Post-Mortem Analysis of Student Game Projects in a Software Architecture Course

Successes and Challenges in Student Software Architecture Game Projects

Alf Inge Wang[1]
Dept. of Computer and Information Science,
Norwegian University of Science and Technology
Trondheim, Norway
alfw@idi.ntnu.no

Abstract—**In 2008, a game development project was introduced in a software architecture course at the Norwegian University of Science and Technology. The motivation for introducing the project was to let students learn how software architecture practices and processes can improve the final product in an inspiring and practical way. In the project, students organized in groups had to establish functional and quality requirements, design the software architecture of the game, evaluate the architecture, implement the architecture and test the architecture. After completing the project, all groups had to perform a post-mortem analysis of the project to reflect on the positive and the negative issues related to the project. This paper summarizes and describes the results of this post-mortem analysis along with the students' experiences from performing a post-mortem analysis of a game development project. The results show that there are both positive and negative effects of teaching software architecture in the context of a game development project. Students found it motivating to learn about software architecture through game development, but some students found it hard to apply the theory when developing the game. Most students were very positive to learn about new game technology as a part of the course and it was very stimulating to create an actual product. The main complaints were shortage of time, that many found the evaluation of architecture (ATAM) worthless, and that the project demanded too much documentation. Most students commented positive on doing a post-mortem analysis as a part of a game development project.**

Game development project, software architecture, Post-mortem analysis, XNA.

I. INTRODUCTION

Games are commonly used to teach kids, and have proven to 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 games concepts and game development used in higher education is not unique, e.g. [3-5], but there is an untapped potential that needs to be explored.

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 students [8] [9] through knowledge-based multiplayer games played by the students and the teacher. *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 a post-mortem analysis (PMA) of the latter, where a game development project was introduced in a software architecture course. 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 work more with course material through the project. Many students dream of making their own games, and game development projects stimulates the creativity of the students. In addition, game technologies and game user interfaces are now more commonly used in serious applications [12-15], 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 [16] [17]. Well-designed software architectures are needed, as games are growing in size and becoming more complex [18]. 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 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 [19].

This paper describes the results of a PMA of a game development project in a software architecture course conducted by students. The motivation for performing a PMA was for the students to share and learn from their experience. The intention of analyzing the students' PMA data was to get a detailed analysis of the positive and negative effects of

Work carried out as a guest researcher at Institute on Software Research at University of California, Irvine.

978-1-4244-4460-1/09/$25.00 ©2009 IEEE

combining game development and software architecture in the same course. We wanted to analyze how the students perceived the experience of doing a game development project in the context of a software architecture course. The results of the PMA highlight the positive and negative experiences learned from projects and reveal the advantages and disadvantages of having a game development project in a software architecture course. This information should be used to improve the course.

The rest of the paper is organized as follows. Section II describes the software architecture course. Section III presents the PMA method the students used. Section IV describes the research approach used to analyze the results of the PMA. Section V presents the results of analyzing PMA data. Section VI discusses the results. Section VII describes related work, and Section VIII concludes the paper.

II. DESCRIPTION OF THE SOFTWARE ARCHITECTURE COURSE

The software architecture course is a post-graduate course offered to CS and SE students at the Dept. of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU). The course's workload is 25% of a semester, and about 70-80 students attend the course every spring. The students are mostly of Norwegians (about 80%), but there are also 20% foreign students mostly from EU-countries. About 10% of the students are female. The textbook used in this course is the "Software Architecture in Practice, Second Edition", by Clements, Bass, and Kazman [20]. 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 post-mortem analysis [21-25]. 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 that focuses on software architecture

A. An Unusual Approach

The TDT4240 software architecture course at NTNU is taught different 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. Through the project, the students have to use software architecture techniques, methods, and tools to succeed according to the specified project requirements and the documents 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 TDT4240 software architecture course has been rated as one of the most useful and practical courses offered at the Computer and Information Science department in surveys conducted among prior 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 design.

B. Course Evaluation

30% of the grade given in the software architecture course is the evaluation of the software architecture project all students have to do, while 70% is given from 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 software architecture itself, but the implementation should also reflect the architecture and the architectural choices.

C. 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 the quality requirements, describe the architectural drivers, and design and document the software architecture of the application in the project including several views and view-points, stakeholders, stakeholder concerns, architectural rationale, etc.
4. *Architecture evaluation:* Use the Architecture Trade-off Analysis Method (ATAM) [20, 26-27] to evaluate the software architecture in regards to the specified quality requirements.

5. *Implementation:* Do a detailed design and implement the application based on the designed architecture and based 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 [28]. In this phase, the students will elicit and analyze the successes and problems they had during the project.

In the two first phases of the project, the students work on their own or in pairs. For the phases 4-6, 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 the WSU Khepera robot simulator [29] in Java with emphasis on an assigned quality attribute such as availability, performance, modifiability or testability. The students were asked to program the robot to move a robot around in a maze, collect four balls and bring them all to a light source in the maze. Robot controller was chosen to be a case 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 [30], Elfes [31], Task Control [32], CODGER [33], Subsumption [34], and NASREM [35].

In 2008, a game development project was introduced. In the Game project, the students were asked to develop a game using Microsoft XNA framework [36] and C# [37]. All our students have good skills and knowledge in Java, but very few knew C#. The students got to decide what type of game they want 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 pattern for games. There are some papers and presentations that describe architectures of specific games [38-42], and books that give a brief overview of game architectures [43-44], but no literature that gives depth study of the typical abstractions you can observe in game software. The most recurring architectural patterns described in books and papers are model-view controller, pipe-and-filter, layered and hierarchical task trees. In the 2008 version of the software architecture course, the students could choose between a robot and a game project. This paper only focuses only on the game project.

III. POST-MORTEM ANALYSIS

According to Rising et al. [45], retrospective analysis as a method for learning from work experience was identified in 1988 by Joseph Juran and named ”Santayana review” in homage to the philosopher George Santayana. Since then, many organizations have used many variations of the method and under many different names. We adopt Dingsøyr’s definition [46], such that retrospective analysis is a:

“collective learning activity, which can be organized for projects either when they end a phase or are terminated. The main motivation is to reflect on what happened in a project in order to improve future practice - for the individuals that have participated in the project and for the organization as a whole.”

Dingsøyr lists the most common names for retrospective analysis in [46]: ”project retrospectives”, ”post-mortem analysis”, ”post-project review”, ”project analysis review”, ”quality improvement review”, ”autopsy review”, ”after action review”, and ”touch down meetings”. In the software architecture course, we have used a post-mortem analysis (PMA) method that can be classified as a lightweight semi-structured brainstorming process for eliciting experience from a project. These characteristics fit well for a retrospective analysis method used for student projects with limited time and limited patience of the students.

The PMA method we used in the software architecture course is a modified version of the method suggested by Birk et al. [47]. The PMA process consists of four steps [28]:

1. *PMA introduction:* Introduce the PMA method and explain the purpose of the review.
2. *KJ-session 1:* Elicit positive experience.
3. *KJ-session 2:* Elicit negative experience.
4. *Causal map analysis:* Perform root-course analysis on the most important positive experience and the most important negative experience using causal map.

