Teaching a software engineering course on developing video games: a Unified Process versus Extreme Programming

Jason R. Stewart & Arvin Agah

University of Kansas
Lawrence, Kansas, United States of America

ABSTRACT: Video games have become an increasingly complex form of software and one of the most difficult types to develop. In parallel, teaching software engineering courses on developing video games presents a number of challenges. Although video games contain many unique characteristics, they also share many of the same challenges of the other forms of software. The need for a defined process for developing software in any domain has become apparent. A number of software development processes have evolved in recent years. Some of these models can be effectively incorporated in teaching software engineering at the undergraduate level. In this article, the authors endeavour to evaluate and analyse the use of two popular software development processes: namely, the Unified Process (UP) and Extreme Programming (XP), and present the results of an empirical study on the comparison of these models for teaching the software engineering of video games to undergraduate students in computer science and computer engineering.

INTRODUCTION

Video games continue to be one of the most profitable forms of software developed for the general public, grossing around seven billion dollars in US software sales in 2005, a profitable industry with an established audience [1]. The intense interest of many college students in general, and computer science and computer engineering majors specifically in playing video games, makes developing video games a good choice for an undergraduate software engineering course. The advantages of using large projects in software engineering courses are recognised and approaches have been proposed to facilitate their use [2]. In an academic setting, one main challenge is the selection of a game, which can be developed in one academic semester by a team of a few college students, and yet the game is fun and interesting.

The identification of a software development process is an important decision in teaching software engineering. The hypothesis explored in this article is that the choice of the process does not have a major impact, i.e., the use of video games is beneficial regardless of that choice. This article presents the study of two widely-used software development processes namely, Unified Process (UP) and Extreme Programming (XP), and how they suit the development of video games in a college level course in software engineering for computer science and computer engineering majors in their junior/senior years. The approaches were evaluated based on an empirical study performed in the course where several teams of students participated in two game development projects using these two processes.

The Unified Process (UP) is currently the most widely used object-oriented design model for software development [3]. It is structured around four key elements and it is composed of four phases and five disciplines. The four key elements of the Unified Process, as defined by Hunt are: iterative and incremental, user case driven, architecture-centric and risk acknowledging [4]. There are four separate phases that make up UP: inception, elaboration, construction and transition. These phases occur sequentially, with each phase accomplishing a unique set of tasks and producing major milestones and artefacts for the software project. UP is comprised of five distinct disciplines (formerly called workflows) through which the various activities of software development take place: requirements, analysis, design, implementation and testing disciplines. UP is commonly referred to as a two-dimensional process because each of the disciplines is carried out several times throughout the various phases.

Agile methods seek to uphold four principles defined as valuing: individuals and interaction over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan [5][6]. Extreme Programming is a widely-used agile method [7]. XP is more focused on the production of the software product rather than on the documentation that defines and describes the product. It is an incremental approach that focuses on the most important parts of the product as defined by the client; instead of attempting to plan the entire system all at once. XP is based on four main values of communications, feedback, simplicity and courage. XP requires that developers and clients be in constant communication. Feedback allows the

developers and the clients to constantly understand the state of the project. XP contains 12 core practices that enforce the four main values, and practices are broken into three groups of coding, developer and business practices.

There are numerous similarities between video game software and traditional forms of software, such as strict performance requirements, highly optimised code, many usability requirements, affected by other software companies and technological changes, being very large and complex and having multiple independent development teams working towards a common goal. Software engineering standardised approaches to game development have been proposed by Flynt and Salem, using frameworks such as function, object-oriented and patterned [8]. There are a number of unique features of video game software. The most notable characteristics are: fun and entertaining, story driven, gameplay and requirements are from within the company. The development of a story and gameplay are essential parts of video games [9].

