of patterns, Number of levels in the architecture]* Quantitative data: *[Diagrams, Textual descriptions]*	Kruskal-Wallis Test. Percentwise distribution chart. Comparison of statistical average, standard deviation, min and max.
RQ4	Source Code, Implemented Applications	Numeric data: *[Number of Source files, Lines of Source code, Number of Comments]* Quantitative data: *[Tests from running applications]*	Kruskal-Wallis Test. Comparison of statistical average, standard deviation, min and max.
RQ5	Evaluation of Projects, Evaluation of Examination	Numeric data: *[Project Score and Final Examination Score]*	Kruskal-Wallis Test. Comparison of statistical average and median. Percentwise distribution chart.

Here is a more detailed description of the data sources in this evaluation:

- **Course data:** This is data that describes how many students participate in the course, how many students chose the two types of project, and what kind of students chose a particular project type.
- **Project Survey:** The survey consisted of 10 statements where the students should choose from a 5-level Likert scale (from Strongly Agree to Strongly Disagree). The survey was published on an e-learning system one week after the students have completed their project. 83% of the students that had signed up for the course responded (66 students).
- **Project reports:** Project reports from 22 teams were analyzed by counting differences, analyzing diagrams and reading through the textual descriptions.
- **Source code:** The source code of all 22 teams was analyzed using the *cloc* application [Danial, 2009] for counting comment lines, lines of code and number of source code files.
- **Implemented applications:** The implemented applications were tested to discover their robustness and how advanced they were.

- **Evaluation of projects:** The course staff gave a score and a grade (A to F) based on an evaluation of the final project delivery according to a specified set of project evaluation criteria. The score spans from 0-30 points.
- **Evaluation of examination:** The course staff gave a score and a grade (A to F) on the final written examination. The score spans from 0-70 points.

4 RESULTS OF THE EVALUATION

This section presents the results of the evaluation giving answers to the five research questions introduced in Section 3.1.

4.1 RQ1: Game Project Popularity and Demographics

One of the main reasons for introducing a game project to the software architecture course was to motivate students to put extra effort into the project. One indicator of whether students were motivated by the Game project was to see how many students preferred the Game project to the Robot project. In spring 2008, 82 students had registered for the software architecture course. The distribution of the students' selection of type of project is shown in Fig. 1. The pie chart shows that almost three out of four chose the Game project. The number of students that preferred the Game project to the Robot project was overwhelming and much higher than expected.

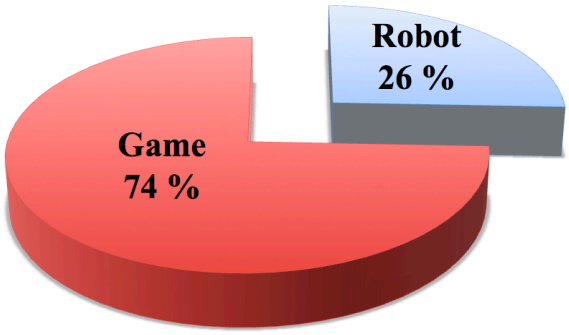

Fig. 1. The distribution of selection of type of software architecture project

Research question one (RQ1) also stated if there were any specific groups that favored the Game Project. The only demographical data available for the students taking the software architecture course were citizenship and gender. Fig. 2 shows how the groups Norwegians, foreigners, males and females chose the project type.

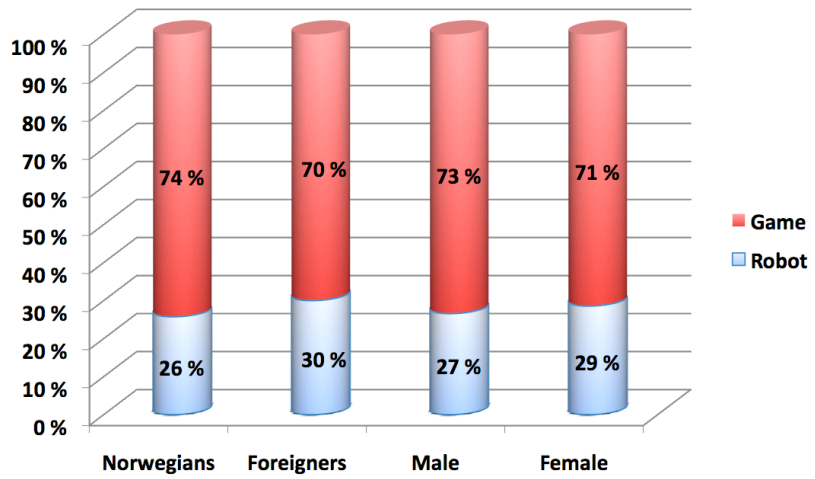

Fig. 2. Student demographics and selection of project type

The chart shows that there are *only minor variations* in how these four groups choose the project type. We expected more difference between the two groups male and female. However, one can argue that both domains are rather masculine and that none of the offered domains are especially tempting to females. It would be interesting to see if a domain such as social network applications would have changed the choice of domain for female students.

4.2 RQ2: Differences in how Students Perceived the Project

A project survey was conducted one week after the students completed their software architecture project. The goal of this survey was to reveal possible differences in the students' perception of the project between teams working with Robot projects vs. teams working with Game projects. Most statements in the survey made the students reflect on how the project helped them to learn software architecture. First, the students had to specify whether they had worked with a Robot or a Game project. Then, the students were asked to do grade nine statements (PS1-PS9) by choosing an alternative from 1 (Strongly Agree) to 5 (Strongly Disagree). Finally, the students were asked whether they would choose the other type of project next time (PS10), where the alternatives to choose from were 1 (No) or 2 (Yes).

The hypothesis defined for this survey was the following:

> H_0: *There is no difference in how students doing the Robot project vs. the Game project perceive the software architecture project.*

The Kruskal-Wallis Test was used to do the hypothesis tests. The Kruskal-Wallis test is a non-parametric method for testing equality of population medians among groups [Kruskal&Wallis, 1952]. This test was suitable for this survey, as we cannot assume a normal population and the sample sizes of the two groups are different. Table II and Table III show the results of the Kruskal-Wallis Test on the statements PS1-PS10. Note that two p-values are given. The unadjusted p-value is conservative if ties are present,

while the adjusted p-value is usually more accurate, but it is not always conservative. Of the 68 students answering the survey, 20 students worked with Robot projects while 48 students worked with Game projects.

Table II shows that for PS2 there is a significant difference ($p \leq 0.05$) for the two groups' responses. The students doing the Game project claimed that the *COTS to a less degree hindered good architecture design* compared to Robot project students. This result was unexpected, as the course staff believed the opposite would be true due to more available software architecture resources related to robot controllers, and few architectural restrictions in the Khepera robot simulation framework. Also the statement PS3 had a rather low p-value ($p=0.097$) where the Game project students found it more difficult to evaluate the other team's architecture using ATAM than the Robot project students.

