Cowboys, Ankle Sprains, and Keepers of Quality: How Is Video Game Development Different from Software Development?

Emerson Murphy-Hill
North Carolina State University
Raleigh, North Carolina, U.S.
emerson@csc.ncsu.edu

Thomas Zimmermann and Nachiappan Nagappan
Microsoft Research
Redmond, Washington, U.S.
{tzimmer,nachin}@microsoft.com

ABSTRACT
Video games make up an important part of the software industry, yet the software engineering community rarely studies video games. This imbalance is a problem if video game development differs from general software development, as some game experts suggest. In this paper we describe a study with 14 interviewees and 364 survey respondents. The study elicited substantial differences between video game development and other software development. For example, in game development, "cowboy coders" are necessary to cope with the continuous interplay between creative desires and technical constraints. Consequently, game developers are hesitant to use automated testing because of these tests' rapid obsolescence in the face of shifting creative desires of game designers. These differences between game and non-game development have implications for research, industry, and practice. For instance, as a starting point for impacting game development, researchers could create testing tools that enable game developers to create tests that assert flexible behavior with little up-front investment.

Categories and Subject Descriptors
D.2.0 [**Software Engineering**]: General – *Standards.* K.8.0 [**Personal Computing**]: General – *Games.*

General Terms
Human Factors, Management

Keywords
Software engineering, games, practices

1. INTRODUCTION
Games are becoming an increasingly important part of the software development industry. Beyond simply entertainment, video games are increasingly being used to train students, soldiers, and medical professionals [1] [2]. Congruent with their growing importance, video games' revenue is increasing as well; video games earned more than three times the revenue of retail software in 2012 [3].

Despite games' importance, they are rarely studied in software engineering research. Of the 116 open and closed source software projects studied in the last two years at major software engineering venues, only 3 were games [4]. Of the projects in two major software engineering corpora, SIR [5] and Qualitus [6], 0% and 3% are games, respectively. The lack of software engineering research about games, despite their importance, presents two problems.

First, if non-game software development is indeed different than game development, past software engineering research will have little impact on games. By analogy, the medical community faced significant criticism for over-enrolling men in coronary heart disease studies. As a result, "procedures and therapies currently used" for the disease are "developed predominantly or exclusively for men" [7]. Software engineering researchers' practice of "under-enrolling" games in studies may likewise result in tools and practices that are inapplicable to game development.

Second, if game development is indeed different from "traditional" software engineering, there are educational and practical impacts. In his book on game development, Bethke states

> *Too often game developers hold themselves apart from formal software development and production methods with the false rationalization that games are an art, not a science. [8]*

If this statement is true, then software engineering educators need to teach their students different skills for game development than for developing other types of software. If this statement is false, then game developers would benefit from adopting the practices of software engineering that are empirically validated.

So: is game development different from traditional software engineering, or is it not? Like most questions, the answer is likely that it is different in some ways but similar in others. Unfortunately, which way it is similar or different has not been systematically studied. This paper's primary contribution is the first broad-based empirical study to explicitly contrast traditional software engineering against video game development.

In Section 2, we survey research on game development and discuss the few empirical studies that do exist. In Section 3, we describe our interview and survey study methodology, then discuss the results in Section 4. We discuss limitations to our study in Section 5, the implications in Section 6, and conclude in Section 7.

2. RELATED WORK
Many books exist with prescriptive practices for developing games. Some describe the developer roles and the high-level process that developers and organizations should use when creating games [9] [10] [11] [8] [12]. In the same vein, Blow's magazine article details his experiences with the fundamental difficulties in game development [13]. These works are based on the experience of the authors and largely do not contextualize game development as a special type of software engineering. In contrast, our findings are

ICSE'14, May 31 – June 7, 2014, Hyderabad, India

based on empirical observations that explicitly focus on the differences between general software engineering and game development.

Recently, several researchers have focused on studying the process of developing games. Ampatzoglou and Stamelos provide an overview of the intersection of software engineering and games, noting the dearth of empirical studies [14]. One such study is Tschang's qualitative investigation of 65 game development project postmortems, finding significant differences between game development and other creative industries [15]. Tschang also developed a grounded theory of creativity in game development [16] and a theory of innovation [17]. Baba and Tschang contrast the spiral model of software development [18] against a new "outward spiral model" of game development, empirically derived from business practice manuals and some number of interviews with Japanese "managers and project team members" [19]. Our work builds on this work by studying differences between traditional software engineering and game development.

Like our work, existing work has empirically investigated game development. Burger-Helmchen and Cohendet interviewed 8 game developers and discovered how communities of developers and users interact [20]. Kultima and Alha interviewed 28 game professionals, finding that they viewed their development process was organic and uncontrollable [21]. Stacey and Nandhakumar interviewed 20 developers, finding that predefined phases in traditional software development models may be harmful to game development [22]. Callele and colleagues analyzed 50 postmortems of game development projects and found most requirements failures occur between the preproduction and production phases [23]. Kasurinen and colleagues interviewed 27 game developers and found they expect adaptability in the tools they use [24]. Musil and colleagues surveyed 13 Austrian game companies, revealing that the industry largely uses Agile practices [25]. Lewis and colleagues created a taxonomy of bugs in video games [26]. In contrast to this prior work, our paper studies broad differences between game development and traditional software engineering.

Also like our work, some existing research has investigated differences between game development and traditional software engineering. Specifically, Petrillo and colleagues analyzed 20 publically-available game postmortems and found that problems encountered [27] [28] and processes used [29] by game developers were largely the same as those for traditional software engineers. One significant limitation to this work is that contributing game developers may be reticent to report some negative aspects of their work, because the postmortems were publically available. In contrast, our work uses anonymized interviews and surveys, which we believe helped respondents be more candid.

Prior position papers have explicitly compared software engineering and game development, namely that of Lewis and Whitehead [30] as well as Kanode and Haddad [31]. In contrast, the work presented here derives its results from empirical grounding.

3. METHODOLOGY

Our study methodology involved two parts, qualitative interviews and quantitative surveys, which we describe below. All study materials can be found at our website.[1]

3.1 Interviews

Protocol. We interviewed developers with experience in both game development and non-game development. The first author interviewed developers either in person if they worked in the Seattle area, or via Skype or phone if they did not. Each interview was completed in an average of about one hour. The interview had four parts.

In the first, the interviewer asked a few demographic questions relating to how much experience the interviewee had.

In the second part, the interviewer asked an open-ended question about what differences the interviewee noticed between software development for games versus non-games. This part allowed interviewees to speak freely about differences without the interviewer biasing their responses.