RELATED WORK

Abrahamsson et al performed a qualitative study, comparing several different agile methods, including XP, showing that XP, however, lacks support for project inception, system testing, acceptance testing and maintenance, while the system is in use [10]. They also stated that XP is a process that can be adjusted to fit the needs of the situation, but most agile methods are universally defined and not applicable to all situations. Agile software development models have been integrated into software engineering courses [11]. The tradeoffs between using agile methods and plan-driven models in software engineering courses have been studied, showing that both have advantages and disadvantages, depending on project characteristics [12]. Zuser et al performed an analysis of the Rational Unified Process (RUP), Microsoft Solution Framework (MSF) and XP, comparing how well each model supports software quality assurance and software quality development [13]. The results of this study showed that RUP performs well for providing support for software quality and that XP does not provide as much support for software quality. In a similar study, Runeson and Greberg concluded that RUP provides extensive process descriptions, comprising artefacts, roles, activities, integrated tool sets, and that XP instead focuses on values and principles, rather than prescriptive instructions, allowing freedom and simplicity [7].

The results of using XP in software engineering courses have been reported [14][15]. A number of studies have reported on the benefits of using games in software engineering; however, the studies have not focused on the use of different software development processes, which is the topic of this article. Computer games have been shown to motivate students, and have a higher learning effect [16], and in cases where computer games have been used in the software engineering courses [17]. The use of game design in teaching software engineering has been shown to increase interest and enhance retention of students [18]. The benefits of introducing games in computer science curricula have been discussed [19].

The use of computer games has also been reported in object-oriented programming courses [20]. Another approach to teaching software engineering is the use of studio courses, where attention must be given to the selection of suitable projects for students when the students have different backgrounds in software development [21]. The assessment is also a challenge as a formal evaluation requires a pre-class and a post-class assessment. The use of large group projects in computer science courses is advocated by Blake [22]. XP in project-based software development courses, including development of metrics for assessment and monitoring the teaching process, is reported by Dubinsky et al [23].

RESEARCH METHODOLOGY

In order to evaluate UP and XP for developing video games in a software engineering course, an empirical analysis was carried out involving both methods. The experiments were conducted in a 15 week, undergraduate, introductory software engineering course, the first software engineering course required for computer science and computer engineering undergraduate students. This class had 22 students, who were organised into six teams. This resulted in four teams of four people and two teams of three people. Once teams were formed, they were then divided into two groups with three teams in each group. The experiments involved the development of two games in Java using only the built-in features of the language. Each team was asked to develop their product using one of the two models. Teams in group 1 were assigned to use UP for project 1 and XP for project 2 and teams in group 2 used XP followed by UP.

The first project's requirements were based on the Super Nintendo game Super R-Type [24]. This game is a two-dimensional side-scrolling game, in which a player controls a space ship. The player is able to move the ship up, down, right and left on the screen. The player is also able to fire the weapon of the ship to destroy enemies or certain obstacles around them. The game is made up of various levels, each containing multiple obstacles (walls, ceilings, floors, etc) and several types of enemies that attack and move in different ways. At the end of each level, the player must fight a boss, which is more powerful than the other enemies of the level. If the player's ship runs into any of the obstacles or is hit by an enemy ship, either directly or from an attack, the player's ship is destroyed, resulting in the loss of one credit. The game ends when the player's credits are reduced to zero or when all of the levels of the game have been successfully completed by the player. The second project's requirements were based on the Super Nintendo game Castlevania: Dracula X [25]. This is a two-dimensional side-scrolling platform game, in which the player is required to jump or manoeuvre a character while avoiding enemy attacks. The game consists of various levels, where each level varies in

appearance and layout. At the end of each level the player fights a boss. This battle typically takes place in a room with a few platforms that the player must use to avoid attacks from the boss. The player uses weapons to attack.

Each project was broken up into a set of five steps. Each of these steps corresponded to the completion of a phase and/or discipline in UP or a build in XP. The deliverables due at the end of each step varied depending on the development model being used. Two methods were used for evaluating the projects developed by the student teams, namely, a set of questionnaires and a set of metrics to analyse the software products. Two questionnaires were given to the students, one after each of the projects was completed. The questionnaires included questions on the students' development of the software and their feelings towards the various practices of the development mode. A set of defined metrics was used to examine the code developed by each of the teams, including 22 metrics collected using the Eclipse Metrics plug-in [26-28]. Full experimental results are reported in Stewart's work [29]. The development activities were grouped into four categories of requirements: gathering and analysis, system design, system implementation and system testing (Table 1).