Table II. Kruskal-Wallis Test of the statements PS1-PS5

Statement	COTS	N	Median	Rank	Z
PS1: I found it hard to come up with good requirements versus COTS	Robot	20	3.000	32.9	-0.17
	Game	46	3.000	33.8	0.17
	Overall	66		33.5	
	H = 0.03 DF = 1 P = 0.862				
	H = 0.03 DF = 1 P = 0.855 (adjusted for ties)				
Statement	**COTS**	**N**	**Median**	**Rank**	**Z**
PS2: I think the COTS did not hinder the design of a good architecture versus COTS	Robot	20	4.000	41.8	2.31
	Game	46	3.000	29.9	-2.31
	Overall	66		33.5	
	H = 5.33 DF = 1 P = 0.021				
	H = 5.79 DF = 1 **P = 0.016** (adjusted for ties)				
Statement	**COTS**	**N**	**Median**	**Rank**	**Z**
PS3: I found it difficult to evaluate the other team's architecture in the ATAM versus COTS	Robot	20	3.000	39.2	1.59
	Game	46	2.000	31.0	-1.59
	Overall	66		33.5	
	H = 2.53 DF = 1 P = 0.112				
	H = 2.76 DF = 1 P = 0.097 (adjusted for ties)				
Statement	**COTS**	**N**	**Median**	**Rank**	**Z**
PS4: I think the COTS made it easier to identify architectural drivers versus COTS	Robot	20	3.000	32.0	-0.41
	Game	46	3.000	34.1	0.41
	Overall	66		33.5	
	H = 0.17 DF = 1 P = 0.681				
	H = 0.20 DF = 1 P = 0.654 (adjusted for ties)				
Statement	**COTS**	**N**	**Median**	**Rank**	**Z**
PS5: I found it difficult to focus on our assigned quality attribute versus COTS	Robot	20	3.000	36.9	0.94
	Game	46	2.000	32.0	-0.94
	Overall	66		33.5	
	H = 0.89 DF = 1 P = 0.346				
	H = 0.95 DF = 1 P = 0.329 (adjusted for ties)				

Table III shows the results of Kruskal-Wallis tests where the statements PS7, PS8 and PS10 had significant difference with $p \leq 0.05$. The most noticeable result is for PS7 where

the students doing the Robot project claim to have spent *significantly more time on technical matters* than the students doing the Game project (p=0.001). This result was unexpected as the XNA framework is much more extensive than the Khepera robot simulator, and C# had to be learned. One explanation to this response can be that the students found it very difficult to program the navigation of the robot in a maze utilizing the sensors of the robot. The results from PS8 show that the students doing the Robot project to a larger degree responded to have *spent too much time trying to learn the COTS* at the start of the course compared to the students doing the Game project. This result was even more unexpected than the result for PS7, as the XNA framework and C# is much more complex than the Khepera robot simulator. Two possible explanations can be that the students doing the Game project were better motivated to learn the COTS or simply that the documentation for XNA was better than the Khepera robot simulator. From the result of the test, we can also see that there is a tendency that Game project students found it easier than the Robot project students to integrate architectural and design patterns with the COTS (p=0.114), and that the Robot students perceived that they have learned more about software architecture from the projects compared to the Game students (p=0.187).

In statement 10 (PS10), the students were asked if they would have chosen the other project next time. Table III shows that there is a statistically significant difference between the Robot and the Game projects (p=0.028).

Table III. Kruskal-Wallis test of the statements PS6-PS10

Statement	COTS	N	Median	Rank	Z
PS6: I found it easy to integrate known architectural or design patterns versus COTS	Robot	20	3.000	38.8	1.48
	Game	46	2.500	31.2	-1.48
	Overall	66		33.5	
	H = 2.19 DF = 1 P = 0.139				
	H = 2.50 DF = 1 P = 0.114 (adjusted for ties)				

Statement	COTS	N	Median	Rank	Z
PS7: I spent more time on technical matters than on architectural matters versus COTS	Robot	20	1.500	21.9	-3.23
	Game	46	3.000	38.5	3.23
	Overall	66		33.5	
	H = 10.43 DF = 1 P = 0.001				
	H = 11.18 DF = 1 **P = 0.001** (adjusted for ties)				

Statement	COTS	N	Median	Rank	Z
PS8: I spent too much time trying to learn the COTS at the start of the course versus COTS	Robot	20	2.000	26.7	-1.90
	Game	46	3.000	36.5	1.90
	Overall	66		33.5	
	H = 3.63 DF = 1 P = 0.057				
	H = 3.90 DF = 1 **P = 0.048** (adjusted for ties)				

Statement	COTS	N	Median	Rank	Z
PS9: I have learned a lot about software architecture during the project versus COTS	Robot	20	2.000	29.0	-1.25
	Game	46	3.000	35.4	1.25
	Overall	66		33.5	
	H = 1.56 DF = 1 P = 0.212				
	H = 1.74 DF = 1 P = 0.187 (adjusted for ties)				

Statement	COTS	N	Median	Rank	Z
PS10: I would chosen the other project (game instead of robot and vice versa)	Robot	20	1.000	38.4	1.37
	Game	46	1.000	31.4	-1.37
	Overall	66		33.5	
	H = 1.87 DF = 1 P = 0.172				
	H = 4.85 DF = 1 **P = 0.028** (adjusted for ties)				

Fig. 3 shows the distribution of the students' responses for PS10. Here there is a much higher percentage of the Robot project students that would have chosen the other project (30%) compared to the Game project students (9%).

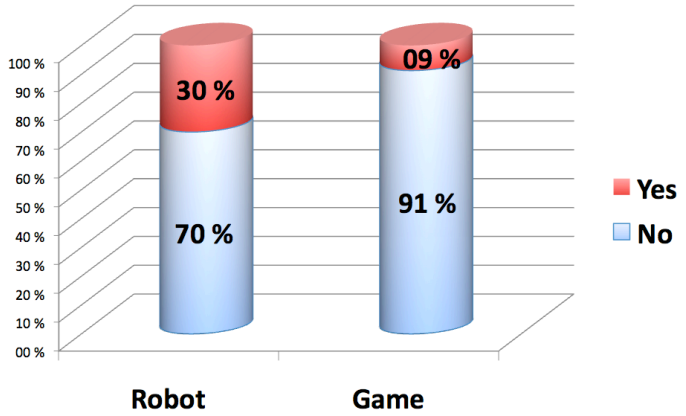

Fig. 3. Response to PS10: Would have chosen another project

4.3 RQ3: Differences in the Design of Software Architectures

It is difficult to evaluate software architectures empirically, but we have chosen to do so by comparing the number of *architectural and design patterns* the students used, the *number of main modules/classes* identified in the logical view of the software architecture, and the *number of hierarchical levels* in the architecture. We admit that that there are many sources of errors in this comparison, as the two domains are so different. However, the emphasis in this course is on using software architecture and design patterns and presenting the different views of the software architecture in sufficient detail with emphasis on the logical view. The empirical data should highlight the differences between the two types of projects if any. The empirical data has been collected by reading through and analyzing the final project reports from 6 Robot project teams and 16 Game project teams. Note that the process and document templates are the same for both types of projects, so any noticeable differences were not expected.

Use of Architectural and Design Patterns

Table IV presents the descriptive statistics of the number of architectural and design patterns used in the Robot and the Game projects. The table indicates that there are some differences between the two types of projects.

Table IV. Number of architectural patterns and design patterns used

	Architectural patterns		Design patterns	
	Robot	**Game**	**Robot**	**Game**
Average	1.00	1.63	1.0	1.13
Standard deviation	0.00	0.72	1.67	1.20
Max	1	3	4	3
Min	1	1	0	0

Table V presents Kruskal-Wallis tests to examine if there are any statistically significant differences in the number of architecture and design patterns produced by the two different project types. The results show that *for architectural patterns, there is a statistically significant difference* (p=0.037). The main difference from looking at the actual architectures and implementations was found to be that the Game projects were larger and contained more complex structures. Five out of six teams working with Robot projects chose different architectural patterns notably; Control loop (2 teams), Layers, Subsumption, Switchboard, and Elfes.

