Goal : To provide a dataset about the problems faced.

Dataset of Video Game Development Problems*

Cristiano Politowski
c_polito@encs.concordia.ca
Concordia University
Montreal, Quebec, Canada

Fabio Petrillo
fabio@petrillo.com
UniversitÃĽ du QuÃĽbec Ãă
Chicoutimi
Chicoutimi, Quebec, Canada

Gabriel Cavalheiro Ullmann
gabriel.cavalheiro@sou.unijui.edu.br
UNIJUI
Santa Rosa, RS, Brazil

Josias de Andrade Werly
alu201460044@uniriter.edu.com
UniRitter
Porto Alegre, RS, Brazil

Yann-Gaël Guéhéneuc
yann-gael.gueheneuc@concordia.ca
Concordia University
Montreal, Quebec, Canada

ABSTRACT

Different from traditional software development, there is little information about the software-engineering process and techniques in video-game development. One popular way to share knowledge among the video-game developers' community is the publishing of postmortems, which are documents summarizing what happened during the video-game development project. However, these documents are written without formal structure and often providing disparate information. Through this paper, we provide developers and researchers with grounded dataset describing software-engineering problems in video-game development extracted from postmortems. We created the dataset using an iterative method through which we manually coded more than 200 postmortems spanning 20 years (1998 to 2018) and extracted 1,035 problems related to software engineering while maintaining traceability links to the postmortems. We grouped the problems in 20 different types. This dataset is useful to understand the problems faced by developers during video-game development, providing researchers and practitioners a starting point to study video-game development in the context of software engineering.

KEYWORDS

Dataset, video game, postmortem, development problems

ACM Reference Format:
Cristiano Politowski, Fabio Petrillo, Gabriel Cavalheiro Ullmann, Josias de Andrade Werly, and Yann-Gaël Guéhéneuc. 2020. Dataset of Video Game Development Problems. In *17th International Conference on Mining Software Repositories (MSR '20), October 5–6, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3379597.3387486

1 INTRODUCTION

Video games are a profitable industry [7] for big companies like EA[1] and for *indie* developers[2] brave enough to face a competitive endeavor. Successful games are plenty, although they usually come at a price. Indeed, video-game development is known for its management problems [8, 13], which translate into large numbers of issues in the final products. For example, 80% of the top 50 games on Steam[3] need critical updates [6].

Video games form also a competitive market in which knowledge is the main weapon against competitors. The lack of information about the process, the techniques, the game engine used for games are but a few examples of how hard it is to understand video-game development. This lack of information also prevents new developers (and even veterans) to avoid common problems by learning from previous mistakes.

One source of information about video-game development that is public are postmortems. Video-game postmortems are documents that summarize the developers' experiences with the game development, written often right after the game is launched [13]. These documents are usually written by managers or senior developers [3]. They often include five sections either about "what went right" and "what went wrong" during the game development.

- ✓ *"What went right"* discusses the best practices adopted by 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.

Postmor-tem.

Thus, postmortems offer an "open and honest window into the development of games" [13]. However, these documents are written without formal structure and often providing disparate information, in particular with respect to the software-engineering of the games. Consequently, complete, trustful information about game development is hard to find, which limits the number of studies about games from the point of view of software engineering. Yet, we believe that most of software-engineering problems could be mitigated if developers had information beforehand.

1
2
problems faced.

*The dataset is available in the following page: https://github.com/game-dev-database/postmortem-problems.

[1] Eletronic Arts report from October 2019 shows $3.8 bilions of revenue (http://bit.ly/2LA7us1).

[2] Generally, indie developers are small, self-funded companies that create games on a small scale.

[3] Steam is a video-game, digital, distribution service platform available at https://store.steampowered.com/.

MSR '20, October 5–6, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7517-7/20/05…$15.00
https://doi.org/10.1145/3379597.3387486

Through this paper, we propose a curated dataset describing software-engineering problems in video-game development. We compile a large set of information about problems in game development related to software engineering. We analyze 200 postmortems from the Gamasutra Website[4] and collected 1,035 problems. We categorized these problems into 20 different problems types and provide a structure to store this information. The dataset is available in the following page: https://github.com/game-dev-database/postmortem-problems.

