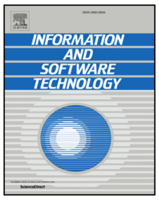
Information and Software Technology 100 (2018) 103–118

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



Learning from the past: A process recommendation system for video game projects using postmortems experiences[☆]

Cristiano Politowski[*,a], Lisandra M. Fontoura[a], Fabio Petrillo[b], Yann-Gaël Guéhéneuc[b]

[a] Departamento de Computação Aplicada (DCOM), Universidade Federal de Santa Maria, Santa Maria, RS, Brazil
[b] Department of Computer Science & Software Engineering, Concordia University, Montréal Quebec H3G 1M8, Canada

ARTICLE INFO

Keywords:
Software development process
Video game development
Recommendation system

ABSTRACT

Context: The video game industry is a billion dollar industry that faces problems in the way games are developed. One method to address these problems is using developer aid tools, such as Recommendation Systems. These tools assist developers by generating recommendations to help them perform their tasks.

Objective: This article describes a systematic approach to recommend development processes for video game projects, using postmortem knowledge extraction and a model of the context of the new project, in which “postmortems” are articles written by video game developers at the end of projects, summarizing the experience of their game development team. This approach aims to provide reflections about development processes used in the game industry as well as guidance to developers to choose the most adequate process according to the contexts they’re in.

Method: Our approach is divided in three separate phases: in the first phase, we manually extracted the processes from the postmortems analysis; in the second one, we created a video game context and algorithm rules for recommendation; and finally in the third phase, we evaluated the recommended processes by using quantitative and qualitative metrics, game developers feedback, and a case study by interviewing a video game development team.

Contributions: This article brings three main contributions. The first describes a database of developers’ experiences extracted from postmortems in the form of development processes. The second defines the main attributes that a video game project contain, which it uses to define the contexts of the project. The third describes and evaluates a recommendation system for video game projects, which uses the contexts of the projects to identify similar projects and suggest a set of activities in the form of a process.

1. Introduction

The video game industry is a multi-billionaire market, which revenues have been growing up through the years. According to the digital marketing company Newzoo, in 2016, the video game industry achieved a revenue of US$99.6 billions, 8.5% more than in 2015, expecting US$118 billions in 2019 [1].

Video game development is a highly complex activity [2]. Studies about the video game industry concluded that video game projects do not suffer from technical problems but face management and processes problems [3,4].

Problems like “unrealistic scope”, “feature creep”[1] and “cut of features” [3,4] are recurrent in game development. In addition, because of lack of maturity, development teams often do not adopt systematic processes [5] or they use traditional approaches, such as the waterfall process [6]. Other problems were identified in the IGDA annual report [7] in which, for example, 52% of the interviewees answered “yes” when asked if “crunch time”[2] is necessary during game development.

To help game developers minimize these problems, researchers proposed approaches to apply agile practices in game development [8–10]. However, recent work showed that at least 35% of the development teams continue using hybrid processes (mixing waterfall and agile) or *ad-hoc* processes [6]. Other researchers have tried, through a survey, understand the problems faced by game developers and the role

[☆] All the material of this project, including the extracted processes, video game contexts, and validations data is available on-line at https://polako.github.io/rec-sys-proc-games/.
[*] Corresponding author.
E-mail addresses: cpolitowski@inf.ufsm.br (C. Politowski), lisandra@inf.ufsm.br (L.M. Fontoura), fabio@petrillo.com (F. Petrillo), yann-gael.gueheneuc@concordia.ca (Y.-G. Guéhéneuc).
[1] *Feature creep* occurs in any software project when new functionalities are added during the development phase, thus increasing the project size [4].
[2] *Crunch time* is the term used in the video game industry for periods of extreme work overload, typically in the last weeks before the validation phase and in the weeks that precede the final deadline for project delivery [4].

https://doi.org/10.1016/j.infsof.2018.04.003
Received 5 October 2017; Received in revised form 15 March 2018; Accepted 11 April 2018
Available online 12 April 2018

of software engineering in the game industry [11]. Their results showed that (1) the use of systematic processes, opposed to the development lacking any process, had a correlation to project success and (2) the most frequent problems were *delays, non realistic scope*, and *lack of documentation*. They also analyzed 20 postmortems and extracted their corresponding software processes and showed that the adoption of agile processes is increasing, corresponding to 65% of the identified processes [6].

Consequently, we claim that game development teams must improve their project management practices by adopting systematic processes. We also claim that an approach to improve processes is to learn from past project problems. Thus, we use as main source of information "postmortems", which are articles written by game developers at the end of their projects, summarizing the experiences of the game development team [12]. However, to be a useful source of information for developers, the non-structured data contained in the postmortems must be structured. The developers should also be warned of possible problems that may occur during development, given the lack of video game developers' knowledge about software process [4]. Thus, we want to develop a recommendation system to support video game development teams in building their development processes, recommending a set of activities and characteristics from past projects with similar contexts.