In the third and fourth part of the interview, we presented interviewees with a list of topics to prompt them to discuss topics that they had not explicitly considered. We gave half of interviewees the topics from the 10 areas in the Software Engineering Body of Knowledge (SWEBOK) [32], such as *software maintenance* and *software testing*. We gave the other half of interviewees Humphrey and colleagues' list of general work features from applied psychology [33], such as *social support* and *problem solving*. We chose SWEBOK to ensure that software engineering topics were discussed, and the general list to make sure that we covered a breadth of potential differences. The difference between the third and the fourth part was that in the third, interviewees chose 2 or 3 topics to discuss, whereas in the fourth, the interviewer chose 2 or 3 topics. Moreover, in the fourth part, the interviewer selected topics that been discussed the least in previous interviews, to ensure even coverage of the topics. As a result, each topic was discussed at least twice across all interviewees. Finally, we thanked interviewees and debriefed them by informing them about what we planned to do with the data.

Participants. We interviewed people with experience with both game and non-game development by searching LinkedIn,[2] which contains resumes of professionals. We searched for LinkedIn members who were part of the "Game Development" group, which included more than 65 thousand members at the time of the study.

Our initial search results included non-developers, including designers with experience only in entertainment. We thus added the "engineer" keyword to our search. We also aimed to focus on developers who made video games, so we included the following keywords in our search: PSP, PS1, PS2, PS3, PlayStation, Xbox, Wii, and GameCube. This left 207 potential candidates to interview.

We further narrowed our selection of potential interviewees by manually scanning the search results for several criteria, making sure that each potential interviewee reported at least 2 years of game development experience within the last 10 years; at least 2 years of non-game development experience within the last 10 years; and listed contributing to specific game titles. We performed this search through each of the three LinkedIn accounts of the authors. We chose candidates from "2nd degree connections", meaning associates of associates, because LinkedIn does not allow the unfiltered viewing of profiles of community members of 3rd or more degree.

[1] http://people.engr.ncsu.edu/ermurph3/experiments/Games.pdf

[2] http://www.linkedin.com

Thirty-eight people fit our criteria, all of whom we contacted by email or social networking. Because many developers did not respond immediately, we followed up repeatedly until we had interviewed enough developers to reach *saturation*, that is, until we were not discovering any new differences. We reached saturation at 14 interviewees. In the remainder of the paper, we label each interviewee P1 through P14.

Nine interviewees were working on a game at the time of the interview and five worked on other software. Five interviewees worked at Microsoft. Thirteen interviewees were male. Below, we summarize the self-reported game and non-game development experience data from interviewees:

		Games	Non-Games
Median years of development experience		8.5	8.5
Number of interviewees with "extensive" experience in…	Programming	10	12
	Design	6	5
	Management	7	4
	Audio/Visual	2	3
	Testing	3	5

After recruiting, we found that P13 did not have software engineering experience but instead worked as a hardware engineer with software developers, prior to working in games. We included him in our interviews because we felt his current game role, as a producer, would provide a valuable perspective. However, because of P13's lack of software experience outside of games, we only use P13's data to illustrate game development themes brought up by other interviewees.

Data Analysis. We used a transcription service to transcribe the audio, then coded the interviews using Qualyzer.[3] We coded transcripts using the same SWEBOK [32] and general work [33] topics we used to prompt interviewees.

3.2 Surveys

Protocol. We created a 10-minute survey designed to assess differences between game and non-game development. Our survey aimed to quantify the qualitative differences expressed by interviewees over a range of developers.

We used our results from the interviews to write 84 candidate statements that asked respondents to rate their agreement with each statement on a 5-point Likert scale, from Strongly Disagree to Strongly Agree. For example, one statement was "Creating my software is challenging."

We removed statements that we felt were the most ambiguous, were the most difficult for developers to accurately self-assess, or were most similar to one another. This reduced our list to 28 final statements, which we felt would keep the survey sufficiently brief.

The survey also collected demographic information.

Participants. We recruited engineers and testers to participate because many statements on the survey reflected technical concepts that engineers and testers would be most qualified to rate.

We recruited three sets of potential respondents within Microsoft: 300 who worked on games (who we will refer to as the "Games" set), 300 who worked on Microsoft Office ("Office"), and 300 from across the company but did not work on games or Microsoft Office ("Other"). We chose these sets in order to contrast responses; if Games respondents provide significantly different responses than Office and Other, this provides quantitative evidence to establish a difference between game and non-game development. The reason for choosing two types of non-game developers (Office and Other) was that we were unsure whether high variances in product differences would overwhelm game versus non-game differences. Thus, to augment a diverse sample of developers (Other), we also sampled a more homogenous set from single product (Office).

40% of recruits completed the survey. Below we summarize the self-reported experience and backgrounds from respondents:

	Games	Office	Other
Mean years at Microsoft	4.4	7.1	5.1
Mean years of development experience	10.7	11.0	8.8
Number of engineers	113	61	82
Number of testers	32	39	37

Data Analysis. We examined distributions of Likert responses for each of the three participant sets and compared them using a Wilcoxon rank-sum test. Although we report the full results in Section 4.3, along the way in Sections 4.1 and 4.2, we link interviewee comments with survey responses by referring to survey statements like so: [S1]. We number statements in the order in which they appeared in the survey, S1 through S28. We annotate each with whether they are statistically significant, like so:

[✓✓S1]	Significant differences between Games and both Office and Other that confirm interviewees' responses
[✓S1]	Significant differences between Games and either Office or Other that confirm interviewees' responses
[S1]	No significant differences
[×S1]	Significant differences between Games and Office or Other, but opposite of interviewee responses

Other outcomes were theoretically possible, but did not occur. One such example could be [×✓S1], meaning that both Office and Other were significantly different from Games, but Other confirmed interviewee responses while Office opposed them.

4. RESULTS

In this section, we report results based on the interview topics. We combine several topics into one when interviewees had little to say about an individual topic. In some cases, we have anonymized parts of quotes to maintain interviewees' privacy.

4.1 SWEBOK Topics

4.1.1 Software Requirements

Nearly every interviewee made a strong statement about differences between game requirements versus requirements in other software. In essence, games generally have one and only one requirement – that they are "fun."

Interviewees noted that functional requirements are better suited for non-games than games [✓✓S6]. As P13 noted, as a game developer, you are "designing an experience, an emotional experience... It's something supposed to be fun which is very subjective" and is an "artistic achievement." Rather than strict requirements, the game designer "will give you a set of high-level

[3] http://qualyzer.bitbucket.org

goals that they want out of a certain feature, but they don't even really know what they want" (P3).