Table V. Hypothesis tests on number of patterns used

Hypothesis	COTS	N	Median	Rank	Z
No difference in number of used Architecture patterns	Robot	6	1.000	7.5	-1.73
	Game	16	1.500	13	1.73
	Overall	22		11.5	
	H = 3.13 DF = 1 P = 0.0836				
	H = 4.33 DF = 1 **P = 0.0374** (adjusted for ties)				

Hypothesis	COTS	N	Median	Rank	Z
No difference in number of used Design patterns	Robot	6	0.000	10.4	-0.44
	Game	16	1.000	11.9	0.44
	Overall	22		11.5	
	H = 0.23 DF = 1 P = 0.66				
	H = 0.27 DF = 1 P = 0.66 (adjusted for ties)				

Fig. 4. The distribution of chosen architectural patterns for Game projects illustrates the distribution of the architectural patterns used in Game projects. Half of the teams that worked with Game projects have used the model view controller pattern and nearly quarter of the teams have used the pipe & filter pattern. Note that some of these patterns can also be denoted design patterns, but the students have described them as architectural patterns if they have been used to form the main structure of the software architecture.

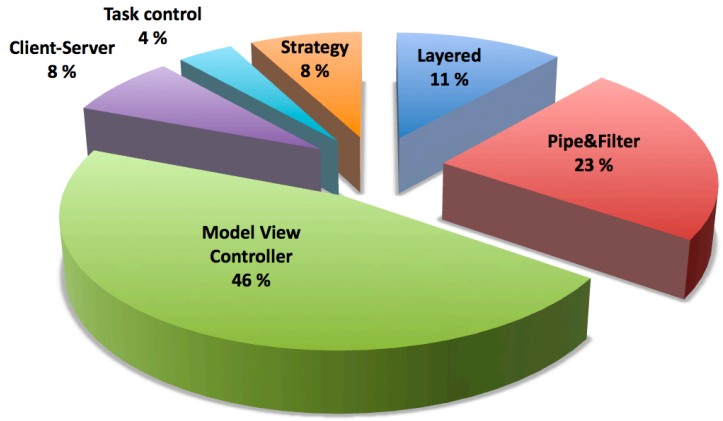

Fig. 4. The distribution of chosen architectural patterns for Game projects

Table IV also gives the number of design pattern used by the two different groups (Robot vs. Game). The table shows that the Game project teams have on average 13% higher number of design patterns (1.13) compared to the Robot teams (1.0). However, Table V indicates no statistically significant difference for the number of design pattern used for the two types of projects. From reading through the projects reports we found that only two of six teams that worked with the Robot projects (33.33%) documented the use of design patterns, while nine of sixteen teams that worked with Game projects (56.25%) did so. This result was unexpected, as the teams consist of four members and usually one of the team members stresses the use of design patterns. We do not know the reason for this difference, but we suspect that the Game teams were more implementation-oriented and thus were more interested in structuring the code using patterns. The difference between the two domains does not explain why it should be easier to utilize design patterns for games rather than for robot controllers.

Fig. 5 presents the distribution of design patterns used by Robot teams (to the left) and by Game teams (to the right). The charts show that the *Observer*, the *Abstract factory* and the *State* patterns were the most popular for both types of project. Further, that the *Singleton* pattern was among the top three for Game teams.

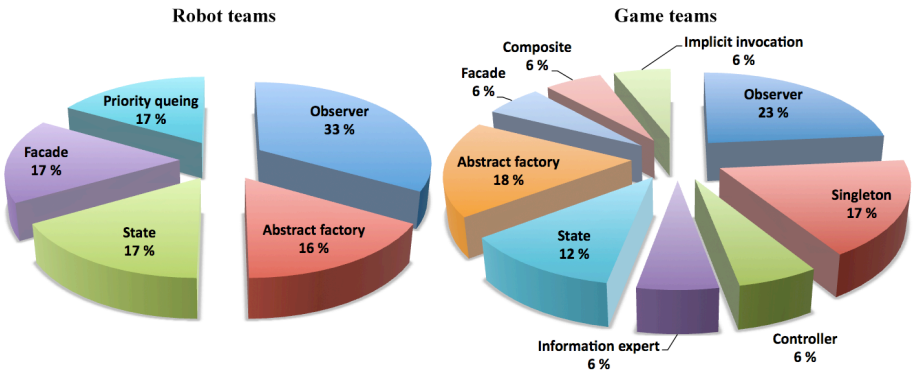

Fig. 5. Distribution of usage of design patterns for Robot and Game teams

Software Architecture Complexity

It is hard to find one metric to measure the complexity of a type of software architecture. Thus two metrics were chosen to indicate such complexity: 1) The number of main modules or main classes described in the logical view of the software architecture, and 2) The number of hierarchical levels in the model presented in the logical view of the software architecture. The reason the logical view was chosen for computing complexity is that the logical view is the main one that gives the best overview of the designed architecture. Table VI lists the measurements of the number of main modules/classes and the number of hierarchical levels in the logical view of the software architecture for Robot and Game projects. The table shows that the Game project teams on average have almost three more main modules/classes (25%) than the Robot teams. The difference in the maximum number of main modules/classes is six (32%). Further, there is on average 14% higher number of levels in the architecture of Game projects compared to Robot projects.

Table VI. Measurements of software architecture complexity

	Number of main modules/classes		Number of levels in architecture	
	Robot	**Game**	**Robot**	**Game**
Average	8.67	11.63	1.50	1.75
Standard deviation	3.39	4.36	0.55	0.77
Max	13	19	2	4
Min	5	5	1	1

Table VII gives the results from Kruskal-Wallis tests on a number of main modules/classes and numbers of levels in the architecture. The tests show that there is no statistically significant difference between the two types of projects, although the difference in number of main modules/classes has a rather low p value (p=0.13).

Table VII. Hypothesis tests on architectural complexity

Hypothesis	COTS	N	Median	Rank	Z
No difference in number of main modules/classes	Robot	6	8.500	8.1	-1.47
	Game	16	11.500	12.8	1.47
	Overall	22		11.5	
	H = 0.26 DF = 1 P = 0.1416				
	H = 2.31 DF = 1 P = 0.1283 (adjusted for ties)				
Hypothesis	**COTS**	**N**	**Median**	**Rank**	**Z**
No difference in number of levels in architecture	Robot	6	1.500	10.3	-0.52
	Game	16	2.000	12.0	0.52
	Overall	22		11.5	
	H = 0.03 DF = 1 P = 0.6031				
	H = 0.40 DF = 1 P = 0.5288 (adjusted for ties)				

4.4 RQ4: Differences in the Effort put into the Project

We have no hard number or estimates on how many hours the project teams worked during the software architecture project, so we needed to make an estimate. The project reports did not reveal large differences in terms of quantity and quality that could indicate

that students choosing one type of project worked more than the other. We chose to look at metrics from the implementation to give an estimate on how much effort was put into the project. This is not a perfect measure, but it should give a good indication of the complexity of the software architecture and the resulting implementation of the application. Both COTS (Khepera simulator and XNA) provide high-level APIs approximately at the same abstraction level, and Java and C# are comparable programming languages with similar characteristics. The students were not given any code base or code examples apart from that being given by the COTS. However, there are a lot more code examples available for XNA compared to Khepera. The following metrics were chosen to compute the effort of the student teams:

- Number of source Files (NoF)
- Number of Comments in code (NoC)
- Lines of source Code not counting empty lines or comments (LoC)

The following metrics that can indicate how the code was structured and how the code is commented can be computed from the above:

- Lines of code per File (LpF): `[LpF = LoC / NoF]`
- Lines of code per Comment (LpC): `[LpC = LoC / NoC]`

Table VIII presents a comparison of the implementation metrics for the Robot and the Game projects.

Table VIII. Implementation metrics from the architecture projects

	Number of files (NoF)		Number of Comments (NoC)		Lines of Code (LoC)		LoC per File (LpF)		LoC per Comment (LpC)	
	Robot	**Game**	**Robot**	**Game**	**Robot**	**Game**	**Robot**	**Game**	**Robot**	**Game**
Avr	31	40	345	758	1798	3396	78	86	8	8
StD	19.7	29.0	260.1	716.5	568.5	2617.1	46.9	26.0	7.6	5.9
Max	58	118	779	2374	2453	11759	156	144	20	17
Min	10	14	77	47	853	802	30	52	2	2

Table IX shows the results from Kruskal-Wallis tests on the difference in the number of files and the number of lines of code produced by the two different types of project.