We describe an approach to build a recommendation system to recommend **software processes**, using **context** and a **similarity degree** with previous game development projects. We want the recommendation system to support game developers in choosing a sequence of activities and practices that fit within their contexts. Fig. 1 summarizes our approach: (1) the construction of the **processes database** with characteristics extracted from postmortems; (2) the definition of the **video game context** to characterize video game projects; and (3) the building of a **recommendation system** that suggests the **software process** using the defined context. Finally, we evaluate the recommendation system quantitatively, qualitatively and by performing a case study with a video game development team.

The remainder of this paper is structured as follows. Section 2 describes the background of our approach. Section 3 describes the method used to build and validate the recommendation system. Section 4 explains how we build and apply the recommendation system. Section 5 describes the validation of the recommended processes. Section 6 discusses our approach, the recommendation system, and its recommendations. Section 7 summarizes threats to the validity of our validation. Section 8 describes the related work. Section 9 summarizes our approach, its results, our contributions, and future work.

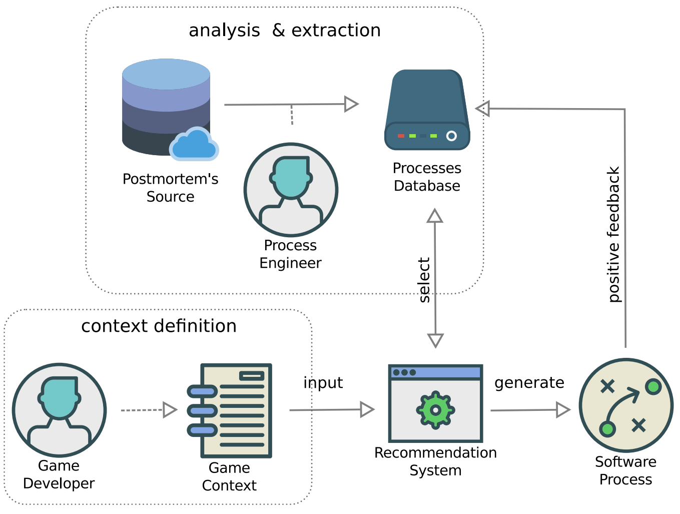


Fig. 1. Summary of the approach.

2. Background

This section briefly describes the concepts necessary to understand the rest of the article.

2.1. Recommendation systems

Recommendation systems (RS) are programs that use data to build models to help users in making decision. They recommend items based of explicit or implicit users' preferences. They help in alleviating the problem of information overload, showing users the most interesting items according to some criteria, offering relevance and diversity [13].

For example, e-commerce Web sites use recommendation systems to suggest their customers items. They may also be used in technical domains as well to help developers in their tasks. They are then labeled Recommendations Systems for Software Engineering (RSSE). They offer informations regarding software development activities. RS are divided in four different groups [14]:

1. **Collaborative filtering**: A technique for generating recommendations in which the similarity of opinions of agents on a set of items is used to predict their similarity of opinions on other items.
2. **Content-based**: A technique for creating recommendations based on the contents of the items and the profiles of the users' interests.
3. **Knowledge-based**: A technique for providing recommendations using knowledge models about the users and the items to reason about which items meet users' requirements.
4. **Hybrid**: A technique to generate recommendations that combines two or more of the previous techniques to build its recommendations.

RSSE have been used to a wide range of purposes. For example, *source code within a project*, helping developers to navigate through project's source code; *reusable source code*, providing new discoveries to programmers, like classes and functions; *code examples*, elucidating coders who does not know how implement a particular library / framework; *issue reports*, gathering and providing knowledge about past issues; recommending *tools, commands and operations* to solve certain problems in software projects with big scope; and recommending the best *person* to a perform a task.

Our problem is similar to these previous scenarios, particularly with *issue reports, recommending operations and persons*, because we want to help developers by recommending processes based on previous successful and unsuccessful video game projects, described in form of postmortems. Thus, we will use the *collaborative filtering* technique, however, instead of *opinions of agents* we will use game project details to compare the similarities between video game projects.

2.2. Software process

Humphrey [15] defined a software process as a set of activities, methods, and practices used in the production and evolution of software systems. There are many process types available in the literature, both academic and professional. Fowler [16] divides software processes into **predictive** and **adaptive** processes, according to their characteristics. Predictive processes emphasize sequential flows of activities and requirements definitions before the beginning of software development. Activities are defined beforehand. Such processes require strong requirements definitions. *Waterfall* is an example of this type of processes [17]. Adaptive processes imply short development cycles performed continuously, delivering ready-to-use features at the end of each cycle [18]. They emphasize continuous improvement of the processes [16]. Examples of this type of processes are *Scrum* [19] and *Extreme Programming* [20].

