Adoption of Requirements Engineering Methods in Game Development: A Literature and Postmortem Analysis

Miikka Lehtonen, Chien Lu, Timo Nummenmaa, and Jaakko Peltonen(✉)

Tampere University, Tampere, Finland
{miikka.lehtonen,chien.lu,timo.nummenmaa,jaakko.peltonen}@tuni.fi

Abstract. As the game industry continues to grow in size and revenue, the cost of creating games increases as well, and the successful outcome of game development projects becomes ever more important. In traditional software engineering, it is generally agreed that a successful requirements engineering process has a significant impact on the project. In game development, requirements engineering methods do not seem to be commonly used. As the development of digital games includes specialized aspects of software development, it seems likely that game developers could benefit from adopting these techniques and processes. In this paper, a thorough reading of central and current academic research on the topic is performed to form a holistic picture of the central issues and problems preventing the adoption and widespread use of requirements engineering processes and methods in game development. Additionally, algorithmic analysis of 340 post-mortems written by game developers and published on industry websites is conducted. These post-mortems discuss the factors which contributed to or hindered the successful outcome of these game development projects, and the analysis further supports the identified central issues.

Keywords: Requirements engineering · Game development · Postmortem analysis · Text mining · Literature analysis

1 Introduction

Requirements engineering is a process for handling hardware and software requirements that has been a part of software development for decades, and much has been written on its applications in various domains. One definition for requirements engineering by Hull et al. [7] is "the subset of systems engineering concerned with discovering, developing, tracing, analyzing, qualifying, communicating and managing requirements that define the system at successive levels of abstraction". Digital game development is no exception. As game development is a specialized form of software development, it logically follows

A. Brooks and E. I. Brooks (Eds.): ArtsIT 2019/DLI 2019, LNICST 328, pp. 436–457, 2020.
https://doi.org/10.1007/978-3-030-53294-9_32

that at least some portions of requirements engineering could be applied to the game development process. Several articles and papers have been written on this topic, presenting problems, limitations and concerns which need to be addressed if such an attempt were to be successful.

Academic research on the topic covers a wide spectrum from purely theoretical academic works to research focusing on the developers and their practices and concerns through questionnaires and interviews (e.g., [9,11,12,20]). This study aims to form a holistic picture of this current research, and to tie together knowledge from multiple sources to discover and present central problems and limitations. To verify the validity of these problems and limitations, 340 developer-written post-mortems were analyzed. Post-mortems are a common industry practice where developers reflect on completed software development projects and bring up problems, concerns and issues which affected the outcome of the project, either positively or negatively.

The research questions of this study are (1) Based on a reading of current academic research on the topic, can central problems, concerns and issues be identified? (2) Can these findings be supported by analysing developer-written post-mortems? The 340 post-mortems were analyzed algorithmically to determine whether keywords related to the discovered problems appear in them. In addition, a word correlation analysis was conducted to determine the contexts these keywords might be used in. Our analysis assesses if the problems related to these concepts and keywords are common in the industry, as the expectation was that if game developers are frequently encountering these issues and problems, they would also mention them as contributing factors in post-mortems.

2 Game Development from a Software Development Perspective

In the year 2018 the video games industry was bigger than ever. According to a report by Newzoo [15], there are over 2.3 billion video game players across the world. The games industry is expected to generate over 108 billion dollars in revenue, representing a growth of 7.8% from 2016 [15]. This growth industry contains innumerable game development studios ranging from lone developers to small companies and large multinational corporations. According to the Entertainment Software Association's 2017 report on the American video game industry [2], in 2016 there were over 2450 active game companies in the United States alone. Among those, 99.7% of them are qualified as small businesses, meaning they have under 250 employees and less than $7.5 million in annual revenue. Similar numbers have been reported elsewhere in the world, as according to the UK Interactive Entertainment Association (UKIE), there were 2261 active game companies in the United Kingdom as of June 2018 [25].

These thousands of game developers are working on varied games ranging from huge titles with budgets in the hundreds of millions to eSports titles, mobile games, small independent projects and everything in between. Game development presents a unique challenge from a software development perspective. Contrary to the more disciplined and theory driven world of traditional software

development, games development is a more fractured landscape. Whereas large corporations such as Electronic Arts or Activision, or even larger independent developers, might adhere to traditional software development roles and practices – agile methodologies and Scrum being particularly popular in game development – for smaller independent studios development is probably less regimented and more free form [13].

Games as a form of software development also have several other unique characteristics. As an example, whereas traditional software engineering teams consist of software developers of various disciplines, a game development project will usually have the normal complement of software developers, but additionally artists, writers and other purely creative people. These disciplines do not often share a vocabulary and might differ widely in their needs, methods and work flows. Yet all these disciplines need to find common ground if the project is to succeed. Additionally, these highly multi-disciplinary teams seek to create software which philosophically differs greatly from traditional software. Traditional software development projects aim to create solutions to discrete problems, whereas games are mass-marketed products aimed to entertain and prompt emotional responses [9].

From this it follows that the models and theories which drive traditional software development projects might not be directly and fully applicable to game development. This is also true for requirements engineering.

3 Requirements Engineering in Game Development Literature

Requirements engineering is a collection of different phases, processes and methodologies, which seek to take in information from a variety of sources and transform it into concrete requirements; singular and unambiguous physical or functional needs that the product or service must be able to meet. Together, these requirements form the specification of the project, essentially the blueprint the engineers can design according to and refer to when there is ambiguity [6]. While requirements engineering in traditional software development is a heavily covered field with academic publications, books, magazines and even conferences dedicated to the subject, this is not the case for requirements engineering as part of the game development process. We performed a literature survey to map the current state of academic writing on requirements engineering and game development. Our approach uses elements of the systematic literature review process used in previous studies [1,16]. However, the search was executed specifically in order for the results to be used as background information for our postmortem analysis, and to be contrasted with the data-driven analysis of postmortems.

