Software Architectures and the Creative Processes in Game Development

Alf Inge Wang[1] and Njål Nordmark[2]

Norwegian University of Science and Technology,
N7491 Trondheim, Norway
alfw@idi.ntnu.no, njaal.nordmark@gmail.com

Abstract. Game development is different from traditional software engineering in that there are no real functional requirements and the customers buy and use the software only because it is engaging and fun. This article investigates how game developers think about and use software architecture in the development of games. Further, it looks at how creative development processes are managed and supported. The results presented in this article come from responses to a questionnaire and a survey among thirteen game developers. The research questions answered in this study are: what role does the software architecture play in game development, how do game developers manage changes to the software architecture, how are creative development processes managed and supported, and how has game development evolved the last couple of years. Some of our findings are that software architectures play a central role in game development where the focus is mainly on achieving software with good performance and high modifiability, creative processes are supported through flexible game engines and tools, use of scripting and dynamic loading of assets, and feature-based teams with both creative and technical professions represented, and game developers are incrementally using more game-specific engines, tools and middleware in their development now compared to earlier.

Keywords: Game development, Creative software development, Software architecture.

1 Introduction

Game development can be incredibly challenging as game technology such as game engines and game platforms changes rapidly, and code modules crafted for specific games offer less than 30 percent reuse [1]. In the early days of the video games era, game development was carried out by small teams, where the software architectures were made out of a few modules such as 2d graphics, simulation, sound, streaming of i/o and main. At this time, there was not much focus on software architecture and software engineering, but rather on how to how to create an interesting game with the limited hardware resources available. The success of the video game industry, the development of game technology and the increasing demands from the players have resulted in large and complex games developed by large teams of multiple professions. The evolution of games has also resulted in an evolution of game architectures that have grown in size and complexity [2]. Today, some game projects are very large, and the game software itself has a complex software architecture with many interconnected modules. One aspect that makes software architectures for games challenging is the absolute real-time requirements and the need to support the creative processes in game development [1]. Further, the development of AAA games (major game titles) requires a multitude of computer science skills [3] as well as other disciplines as art, game design, and audio/music [4]. The direct involvement of professions with very different background (e.g. the technical team vs. the creative team) poses challenges for how a game is developed.

So far, research related to game development and software engineering has focused on requirement engineering, and there is a lack of empirical work [5]. This article presents a study on how game developers think about and manage software architecture and the creative processes that characterize game development. The study investigates the relationship between creative design and software development, and how the technical and creative teams collaborate.

2 Material and Method

In this section, we present other research relevant to this article as well as the research goal, the research questions and the research method used.

2.1 Related Work

As far as we know, there are no similar studies that focus on software architecture and creative processes in game development. In this section, we will present work in the field of software engineering and game development.

As games over the years have grown into large complex systems, the video game industry is facing several software engineering challenges. Kanode and Haddad have identified the software engineering challenges in game development to be [6]: *Diverse Assets* – game development is not simply a process of producing source code, but involves assets such as 3D modes, textures, animations, sound, music, dialog, video, *Project Scope* – the scale of a video game can be massive and the game industry is known for poorly describing and defining project scope, *Game Publishing* - brining a video game to market involves convincing a game publisher to back up financially that will affect the deliveries and development process, *Project Management* – project management in game development can be extra hard due to very tight schedules and involvement of many professions, *Team Organization* - involves building teams that enhances communication across disciplines, *Development Process* – the development process includes more than just software development, and *Third-Party Technology* – for many game developers third-party software represents the core of the project due to rising development costs and increasing complexity. In this article we will mainly focus on project management, team organization, development process, and third-party technology.

