Decreasing Rework in Video Games Development from a Software Engineering Perspective

Hugo A. Mitre-Hernández
Centro de Investigación en Matemáticas (CI…
21 PUBLICATIONS **22** CITATIONS

Carlos Lara-Alvarez
Centro de Investigación en Matemáticas (CI…
18 PUBLICATIONS **25** CITATIONS

Decreasing Rework in Video Games Development from a Software Engineering Perspective

Hugo A. Mitre-Hernández[1], Carlos Lara-Alvarez[1], Mario González-Salazar[1], Diego Martín[2],

[1] Center for Research in Mathematics (CIMAT), Computer Science Department,
Av. Universidad 222,
98068 Zacatecas, Mexico
{hmitre, carlos.lara, remylebv}@cimat.mx

[2]Technical University of Madrid, Telematics Department,
Av. Complutense 30,
28040. Madrid, Spain
diego.martin.de.andres@upm.es

Abstract. Video game industry is becoming increasingly important due to its revenues and growing capabilities. Information complexity and process agility are limitations for developing a videogame and they may lead to rework. Many rework problems are related to unspecified or ambiguous requirements in game design. For reducing rework, this article proposes an agile development process for video games that aligns the Scrum instance of the software development Project Pattern (sdPP) and the improved Game Design Document (iGDD). For measuring the rework induced by different alternatives, we conducted a case study that compares the proposed approach against a conventional counter proposal in game industry; the results prove that our proposal generates less normalized rework than the counter proposal.

Keywords: Rework, Video Games, Software Engineering, Requirements Engineering.

1 Introduction

Video games are important economically, as innovative leaders and as an alternative to solve issues outside the entertainment area; they constitute the main entertainment industry, with continuous growth since their appearance and billions of dollars in sales and revenues [1]. Video games have proven useful outside the entertainment area with good results for solving problems, training, diagnosing, predicting, and teaching among others [2].

In the video game industry, the development process is commonly composed by three stages: pre-production, production and post-production [3]. The *pre-production* stage focuses mainly on creating the game concept and design. The *production* stage creates and validates the software; this stage also produces sounds and music required by the game. Finally, the *post-production* stage distributes and maintains the game; it also manages the feedback coming from different sources –i.e. specialized video game reviewers–.

Some games are complex systems requiring significant effort in the first two stages; these complexities can increase the amount of rework and consequently, the cost of the game. *Rework* is defined as any additional effort required for finding and fixing problems after documents and code are formally signed-off as part of configuration management [4]. Thus, end-phase verification and validation are usually excluded, but debugging effort during integration and system testing is included. To compare different products, rework effort is sometimes "normalized" by being calculated as a percentage of development effort [4]; the rework is generally considered to be potentially avoidable work that is triggered to correct problems.

Rework issues in game development usually start at the pre-production stage. It has been reported that 65% of problems in game development are generated at this stage and are related to unspecified or ambiguous requirements in game design [5, 6]. At the production stage, developers can ask the game designer for clarification of the missing information or can make their best assumption. In either case, a rework is done; even, rework can rise when wrong assumptions about the requirements are made. Hence, we need a way to describe the game design with enough formality and detail, a Software Requirements Specification (SRS) document can be used for designing the software of the game [7, 8]. In sum, a more formal requirement specification is needed to avoid rework problems in the game development process.

The game *design* is central in game development; this activity transforms an idea into a detailed description of the game. Formally, the game design is the process of imagining a game, defining the way it works, describing the elements that make up the game (conceptual, functional, artistic, and others), and transmitting that information to the team that will build the game [9]; this information is reflected in the *Game Design Document* (GDD).

We consider that the rework can be substantially reduced in the pre-production stage. The approach proposed in this article is composed by (i) a project pattern adapted to game agile development, and (ii) an improved GDD [10] based on requirements engineering. Software process patterns encapsulate the solution for a specific development project; therefore, it is possible to adapt them to the game development process. On the other hand, the requirement engineering is helpful because it offers characteristics such as correctness, unambiguously, completeness, consistency, stability, verifiability, modifiability, traceability [11] and structurability [12].

The rest of this article is organized as follows: section 2, summarizes the game development methodologies directly related to our approach; section 3, introduces the proposed approach; section 4 compares empirically our approach against a classical methodology for game development in terms of rework; section 5, discusses the results; and finally, section 6 concludes this work.