3.1 Survey Process

Academic search engines such as Google Scholar and the library search engine of the University of Tampere were utilized. The latter allowed access to various

digital libraries, which further widened the field of possible results. The goal of these searches was to discover peer reviewed articles, academic publications, conference proceedings and published books which dealt with games development and requirements engineering. No specific time constraints were placed on the results. Software development is a fast moving field, which means that some of the older findings might be outdated. However, they could also reveal newer research which builds upon their findings or expands it. Initial searches were performed using the search term "game development" together with the terms "requirement", "requirements engineering", "formal specification", "methods" and "processes", both in singular and plural forms, and with wildcards.

Together, these searches produced a pool of over 16 000 results. As is to be expected with such broad search terms, most of these results were either marginally related to the actual research question, or not at all related. Even after discarding most of the less applicable results, the pool still contained several hundred articles which might be tangentially related to the research questions. At this point any articles with titles or abstracts which seemed promising were stored in a separate list to be more carefully examined later.

As this list of promising articles was read and processed, references to new papers which seemed relevant to the topic were noted down and added to the list, as were author names who had written relevant works. They were then read, and the process was repeated. In this recursive process new papers that were much more relevant to the topic than those found in the initial search process were discovered, and most of the actual sources used in the literature survey came from this phase of the process.

3.2 Overall Findings From the Literature Survey

Based on this study, the state of research on the topic proved to be rather healthy, if not comprehensive. There is certainly a larger volume of research than anticipated, and requirements engineering proved to be a central topic: as of 2014, 39% of papers submitted on the topic dealt with requirements engineering in some way [3]. This does not mean there are no gaps to be found in current research.

Whereas traditional software development and its issues are a topic of some 40+ years of discussion, the same is not true for games. It is generally accepted that there are similarities and unique factors between the two areas. In recent years more studies have been conducted as to what actual problems game developers are facing, which is a crucial area of research [9,11,12,20]. However, concrete solutions and suggestions are still few and far between: one notable problem area is that many articles bring up problems in processes and methods, but rarely offer any concrete suggestions beyond vague calls to adapt best practices from the world of traditional software engineering. Due to the central differences between traditional software development and games development, this adaptation would have to be handled with care and consideration, so academic research on the topic would be beneficial.

We found that a central type of data used for the research was developer interviews and surveys, however, as an alternative data source Petrillo and Pimenta [19] explored post-mortems published on Gamasutra, an industry-focused website by developers for developers. Developer communication in post-mortems can be a useful complementary information source and we will use it ourselves later in this paper.

Studies, such as those conducted by Kasurinen et al. [9,11,12] have been conducted on industry practices among various groups of developers. They highlight issues developers grapple with, as well as the methods and practices used to deal with these issues. Kasurinen's large scale surveys and interviews among Finnish industry professionals found that there are several similarities, but also meaningful differences which mean that traditional software development methods and lines of thinking will not apply directly.

We identified four key problems, which we introduce below and will discuss in more detail in separate subsections. The first of these deals with the incompability of the game design document and the requirements document. Kanode and Haddad [10] talk about an important topic, the game design document. In game development, the game design document is a repository of information about the game. It details the setting, plot, gameplay, characters and themes of the game. It is not a formal document, and as such is poorly suited for actual software development. However, the findings of Kanode and Haddad seem contrary to evidence presented in other papers: they suggest that a game designer needs to capture all the requirements from a game design document before the actual production work on the game can begin. Callele et al. [4] counter this by stating that translating this informal document into something resembling a requirements document is a massive and complicated process. Even a short, simple gameplay description in the game design document can generate dozens of pages of requirements, and even more problematically generating those requirements requires unrealistically strong and specialized domain knowledge in many fields. Callele et al. [4] have studied this issue through analyzing real-life game design documentation, discussions with actual game developers and observing actual development processes. They conclude that this transition from pre-production to production (taking informal and often very casual documentation, turning it into a formal document suitable for development, and then beginning to realize the vision outlined in that document) is one of the biggest problems in game development and alone responsible for many project failures.

Another common theme in the discussion of requirements engineering and games is the unique nature of requirements in games. Traditional software development places a heavy emphasis on functional requirements (i.e. concrete features in a project), whereas in game development these are almost standardized among games of the same genre. Instead, the differences between games come largely from non-functional requirements, which play a heavier role. Of special interest are so called affective, or emotional requirements [4]. Games are intended to prompt emotional responses in their players, and these should also be modelled through requirements engineering. However, the tools and techniques to do so are still in their infancy.

As a third problem, Kasurinen et al. [9,11,12] point out that whereas change to the original specification is something needs to be very carefully managed in traditional software development, in game development changes through iteration are a desired outcome. As the developers try to "find the fun", i.e. create the combination of gameplay and features which makes the game fun, they must be prepared to make even drastic changes late in the project. We will discuss the issue of iteration, scope, and change management in more detail in a later subsection. Based on these findings, Kasurinen [11] concludes that while some common traditional software development methods such as Scrum can very easily work with game development, others are not so easily compatible and need special consideration.

As a fourth problem, the literature also suggests [13,14] that currently game developers do not make widespread use of formal, theory-based methods and processes. At the same time there is evidence to for the presence of problems traditionally thought to be fixed by these methods and processes, which suggests that developers could benefit from a less informal development process [20].

Based on this literature survey, several key concerns and problems were identified, which will be analyzed and discussed in more detail in the following section.

4 Key Differences Between Game Developers and Traditional Software Developers

Game development and traditional software development methods and tools, for instance requirements engineering, are not fundamentally incompatible. There is evidence that game developers make use of these methods, and get benefits from them [9].

That being said, there do seem to exist some fundamental differences and problems, which make adapting these traditional processes and methods to game development difficult. Game developers do seem to suffer from many problems which could be alleviated or eliminated through better requirements engineering processes and methods. For example, in post-mortems published on Gamasutra.com, game developers cite factors such as "inadequate planning", "underestimating the scope of tasks" and a schedule that was "too aggressive" [4] as aspects of the project which went wrong and hindered them.

It is worth noting that these findings are not universal. Game development is a wildly varied field, with studios ranging from one-person teams to massive international companies. Many developers, especially larger companies, tend to regard their methods and practices as trade secrets and are not open to discussing them with journalists or academics. Despite this, from merely reading recruitment posts and requirements for open positions, it is clear that at least larger companies do value degrees and formal training when seeking to hire developers.