Only one systematic literature review concerning game development was found [5]. This literature review assessed the state of the art on research concerning software engineering for video games. The result of this literature review showed that the main emphasis in this research domain is on requirement engineering, as well as coding tools and techniques. Articles related to requirement engineering focuses on the problem of going from a game concept that should be fun and engaging to functional requirements, and software architectures and software designs that can produce game software realizing the game concept [7]. The initial requirements for a game can be labeled emotional requirements that contain the game designer's intent and the means which the game designer expects the production team to induce that emotional state in the player [4]. Research on coding tools and techniques include articles on development of game engines [8-10], component-based game development [11], the use of game engines [12], development of serious games [1], and challenges and solutions for networked multiplayer games [13-17]. There are also several articles that focus on software architectures for games [18-20], and design patterns for games [21-23]. These articles propose various software architectures and/or design patterns to solve problems in game development. However, unlike our article, they say very little about the processes in which the architectures and patterns are used, and how the various roles in game development affect them.

There are also articles that focus on the game development process and the involved roles. In [24], Scacchi presents how the free and open source development practices in the game community differs from traditional software engineering practices. In [25], a survey of problems in game development is presented based on an analysis of postmortems written by various game developers. According to Flood, all game development postmortems say the same things: the project was delivered behind schedule; it contained many defects; the functionalities were not the ones that had originally been projected; and it took a lot of pressure and an immense number of development hours to complete the project [26]. Petrillo et al. further details the specific problems found in game development postmortems to be unrealistic scope, feature creep, cutting features during development, problems in the design, delayed schedules, technological problems, crunch time of the developers, lack of documentation, communication problems between teams, lack of effective tools, insufficient testing, difficulties in team building, great number of defects found in the development phase, loss of key personnel during development, and over budget [25]. The problems clearly differentiate game development from conventional software development are unrealistic scope, feature creep, lack of documentation, and crunch time. You can also find these problems in traditional software development, but not to such a great extent.

Another study by Petrillo and Pimenta investigates if (and how) principles and practices from Agile Methods have been adopted in game development by analyzing postmortems of game development projects [27]. The conclusion of this study was that game developers are adopting a set of agile practices, although informally. One aspect of agile methods that is very relevant to our research is the emphasis on frequently gathering relevant stakeholders to bridge the gap between of all involved in the project [28]. This is related to our study where we investigated how the creative team, the technical team and the management collaborate and coordinate.

2.2 Research Goal, Questions and Method

The research method used in case study is based on the Goal, Question, Metrics (GQM) approach where we first define a research goal (conceptual level), then define a set of research questions (operational level), and finally describe a set of metrics to answer the defined research questions (quantitative level) [29]. The metrics used in our study is a mixture of qualitative and quantitative data [30].

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

The purpose of this study was to *examine how software architecture is used and how creative processed are managed* from the point of view *of a game developer* in the context *of video game development.*

The following research questions were defined by decomposing the research goal:

- **RQ1:** What role does software architecture play in game development?
- **RQ2:** How do game developers manage changes to the software architecture?
- **RQ3:** How are creative processes managed and supported in game development?
- **RQ4:** How has game development evolved the last couple of years?

To find answers to the research questions, we used a combined approach that included a questionnaire and a literature study to back up the findings. The questionnaire consisted of 20 statements where the respondents should state how they agree using the Likert's scale [31]. In addition, we added an opportunity for the respondents to give a free text comment on every statement. The statements in the questionnaire were constructed on the basis of the research goal and the research questions presented above, and was a result from a preliminary study of the role of software architecture and creative processes in game development. The subjects of the study were recruited from the Nordic booth at Game Developer Conference (GDC 2012) as well as direct emails sent to game developers. The questionnaire and results are described in detail in [32].

3 Results

This section presents the quantitative results from the questionnaire, comments from the respondents, as well reflections from a research literature.

3.1 Design of Software Architecture (RQ1)

Table 1 shows the response to statements related to design of software architecture in game development.