Interviewees pointed to several reasons why requirements are so much more subjective in game development. As P3 said, even with a clear vision from a designer, when the vision is implemented, it may not be fun. Another reason is that the consequences of unfulfilled requirements in a game are less problematic than in other software; game users move on quickly after a single incomplete experience, but if a user is using email software and the email does not get to his boss, the user "could get fired" (P1). Another reason for requirement differences between games and non-games is because game user experiences tend to be significantly different from non-game experiences. P8 gave an example: in an e-commerce application, a user has a task to complete that typically takes only a few minutes. In contrast, in some games people play for hours straight on a daily basis over the course of months. As a result, the requirement for games is that the user should be able to stay engaged on multiple timescales, and the mechanism to achieve that will vary from game to game.

Instead of requirements specifications that may be found in general software development, guidance for what a game should do comes from other sources. Game designers with a particular vision are one source. If a game has a predecessor, game requirements can come from users, yet the "fun" requirement caveat applies – if a user wants something, it may not be implemented because it may not enhance fun. For games that are re-released every year (such as sports games), another source of guidance can come from previous iterations; as P8 suggested, game developers may ask

> *"What did players play two years ago?" and really fixing the issues that were there some time ago. And also playing somebody else's games and comparing your own game to somebody else's, trying to make it better or trying to solve some of the problems, try to differentiate.*

Although usability and user experience is often an important quality of non-game software, the way users interact with games means that the user experience requirements for games and non-games are different. For example,

> *Game play is more about feel. It's hard to dissect scientifically. (P13)*

In sum, requirements appear to be more subjective in games. As we will discuss in subsequent sections, this has several consequences for the way games are developed, compared to non-game software.

4.1.2 Software Design

Interviewees explained that in games they tended to do less design as a planning activity for a few reasons.

First, because the "fun" requirement is nebulous, many plans that are made will not produce a fun game. Thus, participants said that there is wasted effort [S5] if part of a game turns out not to be fun, and that waste is multiplied if additional effort was put into design. For example, according to P11, the game producer

> *…doesn't want you [the developer] to go and spend a whole bunch of time planning out how you're going to do this thing that he's asked for because he might change his mind in a week or two. Knowing that, he knows deep down that designing is useless because he's going to be constantly changing things…*

As a consequence of less design-as-planning, interviewees reported very little up-front thought put into architecture [S8]. As P11 put it, "there is very little design… on the architecture of games. It's more of a 'we needed to do this yesterday, go do it.'" Interviewees did not totally discount architecture in games, instead noting that it was less important in one-off games but more important in game series where components are reused across releases. While the lifespan of non-game software similarly has an influence on how much architecture is designed up-front, the problem appears to be especially acute for games because games' lifespans are less predictable. Paradoxically, game developers may go into "architectural debt" early on [S3][S4] because there is such a high probability that parts of their code will be thrown away, yet if the game is successful and they wish to maintain or extend it, the architectural debt must be paid down.

Second, interviewees reported less design-as-planning for cultural reasons. As P5 explains it, "the design process as well is a little bit different because… creativity tends to be rewarded more than technical prowess." Likewise, P11 pointed at the culture as well, noting that the fundamental difference is that game development is "a young man's game and because of that, the whole process of building games is an immature process."

4.1.3 Software Construction, Tools, and Methods

A theme that interviewees repeatedly mentioned was code and tool reuse. Interviewees mentioned that they believed that there was little code reuse between and within games, compared to non-game software [S1]. For example, in games:

> *There's a lot of hacks and kludges to get things working… I'm sure you would find tons of duplication of effort, definitely. I've been an audio programmer on [X] different games and I've written [X] different audio engines. (P11)*

One reason that there appears to be less code reuse is because games frequently have a significant emphasis on performance, and that project-specific performance tuning is necessary:

> *It's difficult to find a highly-optimized solution that's going to work for your particular game because [your situation is] specific to the type of experience that you're trying to create. That just trickles to everything, the kind of physics that you have in your game, the kind of visuals you have in your game. (P5)*

The above quote implies that another reason why there is less code reuse in games is because reuse implies similarities between software, yet games emphasize innovation. P5 echoed this, saying,

> *The thing that makes video games unique is gamers want unique experiences… With [general software], you don't want to change too much [because of, for example, the] backlash that Microsoft received every time they move a button or change an interface.*

However, several interviewees noted that reuse takes place frequently in games, just in different ways. One way was that code is recycled between subsequent game releases (P11). Another source of reuse is in game engines, where multiple games and multiple companies can reuse a core framework (P2, P9, P11).

Another source of reuse is the reuse of tools. Interviewees mentioned that tool pipelines are critical for building games, and that these pipelines may be used within organizations. Whereas general software development tools, such as refactoring tools, enjoy widespread availability to programmers, game tools appeared to be developed more commonly within organizations [S15]. For example, P5 summarized the tooling differences as:

> *[In general software development,] you might be building Word or Excel or something like that [and use] some zip… tool or something. But you're building… video games, you are sometimes building the resource compiling tools or tools that*

are intended to extract 3D assets from other software like Maya or Max and then convert it into a native format that then your engine can load and render and process. So the tool pipeline is incredibly important to video game development and it's probably I would say almost larger than the game itself.

4.1.4 Software Testing and Quality

Although software quality is important in both games and non-games, the practice of testing appears to differ significantly. The reason that quality is important in games is that, as P10 put it:

It's almost like watching a movie... so that experience for you to immerse [yourself] in the game experience, it has to not create anything that would take you out of that immersion.

One significant difference is that there appears to be significantly less test automation in game development:

In general, that's something that is very heavily done in games – unit testing, regression testing, all the different types of software testing you don't really see in video games, either. Games are tested, but at the game play level." (P9)

As this quote suggests, rather than using automated, low-level testing, testing in games tends to be run more often at a high level, either by a human playing through the game [S16], or as a script simulating what a human would do [×S17]. Traditional software engineering best practices dictate that neglecting low-level testing, like unit testing [✓S18], is risky [32], so it is worth investigating why game developers appear to ignore this advice.

One reason that games are difficult to write automated tests for is that it is so difficult to separate the user interface from the rest of the game. For example, P4 noted, best testing practice is to

use MVC or one of the other patterns in order to try to separate things so they're more easily testable. This, in general, is more difficult in computer game development because of the extent to which the user interaction is so pivotal… A lot of times, I'll see developers just throw up their hands and say, "No, I'm … not gonna worry about unit tests at all." It's much more common in the game industry than it is in other places.

Another reason that it is difficult to write automated tests for games is that it is harder to explore the state space in games [✓S19]:

[Games tend] to have a large number of states that are user driven… If you tried to create a test matrix for it, you end up either having an immense test matrix or you end up restricting the game design. In many cases, severely. (P4)

Another challenge is simply asserting what the correct behavior is:

If I'm playing a game, and maybe like I shoot this guy and I see like a visual artifact like bouncing, do I care? (P1)

I could… write unit tests and say this enemy dies in two hits but it's not really meaningful because it's not really that he dies in two hits that's so important, it's that he dies in the right amount of hits that the game designer thinks is the good amount. (P12)

Yet another challenge to writing tests is the non-determinism that occurs in games due to multithreading, distributed computing, artificial intelligence, and randomness injected to increase gameplay difficulty. As P5 put it,

Definitely maintaining determinism in a sort of multi-player environment is much more crucial that a single player [environment]. You can definitely introduce very, very strange bugs in, say, a game that wasn't designed to be multi-player and is multi-threaded as well and is doing a lot of these complicated AI behaviors and physics.