A. KJ-session

The KJ-method is a focused brainstorm method [48], resulting in affinity diagrams. KJ-sessions are conducted as follows. Each participant receives a number of post-it notes and is asked to write down what they regard as the most significant experiences from the project. After everyone has finished writing, each participant puts a note on a whiteboard while explaining what he means by it. The process is repeated until all the notes have been presented, as illustrated in Figure 1a). Once all the notes have been placed on the whiteboard, the whole group discusses them and groups them according to similarity in concept. Each group of notes is then given a name, as illustrated in Figure 1b). Possible connections between groups can be marked with arrows if required. In our study, each participant received five post-it notes and the entire process was repeated twice; first for positive experiences (KJ-session 1), then for negative experiences (KJ-session 2).

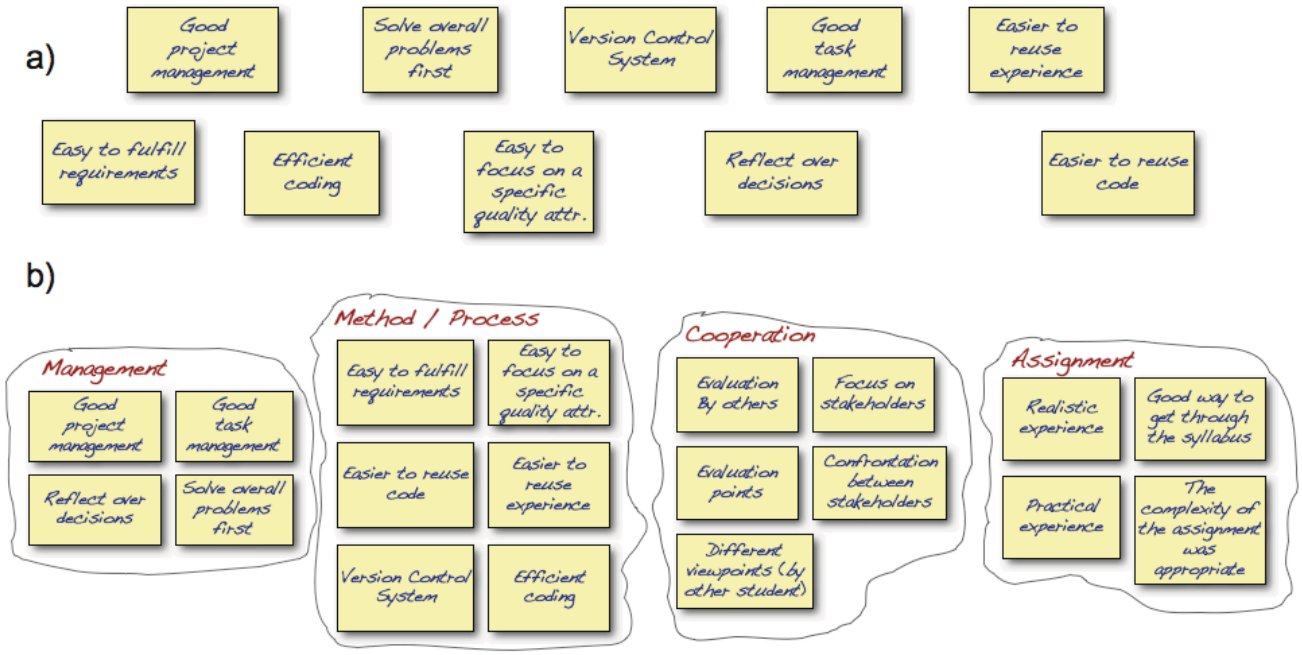

Figure 1. KJ example

B. *Causal Map Analysis*

The group selects a (positive or negative) experience they want to analyze. All participants are given post-it notes and are asked to write down the causes of the experience to be analyzed. These notes are then presented and placed on the whiteboard, much in the same way as when using the KJ-method. The group then gathers at the whiteboard and groups the causes where applicable. Arrows indicate the cause-effect relationships. The members are then allowed to write new notes that state deeper causes, or if causes are seen to be missing, write those in and indicate them with arrows. When the new causes have been placed on the whiteboard, the process is iterated until the group is satisfied with the analysis.

Figure 2 shows a possible outcome of a causal map analysis. The figure shows the resulting causal map from a positive root-cause analysis on good assignment, which is an identified group from the KJ-diagram in Figure 1. Here, every oval represents a concept, every arrow indicates a cause-effect relationship, and the whole map represents a specific situation.

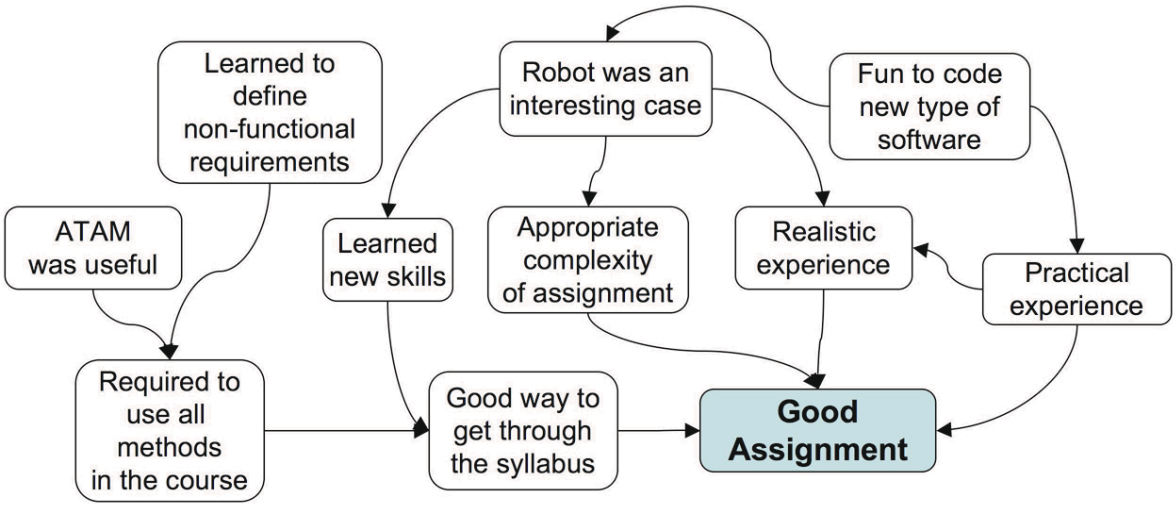

Figure 2. Causal map example

C. *Post-Mortem Analysis in Game Projects*

The main motivation for bringing PMA into any software project is to learn from own experiences in order to improve. The improvement is a result of continuing the identified good practices (successes) and changing the identified bad practices (failures) into good practices. Game projects benefit also from this same experience-based improvement process. One can argue that PMA is even more important in game projects, as game development teams are usually more diverse than ordinary software teams consisting of programmers, artists, designers, musicians, audio-engineers, testers, QA-staff, modelers, etc. Because of the multi-disciplinary characteristics of game development teams, it is very important to get reflections from the various people involved to catch in problems due to people from various professional background. In the PMA conducted in the TDT4240 software architecture course, the multi-disciplinary aspect was present to a large degree, as the student were all programmers.