The majority of the companies in our survey agreed that software architecture is an important part of the game development process (Q1), backed up with the following comment: "Oversight in the game software architecture may lead to serious dead ends, leading to a need to rewrite the entire system". One reason the software architecture is so important, is to properly manage changing requirements as well as the complexity of game engines, libraries and APIs during the project [11]. Another reason is the importance of quality attributes such as performance (frame rates), portability, testability, and modifiability, which are very hard to change after release [18]. Further, the increasing focus on network games demand more focus on security (to avoid cheating) and availability for game servers [33]. Careful design and evaluation of a software architecture is the main approach to achieve predictable and acceptable quality attributes in software development [34].

Table 1. Statements related to the design of software architecture in game development

ID	Statement	Agree	Neutral	Disagree	N/A
Q1	Design of software architecture is an important part of our game development process	69%	15%	8%	8%
Q2	The main goal of our software architecture is performance	54%	15%	23%	8%
Q3	Our game concept heavily influences the software architecture	69%	8%	15%	8%
Q4	The creative team is included in the design of the software architecture	69%	15%	8%	8%
Q5	Our existing software suite provides features aimed at helping the creative team do their job	92%	8%	0%	0%
Q6	Our existing software architecture dictates the future game concepts we can develop	15%	47%	38%	0%

To the statement whether the main goal of their software architecture was performance (Q2), over half of companies agreed. However, the comments to this statement diversify the picture somewhat: "Performance plus functionality"; "Also future change, ability to be data-driven, optimized deployment processes, ease of automation, and testability"; and "Main goals are: Performance and Memory consumption."

Almost 3 out of 4 of the game developers agreed that the game concept heavily influences the software architecture (Q3). This result was a bit surprising, as game engines should ideally make the software architecture less dependent on game concept. One respondent provided the following comment: "Entirely depends on the game concept requirements, but in general: more generic – within boundaries – the better." This highlights that the importance of separating generic modules (core) with specific game play modules. Such an approach will allow reuse of core components, and at the same time provide sufficient freedom in development of game concept. "

How much the game concept will influence the game software architecture is really a question about where the boundary between the game and the game engine. Currently, game engines are targeting one or few game genres, such as real-time strategy games (RTS) or first-person shooters (FPSs). As there is yet no taxonomy that can be used to specify all types of games, there exist few game engines that are independent of genres [18]. Plummer tries to overcome this problem by proposing a flexible and expandable architecture for video games not specific to a genre [20]. However, too general game engines will most likely provide overhead in code and thus result in poor performance and memory usage. Thus, games stretching game genres will result in software architectures that deviate from the architecture of the game engine [18].

The large majority of the respondents agreed that the creative team was included in the design of the software architecture (Q4). One way this can happen is when the same person both work with code and game design as illustrated by this comment: “Only because I am a programmer and also the lead designer. Other creative people don’t know enough to be productively included.” There are several ways the creative team can contribute to the software architecture, such as making decisions of what game to make, request for new in-game functionality, and request for new development features in tools. Another comment related to this statement was: “This is mostly true when working on the tools the creative team will be using. It rarely applies to in-game specific features.“ Experiences from postmortems of game development projects show the importance of making the technical and creative team overlap going from game concept into developing the actual game software [25]. An overlap in roles in technical and creative teams is recommended to bridge the code/art divide that many game development projects suffer from [45].

To the statement Q5, the response concludes that the game engine and the supporting tools provide features that help the creative team. This is illustrated through the following comments: “Our third-party tools do not do this, but we’ve developed in-house extensions that do”; and “Use two software tiers that aims at very different levels of artist integration: Visual Studio and Unity3D”. The latter comment describes the situation that the creative teams not always can work with high-level GUI editors and high-level scripting, but sometimes must dive into the source code to get the game where the creative team wants it to go.

Game development is all about creativity and coming up with new game concepts. The response from statement Q6 shows that the software architecture does not to a large degree dictate future game concepts (15%). One comment that illustrates this point was: “We have engines that gives us a great benefit when building new games and we would prefer to continue on the same engines. However, it doesn’t fully dictate the games we will make in the future. This is primarily market-driven.” Other comments that clarify this point were: “It may influence, but not dictate whenever possible”; and “It makes it a bit more expensive to go to certain genres, but that’s it.” These comments indicate that the influence exerted by the existing software architecture is a direct result of a cost-benefit trade-off. The higher cost of change, the more influence the existing software architecture exert on the game concepts.