Some authors claim that adaptive or agile development has been used since the beginnings of software development [21], meanwhile

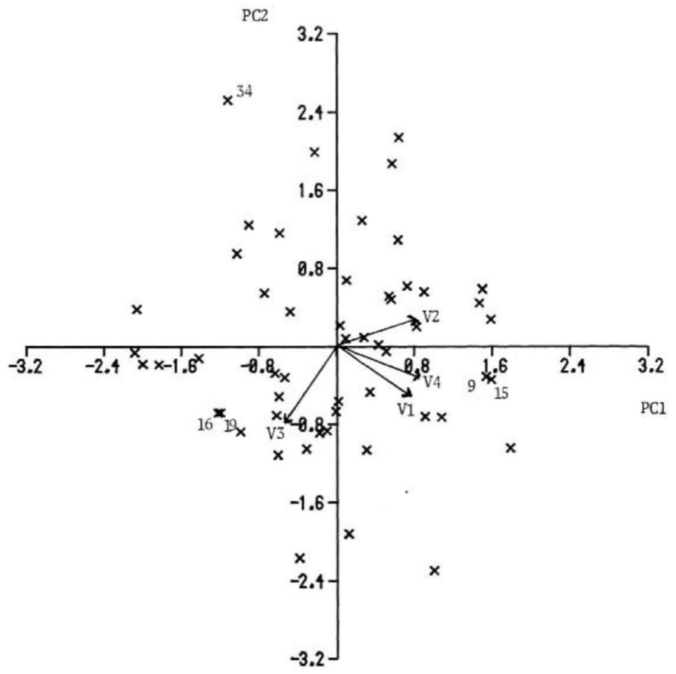


Fig. 2. PCA biplot example.

others see the adoption of agile methods as a "natural evolution" in the way software is developed [16]. In spite of that, the fast adoption of agile processes is notorious, principally after the Agile Manifesto in 2001[3]. According to the annual report made by *Version One* [22], in 2016, 96% of respondents claimed making use of agile practices. A fact also evidenced in the video games area, where the agile methodologies are majority [6,23].

2.3. Software context

Knowing a project context—its characteristics—is crucial to the success of a game development project. However, the variety of situations in which software projects are used is expanding, making existing premises outdated [24].

However, to the best of our knowledge, there has been only few attempts to categorize projects contexts. Boehm [25] defined five characteristics that distinguish predictive processes from agile processes: size, criticality, personnel, dynamism and culture. Kruchten [26] defined an approach to contextualize software projects by defining characteristics of agile projects in a model called *Octopus*. This model defines eight characteristics for software projects: size, criticality, business model, architecture, team distribution, governance, rate of change and age of the system.

Both previous works tried to contextualize software projects. However, video game projects, although software as well, have greater multidisciplinarity [3,4] and, consequently, we claim that these small sets of characteristics do not properly define video game software development projects.

2.4. Video games postmortems

Video games postmortems are documents that summarize the developers' experiences of a game development project after (either successful or unsuccessful) completion. They emphasize positive and negative outcomes of the development [12]. They are usually written right after the ends of game development projects by managers or senior developers [27]. They are an important tool for knowledge management and sharing [28]. They help teams to learn from their own experiences and plan future projects and also learn from the successes and problems of other teams. They are pieces of knowledge that can be reused by any development team and they capture examples and real-life development experiences. They are so important that some authors argue that no project should finish without postmortem [28].

Usually, postmortems follow the structure proposed in the *Open Letter Template* [29] and divide in three sections. The first section summarizes the project and presents some important outcomes of the project. The next two sections discuss the two most interesting characteristics of any of game development:

- **What went right** discusses the *best practices* adopted by developers, solutions, improvements, and project management decisions that help the project. All these characteristics are critical in planning future projects.
- **What went wrong** discusses difficulties, pitfalls, and mistakes experienced by the development team in the project, both technical and managerial.

It is important to note that postmortems are not exclusively about process. As a matter of fact, this kind article is, most of the time, very informal and does not contain any useful information regarding the development process. Given its free structure, authors used to write about a wide range of subjects including game design, game balancing, *gameplay* and many others details that do not add to development process details.

2.5. Principal Component Analysis

Principal Component Analysis (PCA) reduces the dimensionality of a set of data constituted of many related variables, keeping the majority of the variance contained in the data [30].

PCA is useful when there is a large set of variables because the data can be plotted in two dimensions, allowing a straightforward visual representation, instead of comparing numbers. A plot of the first two sets of variables, that is, a *biplot*, provides the best possible data representation in two dimensions, useful to find patterns as well [30]. Fig. 2 shows an example of *biplot* with four variables: V1 to V4. It is important to note that the more close a sample is from another, the more correlated they are, that is, more similar.

3. Research method

Our approach, illustrated by Fig. 1, is inspired by the "design concerns" related to build recommendation systems described by Robillard et al. [14]. It consists in four challenges:

1. *Data preprocessing*. Transform raw data in a sufficiently interpreted format. For instance, source code has to be parsed or commits have to be aggregated. See Section 3.1.
2. *Capturing context*. Different from traditional RS which are dependents of user profiles, RSSE focus on *tasks*. In this regard, a *task context* is all information which the recommender has access to produce recommendations. This information is generally incomplete and/or noisy. It is described in Section 3.2.
3. *Producing recommendations*. It is the algorithms' work on the data. Different strategies can be used, however, the most suitable approach for RSSE is *Collaborative Filtering*. See Section 3.3 for details.
4. *Presenting the recommendations*. Presents the items of interest, that is, *activities* in our case. Described in Section 3.4.

