Game Industry Problems: an Extensive Analysis on the Gray Literature

Cristiano Politowski\textsuperscript{a,*}, Fabio Petrillo\textsuperscript{b}, Gabriel C. Ullmann\textsuperscript{c} and Yann-Gaël Guéhéneuc\textsuperscript{a}

\textsuperscript{a}Concordia University, Montreal, Quebec, Canada
\textsuperscript{b}Université du Québec à Chicoutimi, Chicoutimi, Quebec, Canada
\textsuperscript{c}Universidade Regional do Noroeste do Estado do Rio Grande do Sul, Santa Rosa, Rio Grande do Sul, Brazil

1. Introduction

\begin{quote}
\textit{"The history of science, like the history of all human ideas, is a history of irresponsible dreams, of obstinacy, and of error"}
\end{quote}

Karl Popper

Context: As technology evolves, it offers improved videogame experiences that attract more and more players\textsuperscript{1}, therefore making the video-games the most profitable entertainment industry nowadays\textsuperscript{2}.

The game industry is known for its problems. They range from technical problems, i.e., 80% of the games on Steam require critical updates \cite{16}, to management problems, i.e., crunch time \cite{6} and unrealistic scopes \cite{18}. The problems in game industry also include mistreatment of employees\textsuperscript{3} and harassment\textsuperscript{4}. Yet, the game industry continues to make profits\textsuperscript{5} as players keep on buying its games, reinforcing a loop of bad practices.

These high-profile problems are probably only the proverbial tip of the iceberg. Yet, little is known of the problems faced day-to-day by game developers. Indeed, given its competitiveness, game studios are secretive and have a closed-source culture. Open-source games are rare and the main tools used by game developers, i.e., game engines, are proprietary (Unity and Unreal). Lessons learned by game developers are treated by game studios as a competitive advantage and, with rare exceptions in few game conferences\textsuperscript{6}, are not shared or discussed.

Yet, contrary to other software industries, game developers do share information about their games projects in the form of “war stories”. These war stories are postmortems, which are informal texts that summarise the developers’ experiences with their games projects, often written by managers or senior developers \cite{5} right after their games launched \cite{27}. They often include sections about “What went right” and “What went wrong” during the game development:

- “What went right” discusses the best practices adopted by the game developers, solutions, improvements, and project-management decisions that helped the project.
- “What went wrong” discusses difficulties, pitfalls, and
mistakes experienced by the development team in the project, both technical and managerial.

**Objective:** Game developers and the software-engineering research community would benefit from a state of the problems of video-game development, in particular the problems faced by game developers, their evolution in time, and their root causes. This state of the practice would allow researchers and practitioners to work towards solving these problems.

**Method:** We analyse 200 postmortems written between 1997 and 2019 available in our public dataset [22] of grey literature related to game development. These postmortems include 927 problems divided in 20 types. Through our analysis, we draw a landscape of game-industry problems in the past 23 years and how these problems evolved over the years. We give details on the most common problems, their root causes, and possible solutions. We also provide recommendations for future projects.

**Results:** For each of the 927 problems in the dataset, we identify its root causes and solutions. We show that:

- Based on the number of problems groups and types, the game industry suffer from management and production problems in the same proportion. However, production problems are concentrated mostly in technical and design problems while management problems are more spread across problems types.
- Based on the problems groups over the years, management problems decreased over the years giving space to business problems, while production problems remained constant;
- Based on the evolution of the problem types over the years:
  - Technical and game design problems are decreasing, the latter only since the last decade;
  - Problems related to teams increased over the last decade;
  - Marketing problems have the highest increase over the 23 years compared to other problem types;
- Finally, considering the problem sub-types, the majority of the main root causes are related to people, not technologies.

Furthermore, we provide recommendations for future projects to avoid/decrease the most common problems. Among these recommendations, video game developers could:

1. Consider “sharing the load” of complex tasks, avoid blindly hiring more workforce, and outsource if the budget is limited (Section 4.1);
2. Balance the expertise levels among developers and keep small teams with small scopes as well shield the developers to external interference (Section 4.2);
3. Promote their games more carefully, focusing on strong points of their games, and strengthen the relationship with players (Section 4.3);
4. Avoid relying solely on human expertise and invest in building a knowledge base about past projects to better estimate future tasks and projects (Section 4.4);
5. Keep a clear vision of the game design, avoid wasting time with static documents, spend more time in prototyping and playtesting (Section 4.5);
6. Allocate time to find the “fun” by polish the mechanics, art, and story of the games, as well as extensive playtesting to identify the weak points (Section 4.6);
7. Seek a better understanding of the platform architectures and limitations before committing to a project (Section 4.7);
8. Use KISS and Occam’s razor principles to keep the game design simple (Section 4.8);
9. Allocate time to experiment with multiple tools and game engines before choosing or building one (Section 4.9);
10. Consider changing the team structure if there are communication problems among developers (Section 4.10).

**Conclusions:** This analysis describes the problems that game developers faced during their game projects, some solutions, and some recommendations for future projects. It shows that many problems require project-specific solutions that are hard to generalise, while others do not have clearly defined solutions yet. Thus, we provide a state of the practice for researchers to understand and study video-game development problems. We also offer recommendations to help practitioners to avoid the most common problems in future projects.

The paper is structured as follows: Section 2 describes the dataset metadata used in this paper. Section 3 shows the overall analysis of the problems and its evolution over the years. Section 4 further discuss the top 10 most common root causes, outlining the developers solutions and our recommendations. Section 5 presents the related works and compare their findings with our data. Section 6 discusses additional details not covered in the previous sections. Section 7 describes the threats to validity. Section 8 concludes the paper with future work.

2. **Dataset**

In this Section, we summarize how we analyse the dataset and its metadata.

2.1. **Method**

Our analysis process is iterative, where the data from the postmortems keeps in constant evolution, allowing refactoring each new iteration. Figure 1 shows the process of collecting and compiling the data from the postmortems.

The method starts by randomly picking 20 postmortems (for each author) from the Gamasutra Website between the years 1997 to 2019. Each author reads the postmortems, focusing on the “What went wrong” sections and using the
coding technique from Grounded Theory [25]. During the reading, each author identifies all the problems reported by the postmortems, extracting quotes and grouping similar problems, based on previous literature definitions [18, 27].

In the next step, the authors discuss their findings, review any doubts, and ensure the correctness of the catalogue. Any change at this point, on the catalogue or on the set of problems, results in updates to both documents, which sometimes demands re-analysing postmortems. This iterative process continues until it reaches saturation: until no more new type of problems appears (at this point, the catalogue of problems is no longer updated).

To keep the distribution of the postmortems read by year balanced, some of them are manually chosen instead of randomly picked. Other postmortems have been replaced when they did not contain useful information regarding game development.

2.2. Dataset Metadata

The catalogue of problems is a document listing the problems found in the postmortems. First, it includes only the problem types gathered from the literature and then is updated during the postmortems analysis described in the previous sub-section.

We update the catalogue with every newly discovered problem type. Table 1 shows the final version of the catalogue where a problem type is an index to the problems’ descriptions (quote).

Then, we decrease the granularity of the problem types by clustering the types into four groups: Production describes practical problems that often happen during the production phase; Management - People describes management problems related to people; Management - Feature is about management problems related to the games features; and, Business is about the marketing and the strategy to generate revenue.

To store the problems gathered from postmortems, we define a data model. Figure 2 shows its UML class diagram. Each Postmortem relates to one Game as it is the nature of the document describe what happened in only one project. The Game has a collection of Problems. From increasing granularity, each Problem has a SubType, a Type, and a Group.

A Game also has a Platform [1-3] (PC, Console, Mobile), a Genre [1-12] (Action, Adventure, RPG, Simulation, Strategy, Puzzle, Sports, Platformer, Shooter, Racing, Roguelike, Running7), and a Mode [1-3] (Single-player, Multi-player, Online).