IV. RESEARCH APPROACH

The main contribution described in this paper is a summary and a meta-analysis of the results of the students PMA of their game project conducted in the context of a software architecture course. The data in the analysis are collected from PMA reports from 15 groups, which contain a positive and a negative KJ-diagram, a positive and a negative causal map, and experiences from conducting a PMA in a game development project. This paper does not include an analysis of the PMA data from the robot project, as the focus is only on the game project. A comparison of game project and robot project can be found in [49]. The analysis of data of the KJ-diagram and the causal-maps must be performed in two different ways. For the KJ-diagrams, it is possible to cluster similar item and identify the most frequent positive and negative issues in the PMA reports. However, this approach cannot be used for the causal-maps as they represent an analysis of *one particular case*. In order to analyze the results of the KJ-diagram, the following approach was used:

1. *Collect data:* Extract the items identified in the KJ-diagram in the PMA reports.
2. *Group items according to main theme:* All the KJ-items was grouped according to a theme such as Assignment (issues related to the assignment), Programming (issues related to programming), and Educational (issues related to learning).
3. *Uniform items:* Change to a uniform description of items. In this step, all items were described on according to a pre-defined pattern: [noun] [adjective]/[verb] ([details]). Examples of uniform descriptions of items can be "Assignment fun", "XNA good", and "Group member bad".
4. *Merge items:* Similar items were merged to the same item description. Example of merging of items can be "XNA framework good" and "XNA good" were merged into "XNA good".
5. *Analysis:* Find the most frequent themes that most groups have covered, and find the most overall frequent items found in the positive and negative KJ-diagrams.

V. RESULTS

This section is divided into three main parts. The first part describes the results of analyzing the KJ-diagrams, the second part describes the most important root-cause analyses, and the third part summarizes the students' own experiences of conducting a PMA in a game development project.

A. *Results from Analyzing KJ-diagrams*

The results from analyzing the KJ-diagrams consist of two parts on two different abstraction levels: Themes the groups have identified (high-level) and items (low-level).

1) *Positive KJ-diagram Themes*

The first part of the KJ-diagram analysis was to find the themes most groups covered in their PMA. These themes were found based on how the students grouped their items in the KJ-diagram and identified what the students perceived as the most important impacts on their project at a high abstraction level. Figure 3 shows the distribution of themes from all groups that were regarded as a success in relation to the game development project.

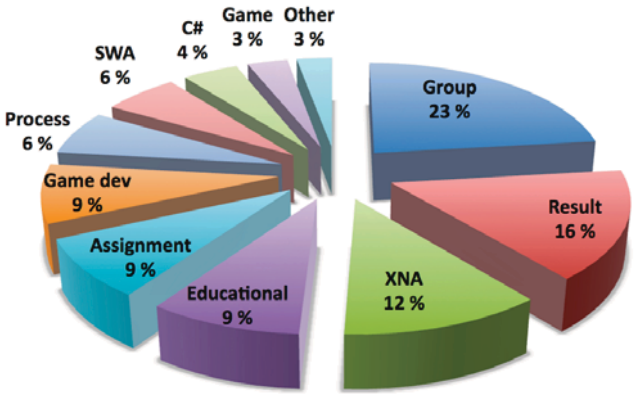

Figure 3. Distribution of themes from positive KJ-diagrams

Figure 3 shows the 23% of items from the positive KJ-diagrams were related to student *group* and group processes. This is not a surprise, as the students have to work hard together for two months in order to make a successful project. 16% of the items from the positive KJ-diagrams were related to the *result* of the project. This result includes both the software architecture documentation and the implementation of the game itself. 12% of the items from the positive KJ-diagrams were related to the *XNA* game framework. Even though the students had to learn a new game framework as a part of the software architecture course, most students regarded this as a good thing. 9% of items from the positive KJ-diagrams were related to the three themes *educational* – that the students learned from the project, *assignment* – the assignment was

regarded as exciting, challenging and fun, and *game development* – most students found it fascinating to learn how to develop a game. 6% of the items found from the positive KJ-diagrams were related to the two themes *process* – the software development process, and the *software architecture (SWA)* – apply a software architecture in practice. The remaining items were related to positive aspects of *C#* (4%), the *game* the students developed (3%), and *other* minor issues (3%).

2) Negative KJ-diagram Themes

The negative KJ-diagram themes were found based on how the students grouped their items in the KJ-diagram and identified what the students perceived as the most important negative impacts on their project at a high abstraction level. Figure 4 shows the distribution of themes from all groups that were regarded as challenges or problems in the game development project.

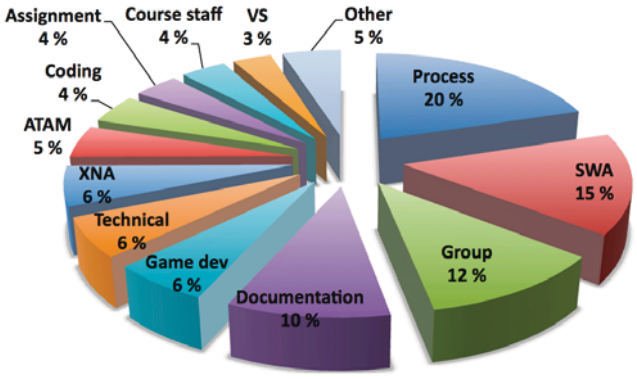

Figure 4. Distribution of themes from negative KJ-diagrams

Figure 4 shows the 20% of items from the negative KJ-diagrams were related to software development *process*. Typical issues were bad planning, bad estimation of resources, poor follow-up and problems with coordination. 15% of the items from the negative KJ-diagrams were related to the *software architecture (SWA)*. These issues ranged from problems applying software architecture to a game to understanding the concepts of software architecture. 12% of the items from the negative KJ-diagrams were related to problems related to *group*, group members and group dynamics. 10% of items from the negative KJ-diagrams were related to *documentation*. Documentation issues ranged from too much documentation required to problems maintaining an updated documentation of the project. 6% of the items from the negative KJ-diagrams were related to the three themes difficulties related to *game development*, *technical* problems related to hardware and software issues and challenges of handling the *XNA* framework. Other identified themes from the negative KJ-diagrams lack of usefulness of the *ATAM* (5%), problems related to *coding* (4%), ambiguity in *assignment* documents (4%), lack of support from *course staff* (4%), problems with *Visual Studio (VS)* (3%), and *other* issues (5%).

In the next two sections, we will look further into details about the specific positive and negative issues related to the game development project.

3) Positive KJ-diagram Items

The second part of the KJ-diagram analysis consisted of looking at all issues identified in the students' KJ-diagram, clustering the items by describing them in an uniform way, merging similar descriptions, and counting the number of items identified by several groups. Table I shows a sorted list of positive items identified by 20% of the groups or more. Note that the descriptions of items are rephrased from original wording (short keyword description) to improve readability.

Table I shows that the positive item identified by two out of three groups was that they learned C# from the doing the game development project. As most of our students are technology focused and only knew Java from before, this was not a big surprise. We can recognize the same trend in that 60% of the groups identified the positive effect of learning XNA through the project, XNA was regarded as exciting, fun and user friendly (26.67%), and that it was positive to learn how to use Visual Studio (20%).

TABLE I. POSITIVE ITEMS FROM KJ-DIAGRAMS

#	Description of positive KJ-item	% of grps
1	Learned C#	66,67 %
2	Collaboration was good	60,00 %
3	Learned XNA	60,00 %
4	Game development is fun	53,33 %
5	Assignment was fun	46,67 %
6	Group members were good	46,67 %
7	Assignment that produces a real result	40,00 %
8	Games are fun	40,00 %
9	Group was good	26,67 %
10	Learned software architecture from practical project	26,67 %
11	Work distributed in the group was good	26,67 %
12	XNA was exciting	26,67 %
13	XNA was fun	26,67 %
14	XNA was user friendly	26,67 %
15	Assignment was challenging	20,00 %
16	Made the final delivery before the deadline	20,00 %
17	Game concept was good	20,00 %
18	Game was working	20,00 %
19	Graphical work is fun	20,00 %
20	Group dynamics was good	20,00 %
21	Group members were skilled	20,00 %
22	Group was motivated	20,00 %
23	Learned a lot	20,00 %
24	Motivation was good	20,00 %
25	Project was fun	20,00 %
26	The Software Architecture was good	20,00 %
27	Learned Visual Studio	20,00 %

Issues related to group and group dynamics are also clearly prominent in Table I with identified items like collaboration was good (60%), group members were good (46.67%), group was good (26.67%), and work distribution in group was good (26.67%). Based on observations from previous years, it seams that the game development project has improved the group dynamics in the software architecture course. A possible

explanation for this could be that the groups get a stronger ownership to the project, as the product is specified by themselves and not by the course staff like for the robot project.