Table IX. Hypothesis tests on project effort

Hypothesis	COTS	N	Median	Rank	Z
No difference in number of files	Robot	6	29.500	10.3	-0.52
	Game	16	34.000	12.0	0.52
	Overall	22		11.5	
	H = 0.30 DF = 1 P = 0.6031				
	H = 0.31 DF = 1 P = 0.5797 (adjusted for ties)				

Hypothesis	COTS	N	Median	Rank	Z
No difference in number of lines of code	Robot	6	1833.5	7.8	-1.59
	Game	16	2722.0	12.0	1.59
	Overall	22		11.5	
	H = 2.04 DF = 1 P = 0.1118				
	H = 2.63 DF = 1 P = 0.1048 (adjusted for ties)				

The results from the Kruskal-Wallis tests indicate that there is no statistically significant difference between the two types of project, although there is a relative low p-value for the difference in number of lines of code (p=0.10). Table VIII gives two results: *first*, the Game project teams have produced on average almost twice as much code (190% more) and *second*, the variation in LoC is much higher for Game projects (460% higher). The great variation in the LoC written by Game projects is shown as this project type has the team that has produced *most* lines of code as well as the team that has produced *least* lines of code. The main difference between the two types of project is that the most productive Game teams have implemented significantly more than the Robot teams (almost 12000 LoC vs. 2500 LoC). This phenomenon can be explained by the simple fact that the students get carried away with their game projects by adding new gameplay elements and features. A study of the code of the student projects reveals that Game projects on average contain more code directly related to the architecture compared to Robot projects in addition to code related to game features. Another possible explanation of the tendency of game projects producing more code can be that a lot of XNA code is available on the Internet. As far as we could tell from the code in the student projects, little of the code was taken from other sources as all game concepts were original and they had to focus on the software architecture through the whole project.

Another noticeable tendency is that Game teams put more lines of code in each file and that there is less variation between the Game teams in respect to how much code they put in each file. Finally, there is little difference in how teams from the two types of projects comment on the source code.

4.5 RQ5: Differences in the Project and Course Grades

As mentioned, the grade awarded in the software architecture course is computed by combining the group grade on the project (30%), and the individual grade on the written examination (70%). Ideally, these two components in the course should have counted 50% for each part to give appropriate credit to how much effort the students put into the project. However, the regulations at the university do not allow project work to be credited more than 30% of the grade when combined with a grade from a written examination. The grading system at NTNU suggests the following template for grading courses:

- A: Score ≥90%
- B: Score ≥80% and score<90%
- C: Score ≥60% and score<80%
- D: Score ≥50% and score<60%
- E: Score ≥40% and score<50%
- F: Score<40% (fail).

Since the grade on the project is given to groups and the grade on the written examination for the individual, these two components cannot be compared directly. We have chosen to investigate if there were any differences in how the group scored (0-30 points) on the project, and the individual score (0-70 points) on the written examination for students that have chosen Game and Robot projects. The Kruskal-Wallis Test was also used here to test if there were any statistically significant differences between the two types of projects (Robot and Game), as we cannot assume a normal population and the sample size of the two groups is different. Table X presents the results of the Kruskal-

Wallis test on the difference in project grades for Robot and Game groups, and the individual examination grades given to students doing Robot and Game projects.

Table X. Kruskal-Wallis Test on difference in project score

Hypothesis	COTS	N	Median	Rank	Z
No difference in project score groups get from doing Robot vs. Game project	Robot	6	24.000	9.8	-0.74
	Game	16	25.000	12.2	0.74
	Overall	22	25.000	11.5	
	H = 0.60 DF = 1 P = 0.4593				
	H = 0.61 DF = 1 P = 0.4324 (adjusted for ties)				
Hypothesis	**COTS**	**N**	**Median**	**Rank**	**Z**
No difference in the examination score students get from doing Robot vs. Game project	Robot	21	49.000	46	1.26
	Game	59	44.000	38.5	-1.26
	Overall	80	46.000	40.5	
	H = 1.61 DF = 1 P = 0.2077				
	H = 1.60 DF = 1 P = 0.2058 (adjusted for ties)				

Table X shows that there is no statistically significant difference in the scores for the two types of projects. The p-value of the examination score was rather low (p=0.21) indicating that there is a tendency that students doing Robot projects do better on the final examination. Considering the median, the Game project score is only slightly higher than the Robot project grade (4% higher), while the examination score for students doing the Robot project is 11% higher than Game projects. Fig. 6 illustrates the distribution of final grades (the project and written examination combined) for the students that worked with Robot projects vs. Game projects. The chart shows that there are only minor variations between the two groups.

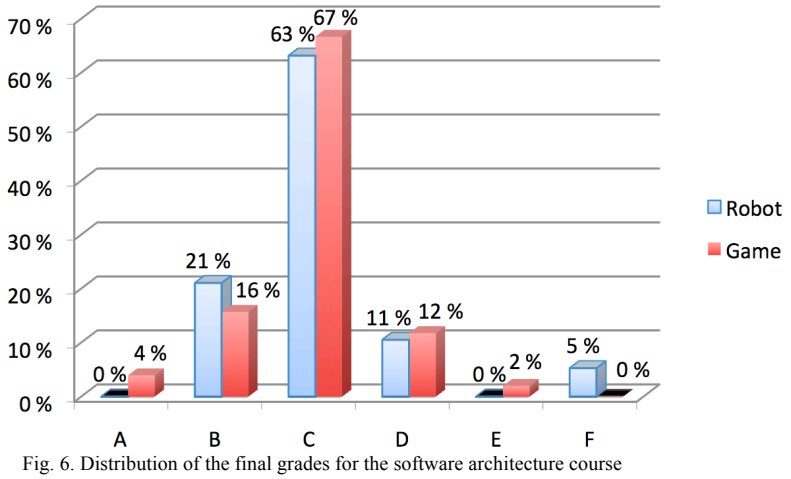

Fig. 6. Distribution of the final grades for the software architecture course

Fig. 7 gives the distribution of grades on the project and the written examination respectively for the two types of projects (Robot vs. Game). Fig. 7 visualizes a tendency that Game project students do better in the project and Robot project students do better in

the written examination. Note that the comparison of project grades shown in Fig. 7 gives the distribution of grades of *individual students*, while the Kruskal-Wallis test was performed on *differences between groups*.

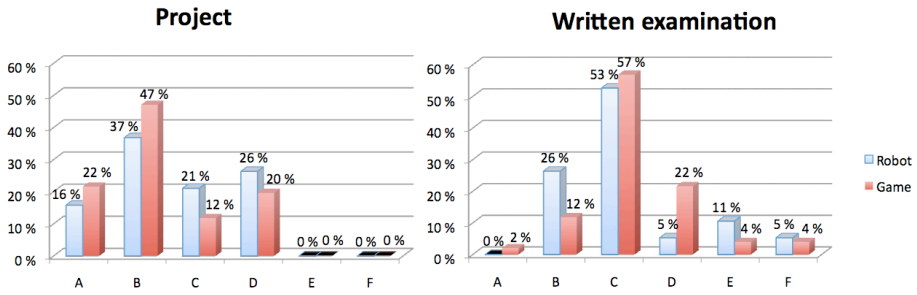

Fig. 7. Distribution of grades on the project and the written examination

5. DISCUSSION

This section discusses the results presented in previous section and discusses some threats to validity.