3.2 Changes to the Software Architecture during Development (RQ2)

Table 2 shows the responses to the statements Q7 to Q11 that focus on how game developers cope with changes to the software architecture.

We could not draw any conclusion regarding whether the creative team must restrict their ideas because of an existing game engine or not (Q7). The comments explain the diversity of the responses: “Technical realities are always something the creative side has to work around"; "Depending on structure. For assets handling, yes, but creatively, not so much. In latter case, the challenge is put to programmers to extend usage"; "Most of the time, the creative team is not fully aware of the game engine limitations so it is not their job to make it work by locking the creativity to things known to have been done with the engine before, the people who implements just need to make the ideas work one way or another"; and "That is not the way we do it here. The game design comes first, then we build what is necessary to make it happen." These comments indicate a trade-off between creative freedom and the technical limitations. Either the ideas must be adapted to the technology, or the technology to the ideas.

Table 2. Responses on how game developers cope with change to the software architecture

ID	Statement	Agree	Neutral	Disagree	N/A
Q7	The creative team has to adopt their ideas to the existing game engine	31%	46%	23%	0%
Q8	During development, the creative team can demand changes to the software architecture	69%	31%	0%	0%
Q9	The technical team implements all features requested by the creative team	69%	15%	8%	8%
Q10	It is easy to add new gameplay elements after the core of our game engine has been completed	70%	15%	0%	15%
Q11	During development, the creative team has to use the tools and features already available	47%	15%	38%	0%

The majority of the respondents agreed and none disagreed that the creative team can demand changes to the software architecture (Q8). There were two comments to this statement: "Depends how far in development and the size of a change, the odds of re-factoring an entire system late in production are close to nil, but the development team keeps an open mind at all times"; and "But again, only because the head of the creative team is president of the company and also wrote the original version of the game engine. If someone who doesn't know how to program came demanded changes to the software architecture, I would probably not listen very seriously." Game developers are inclined to prioritize the wants and needs of the creative team, given that the cost-benefit trade-off is favorable. Another important issue is what phase the project is in. The later in the project (production), the less changes are possible from the creative team. Boehm and Basili estimate that requirements error can cost up to 100 times more after delivery if caught at the start of the project [35]. A possible solution to this problem is to spend more time in the preproduction phases before moving to production, as it would leave relatively few surprises in the production phase [36].

The majority of the game companies agreed that the technical team implements all features requested by the creative team (Q9). There were several comments to this statement that provided more details: "It can happen that the creative team contributes on technical aspects during prototyping phase. Production quality code is left to the technical people"; "Things just aren't segmented this way in our situation"; "Of course, if the requests are decided to be implemented in the first place"; "It's very much a dialogue, we try not to have too formal split between tech and creative team when thinking about this, but prioritize what the user experience should be and when we can ship at target quality"; and "Some requested features are not tech. feasible."

Also the majority of the respondents agreed that was easy to add new gameplay elements after the core game engine has been completed (Q10). However, the comments suggest that adding new gameplay elements after completing the core game engine is often not possible, recommended or wanted: "It is simple during prototyping phase, technology-wise. However from a game concept point of view, it is highly disrecommended and the fact it is simple does not motivate the team to stack up features because the existing one are just not convincing enough :)"; "This really depends a lot, and can only be answered on a case to case effect"; and "Depends on the type of element – some may require significant underlying engine changes". One of the most common motivations for designing a software architecture is to provide a system that is easier to modify and maintain. In game development, modifiability must be balanced with performance. There are mainly two contrasting approaches to design modifiable game environments [38]: *Scripting* that requires developers to anticipate, hand-craft and script specific game events; and *Emergence* that involves defining game objects that interact according to rules to give rise to emergent gameplay. The most common approach is to create or acquire a game engine that provides a scripting language to create a game with predefines behavior. The emergence approach involves creation of a simulation of a virtual world with objects that reacts to their surroundings. The use of scripting makes it complex to add new gameplay elements, as everything is hardwired. The emergence approach makes it much easier to add new gameplay elements later in the project, with the price of being harder to test.