One of the reasons for introducing a game project in a software architecture course was to motivate the students to put more effort into the project and get motivated to take the course. The results in Table I shows that a game development project motivates the students: game development is fun (53.33%), assignment is fun (46.67%), games are fun (40%), assignment with a real result (40%), motivation was good (20%), and project was fun (20%).

For the game development project to be successful in the software architecture course, the students must learn software architecture. The KJ-item analysis revealed that in addition to learning new technology, the students identified positive effects of learning software architecture through the project: learned software architecture from a practical project (26.67%), assignment was challenging (20%), and learned a lot (20%). Also other KJ-items identified by less that 20% of the groups were related to learning software architecture (not shown in Table I), such as assignment shows need for good software architecture, ATAM evaluation was good, made a reusable game framework, various architecture and design patterns were learned, software architecture eased the development, and learned how to document a software architecture.

4) Negative KJ-diagram Items

The last part of the KJ-diagram analysis was to identify recurring negative KJ-items described by the students in their PMA reports. Table II shows a sorted list of negative items identified by 20% of the groups or more.

Table II shows by far that the major headache in the game development project was shortage of time (80%). This problem is routed in over-ambitious game design and software architecture, starting to late with the project (20%) and time pressure from other courses (26.67%). All projects were delivered within a couple day after then deadline. Some project suffered from having incomplete implementation, but most projects were complete. In this course, students have a tendency to underestimate the effort to make a proper report and to make a proper implementation. 60% of the groups reported that the documentation required in this course was massive. One third of the groups reported that documentation was boring in their negative KJ-diagram. Another documentation issue was that one third of the groups found the *documents requirements unclear*. Other document-related negative items found in KJ-diagrams in less than 20% of the groups (not in Table II) included documentation got too low priority, the document templates were to rigid, the updating of documents were bad, the documentation was complex, and that document requirements were not always followed. The assignment requires the students to document the software architecture according to the IEEE 1471 [23], which feels like an overkill for many of the students. However, to make proper software architecture documentation is a part of the education goal of the course and must be a part of the course even if students find this part boring and annoying. The students' KJ-diagrams reveals that there is room for improving documentation templates and document requirements.

TABLE II. NEGATIVE ITEMS FROM KJ-DIAGRAMS

#	Description of positive KJ-item	% of grps
1	Time was too short	80 %
2	Documentation was massive	60,00 %
3	ATAM was useless	46.67 %
4	Document requirements were unclear	33.33 %
5	Documentation is boring	33.33 %
6	Group meeting scheduling was difficult	33.33 %
7	Planning was bad	33.33 %
8	Quality attribute had too little focus	33.33 %
9	XNA was unknown	33.33 %
10	C# was unknown	26,67 %
11	Communication was bad	26,67 %
12	Game development was unknown	26,67 %
13	PC lab was limited	26,67 %
14	Process was bad	26,67 %
15	Time pressure from other courses	26,67 %
16	Visual Studio was unknown	26,67 %
17	XNA is Windows only	26,67 %
18	ATAM feedback was not used	20,00 %
19	Code was messy	20,00 %
20	Cooperation was bad	20,00 %
21	Course staff give feedback late	20,00 %
22	Feedback from course was poor	20,00 %
23	Game logics took too much time	20,00 %
24	Group members were not punctual	20,00 %
25	Patterns were not implemented	20,00 %
26	Quality attribute was difficult	20,00 %
27	Sickness of group member	20,00 %
28	Sleep little in the last part of the project	20,00 %
29	Started too late with the project	20,00 %
30	Subversion (SVN) and Visual Studio caused problems	20,00 %
31	Software architecture focus difficult due to XNA	20,00 %

46.67% of the groups described ATAM as useless in their KJ-diagram and that the feedback from the ATAM session was not used (20%). ATAM is an evaluation method where the software architecture is evaluated against specified quality requirements (quality scenarios). In our course, one group acts as the evaluation team investigating another group's architecture. After completing the evaluation of one group, the two groups switch roles. The educational goals of this part of the project are 1) to force the students to learn the ATAM, and 2) to give the students an opportunity to get feedback on their software architecture. The first goal is not so hard to achieve, but the second goal is harder as the students lack experience to give useful feedback on software architecture decisions. A possible approach to improve issues related to the ATAM is for course staff to participate more actively in the ATAM-sessions. The second education goal of ATAM would benefit from this approach, but the first would most likely suffer.

Large portion negative items found in the students' KJ-reports are related to the group and how the groups/project were organized. One third of the groups found it difficult to schedule group meetings, and planned the project badly. Other related issues identified were bad communication (26.67%), bad process (26.67%), bad cooperation (20%), not punctual group members (20%), sickness (20%), and little sleep in the

last part of the project (20%). These issues are perfectly normal challenges most team-based projects have to face.

Several groups also reported negative issues that were related to the software architecture domain, such as the quality attribute had too little focus in the project (33.33%), quality attribute specification and usage were difficult (20%), software architectural patterns and design patterns were not implemented (20%), and it was difficult to focus on the software architecture due to constraints in XNA (20%). For most students, the most challenging part of the software architecture course is to go from specifying the requirements and software architecture of an application (game) to implement the game accordingly. XNA puts restrictions on how the software architecture can be designed, which can be difficult to comprehend by the students in the first phase of the project. To succeed, the project groups need to re-design the architecture and implement the game through several iterations. This concept is new and challenging for the students as they are used to the waterfall software process [50]. Another possible explanation for negative issues related to the software architecture domain can be that 20% of the groups reported that they got feedback on their software architecture late during the project and that 20% reported that the feedback on the software architecture from the course staff was poor.

Other issues that were a challenge in the game development project are related to learning new technology and issues related to this technology. One third of the groups mentioned that XNA was unknown, and 26.67% that C# and Visual Studio were unknown. The main effect of the new technology was that the students had to spend extra time to learn a new programming language (although very similar to Java), a new programming framework, and a new programming environment. The course staff were considering using game frameworks in Java, but did not find any that provided a high-level API, expressiveness, maturity, flexibility, and the level of performance found in XNA [51]. Other negative issues related to choosing XNA as a developing platform was that it runs only on Windows (26.67%), it was difficult to provide sufficient PC labs (26.67%) as thin-clients are used, and it was difficult to get configuration-tool Subversion to work with Visual Studio (20%). More and more students have laptops running Mac OS X and Linux, making it hard to work with XNA, which only runs on Windows.

Game development was identified in the KJ-diagrams to have a negative impact on the project. 26.67% of the groups identified the problem that game development was an unknown domain (26.67%), and 20% documented that it took too much time to implement the game logics.

B. *Root-cause Analysis / Causal Maps*

The second part of the PMA performed by the students consisted of a root-cause analysis where the students focused on finding the causes for the most important successes and the most important challenge or problem in the project. The result of the root-cause analysis was one positive and one negative causal-map per group.