2 Related Work

As stated earlier, our proposal aligns a project pattern with the improved GDD; the following paragraphs overview the work related to these two components.

Development Project Pattern

The videogame development is a form of software development that adds additional requirements, e.g., artistic aspects; hence, many of the management tools and standards from the software industry can be useful for game development. Game projects are usually more complicated than software projects because they involve a multidisciplinary team and they usually have more uncertainty around project goals.

Software development models –e.g, waterfall, iterative, or extreme– can be used for developing videogames [13]. In general, the waterfall model is considered inadequate because it is highly structured and it cannot be adapted to changes in the requirements; therefore, more flexible models are needed [14–16].

Agile methodologies –i.e. Scrum [17] or eXtreme Programming [18]– are better suited for the challenges of game development [19, 20]; they have been adapted to game development by using other tools as complements: user stories [19], game design documentation [21], or workshops for strengthening the interaction between clients and developers [22].

Patterns [23] are used to solve a generic problem: given a narrative and context of the problem to be solved, they propose a solution. They can be used for formalizing the knowledge about the development process; [24] proposes the *Software Development Project Pattern* (sdPP) framework. For testing this approach [24], generates four instances of the sdPP with agile development models; one of these instances –Scrum sdPP– is suitable for game agile development because it allows to follow an iterative process without sacrificing creativity. The resulting workflow and productflow can guide game developers between the activities and their corresponding input and output products.

Improved Game Design Document

The *Mechanics, Dynamics and Aesthetics* (MDA) framework [14] is an iterative approach for designing and tuning video games; it first defines the aesthetics, then the dynamics –trying to fulfill aesthetics–, and finally the game elements –that bring the required interaction–; this framework offers an interesting way of designing and tuning games, but it does not provide the tools or methods to construct the details of such mechanics, dynamics and aesthetics. On the other hand, several approaches suggest to use different abstraction levels for the game design documentation, but they are too specific for certain types of games [25, 26].

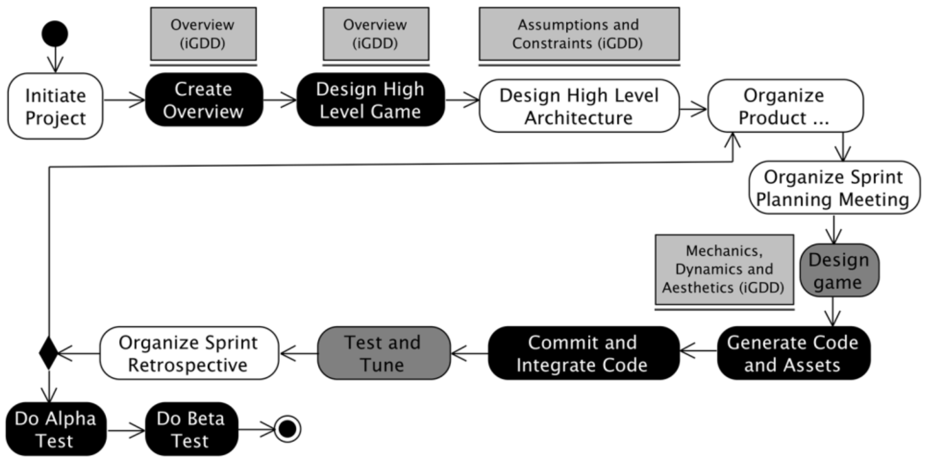
Table 1 Description and characteristics of sections of the iGDD [10].

iGDD Section	Description	SRS Characteristics
Overview	Describes briefly the most important aspects of the game.	Relations with other documents, and common language for better understanding.
Mechanics	Describes the elements of the game.	Organization of game requirements (objects organization).
Dynamics	Describes how the elements of the game will take action in the game.	Organization of game requirements. Relation of complexity with gamer profile.
Aesthetics	Describes what the player perceives directly through their sense, like what he sees and hears.	–
Experience	Highlights important aspects of the game and what you hope to achieve from these aspects.	Decision making based on tradeoffs of game parts. Quality attributes on video games.
Assumptions and constraints	Narrates the aspects of the design assumptions and limitations of the game, either technical or business.	Knowledge of game parts for reviews. Limitations or boundaries of video game

The *Game Design Document* (GDD) plays a key role for every game project, i.e., a poorly elaborated GDD can lead to rework and loss of investment in production and postproduction phases; to address these issues, in [10] we propose the *improved GDD* (iGDD). The iGDD has different abstraction levels [9]; it is based on the Taylor's GDD template [25], but it incorporates the (MDA) framework and the best practices from Software Requirements Specification (SRS) [7, 8]; sections of the iGDD are shown in **Table 1**.