Interviewees reported that there was also a strategic reason for doing less automated testing and more human testing: automated testing is fragile to frequent changes, whereas human testing tends to be more resilient. In this sense, automated tests reduce agility. As P12 put it, "the game designer changes his mind so often that tiny [test] tweaks happen all over the place."

Interviewees also stated that another reason human testing is so common is because it is relatively cheap, because game play testers are less expensive than software testers:

So the cost is less; so that's the thing. Would you rather have one guy that can do this automation or have four guys who can actually go play the game? (P1)

Game testers illustrate Braverman's sociological notion of *deskilling*, where technology enables skilled workers (developers who can write automated tests) to be replaced by unskilled workers (play testers) [34]. This deskilled work stands in stark contrast to that of game programmers, who can be described as doing *craft work*, which depends "on special skills [which is marked by] the lack of standardization of the product" [35].

One of the consequences of lack of test automation is that, once a bug is reported, it is difficult to diagnose and debug [S20]:

A lot of times the bug reports tend to come back and it's more like, 'oh well I pushed this button and I was in this corner and the game locked up.' So you have to kind of go back and try and reproduce that type of situation. (P5)

4.1.5 Software Maintenance

Similar to game developers' delay of architectural design, maintenance also appears to be something that is often delayed in games later than it would be in non-game software. As P12 put it,

There's always a feeling in games that you almost don't really have to maintain it. In [non-game software], what's going to happen is most of your development time is actually going to be in maintenance, you really have to make sure that the code you write, the abstractions that you come up with for your code are clean, that they're maintainable, that you'll be able to go in and make changes as the years go by and presumably your system stays in operation. With a video game… there's kind of a sense that you're the last one to touch the code.

Also like the up-front design of architecture, there is a tradeoff between improving maintainability early and the likelihood that this effort will result in waste because the game will not be a success (P2, P3). Interviewees also delayed improving maintainability in games due to lack of management buy-in. As we will discuss in Section 4.1.7, one reason for less management buy-in appears to be because managers tend to be non-technical [S21]:

Anytime you're going to be working on clean code, you have to have buy-in from management or you have to have an engineering team that's willing to tell management to back off because in the end, you're going to be sacrificing time to do that. (P12)

Another reason for lack of maintenance in games, at least from a programming perspective, is that product releases may entail changes to content rather than changes to behavior. While non-games may compete in the marketplace based on new features, that may not be the case for some games:

[In new releases] they put in a bit of stuff for rendering improvements and usually they would add maybe one gameplay feature. Other than that, it's just [content] changes, so for [a specific game company], a game like [a specific sports title] is basically printing money. (P11)

Several interviewees indicated that the cloud changes the way they maintain games. The increasing popularity of cloud-based game services such as Steam [36] means that the maintenance process for games is starting to look much like the maintenance process for non-games.

4.1.6 Software Configuration Management

Several interviewees noted that configuration management is especially important in game development compared to other types of software development. Part of the reason appears to be that because traditional automated testing is so rare, there exists a more urgent need to automatically build a system as a kind of smoke test:

It has to run on different machines…. [Two large game companies] have huge testing centers where they've got pretty much every combination you can think of CPU, RAM, hard drives, and different graphics cards and drivers… And they test your game on each piece of hardware to make sure that there are no faults. (P9)

Still, interviewees reported that the configuration management was sometimes "very chaotic" (P12). Part of the problem seems to be lack of code review [✓S2]:

When we did [a specific game title] there wasn't really any code review at all. (P12)

I've also seen quite a lot of damage done to the software by people checking in patches that never should have been checked in, or they should have been reviewed by someone… [it's] still a major pain to undo changes. (P7)

Another difference in configuration management between game and non-game development is that games tend to have significant amounts of content, also called assets or resources:

You just end up having lots more resources, lots more properties that you have to track and trigger at the right time. (P4)

P11 indicated that content can become a liability to configuration management, both for technical reasons and for social ones:

If the artist is lazy or if the artist is new and inexperienced they don't know all of the ins and outs of things that they can't do so they tend to do whatever they were trained to do and break that thing a lot… [In non-games,] code tends to be more segregated so if a guy checked in code that complied but doesn't … I don't care… When somebody checks in broken art, that can crash you right off the bat, even if it's got nothing to do with what you do.

Based on this, configuration management is a challenge for games, both because asset changes more likely to induce failures, and because the people who are checking in asset changes may be less familiar with how to use configuration management.

4.1.7 Software Engineering Management

Interviewees suggested that a significant difference between management in games versus non-games is that people in game management positions tend not to have technical backgrounds:

Engineering management, in my time with [two large game companies], the person that I always reported to… didn't contribute code…. half of them had no engineering experience at all. (P3)

One reason appears to be a culture that discourages engineers from moving into management [S24]. P7 notes the attitude that

if a technical person [moves to management, they are] probably wasting their time. And that's an attitude that I've seen in just about every [game] company.

Interviewees pointed out that a consequence of non-engineers in management roles is that it is hard in games to communicate engineering issues [S22]. For example, P3 gave a recent example of a manager not understanding why one developer was unable to fix a bug while another developer had fixed thirty bugs in the same amount of time. P3 had to explain to the manager that bugs are not equivalent; the first developer had a very difficult frame rate bug, while the second developer was fixing typos in the user interface.

Interviewees also gave examples where non-engineering management did not respect important engineering activities because the activities had no immediate impact:

There's a lot of short-term thinking… maybe the benefits [of long term investments] are more nuanced because being able to tell you that I have a more flexible or more Agile piece of technology is the possibility of a benefit in the long-term… if I'm a project manager… those things don't necessarily translate on a balance sheet the same way. (P3)

Interviewees also discussed non-technical management being shielded from the consequences of engineering activities:

If you have the engineers, the cowboy programmers that'll go in and save the day, that can do a disservice actually because it may hide some serious problems. So for some of the management, they could legitimately say, "Well I didn't know it was that bad," and that might actually be true because we always ship their games on time." (P3)

As a consequence, P3 noted that when software engineering practices are introduced into a game organization, it typically comes from engineers rather than management [S23]:

I can tell you it's certainly a grassroots thing. It's engineers who had sort of been there for a few years. They're the ones who ultimately have to endure the pain when you're working with software that's breaking all the time. They have to fix it.