Table 1: Development activity categories and the corresponding UP and XP activities.

Activity	UP	XP
Requirements gathering and analysis	Requirements and analysis disciplines	Story cards
System design	Design discipline	Task cards
System implementation	Implementation discipline	Implementation
System testing	Testing discipline	Testing

QUESTIONNAIRE RESULTS

The students ranked the four activities based on the amount of time needed to complete them. For project 1, in both models the activities for requirements gathering and analysis required the shortest amount of time for the majority of the students. Testing and implementation were clearly the categories of longest activities for XP. These activities were also the most time consuming for the UP teams. For project 2, XP teams reported that testing and implementation took the longest time. However, implementation was voted to be the longest activity among the XP teams instead of testing activities as in project 1. The ranking of the activities for the UP teams remained unchanged, with implementation being identified as the longest activity among the students. Requirements gathering and analysis was again reported to be the shortest activity for both models.

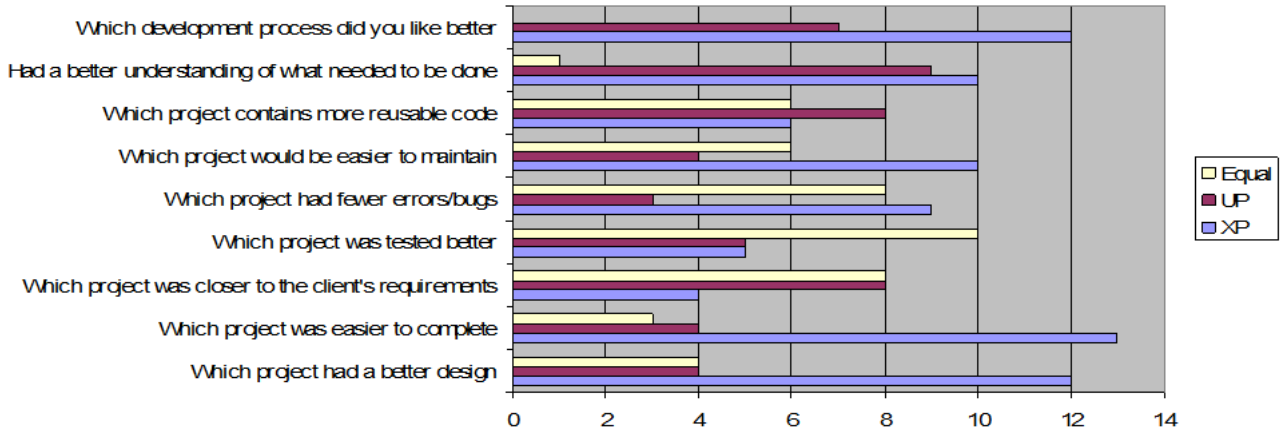

Figure 1: Students' comparison of XP and UP for development.

Students ranked the four activities in terms of their difficulty. For project 1, the most difficult activity was implementation for both models; and the requirements and analysis were the easiest. The students using XP identified the testing activity to be one of the more difficult activities, while the students using UP identified it as one of the easier activities. For project 2, implementation was reported for both models to be the most difficult activity, while the requirements gathering and analysis were voted the easiest. Testing was again stated to be one of the more difficult activities for the XP teams. Even with the test-driven development practice, there seems to be a trend towards XP teams having more difficulty testing their software than the UP teams, who identified design to be their second most difficult.

The second questionnaire included an extra set of questions that focussed on the students' opinions on various aspects of their software based on the two development models (Figure 1). Each question had three possible choices for an answer, XP, UP or equal for cases where the students felt that both development models performed equally. Although XP stresses a simple design, the majority of students felt that XP produced a better design for their program than UP. Even with XP's test-driven development, over half the students felt that UP performed equally or better for testing the code. A majority of the students felt that XP was equal or better at developing code with fewer defects, and that this

code would be easier to maintain. The majority of the students felt that using XP was easier to complete than using UP. The students felt that the project developed using UP was as close as or closer to the requirements of the client than XP.