The paper is structured as follows. Section 2 discusses papers that used video-game postmortems as a knowledge base. Section 3 describes how the data was gathered. Section 4 presents how the metadata is organized. Section 5 shows the results of the database. Section 6 concludes the paper with future work.

2 GAME DEVELOPMENT PROBLEMS

Callele et al. [3] analyzed 50 postmortems from the Game Developer Magazine and investigated how requirements engineering is applied in game development. They grouped "What went right" and "What went wrong" in the five categories: (1) *pre-production*, problems outside of the traditional software development process; (2) *internal*, those related to project management and personnel; (3) *external*, those outside of the development team's control; (4) *technology*, those related to the creation or adoption of new technologies; and, (5) *schedule*, those related to time estimates and overruns. They concluded that project management is the greatest contributor to the success or failure in video-game development.

Petrillo et al. [8] analyzed 20 postmortems from the Gamasutra Website, searching for the most common problems, which they compared with traditional software problems. They concluded that (1) video-game development suffer mostly from management problems instead of technical problems; (2) the problems found 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*.

Kanode and Haddad [4] used postmortems to discuss the challenges of traditional software engineering in video-game development. They reported differences between game development and traditional development and concluded that video-game development must adopt and adapt software-engineering practices.

Lewis and Whitehead [5] used two previous papers [2, 12] to identify problems in games and whether/why these could be of interest to the software-engineering research community. They highlighted some areas to explore further and differences between games and traditional software.

Washburn et al. [13] analyzed 155 postmortems, identified some characteristics and pitfalls, and suggested good practices. They reported the following main problems: *obstacles* (37%), *schedule* (25%), *development process* (24%), and *game design* (22%).

Politowski et al. [9] extracted development processes from 20 postmortems. They concluded that the majority of the game industry uses agile instead of the waterfall method. In their next work, Politowski et al. [10] used 55 postmortems to create a recommendation system for new video-game projects using previous mistakes and good practices.

From the gray literature, Ara Shirinian [1] wrote an article analyzing 24 postmortems from 2008 to 2010 and created a short list of definition from "what went right" and "what went wrong": *Design* (covers all situations and decisions that were made that are clearly external to the direct team and development process); *Art* (Relating to art decisions); *Production/Process* (This relates to scheduling, work prioritization, etc); *Programming* (technical issues); and *Testing* (all traditional QA functions)

These previous papers used postmortems to analyze video-game development, in particular Petrillo et al. [8] and Washburn et al. [13]. We extend these papers with more methodical approach and a study of a larger number of postmortems. We study postmortems from the point of view of software engineering using an iterative approach, which we summarise in the following section.

3 METHOD OF ANALYSIS

We designed the process of analysis to be iterative where the data keep in constant evolution as we add and refactor the data each new iteration. Figure 1 shows the steps performed to analyse the data from the postmortems.

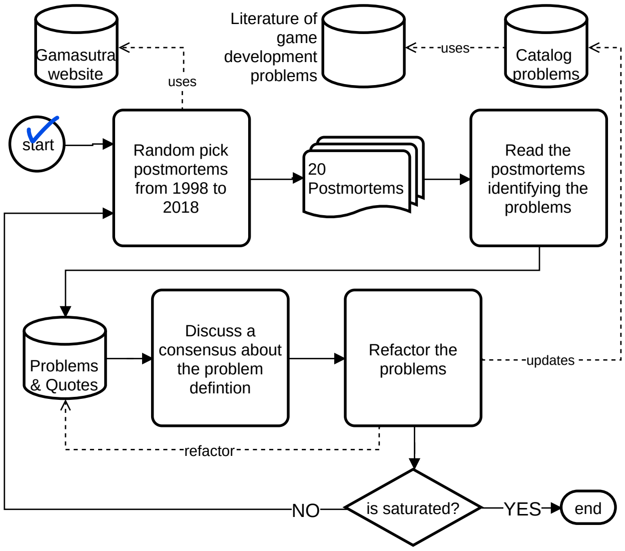
Gamasutra website
uses
Literature of game development problems
uses
Catalog problems
start
Random pick postmortems from 1998 to 2018
20 Postmortems
Read the postmortems identifying the problems
Problems & Quotes
Discuss a consensus about the problem defintion
Refactor the problems
updates
refactor
NO
is saturated?
YES
end

Figure 1: Steps performed to analyse the data.