It is also worth noting that these issues are heavily linked and could also be thought of as different aspects of the same problem. After all, any differentiation between "a lack of formal processes and methods" and "poor change control" is going to be somewhat arbitrary, as the latter could easily be considered a part of the former.

What, then, could be some of these key differences that need to be considered, and key problems that need to be overcome?

4.1 The Incompatibility of the Game Design Document and Requirements Engineering Documentation

In traditional software engineering projects which utilize requirements engineering methods and processes, a common guideline for the design work is the requirement documentation. It is essentially the blueprint against which the product and its features are compared for specifications and verification.

In game development, a similar role is played by the game design document [4]. While its contents and size vary from team to team and project to project, commonly it includes descriptions for plot, characters and events as well as gameplay mechanics, puzzles and so on. Much like the requirement document, the game design document is often created during pre-production [4].

While these two documents share a similar role, they are not stylistically equal or even similar. A game design document is usually more free form and written in natural language [4]. Since it is the primary design document for game development, it has been proposed that the game design document would also be a major source for requirements [4,10]. This is logically sound, after all if the document contains descriptions of gameplay mechanics and elements, it stands to reason that requirements could be generated from these descriptions. In fact, some have gone as far as stating that all of the game design document should be captured as requirements before production should start [10].

Evidence has shown this to be an unrealistic expectation, however. Even a single paragraph length description of a game design element from the game design document could produce several pages of requirements. Even worse, many of these requirements are merely implied, and capturing them requires high level domain knowledge in game design, genre conventions, technical matters and many other fields [4]. A skilled and experienced game developer will be able to pick up on some of these cues and implications, depending on how well versed they are in the different disciplines of game development (e.g. programming, art and sound design, writing), their team's own culture, the capabilities, features and limitations of the game engine the team is using, and the genre of the game they are working on.

Expecting this kind of expertise from a single person is unrealistic, as is the expectation of being able to generate good requirements based on heavily implicational natural language. The latter half of the problem could possibly be alleviated by employing technical writers, who are skilled in writing precise and unambiguous language, but they would probably not have the required domain knowledge. The common feeling is that it is "easier to do it myself than to

explain it to someone else" [4] which may be true, but does not help eliminate the problem.

Even if suitable candidates could be found, or if the job of capturing the implied requirements were divided among a versatile group of skilled developers, the process would be extremely time consuming. Game development projects are usually executed under extremely tight, publisher-driven deadlines, and extending the pre-production phase to accommodate a lengthier requirements engineering process would probably not be welcomed [4]. For example, according to a study conducted in Finland, most Finnish game development projects last under 12 months, with many of them lasting less than 6 months [13].

It would therefore seem that there is a base level incompatibility between traditional requirements engineering documentation and the artefacts of game development.

4.2 Emphasis on Non-functional Requirements and Affective Requirements

In traditional software engineering, the emphasis is on functional requirements. They describe the key features of the system to be implemented, and are what ultimately distinguishes it from its competition and allows it to fulfill its stated and desired goals. In game development, non-functional requirements are considered much more important. In what is called "horizontal differentiation", it is claimed that the functional requirements for games of a particular genre of game are often quite similar to begin with, and non-functional requirements make the difference and help distinguish a game from its peers [4,18].

Additionally, unlike in traditional software engineering, more and more game developers are using pre-made game engines such as Unity[1], Unreal Engine[2] or CryEngine[3], which further removes emphasis from functional requirements, as these requirements are already fulfilled by the pre-packaged engine [9].

This in and of itself might not be a problem, as tools for capturing and modelling non-functional requirements have existed for decades. In game development, however, non-functional requirements deal with more difficult concepts. In traditional software engineering, requirements generally refer to concrete and measurable real-world conditions, whereas game-domain specific requirements are more abstract and harder, if not impossible, to measure [12]. Requirements related to concepts such as fun, storytelling, aesthetics and so on are key in video game projects, but of course not at all relevant in traditional software engineering [4]. These requirements also vary from genre to genre [18]. What is important in a racing game might not be at all relevant in a puzzle game, or an adventure game.

Unlike traditional software projects, games are intended to produce emotional responses in their users. Requirements relating to these emotions are referred to

[1] https://unity3d.com. Retrieved 27.11.2018.
[2] https://www.unrealengine.com. Retrieved 27.11.2018.
[3] https://www.cryengine.com. Retrieved 27.11.2018.

as emotional, or affective, requirements and they are viewed as a key component in creating an engaging gaming experience [4]. The tools and techniques for capturing and modelling these requirements either do not exist, or are not as developed as they should be. Additionally, validating these requirements is also extremely difficult, as they deal with highly subjective concepts. Traditional validation methods such as testing are not easy to implement or very reliable [4].

4.3 Iteration, Scope and Change Management

Change is an inevitable part of almost any software product. No matter how thorough the pre-production planning, how well executed the requirements engineering process and how accurate the model, something will eventually change. Change control and management are considered essential parts of the requirements engineering process, and significant work both during pre-production and production is carried out to ensure changes can be tracked and managed as efficiently as possible [5,17,22].

Despite this, change is not seen as an outright goal, and instead more of an unavoidable necessity. This is in contrast with game development, where change is often outright desired. Game development is a heavily iterative endeavor, as the developers try to find the magical formula of features and gameplay executed just right to make the game as fun as possible [12,24]. This will inevitably lead to many and in some cases quite drastic changes to the design and scope of the project.

Due to the emphasis on non-functional and affective requirements, change is also often the outcome of testing. A version of the game is given to testers, and based on their feedback changes can be made. Sometimes these changes can be quite drastic, and in many cases these iterations will carry on quite late in the actual development phase of the game and changes will occur very close to the end of the project. This is in part due to the fact that this user-driven testing is not only a tool for validation, but also defining the quality of the product [12,24].

With this in mind, it would stand to reason that game development could benefit from more robust change management procedures and methods. A common problem in game development is scope management. The game will be designed to have a certain set of features, and time and resources are budgeted to fulfill these design criteria in the available time.