INDIVIDUAL TEAM METRIC RESULTS

The software metrics of complexity, coupling, cohesion, size and reusability for each team were examined individually to determine if one development model consistently produced higher quality code. The project model for each team that had the greater number of metrics in its favour for each of the measures is listed in Table 2. There was little evidence that either model consistently performed better in any of the five categories. The only possible trend that can be identified is that the projects using UP in all except one case produced code with more cohesion.

However, performing a paired student's *t*-test on the cohesion metrics showed that this cannot be stated with 95% confidence. Another trend that seems to be apparent in the results is that project 2 was a larger project than project 1. Even though UP stresses a good architecture and has a discipline devoted to system design, it did not perform any better than XP in the measures of coupling and cohesion. The complexity measures of the XP projects were not consistently lower than that of the UP projects, even though XP stresses a simple design and constant code refactoring.

Table 2: Software measures for each team showing the development model that had the most metrics in its favour.

Team	Complexity	Coupling	Cohesion	Size	Reusability
A	XP	UP	UP	UP	UP
B	UP	UP	UP	UP	XP
C	UP	equal	UP	UP	XP
D	XP	equal	XP	XP	UP
E	UP	equal	UP	XP	UP
F	XP	XP	UP	UP	UP

PROJECT METRIC RESULTS

The projects were examined individually to determine if one development model consistently produced higher quality code. The results for each of the projects were compiled by taking the averages of each of the metrics for the teams using the same development model for the project.

Project 1: The projects using UP were slightly less complex. The UP and XP complexity metrics were very close, except for the weighted methods per class. However, UP showed better performance in four of the nine metrics. XP showed better performance in one of the metrics and the two models were equal in four of the metrics. Similarly, the coupling and cohesion metrics both showed better performance for UP. The results of the coupling metrics (Figure 2) showed that UP performed better in two of the four metrics, while XP performed better in one. Again, the difference between these averages was very small.

The results of the cohesion metrics (Figure 3) showed that UP barely edged out XP in performance in two of the three metrics. The weighted methods per class metric showed a more substantial difference between the performances of the models than the previous two and again showed UP as performing better than XP. The results of the average size metrics showed that UP resulted on average in smaller product code than XP. UP size metrics were always less than or equal to the size metrics of XP. On average, the XP teams produced about 900 more total lines of code than the UP. The reusability metrics showed that XP performed better on average. XP performed better on average in five of the nine metrics, while two of the metrics showed equal performance.

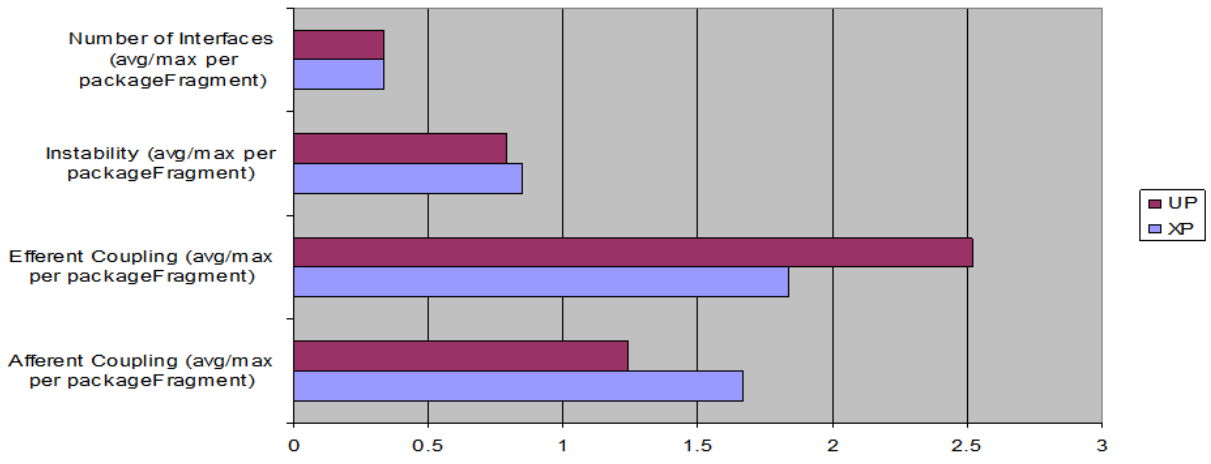

Figure 2: The average of the coupling metrics for XP versus UP in project 1.