1) *Positive Root-cause Analysis/Causal maps*

Figure 5 shows the distribution of topics the students focused on in their positive root-cause analysis.

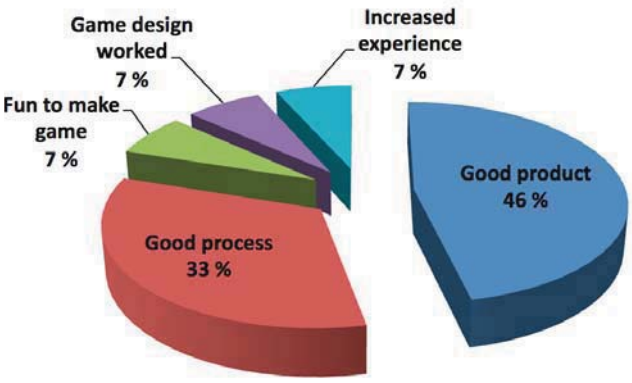

Figure 5. Distribution of topics described in positive causal maps

Almost half of the groups (46%) focused on that the project produced a good product being the game, the architecture, or the implementation. One third of the groups (33%) focused their positive causal root analysis on that they had a good process in their project. Sub topics for the good process were no problems during implementation, finished the project in time and good group cooperation. The remaining groups did a root cause analysis on topics "fun to make a game" (7%), "the game design worked" (7%), and "increased experience" (7%).

Figure 6 shows a positive causal map from a group that focused on "Nice product". The figure shows a mixture of a good development process, the focus on the software architecture, XNA, fun assignment, and skilled group members that caused the success of the project and the product. This causal map is very representative for groups that focus their root cause analysis on good product.

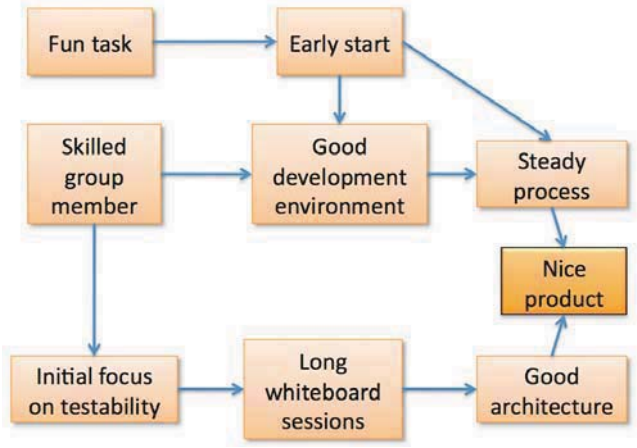

Figure 6. Positive causal map for nice product

Figure 7 (on next page) shows a positive causal map from a group that focused on "Good process".

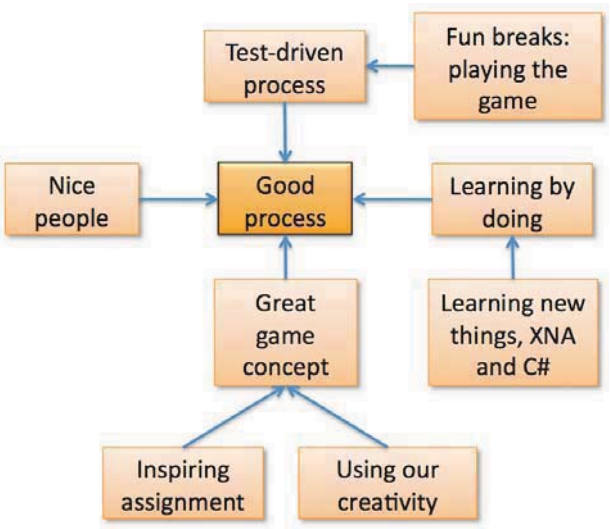

Figure 7. Positive causal map for "Good process"

The causal map in Figure 7 reveals that game development projects benefits from being inspirational and they allow the students to use their creativity. The causal maps also reveals that XNA had a positive effect on the development process. Four out of five of the groups (80%) that focused on "Good process" in their positive root cause analysis mention XNA as a positive contribution in their causal map.

2) Negative Root-cause Analysis/Causal maps

Figure 8 shows the distribution of topics the students focused on in their negative root-cause analysis.

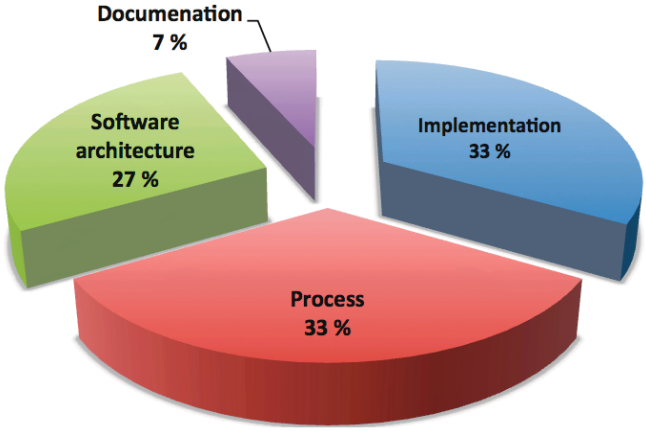

Figure 8. Distribution of topics described in positive causal maps

The pie diagram in Figure 8 shows that the three major problems or challenge in the game development project were related to the implementation (33%), the development process (33%), and issues concerning the software architecture (27%). Negative issues related to the implementation were suboptimal implementation, performance problems, challenges to implement physics, and incomplete implementation. Negative issues related to the process were that the plan was not followed, bad documentation process, intense work in the last moment, bad time management, and insufficient level of effort. Negative issues related to the software architecture were difficulties to focus on quality attributes (quality requirements), challenges related to ATAM and wrong architectural approach. Finally, 7% of the groups performed a root cause analysis related to unclear report requirements (documentation).

Figure 9 shows a negative causal map from a group that focused on "Incomplete implementation" (implementation).

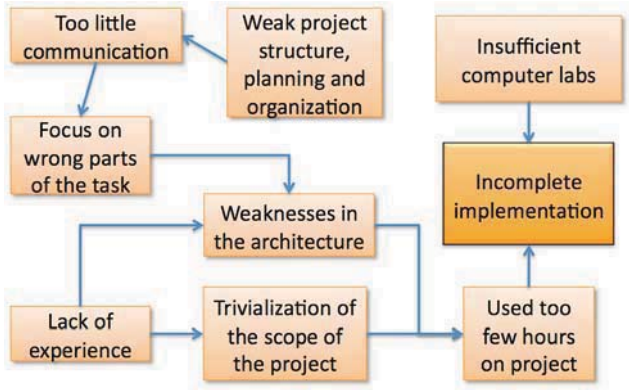

Figure 9. Negative causal map for "Incomplete implementation"

The figure shows that the main causes for an incomplete implementation was found to be insufficient computer labs, trivialization of the scope of project, lack of experience and too little communication. These are typical issues identified by several groups.

Figure 10 shows a negative causal map from a group that focused on " Did not exactly follow plan and documents" (process).