During development features get added either due to outright planning, because testing suggested they might work well in the game, or sometimes even because individual developers felt they were “cool”. Suddenly there are no longer enough resources or time to finish the game as specified, and sometimes the revised and changed version of the game no longer works as well as originally planned. This process is referred to as “feature creep”, and according to some sources, it is one of the biggest problems in game development [20].

Feature creep is seen as a large problem not only because it creates scheduling problems, causes games to be delayed and costs money, but also because of its human cost. Game development is a massive industry, and publishers will often

not be willing to delay projects significantly. Instead what happens is, game developers will work longer and longer days as deadlines approach. From an International Game Developers Association (IGDA) report [8], there are stories of people literally living at work, sleeping under their desks for a few hours when they can. Burnouts and people quitting the games industry inevitably follow because of these heavy periods of crunch, as it is called.

However, at the same time, this iteration and change is both desired and necessary. Often developers will "find the fun" quite late in the development process, which means that if change and experimentation were to be avoided, these games might never have been finished, or at least not in their final conditions. This issue is compounded by game development being notoriously difficult for scheduling in general. Evidence suggests many possible factors as the reasons. One popular suggestion is the multidisciplinary nature of games development. Different types of developers (e.g. artists, coders, writers, designers) have different workflows and different types of "production pipelines", which can cause delays when some parts of the development team must wait for dependencies to be completed [20].

In traditional software development, several processes and methods exist for managing changes and maintaining scope and product integrity despite changes. Therefore, it seems that game development could benefit from more robust change and scope control and scheduling mechanisms. Unfortunately, it seems that right now these mechanisms either do not exist, or are not utilized frequently, in game development.

4.4 Lack of Formalized Methods and Processes

According to two studies conducted in Austria [14] and Finland [13], game developers do not make good or widespread use of typical methods and processes. In Finland, 61% of the respondents to the survey indicated that they did not use any systematic development methods. In Austria, 23% of respondents indicated they did not use any kind of formalized methods or processes. Even those who did self-report using theory-based methods and processes mostly used adapted and flexible processes which were said to be comparable to Scrum and XP (Extreme Programming) [11,14]. Further, according to the Finnish study, developers do not collect metrics or document their activities [11].

This kind of laissez-faire approach permeates all levels of development. For instance, developers prefer to not engage with traditional requirements engineering activities and instead prefer the approach of "test and tune" to replace it. This testing is largely user-driven, as feedback received from users is used to gauge quality and drive development. Despite this, the feedback is not commonly collected in any kind of formal or systematic fashion [9].

Some of this approach can be explained by base level incompatibilities in game development and traditional software engineering. Whereas traditional software engineering projects are launched to answer specific problems, game development can be iterative even at the ideation stage. It is common for developers to briefly explore tens of ideas initially, but only choose a few for detailed

implementation, at which point the project has already moved at least partially to production and any kinds of pre-production processes are incompatible [9]. There could also be other explanations. A lack of formal training and the tendency to promote from within could play a role. If a project manager does not have any training or knowledge about theory-based methods and processes, how could they hope to make use of them?

Despite this, there is evidence to support the claim that game development could benefit from adopting more formal, theory-based methods and processes based in established software development theories. According to research conducted in Finland, many developers do already utilize some aspects of project management processes, but do so informally and in an ad-hoc fashion [9]. This would seem to indicate that the need for these processes and their benefits exists within the developer community. The problems formal processes and methods are intended to fix are observable within the game development community: difficulty transitioning from pre-production to production, difficulty in capturing requirements, difficulty in change and scope management and so on [20].

According to research, game development falls into two broad and informal categories. Larger, more traditional developers still make use of more linear processes which bear a strong resemblance to the traditional waterfall model, whereas increasingly especially smaller developers are making use of agile and flexible methods. These agile methods are often self-created to some degree and might mostly draw inspiration from more formal schools of thought such as Scrum, Kanban and XP [9,14].

Both styles of development could benefit from requirements engineering processes. For the more traditional project style, structured requirements engineering processes could be utilized in largely the same way as in traditional software engineering projects, hopefully with similar results. Even the more informal projects, which are driven by iteration and user feedback, could benefit from structured processes and methods to capture and document this feedback and the requirements it generates [9].

5 Post-mortem Analysis

Based on the literature analysis conducted, certain key problem areas and problems could be identified. As some of these academic writings leaned on industry-focused studies and were based on the thoughts and opinions of game developers, it can be assumed that these problems do in fact exist in game development at least to some degree. It was felt, however, that it would be beneficial to get more context for these findings. How common are these problems in actual game development?

Developer-written post-mortems on websites such as Gamasutra.com[4] and Gamecareerguide.com[5] offer insight to industry professionals' opinions and thoughts on game development. It was felt that they could provide a revealing and adequate source for data on the issue, and an alternative to conducting large scale interviews. For examples, Petrillo et al. [21] tried to understand problems in the development process of electronic games through analyzing 20 post-mortems. Petrillo & Pimenta [19] further analyzed the same 20 postmortems to investigate the adoption of agile methods in game development. Washburn et al. [26] have analyzed 155 public postmortems qualitatively to outline the characteristics of game development. Post-mortems are a common industry practice, where a developer who served a central role in the project is invited to reflect on their project. According to Gamasutra.org's instructions [23], each post-mortem should include a few aspects that went right in the project, as well as a few aspects that went wrong. These should be unique to the project, and should offer concrete thoughts other developers can learn from.

Due to their nature, these post-mortems were assumed to provide a valuable and reliable insight to the pros and cons of a wide variety of game development projects. They were therefore fetched and analyzed algorithmically using simple data mining and natural language processing scripts. The purpose of this analysis was to see if key topics and words related to what were perceived as central problems in the field, were present in these post-mortems.

The tests were conducted to test two assumptions.

1. If, for instance, requirements engineering methods and practices are not widely used in game development, keywords related to the topic would not appear frequently (or at all) in post-mortems.
2. If game development could benefit from requirements engineering methods and practices, common problems believed to be alleviated using these methods would appear at least relatively frequently.