3.1. Data and preprocessing

Because of constraints, in particular trade secret, to access real

[3] http://agilemanifesto.org .

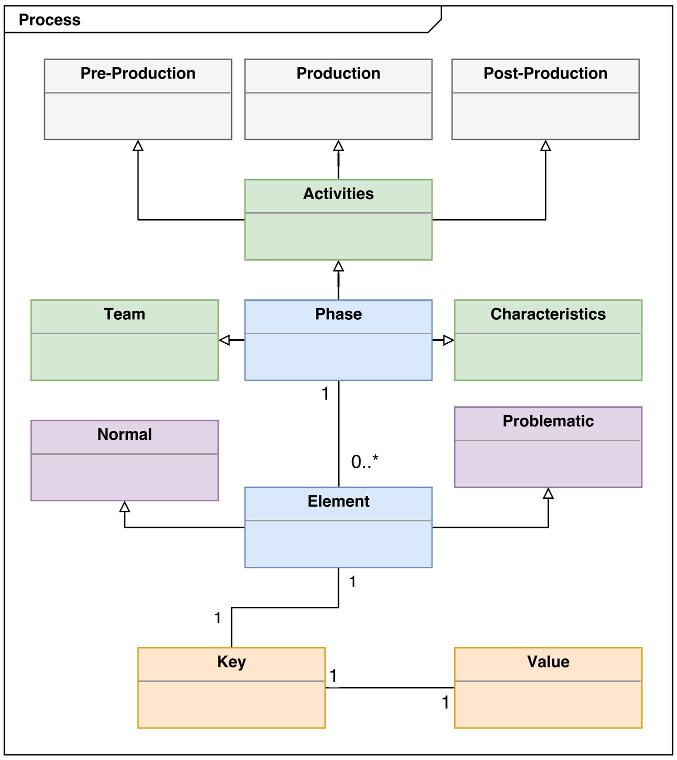


Fig. 3. Architecture used to store process' elements.

documentation and artifacts from video game development projects, we gathered data from public-domain source: postmortems from game development Web sites, like *Gamasutra*[4], and *Wikipedia* entries of game projects. We observed that only *Gamasutra* publishes postmortems about video games. Its dataset is large enough to fulfill our needs in this article. Moreover, all postmortems offered by *Gamasutra* follow the same format, thus a single source of data helps mitigating the lack of formal structure of postmortems.

We analyzed the data gathered from *Gamasutra* by reading every postmortem, collecting information about the development processes, activities, and team characteristics. We ignored details about game design or other information that does not pertain to the development processes. We organized data—process' elements—in a database following the model described in Fig. 3.

We defined a development process as a set of elements separated in different phases and sub-phases. The **Phase** categorizes the elements of the process, which can be:

- **Activities**, which represent tasks or steps describing work units that result in artifacts. For example, the activity of *prototyping*.
- **Team**, which pertain to the characteristics of teams. For example, the team's characteristic of *outsourcing*.
- **Characteristics**, which describe meta-data about the development process. For example, the characteristic of *project duration*.

Based on previous work [31,32], we define three main process sub-phases in which **Elements** in Activities can be contained:

- **Preproduction**, containing activities like *brainstorming, prototyping, validation* and tasks to find the "fun factor";
- **Production**, where technical tasks occur;
- **Post-production**, containing any activities executed after the game has been launched.

[4] https://www.gamasutra.com/.

```
# Slow Down, Bull
## team
* team
  * Besides me, the team for Slow Down, Bull was
      composed entirely of contractors (though some
      still had some (...)
## activities
* **initial prototyping**
  * Because the whole initial process was a bit of an
      experiment, we spent a long time with just me
      working on (...)
## characteristics
* expertise source
  * In a way, the act of consulting an expert became a
      form of delegation, and pure brain-expertise
      became a (...)
## feedback
* I would switch the position of initial prototyping and
     early play testing. The prototype had to exist
    before (...)
```

Listing 1. Example of a map data-structure used to store data from post-mortems analysis.

Each Element can have one of two possible status: **Normal**, when the postmortem author does not report any problem regarding the activity execution, and **Problematic**, when developers encountered some problems related to the activity.

Each element has an index name, which we call **Key**, as well as a description, called **Value**. We put the postmortem extracted data in a map data-structure in which the *key* is an element name and the *value* is a quotation gathered from postmortems. Listing 1 shows an example of data extracted from postmortems and stored in the map data-structure.

For ease of conveying information from the processes to developers, we defined a process structure template file with the elements of process described earlier to generate a graphical representation of the processes. Fig. 4 shows the elements.

Finally, to improve the reliability of the **extracted processes**, we showed these processes to the postmortem authors and game developers, asking for feedback about these processes. We showed the extracted process and ask the following questions:

1. "Is this process similar of what you used when developing your game?"
2. "What are the elements that does not make sense?"
3. "Would you add something important?"