To further investigate the root causes of the problems, once the dataset was done we re-read the problems classifying them with a more specific description, which we called problem sub-types. For example, the Table 2 shows one dataset entry (one problem). The game “Baldur’s Gate II” from 2001 was analyzed and it has a problem type “Testing”, which belongs to the group “Production”. After the second analysis we defined the sub-type as “Scope too big to test properly”. The quote is an excerpt from the postmortem.

The dataset is available in an open repository on GitHub8 so that researchers and practitioners can access and contribute through pull requests. We choose this approach to curate contributions before inclusion. Contributors can also add to the catalog of problems and other metadata, like platforms, genres, and modes, to evolve the dataset overtime.

3. Overview of the 20 Problem Types

This Section shows the results of the dataset analysis. Section 3.1 describes the overall dataset results, the problem groups and types, and the problems in each platform (PC, Console, Mobile). Section 3.2 shows the evolution of problems over the years.

7This is a short list of the most popular games (https://en.wikipedia.org/wiki/Video_game.Genre).
8https://github.com/game-dev-database/postmortem-problems
Table 1
Catalogue with the video-game development problems identified through the postmortem analysis.

| Group        | Type                          | Description                                                                 |
|--------------|-----|-------------------------------|
| **Production** | Bugs | Bugs or failures that compromise the game development or its reception. |
|              | Game Design | Game design problems, like balancing the gameplay, creating fun mechanics, etc. |
|              | Documentation | Not documenting the code, artifacts or game plan. |
|              | Prototyping | Lack of or no prototyping phase nor validation of the gameplay/feature. |
|              | Technical | Problems with code or assets, infra-structure, network, hardware, etc. |
|              | Testing | Any problem regarding testing the game, like unit tests, playtesting, QA, etc. |
|              | Tools | Problems with tools like Game Engines, libraries, etc. |
| **Management People** | Communication | Problems communicating with any stakeholder, team, publisher, audience, etc. |
|                | Crunch Time | When developers continuously spent extra hours working in the project. |
|                | Delays | Problems regarding any delay in the project. |
|                | Team | Problems in setting up the team, lost of professionals during the development or outsourcing. |
| **Management Feature** | Cutting Features | Cutting features previously planned because some other factors like time or budget. |
|                | Feature Creep | Adding non-planned new features to the game during its production. |
|                | Multiple Projects | When there is more than one project being developed at the same time. |
|                | Budget | Lack of budget, funding, and any financial difficulties. |
|                | Planning | Problems involving planning and schedule, or lack of either. |
|                | Security | Problems regarding leaked assets or information about the project. |
|                | Scope | When the project has is too many features that end up impossible to implement it. |
| **Business** | Marketing | Problems regarding marketing and advertising. |
|              | Monetization | Problems with the process used to generate revenue from a video game product. |

Table 2
Example of one entry in the dataset.

| Column | Value |
|--------|-------|
| ID     | 61    |
| Title  | Baldurs Gate II – The Anatomy of a Sequel |
| Year   | 2001  |
| Source | http://bit.ly/2IDsVa0 |
| Name   | Baldur’s Gate II |
| Platform | PC |
| Genre  | RPG Strategy |
| Mode   | Multi Single |
| Group  | Production |
| Type   | Testing |
| SubType| Scope too big to test properly |
| Quote  | (...) We put a number of white-boards in the halls of the testing and design area and listed all of the quests on the boards. We then put an X next to each quest. We broke the designers and QA teams into paired subgroups - each pair (one tester and one designer) had the responsibility of thoroughly checking and fixing each quest. After they were certain the quest was bulletproof, its X was removed. It took about 2 weeks to clear the board (on the first pass). |

Figure 3: Number of problems related to each Group.

3.1. Overall Dataset Results
The dataset contains 200 video-game projects ranging from 1997 to 2019, describing 927 problems. On average, there are five problems by game title and 40 by year. Figure 3 shows the problems by groups: 46% of the problems relate to production, 45% to management, 9% to business.

Figure 4 shows the distribution of the problems by types. Game design, technical, and team problems are the most frequent, with 30% overall. Although problems are almost equally divided between management and production problems, while the two most common problems types, technical and game design with 11% each, are related to production group. Management problems are more spread between the problem types.

Figure 5 shows the problems by game platforms: PC, Console, and Mobile. The problems described in the post-mortems mainly occur in PC games, with 707 problems, followed by 432 Console problems, and 222 Mobile problems. Only 78 problems pertain to the three platforms (multi-platform games). Mobile games are more likely to be ported to Consoles and PC games to Consoles. None of the games were made only for Mobile and Console (no intersection between them without also being on PC).

3.2. Problems over the Years
Figure 6 shows the normalised number of problems per group, dividing by the total number of problems that year. For example, in 2018, there were five business problems among 16 problems. Production problems remain constant.
until today. Management problems peaked in 1998 and are less frequent now. While business problems increased over the years.

Figure 7 shows the four different patterns in the dataset. To normalise the numbers of problems each year, we divide their numbers by the total numbers of problems that year. The red line (curved line) is a second-degree polynomial function. The grey area represents the confidence interval (0.95 by default) of the function.

Figure 7a shows that Marketing problems increase over the years. Monetization and Bugs are also problems that follow this trend, but in a lesser degree.

4. Details of the 10 Top Problem Sub-Types

We further investigate the problems and identify the root causes of each problem type. We read all the problems again classifying the types into sub-types. We found a total of 105 different sub-types. Table 3 describes the top 10 most common sub-type problems. We focus on the top 10 sub-types for lack of space. Appendix A presents all the sub-types. In the following, each subsection discusses one problem sub-type. It provides an Explanation and a Solution to the problem sub-type, illustrated by excerpts from post-mortems. It also provides Recommendations for the future projects to prevent or decrease the occurrence of this problem sub-type.

- Explanation: Description of the problem reported by the developers.
- Solutions: Solutions reported by the developers.
- Recommendations: Our recommendations for the future projects based on the dataset and the literature.
Figure 7: Four common patterns of the importance of the problems over the years. (a) Shows Marketing problem that increased since 1997. (b) Shows Technical problems that decreased since 1997. (c) Shows Game Design problems that decreased in the last decade. (d) Shows Team problems that increased in the last decade.

Table 3
The top 10 most common sub-type problems.

| Type          | SubType (root cause)              | N  |
|---------------|-----------------------------------|----|
| Team          | Insufficient workforce            | 49 |
| Team          | Environment problems              | 48 |
| Marketing     | Wrong marketing strategy          | 35 |
| Planning      | Underestimation                   | 34 |
| Game Design   | Unclear game design vision        | 28 |
| Game Design   | Lack of fun                       | 27 |
| Technical     | Platform and technology constraints| 24 |
| Game Design   | Game design complexity            | 23 |
| Tools         | Inadequate or lack of tools       | 22 |
| Communication | Misaligned teams                  | 22 |

4.1. Insufficient workforce

Explanation: Insufficient workforce is the main problem of teams. It happens when a game company does not have enough developers for all the tasks or when a developer has too many tasks. This problem happens often when there is a short budget for the game project. Other causes include a lack of planning (tasks, schedules, testing, etc), the difficulty to find developers with certain skill sets, experience, and willingness to work on a game project.

“For the first six months of production, one person was juggling design, project management, and a number of significant project-external responsibilities. They were – obviously – over-tasked. It led to a lack of communication on scheduling between studio management and the development team.” – P#604

Solutions: “Hire more, share the load” is the most frequent advice given by developers. They also mention outsourcing and remote work as solutions to the budget constraints. Other alternatives to mitigate this problem include dividing the tasks among more people to improve efficiency, calling for help and staffing up sooner when needed. Moreover, be sure to dismiss the outsourced professionals when his job is 100% complete and integrated in the game.

Recommendations: We observe that Insufficient workforce is mainly caused by poor management of the project Scope (requirements), which leads to other problems like Cutting Features and Crunch Time. How to mitigate this problem sub-type depends on the game company. Game developers
report that pair programming [13] and code reviews [1] are not common in game industry while they are well established practices in traditional software development. Given that game development is a multi-disciplinary endeavour, similar practices must be adapted to the game industry, for example, to accommodate artists.