It was not possible to draw a conclusion on whether the creative team has to use the tools and features already available during development or not (Q11). The comments elaborated this issue: "The ones already available and the ones they request along the way"; "New tools can be made. However, it is certainly best to keep within the suite offered"; and "Our current engine (Unity) is easily extensible". This statement is really about cost. Adding new tools and features during development is costly and might also add risk to the project. However, in some cases new tools and features must be added to get the wanted results.

The response to the question about who decides if change-requests from the creative team are implemented is shown in the table below (Q12):

Technical team	Management	Creative team
10%	40%	50%

The responses to this question were mainly divided between management and the creative team, detailed through these comments: "Ultimately, the management can overrule everybody, but I would like to check the 3 options here, the creative team judges how important the change is, the technical team decides if it is realistic and the management makes sure it can be afforded. So mostly, it is a team decision"; "Actually it is all of the above, but the question would not let me put that as an answer"; "Sort of. The technical team advices what is possible, and as such has the final word. If it is possible, the decision falls on management, as it is usually related to economic costs"; and "Depends very much on the scale of change, we try as much as possible to keep this within and as a dialogue between the tech/creative teams, but if it means a major change it goes to management. We also aim to be as much product/feature driven as possible, as the primary owner is in the creative team." The responses from the developers indicate that all three branches (administration, technical and creative) are involved in the decisions about change. More game developers have also started to adopt agile development practices, where it is more common to have frequent planning and decision meetings involving all relevant stakeholders [37].

3.3 Supporting the Creative Processes

The results that relates to how creative processes are supported through technology and processes are shown in Table 3.

Table 3. Responses to how creative processes are supported

ID	Statement	Agree	Neutral	Disagree	N/A
Q13	Our game engine supports dynamic loading of new content	92%	8%	0%	0%
Q14	Our game engine has a scripting system the creative team can use to try out and implement new ideas	70%	15%	15%	0%
Q15	The creative team is included in our development feed-back loop (e.g., scrum meetings)	86%	8%	0%	8%
Q16	Our game engine allows rapid prototyping of new levels, scenarios, and NPC's/behavior	86%	8%	0%	8%

The response from the game developers shows that the game engines they use allow dynamic loading of new content (Q13). However, the comments show that there are some restrictions in when and how it can be done: "At some extent, in editor mode yes, at run-time only a subset of it"; and "With some constraints, but the content must be properly prepped of course." Different game engines provide different flexibility regarding changes that can be carried out in run-time. Most game engines support changes to the graphic as long as the affected graphical structures are the similar. Similarly, many game engines allow run-time changes using a scripting language that can change the behavior of the game. However, substantial changes to game play and changes of the game engine itself usually cannot be changed in run-time.

Most of the respondents say they have a scripting system that can be used by the creative team (Q14). However, there are also game developers in the survey that uses with their own game engines without scripting capabilities. Especially for small game developers, it can be too expensive, too much work or lack the competence to create support for scripting in their own game engine. The comments related to this statement were: "Yes, but could be better and more flexible (as always...)"; and "Our "scripting system" is typing in C++ code and recompiling the game." A recognized problem of letting the creative team script the game engine is that they usually do not understand the underlying low-level mechanisms related to performance [39]. Until the game engines can optimize the scripts automatically, the technical team often must assist the creative team with scripting.

The majority of the game developers in this survey include the creative team in the development feedback loop (Q15). This was not only true for smaller game developers, as a large one (500+ employees) also said that the creative team was included in development feedback loops. This is in alignment with what has been found in other studies [24, 27, 37]. The only comment related to this statement was: "Depends on the phase of the project".