This approach does have some limitations. As each developer is instructed to only include a few problems in each article, post-mortems are not exhaustive. Problems may not have been brought up among the few listed in a post-mortem despite influencing the actual development process. An interview or even a survey would give more focused information on the topics of this paper, but this is not necessarily a weakness. As these post-mortems are not guided or directed by research questions or prompts, they do offer a view into what the developers themselves viewed as central and significant factors in the success or failure of their games.

Additionally, it is worth noting that correlation does not necessarily equal causation. Even if both assumptions turned out to be true, it does not automatically mean that all these problems are caused by the lack of requirements engineering methods and processes, nor that would they be fixed merely by

[4] https://www.gamasutra.com/features/post-mortem/. Retrieved 27.11.2018.
[5] https://www.gamecareerguide.com/archives/postmortems/1/index.php. Retrieved 27.11.2018.

adopting these methods and processes. A much more exhaustive study would be required for conclusive results, but that does not detract from the value of this study.

5.1 Data Gathering

Post-mortem articles are collected with a self-made crawler from Gamasutra.com and GamecareerGuide.com. By March 2018, we had gathered 218 and 129 post-mortem articles respectively from the above-mentioned websites. The 347 post-mortems retrieved in total from the two websites range from 1997 to 2018, and cover everything from small independent teams to large studios, and everything from small browser games to large, big budget productions. Games from a variety of different genres are included. Not all the post-mortems are suitable for this study, as some of them are small "post cards" from industry events. After eliminating these obviously non-related articles, there were 340 post-mortems left for analysis. These post-mortems cover roughly 300 unique games, as a few projects were discussed from different perspectives, such as general design and audio design.

5.2 Initial Analysis

Based on the central problems in game development presented in Sect. 3.2, a list of keywords was created. These keywords were thought to be related to these central problems based on existing domain knowledge on the topic. There was no specific methodology for creating this initial list of keywords, and instead it was always intended as a simple jumping off point which would hopefully generate interesting and promising articles, based on which additional keywords could be discovered.

- **Project management:** crunch, schedule, management, overtime, estimation, feature creep, creep, feature, scope, communicationm, multi-disciplinary.
- **Methods and processes:** agile, process, method, Scrum, Kanban, engineering, development, transition, extreme programming, backlog, formal.
- **Requirements engineering:** requirement, emotional, affective, game design, document, pre-production, production, requirement engineering, requirements engineering, specification.

The initial intent was to narrow down the list of 340 post-mortems to find which post-mortems should be studied more closely, and which could be discarded, as analysing all the post-mortems would not have been practical and quite probably also not useful. Therefore, the intent was to prioritize the post-mortems based on how many of these keywords appeared in them. This analysis was conducted using a self-built programming script, which iterated through all 340 post-mortems. The script searched for instances of keywords, noting down the articles in which they appeared, and the results were exported into a file for analysis.

As the scripts were being refined, the study evolved beyond simply trying to narrow down the list of post-mortems. It became apparent that getting statistical information about how often given keywords appeared in articles would be easy, and the focus was shifted towards this approach.

This approach has some limitations. The first of these is the list of keywords used. If some relevant or useful term was not thought of, it would not be included on the list of search terms. As this part of the study was conducted by a single researcher, albeit with some supervision, it is quite probable that something was overlooked. This problem was probably compensated at least in part for by repeated versions the keyword list and repeated analysis of the subject text. The list of keywords grew significantly over time as additional terms were discovered through further readings of the source texts, or derived from results of earlier iterations of the analysis.

Additionally, this approach offers next to no context. While the algorithm will find all instances of a keyword such as "scope", it has no way of knowing the context the term was used in. Did the article refer to the scope of the project, or was the author talking about a physical scope item in the game? Many of the terms used have multiple meanings, only some of which are relevant to this paper, so this could have been a real problem. To compensate for this lack of context, a second test was devised and run.

5.3 Extended Analysis

Our aim in this extended analysis was to 1) gather occurrences of keywords in a more relaxed fashion allowing multiple word forms of each keyword to be detected, and 2) find context of the keywords by statistical analysis to detect other keywords that tend to often appear together with them.

To detect keywords in a permissive manner, a natural language processing script was written. The script first breaks the input text into smaller, sentence length chunks. Next, the text was lemmatized (i.e. the inflected forms of each word were grouped together in their dictionary form), and so called "stop words", or common, short function words such as the, is, that and which, were removed. After these steps the remaining text was analyzed. Next, simple statistical analysis was done to detect co-occurring keywords. For this analysis, sentences which contained words from the keyword list were kept, while the others were discarded. The remaining sentences were analyzed for word correlation: correlation of occurrence of one keyword and occurrence of another keyword across the sentences. The analysis produced a list of found search terms as well as lists of words they appear together with. This would then give context to these results.

Due to the way the algorithm parses words, it will distinguish between multiple word keywords such as "feature creep" and individual components of the keyword, in this case "feature" and "creep". Thus, the algorithm will not produce skewed false hits for these component words.

As with the first test, this test was also run several times, first using co-occurrence analysis, and later with improved correlation analysis. The original

list of keywords grew and changed after each iteration as new keywords were discovered externally, prompting repetitions of the first study as well. Additionally, the results of this test also helped refine the list of keywords, as interesting or relvant terms are actually correlated to original keywords and were subsequently included as keywords themselves.

5.4 Findings

The two studies have produced : a full list of all 340 post-mortems, and the keywords which appear in them, the total count of how often any keyword appears in each post-mortem, a list of all the keywords and the most common words that appear near them, and statistical information about the total number of occurrences for each keyword across all 340 articles, as well as the percentage of articles each keyword appears in Table 1.

Table 1. Occurrences across all articles for a given keyword.