4.1.8 Software Engineering Process

Nine interviewees used the word "Agile" to describe the process of developing games [✓✓S9]:

When I worked in games, I got exposed to Agile, Sprint, Scrum… In the more traditional [large company] way, … they're the opposite…. One year long, they know exactly what's going to happen every week. It's very different. (P1)

It appears that the unpredictability in games is what makes Agile a good fit. In fact, arguably some game organizations developed their Agile processes well before Agile became popular in other kinds of software; P2 mentioned the Cabal group structure used at the game company Valve in the late 1990's [37], likening it to the Scrum methodology with a greater focus on interdisciplinary teams.

Adhesion to Agile varied, according to interviewees, who implied that Agile is sometimes a euphemism for a lack of process [✓S10]:

[Game developers] operate in a more, we'll call it, Agile mode, or if you like, hacking mode... It's almost a full hacking thing, but there's a lotta really smart people here, so it works. (P2)

Perhaps one reason game teams exhibit lack of process is because the notion of imposing control goes against a core principle of game development, that is, the importance of creativity [✓✓S11]:

We've got so many specialists on the team, so the kind of planning that you usually do in Agile doesn't work quite so well… You know [specialists] are more concerned about the creative process than an engineering process. (P4)

Finally, with respect to process, interviewees reported being under significantly more pressure to release the software on time for games than for non-game software:

I think perhaps part of it is that the schedules are usually very tight, and there's a sense that you can't afford to lose any hours. In fact, a lot of teams put their teams on incredibly, incredibly pressured schedules. (P7)

Inflexible deadlines [S14] may be one reason for the intensity:

You can't shift Christmas, where when [a non-game] is going to ship, somewhat you can slip lots of things… some games choose to ship at other times of the year, but most games ship at Christmas because that's where most of the buying is done. (P4)

4.2 General Work Topics

In this section, we discuss differences between game and non-game development in terms of general work topics. No common themes emerged from several general work topics (*job complexity, information processing, social support, feedback on the job, feedback from others, experienced responsibility, ergonomics, task identity, or task variety*), so we did not discuss them in this section.

4.2.1 Problem Solving and Skill Variety

Interviewees identified three main differences between game and non-game development in job complexity and problem solving.

First, interviewees noted that developing games presented distinct technical challenges [S25]. For example, P2 suggested that the often intense use of graphics in games meant that there tended to be a need for "higher-level math and specialized knowledge." P11 summarized the differences as:

Some of the hardest programming I've done has been in games… In business programming, you don't care how fast that code runs, you don't care how much memory it [consumes… In] games, you do have to care about… [making] small block allocators for memory allocation or worrying about memory fragmentation or disk speed load times and things like that.

Second, the subjectivity inherent in meeting the "fun" requirement gives game developers complex problems to solve:

Game companies might be tougher, just because… it's more creative and less structured. (P14)

Third, interviewees suggested that a wider variety of skills is required to develop games [✓✓S12], which can make game development more challenging if a developer lacks those skills:

So a wide range of… topics go into making a game whereas regular non-game development tends to be a little bit more domain-specific, a little bit more narrower in terms of the knowledge you need to employ to specific development. (P10)

P14 gave a specific example, suggesting that monetization of games is one special skill that is required in game development that makes the job more challenging:

Now it's more about monetization… the majority [of game companies] today are going to be much more focused on "what tactics can we use to entice our users to spend money while they're interacting with our product?

4.2.2 Autonomy

Of the interviewees we asked, most said there were no differences in how much autonomy they had between game and non-game software development. One interesting exception was from P10, who suggested that, because many of the challenges that face video game developers are creative in nature, game developers must have a high level of autonomy.

4.2.3 Specialization and Interdependence

Interviewees' remarks regarding specialization and interdependence focused on the broad range of skills required to make games, compared to non-game software development:

That's part of how I've been successful because communication is a very important skill in being able to sort of bridge these gaps between different disciplines. (P3)

As P3 suggests, the interdisciplinary workplace in game organizations [✓✓S13] entails the need for conflict resolution:

I've heard that exchange going on many times during a variety of game development project. And, typically, "Okay, well no, you can't have that. But lets try to get good sound and a good tradeoff."… That's how you negotiate something." (P4)

Beyond expertise, discipline diversity has benefits in game teams:

[Designers and artists are] keepers of quality and so … anything that sticks out may bug them more than it would a developer. (P6)

Other than these keepers of quality, game development groups sometimes use specialists typically not seen in other types of software development. P2 related the story of the game Diablo III, whose fun was reduced after the virtual money supply exceeded people's ability to spend it, so "all the prices got driven [up] and hyperinflation set in." To foresee and prevent this problem in future games, P2 noted that game companies are now "hiring things like sociologists and anthropologists and economists, people you think wouldn't have much of anything to do with games."

Interviewees noted that non-game software sometimes also has specialists as well, but the need is not as great. Instead, developers in non-game groups tend to be generalists:

My team is probably a typical team for a Web development company, where you have a number of people who know how to write JavaScript… and they know also how to write… server code… Typically, they do both. (P8)

4.2.4 Interaction Outside the Organization

Interviewees indicated that game developers tend to have a stronger tie to the customer, both because game players tend to be more engaged than general software users, and because meeting the "fun" requirement is difficult without understanding the customer:

I doubt that there would be the same level of engagement from consumers who are using, you know, even [a productivity application] I would say. There's probably not as many people who are contributing bug reports and things like that as there are in games. (P3)

4.2.5 Knowledge of Results

Interviewees reported few differences in terms of knowing the impact of their software. One difference was suggested by P2, who reported that one clear indicator that an organization did well is whether is game is profitable and wins awards. P2 and P11 both noted that, as with all software, in a large organization it is difficult

to isolate the contributions of a single individual. Nonetheless, P2 and P7 both reported on a kind of "celebrity status" afforded to individual developers of popular games that does not exist for non-game software developers [✓✓S26]. P7 related the story of visiting a foreign country and telling the cab driver that he was a developer of a popular driving game franchise:

> *He literally just got his cell phone and started calling all of his friends telling them that, you know, "Here's a guy who built [this specific driving game]."*

Finally, P2 made an interesting comment that, although he plays the games he develops for testing purposes,

> *I don't play the full game, so don't know the full experience. For instance, [a specific first person shooting game], even though I worked on it for years and years and years, almost five years, I've never actually played it… It's kinda like film stars that star in movies often don't wanna see themselves on the screen after they're done with production.* [S27]

4.2.6 Significance and Experienced Meaningfulness

Interviewees reported finding meaningfulness and significance in largely by seeking out challenging problems and through innovation, which are present in both games and non-games. However, interviewees appeared to view non-game software as having a more meaningful impact than game software [✓S28]:

> *I mean we all know that we're just building a game and … it's not going to be as important as some other business-critical software. (P8)*

Nonetheless, interviewees felt that developing games was still meaningful for a number of reasons.