3 Proposed Approach

The sdPP model is defined as a problem–solution pair; in our case, the solution is closely related to agile game development with Scrum and the use of the iGDD into the development process. Proposed improvements to the Scrum sdPP are: (i) the alignment of activities described in the sdPP's workflow and productflow to the iGDD; and (ii) the incorporation of the iGDD sections to the Scrum sdPP as products. For adapting Scrum sdPP to game developing new activities were added while others were modified; some of these activities are associated to iGDD sections (products) as shown in Fig. 1. The modified (or added) activities and their associated iGDD products are described in the following paragraphs:



Overview (iGDD)
Overview (iGDD)
Assumptions and Constraints (iGDD)
Initiate Project
Create Overview
Design High Level Game
Design High Level Architecture
Organize Product ...
Organize Sprint Planning Meeting
Mechanics, Dynamics and Aesthetics (iGDD)
Design game
Generate Code and Assets
Commit and Integrate Code
Test and Tune
Organize Sprint Retrospective
Do Alpha Test
Do Beta Test

Fig. 1. Productflow of the Scrum sdPP adapted to agile video game development. The added activities are in *black* and the modified ones are in *gray*; these activities are associated to iGDD sections (products) in *light gray*.

- **Create overview (product: *overview* of the iGDD** [10]**),** describe the game in a brief abstract, identify the main objectives of the game, the genre of the game, ask questions –e.g., why the game is worth doing–, define which type of players would like to play the game, and what will be the main activities that the player will be doing while playing the game.
- **Design High level game (product: *overview* of the iGDD** [10]**),** define some main features of the game: the game modalities (single player, multiplayer, online, arcade mode, history mode, among others), the platform or platforms on which the game is intended to run, the game theme (medieval, futuristic, western, among others), the game story and an initial scope of the levels, size and time that the game may require.
- **Design high level game architecture (product: *assumptions and constraints* of the iGDD** [10], Table 1**)**. Review the technical settings to modify the *assumptions and constraints* and determinate any technical constraint that the game may have. *Technical settings* include: the standards, conventions, technology, resources and architecture selected for the game.
- **Design Game (products: *mechanics, dynamics and aesthetics* of the iGDD** [10] , Table 1**)**. The design of a level of the game will involve all three categories, while the design of the main character and all his action will involve only mechanics and aesthetics.
- **Generate code and asset**. Create game elements, as suggested by Keith [19]. These elements include code and assets, i.e., music or animations.

- **Commit and integrate code**. Integrate game elements, as suggested by the Agile Alliance [27]; a totally integrated game allow to have "at any time" a version suitable for release.
- **Do test and tune.** Test the resulting product of the sprint, as suggested byKeith [19]; small adjustments can be made to polish the game, but radical changes should be placed in the product backlog to consider them in the next sprint. The result from this activity will be a potentially shippable product.
- **Do alpha and beta tests**. Find and remove bugs in alpha test [9, 19, 28], and test the experiences of the possible market that will play the game in the beta test.

4 Research Process

The research process is based on guidelines proposed in [29] for empirical evaluation in software engineering.

4.1 Plan

The objective of the case study is to measure whether or not our approach helps to decrease rework while developing a game. For measuring the rework, any artifact put to test for the first time starts to register rework time after the test is done.

Twelve junior software engineers at the Center for Research in Mathematics carried out this empirical study; these engineers (hereafter, participants) were students of the videogame course in the software engineering master degree program. For our study two groups were considered:

Group A: uses the approach proposed in this article;

Group B: uses the Taylor's GDD [25] and the agile game development with Scrum [19].

4.2 Execution

The following activities were carried out:

1. **Team Creation**. The professor interviewed the students for knowing their experience in similar projects and created two groups as homogeneous as possible.
2. **Team Training**. The researcher trained the teams in the required design documents and development methods.
3. **Conduct Experiment and Data Collection**. The researcher conducted a sprint planning meeting with each team. In this meeting the base requirements and its priorities were presented. The team divided the base requirements into user stories for planning the sprint. Then, the teams

worked on the user stories of the sprint for two weeks recording the time spent on each user story. The researcher conducted a sprint end meeting where the user stories were classified as completed, unfinished or rejected, and the review of the correct use of objects (GDD and game development model) used by teams during the sprint.