With their answers, we modified the extracted process according to their feedbacks. This step helps us to ensure that the extracted processes are as close as possible to the development processes that they used during the video game development.

3.2. Capturing context

We encoded the data gathered from postmortems in JSON format and put it in a database to perform queries. The JSON structure consists of five *key:value* for each process element. A process is thus a collection of *N* elements. Listing 2 shows an example of the JSON format.

Although the data is structured, it is not normalized. Some elements

```
{
  "game" : "Slow Down, Bull",
  "phase" : "activities",
  "element" : "exploration phase",
  "desc" : "We were able to iterate through a ton of
      different experiments, many of which were
      discarded failures, but which paved the path for
      the strongest mechanics in the game",
  "prob" : false
}
```

Listing 2. JSON structure used to storage process' elements.

Table 1
Video Game context variables.

#	Group	Description
v01	Activities	Agile
v02		Prototyping
v03		Performance Optimization
v04		Tools Development
v05		Outsourcing: Assets
v06		Outsourcing: Work
v07		Pre-Production: Short
v08		Pre-Production: Long
v09		Post-Production: Normal
v10		Post-Production: Heavy
v11		Reuse: Code
v12		Reuse: Assets
v13		Testing: In-Ouse Qa Team
v14		Testing: Closed Beta
v15		Testing: Open Beta
v16		Testing: Early Access
v17		Marketing/Pr: Self
v18		Marketing/Pr: Outsourced
v19	Team	Size: $< =5$
v20		Size: $5 - 25$
v21		Size: > 25
v22		Type: Single
v23		Type: Collaborative
v24		Distributed
v25	Management	Developer Type: First-Party
v26		Developer Type: Second-Party
v27		Developer Type: Third-Party
v28		Indie
v29		Funding: External
v30		Funding: Self
v31		Funding: Crowdfunding
v32		Publisher: External
v33		Publisher: Self (same Developer)
v34	Technical	Intelectual Property: Port
v35		Intelectual Property: Remaster / Reboot
v36		Intelectual Property: Franchise Sequence
v37		Intelectual Property: Expansion
v38		Intelectual Property: Mod
v39		Intelectual Property: New Ip
v40		Engine: In-House (new)
v41		Engine: In-House Ready
v42		Engine: Off-The-Shelf
v43	Platform	Console: Microsoft
v44		Console: Sony
v45		Console: Nintendo
v46		Pc: Windows
v47		Pc: Mac
v48		Pc: Linux
v49		Mobile: Android
v50		Mobile: Ios
v51	Design	Genre: Action
v52		Genre: Action-Adventure
v53		Genre: Adventure
v54		Genre: Role-Playing
v55		Genre: Simulation
v56		Genre: Strategy
v57		Genre: Puzzle
v58		Genre: Sports
v59		Mode: Single-Player
v60		Mode: Multi-Player (offline)
v61		Mode: Multi-Player Online

have the same meaning and, therefore, must be merged to a common description. We create an **abstraction dictionary** that contains all the elements and their respective indexes. For example, the elements *"local play testing"* and *"beta testing"*, refer to the activity *"testing"*.

Lastly, we must define the variables that will be used by the algorithm as project context. A project context defines its characteristics, which are the variables used by the algorithm. There exist some attempts to contextualize non-video game software projects [25,26]. However, the sets of variables are larger in video game projects than in other projects because of their multidisciplinarity.

We divide the **video game context** into six categories, totalizing 61 variables: *activities, team, management, technical, platform*, and *design*. Table 1 shows all the variables of the video game context.

3.3. Producing recommendations

The project contexts obtained from different postmortems are input for *Principal Component Analysis* (PCA), discussed in Section 2.5, to identify and quantify the relations in large sets of variables. Principal Component Analysis (PCA) was used in this research because it was well suited to the problem and its simplicity, enabling a quick prototyping. Others techniques were used, like *Decision Trees* and *K-Means* algorithms. However, given the *labels* restrictions, we changed the approach.

The result of applying PCA on the project context is an array of similar projects. With this list, we query the database for all elements related to the projects and perform a merge, respecting the elements' phase. Because we stored the process' elements without its relationship, that is, flow between the activities, after put all similar elements together, we had to allocate the elements manually.

3.4. Presenting the recommendations

The last step is to present the recommended activities: the development process. We choose to display recommended processes graphically, using the same graphical representation used for the extracted processes. Consequently, we use the same visualization template (see Fig. 4).

4. Recommendation system development

We now present the recommendation system and how we built it. We followed the same four steps as in Section 3.

4.1. Data preprocessing

We used a Web scrapping tool to gather postmortems from *Gamasutra* automatically. *Gamasutra* was our source of information of choice because of the size of its postmortems database and its reliability. We check and filtered data, extracting the details of the video game projects, like game title, author(s), release date, url, as well as data from *Wikipedia*, like game mode, genre, developers, engine, etc. The total amount of game projects was 234 from 1997 (which is the oldest postmortem we found) to 2016.