5.1 Discussion of the Results

The results presented in Section 4 gave strong indications that student are motivated by game development projects. The popularity of the game project shows for our case that the majority of students prefer to work with game development projects. The students also seem motivated to put more effort into a game development project compared to another type of project, which gave good results in terms of using course related techniques and methods and produce proper documentation and implementation. The results also indicate that the students choosing Game projects are more eager to utilize software engineering practices like architectural patterns to improve the final product. However, the evaluation results do not show that the students get better grades from being motivated by the game project. Our results do not reveal any significant differences between the grades students get on the project and/or the written examination based on their selection of type of project. We noticed a weak tendency that Robot project students do better on the written examination, but this result is not statistically significant.

Our data show that the Game project students put more effort into the project, but the extra effort does not give higher final grades in the course. Through years of experience teaching the software architecture course, we have seen that the things the students had to learn to be able to do the project were forgotten before the final written examination. As the projects are carried out by teams, there is always a danger that only few students in a team read and learn the necessary skills to document and do the project, while other team members only focus on other things such as programming. Another potential problem with the game projects is that students focus too much on the game itself instead on issues related to software architecture. If we just consider what the students have delivered and documented in the project, game projects work very well for learning software architecture in practice. The final game project reports were at least as good as the ones from robot projects especially on the use of design and architectural patterns and the design of the software architecture (the emphasis of the course). If game projects are to be used in other CS or SE courses, it is very important that the stated learning

objectives from the project are enforced through evaluation criteria and delivery templates.

To further investigate differences in grade variations between Robot project students vs. Game project students, we analyzed the difference between project and examination grades for every student. This analysis was carried out to examine our belief that Game project students performed better in their project compared to the written examination, and look at the relationship between the two subcomponents in the final grade. In the analysis the students were classified into the following three groups:

- **Same grade**: Students with the *same grade* on the project and the written examination (e.g. B in both the project and the written examination)
- **Proj>Exam**: Students with a higher grade on the project than the written examination (e.g. B in the project and C in the written examination)
- **Exam>Proj**: Students with a higher grade on the written examination than the project (e.g. C in the project and B in the written examination)

Fig. 8 shows the distribution of Game project and Robot project students classified as above. The figure describes the tendency that Game project students (compared to Robot project students) to a larger degree were awarded a higher grade on the project than the written examination (76% for Game project students vs. 63% for Robot project students). Further, that 16% of the Robot project students were awarded a higher grade on the written examination than the project compared to the 8% of the Game project students.

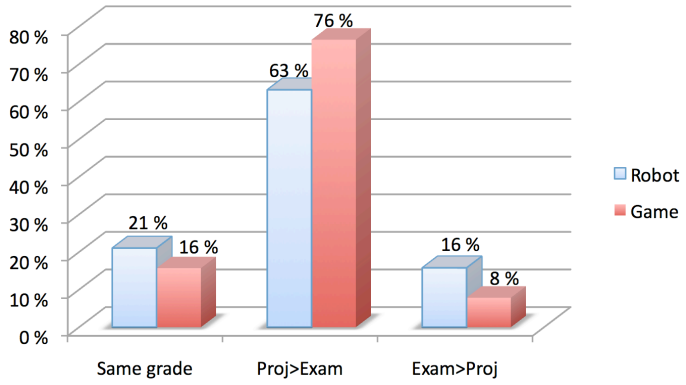

Fig. 8 Distribution of grade differences on the project and the written examination

The Kruskal-Wallis test was used to see if there were any statistically significant differences between the two groups. The data used in this test not only included the classification of students as shown above, but the amount of difference between the project and the exam grades. The results of the Kruskal-Wallis tests were as follows:

- $Grade_{Project} > Grade_{Exam}$: H=15.695 DF=1 p=0.0001
- $Grade_{Exam} > Grade_{Project}$: H=1.559 DF=1 p=0.2117

The results show there is a statically significant difference between Game project students getting a higher grade on the project than the examination compared to the Robot project students (p=0.0001). The main difference between the two domains is that

15% more Game project students compared to Robot project students have a grade difference of two or more grades between the project and the examination (e.g. B on the project and D or lower on the examination). The p-value regarding students getting a higher grade on the examination than the project is not statistically significant, but it shows the tendency that Robot project students to a higher degree than Game project students score higher on the examination than the project.

The results from evaluating the use of a game project in a software architecture course are both positive and negative. The *positive effect* is that students are better motivated for the project and put more effort into it. This is shown through the utilization of architecture and design patterns, the complexity of the software architecture, and the size of the implemented application. The *negative effect* observed is that this extra motivation of students does not necessarily improve their final grade. To counter this negative effect, it is important to make the students learn more theory from the project and convince the students as to why software architecture theory is important for game development. A possible approach could be to give more examples of how software architectures are used in game development projects, and have guest lectures from the game industry emphasizing the need for software architecture in game development. However, to succeed with such integration, it is crucial that the theoretical topics in the course are better integrated with the game project. Further, it is important that the course staff clarify how game development and software architecture are related in the syllabus.

5.2 Threats to validity

We now turn to what are considered to be the most important threats to the validity of this evaluation.

Internal Validity

The internal validity of an experiment concerns "the validity of inferences about whether observed covariation between A (the presumed treatment) and B (the presumed outcome) reflects a causal relationship from A to B as those variables were manipulated or measured" [Shadish et al., 2002]. If changes in B have causes other than the manipulation of A, there is a threat to internal validity. Although our evaluation cannot be described as a controlled experiment, it is worth considering some of the most evident internal validity concerns.

There are two main internal validity threats to this evaluation. The *first internal threat* is that the sample of two groups used in the evaluation (Robot and Game) is not randomized. The students were allowed to choose either a Robot or a Game project. We do not believe that one specific type of student chose one project over the other, thus harming the evaluation results. The demographic data did not reveal any major difference between the two groups. The *second internal threat* is if there were any differences how the students had to perform the project independently of the domain chosen. Independently of doing a Robot or a Game project, the students had to go through exactly the same phases in the project and deliver exactly the same documents based on the same document templates. We have *identified two differences* in how the two types of projects were carried out. The first phase of the project was different for the two types of projects. The students doing the Robot project had an exercise where they had to make the robot do simple navigation and pick up balls, while the students doing the Game project had to make sprites move and change and implement the Pong game in XNA. This exercise was not a part of the data and material used to evaluate the project. In addition, the way we

evaluated the implementation was different for the two types of projects, as the Robot project had fixed requirements while the Game project did not. We do not believe that these differences have had any major impact in the way the students did or performed in their projects.

Construct validity

Construct validity concerns the degree to which inferences are warranted, from (a) the observed persons, settings, and cause and effect operations included in a study to (b) the constructs that these instances might represent. The question, therefore, is whether the sampling particulars of a study can be defended as measures of general constructs [Shadish et al., 2002].

In the evaluation of using a game project in a software architecture course our research goal was to investigate whether a game development project was suited for teaching software architecture. The GQM approach was chosen to detail this goal into five research questions with supporting metrics. In order to give answers to these five research questions the data sources and metrics available from our software architecture course were chosen. It cannot be claimed that the selected data sources and metrics in our evaluation give evidence for all the conclusions, but they are all strong indicators contributing to a picture that describes the differences between the two project types. Through the evaluation we have used various methods for comparing the results. The choice of methods is based on the best way of describing and visualizing the differences between the two groups using the available data.

External validity

The issue of external validity concerns whether a causal relationship holds (1) for variations in persons, settings, treatments, and outcomes that were in the experiment and (2) for persons, settings, treatments, and outcomes that were not in the experiment [Shadish et al., 2002].