Finally, each team and researcher planned the date for the next sprint planning meeting, ensuring that the time difference between sprint end and sprint planning was less than five days. This process continued until the base requirements were finished or the time available ended. The time for these steps were three months, where the teams should have from four to five sprints and finish from ten to fifteen base requirements.

4.4 Results

The data of times and user stories associated with base requirements collected during the case study were recorded in the log of Kanban web. The data was validated at the end of each sprint planning.

The Wilcoxon signed-rank test was used to compare the rework of groups A and B.

As shown in Fig. 2, the normalized mean of rework for group A was 2.73%, while for group B was 11.50%. A Wilcoxon signed-rank test showed that the iGDD + sdPP technique induces significantly less rework than the rework induced in the group that used GDD + Agile GD with Scrum ($p = 0.0085$). *This proves that our proposal generates less rework than the counter proposal*.

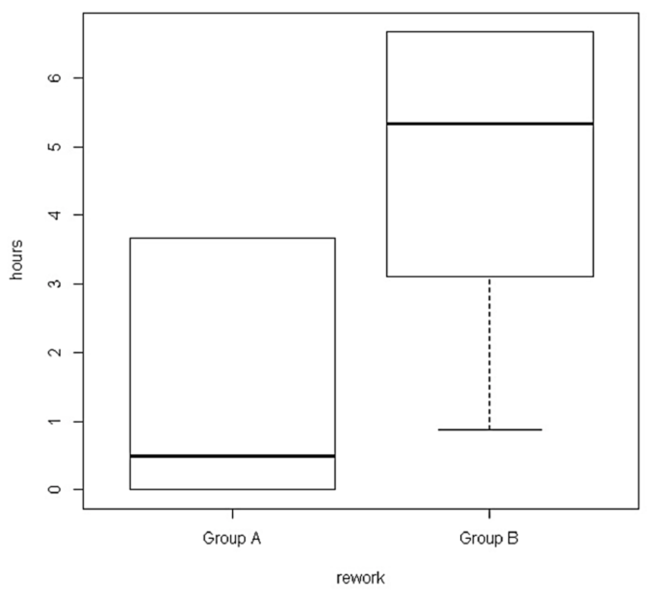


Fig. 2. Productflow Comparison of rework between groups A and B.

5 Discussions

We observe the following benefits of the iGDD for reducing rework:

- By defining the overview section of the iGDD, participants clarify the information of the objectives, justification, gameplay, game features and player characteristics. This section allows to easily interpreting the requirements in the following stages.
- By defining the assumptions and constrains of the iGDD, participants discover the context and limits of the high level architecture. In terms of requirements engineering, we learned that limitations and boundaries clearly defined can lead to a feasible development.
- By defining the mechanics and dynamics of the iGDD, participants obtain: (i) well organized game elements, and (ii) good alignment of complexity with gamer profile. The traceability of game elements facilitated participants to reduce rework in the development of game elements.

Our main observation to the adapted Scrum sdPP was the traceability of the activities to the iGDD sections, allowing participants to retrieve information about the game elements at any time during process execution.

Summarizing the opinions of the participants, the adapted sdPP is clearer, more detailed, concise, and accurate than the conventional approach. Moreover, the documentation was considered better structured, and it requires less effort in interpretation and design.

The main limitation in the case study is that the teams didn't have the time for creating game elements –e.g., the sound part of the game–; hence, external sources were used to complete the game. Another observed limitation is that it is hard to coordinate the scrum meeting when the members of a team have different schedules.

6 Conclusions and Further Work

This article describes how software development patterns make easer the use of agile game development process; the proposed approach is composed by an agile development process –an adapted Scrum instance of the sdPP– and an improved game design document iGDD that takes advantage of the requirements engineering perspective.

A case study was conducted, validating that our proposal generates less rework than a conventional counter proposal. Finally, aspects that contributed to the reduction of rework with the use of the iGDD and the sdPP's instance were discussed.

We consider that the proposed approach not only reduces the rework but also can give better quality products that enhance the user experience. On the other hand, a management tool can be used for increasing the productivity of medium or large scale game projects. We are studying empirically both the quality and productivity induced by this approach.

Acknowledgements

This research was partially funded by the National Council of Science and Technology of Mexico (CONACyT) through the project "Strengthening of the master of software engineering program with the integration the research line in Human-Computer Interaction" (ZAC-2013-C04- 226098) and the project "Optimization of industrial processes based on simulators, interfaces and software assurance" (CATEDRAS-3163).