Starting from the most recent postmortem, we analyzed and extracted the data to populate our JSON database. To analyze the postmortems we made use of *Mendeley* and its feature to create notes. Every information regarding the development process was highlighted and tagged with an index. For example, Fig. 5 shows a piece of a postmortem in which we highlighted some excerpts and associated a index with each one, like *"initial prototyping"*.

Some postmortems do not have useful data and, thus, among the 100 postmortems that we analyzed we kept only 55 that had sufficient information to create a development process. The other postmortems mainly focused on game designs or gave too little details about team and the project itself.

We use *GraphViz* to display processes extracted from the notes graphically. *GraphViz* uses a special text format to generate graphical files, which can be saved as PDF and PNG files. We created a script that query the database, which contains the process elements, and generate a new file, using the process template (see Fig. 4), in DOT syntax, which can be converted into an image.

We requested feedback from the developers, authors of the postmortems, about the extracted processes. We sent a on-line form asking about the completeness and viability of the processes. A total of 20 forms were sent with 7 answers (33% answer rate). We refactored few

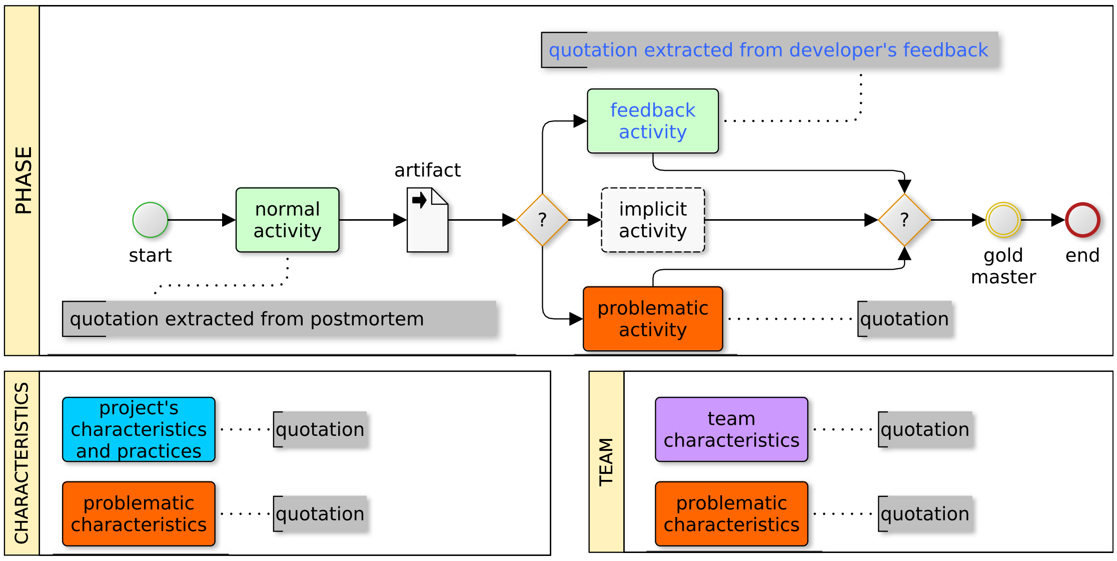
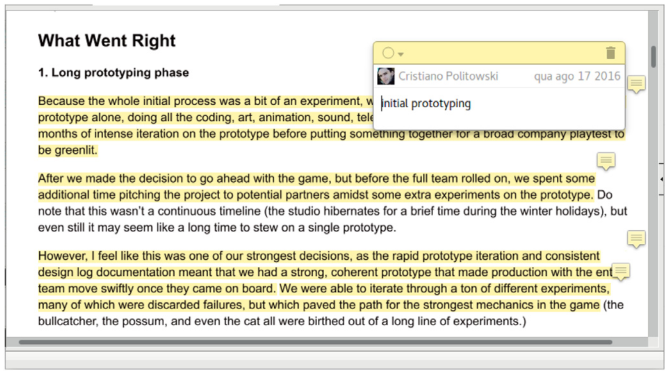


Fig. 4. We use a visualization template (BPMN) to display uniformly processes graphically.



What Went Right

1. Long prototyping phase

Because the whole initial process was a bit of an experiment, w prototype alone, doing all the coding, art, animation, sound, tele months of intense iteration on the prototype before putting something together for a broad company playtest to be greenlit.

Cristiano Politowski qua ago 17 2016

initial prototyping

After we made the decision to go ahead with the game, but before the full team rolled on, we spent some additional time pitching the project to potential partners amidst some extra experiments on the prototype. Do note that this wasn't a continuous timeline (the studio hibernates for a brief time during the winter holidays), but even still it may seem like a long time to stew on a single prototype.

However, I feel like this was one of our strongest decisions, as the rapid prototype iteration and consistent design log documentation meant that we had a strong, coherent prototype that made production with the ent team move swiftly once they came on board. We were able to iterate through a ton of different experiments, many of which were discarded failures, but which paved the path for the strongest mechanics in the game (the bullcatcher, the possum, and even the cat all were birthed out of a long line of experiments.)