Less common than insufficient workforce is overstaffing, which happens when a game company assigns too many developers to a given task, making communication and organisation difficult. Overstaffing is also the result of poor estimations, often when too many managers and stakeholders interfere with one another. Despite the developers’ complaints about staff shortage, game companies should follow Brooks’ Law [4].

“When there’s so much work to do, one of the first reactions is to throw more hands into the mix thinking that this will lighten the load. (...) Rather, we found that, an increasing the number of people on the team to aid the workload inhibited the project, and placed great strain on the lines of communication.” – P#518

We observe that Insufficient Workforce is the main root cause among the problems related to Team. It is linked to project scope and leads to other problems, like cutting-features and crunch-time. Developers should consider “sharing the load” of complex tasks, avoid hiring more workforce (e.g., Brooks’ Law), and outsource if the budget is limited.

4.2. Environment problems

Explanation: Even a properly staffed and well experienced team may suffer if their corporate environment has problems. Environments, especially in large studios, are a source of problems when, for example, there is a lack of a departmental organisation or hierarchy. Low wages, lack of incentives, toxic behaviours (e.g., harassment or bullying), excessive or mandatory crunch times, lack of open communication, lack of working standards are all environmental problems.

“We would’ve been better off had we realized that personality fit and talent aren’t enough: people need to mesh with your working style. When people are unhappy, it spreads through the whole team.” – P#467

Solutions: Avoiding environmental problems in game studios requires balancing the developers’ experience and having experienced developers mentor junior developers. It also includes keeping the team cohesive while decentralising decisions and putting the “right” developer on the “right” task. Moreover, it also includes having small sub-teams with a smaller scope. It involves also supporting the team, with material and psychological resources and keeping the moral high. Finally, it helps to have fixed but spaced meeting dates, especially when following a well defined process with a clear hierarchy.

4.3. Wrong marketing strategy

Explanation: Developers may overlook the importance of marketing because they focus on finishing their games. Developers commonly - and often wrongly - try reaching a broad audience by giving away copies of their games to specialized media and demoing their games in game conferences. However, according to the developers, this strategy is not effective.

Related problems include targeting the “wrong” players, miscommunicating with the players, promoting the game too much, and losing marketing opportunities (e.g: major sales holidays such as Christmas). Developers also reported specific marketing problems in crowdfunding campaigns and in early-access programs.

“The launch week I started looking at [YouTube] streamers for the PC version, so I basically searched

Dealing with environmental problems means managing people, similar to any tech company. Therefore, common sense dictates following best practices, like balancing the expertise levels among developers and keeping small teams with small scopes. Game development, however, has some specific characteristics, like crunch times and the mistreatment of certain types of employees. Solving these problems requires shielding developers and allowing them to perform their job properly.
for big youtubers that covered games like: Spelunky, Meat boy, and a few other more recent pixel-art indie games that fit the same category as [the game]. I mailed all of them, close to a 100, with at least one steam-key included (...) and this all resulted in an awesome 0 [streams]. I did a follow up email to a large portion of them a week later, and this resulted in 1 Streamer playing it, yay results!” – P#430

**Solutions:** “Don’t announce until you’re much closer to release” is the most common developers’ advice. Game studios often announce their games years in advance in hope to “build the hype” and get noticed. Developers also recommend working with the players, invest time producing marketing material, and, if possible, producing a demo of the game to create awareness and build a community. They also warn not to oversell games and focus on the game development, in particular if the game studio does not have marketing expertise. They also suggest launching games in more than one store/marketplace, producing the launch trailers, and focusing the marketing on the games strengths. Finally, they recommend distributing a strictly limited and continuously monitored amount of game copies to reviewers to avoid piracy problems.

**Recommendations:** Marketing is the problem sub-type that increased the most over the years. We observe that the main causes of this problem are threefold: new audience (therefore a need for new marketing strategies), lack of expertise in promoting the game (especially in indie game companies), and saturation of the game market (need to stand out).

The way to communicate and expose developers’ games to players evolved from magazines in 1997, through forums, social media, online stores, to current streamers and independent reviewers. Twitch, the most popular streaming platform, alone has 1.645 billion hours watched per month. The most popular independent youtuber has millions of views every day. Large outlets, like Polygon and IGN, still have an important role to play but streamers might increase sales dramatically.

Yet, taking advantage of new media platforms is difficult. Indie developers, with low marketing budget, often fail by trying to reach as many “influencers” as they can. This lack of expertise in marketing and the saturation of the game market make it difficult to be noticed.

Problems in marketing are increasing as game developers fail to promote their games using strategies suitable to today’s standards, showing their lack of expertise. The game market demands that developers have a more transparent relationship with the players, similar to what streamers do. Developers should create awareness, focusing on the strong points of their games, strengthen the relationship with the players (using alpha/beta testing, answering constructive feedback, creating development logs), and let professionals do the marketing, outsourcing if needed.

4.4. Underestimation

**Explanation:** The majority of the planning problems are due to optimistic estimation: typically due to (1) tasks that developers thought easier and faster to complete and (2) the time needed to create the game assets (like 3D models and music).

“Back in the days when we crafted our first budget and milestone plan we had the development of [the game] ironed out to five full-time developers working for six months. Fact: [the game] took eight full-time and between two and four part-time developers 24 months to barely finish. Our initial estimate was off by more than 700 percent.” – P#417

**Solutions:** Developers believe that they must allocate time to do “everything correctly” to achieve a more “solid game”. Thus, goals and deadlines must be defined early and, if needed, re-defined often during the production. Also, they must spend more time assessing risks during pre-production, allocating more time for every details of the game, not letting anything as “afterthought”.

**Recommendations:** Hofstadter’s Law states that “It always takes longer than you expect, even when you take into account Hofstadter’s Law” and holds true for game development. Software estimation is a well studied field of software engineering. It uses previous data to estimate the effort and cost required for a project. It uses different methods, e.g., COSMIC or Agile Story Points, and techniques, e.g., the analysis of experts in the field, parametric models or machine learning. Yet, software estimation in game development is often performed manually, using the experience from senior developers. Developers, specially seniors, aside of using their previous experiences to judge future projects, should also document it for future use, for example, creating ML models.

Beside estimating the technical aspects of the project, e.g., game engine or infrastructure, similar to traditional software development, game developers must also estimate the effort of producing the art assets and integrating the assets with the rest of the game. In this case, parametric estimations might not be suitable.

However, estimation, like with any other software projects, varies across game projects. Teams move from one game genre to another, must adapt to technological advances, etc., which make estimation more difficult. When these situations happen, team must invest in a longer pre-production cover-
ing the research of a new technology, tools, and game design suitability.

The closed nature of the game industry prevents the sharing of data about game projects. Postmortems are an important source of information but they are not enough [20]. Game developers must gather metadata about past game projects and extend traditional estimation methods. For example, although the COSMIC method is “technology independent” [23], it must be adapted to game projects and their particularities, like art assets.

4.5. Unclear game design vision  
**Explanation:** Teams often face difficulties in specifying the core mechanics of a game. They normally write a Game Design Document (GDD), defining the project and its scope, during the pre-production phase. This document is also used to divide tasks and define the artistic designs of games. However, writing such document is difficult and requires game development expertise. This document is also rarely updated during the projects’ life and the game design visions change regardless of the definitions contained within it. Unclear game design vision is also caused by the absence of a clear playtesting process and by problems in the team, e.g., poor division of tasks, lack of brainstorming. The divergence of creative views between game designers and a publisher is also a common problem.

“Although all of the changes we made along the way made for a better final product, the ever-changing design definitely added to development time and made it more difficult to balance the gameplay experience.” – P#381

**Solutions:** A solution to achieve a clearer game design vision is to spend more time on pre-production and prototyping. It also includes investing more time playtesting the game. Also, game developers should follow traditional software-engineering processes, in particular enforcing “feature lockdown dates” to stop new additions to the game, do less review cycles to avoid staggering the workflow, and define the set of tools before going into production.