We start by randomly picking 20 postmortems (for each author) from the Gamasutra Website between the years 1998 to 2018. Next, each author reads the postmortems, focusing on the "What went wrong" section and using a coding technique from Grounded Theory [11]. Each author identifies all the problems reported by the postmortems, extracting quotes and grouping similar problems, based on the previous literature definitions [8, 13]. Previous definitions were used to create a *catalog of problems*, which contains *problem types*, *problem groups*, and *descriptions*. We add each new problem type absent from the catalog with its description.

[4] Gamasutra is the main and most complete source of video-game postmortems: http://www.gamasutra.com. It is a descendant of the Game Developers Magazine that ended in 2013.

In the next step, we discuss the findings, review some doubts, and analyse the correctness of the catalog. Any change at this point, on the catalogue or on the database of problems, results in a update in both documents, which sometimes lead us to re-read the postmortems. We reiterate this process until it reaches saturation: until no more new type of problems appears (stop updating the the catalog of problems).

To keep the distribution of the postmortems read by year balanced, eventually we chose specific postmortems to analyse instead of randomly picking them. Also, not all postmortems are relevant for this study as they might not have useful information regarding the development process. When that happens, we discard the postmortem.

4 METADATA OF THE DATASET

The *catalog of problems* is a document listing the possible problems we could find while reading the postmortems. We started building it using problems’ types we gathered from the literature and we update it during during the postmortems analysis.

The catalogue document is updated for every new problem type we end up discovering. The authors review each new problem type and reach to a conclusion. The final version of the document contains 20 different types of 1,035 problems divided in three groups.

Table 1 shows the final version of the catalog where a *Problem Type* is a index to define the problem which has a short *Description* and belongs to one *Problem Group*.

To store the problems gathered from postmortems we defined a data model which Figure 2 shows in UML class diagram. Each *Postmortem* has a *Game* which has a collection of problems. Each *Problem* has a *Type* and *Group*. Also, the Game must have: a Platform [1-3] (PC, Console, Mobile), a Genre [1-12] (Action, Adventure, RPG, Simulation, Strategy, Puzzle, Sports, Platformer, Shooter, Racing, Roguelike, Running), and a Mode [1-3] (Single-player, Multi-player, Online).

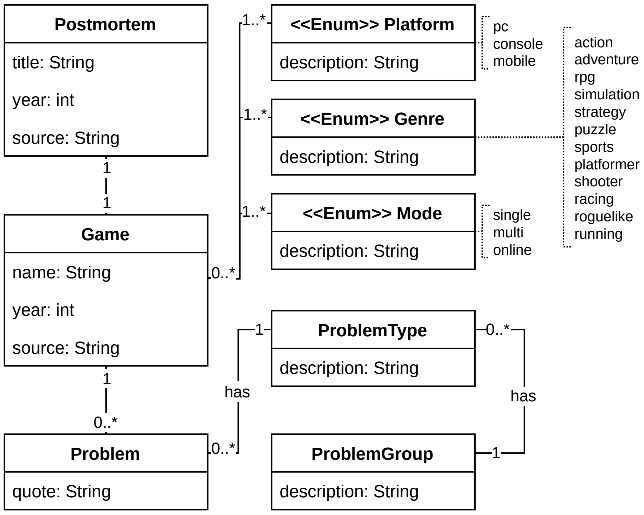

Figure 2: Class diagram with the structure of each entry on the dataset.

Table 1: Catalogue with the video game development problems identified from the postmortem analysis.

Group	Type	Description of the problem
Production	Design	Any problem regarding the design of the game, like balancing the gameplay. Not a technical detail.
	Documentation	Not planning the game beforehand, not documenting the code, artifacts or game plan.
	Tools	Any problem with tools like engines, APIs, development kits, third-party software, etc.
	Technical	Problems with the team code/assets infrastructure.
	Testing	Any problem regarding the testing.
	Bugs	When there are too many bugs in the game/engine, any failure in the game design or technical issues.
	Prototyping	Lack of or no prototyping phase nor validation of the gameplay/feature.
Management	Unrealistic Scope	Planning too many features that end up impossible to implement it in a reasonable time.
	Feature Creep	Adding unplanned new features to the game during its implementation.
	Cutting Features	Cutting features previously planned because of other factor like short deadlines.
	Delays	Problems regarding any delay in the production.
	Crunch Time	When developers continuously spent extra hours working in the project.
	Communication	Problems regarding communication with any stakeholder.
	Team	Problems in setting up the team, loss of professionals during the development or outsourcing.
	Over Budget	Project cost more money than expected.
	Multiple Projects	When there is more than one project being developed at the same time.
	Planning	Problems involving too much time planing/scheduling or the lack of it.
	Security	Problems regarding leaked assets.
Business	Marketing	Problems regarding marketing/advertising
	Monetization	Problems with the process used to generate revenue from a video game product.