The results reported in this paper are *most relevant* for other teachers thinking of introducing game projects as a part of their software architecture course. Further, the results are also relevant for teachers that want to introduce game projects in SE and CS courses, as many of these courses have similar characteristics. A limitation of this study is that the subjects in the evaluation are CS or SE students that have completed their first three years. It is not evident that the results are valid for students without any or less than three years background in CS or SE.

6. COMPARISON WITH PREVIOUSLY PUBLISHED RELATED WORK

This paper describes an evaluation of integrating a game project using XNA and C# in a software architecture course. The main benefits from using a game project are that the students get more motivated during the software development project. There are only few papers available that describe an evaluation of using game projects in CS or SE courses. This section describes the work presented along with papers that describe the integration of games and CS/SE courses in general.

Youngblood [2007] describes how XNA game segments can be used to engage students in advanced computer science education. Game segments are developed solution packs providing the full code for a segment of a game with a clear element left for implementation by a student. The paper describes how XNA was used in a 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 that are specific for using game segments in CS education based on lessons learned. It would be possible to use game segments in our software architecture course as well, but this approach would be likely to limit the architecture components too much.

El-Nasr and Smith [2006] describes how the use of modifying or modding existing games can be used to learn computer science, mathematics, physics and ascetic principles. The paper describes how they used modding of the WarCraft III engine 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. The context of this paper is very different from ours, as the students are focusing on game design and not game architecture design or implementation of the game from scratch.

Sweedyk and Keller [2005] describe how they introduced game development in an introductory SE course. The students learned principles, practices and patterns in software development and design through three projects. In the *first project*, the students were asked to develop a 2D arcade game with a theme based on campus life using the POP framework over four weeks. The educational focus of the first project was to gain 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 built a one-hole miniature golf game over five weeks. The educational focus of the second project was on learning and practicing evolutionary design, prototyping and re-factoring, 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 developed a game of their own choice over five weeks. In this phase, the learning objectives were to reinforce 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 of this paper was extremely positive. They argue that game projects allow them to better achieve the learning objectives in the SE course. Their main concern was related to gender, as women were less motivated to learn SE through game development projects. The main difference with Sweedyk and Keller's approach and ours was 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 learning and miss the opportunity to follow the evolution of the software architecture through one project.

Kajal and Mark Claypool [2005] describe another SE course where a game development project was used to engage the students and make the course more fun. 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 higher average grades, a higher distribution of A grades, and had a lower number of dropouts. The feedback from the students was also very positive. The results found here are very similar to the results reported from our work. The paper does not detail what was graded in the course.

Volk [2008] describes how a game engineering course was integrated into a CS curriculum 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 experience from running this course showed that it was a good idea to 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 pre-production to production was difficult (extracting the requirements), and that some student teams were overambitious about 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: problems with extracting requirements and overambitious teams.

Linhoff and Settle [2008] describe a game development course where the XNA platform was used to allow the students to gain experience in all aspects of console game creation. The course focused on the creation 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 were required to edit the source code of a game to change variables, and copy-and-paste code. The students' general response to the course was positive. The results also showed that students with a programming background did better in the class. The focus of this study was very different from ours, as the focus was not on architecture and programming rather on game content development.

Zhu, Wang and Tan [2008] describe how games can be introduced in SE courses to teach typical SE skills. The paper described how the two games SimSE and MO-SEProcess were used to give students an opportunity to practice SE through simulations. In SimSE, the students can practice a "virtual" SE process in a fully interactive way with graphical feedback that enables them to learn the complex cause and effect relationships underlying the process of SE. MO-SEProcess is a multiplayer online SE process game based on 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 were much more extensive than the ones in MO-SEProcess, the use of Second Life brought some advantages. Among these were better support for group sharing and collaboration and it made it possible to create interactive learning experiences that would be hard to duplicate in real life. The focus in this paper was also different from ours, as it focused on use of games in a SE course and not game development per se.

Rankin and Gooch [2008] describe a study on how a game design project impacts on students' interest in CS. In a Computer Science Survey course, the students were given the task to apply SE principles in the context of game design. The pre and post survey results revealed that game design can have both positive and negative impacts on students' attitudes concerning enrollment in a game design course, pursuit of a CS degree, further development of programming skills and enrollment in additional CS courses. The post survey showed a 100% increase for students that previously had not interest in game design, but an overall 70% decrease in the interest in game design. The interest for pursuing a CS degree dropped from 50% to 30% after the game design project. Further, 25% of the participants indicated an increased interest in further developing their programming skills, while 40% demonstrated reduced interest. Finally, 20% of the students indicated they were less likely to enroll in CS classes, while 15% indicated the opposite. The focus of this paper is on game design and not game architecture and game implementation. The results described in this paper are very different from the results we found in our evaluation.

Ryoo et al. [2008] describes an approach for teaching Object-Oriented Software Engineering (OOSE) through problem-based learning in the context of a game project. The OOSE concepts are taught through a group project going through several phases:

inception, elaboration, construction, and transition. The focus in the course is on documenting a game using UML and implementing a prototype using Java. The approach has not been evaluated.

7 CONCLUSIONS

This paper has evaluated the introduction of a Game project in a software architecture course using an existing Robot project as a benchmark. The goal of this evaluation was to get answers to five research questions.

The *first research question* asked if game projects are popular among students and if there were any specific groups preferring game projects (RQ1). The results showed that three out of four students chose the game project the first time this type of project offered. There were not major differences in how female vs. male, or Norwegian vs. foreign students chose their project type.

The *second research question* asked if there are any differences in how students choosing Robot vs. Game projects perceived the software architecture project (RQ2). The statistically significant findings ($p \leq 0.05$) were that students that worked with Robot projects to a larger degree thought that the COTS (the robot simulator) hindered the design of good architecture, that more time was spent on technical than architectural issues, and that too much time was used in the beginning of the course to learn the COTS. Some less significant findings ($p \leq 0.11$) revealed that students doing a Game project to a larger degree thought it was more difficult to evaluate other teams using ATAM, and it was easy to integrate known architecture and design patterns. Further, the students doing a Robot project to a larger degree ($p=0.19$) claimed to have learned more about software architecture during the project. Finally, the results showed that 30% of the students doing a Robot project would have chosen the Game project if they had to do the project again.

The *third research question* asked if there are any differences in how students choosing Robot vs. Game projects designed their software architectures (RQ3). The analysis of the project reports showed that students doing a Game project utilized architectural patterns to a larger degree ($p=0.04$). The most popular design patterns used by all students were the Observer, the Abstract factory and the State pattern. Finally, the evaluation found indications that the software architectures produced in Game projects were on average more complex than the architectures produced in Robot projects (but not statistically significant, $p=0.13$).

The *fourth research question* asked if there were any differences in the effort the students put into the project when they worked with a Robot or a Game project (RQ4). Since, we did not have the actual number of hours the teams worked, we compared the two different project types by comparing the implementation. The results show that teams working with Game projects produced on average almost twice as much code as teams working with Robot projects (not statistically significant, $p=0.10$). The results also showed that Game projects on average put more source code in each source file and the amount of commenting was about the same for both types of projects.

The *fifth and final research question* asked if there are any differences in the performance for students doing a Robot project vs. students doing a Game project (RQ5). The comparison of the two types of projects showed that there was no statistically significant difference in the project and examination grades for students doing Game projects vs. students doing Robot projects. However, the comparison indicated that students doing Game projects performed better on the project on average, while students doing Robot projects performed better on the written examination on average. Further,

we found that Game project students to a larger degree than Robot project students were awarded with higher grades in their project than in the written examination.

The goal of the work described in this paper was to evaluate the use of a game development project in a software architecture course. The Robot project was used as a benchmark for a successful project we have run for five years. Our results show that a game development project can successfully be integrated into a software architecture course. The most notably positive effect is that students are clearly motivated by game projects which likely resulted in higher enrollments and more effort put into the project. Further, that the improved motivation can improve the use of software engineering practices such as use of architectural patterns, and higher programming productivity. However, the improved motivation does not necessarily result in better grades in the final examination.