First, interviewees found meaning in their work party by knowing how many people use it. Applications like Microsoft Office are generally used by a larger number of people than games like Halo, though this varies from software to software.

Second, interviewees distinguished games from non-games in the kinds of interactions people had with them. Respondents largely reported games were built for entertainment purposes (with the notable exception of "serious games" [1] [2]) whereas other applications allow people to be productive or creative. While respondents felt there was value in both, non-game software is meaningful in that it can allow users express themselves and create things they may not easily be able to create, while games can provide positive memories and experiences. Games also influence users at a more emotional level than non-games, which adds meaningfulness to the experience of developing those games. Interviewees implied that there were other nice side effects of games, such as that they can promote socialization in families.

While P2 worked in the game industry for a significant amount of time, he felt conflicted about the meaningfulness of game development. While it is meaningful for the reasons expressed above, he likened his role to pushing drugs on people in that both games and drugs can provide positive, escapist experiences. Likewise, P13 expressed concern about games that promote violence or that are demeaning to women. These two themes rarely arise in non-game software.

4.2.7 Physical Demands and Work Conditions

At the beginning of this study, we considered removing physical demands and work conditions from the list of topics that we would ask interviewees about, because we assumed that there were few differences between game and non-game software development. While several interviewees confirmed this assumption, others surprised us with differences in these areas.

First, several interviewees reminded us that the video game industry is notorious for requiring developers to work long hours, which requires a certain type of physical endurance. Others spoke of the emotional strain that long hours places on developers. For example, P4 noted that "even though I love games, I don't want to work in a game, because I know it's going to be like 12 months of, like, not seeing my family on the weekends." A couple of interviewees retold the story of an anonymous spouse of a game developer who posted an open letter in 2004 to Electronic Arts' executives, a letter that lamented the impact that the poor working conditions have on developers' family lives.[4] This letter struck a chord with many game developers, although apparently long hours are still the norm, according to interviewees. While interviewees acknowledged that long work hours also occur in other kinds of software development, they held that the phenomenon is significantly more severe in game development. P1 went so far as to compare game development managers to "slave drivers."

Second, interviewees noted a few environmental effects of working in games that they had not noticed in non-game environments. P6 described working on a game title for a motion-based game system (such as Nintendo Wii or Microsoft Kinect), where the test team found development physically demanding – in fact, some testers sprained their ankles as they were jumping and ducking in an attempt to test the motion-based aspects of the game. P13 noted that in some game development organizations that make first-person shooters, developers are essentially exposed to years of constant simulated gunfire as developers test their software. Similarly, P6 said that game development teams tend to be high energy, where there is "more music playing and more people talking and more footballs being tossed down the hallway than non-game teams." Some developers found this so distracting that they worked from home.

Finally, P2 noted that an "occupational hazard" that is "unique to gaming" is that most developers at one company had "acquired motion sickness." According to P2, in the course of testing game via play-through, developers are exposed to suboptimal game experiences, such as low frame rates and unresponsive controls:

> *If [developers] do it for a long time… they actually can't stand to look at games anymore because that makes them ill. So what happens is that they learn to feel motion sickness by looking at laggy stuff all the time. Now the reason that this doesn't appear in the general populace [is because] by the time you're done [with the game] and you ship, the performance is tuned and optimized.*

We found this story interesting, so we conducted a survey of 165 Microsoft employees who just got off public buses to determine whether game developers are more susceptible to motion sickness. The study did not uncover any significant differences between the two groups. The curious reader can find details in our companion technical report [38].

4.3 Survey Results

Table 1 summarizes the survey results (refer to Section 3.2 for methodology). The Statement column shows the statements presented to interviewees. The next column indicates the label,

[4] http://ea-spouse.livejournal.com/274.html

Statement		Likert Distributions			Effect Size		p-values	
		Games	Office	Other	Games vs. Office	Games vs. Other	Games vs. Office	Games vs. Other
Being able to communicate with non-engineers is highly valuable in my job.	S13				1.2	1.3	✓✓.000	✓✓.000
My software is well tested by unit tests.	S18				-0.2	-0.8	.245	✓✓.000
When I tell people outside of my company about the software I work on, they are impressed.	S26				0.5	0.5	✓✓.000	✓✓.000
My team uses a waterfall process, rather than an agile process.	S9				-0.3	-0.6	✓✓.006	✓✓.000
Creativity is highly valued on my team.	S11				0.4	0.3	✓✓.000	✓✓.000
Creating my software requires a team of people, each with different skills.	S12				0.3	0.3	✓✓.004	✓✓.001
It's difficult to write thorough automated tests for my software because it's so complex.	S19				0.1	0.4	.507	✓✓.011
My software has clear functional requirements.	S6				-0.4	-0.3	✓✓.003	✓✓.013
My software is well tested by manual simulation (e.g., scripts that thoroughly use the software).	S17				-0.5	-0.4	✗.001	.020
The last bug I fixed was difficult to diagnose.	S20				-0.3	-0.3	.082	.021
After my software is released, I would like to use it outside of work.	S27				0.2	0.3	.192	.023
My software is well tested manually(e.g., paid testers thoroughly use the software).	S16				-0.3	0.3	.064	.024
Whether requirements are met in my software is highly subjective.	S7				0.3	0.3	.081	.044
Most of the feature code I write will probably be included in the shipped software.	S5				0.2	-0.3	.579	.045
Creating my software is challenging.	S25				0.0	0.1	.533	.048
From a technical perspective, it is easy to reuse others' code when creating my software.	S1				0.1	0.2	.299	.080
Most of the code I write is reviewed by other people.	S2				-0.4	-0.3	✓✓.004	.083
My software creates value for society.	S28				-0.3	-0.2	✓✓.010	.121
When my team introduces a software engineering practice, the initiative usually comes from managers.	S23				-0.2	-0.2	.101	.153
My team has flexible release deadlines.	S14				0.0	-0.2	.924	.180
My team makes most of the tools I use.	S15				0.2	0.2	.178	.201
My team adheres strictly to a process (for example, scrum or waterfall).	S10				-0.4	-0.2	✓✓.011	.215
I often discuss technical issues with my manager.	S22				-0.3	-0.2	.159	.237
In my team, engineers are encouraged to move into management positions.	S24				0.0	0.1	.858	.453
My manager has a lot of engineering experience.	S21				-0.2	-0.2	.975	.554
My software has high technical debt (for example, a lot of hacks).	S3				-0.1	0.0	.436	.820
My software's architecture evolves significantly as the software gets more mature.	S8				0.1	0.0	.237	.876
The technical debt is likely to be paid down in the future (for example, through refactoring).	S4				0.2	0.0	.276	.992

Table 1. Survey Results. Orange cells indicate where game developers disagree more strongly with the statement than Office or Other developers, blue cells where they agree more strongly. Green cells represent statistically significant differences.

used previously in this paper. The three Likert Distribution columns indicate the distribution of agreement for each respondent set (Games developers, Office developers, and Other developers). The leftmost bar indicates strong disagreement, the middle bar indicates neutrality, and the rightmost bar indicates the strongest agreement. For example, most game engineers strongly agreed with S13.