The majority of respondents agreed that their game engine allows rapid prototyping of new levels, scenarios, and NPC's/behavior (Q16). Game engines supporting scripting normally provide rapid prototyping. There was only one comment related to this statement: "While most of the systems are designed with simplicity and fast iteration time in mind, certain things still requires time consuming tweaking tasks".

3.4 Changes Over Time

Table 4 shows the results on how game development has changed the recent years.

Table 4. Responses to how game developer has changed the last couple of years

ID	Statement	Agree	Neutral	Disagree	N/A
Q17	Today our company uses more 3rd-party modules than 3 years ago	46%	15%	8%	31%
Q18	It is easier to develop games today than it was 5 years ago	77%	8%	15%	0%
Q19	Middleware is more important to our company today than 3 years ago	55%	15%	15%	15%
Q20	Game development is more like ordinary software development today than 5 years ago	38%	24%	38%	0%

Close to half of the respondents agree that they use more third-party modules than 3 years ago (Q17). This confirms the predictions that buying a good middleware will provide a better result than what an organization can produce at the same prize [40]. The only comment to this statement was "It is about time …". On the importance of middleware to game companies today, the majority of respondents agreed that the importance has increased over the last 3 years (Q19).

Further that the vast majority in the survey agrees that it is easier to develop games today than it was 5 years ago (Q18). The complexity of games and the players' expectations have increased over the years [2], but the tools and the engines have also made it easier to manage complexity as well as achieving higher fidelity. The comments from the respondents highlight that the technical part has probably become easier, but the overall challenge of game development probably not: "The challenges have changed and the quality bar has risen, it is more accessible to people less interested in nerdy things nowadays (engines like Unity reduced the low-level aspect of the development), but developing a great game is still as challenging as before, the problems to solve just have evolved"; and "Technically and graphically, yes. Conceptually, no."

The feedback on the statement regarding whether game development now is more like ordinary software development than 5 years ago was mainly divided into two camps (Q20). The only comments to this statement came from those disagreeing: "Game development requires a more eccentric creative problem solving than development in most of other industries and this will probably remain true forever ;)"; "Nope. It was software development then, and still is now"; and "I think the tools available today moves game development further away from 'ordinary software development'.)". Several differences between game development and conventional software development have been identified in the literature. One example is that games usually have more limited lifecycle than conventional software products and that the maintenance of games mainly only focuses on bug fixing without charging the end-user [41]. Another example is that game development does not include functional requirements from the end-users. Typical end-user requirements to a game is that the game must be fun and engaging [7]. The latter poses a challenge of going from preproduction phase that produces a game design document (and maybe a prototype), to the production phase where all the software, game design, art, audio and music will be produced [7]. From a software engineering point of view, a challenge in game development is to create functional requirements from a game design document that describes the game concept. Another difference between conventional software systems and games is the importance of usability. A software system might be used if it provides much needed functionality even if the usability is not the best. However, a game with low usability is very unlikely to survive [42]. Usability tests and frameworks are also used within game development, but they are tailored specifically for the game domain [43, 44].

4 Conclusion

This article presents the results from a survey and research literature on how game developers use and manage software architecture and creative development processes.

The *first* research question (RQ1) asked about the role software architecture plays in game development. The response was that software architecture is important in game development, and it is important for managing the complexity of game software as well as achieving the quality in performance, availability, security and modifiability. We also found that the game concept heavily influences the software architecture mainly because it dictates the choice of game engine. Further, that the creative team can affect the software architecture through the creation of a game concept, by adding in-game functionality, and by adding new development tools. Finally, existing software architecture may or may not dictate future game concepts depending on a cost/benefit analysis (reuse of the software architecture if possible).