Keyword	Frequency	Pct.	Count	Keyword	Frequency	Pct.	Count
Development	321/340 art.	94.41%	3156	Emotional	39/340 art.	11.47%	149
Feature	278/340 art.	81.76%	1582	Formal	35/340 art.	10.29%	41
Process	270/340 art.	79.41%	1160	Feature-creep	33/340 art.	9.71%	43
Document	228/340 art.	67.06%	453	Scrum	33/340 art.	9.71%	84
Schedule	206/340 art.	60.59%	811	Agile	27/340 art.	7.94%	41
Production	205/340 art.	60.29%	961	Overtime	25/340 art.	7.35%	43
Communication	146/340 art.	42.94%	402	Specification	22/340 art.	6.47%	35
Management	143/340 art.	42.06%	332	Game-design-document	19/340 art.	5.59%	41
Scope	134/340 art.	39.41%	306	Creep	19/340 art.	5.59%	23
Method	108/340 art.	31.76%	205	Engineering	15/340 art.	4.41%	28
Requirement	94/340 art.	27.65%	179	Estimation	8/340 art.	2.35%	11
Engineer	92/340 art.	27.06%	308	Backlog	6/340 art.	1.76%	6
Crunch	88/340 art.	25.88%	198	Multi-disciplinary	3/340 art.	0.88%	3
Pre-production	56/340 art.	16.47%	166	Affective	1/340 art.	0.29%	3
Discipline	47/340 art.	13.82%	76	Kanban	0/340 art.	0.29%	0
Transition	45/340 art.	13.24%	58	Requirement engineering	0/340 art.	0.29%	0

It becomes apparent that some terms were too broad especially for the initial intent of the studies even from a cursory glance at the list of keywords. The words “development”, “feature” and “process” appear in almost all of the articles. However, due to the word co-occurrence analysis, it is apparent that they do not appear without context and were as such deemed interesting enough to be left in the pool of keywords.

The word correlation analysis produced a list of each keyword and the most correlated words they appear together with in the analyzed material. In order to avoid spurious correlations, we remove infrequent terms and verify remaining correlations with a student-t based significance test, the t-statistic is computed

as $\rho\sqrt{\frac{N-1}{1-\rho^2}}$ where ρ is the correlation coefficient and N is the sample size. In the final list, we keep the top 10 correlated words that co-occur (document-level) more than or equal to 5 times with the keyword. We also conducted the association test, the p-value is provided in the parentheses. Correlations for 4 terms are listed in Table 2. The term schedule is correlated to terms such as "tight", "slip" and "milestone". This indicates that the underestimation of the schedule is a common issue in game development. The term communication is correlated to different words that represent different perspectives such as frequency ("occurrence"), target ("team") and method ("verbal"). Words that are correlated to the term development are related to scheduling ("cycle" and "length") or appliance ("software"). However, many of the words that are correlated to the term requirement do not seem to be related with requirements engineering. The full list can be found online via the link https://bit.ly/2FeQ42U.

Table 2. Words correlated to search terms, shown for four example terms. ρ: correlation coefficient, n: number of co-occurrences.

Search term	Word	ρ	p-value	n	Other correlated words
Schedule	Tight	0.116	2.63e-175	60	Occasional, task, behind,
	Slip	0.095	1.22e-117	45	project, budget, aggressive,
	Milestone	0.069	1.73e-62	79	instructor
Communication	Occurrence	0.124	2.77e-198	7	Skype, lack, facilitate,
	Team	0.079	2.46e-81	143	inter, proximity, constant,
	Verbal	0.078	1.09e-79	5	apart
Development	Cycle	0.170	0.00e+00	139	Month, ram, date, photoshop,
	Length	0.130	5.12e-218	137	platform, hardware,
	Software	0.114	1.30e-168	186	process
Requirement	Mock	0.126	5.82e-206	6	Experimental, viable, nail,
	Fulfill	0.126	2.85e-204	9	skin, export, nature,
	Playback	0.110	3.95e-155	9	application

In general, terms thought to be related to the requirements engineering process and its methods appear either very rarely or not at all. "Requirements engineering" (and its alternative spelling "requirement engineering") do not appear once. The broader keyword "requirement" appears in 27.65% of the articles, but it is practically always used in the non-requirements engineering sense. "Affective" is used precisely once, and while the keyword "emotional" does appear in 11.47% of the articles, it is not used to talk about emotional requirements.

Terms related to agile methods and Scrum appear relatively frequently in more recent postmortems: "Agile" or "Scrum" are mentioned in 17.86% of postmortems from 2006 onwards. "Extreme programming" is mentioned once. Theory-based methods and specifications in general do not seem to be a frequent

topic in post-mortems, as the keyword "formal" is used in 8.82 % of the articles. Context analysis suggests that when the term is used, it is rarely used in the context of formal processes: it appears three times close to the term "process", and three times close to the term "development". "Specification" is used in 6.47% of the articles, and is usually used in the context of formal design methods.

These findings would seem to back up the arguments presented in current academic research, and suggest that formal methods and practices, requirements engineering and other accepted industry best practices are not widely used in game development.

The problems these methods and processes are thought to alleviate appear in the post-mortems quite frequently. The keyword "crunch" appears in 25.88% of the post-mortems, and when it appears it is often mentioned several times in the same post-mortem. Additionally, the term "overtime" appears in 7.35% of the post-mortems, usually in the context of having to work overtime. The algorithm does not guarantee that there is no crossover between these results, so both keywords could appear together in at least some of the post-mortems. Phrases such as "It was an expensive lesson, given the amount of overtime we had to work to finish the game", "building several levels, working a tremendous amount of overtime" and "others were totally fried from the tremendous amount of overtime" indicate that "overtime" usually appears in the intended sense rather than describing, for instance, a system working overtime.

"Feature creep" is used in 9.71% of the post-mortems, and additionally "creep" is used in 5.59% of the articles, often in a context which suggests it is used to describe feature creep rather than an action by a game character. Terms such as "schedule" (60.59%), "management" (42.06%), "communication" (42.94%) and "document" (67.06%) appear very often, both in positive and negative contexts, indicating they are factors in the successes or failures of game development projects.

6 Discussion

The topic of requirements engineering and game development is by no means a new one. As game development is a specialized field of software development, and requirements engineering is an accepted and commonly used part of the software engineering process, the assumption that game development could benefit from requirements engineering processes and methods is only natural.

Along with this long-standing interest in the topic comes a lot of previous research. This body of work varies greatly in scope and style. As game development is a practical real-world problem, it stands to reason that for it to truly be useful, research carried out on the topic should be conducted with the realities of the discipline in mind, if the goal is to solve real problems faced by developers.