References

1. Essential facts about the computer and video game industry. (2014).
2. Jason: Gaming is Good for You (Infographic), http://www.affordableschoolsonline.com/gaming-is-good-for-you-infographic/.
3. Bethke, E.: Game Development and Production. Wordware Publishing Inc.; Pap/Cdr edition (2002).
4. Kitchenham, B., Pfleeger, S.L.: Software quality: The elusive target. IEEE Softw. 12–21 (1996).
5. Petrillo, F., Pimenta, M., Trindade, F., Dietrich, C.: What Went Wrong ? A Survey of Problems in Game Development. Comput. Entertain. 7, 1–22 (2009).

6. Petrillo, F., Pimenta, M., Trindade, F., Dietrich, C.: Houston, we have a problem...: a survey of actual problems in computer games development. In: ACM symposium on Applied computing. pp. 707–711. ACM (2008).
7. Callele, D., Neufeld, E., Schneider, K.: Requirements engineering and the creative process in the video game industry. In: Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on. pp. 240–250. IEEE (2005).
8. Callele, D., Neufeld, E., Schneider, K.: A report on select research opportunities in requirements engineering for videogame development. 2011 Fourth Int. Work. Multimed. Enjoyable Requir. Eng. 26–33 (2011).
9. Rollings, A., Adams, E.: Andrew Rollings and Ernest Adams on Game Design. New Riders; 1 edition (2003).
10. Gonzalez, M., Mitre, H.A., Lemus, C., Gonzalez, J.L.: Proposal of Game Design Document from software engineering requirements perspective. In: 2012 17th International Conference on Computer Games (CGAMES). pp. 81–85. IEEE (2012).
11. Wiegers, K.: Software Requirements 2. Microsoft Press (2003).
12. IEEE SA - 830-1998 - IEEE Recommended Practice for Software Requirements Specifications, http://standards.ieee.org/findstds/standard/830-1998.html.
13. Bethke, E.: Game Development and Production. Wordware Publishing Inc.; Pap/Cdr edition (2002).
14. Hunicke, R., LeBlanc, M., Zubek, R.: MDA: A formal approach to game design and game research. In: Proceedings of the AAAI Workshop on Challenges in Game AI. pp. 04–04 (2004).
15. Schell, J.: The Art of Game Design: A book of lenses [Paperback]. Morgan Kaufmann (2008).
16. Boehm, B.W.: A spiral model of software development and enhancement. Computer (Long. Beach. Calif). 21, 61–72 (1988).
17. Schwaber, K., Beedle, M.: Agile Software Development with Scrum. Pearson Education International (2002).
18. Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley Longman Publishing Co (1999).
19. Keith, C.: Agile Game Development with SCRUM. Addison Wesley; 1 edition (2010).
20. Kasurinen, J., Laine, R., Smolander, K.: How Applicable Is ISO / IEC 29110 in Game Software Development? In: 14th International Conference, PROFES 2013. pp. 5–19. Springer Berlin Heidelberg (2013).
21. Godoy, A., Barbosa, E.: Game-Scrum: An Approach to Agile Game Development. IX SBGames. (2010).
22. Kortmann, R., Harteveld, C.: Agile game development: lessons learned from software engineering. In: Learn to Game, Game to Learn; the 40th Conference ISAGA 2009. Society of Simulation and Gaming of Singapore (2009).
23. Alexander, C.: The Timeless Way of Building : Oxford University Press (1979).
24. Martín, D., Guzmán, J.G., Urbano, J., Llorens, J.: Patterns as objects to manage knowledge in software development organizations. Knowl. Manag. Res. Pract. 10, 252–274 (2012).
25. Taylor, C.: Design Template, http://www.runawaystudios.com/articles/chris_taylor_gdd.asp.
26. Rogers, S.: Level Up!: The Guide to Great Video Game Design [Paperback]. John Wiley &amp; Sons (2010).
27. Agile Alliance: Continuos Integration, http://guide.agilealliance.org/guide/ci.html.
28. Brinkkemper, S., Weerd, I., Weerd, S.: Developing a Reference Method for Game Production by Method Comparison. IFIP Adv. Inf. Commun. Technol. 244, 313–327 (2007).
29. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer Berlin Heidelberg, Berlin, Heidelberg (2012).