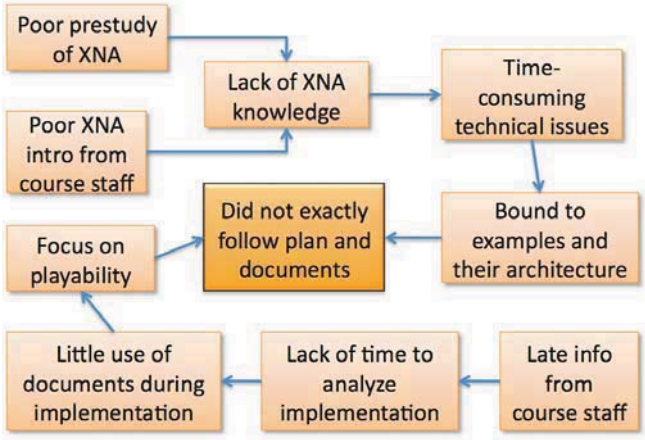

Figure 10. Negative causal map for "Not exactly follow plan"

Figure 10 shows typical issues that caused problems related to following the process and documentation, such as poor prestudy of XNA, poor XNA introduction from course staff, and little documentation during implementation. The latter is always a problem for students project, as they tend to do the

fun bit first (programming) and postpone the boring parts (documentation).

Figure 11 shows a negative causal map from a group that focused on " Wrong architectural approach" (architecture).

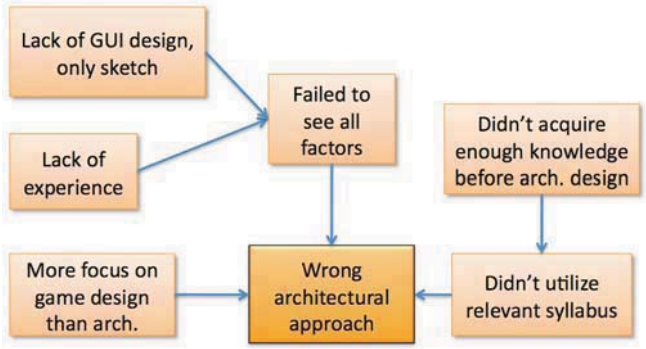

Figure 11. Negative causal map for "Wrong architectural approach"

The most recurring issues related to software architecture problems by several groups are that they do not read required syllabus during the project and that they have too much focus on game design instead of architectural design. The latter is one of the main disadvantages of introducing a game development project in a software architecture course.

C. *Experiences from Performing a PMA in a Game Development Project*

The overall comments about doing the PMA as a part of a game development project was very positive from all groups. Our PMA sessions last for four hours in total, and most students expressed that this time was well spent. Few groups mentioned that the PMA could just have been done through just talking about the project, but the majority found it very useful to have a more structured way of brainstorming about the positive and negative aspects of the project. Here are some examples of the experiences expressed in the students PMA reports:

"Through the session we identified both factors which had made the project go more fluidly, e.g. the group members experience, the XNA framework, and factors that made the project more difficult, e.g. a short time frame, lack of experience with C# and problems grasping the testability tactics in the book."

"During the brainstorming analyzing the positive and negative experiences and the reasons for it we got quite excited. We realized that even though we had thought about the negatives and the positives during the project, there were some that we forgot or took for granted like a great group room and well-organized project from the staff's side. We had also thought little of how some of the experiences had an effect overall. All in all we think PMA is quite useful to mark the end of that development stage and notice important experiences to take with you for the next project."

"We had never participated in a PMA before and were curious about the execution of and results from such an event. Our main concern was dealing with more paper work, but was positively surprised by the post-it workshop and time set off to discuss."

"It is very useful for next projects we will develop in future actions since we learnt the causes of positive and negative aspects when developing a game/project."

"We learned that skilled and experienced programmers are valuable, in both the documentation phase, the ATAM session and of course during implementation. We had to learn a great deal about XNA and C# - in addition to methods and architectural theory, that now makes us more competent of doing such a project if we were to do it once again. We also confirmed that it was the right thing to choose the game project."

"By brainstorming, we discussed about the positive and negative aspects of the project. This is a good way to understand how to improve our implementation process: in fact, we can know both which steps we did in a good way and which steps we didn't' do enough good and must be improved in the next projects."

"The group came very far on a good concept, but having good quality in the code (well modularized and commented code) eases the development process and saves time."

"The group learned the importance of a good project management and the use of internal milestones to increase the effort at an early stage. "

"In our project the open discussion around was the best experience, it led us to think about the project in a different way than we normally do, and the group was able to make some conclusions that was not that clear in the first place."

The experiences from students on the PMA indicate that the PMA-methods used in our software architecture course should be well suited to be used in commercial game development project as well. Such PMA sessions do not required much time (four-hours) and is easy to learn and comprehend.

VI. Discussion and Leassons Learned

In this section, we will discuss the main problems and challenges in the game development project identified by the students in the PMA and present some course improvements to limit the negative effects of having a game development project in a software architecture course.

The overall feedback from the students was positive to learn software architecture through a game development project. However, the PMA revealed several issues that are possible to improve in future versions of the course.

Although, most students were very pleased having XNA as a development platform in the software architecture course, there is room for improvement. From the students' feedback, the introduction of XNA needs to be improved to make it easier to learn the new technology. Another issue was students with laptops not running Windows. One solution could be to help the students to make it possible to run more than one operating system on the laptop (this is possible both for Linux and Mac OS X). Another solution is to use the monoxna cross platform implementation of Microsoft's XNA framework [52].

However, the monoxna project is still in early developing phase. In addition, the PC labs need to be upgraded with standalone PCs that can be used for game development (not thin-clients like today). Another issue that was raised by the students was that they did not understand enough of XNA and the constraints of XNA to design a proper software architecture that builds upon the framework. A solution to this problem is to extend the COTS exercise in phase one of the project (see Section II-C) to make the students learn the architectural aspects of XNA better. If the COTS exercise is extended the, pattern exercise must be changed, reduced or eliminated. Another approach could be to combine the two first exercises (COTS and pattern) in one bigger exercise where the students have to learn XNA and to learn architectural and design pattern in the context of XNA. This approach would help the students to reflect on the architectural constraints of XNA. Gestwicki and Sun describe on approach of how to teach design patterns through game development [53].

Limited time was the main challenge for most groups in the game development project. For many groups, starting too late on the implementation is the main reason for the limited time problem. This problem is most likely due to that the implementation phase of the project has allocated most time, and the students have a tendency to wait until the very last moment to start doing the work. A possible course improvement to minimize this negative effect would to make the students deliver an intermediate result during the implementation phase. This would force the students to start the implementation earlier and in addition improve their software architecture and implementation based on the feedback from course staff.

Many groups identified the ATAM evolution to be useless. This is not an easy problem to solve. It is important that the students learn how to be the evaluation team in an ATAM process as well as being evaluated. Thus, due to lack of experience the result of the ATAM evaluation is less likely to be as useful as a real ATAM evaluation would be. A possible course improvement could be to use course staff in the beginning of the ATAM sessions as a part of the evaluation team to help the evaluation team to look into the right things. Currently, this is a resource problem as there are only five persons (including student assistants) are involved in teaching the course and only one has practical ATAM experience.

The PMA analysis revealed some issues that must be improved: feedback from course staff on the students deliveries, document templates and improved teaching on the process of game development.

VII. Related Work

To our knowledge, an analysis of students' post-mortem analysis of their own game development projects in the context of a software architecture course is unique and is not published elsewhere. This section presents related work that describes evaluation of using game development project in computer science and software engineering courses, and some other work related to computer science or software engineering and game development.

Sweedyk and Keller describe how they introduced game development in an introductory SE course [54]. 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 been 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. The evaluation of the game projects in [54] was a survey and was not a depth evaluation of the project like presented in this paper.