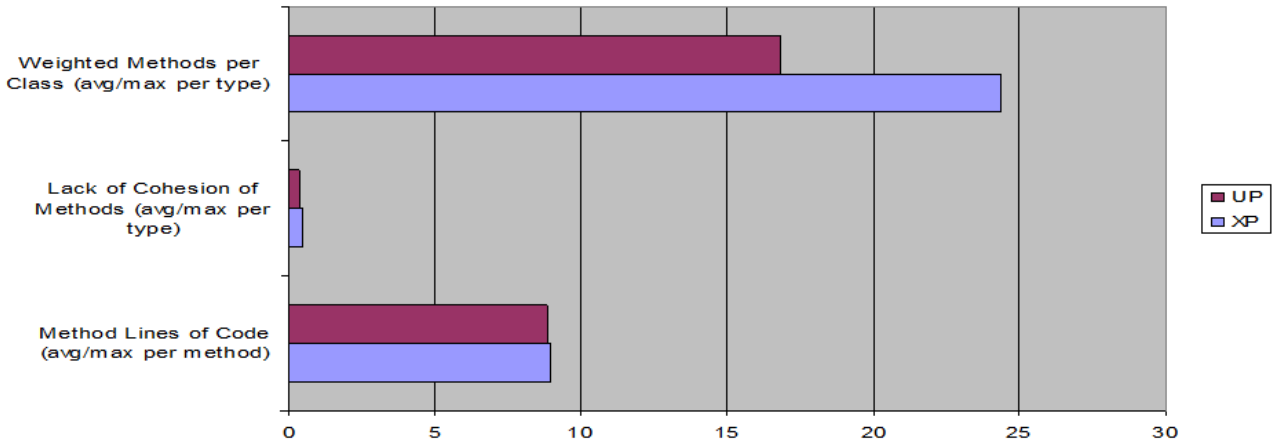

Figure 3: The average of the cohesion metrics for XP versus UP in project 1.

Project 2: In terms of the complexity and coupling metric, both models showed approximately equal performance. XP and UP both performed better than the other in four metrics, while they showed equal performance in the average nested block depth metric. The average coupling metrics showed that each model performed better than the other in two of the metrics. The average cohesion and size metrics showed the opposite results of those obtained for project 1. XP performed better than the UP projects on average. The average cohesion metrics illustrated that two of the three metrics had better performance in XP than in UP. The size metrics (Figure 4) showed that all the XP metrics were always less than or equal to those of UP. The only metrics in which the two models were equal were the number of packages and number of interfaces metrics (both were zero). The average metrics for the reusability measurement showed that UP performed better for these metrics. UP performed better in seven of the nine metrics. Most of the differences of the two models were very small.