The *second* research question (RQ2) asked how game developers manage changes to the software architecture. The survey response was that the creative team has to some degree adjust their game play ideas to existing software architecture based on a cost/benefit analysis. The creative team can demand changes to the software architecture during development, but this decision depends on how far the project has progressed and the cost and benefit of making the change. Decisions on change-requests are usually made by involving personnel from technical team, creative team and management, but the management has the final word. Further, we found that the technical teams to a large extent implement all features and tools requested by the creative team (within reasonable limits), and that most developers said it was easy to add new game play elements after the core game engine was complete (although not recommended late in the project). The literature highlighted two approaches to deal with adding game play elements to a game: Scripting – where the behavior of the game is predeterministic and acting according to a script, and Emergence - where the behavior is non-deterministic and a virtual world is created by game objects that reacts the environment around them. The former has the advantage of being easier to test, and the latter has the advantage of being easier to extend game play.

The *third* research question asked about how the creative processes are managed and supported in game development. Almost all of the game developers in this study said they used game engines that support dynamic loading of new game elements (although not everything in run-time). The majority of the respondents use game engines that support scripting. Only game developers with own developed game engines did not support scripting. Finally, the majority of the developers said they used game engines that enabled rapid prototyping of new ideas. The conclusion of this research question is that current game engines enable creative processes through support of GUI tools, scripting, dynamic and loading of element.

The *fourth* research question asked how game development has evolved the last couple of years. This question can be summarized with the following: There has been an increased use of third-party software, middleware has become more important, and it has become technically easier to develop games. Although the majority of respondents said the technical aspects of game development have become easier, game development in itself has not become easier due to higher player expectations and higher game complexity. Similarly, there was no clear conclusion whether game development has become more like conventional software development. The main differences were identified to be that in game development there are no real functional requirements, the quality attributes performance and usability are more important, and game development has its own set of tools and engines.

References

1. Zyda, M.: From visual simulation to virtual reality to games. Computer 38(9), 25–32 (2005)
2. Blow, J.: Game Development: Harder Than You Think. Queue 1(10), 28–37 (2004)
3. Crooks, C.E.: Awesome 3D Game Development: No Programming Required, Cengage Learning (2004)
4. Callele, D., et al.: Emotional Requirements. IEEE Softw. 25(1), 43–45 (2008)

5. Ampatzoglou, A., Stamelos, I.: Software engineering research for computer games: A systematic review. Info. and Software Technology 52(9), 888–901 (2010)
6. Kanode, C.M., Haddad, H.M.: Software engineering challenges in game development. In: Sixth International Conference on Proc. Information Technology: New Generations, ITNG 2009, pp. 260–265. IEEE (2009)
7. Callele, D., et al.: Requirements engineering and the creative process in the video game industry. In: Proceedings of the 13th IEEE International Conference on Requirements Engineering, pp. 240-250. IEEE (2005)
8. Bishop, L., et al.: Designing a PC Game Engine. IEEE Comput. Graph. Appl. 18(1), 46–53 (1998)
9. Cheah, T.C., Ng, K.-W.: A practical implementation of a 3D game engine. In: Proc. Of the International Conference on Computer Graphics, Imaging and Vision: New Trends, pp. 351–358. IEEE (2005)
10. Darken, R., et al.: The Delta3D Open Source Game Engine. IEEE Comput. Graph. Appl. 25(3), 10–12 (2005)
11. Folmer, E.: Component Based Game Development–A Solution to Escalating Costs and Expanding Deadlines? Component-Based Software Engineering, pp. 66–73 (2007)
12. Antonio, C.A.M., et al.: Using a Game Engine for VR Simulations in Evacuation Planning. IEEE Comput. Graph. Appl. 28(3), 6–12 (2008)
13. Bouras, C., et al.: Networking Aspects for Gaming Systems (2008)
14. Smed, J., et al.: A review on networking and multiplayer computer games. Citeseer (2002)
15. Hampel, T., et al.: A peer-to-peer architecture for massive multiplayer online games. ACM (2006)
16. Triebel, T., et al.: Peer-to-peer infrastructures for games. ACM (2008)
17. Cai, W., et al.: A scalable architecture for supporting interactive games on the internet. In: Proceedings of the Sixteenth Workshop on Parallel and Distributed Simulation, pp. 60–67. IEEE Computer Society (2002)
18. Anderson, E.F., et al.: The case for research in game engine architecture. ACM (2008)
19. Caltagirone, S., et al.: Architecture for a massively multiplayer online role playing game engine. J. Comput. Small Coll. 18(2), 105–116 (2002)
20. Plummer, J.: A flexible and expandable architecture for computer games. Arizona State University (2004)
21. Gestwicki, P.V.: Computer games as motivation for design patterns. SIGCSE Bull. 39(1), 233–237 (2007)
22. Ampatzoglou, A., Chatzigeorgiou, A.: Evaluation of object-oriented design patterns in game development. Info. and Software Technology 49(5), 445–454 (2007)
23. Nguyen, D., Wong, S.B.: Design patterns for games. ACM (2002)
24. Scacchi, W.: Free and Open Source Development Practices in the Game Community. IEEE Softw. 21(1), 59–66 (2004)
25. Petrillo, F., et al.: What went wrong? A survey of problems in game development. Computer Entertainment (CIE) 7(1), 1–22 (2009)
26. Flood, K.: Game unified process. GameDev. net (2003)
27. Petrillo, F., Pimenta, M.: Is agility out there?: agile practices in game development. In: Proceedings of the 28th ACM International Conference on Design of Communication, pp. 9–15. ACM (2010)
28. Schwaber, K., Beedle, M.: Agilè Software Development with Scrum (2002)
29. Basili, V.R.: Software modeling and measurement: the Goal/Question/Metric paradigm. University of Maryland for Advanced Computer Studies (1992)
30. Wohlin, C., et al.: Experimentation in software engineering. Springer (2012)