ACKNOWLEDGEMENTS

I would like to thank Jan-Erik Strøm and Trond Blomholm Kvamme who were students at my department for conducting the project survey. I am also grateful to Erik Arisholm at the Simula Research Laboratory for helping out with statistical testing of data. I 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. Further, I acknowledge the assistance from Stewart Clark at the Norwegian University of Science and Technology for improving the language. The Leiv Eriksson mobility program supported by the Research Council of Norway has sponsored this work. Finally, I wish to thank the reviewers for giving very constructive feedback that was used to improve the paper.

REFERENCES

AHN, L.V. 2006. Games with a Purpose. IEEE Computer Magazine: 39(6), June, 92-94.

ANDERSON, E. F., ENGEL, S., COMNINOS, P., AND MCLOUGHLIN, L. 2008. The case for research in game engine architecture. In Proceedings of the 2008 Conference on Future Play: Research, Play, Share, Toronto, Ontario, Canada, November 03 - 05, pages 228-231.

BAKER, A., NAVARRO, E. O., AND HOEK, A.2003. Problems and Programmers: an Educational Software Engineering Card Game. In Proceedings of the 25th International Conference on Software Engineering (ICSE 2003), pages 614–619.

BASILI, V. R., CALDIERA, G. AND ROMBACH, H. D. 1995. Goal Question Metric Paradigm. In Encyclopedia of Software Engineering, Volume 1, John Wiley & Sons, pages 527-532.

BINSUBAIH, A. AND MADDOCK S.C. 2006. Using ATAM to Evaluate a Game-based Architecture. Workshop on Architecture-Centric Evolution (ACE 2006), Hosted at the 20th European Conference on Object-Oriented Programming ECOOP 2006, July 3-7, Nantes, France.

BOOCH, G. 2007. Best Practices in Game Development. IBM Presentation March 12.

BLOW, J. 2004. Game Development: Harder Than You Think. In Queue: 1(10), February 28-37.

CALLELE, D., NEUFELD, E., AND SCHNEIDER, K. 2008. Emotional Requirements. IEEE Software Magazine 25(1), January 43-45.

CALTAGIRONE, S., KEYS, M., SCHLIEF, B., AND WILLSHIRE, M. J. 2002. Architecture for a massively multiplayer online role playing game engine. Journal of Computing Sciences in Colleges 18(2), December 105-116.

CHEN, W.-K. AND CHENG, Y.C. 2007. Teaching Object-Oriented Programming Laboratory With Computer Game Programming. IEEE Transactions on Education 50, 197-203.

CLAYPOOL, K. AND CLAYPOOL, M. 2005. Teaching software engineering through game design. In Proceedings of the 10th Annual SIGCSE Conference on innovation and Technology in Computer Science Education (ITiCSE '05), Caparica, Portugal, 123-127, June 27 – 29.

CLEMENTS, P., BASS, L. AND KAZMAN, R. 2003. Software Architecture in Practice Second Edition. Addison-Wesley.

COPLIEN, J.O. 1998. Software Design Patterns: Common Questions and Answers. The Patterns Handbook: Techniques, Strategies, and Applications. Cambridge University Press, New York, 311-320.

DANIAL. A. 2009. CLOC – Count Lines of Code. Web: http://cloc.sourceforge.net/, Accessed March 12th 2009.

DARKEN, R., MCDOWELL, P. AND JOHNSON, E. 2005. The Delta3D Open Source Game Engine. In IEEE Computer Magazine, May/June.

DELDEN, S.V. AND ZHONG, W. 2008. Effective integration of autonomous robots into an introductory computer science course: a case study. J. Comput. Small Coll. 23, 10-19.

DISTASIO J. AND WAY T. 2007. Inclusive computer science education using a ready-made computer game framework. In ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education, 116-120.

EL-NASR, M. S. AND SMITH, B.K. 2006. Learning through game modding. In ACM Computer Entertainment 4(1), Jan.

ELFES, A. 1987. Sonar-Based Real-World Mapping and Navigation. In IEEE Journal of Robotics and Automation, no.3, 249-265.

FALTIN, N. 1999. Designing courseware on algorithms for active learning with virtual board games. SIGCSE Bull. 31, 135-138.

FLOWERS, T.R. AND GOSSETT, K.A. 2002. Teaching problem solving, computing, and information technology with robots. J. Comput. Small Coll. 17, 45-55.

FOSS, B.A. AND EIKAAS, T.I. 2006. Game play in Engineering Education - Concept and Experimental Results. The International Journal of Engineering Education 22(5).

GESTWICKI, P.V. 2007. Computer games as motivation for design patterns. SIGCSE Bull. 39, 233-237.

GLASER, B. 1992. Basics of grounded theory analysis. Mill Valley, CA: Sociology Press.

GROSSMAN, A. 2003. Postmortems From Game Developer. Focal Press, January.

HOLMES, N. 2005. Digital Technology, Age, and Gaming. Computer 38, 11 (Nov. 2005), 108-107.

IMBERMAN, S.P. 2004. An intelligent agent approach for teaching neural networks using LEGO handy board robots. Journal on Educational Resources in Computing 4, 4.

IEEE. 2000. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. Software Engineering Standards Committee of the IEEE Computer Society.

LAHEY, B., BURLESON, W., N, C., JENSEN, R., FREED, N. AND LU, P. 2008. Integrating video games and robotic play in physical environments. In Proceedings of the 2008 ACM SIGGRAPH symposium on Video games, Los Angeles, California, ACM.

LINDER, S.P., NESTRICK, B.E., MULDERS, S. AND LAVELLE, C.L. 2001. Facilitating active learning with inexpensive mobile robots. In Proceedings of the Proceedings of the sixth annual CCSC northeastern conference on The journal of computing in small colleges, Middlebury, Vermont, United States.

LINHOFF, J. AND SETTLE, A. 2008. Teaching game programming using XNA. In Proceedings of the 13th Annual Conference on innovation and Technology in Computer Science Education (ITiCSE '08), 250-254, Madrid, Spain, June 30 - July 02.

LOZANO-PÉREZ, T. 1990. In Preface to Autonomous Robot Vehicles, Springer Verlag, New York, NY.

LUMIA, R., FIALA, J. AND WAVERING, A. 1990. The NASREM Robot Control System and Testbed. In International Journal of Robotics and Automation, no.5, 20-26.

KAZMAN, R., KLEIN, M., BARBACCI, M., LONGSTAFF, T., LIPSON, H., AND CARRIERE, J. 1998 The Architecture Tradeoff Analysis Method. Fourth IEEE International Conference on Engineering Complex Computer Systems (ICECCS'98).

KRIKKE, J. 2003. Samurai Romanesque, J2ME, and the Battle for Mobile Cyberspace, IEEE Computer magazine, 23(1).

KRUCHTEN, P. 1995. The 4+1 View Model of Architecture, IEEE Software, 12, 6, Pp. 42 – 50.

KRUSKAL, W. H. AND WALLIS, W. A. 1952. Use of ranks in one-criterion variance analysis. Journal of the American Statistical Association 47 (260): 583–621.

MICROSOFT. 2009A. XNA Development Center. Web: http://msdn.microsoft.com/en-us/xna/, Accessed March 12th 2009.

MICROSOFT. 2009B. The C# Language. Web: http://msdn.microsoft.com/en-us/vcsharp/aa336809.aspx, Accessed March 12th 2009.

MILI, F. BARR, J., HARRIS, M. AND PITTIGLIO, L. 2008. Nursing Training: 3D Game with Learning Objectives. In Proceedings of the First international Conference on Advances in Computer-Human interaction, February 10 – 15.