It is worth stressing that the findings in this paper apply mostly to smaller independent developers. Larger and more organized studios may have their own methods and processes for dealing with these issues and approach the development process much in the same ways as a traditional software development

project would, but as these larger studios and corporations tend to regard their practices and methods as trade secrets, little information is available on the subject.

6.1 Discussion of the Literature Survey Results

Key problems and issues were identified based on academic research conducted through interviews and studies conducted among game developers and game publishers. Some of these problems make it harder to adopt requirements engineering processes as a part of the game development process, while some are areas where game development could clearly benefit from adopting these processes.

Of the problems discovered, the general lack of formal processes and methods among developers seems to be the most fundamental one. While the emphasis on non-functional requirements and the lack of tools for capturing and modelling affective requirements are also significant problems, they can be overcome with work.

That work will not be conducted if developers are not interested in utilizing theory-based methods and processes, or applying requirements engineering techniques to their work. The reasons for this perceived lack of interest and its remedies are beyond the scope this paper, and a large survey would be needed to chart attitudes and problems before educated guesses could be made. It could be that developers are interested in utilizing more formal methods, but do not have the knowledge and skills needed, or they might not be aware of the possibility, having grown used to doing things their own way.

Note that these are not the only significant challenges or problems game developers are facing, nor are they the only factors making it harder to adapt requirements engineering methods and processes to game development. As an example, game development is a much more multi-disciplinary activity than normal software development. Game development teams employ software engineers, designers, producers, project managers and other computer science professionals just like traditional software engineering teams, but additionally make use of different types of artists (e.g. writers, graphical artists, musicians, animators, sound technicians) and others. Merely finding common vocabulary among these wildly varied disciplines can be challenging, but their variety alone introduces difficulties into the requirements engineering process. Capturing and modelling requirements specific to each of these disciplines requires strong domain knowledge.

Beyond the need for specialized knowledge, all the disciplines of game development may have their own considerations that need to be taken into account, and scheduling can also be challenging. Not all of these components might even be actively worked on during the pre-production phase, where most of the requirements engineering work takes place. While a significant problem, this is not unique to game development, as traditional software development projects need specialized domain knowledge for requirements engineering work as well.

For instance, experts on legal concerns, data privacy or sociology might have specialized domain knowledge needed in the project.

As so many different problems could be discovered so easily, the topic is clearly ripe for further research, discussion and future work.

6.2 Discussion of the Postmortem Analysis

The postmortem analysis further supported the finding that requirements engineering methods and processes are not commonly used in game development. The analysis shows an almost complete lack of keywords relating to requirements engineering in the postmortems. The topic itself was not mentioned once in the 340 post-mortems, which include everything from big budget games to smaller indie products, games created using traditional waterfall methods to agile projects and so on.

As post-mortems deal with factors which contributed, positively or negatively, to the outcome of each individual project, the total lack of mentions could mean that requirements engineering is simply not a concern to any of these developers. This result is not conclusive, of course, as post-mortems are not all-inclusive lists of all contributing factors. However, the fact that no developer mentioned requirements engineering as a factor in the outcome of the project, does give validity to the claim that game developers do not utilize, nor even think to utilize, requirements engineering methods or processes.

As there is next to no discussion on keywords related to requirements engineering, this study did not reveal any conclusive evidence for or against the incompatibility between requirements engineering and the game design document. Keywords such as scope (39.41%) and document (67.06%) are often mentioned in post-mortems, so clearly some kind of issue exists, but based on this study little can be said on the topic.

One notable finding could be the relative low frequency at which terms related to crunch appear in the post-mortems. Crunch is generally considered a widespread problem in the industry. According to a 2016 survey conducted among the International Game Developers Association members [27], 65% of developers reported having experienced crunch, with 52% reporting having experienced it more than twice in the previous year, and the topic has been heavily discussed in media as well. Despite this, the keyword “crunch” appeared in 25.88% of the articles, and the clearly related term “overtime” appeared in 7.35% of the articles.

This inconsistency could be explained by several factors. The post-mortems deal with individual projects, rather than individual developers, the contrary of which is true on the IGDA survey. Thus, even a project where multiple developers reported experiencing crunch would only represent a single item in the post-mortem data. The post-mortems also include many smaller indie projects, which might be more loosely scheduled and could afford to postpone the project rather than crunch to finish it on an external schedule. Finally, the post-mortems include material from 1997 to 2018, and it could be that in the earlier material

crunch simply was considered an inevitable part of working in the game industry and not worth reporting as a factor.

Terms related to formal project management processes and methods appear in the post-mortems quite often, and in contexts which relate to project management: document (67.06%), production (60.29%), schedule (60.59%), management (42.06%), communication (42.94%), scope (39.41%). This means that these issues were considered by developers to be a key factor in the success of the project, whether a positive or negative one. This would seem to be in line with Kasurinen's claim that game development would benefit from more formal, commonly used methods and practices, as they are generally agreed to improve and facilitate these key areas of the development process [9].

These findings demonstrate that there is clearly need for further and deeper studies on the issue. Game development is a growth industry where ever-increasing amounts of money are on the line, depending on the successful outcome of large, expensive and extremely complex software development projects. It is clear that game development could benefit from additional formalization, but in order for that to happen, several hurdles need to be crossed.

Developers need training, and methods and processes need to be adapted and created to better suit the needs of the industry. While these initiatives probably need to be driven by developers themselves, academic research has an important role to play as well. Studies conducted by academics could hopefully breach the wall of secrecy surrounding many developers and help discover both the causes and eventual fixes for these problems.

7 Conclusions

This paper explored the question of adapting requirements engineering methods and processes to game development projects. Based on a thorough reading of state-of-the-art academic research, key problems and limitations were identified. These included: (1) a general lack of formal processes and methods in game development (2) the emphasis on non-functional, affective requirements, which traditional requirements engineering methods and processes are not well suited to (3) emphasis on change as a central development tool, and the need for better change control, which requirements engineering could provide (4) the incompatibility between the requirements document and the game development document, central artefacts in requirements engineering and game development respectively.