An example of the dataset structure is shown in Table 2. In this case, the game “Baldur’s Gate II” from 2001 was analyzed and the entry shows a problem regarding “Testing”, which belongs to the “Production” group. The quote relates the problem to a non-modified version taken from the postmortem.

We began storing the data using JSON format following the structure described in Figure 2 and validated using JSON Schema. Later we migrated the data to a table format (.CSV) where each entry (line) refers to a problem. This makes the data easy to be analyzed with scripts, specially in *R* language.

5 DATASET ANALYSIS

The dataset contains 200 video-game projects ranging from 1998 to 2018. There are, in total, 1,035 problems with its respective quotes. The median of problems by title (video game) is 5 and the median

Table 2: Example of the dataset structure with one entry.

Column	Value
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
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).

of problems by year is 48. Figure 3 shows the problems by group: management and production problems are the most common with around 45% each while business issues sum up to 10%. Figure 4 shows the distribution of the problems related to each type. In this case, Design, Technical, and Team problems are the most frequent with 35% overall.

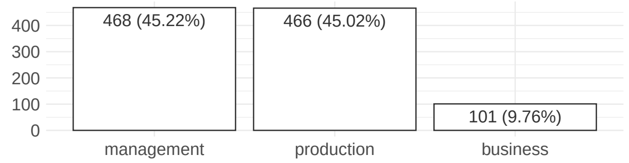

Figure 3: Number of problems related to each Group.

Regarding the platforms, the projects are mainly for PC, with 787 problems, followed by 475 Console problems, and 254 Mobile problems. 90 problems are related to multi-platform projects, that is, PC, Console, and Mobile.

The dataset is hosted in a open repository on Github. This allows anyone to access and contribute by using the *pull request* feature. We decided for this approach to allow the contribution to be reviewed before being accepted. The contributor can also add to the catalog of problems and other meta-data like game platform, game genre, and game mode, allowing the dataset to keep evolving overtime.

About the limitations of the paper, this dataset is related only to postmortems problems, that is, things that "went wrong". Our plan is to also implement good practices gathered from the "what went right" sections in the future. Also, the problems described by the authors are, in general, abstracts, without technical details.

6 CONCLUSION

We presented a database of video-game development problems gathered from 200 postmortems. The database contains 1,035 problems categorized in three 20 different types. It fills the gap between video-game development and software developers by providing a larger and more trustful information about video-game development problems. It thus provides a starting point for new researchers and practitioners to understand and propose solutions to video-game development problems.

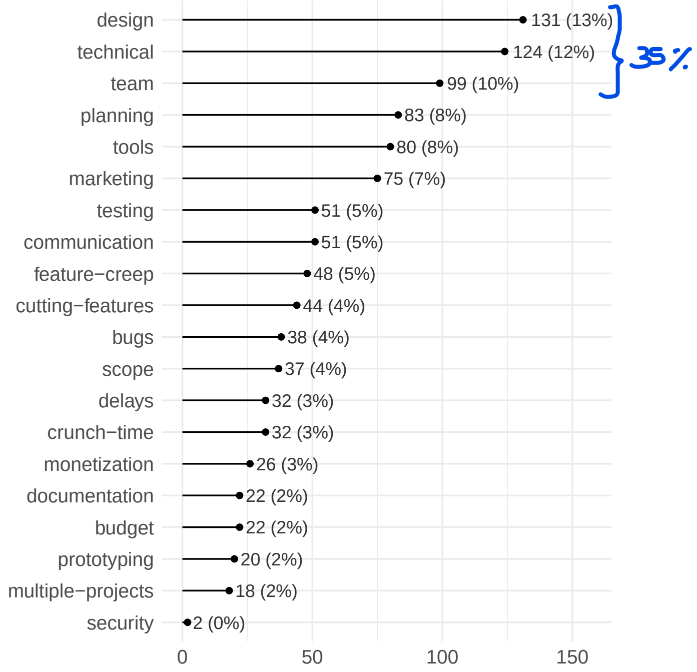

Figure 4: Number of problems related to each Type.