**Recommendations:** An unclear game design vision impacts the whole game project, including management and testing. Although related to game design and art, the game design vision must be embraced by the whole team and, thus, is also a management problem. In traditional software engineering, methods and techniques exist to abstract the systems and languages, e.g., UML, and to ease the communication among developers. Similar methods and techniques should be used/devised to express game designs and their impact on game development [15].

Teams must understand the project vision to avoid wasted work. Recommendations are to spend less time defining static documents that will be neglected if not updated and spend more time in pre-production until the core mechanics and the fun factor is clear. They involve more prototyping and playtesting. Finally, do not renounce the creativity control over the project.

4.6. Lack of fun  
**Explanation:** Game development generally includes iterations to find and refine the game “fun factor”. In large game projects, the core concepts of the game, including its “fun”, are established during pre-production phase, while in indie games, these are defined during development. Yet still, during production, a game may prove less fun than expected in pre-production. A game that is not fun is a software without purpose. Developers must then add new features or change existing ones during development to increase the “fun” factor, which leads to wasted work and delays. The causes of a lack of fun vary as the games also vary in their premises. The most common ones are weak mechanics, ugly art, unrealistic or unappealing story, and lack of tutorial for new players.

“[The game] really is a simple game, and in some respects, it’s too simple. There’s no character progression, no levels, and no real incentive for the player to keep coming back.” – P#123

**Solutions:** Developers recommend three steps to prevent/overcome this problem. First, they recommend more playtesting sessions to identify the weak points of the game and to survey (early adopter) players about the game. Second, they suggest spending more time balancing the game and polishing the players’ experience, for example, by reducing the players’ frustrations. Third, they advise investing in tutorials to help players. Developers also advocate investing more time in prototyping during pre-production.

**Recommendations:** Lack of fun is essentially a game design problem, out of the scope of software engineering. Research about “fun” in games are related to the study of human cognition, but also linked to the human capacity of learn new things [12].

Based on the suggestions from the postmortems, developers should allocate time to polish the mechanics, art, and story of their games. Also, they should use extensive playtesting (and surveys) to identify weak points in their games.

4.7. Platform and technology constraints  
**Explanation:** Game developers often face problems with platform and technology constraints. They must contend with different consoles, mobiles and variety of hardware/software in PCs. They must consequently write code dedicated to
manage, for example, the memory allocation, graphical details, and load times. They must account for old technology slowing down development and difficulties when creating a multiplayer experience.

Developers also routinely face problems with memory, especially when working with low-level code, “closer” to the hardware. They often do not consider platform constraints when designing their games, adding content regardless of these constraints, yielding long build times, long starting times, and games that may not run at all on certain platforms.

“Developing a PC game is very different from a console game, particularly in terms of memory management, loads and saves.” – P#358

“We knew [the open sandbox world] would be an issue, but we underestimated the painful impact this would have on total memory usage and we planned poorly from the start.” – P#441

“It took about five minutes to load a single level on a developers station. Therefore, it took about five minutes to test the smallest change.” – P#106

**Solutions:** Developers have implemented different solutions to overcome these technical problems. These solutions often cannot be generalised to different games. Yet, a general solution is taking time to study the target platforms’ architecture, working closely with the platform developers if possible. Developers also recommend implementing the core mechanics first and prioritising its related game features.

**Recommendations:** Platform constraints are often defined by the “low end” specifications of consoles, mobiles, and PCs. These constraints include slow read-and-seek times on hard drives, CPUs with low clock speed/number of cores, and obsolete graphic cards. For example, consoles are built to sell en masse and must remain cheap, no matter their generation, and thus often include “outdated” technologies. However, developing for consoles has benefits: developers do not need to handle a large variety of hardware configurations and can deliver the same gameplay experience to all players.

Platform constraints also arise from the differences between consoles, mobiles, and PCs. Developers must learn how to deal with each platform in particular while developing for multiple platforms adds complexity to the game project. The gaming market is scattered across different platforms and developers must publish on different platforms to reach more players and sell more games. Yet, even with multi-platform game engines, they must deal with each platform constraints and must degrade their games to accommodate the lowest common denominator, in particular in relation to frames per second.

Game developers must assess the viability of their game design wrt. the technical specifications of the targeted platforms. They must include strategies to gracefully degrade their games on lower-end devices or progressively enhance them on more capable platforms. They must also have a detailed understanding of the platform architectures and limitations and must allocate time for experimentation.

4.8. **Game design complexity**

**Explanation:** Games are complex products and often developers struggle with the game design complexity. Game design complexity stems from the scope of the features: ambitious features are abandoned before the project even starts, by lack of resources. Even when a scope is reasonable, a large numbers of features makes it difficult to follow the initial vision. Tight deadlines and parallel projects also damage the game design.

“All of us had extremely high expectations for the game, but the total feature set turned out to be unrealistic given our small development staff and fixed schedule.” – P#58

**Solutions:** Developers advise simplifying the game design: visual style, scenario, and even the game achievements19. They also suggest planning carefully the game levels, instead of rushing into production. They recommend using better tools for the tasks and pay attention to not misuse the camera.

**Recommendations:** Game design is difficult and there are not clear process to define it for a given game. We observe that balancing the game experience requires: (1) a clear game design vision, something on which all developers should agree; (2) understanding of the players’ needs and expectations; (3) constant playtesting sessions to verify the game state.

As developers suggest keeping the design simple, we recommend keeping in mind the KISS principle and the Occam’s razor when designing the game. Also paying attention to players needs and extensive playtesting.

4.9. **Inadequate or lack of tools**

**Explanation:** Tools rarely offer all the features needed by developers to build their games. Developers report three main problems with tools: (1) inadequate or buggy game

---

18This trend seems to come to an end with new XBox series X and PS5 consoles with SSDs and GPUs.

19In video gaming, an achievement is a meta-goal defined outside a game’s parameters.
engines, (2) tools not fulfilling the special requirements of games, and (3) expensive tools that cannot be purchased for lack of financial resources.

“Your game engine shapes your entire development and limits what you can and can’t do. In this matter, I chose poorly.” – P#898

**Solutions:** Developers advise evaluating the tools to be used during production during pre-production, since any later change become financially costly. They suggest carefully assessing whether working with their own tools or purchasing a third-party software is more adequate, and this decision depends on the type of game project.

**Recommendations:** Tools often frustrate developers, in particular game engines. For example, in two game projects, EA\(^{20}\) forced their developers to use their proprietary Frostbite engine, causing delays and reworks for the game “Dragon Age 3” \(^{24}\) and a failed “Anthem”\(^{21}\).

Game engines can speed up game development but also limits game designers and developers. Game engines are few and include Unity, Unreal, and proprietary, closed-source engines, built by large game studios. Although open source, Godot\(^{22}\) is not yet as mature as its proprietary counterparts \(^{21}\). Developers may consider building their own game engine but should carefully assess the benefits and the risks in doing so.

**Solutions:** Developers advise evaluating the tools to be used during production during pre-production, since any later change become financially costly. They suggest carefully assessing whether working with their own tools or purchasing a third-party software is more adequate, and this decision depends on the type of game project.

**Recommendations:** Tools often frustrate developers, in particular game engines. For example, in two game projects, EA\(^{20}\) forced their developers to use their proprietary Frostbite engine, causing delays and reworks for the game “Dragon Age 3” \(^{24}\) and a failed “Anthem”\(^{21}\).

Game engines can speed up game development but also limits game designers and developers. Game engines are few and include Unity, Unreal, and proprietary, closed-source engines, built by large game studios. Although open source, Godot\(^{22}\) is not yet as mature as its proprietary counterparts \(^{21}\). Developers may consider building their own game engine but should carefully assess the benefits and the risks in doing so.

**4.10. Misaligned teams**

**Explanation:** In large game companies, where many different teams work on a same game, people may establish different/diverging visions on the game design and development. In small game companies, misalignment may happen when developers cannot reach an agreement regarding game design or development choices, due to lack of dialogue or conflicting personalities.

“(…) my style of development was in conflict with what they wanted to do. I tended to be more...

\(^{20}\)Electronic Arts is a publisher and owner of many video game studios ([https://www.ea.com/en-ca](https://www.ea.com/en-ca)).