Kajal and Mark Claypool describe another SE course where a game development project was used to engage the students and make the course more fun [55]. 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 in a survey conduced during the course was also very positive. The focus of the evaluation described in [55] was very different than in our study and focused on the course and not the project in particular.

Volk 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 [56]. The evaluation of the project was carried out in form of post-mortems during the post-production phase of the project similar to what described in this paper. However, the actual post-mortem method used is not described. 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 pre-production to

production was difficult (extracting the requirements), and that some student teams were overambitious for what they wanted to achieve in their project. Compared to our study, the only similar finding was a tendency of overambitious teams.

McGovern and Fager describe how introductory AI was taught in the context of an arcade-style gaming environment [57]. The students were asked in a project to implement three fundamental areas of AI (search, learning and planning) in an already existing Spacewar game implementation. The learning experience was evaluated using an anonymous survey where the students should answer questions according to the Likert scale as well as add comments of their own. The data analyzed from the survey showed that the introduction of game had made a significant contribution to the learning experience, and motivated the students to take more AI courses. The project described in this project is very different from our project, as the students are only asked to add minor parts of code to an existing system, while our students have to develop a game from scratch. McGovern and Fager's finding of improved learning experience is alignment with our finding that showed that students are motivated doing projects related to games.

Drake and Kerr describe an undergraduate course in software development where the students develop a computer strategy game using extreme programming (XP) [58]. They argue that it is possible for undergraduate students to work on large real-world projects such as a strategy game. The evaluation of the student projects revealed some of the issues found in our analysis of the students' PMA such as it was hard to create an architecture for the game early in the project, the students were over-optimistic of how much they could implement within the allocated time, that some games were a bit fragile, and that the students were excited about the project. As the students in Drake and Kerr's study used XP, the students managed to get a lot of the work in the project done in an early phase unlike many of our students that got a very heavy workload at the end of the project.

Youngblood describes how XNA game segments can be used to engage students in advanced computer science education [59]. 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 an AI course where the students was asked to implement a chat bot, motion planning, adversarial search, neural networks and flocking. Finally the paper describes seven design-principles specific for using game segments in CS education based on lessons learned. Game segments are not particularly relevant to a software architecture course since they put to heavy constrains on the design of the software architecture.

There are also some other papers that describes computer science or software engineering courses where game have been used as a part of the course [10, 60-61], but these they do not give any insight into the students' perception of the project.

VIII. Conclusion

This paper has described an evaluation of a student game development project introduced in a software architecture course seen from the students' perspective. The issues that contributed most to the positive aspects of the project were found to be related to the *group* or group processes, the *product* of the project, *XNA*, *learning*, the *assignment,* and *doing game development*. More specifically, the issues that contributed most to a positive project experience were that the students had positive experience with and learned C# and XNA, the group work and collaboration in the group were good, game development and the games were fun, the assignment was fun, and that they learned software architecture from a practical project. The most prominent issues that contributed to a negative perception of the game development project were related to difficult development process, the software architecture or software architecture theory, group issues and the documentation. More specifically, three issues that were perceived most negative by the students in relation to the project were that the *time was too short*, the *documentation was too massive*, and that *ATAM was useless*. Several groups also reported that it was difficult to focus on the software architectural issues of the project instead of game design and that XNA made it difficult to design and implement the architecture. The choice of using XNA as the game development platform in the software architecture course had both positive and negative effects on the project. Most students were positive to the technology and to learning the technology. However, some students felt that the XNA made it harder to design and implement the architecture and lost time in the project in having to learn new technology and tools. XNA being a Windows only platform is also an issue as more and more students run different operating systems on their laptops.

The results from PMA also revealed areas that should be improved in the course such as better and faster feedback on project deliveries, a dedicated PC lab for game development, better introduction to the COTS, more course staff guidance in ATAM sessions, and changes of the COTS exercise to include architectural and design patterns to give a better starting point when designing the software architecture.

Based on the overall results from the students' PMA, we conclude that introducing the game development project in the software architecture course was a good idea. The main benefits are motivated students, interesting assignment, good products (software architectures and games), ownership of the product, and good group processes. The main challenge is some students can loose focus on software architecture and spend too much time on game design and game implementation.

Finally, the students' feedback from conducting our particular PMA session was very positive. The PMA session, which consists of doing a positive and a negative KJ-diagram session (structured brainstorming) and a positive and a negative causal-map session (root-cause analysis), is a very effective method for revealing positive and negative issues in game development projects and learning to improve in future projects

Acknowledgment

We would like to thank the 2008 students of TDT 4240 software architecture course at the Norwegian University of Science and Technology for providing the necessary information. We would also like to thank Richard Taylor and

Walt Scacchi 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.

REFERENCES

[1] R. Rosas, M. Nussbaum, P. Cumsille, V. Marianov, M. Correa, P. Flores, V. Grau, F. Lagos, X. López, V. López, P. Rodriguez, and M. Salinas, "Beyond Nintendo: design and assessment of educational video games for first and second grade students," Comput. Educ., vol. 40, pp. 71-94, 2003.

[2] M. Sharples, "The design of personal mobile technologies for lifelong learning," Comput. Educ., vol. 34, pp. 177-193, 2000.

[3] A. Baker, E. O. Navarro, and A. v. d. Hoek, "Problems and Programmers: an educational software engineering card game," in Proceedings of the 25th International Conference on Software Engineering Portland, Oregon: IEEE Computer Society, 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," Proceedings of the 2004 Frontiers in Education Conference, 2004.

[5] E. O. Navarro and A. v. d. 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 Leeds, United Kingdom: ACM, 2004.

[6] B. A. Foss and T. I. Eikaas, "Game play in Engineering Education - Concept and Experimental Results," The International Journal of Engineering Education vol. 22, 2006.

[7] G. Sindre, L. Nattvig, and M. Jahre, "Experimental Validation of the Learning Effect for a Pedagogical Game on Computer Fundamentals," IEEE Transaction on Education, vol. 52, pp. 10-18, 2009.

[8] A. I. Wang, O. K. Mørch-Storstein, and T. Øfsdal, "Lecture quiz - a mobile game concept for lectures," IASTED International Conference on Software Engineering and Application (SEA 2007), November 19-21 2007.

[9] A. I. Wang, T. Øfsdal, and O. K. Mørch-Storstein, "An Evaluation of a Mobile Game Concept for Lectures," in Proceedings of the 2008 21st Conference on Software Engineering Education and Training - Volume 00: IEEE Computer Society, 2008.

[10] M. S. El-Nasr and B. K. Smith, "Learning through game modding," Comput. Entertain., vol. 4, p. 7, 2006.

[11] B. Wu, A. I. Wang, J.-E. Strøm, and T. B. Kvamme, "An Evaluation of Using a Game Development Framework in Higher Education," in Proceedings of the 2009 22nd Conference on Software Engineering Education and Training - Volume 00: IEEE Computer Society, 2009.

[12] N. Holmes, "Digital Technology, Age, and Gaming," Computer, vol. 38, pp. 108-107, 2005.

[13] A. Sliney, D. Murphy, and J. Doc, "A Serious Game for Medical Learning," First international Conference on Advances in Computer-Human interaction, February 10-15 2008.

[14] F. Mili, J. Barr, M. Harris, and L. Pittiglio, "Nursing Training: 3D Game with Learning Objectives," Proceedings of the First international Conference on Advances in Computer-Human interaction, pp. 10-15, 2008.