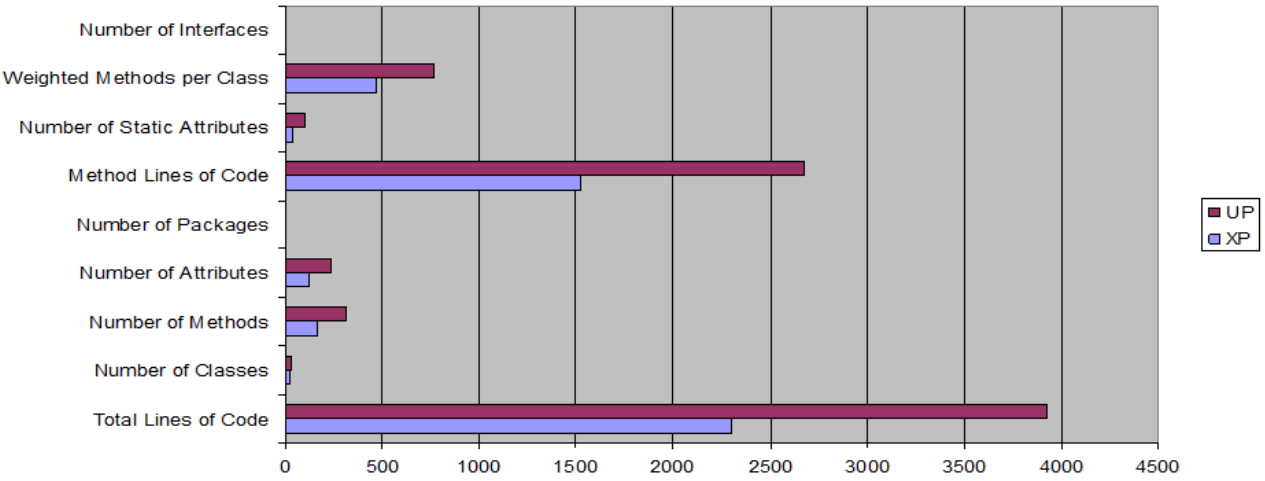

Figure 4: The average of the size metrics for XP versus UP in project 2.

The results of these measurements provided no evidence that either model produces higher quality code. The project model for each team that had the greater number of metrics in its favour is listed in Table 3. The model that produced the best average performance for the measures in project 1 is either the opposite or equal between both the models in project 2. This would suggest that the metrics were affected more by the individual teams and less by the development model being used. Additional results are reported by Stewart [29].

Table 3: Software measures for projects showing the development model that had the most average metrics in its favour.

Project	Complexity	Coupling	Cohesion	Size (smaller)	Reusability
1	UP	UP	UP	UP	XP
2	Equal	Equal	XP	XP	UP

CONCLUSION

As the video game software industry continues to grow, a software development process must be identified that conforms to this software's difficult and unique characteristics. The development process should be utilised in teaching software engineering. This article has examined two commonly used software development processes of Extreme

Programming (XP) and Unified Process (UP), evaluating the models using a group of undergraduate students during the development of two video game software projects.

The results have shown that XP and UP were seemingly equivalent in most aspect when compared on the basis of the quality of the code that was produced. In both models, implementation appeared to be the most challenging and the most time consuming activity. Requirements gathering and analysis activities appeared to be the least difficult and least time consuming. The majority of the students using XP reported testing to be one of the more difficult development activities, more than the students using UP did, even though XP enforces test-driven development.

The majority of students also felt that their code was tested as well or better when using UP over XP. However, the majority of students believed that XP was equal or better than UP for generating code that was free of defects. Another observed fact is that XP appeared to produce a better system design than UP, even though the UP is based around the key element of being architecture-centric, and XP focuses on a simple design and only the task at hand. More students also felt that the project that used UP resulted in software that conformed more to the client's requirements.

Overall results indicate that the choice of the software development process does not have a major impact on the overall software engineering course. The use of video games in software engineering courses has been shown to be beneficial by a number of studies and, therefore, whether the course uses XP or UP, the course can be structured around the development of video games [16-20].