The two Effect Size columns indicate the difference in means between Games and Office in the first subcolumn and the difference between Games and Other in the second subcolumn. For example, the mean response to S13 for game developers was a score of about 4.5 (between "Agree" and "Strongly Agree") whereas the mean response for Office developers was 3.3 (between "Neutral" and "Agree"); as a consequence, the effect size is 4.5 – 3.3 = 1.2. Effect sizes are additionally colored on a gradient from blue to orange; blue colors means game developers were more likely to agree with the statement and orange colors mean they were less likely to agree.

The last column, p-values, indicates the degree of statistical significance between Games and Office in the first subcolumn and between Games and Other in the second subcolumn. The table is sorted by the last column. Statistically significant differences are highlighted in green (originally α=.05, but α=.016 after a Benjamini-Hochberg correction for false discovery [39]).

The survey disconfirmed several of interviewees' claims about differences between games and non-games. For example, engineers' likeliness to be encouraged to move into management [S24] was very similar across all three groups. One explanation is that for this question, and likely several others, this trait is pervasive to company culture across Microsoft.

Overall, the results of the survey do confirm some differences. Based on statistically significant differences between Games and both Office and Other, we can say with some certainty that:

- Game developers have less clear requirements than non-game developers. [✓✓S6]
- Game developers tend to use what they perceive as an Agile process more than non-game developers. [✓✓S9]
- Creativity is valued more in game development teams. [✓✓S11]
- The ability to communicate with non-engineers is valued more on game development teams. [✓✓S13]
- Game development requires a more diverse team. [✓✓S12]
- People are more impressed by game developers' work. [✓✓S26]

5. LIMITATIONS

The reader should consider several limitations when interpreting our results. First, the interviews and survey were limited, albeit in complementary ways. The interview findings have limited generalizability because we interviewed few developers, although the number (14) was on par with other interview studies (Section 2). While the survey's large number of respondents afforded much better generalizability, its short length means we could cover only a few topics. Thus, while the interviews and surveys individually provide limited insights, the combination of the two provide a substantial contribution towards understanding the differences between developing games versus other types of software.

Second, in our interviews we sampled only from people with LinkedIn resumes; it may be that people listed on LinkedIn differ in some way from those who are not. Likewise, we conducted the survey entirely within Microsoft, so the results may not generalize

elsewhere. Although our experience has been that Microsoft developers use a range of software practices, it seems likely that Microsoft's non-game culture in testing, for example, has infiltrated its game development teams. However, that we were able to detect differences between game and non-game groups in Microsoft is noteworthy.

Third, our interview structure focused on differences between game and non-game development, rather than similarities. Thus, interviewees may have been inclined to exaggerate differences.

Fourth, in both the interviews and surveys, we asked people about subjective opinions, which may sometimes differ from reality. For example, participants in both studies reported on doing Agile software development, but different developers likely have different opinions of what "being Agile" means.

Fifth, our study combined all "game developers" into one homogenous group, even though, as interviewees pointed out, practices vary between teams within the game industry. The same limitation applies to our characterization of "non-game developers." Thus, while this kind of conflation is common in this type of study, the reader should be careful to not overgeneralize.

Finally, due to our sampling methodology, our findings may only apply to video games. Further research into other types, like mobile and internet games, could expose other differences for those types.

6. IMPLICATIONS

Research. The results we present here imply that several strands of research could have a significant impact on games. The first is the study of testing with non-determinism, such as the CHESS tool [40], which reruns tests with every thread interleaving. Extending and scaling up this technique to other types of non-determinism exhibited in games remains an open research problem. Other areas of testing remain especially challenging in games, such as exploring the large state space. Other testing topics like how to make concrete yet flexible assertions in games remains an open problem. Finally, most testing literature looks for bugs in source code, but the results here suggests that significant and important bugs also arise from changes to game content. Early detection of build-breaking content changes is a fertile area for future research.

Game developers' significant reliance on in-house tools raises new areas of study for researchers. What functionality do these tools provide that off-the-shelf tools do not? How do these tools evolve over time? Who is responsible for maintaining these tools? Is there duplication of effort in building these tools across teams and companies? Do these tools have defects, and how are they tested? Indeed, in-house tools may face many of the same challenges that a company's main software product faces, yet we know of no existing work that systematically investigates in-house development tools, either in games or elsewhere.

Practice. We were interested to learn that software engineering practices are getting integrated into game teams, especially Agile processes. It appeared that a successful strategy to get management buy-in for new engineering practices is to communicate with management about engineering challenges, rather than isolating management. While high-overhead software engineering practices may yet be inappropriate for most types of video game development due to the high uncertainty, low-overhead practices such as pair programming or remote code review may be especially beneficial.

It appears that not only does game development have something to learn from non-game development, but vice-versa as well. Interviewees found that games provided high user satisfaction in part because of extensive focus on understanding user needs, rather than satisfying pre-defined requirements. More focus on the user in other types of software may be beneficial as well.

Education. Our results suggest that special skills, beyond those taught to most computer science students, would be beneficial for students thinking about moving into games. Chief among them is the ability to communicate with non-engineers. One interviewee even suggested that students would benefit from working in a non-engineering role, so that they could empathize with non-engineers. Creativity appeared to be an especially important non-technical skill that could be enhanced in students headed for game development careers. On the technical side, special focus on math and performance tuning appears to be especially useful for students.

7. CONCLUSIONS

Videos games make up a significant part of modern software development, yet software engineering researchers in the past have made little effort to empirically study games. Our results suggest that games have significant differences from "traditional" software development, and this paper contributes an empirical foundation on which to understand those differences. In a larger sense, this work represents a step towards understanding software development not as a homogenous whole, but instead as a rich tapestry of varying practices involving diverse people across diverse domains.

8. ACKNOWLEDGMENTS

The first author was a Visiting Researcher at Microsoft Research when this paper was written. Thanks to Thomas Stoffregen at the Affordance Perception-Action Laboratory for help designing the motion sickness survey, as well as Thomas Debeauvais and Gifford Cheung for help distributing it. Thanks to Andy Begel, Chris Lewis, Jeff Leiter, Eric Whitmire, and the entire Developer Liberation Front for their reviews. Thanks to the creators of Qualyzer for creating that software. Special thanks to interviewees and survey respondents for participating.