[15] L. v. Ahn, "Games with a Purpose," Computer, vol. 39, pp. 92-94, 2006.

[16] S. Caltagirone, M. Keys, B. Schlief, and M. J. Willshire, "Architecture for a massively multiplayer online role playing game engine," J. Comput. Small Coll., vol. 18, pp. 105-116, 2002.

[17] E. F. Anderson, S. Engel, P. Comninos, and L. McLoughlin, "The case for research in game engine architecture," in *Proceedings of the 2008 Conference on Future Play: Research, Play, Share* Toronto, Ontario, Canada: ACM, 2008.

[18] J. Blow, "Game Development: Harder Than You Think," Queue, vol. 1, pp. 28-37, 2004.

[19] D. Callele, E. Neufeld, and K. Schneider, "Emotional Requirements," IEEE Softw., vol. 25, pp. 43-45, 2008.

[20] P. Clements and R. Kazman, Software Architecture in Practices: Addison-Wesley Longman Publishing Co., Inc., 2003.

[21] J. Coplien, O., "Software design patterns: common questions and answers," in The patterns handbooks: techniques, strategies, and applications: Cambridge University Press, 1998, pp. 311-319.

[22] D. Perry, E. and A. L. Wolf, "Foundations for the study of software architecture," SIGSOFT Softw. Eng. Notes, vol. 17, pp. 40-52, 1992.

[23] M. W. Maier, D. Emery, and R. Hilliard, "ANSI/IEEE 1471 and systems engineering," Systems Engineering, vol. 7, pp. 257-270, 2004.

[24] P. Kruchten, "The 4+1 View Model of Architecture," IEEE Softw., vol. 12, pp. 42-50, 1995.

[25] 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 \& Training: IEEE Computer Society, 2005.

[26] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, "The Architecture Tradeoff Analysis Method," Fourth IEEE International Conference on Engineering Complex Computer Systems, 1998.

[27] A. BinSubaih and S. Maddoc, "Using ATAM to Evaluate a Game-based Architecture," Workshop on Architecture-Centric Evolution (ACE 2006), June 3-7 2006.

[28] F. O. Bjørnson, A. I. Wang, and E. Arisholm, "Improving the effectiveness of root cause analysis in post mortem analysis: A controlled experiment," Inf. Softw. Technol., vol. 51, pp. 150-161, 2009.

[29] WSU, "Download WSU_KSuite_1.1.2.," http://carl.cs.wright.edu/page11/page11.html, March 12 2009.

[30] T. Lozano-Pérez, In Preface to Autonomous Robot Vehicles. New York, NY: Springer Verlag, 1990.

[31] A. Elfes, "Sonar-based real-world mapping and navigation," in Autonomous robot vehicles: Springer-Verlag New York, Inc., 1990, pp. 233-249.

[32] R. Simmons, "Concurrent Planning and Execution for Autonomous Robots " IEEE Control Systems, vol. 1, pp. 46-50, 1992.

[33] S. A. Shafer, S. A. Stentz, and C. E. Thorpe, "An Architecture for Sensor Fusion in a Mobile Robot," Proceedings of the IEEE International Conference on Robotics and Automation, pp. 2002-2011, April 7-10 1986.

[34] D. Toal, C. Flanagan, C. Jones, and B. Strunz, "Subsumption architecture for the control of robots," ACM SIGCSE Bulletin, vol. 37, pp. 138-142, September 2005.

[35] R. Lumia, J. Fiala, and A. Wavering, "The NASREM Robot Control System and Testbed," International Journal of Robotics and Automation, vol. 5, pp. 20-26, 1990.

[36] Microsoft, "XNA Development Center," http://msdn.microsoft.com/en-us/xna/, March 12 2009.

[37] Microsoft, "The C# Language," http://msdn.microsoft.com/en-us/vcsharp/aa336809.aspx, March 12th 2009.

[38] C. Vichoido, M. Estranda, and A. Sanchez, "A constructivist educational tool: Software architecture for web-based video games," 4th Mexican International Conference on Computer Science (ENC 2003), pp. 8-12, September 2003.

[39] J. Krikke, "Samurai Romanesque, J2ME, and the Battle for Mobile Cyberspace," IEEE Computer Graphics and Applications, vol. 23, pp. 16-23, 2003.

[40] G. Booch, "Best Practices in Game Development," in IBM presentation, 2007.

[41] A. Grossman, Post Mortems from Gamedeveloper: Elsevier, 2003.

[42] R. Darken, P. McDowell, and E. Johnson, "The Delta3D Open Source Game Engine," IEEE Comput. Graph. Appl., vol. 25, pp. 10-12, 2005.

[43] Y. Rabin, Introduction to Game Development: Course Technology Cengage Learning, 2008.

[44] A. Rollings and D. Morris, Game Architecture and Design - A New Edition: New Riders Publishing, 2004.

[45] L. Rising and E. Derby, "Singing the Songs of Project Experience: Patterns and Retrospectives," The Journal of Information Technology Management, vol. 16, pp. 27-33, 2003.

[46] T. Dingsøyr, "Postmortem reviews: purpose and approaches in software engineering," Information and Software Technology, vol. 47, pp. 293-303, 2005.

[47] A. Birk, T. Dingsøyr, and T. Stålhane, "Postmortem: Never Leave a Project without It," IEEE Softw., vol. 19, pp. 43-45, 2002.

[48] R. Scupin, "The KJ Method: a technique for analyzing data derived from Japanese ethnology," Human Organization, vol. 56, pp. 233-237, 1997.

[49] A. I. Wang, "An Extensive Evaluation of Using a Game Project in a Software Architecture Course," Submitted to Transaction on Computing Education (ACM). March 2009.

[50] W. W. Royce, "Managing the Development of Large Software Systems: Concepts and Techniques," Proc. WestCon, August 1970.

[51] A. I. Wang and B. Wu, "An Application of Game Development Framework in Higher Education," International Journal of Computer Games Technology, Special Issue on Game Technology for Training and Education, vol. 2009, 2009.

[52] monoxna-project, "monoxna - Google Code," http://code.google.com/p/monoxna/, April 4 2009.

[53] P. Gestwicki and F.-S. Sun, "Teaching Design Patterns Through Computer Game Development," J. Educ. Resour. Comput., vol. 8, pp. 1-22, 2008.

[54] E. Sweedyk and R. M. Keller, "Fun and games: a new software engineering course," in Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education Caparica, Portugal: ACM, 2005.

[55] 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 Caparica, Portugal: ACM, 2005.

[56] D. Volk, "How to embed a game engineering course into a computer science curriculum," in Proceedings of the 2008 Conference on Future Play: Research, Play, Share Toronto, Ontario, Canada: ACM, 2008.

[57] A. McGovern and J. Fager, "Creating significant learning experiences in introductory artificial intelligence," in Proceedings of the 38th SIGCSE technical symposium on Computer science education Covington, Kentucky, USA: ACM, 2007.

[58] P. Drake and N. Kerr, "Developing a computer strategy game in an undergraduate course in software development using extreme programming," J. Comput. Small Coll., vol. 22, pp. 39-45, 2006.

[59] G. M. Youngblood, "Using XNA-GSE Game Segments to Engage Students in Advanced Computer Science Education," 2nd Annual Microsoft Academic Days Conference on Game Development, February 22-25 2007.

[60] 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 Game development in computer science education Miami, Florida: ACM, 2008.

[61] J. Ryoo, F. Fonseca, and D. S. Janzen, "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 - Volume 00: IEEE Computer Society, 2008.