REFERENCES

1. Entertainment Software Association. Facts and Research: Sales and Genre Data, 1 December 2011, http://www.theesa.com/facts/sales_genre_data.php.
2. Coppita, D., Implementing large projects in software engineering courses. *Computer Science Educ.*, 16, **1**, 53-73 (2006).
3. Schach, S.R., *Object-Oriented and Classical Software Engineering*. (6th Edn), New York, New York: McGraw-Hill (2005).
4. Hunt, J., *Guide to the Unified Process*. (2nd Edn). London, England: Springer (2003).
5. Agile Alliance, 1 December 2011, http://www.agilealliance.org.
6. Agile Manifesto, Manifesto for Agile Software Development, 1 December 2011, http://agilemanifesto.org/.
7. Runeson, P. and Greberg, P., Extreme Programming and Rational Unified Process - Contrasts or Synonyms, 1 December 2011, http://serg.telecom.lth.se/research/publications/docs.
8. Flynt, J.P. and Salem, O., *Software Engineering for Game Developers*. Boston, Massachusetts: Muska and Lipman/Premier-Trade (2004).
9. Rollings, A. and Adams, E., *Andrew Rollings and Ernest Adams on Game Design*. Indianapolis, Indiana: New Riders Publishing, (2003).
10. Abrahamsson, P., Warsta, J., Siponen M.T. and Ronkainen, J., New directions on Agile Methods: a comparative analysis. *Proc. 25th Inter. Conf. on Software Engng.*, Portland, Oregon, 244-254 (2003).
11. Hislop, G.W., Lutz, M.J., Naveda, J.F., McCracken, W.M., Mead, N.R. and Williams, L.A., Integrating agile practices into software engineering courses. *Computer Science Educ.*, 12, **3**, 169-185 (2002).
12. Boehm, B., Port, D. and Brown, A.W., Balancing plan-driven and agile methods in software engineering project courses. *Computer Science Educ.*, 12, **3**, 187-195 (2002).
13. Zuser, W., Heil, S. and Grechenig, T., Software quality development and assurance in RUP, MSF and XP: a comparative study. *Proc. Third Workshop on Software Quality*, St. Louis, Missouri, 1-6 (2005).
14. LeJeune, N.F., Teaching software engineering practices with Extreme Programming. *J. of Computing Sciences in Colleges*, 21, **3**, 107-117 (2006).
15. Noble, J., Marshall, S., Marshall, S. and Biddle, R., Less Extreme Programming. *Proc. Sixth Conf. on Australian Computing Educ.*, Dunedin, New Zealand, 217-226 (2004).
16. Sindre, G., Natvig, L. and Jahre, M., Experimental validation of the learning effect for a pedagogical game on computer fundamentals. *IEEE Transactions on Educ.*, 52, **1**, 10-18 (2009).
17. Sweedyk, E. and Keller, R.M., Fun and games: a new software engineering course. *Proc. 10th Annual SIGCSE Conf. on Innovation and Technol. in Computer Science Educ.*, Caparica, Portugal, 138-142 (2005).
18. Claypool, K. and Claypool, M., Teaching software engineering through game design. *ACM SIGCSE Bulletin*, 37, **3**, 123-127 (2005).
19. Sweedyk, E., deLaet, M., Slattery, M.C. and Kuffner, J., Computer games and CS education: why and how. *Proc. 36th SIGCSE Technical Symposium on Computer Science Educ.*, St. Louis, Missouri, 256-257 (2005).
20. Chen, W-K. and Cheng, Y.C., Teaching object-oriented programming laboratory with computer game programming. *IEEE Transactions on Educ.*, 50, **3**, 197-203 (2007).
21. Laplante, P.A., An agile, graduate, software studio course. *IEEE Transactions on Educ.*, 49, **4**, 417-419 (2006).
22. Blake, M.B., Integrating large-scale group projects and software engineering approaches for early computer science courses. *IEEE Transactions on Educ.*, 48, **1**, 63-72 (2005).
23. Dubinsky, Y. and Hazzan, O., A framework for teaching software development methods. *Computer Science Educ.*, 15, **4**, 275-296 (2005).

24. Gamespy, Hardcore Gaming 101 - R-Type. Hardcore Gaming 101, 1 December 2011, http://hg101.classicgaming.gamespy.com/rtype/rtype.htm.
25. Gamespy, Castlevania: Dracula X. The Castlevania Games, 1 December 2011, http://castlevania.classicgaming.gamespy.com/Games/dxx.html.
26. Sauer, F., Eclipse, 1 December 2011, Metrics. http://metrics.sourceforge.net/.
27. Henderson-Sellers, B., *Object-Oriented Metrics: Measures of Complexity*. Upper Saddle River, New Jersey: Prentice Hall PTR (1995).
28. Martin, R.C., *Agile Software Development: Principles, Patterns, and Practices*. Upper Saddle River, New Jersey: Prentice Hall, (2002).
29. Stewart, J.R., Should One be Unified or Extreme when Developing Entertainment Software? MS Computer Science Thesis, University of Kansas (2006).