\(^{21}\)https://bit.ly/39fTnTt

\(^{22}\)https://godotengine.org/

In large game studios, managers have the difficult task to keep the different teams aligned with the game design vision. Instead of organizing teams per roles (design, art, development, test, etc.), they should consider mixed, independent teams, that can design, implement, test, and integrate a new feature into the game. The design process must change to allow the creation of modular games.

**5. Related Work**

We now summarise academic work related to problems in game development. We discuss each work individually and discuss them together and compare them to our own work in Section 5.7.

**5.1. Callele et al., 2005**

Callele et al. \(^{[5]}\) analysed 50 postmortems from the Game Developer Magazine, written between 1999 and June 2004, and investigated how requirements engineering was applied and evolved in game development. They grouped “What went right” and “What went wrong” into the five categories: (1) **Pre-production**, problems outside of the traditional software development process; (2) **Internal**, problems related to project management and personnel; (3) **External**, problems outside of the development team’s control; (4) **Technology**,
problems with the creation or adoption of new technologies; and, (5) Scheduling, problems related to time estimates and overruns.

They reported that internal problems are 300% more prevalent than that in others categories. Most internal problems relate to project management: missing tasks and wrong estimations of the tasks.

“Project management issues are the greatest contributors to success or failure in video game development. In the case of failure, many of these issues can be traced back to inadequate requirements engineering during the transition from pre-production to production.” – Callele et al. [5]

We also found in our dataset that management problems form a large percentage of the problems, even more so if we also consider business problems as management problems. However, we also observed that another large percentage of problems happen during production and include game design problems, technical problems, and problems with tools.

The authors also reported that management problems are due to the transition between pre-production and production. In game development, developers usually use pre-production to validate game concepts through prototypes and plan features and schedule. Similarly, we found in our dataset that planning, feature creep, and, to a less degree, delays, are recurring problems during game development. However, we observed that feature creep decreased over the years.

5.2. Petrillo et al., 2009

Petrillo et al. [18] analysed 20 postmortems published on the Gamasutra Website to identify recurring problems and compare them with traditional software-engineering problems. They concluded that (1) video-game development suffers mostly from management problems rather than technical problems; (2) problems in video-game development are also found in traditional software development; and, (3) the most common problems are related to Scope, Feature Creep, and Cutting Features.

They also reported that multidisciplinary teams in large game studios are also a source of problems:

“The team in traditional software engineering is usually relatively homogeneous. However, the electronic games industry, because it is multidisciplinary, attracts people with a variety of profiles such as plastic artists, musicians, scriptwriters, and software engineers.” – Petrillo et al. [18]

In our dataset, we also identified many problems related to teams, including in indie studios with few developers. In particular, we reported that problems related to teams and communication remain constant over the years. Thus, we confirm the previous observations reported by Petrillo et al.

These authors also reported that requirements engineering being different than traditional software as for is hard to define the fun factor of the game.

“Another important difference is that elaborating game requirements is much more complex, since efficient methods to determine subjective elements such as “fun” do not exist.” – Petrillo et al. [18]

Similarly, we observed in our dataset that, during production, developers add new features to their games, in an effort to come up with a better game, but against their prior requirement analyses. Although developers set the game mechanics (features) during pre-production, they often change/add new features during production, in particular to increase “fun”.

5.3. Kanode and Haddad, 2009

Kanode and Haddad [10] used postmortems to discuss the challenges of adapting traditional software engineering to video-game development. They reported differences between game development and traditional development, which we summarise in Table 4.

Asset Diversity: We identified many problems with assets among the technical problems. These problems seem mostly due to the large numbers of assets, not their diversities. Developers reported problems managing assets and performance problems, e.g., long load times.

Project Scope: Problems with scoping the games are recurring. Developers define too large scopes and time constraints force them to cut features. Budget also leads to changed scope. Feature creep is still a problem but decreases over the years.

Game Publishing: Problems with game publishing also appear in our dataset. Developers have difficulties with one another and publishers, especially indie studios with little experience. However, publishers are important to the successes of games, even if politics and creative interventions hurt game development. The relationship between developers and publishers deserves more research, out of the scope of this work.

Project Management & Team Organization: We also observed many problems related to project management and teams, with the exception of indie game studios, in which one developer perform more than one function.

Development Process: The development phase is usually split into pre-production and production. The testing phase vary depending on the game genre. For example, a multi-player, service-based game like Dota 2 will have a different testing process than a single player game like “Hollow Knight”. In our dataset, we observed many postmortems stating that pre-production was skipped, with dire consequences during production.

Third-Party Technology: Third-party tools help new developers write games. Game engines, for example, were a major contributor to the surge of indie game studios. How-
ever, problem with tools exist over the years, regardless of how advanced they are.

5.4. Lewis and Whitehead, 2011
Lewis and Whitehead [15] used two previous papers [2, 26] to identify problems in game development and whether/why these could be of interest to software-engineering researchers. They highlighted some areas to explore further and differences between games and traditional software.

They reported that, for large game studies, teams in game development are multidisciplinary and tightly coupled and that they suffer from tight budgets and deadlines. They also wrote that larger teams require strong leadership due to constant developers’ turnover. We found similar problems in our dataset. We observed different problems for smaller or indie game studios: for example, small studios do not have budget to build new game engines, which constrains their workflows.

Regarding tools and environments, the authors reported a lack of quality tools. We also identified many problems related to tools. They based their report on the lack of tools to handle the complexity of game development. In our dataset, we also found discussions about tools, in particular game engines, which, although not without flaws, may ease game development.

The authors discussed the lack of design patterns for game development. The information from postmortems does not gave us this level of granularity but showed problems with game design and technical aspects of the games.

The authors referred to game engines as middleware that facilitate game development. They also stated that engines often need rewriting to provide the features needed by the developers. In our dataset, we also noticed that developers struggle to implement features because of games engines or some other technological choices.

The authors classified games as emergent software, for which we cannot predict the outcome. They mentioned that game studios prefer to hire dozens of human testers instead of using unit tests. We concur with these statements: in our dataset, only one problem is related to unit testing. All other testing problems refer to playtesting sessions with players.

“(…) digital game designers have tried to design a game upfront through copious amounts of documentation, but that the documentation is made instantly obsolete by surprises that arise when actually implementing the game.” – Lewis and Whitehead [15]

Finally, the authors states that documenting a game upfront is pointless as new features are added regularly, rendering any documentation obsolete. We observed only 2% of problems related to documentation in our dataset. Some developers stated the need for a clear vision, but not exactly Game Design Documents. Developers are more concern about a clear game design vision rather than common documentation.

5.5. Washburn et al., 2016
Washburn et al. [27] analysed 155 postmortems written over 16 years and identified some characteristics and pitfalls of game development, and suggested good practices. They divided the problems and practices into five categories (Product, Development, Resources, Customer Facing, and Other) and 21 sub-categories. They discussed four of the most common problems, as shown in Table 5, adapted from Washburn et al. [27].