Fig. 5. Example of the postmortem analysis made to extract and index the relevant data about the video games.Example of the postmortem analysis made to extract and index the relevant data about the video games.

wrong or misplaced elements in some of the processes. All of the answers agreed with the extracted processes, for example in the citation of the postmortem author, developer of the game "Prune":

> *"Yeah, that's about right. I would maybe call the first 6 months of work on the game "preproduction", which included things like prototyping, task prioritization, and play testing."*

Fig. 6 shows an example of an extracted process, validated by the authors of the origin postmortem, of the game project *Slow down, Bull*[5]. This process shows two phases: preproduction and production, without post-production. Each phase has a start and an end date, demarcated by the dark discs. In both phases, the elements in green are activities that occurred naturally during the development while the red ones encountered problems.

Considering that we extracted process elements and, normally, they do not had chronological information about its execution, we cannot, precisely describe the process flow. In these cases, we marked such elements with two question marks ("?"). In production, the element described as "gold" indicates the final version of the game, the complete game. Each activity, in the gray boxes, has a direct related to the postmortem. There are two frames containing the characteristics and general practices together with the ones related to the development team.

4.2. Capturing context

We normalized all the elements keys using the **abstraction dictionary** and inserted the elements in the database. We chose *MongoDB*[6] as storage engine because it was easy to import JSON files. We stored a total of **913 different elements** in the database from **55 projects**.

With the data ready, we now needed a *project context* (see Table 1). We analyzed the project data and, according to their characteristics, manually build contexts. The context values (see Table 2) are composed of 61 variables ([v01..vN]) from the 55 extracted process (game projects, [g1..gN]). Every time a new project is added to the database, it helps in future recommendations.

4.3. Producing recommendations

With the projects contexts, we ran a script that uses *PCA biplot* to show the samples' variability through a graph, that is, a graph showing the similarity of the 55 projects analyzed. Fig. 7 shows the *biplot PCA*, which is a representation of the correlation between the samples. The closest the dots, the more similar the projects. The input project context is from the game *Slow down, Bull*, presented in red; the green dots show projects that are more similar, such as *Jetpack High, Vanishing Point*, and *Catlateral Damage*.

With a list of similar projects, we queried the database for their elements. We then joined the elements resulting in a new process. We assembled the elements in a process manually, because we did not stored the processes flows.

4.4. Presenting the recommendations

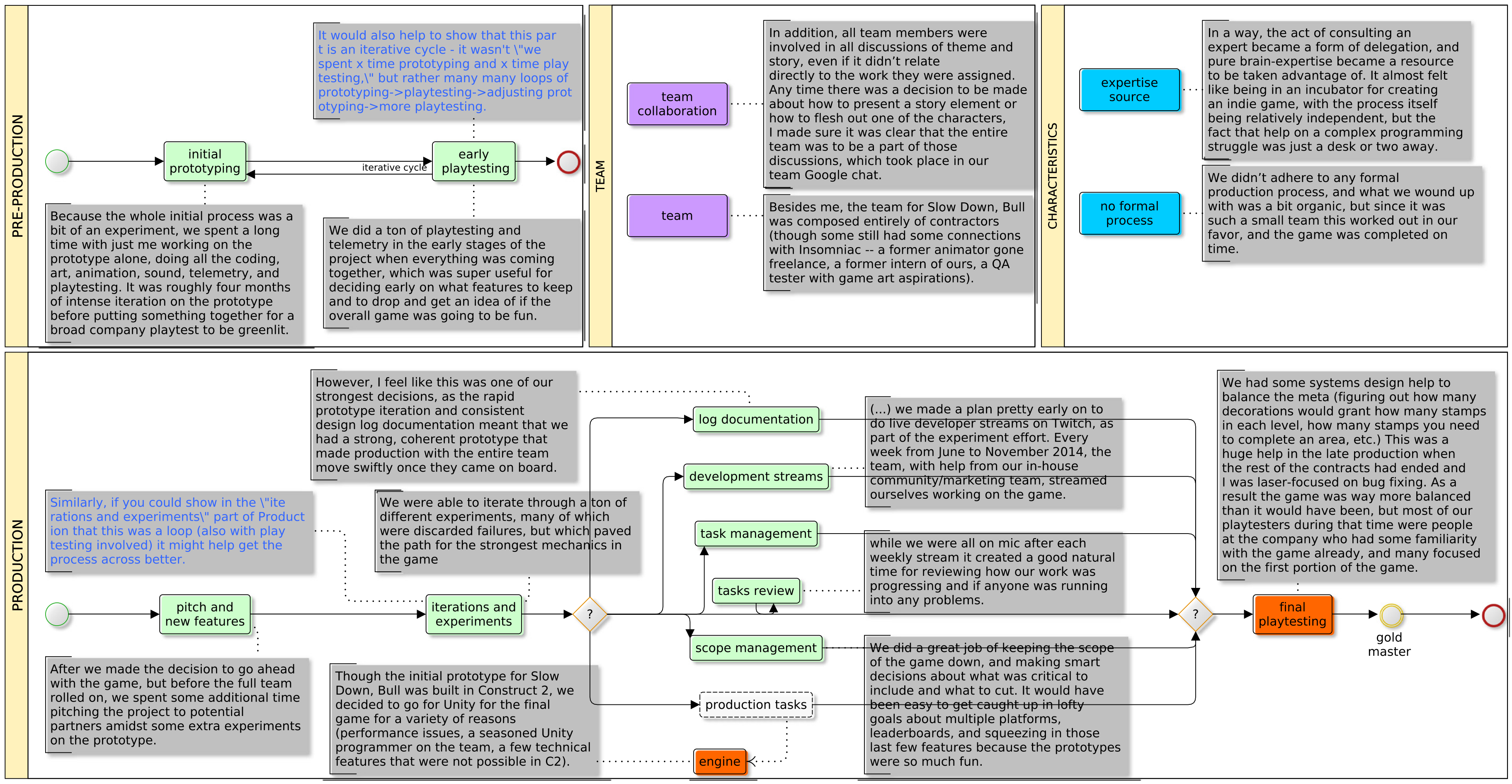
We used again *Graphiz* with the visualization template to display the new recommended process (see Fig. 4). We converted the new recommended process in a textual DOT file, which can be compiled into an image file.