Future plans for this work include (1) keep updating the database as more postmortems appear; (2) include other types of sources as, for example, technical blog posts and conference presentations like Game Developer Conference – GDC; and (3) expanding the database by using the "What went right" section to gather *good practices* instead of problems; 1

This dataset can be used for researchers and practitioners to make a comparison with the common problems in traditional software development to better understand the differences in both 2 domains and check the possibility of implementing traditional software engineering practices to mitigate those issues. Also, the quotes in the problems should be further refined to more precisely define what are the causes of the problems. For example, what are the main 3 causes of "technical" problems? Is it related to the game's type? Is there any correlation between the problems? From there we can draw common solutions from these issues. Finally, investigate if the technical evolution (new hardware and tools) influence the type of problems in game development. 4

ACKNOWLEDGMENTS

The authors have been partly supported by the NSERC Discovery Grant program and Canada Research Chairs program.

REFERENCES

[1] Ara Shirinian. 2011. Dissecting The Postmortem: Lessons Learned From Two Years Of Game Development Self-Reportage. https://www.gamasutra.com/view/feature/6309/dissecting_the_postmortem_lessons_.php?print=1. [Online; accessed 5-March-2020].

[2] Jonathan Blow. 2004. Game Development Harder Than You Think. *Queue* 1, 10 (Feb. 2004), 28. https://doi.org/10.1145/971564.971590

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

[4] Christopher M. Kanode and Hisham M. Haddad. 2009. Software engineering challenges in game development, In 2009 Sixth International Conference on Information Technology: New Generations. *ITNG 2009 - 6th International Conference on Information Technology: New Generations*, 260–265. https://doi.org/10.1109/itng.2009.74

[5] Chris Lewis and Jim Whitehead. 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, 1–4. https://doi.org/10.1145/1984674.1984676

[6] Dayi Lin, Cor Paul Bezemer, and Ahmed E. Hassan. 2017. Studying the urgent updates of popular games on the Steam platform. *Empirical Software Engineering* 22, 4 (2017), 2095–2126. https://doi.org/10.1007/s10664-016-9480-2

[7] Newzoo. 2019. 2019 Global Games Market Report. https://newzoo.com/insights/trend-reports/newzoo-global-games-market-report-2019-light-version/. [Online; accessed 1-October-2019].

[8] Fábio Petrillo, Marcelo Pimenta, Francisco Trindade, and Carlos Dietrich. 2009. What went wrong? a survey of problems in game development. *Computers in Entertainment* 7, 1 (Feb. 2009), 1. https://doi.org/10.1145/1486508.1486521

[9] Cristiano Politowski, Lisandra Fontoura, Fabio Petrillo, and Yann-Gaël Guéhéneuc. 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 (GAS '16)*. ACM, 22–28. https://doi.org/10.1145/2896958.2896960

[10] Cristiano Politowski, Lisandra M. Fontoura, Fabio Petrillo, and Yann Gaël Guéhéneuc. 2018. Learning from the past: A process recommendation system for video game projects using postmortems experiences. *Information and Software Technology* 100, March (2018), 103–118. https://doi.org/10.1016/j.infsof.2018.04.003

[11] K. Stol, P. Ralph, and B. Fitzgerald. 2016. Grounded Theory in Software Engineering Research: A Critical Review and Guidelines. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. 120–131. https://doi.org/10.1145/2884781.2884833

[12] F Ted Tschang. 2005. Videogames as interactive experiential products and their manner of development. *International Journal of Innovation Management* 9, 01 (2005), 103–131.

[13] Michael Washburn, Pavithra Sathiyanarayanan, Meiyappan Nagappan, Thomas Zimmermann, and Christian Bird. 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. *Proceedings of the 38th International Conference on Software Engineering (ICSE 2016 SEIP Track)*. https://doi.org/10.1145/2889160.2889253