The authors reported that the most common problem relates to Teams, similarly to our observations, in which Teams problems are the third most common problems (8%), including lack of communication and disagreement among developers.

| Challenge                  | Description                                                                 | SE Practices                                                                 |
|---------------------------|-----------------------------------------------------------------------------|------------------------------------------------------------------------------|
| Diverse Assets            | Increasing complexity, diversity and size of art assets.                    | Optimize tools and pipeline for integrating assets into the game.             |
| Project Scope             | Poorly established project scope further compounded by feature creep.        | Keep project scope realistic and consider time for game exploration and feature creep. |
| Game Publishing           | Bring a video game to market involves a game development company convincing a game publisher to back them financially. | Better communication with the publisher, keeping requirements clear and inform of project progress. |
| Project Management        | The management of a game development project involves the oversight of multidisciplinary teams. | Invest in managerial training with an emphasis on project management practices. |
| Team Organization         | Teams are segregated by specialty (programming, design, etc) or with functional units (combination of expertises). | Encourage an attitude of the team as a whole and less importance on individuals. |
| Development Process       | The over-arching phases of game development are pre-production, production, and testing. | Understand current process and the problems with it. Identify processes that will benefit the project. |
| Third-Party Technology    | Due to costs, complexity, and higher consumer expectations, game developers are using more components from third parties. | Apply risk management to selection of third-party technology in order to identify which components would work best. |
Main findings of Washburn et al. [27].

| Category   | Description                                        | %    | Details                                                                 | Takeaway                                                                 |
|------------|-----------------------------------------------------|------|-------------------------------------------------------------------------|-------------------------------------------------------------------------|
| Game Design| Good or bad design decisions that impacted the quality of their game. | 22%  | Overly ambitious game designs which could not be implemented and concepts that confused the player. | Keep implementation in mind while creating a design, and create contingencies if it cannot be done. Test key game concepts should before release (audience reception). |
| Dev. Process| The process teams use while developing affects the quality of the product. | 24%  | Developer did not planed before the development and also mismanagement. | To avoid conflicts during the development process, teams need to have proper management and invest time upfront planning before beginning development. |
| Obstacles  | Obstacles are more likely to have a negative impact on a team | 37%  | Lack of team dynamic and unfamiliarity among the team. | Developers should participate in team building. Subscribe to a method of risk management, because they are more likely to face obstacles than more seasoned teams. |
| Schedule   | Missed milestones or delivered them late             | 25%  | Problems in estimation, optimistic scheduling, and design changes late in development. | To avoid schedule slippage, developers need to spend more time to plan out all the work that needs to be done so that no tasks are overlooked when giving estimates. |

They also reported that scheduling and process are recurring problems, which we also support with our findings from the dataset: underestimation, management as the main causes of planning problems.

The authors cited ambitious scope and confusing concepts as examples of game design problems, which we also support via our findings, although we found a more diverse set of game design problems.

### 5.6. Edholm et al., 2017

The authors conducted interviews with staff from four different game studios and 78 postmortems to investigate the culture of crunch-time in game industry. According to their interviewees, crunch is common within the game industry as the majority of game studios applied such practice.

From their postmortem data, 45% mentioned crunch-time. Also, crunch-time has been within game industry from early 2000 to current date (2014). Moreover, small studios are more prone to crunch (54% crunch) than both micro-(33%) and medium-sized (36%) studios. Our data show the first signs of crunch-time in 1998, but it is decreasing after 2015, with zero occurrences in 2018 and 2019 (Figure 8).

“(...) well-being of the product is prioritized over employee welfare. Since people have a personal investment in the product they create, they blame themselves if it ends up badly.” – Edholm et al. [6]

The authors also investigate other problems. They found that the most common ones are Planning/scheduling and Technical while Publisher disagreement or pressure and Unfun game are rarely mentioned. Our data show similar results. Planning is the fifth more common problem and technical is the first one. However, we found many problems related to publisher, specially in communication, planning, and marketing. Also, the lack of fun is one of the most common root causes (Table 3) in our dataset.

### 5.7. Summary

These previous works used postmortems to discuss videogame development problems. Each of the analysed papers used an ad-hoc classification for the problems, even though most of them converge in at least some aspect. Management is found as the main problem of game development by all these works and ours, possibly because the source of information are postmortems, in which senior developers are more willing to discuss “general” problems rather than technical minutiae.

When comparing their findings with our data, we see that we agree with some exceptions. Previous authors did not consider indie game studios or the importance of game engines. Nonetheless, even with the evolution of the technology and two generations of consoles later, the game industry still deal with the similar problems.

### 6. Discussion

#### 6.1. Problems Evolution

Production problems remain constant until today. The most clear spike in the data occurred in 2005, which might be related to the arrival of a new console generation that year:
the seventh generation, e.g., Sony Playstation 3, was released between 2005 and 2006. The Sony Playstation 3, with its new architecture, was notoriously difficult to program and Sony kept performance secrets only for their first-party studios.

Management problems peaked in 1998 and are less frequent now. One factor that helped decreasing management problems might be the adoption of agile methods. The game industry, even today, works with old development methods, e.g., Waterfall [19], yet agile methods, born around the 2000s, are being adopted gradually. Pre-production and Production may not be both amenable to agile methods. However, the concrete development of the game, during production, could be benefit from using agile methods. We discuss further development methods in Section 6.2.

The problems with Business increased over the years. Our hypothesis is that the rise of Indie developers, in particular the “one-man-army” teams in which one developer does all the tasks, contributed to increasing this problem. Indie developers do not have publishers or colleagues to deal with marketing and perform related tasks poorly. Their business knowledge is often limited.

6.2. Development Processes

The game industry still mostly uses waterfall-like development processes [19]. Their processes usually divide into: (1) Pre-production to prototype, find the game’s core “fun” mechanic, create the Game Design Document (GDD), etc. This step is normally done with a small team of developers. (2) Production to implement the game with the full team. The implementation process varies according to the team. (3) Post-production to work on updates and bug fixes. The traditional iterative process may be adequate for game projects. However, the constant evaluation of the team productivity will aid the team to keep track of progress and help in the decision to hire more developers.

6.3. Game Modularity

To help Cutting Features and respect the deadlines, a solution is to design modular games, estimating the production of the main mechanics, somewhat similar to what a Minimal Viable Product (MVP) approach [14], while keeping the level of visual quality expected by players. The traditional MVP concept might not fit well game development because of the game industry peculiarities [8] and of the game publishing strategy, like games that rely on their players’ first impressions.

6.4. Developers Turnover

“The game industry is cyclical, constantly churning employees in and out depending on the needs of a project” [27]. It is common for game developers to change companies for each new project. Therefore, teams are also constantly changing, having to adapt to newcomers. This turnover happens during all the development phases, because game projects are long projects in time. Even with a clear defined process for newcomers, with mentoring from senior developers, their productivity will be low at first, yetmicromanagement must be avoided.

6.5. Game Testing

Given the importance and emphasis given to software testing in traditional software development, we were surprised to find little information about testing in game projects in the postmortems. One hypothesis could be that testing is largely successful and therefore does not need mentioning in the postmortems. However, it is well known that games often suffer from low quality and that game projects often overrun their schedules, hence hinting that testing is probably problematic. Therefore, our next hypothesis is that testing, in particular software-engineering testing, is not or little performed by game developers. Indeed, postmortems mention playtesting but do not mention unit testing or integration testing. This lack of mention is interesting and calls for more research on game developers’ testing habits (or lack thereof) and the reasons for these habits.

7. Threats to Validity

Dataset based only on postmortems: Our results are based on postmortems, which do not represent all the games or the whole game industry. Nonetheless, postmortems are the best (and only) source of information to which we have access, i.e., publicly available.

Dataset has only successful projects: All the postmortems were gathered from the Gamasutra Web site. They pertained to 200 game projects that were released and, for most of them, profitable. Therefore, they did not include failed game projects. The lack of failed game projects may lead to optimistic results in comparison to the reality [18]. Yet, they all reported problems, which we identified and analysed in this paper.

Developers might not tell the whole history: As shown by Washburn et al. [27], some authors of postmortems may not disclose all that happened during their game projects. Thus, postmortems do not represent entirely the reality. Yet, they provide list of meaningful problems to which (1) game studios should pay attention in their own, next projects and (2) researchers should investigate to find solutions.

23https://venturebeat.com/2014/07/06/last-gen-development/
24https://net.co3haIFN
25Some examples of (successful) games written by only one developer are “Stardew Valley” and “Dust: An Elysian Tail”.
26For example, single player games like “The Last of Us” have longer development cycle to produce the most realistic experience possible, similar to a movie. Therefore, they receive few updates after their launch in opposition to games-as-a-service, like “Fortnite”, which receive regular updates and new features.
27https://bit.ly/2WmLWET
28The director of Final Fantasy 14 had to micromanage the team to keep the production pace, but he advises not to do it in https://youtu.be/XsbyQK17yw4.
**Problems are too abstract:** For lack of space and difficulty to convey the whole context, the problems described by the authors of the postmortems are abstract, often without technical details. We identified and formalised the problems from free texts, sometimes written by designers or managers unaware of the technicalities faced by game developers. Yet, the diversity of authors of the postmortems is valuable and reduce any bias towards one particular game studio or one particular game genre. Also, they provide a more complete view on the problems that they faced, including problems with management, design, marketing, etc.