To study the validity of these claims, 340 developer-published post-mortems were analyzed algorithmically, using custom programs created for the purposes of this study. Keywords based on academic findings were searched for, and their total number of appearances, as well as the frequency of these appearances, were noted. Additionally, they were analyzed for word correlation to discover, which words the keywords commonly appeared with. This analysis would seem to support the key problems and limitations identified in the literature survey, although due to the limitations of the analysis, and the scope of the identified

issues, more research is needed. Possible avenues for future research could include a similar study on traditional software development projects, to measure prevalence of requirements engineering methods and processes in these projects and use of the related terms in developer communication regarding the projects, and to contrast such prevalences with the ones found here in game development.

Acknowledgement. The work was supported by Academy of Finland decisions 312395 and 313748, and the Business Finland funded Virpa D project.

References

1. Aleem, S., Capretz, L.F., Ahmed, F.: Game development software engineering process life cycle: a systematic review. J. Softw. Eng. Res. Dev. **4**(1), 1–30 (2016). https://doi.org/10.1186/s40411-016-0032-7
2. of America, E.S.: Entertainment Software of America: Analysing the American Video Game Industry 2016 (2017). http://www.theesa.com/wp-content/uploads/2017/02/ESA-VG-Industry-Report-2016-FINAL-Report.pdf
3. Ampatzoglou, A., Stamelos, I.: Software engineering research for computer games: a systematic review. Inf. Softw. Technol. **52**(9), 888–901 (2010)
4. Callele, D., Neufeld, E., Schneider, K.: Requirements engineering and the creative process in the video game industry. In: 13th IEEE International Conference on Requirements Engineering (RE 2005), pp. 240–250. IEEE (2005)
5. Cao, L., Ramesh, B.: Agile requirements engineering practices: an empirical study. IEEE Softw. **25**(1), 60–67 (2008)
6. Hofmann, H.F., Lehner, F.: Requirements engineering as a success factor in software projects. IEEE Softw. **18**(4), 58–66 (2001)
7. Hull, E., Jackson, K., Dick, J.: Requirements Engineering, 3rd edition (2011)
8. IGDA Quality of Life Committee: Quality of Life in the Game Industry: Challenges and Best Practices. Technical Report, International Game Developers' Association (2004)
9. Kasurinen, J., Maglyas, A., Smolander, K.: Is requirements engineering useless in game development? In: Salinesi, C., van de Weerd, I. (eds.) REFSQ 2014. LNCS, vol. 8396, pp. 1–16. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-05843-6_1
10. Kanode, C.M., Haddad, H.M.: Software engineering challenges in game development. In: ITNG 2009–6th International Conference on Information Technology: New Generations (2009)
11. Kasurinen, J.: Games as software. In: Proceedings of the 17th International Conference on Computer Systems and Technologies 2016 - CompSysTech 2016 (2016)
12. Kasurinen, J., Risto Laine: Games from the viewpoint of software engineering. In: Proceedings of the Federated Computer Science Event, pp. 23–26 (2014)
13. Koutonen, J., Leppänen, M.: How are agile methods and practices deployed in video game development? a survey into finnish game studios. In: Baumeister, H., Weber, B. (eds.) XP 2013. LNBIP, vol. 149, pp. 135–149. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38314-4_10
14. Musil, J., Schweda, A., Winkler, D., Biffl, S.: Improving video game development: facilitating heterogeneous team collaboration through flexible software processes. In: Riel, A., O'Connor, R., Tichkiewitch, S., Messnarz, R. (eds.) EuroSPI 2010, CCIS, vol. 99, pp. 83–94. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15666-3_8

15. Newzoo: Mobile Revenues Account for More Than 50% of the Global Games Market as It Reaches $137.9 Billion in 2018 (2018). https://newzoo.com/insights/articles/global-games-market-reaches-137-9-billion-in-2018-mobile-games-take-half/
16. Osborne O'Hagan, A., Coleman, G., O'Connor, R.V.: Software development processes for games: a systematic literature review. In: Barafort, B., O'Connor, R.V., Poth, A., Messnarz, R. (eds.) EuroSPI 2014. CCIS, vol. 425, pp. 182–193. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43896-1_16
17. Paetsch, F., Eberlein, A., Maurer, F.: Requirements engineering and agile software development. In: Proceedings of the Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE (2003)
18. Paschali, M.E., Ampatzoglou, A., Chatzigeorgiou, A., Stamelos, I.: Non-functional requirements that influence gaming experience. In: Proceedings of the 18th International Academic MindTrek Conference on Media Business, Management, Content & Services - AcademicMindTrek 2014 (2014)
19. Petrillo, F., Pimenta, M.: Is agility out there? agile practices in game development. In: SIGDOC 2010: Proceedings of the 28th ACM International Conference on Design of Communication (2010)
20. Petrillo, F., Pimenta, M., Trindade, F.: Houston, we have a problem...: a survey of actual problems in computer games development. In: Proceedings of the 2008 ACM symposium on Applied computing (2008)
21. Petrillo, F., Pimenta, M., Trindade, F., Dietrich, C.: What went wrong? a survey of problems in game development. Comput. Entertainment (CIE) **7**(1), 13 (2009)
22. Pohl, K.: Requirements Engineering: Fundamentals, Principles, and Techniques, 1st edn. Springer, Heidelberg (2010)
23. Shirinian, A.: Dissecting The Postmortem: Lessons Learned From Two Years Of Game Development Self-Reportage (2011). https://www.gamasutra.com/view/feature/134679/dissecting_the_post-mortem_lessons_.php
24. Stacey, P., Nandhakumar, J.: Opening up to agile games development. Commun. ACM **51**(12), 143–146 (2008)
25. UKIE: The games industry in numbers (2018). https://ukie.org.uk/research. Accessed 08 Jan 2019
26. Washburn, M.J., Sathiyanarayanan, P., Nagappan, M., Meiyappan, T., Bird, C.: What went right and what went wrong: an analysis of 155 postmortems from game development. In: Proceedings of the 38th International Conference on Software Engineering (2016)
27. Weststar, J., Legault, M.J.: Developer Satisfaction Survey 2016 Summary Report. Technical Report, International Game Developers Association (2016). https://cdn.ymaws.com/www.igda.org/resource/resmgr/ortfiles__2016_dss/IGDA_DSS_2016_Summary_Report.pdf