9. REFERENCES

[1] T. Marsh, "Serious games continuum: Between games for purpose and experiential environments for purpose," *Entertainment Computing,* vol. 59, no. 2, pp. 61-68, 2012.

[2] T. Connolly, E. A. Boyle, E. MacArthur, T. Hainey and J. M. Boyle, "A systematic literature review of empirical evidence on computer games and serious games," *Computers & Education,* vol. 59, no. 2, pp. 661-686, 2012.

[3] M. Nayak, "A look at the $66 billion video-games industry," *Reuters,* June 2013.

[4] M. Nagappan, T. Zimmermann and C. Bird, "Diversity in Software Engineering Research," *Proceedings of Foundations of Software Engineering,* 2013.

[5] H. Do, S. Elbaum and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Software Engineering,* vol. 10, no. 4, pp. 406-435, 2005.

[6] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe and H. a. N. J. Melton, "Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies," *Proceedings of the Asia Pacific Software Engineering Conference,* pp. 336-345, December 2010.

[7] T. A. Beery, "Gender bias in the diagnosis and treatment of coronary artery disease," *Heart & Lung: The Journal of Acute and Critical Care,* vol. 24, no. 6, pp. 427-435, 1995.

[8] E. Bethke, Game Development and Production, 2003.

[9] H. M. Chandler, The Game Production Handbook, 2008.

[10] M. McGuire and O. C. Jenkins, Creating Games: Mechanics, Content, and Technology, CRC Press, 2008.

[11] M. McShaffry and D. Graham, Game Coding Complete, Fourth Edition, 4 ed., Cengage Learning PTR, 2012.

[12] M. T. Wyman, Making Great Games: An Insider's Guide to Designing and Developing the World's Greatest Games, CRC Press, 2012.

[13] J. Blow, "Game Development: Harder Than You Think," *IEEE Software,* pp. 28-37, February 2004.

[14] A. Ampatzoglou and I. Stamelos, "Software engineering research for computer games: A systematic review," *Information and Software Technology,* vol. 52, no. 9, pp. 888-901, 2010.

[15] F. T. Tschang, "Videogames as Interactive Experiential Products and Their Manner of Development," *International Journal of Innovation Management,* vol. 9, no. 1, 2005.

[16] F. T. Tschang, "Balancing the Tensions Between Rationalization and Creativity in the Video Games Industry," *Organization Science,* vol. 18, pp. 989-1005, 01 Nov. 2007.

[17] F. Tschang and J. Szczypula, "Idea Creation, Constructivism and Evolution as Key Characteristics in the Videogame Artifact Design Process," *European Management Journal,* vol. 24, pp. 270--287, Aug. 2006.

[18] B. W. Boehm, "A spiral model of software development and enhancement," *Computer,* vol. 21, pp. 61-72, 1988.

[19] Y. Baba and F. Tschang, "Product Development in Japanese TV Game Software: The Case of An Innovative Game," *International Journal of Innovation Management,* 2001.

[20] T. Burger-Helmchen and P. Cohendet, "User Communities and Social Software in the Video Game Industry," *Long Range Planning,* vol. 44, pp. 317--343, oct 2011.

[21] A. Kultima and K. Alha, ""Hopefully Everything I'm Doing Has to Do with Innovation": Games Industry Professionals on Innovation in 2009," in *Games Innovations Conf.*, 2010.

[22] P. Stacy and J. Nandhakumar, "A temporal perspective of the computer game development process," *Information Systems Journal,* vol. 19, pp. 479-497, 2009.

[23] D. Callele, E. Neufeld and K. Schneider, "Requirements engineering and the creative process in the video game industry," in *13th IEEE International Conference on Requirements Engineering (RE'05)*, 2005.

[24] J. Kasurinen, J. P. Stranden and K. Smolander, "What do game developers expect from development and design tools?," in *Proceedings of the Conf. on Evaluation and Assessment in Software Engineering*, 2013.

[25] J. Musil, A. Schweda, D. Winkler and S. Biffl, "A Survey on a State of the Practice in Video Game Development," 2010.

[26] C. Lewis, J. Whitehead and N. Wardrip-Fruin, "What went wrong: a taxonomy of video game bugs," in *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, 2010.

[27] F. Petrillo, M. Pimenta, F. Trindade and C. Dietrich, "What went wrong: A survey of problems in game development," *Comput. Entertain.,* vol. 7, feb 2009.

[28] F. Petrillo, M. Pimenta, F. Trindade and C. Dietrich, "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.

[29] F. Petrillo and M. Pimenta, "Is Agility out there?: Agile practices in game development," in *Proceedings of the International Conference on Design of Communication*, 2010.

[30] C. Lewis and J. Whitehead, "The whats and the whys of games and software engineering," in *Intl. Workshop on Games and Software Engineering*, 2011.

[31] C. Kanode and H. Haddad, "Software Engineering Challenges in Game Development," in *Information Technology: New Generations*, 2009.

[32] A. Abran, J. W. Moore, P. Bourque, R. Dupuis and L. L. Tripp, Guide to the Software Engineering Body of Knowledge (SWEBOK), IEEE, 2004.

[33] S. E. Humphrey, J. D. Nahrgang and F. P. Morgeson, "Integrating motivational, social, and contextual work design features: a meta-analytic summary and theoretical extension of the work design literature.," *Journal of Applied Psychology,* vol. 92, p. 1332, 2007.

[34] H. Braverman, Labor and monopoly capital, 1975.

[35] R. Blauner, Alienation and Freedom, 1964.

[36] C. L. More, "Digital Games Distribution: The Presence of the Past and the Future of Obsolescence," *Media and Culture ,* vol. 12, no. 3, 2009.

[37] S. van der Graaf, "Get Organized At Work! A Look Inside the Game Design Process of Valve and Linden Lab," *Bulletin of Science, Technology, and Society,* vol. 32, no. 6, pp. 480-488, 2012.

[38] E. Murphy-Hill, T. Zimmermann and N. Nagappan, "Motion Sickness Susceptibility in Software Developers," Microsoft Research (MSR-TR-2014-24), 2014.

[39] Y. Benjamini and Y. Hochberg, "Controlling the false discovery rate: a practical and powerful approach to multiple testing," *Journal of the Royal Statistical Society. Series B (Methodological),* pp. 289--300, 1995.

[40] M. Musuvathi, S. Qadeer, T. Ball, G. Basler, P. A. Nainar and I. Neamtiu, "Finding and reproducing heisenbugs in concurrent programs," in *Proceedings of the Conference on Operating Systems Design and Implementation*, 2008.