**Research bias:** The analysis of problems relied on our own interpretation of the postmortems and the reported problems. This interpretation could vary according to each researcher. To reduce any bias, we discussed the problems and our interpretations in each iteration of reading the postmortems, updating the catalogue of problem types only when necessary and only until we reached a fix point, as described in Figure 1.

**Different numbers of problems per year:** At first, we chose postmortems randomly but some years have more postmortems than others so we mitigated this unbalance by dividing the numbers of problems by the numbers of postmortems per year for the historical analysis.

**Recommendations:** Our recommendations are based not only on the data but also on our understanding of the literature and knowledge about game development in particular and software engineering in general. Some of the problems are too specific to one project to be generalised. Therefore, we tried to be general yet avoid being obvious. We did not make any recommendation about game design because related problems are not software-engineering problems.

8. Conclusion

Little is known of the problems faced by game developers during their projects as the game industry has a closed-source nature. We used postmortems to overcome this barrier and better understand the problems of the game industry. We analyzed more than 200 postmortems, comprising 927 problems divided in 20 types from 1997 to 2019.

Through our analysis, we described the overall landscape of game-industry problems in the past 23 years and how these problems evolved over the years. We reported the following main findings:

- Based on the number of problems groups and types, the game industry suffer from management and production problems in the same proportion. However, production problems are concentrated mostly in technical and design while management problems are more spread across the problems types.
- Based on the problems groups over the years, management problems decreased over the years giving space to business problems, while production problems remained constant.
- Based on the evolution of the problem types over the years:
  - Technical and game design problems are decreasing over the years, the later only after the last decade;
  - Problems related to the team increase over the last decade;
  - Marketing problems are the ones that had the biggest increase over the 23 years compared to other problem types;
- Finally, considering the problem sub-types, the majority of the main root causes are related to people, not technologies.

Table 6 summarises these most common problems, their root causes, possible solutions, and our recommendations for future projects.

Our findings show that many problems require project-specific solutions that are hard to generalize. However, we hope that our discussion about these problems, and the recommendations, will help practitioners and researchers better understand the game industry.

In future work, we will study more postmortems to enrich further our analysis. We will also reach out to videogame developers to vet and refine further the identified types and problems as well as to survey their opinions on the identified solutions and our proposed recommendations. We thus wish to start a conversation between academia and the videogame industry on their problems and possible solutions.

**Acknowledgement**

The authors were partly supported by the NSERC Discovery Grant and Canada Research Chairs programs.
Table 6
Summary of the main sub-types (root causes) and the recommendations.

| Group          | Type        | SubType (root cause)     | Recommendation                                                                                                                                 |
|----------------|-------------|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| Management     | people      | Insufficient workforce   | Consider “sharing the load” of complex tasks, avoid blindly hiring more workforce, and try outsourcing if the budget is limited (Section 4.1). |
| Management     | people      | Environment problems     | Balancing the expertise levels among developers and keeping small teams with small scopes, as well shield the developers to external interference (Section 4.2). |
| Business       | Marketing   | Wrong marketing strategy | Stop trying naively promote their games, focus on strong points of the game, and strengthen the relationship with users (Section 4.3). |
| Management     | feature     | Underestimation          | Avoid relying solely on human expertise and invest in building a knowledge base about past projects to better estimate the future tasks (Section 4.4). |
| Production     | Game Design | Unclear game design vision | Keep a clear vision of the game design by not wasting time with static documents, spend more time in prototyping and playtesting (Section 4.5). |
| Production     | Game Design | Lack of fun              | Allocate time to find the “fun” by polish the mechanics, art, and story of the games, as well as extensive playtesting to identify the weak points (Section 4.6). |
| Production     | Technical   | Platform and technology constraints | Have a better understanding of the platform architectures and limitations before committing to a project (Section 4.7). |
| Production     | Game Design | Game design complexity   | Use KISS and Occam’s razor principles to keep the game design simple (Section 4.8). |
| Production     | Tools       | Inadequate or lack of tools | Allocate time to experiment with multiple tools and game engines before choose or build one (Section 4.9). |
| Management     | people      | Misaligned teams         | Consider changing the team structure if there is communication problems among the departments (Section 4.10). |
References

[1] Alami, A., Cohn, M.L., Wąsowski, A., 2020. Why Does Code Review Work for Open Source Software Communities? IEEE, 25–31. doi:10.1109/ICSE.2019.89111.

[2] Blow, J., 2004. Game Development Harder Than You Think. Queue, 1, 28. doi:10.1145/971564.971590.

[3] Bourque, P., Fairley, R.E., Society, I.C., 2014. Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0. 3rd ed., IEEE Computer Society Press, Washington, DC, USA.

[4] Brooks Jr, F.P., 1995. The mythical man-month: essays on software engineering. Pearson Education.

[5] Callele, D., Neufeld, E., Schneider, K., 2005. Requirements engineering and the creative process in the video game industry, in: 13th IEEE International Conference on Requirements Engineering (RE05), IEEE, pp. 240–250. doi:10.1109/re.2005.58.

[6] Edholm, H., Lidström, M., Steghöfer, J., Burden, H., 2017. Crunch Time: The Reasons and Effects of Unpaid Overtime in the Games Industry doi:10.1109/ICSE-SEIP.2017.18.

[7] Hofstadter, D.R., 1999. Gödel, Escher, Bach. Basic Books. URL: https://www.goodreads.com/book/show/24113.G_del_Escher_Bach.

[8] Hyrynsalmi, S., Klotins, E., Unterkalmsteiner, M., Gorschek, T., Tripathi, N., Pompermaier, L.B., Prikladnicki, R., 2018. What is a Minimum Viable (Video) Game? SpringerLink, 217–231 doi:10.1007/978-3-030-02131-3_20.

[9] Jorgensen, M., Shepperd, M., 2006. A Systematic Review of Software Engineering Cost Estimation Studies. IEEE Trans. Software Eng. 33, 33–53. doi:10.1016/j.isee.2007.256943.

[10] Kanode, C.M., Haddad, H.M., 2009. Software engineering challenges in game development, in: 2009 Sixth International Conference on Information Technology: New Generations, IEEE, pp. 260–265. doi:10.1109/itrng.2009.74.

[11] Kasurinen, J., Palacin-Silva, M., Vanhala, E., 2017. What Concerns Game Developers? A Study on Game Development Processes, Sustainability and Metrics. International Workshop on Emerging Trends in Software Metrics, WETSoM, 15–21. doi:10.1109/WETSoM.2017.3.

[12] Koster, R., 2013. Theory of Fun for Game Design. 2nd ed., O’Reilly Media, Inc.

[13] Kud, T., Mathias, S., Schmidt, C.T., Heinzl, A., 2019. How Pair Programming Influences Team Performance: The Role of Backup Behavior, Shared Mental Models, and Task Novelty. Inf. Syst. Res.

[14] Lenarduzzi, V., Taibi, D., 2016. MVP Explained: A Systematic Mapping Study on the Definitions of Minimal Viable Product. 2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 112–119. doi:10.1109/SEAA.2016.58.

[15] Lewis, C., Whitehead, J., 2011. The whats and the whys of games and software engineering, in: Proceeding of the 1st international workshop on Games and software engineering - GAS 11, ACM Press, pp. 1–4. doi:10.1145/1984674.1984676.

[16] Lin, D., Bezemer, C.P., Hassan, A.E., 2017. Studying the urgent updates of popular games on the Steam platform. Empirical Software Engineering 22, 2095–2126. doi:10.1007/s10664-016-9488-2.

[17] Nassif, A.B., Azzeh, M., Capretz, L.F., Ho, D., 2016. Neural network models for software development effort estimation: a comparative study. Neural Comput. &. Applic. 27, 2369–2381. doi:10.1007/s00521-015-2127-1.