NATVIG, L., LINE, S., AND DJUPDAL, A. 2004. Age of Computers: An Innovative Combination of History and Computer Game Elements for Teaching Computer Fundamentals. In FIE 2004: Proceedings of the 2004 Frontiers in Education Conference.

NAVARRO, A. O. AND HOEK, A. 2004. SimSE: an Educational Simulation Game for Teaching the Software Engineering Process. In ITiCSE '04: Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education, pages 233–233, New York, NY, USA. ACM Press.

NGUYEN, D. AND WONG, S.B. 2002. Design patterns for games. In Proceedings of the Proceedings of the 33rd SIGCSE technical symposium on Computer science education, Cincinnati, Kentucky, ACM.

PAPERT, S. 1980. Mindstorms: children, computers, and powerful ideas. Basic Books, Inc.

PAUSCH, R., BURNETTE, T., CAPEHEART, A.C., CONWAY, M., COSGROVE, D., DELINE, R., DURBIN, J., GOSSWILER, R., KOGA, S. AND WHITE, J. 1995. ALICE: Rapid Prototyping System for Virtual Reality. IEEE Computer Graphics and Applications 15, 8-11.

PERRY, D. P. AND WOLF, A.L. 1992. Foundations for the Study of Software Architecture. ACM Sigsoft Software Engineering Notes: 17(4), 40-52.

PFEIFER, R. 1997. Teaching powerful ideas with autonomous mobile robots. Computer Science Education 7, 161-186.

PIAGET, J. AND BARBEL, I. 1969. The Psychology of the Child. Basic Books Inc.

RABIN, S. 2008. Introduction to Game Development, Course Technology Cengage Learning, 2008.

RANKIN, Y., GOOCH, A. AND GOOCH, B. 2008. The impact of game design on students' interest in CS. In Proceedings of the 3rd international Conference on Game Development in Computer Science Education (GDCSE '08), 31-35, Miami, Florida, February 27 - March 03.

RESNICK, M. 1994. Turtles, Termites and Traffic Jams, Explorations in Massively Parallel Microworlds. MIT Press.

RESNICK, M., KAFAI, Y. AND MAEDA, J. 2003. A Networked, Media-Rich Programming Environment to Enhance Technological Fluency at After-School Centers in Economically-Disadvantaged Communities. In Propsal to the National Science Foundation, Web: http://www.media.mit.edu/~mres/papers/scratch.pdf.

ROLLINGS, A. AND MORRIS, D. 2004. Game Architecture and Design - A New Edition. New Riders Publishing.

ROSAS, R., NUSSBAUM, M., CUMSILLE, P., MARIANOV, V., CORREA, M., FLORES, P., GRAU, V., LAGOS, F., LOPEZ, X., LOPEZ, V., RODRIGUEZ, P., AND SALINAS, M. 2003. Beyond Nintendo: design and assessment of educational video games for first and second grade students. Computers & Education, 40(1): 71–94.

RYOO, J., FONSECA, F., AND JANZEN, D. S. 2008. Teaching Object-Oriented Software Engineering through Problem-Based Learning in the Context of Game Design. In Proceedings of the 2008 21st Conference on Software Engineering Education and Training (CSEET 2008), April 14 - 17, pages 137-144.

SHARPLES, M. 2000. The design of personal mobile technologies for lifelong learning. Computer & Education, 34(3-4): 177–193.

SINDRE, G., NATTVIG, L., AND JAHRE, M. 2009. Experimental Validation of the Learning Effect for a Pedagogical Game on Computer Fundamentals. IEEE Transactions on Education, 52(1), pages 10-18.

SLINEY A., MURPHY, D. AND J. DOC. 2008. A Serious Game for Medical Learning. In Proceedings of the First international Conference on Advances in Computer-Human interaction, February 10 – 15.

SIMMONS, R. 1992. Concurrent Planning and Execution for Autonomous Robots In IEEE Control Systems, no. 1, 46-50.

SHADISH, W.R., COOK, T.D. AND CAMPBELL, D.T. 2002. Experimental and Quasi-experimental Designs for Generalized Causal Inference, Houghton-Mifflin.

SHAFER, S.A., STENTZ, S.A. AND THORPE, C.E. 1986. An Architecture for Sensor Fusion in a Mobile Robot. In Proceedings of the IEEE International Conference on Robotics and Automation, April 7-10, 2002-2011.

SHIRAI, A., RICHIR, S., IWAMOTO, T., KOSAKA, T. AND KIMURA, H. 2009. WiiRemote programming: development experiences of interactive techniques that can be applied to education for young engineers. In Proceedings of the ACM SIGGRAPH ASIA 2009 Educators Program, Yokohama, Japan2009 ACM.

SUNG, K. 2009. Computer games and traditional CS courses. Communication of the ACM 52, 74-78.

SUNG, K., SHIRLEY, P. AND ROSENBERG, B.R. 2007. Experiencing Aspects Of Games Programming In An Introductory Computer Graphics Class. In Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education, Covington, Kentucky, USA, ACM.

SWEEDYK, E. AND KELLER, R.M. 2005. Fun and games: a new software engineering course. ACM SIGCSE Bulletin, 37(3), 138-142, September.

TOAL, D., FLANAGAN, C., JONES, C. AND STRUNZ, B. 1996. Subsumption architecture for the control of robots, In 13th Irish Manufacturing Conference (IMC-13), 703-711.

VICHOIDO, C., ESTRANDA, M. AND SANCHEZ, A. 2003. A constructivist educational tool: Software architecture for web-based video games, 4th Mexican International Conference on Computer Science (ENC 2003), 8-12 September, Apizaco.

VOLK, D. 2008. How to embed a game engineering course into a computer science curriculum. In Proceedings of the 2008 Conference on Future Play: Research, Play, Share, 192-195, Toronto, Ontario, Canada, November 3 - 5.

WANG, A.I., MØRCH-STORSTEIN, O.K., AND ØFSDAHL, T. 2007. Lecture quiz - a mobile game concept for lectures. In The 11th IASTED International Conference on Software Engineering and Application (SEA 2007), November 19-21.

WANG, A.I., ØFSDAHL, T. AND MØRCH-STORSTEIN, O.K. 2008. An Evaluation of a Mobile Game Concept for Lectures. In 21st IEEE-CS Conference on Software Engineering Education and Training (CSEE&T 2008), April 14-17.

WANG, A.I. AND STÅLHANE, T. 2005. Using Post Mortem Analysis to Evaluate Software Architecture Student Projects. In Proceedings of the 18th Conference on Software Engineering Education & Training, April 18 – 20.

WANG, A.I. AND WU, B. 2009. An Application of Game Development Framework in Higher Education, International Journal of Computer Games Technology, Special Issue on Game Technology for Training and Education, Volume 2009.

WSU. 2009. Download WSU_KSuite_1.1.2. Web: http://carl.cs.wright.edu/page11/page11.html, Accessed March 12th 2009.

WU, B. AND WANG, A.I. 2009. An Evaluation of Using a Game Development Framework in Higher Education. In 22nd IEEE-CS Conference on Software Engineering Education and Training (CSEE&T 2009), February 17-19, Hyderabad, India.

YOUNGBLOOD, G.M. 2007. Using XNA-GSE Game Segments to Engage Students in Advanced Computer Science Education. In The 2nd Annual Microsoft Academic Days Conference on Game Development, February 22-25.

ZHU, Q., WANG, T. AND TAN, S. 2007. Adapting Game Technology to Support Software Engineering Process Teaching: From SimSE to MO-SEProcess. In Proceedings of the Third international Conference on Natural Computation (ICNC 2007) - Volume 05, 777-780, August 24 - 27.