31. Likert, R.: A technique for the measurement of attitudes. Archives of psychology (1932)
32. Nordmark, N.: Software Architecture and the Creative Process in Game Development, Master Thesis, Norwegian University of Science and Technology (2012)
33. Hsiao, T.-Y., Yuan, S.-M.: Practical Middleware for Massively Multiplayer Online Games. IEEE Internet Computing 9(5), 47–54 (2005)
34. Bass, L., et al.: Software Architecture in Practice, p. 624. Addision-Wesley (2012)
35. Boehm, B., Basili, V.R.: Software defect reduction top 10 list. Foundations of Empirical Software Engineering: The Legacy of Victor R. Basili, vol. 426 (2005)
36. Bethke, E.: Game Developer's Guide to Design and Production. Wordware Publ. Inc. (2002)
37. Stacey, P., Nandhakumar, J.: Opening up to agile games development. Communications of the ACM 51(12), 143–146 (2008)
38. Sweetser, P., Wiles, J.: Scripting versus emergence: issues for game developers and players in game environment design. International Journal of Intelligent Games and Simulations 4(1), 1–9 (2005)
39. White, W., et al.: Better scripts, better games. Communications of the ACM 52(3), 42–47 (2009)
40. Rollings, A., Morris, D.: Game Architecture and Design - A New Edition. New Riders Publishing (2004)
41. McShaffry, M.: Game coding complete. Cengage Learning (2013)
42. González Sánchez, J.L., Padilla Zea, N., Gutiérrez, F.L.: From usability to playability: Introduction to player-centred video game development process. In: Kurosu, M. (ed.) HCD 2009. LNCS, vol. 5619, pp. 65–74. Springer, Heidelberg (2009)
43. Desurvire, H., Wiberg, C.: Game usability heuristics (PLAY) for evaluating and designing better games: The next iteration. In: Ozok, A.A., Zaphiris, P. (eds.) OCSC 2009. LNCS, vol. 5621, pp. 557–566. Springer, Heidelberg (2009)
44. Laitinen, S.: Better games through usability evaluation and testing. Gamasutra (2005), `http://www.gamasutra.com/features/20050623/laitinen_01.shtml`
45. Hayes, J.: The code/art divide: How technical artists bridge the gap. Game Developer Magazine 14(7), 17 (2007)