[18] Petrillo, F., Pimenta, M., Trindade, F., Dietrich, C., 2009. What went wrong? a survey of problems in game development. Computers in Entertainment 7, 1. doi:10.1145/1486508.1486521.

[19] Politowski, C., Fontoura, L., Petrillo, F., Guéhéneuc, Y.G., 2016. Are the old days gone?: A survey on actual software engineering processes in video game industry, in: Proceedings of the 5th International Workshop on Games and Software Engineering, ACM, pp. 22–28. URL: http://doi.acm.org/10.1145/2896958.2896960, doi:10.1145/2896958.2896960.

[20] Politowski, C., Fontoura, L.M., Petrillo, F., Guéhéneuc, Y.G., 2018. Learning from the past: A process recommendation system for video game projects using postmortems experiences. Information and Software Technology 100, 103–118. doi:10.1016/j.infsof.2018.04.003.

[21] Politowski, C., Petrillo, F., Montandon, J.E., Valente, M.T., Guéhéneuc, Y.G., 2020a. Are game engines software frameworks? a three-perspective study. arXiv:2004.05785.

[22] Politowski, C., Petrillo, F., Ullmann, G.C., de Andrade Werly, J., Guéhéneuc, Y.G., 2020b. Dataset of video game development problems. arXiv:2001.00891.

[23] Sadowski, C., Zimmermann, T., 2019. Rethinking Productivity in Software Engineering. Apress, Berkeley, CA. doi:10.1145/1486508.1486521.

[24] Schreier, J., 2017. Blood, Sweat, and Pixels: The Triumphant, Turbulent Stories Behind How Video Games Are Made. HarperCollins.

[25] Stol, K., Ralph, P., Fitzgerald, B., 2016. Grounded theory in software engineering research: A critical review and guidelines, in: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), pp. 120–131. doi:10.1109/icse.2016.110.

[26] Tschang, F.T., 2005. Videogames as interactive experiential products and their manner of development. International Journal of Innovation Management 9, 103–131.

[27] Washburn, M., Sathiyanarayanan, P., Nagappan, M., Zimmermann, T., Bird, C., 2016. 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 Companion - ICSE 16, ACM Press. doi:10.1145/2889160.2889253.
### Table 7
Types and Sub-types of the *Production* group.

| Type       | SubType                           | N     |
|------------|-----------------------------------|-------|
| Bugs       | Lack of proper organization/tracking | 14    |
| Bugs       | Graphic/sound issues              | 9     |
| Bugs       | Platform/hardware issues          | 6     |
| Bugs       | Game mechanic/system issue        | 4     |
| Design     | Unclear game design vision        | 28    |
| Design     | Lack of fun                       | 27    |
| Design     | Game design complexity            | 23    |
| Design     | Balancing issues                  | 19    |
| Design     | Lack of polish                    | 13    |
| Design     | Game too short/simple             | 10    |
| Design     | Release/censorship issues         | 8     |
| Documentation | Lack of design documentation    | 8     |
| Documentation | Lack of technical documentation  | 7     |
| Documentation | Poor assets management           | 4     |
| Documentation | Documentation management issues   | 3     |
| Prototyping | Not enough time or focus          | 7     |
| Prototyping | Prototype is too simple          | 5     |
| Prototyping | Prototype is too complex          | 4     |
| Prototyping | No prototyping                    | 3     |
| Technical  | Platform and technology constraints | 24    |
| Technical  | Optimization and Performance      | 17    |
| Technical  | Game engine and Libraries         | 13    |
| Technical  | Network and Multiplayer           | 9     |
| Technical  | Re-work and Wasted work           | 8     |
| Technical  | Build and Load time               | 8     |
| Technical  | Animation and 3D                  | 7     |
| Technical  | Source control and file management| 6     |
| Technical  | Porting issues                    | 6     |
| Technical  | Networking complexity             | 6     |
| Technical  | Programming language and Algorithms | 5     |
| Technical  | Production pipeline               | 5     |
| Technical  | Physics and Collision             | 5     |
| Technical  | Novelty and change                | 5     |
| Technical  | Performance issues                | 4     |
| Technical  | Patch strategies and Infrastructure | 4     |
| Technical  | Misc: UI and Localization         | 4     |
| Technical  | Coding/architecture issues        | 4     |
| Testing    | Insufficient test coverage        | 13    |
| Testing    | Process and testing plans issues  | 13    |
| Testing    | Specific project requirements     | 7     |
| Testing    | Scope too big to test properly    | 7     |
| Testing    | Poor feedback                     | 5     |
| Testing    | Reproducibility of bugs           | 2     |
| Tools      | Inadequate or lack of tools       | 22    |
| Tools      | Lack of expertise with the tool   | 12    |
| Tools      | Concurrent tool development       | 11    |
| Tools      | Middleware issues                 | 10    |
| Tools      | Maintenance issues                | 9     |
| Tools      | Third-party issues                | 7     |
| Tools      | Hardware compatibility issues     | 5     |
| Tools      | Tool switch                       | 4     |

### Table 8
Types and Sub-types of the *Management* group.

| Type       | SubType                           | N     |
|------------|-----------------------------------|-------|
| Communication | Misaligned teams                | 22    |
| Communication | Poor dev/pub communication    | 10    |
| Communication | Poor PR                         | 7     |
| Communication | Different physical locations    | 3     |
| Communication | Help/support issues             | 2     |
| Crunch-time | Not enough workforce            | 7     |
| Crunch-time | Management/financial issues     | 6     |
| Crunch-time | Growing scope                   | 6     |
| Crunch-time | Publisher set tight deadlines   | 5     |
| Crunch-time | Delays/scheduling issues        | 5     |
| Delays     | Technical/platform issue        | 11    |
| Delays     | Poor resource management        | 11    |
| Delays     | Publishing/business issues      | 4     |
| Delays     | Lack of workforce               | 4     |
| Team       | Insufficient workforce          | 49    |
| Team       | Environment problems            | 48    |
| Team       | Unexpected team disruption      | 11    |
| Team       | Inexperienced staff              | 5     |
| Team       | Overstaffing                     | 2     |
| Budget     | Difficulties with external funding | 10   |
| Budget     | Limited self funding             | 7     |
| Budget     | Poor management                  | 4     |
| Cutting features | Not enough time     | 9     |
| Cutting features | Idea was considered overambitious | 6     |
| Cutting features | Technical limitations        | 4     |
| Feature-Creep | Design increments over time     | 10    |
| Feature-Creep | Design increments over time     | 10    |
| Feature-Creep | Complexity of game mechanics   | 8     |
| Feature-Creep | Complexity of game mechanics   | 8     |
| Feature-Creep | Poor feature planning           | 4     |
| Feature-Creep | Poor feature planning           | 4     |
| Multiple-projects | Resource conflict            | 10    |
| Multiple-projects | Project was part-time job     | 2     |
| Multiple-projects | Procrastination                | 2     |
| Multiple-projects | Building engine at the same time | 2     |
| Planning   | Underestimation                  | 34    |
| Planning   | Ignoring or changing the plan   | 14    |
| Scope      | Overambitious scope             | 15    |
| Scope      | Poor resource estimation        | 8     |
| Scope      | Lack of initial design definitions | 6     |
| Scope      | Poor scope management           | 3     |
| Scope      | Poor complexity estimation      | 3     |
| Security   | Piracy                           | 2     |

### Table 9
Types and Sub-types of the *Business* group.

| Type       | SubType                           | N     |
|------------|-----------------------------------|-------|
| Marketing  | Wrong marketing strategy          | 35    |
| Marketing  | No plan, budget, or not enough marketing | 15 |
| Marketing  | Publisher/platform/hardware problems | 11 |
| Marketing  | Game hard to market               | 9     |
| Monetization | Wrong monetization model         | 9     |
| Monetization | Game did not profit              | 7     |
| Monetization | Publisher/platform/market issues | 4     |
| Monetization | Payment service issues           | 3     |
| Monetization | Lack of business expertise       | 3     |