Fig. 8 shows an example of a recommended process and its preproduction phase while Fig. 9 shows the production phase using the game context of the game *Slow down, Bull* shown in Table 3.

Fig. 8 shows the preproduction and has the activities: *requirements and constraints, exploration phase, planning documentation*, and *milestones planning* linked with quotations from the game *Vanishing Point;*

[5] To save space, we manually redesigned the process, respecting the template rules.

[6] https://www.mongodb.com/.



It would also help to show that this par t is an iterative cycle - it wasn't \"we spent x time prototyping and x time play testing,\" but rather many many loops of prototyping->playtesting->adjusting prot otyping->more playtesting.

Because the whole initial process was a bit of an experiment, we spent a long time with just me working on the prototype alone, doing all the coding, art, animation, sound, telemetry, and playtesting. It was roughly four months of intense iteration on the prototype before putting something together for a broad company playtest to be greenlit.

We did a ton of playtesting and telemetry in the early stages of the project when everything was coming together, which was super useful for deciding early on what features to keep and to drop and get an idea of if the overall game was going to be fun.

In addition, all team members were involved in all discussions of theme and story, even if it didn’t relate directly to the work they were assigned. Any time there was a decision to be made about how to present a story element or how to flesh out one of the characters, I made sure it was clear that the entire team was to be a part of those discussions, which took place in our team Google chat.

Besides me, the team for Slow Down, Bull was composed entirely of contractors (though some still had some connections with Insomniac -- a former animator gone freelance, a former intern of ours, a QA tester with game art aspirations).

In a way, the act of consulting an expert became a form of delegation, and pure brain-expertise became a resource to be taken advantage of. It almost felt like being in an incubator for creating an indie game, with the process itself being relatively independent, but the fact that help on a complex programming struggle was just a desk or two away.

We didn’t adhere to any formal production process, and what we wound up with was a bit organic, but since it was such a small team this worked out in our favor, and the game was completed on time.

Similarly, if you could show in the \"ite rations and experiments\" part of Product ion that this was a loop (also with play testing involved) it might help get the process across better.

After we made the decision to go ahead with the game, but before the full team rolled on, we spent some additional time pitching the project to potential partners amidst some extra experiments on the prototype.

However, I feel like this was one of our strongest decisions, as the rapid prototype iteration and consistent design log documentation meant that we had a strong, coherent prototype that made production with the entire team move swiftly once they came on board.

We were able to iterate through a ton of different experiments, many of which were discarded failures, but which paved the path for the strongest mechanics in the game

Though the initial prototype for Slow Down, Bull was built in Construct 2, we decided to go for Unity for the final game for a variety of reasons (performance issues, a seasoned Unity programmer on the team, a few technical features that were not possible in C2).

(...) we made a plan pretty early on to do live developer streams on Twitch, as part of the experiment effort. Every week from June to November 2014, the team, with help from our in-house community/marketing team, streamed ourselves working on the game.

while we were all on mic after each weekly stream it created a good natural time for reviewing how our work was progressing and if anyone was running into any problems.

We did a great job of keeping the scope of the game down, and making smart decisions about what was critical to include and what to cut. It would have been easy to get caught up in lofty goals about multiple platforms, leaderboards, and squeezing in those last few features because the prototypes were so much fun.

We had some systems design help to balance the meta (figuring out how many decorations would grant how many stamps in each level, how many stamps you need to complete an area, etc.) This was a huge help in the late production when the rest of the contracts had ended and I was laser-focused on bug fixing. As a result the game was way more balanced than it would have been, but most of our playtesters during that time were people at the company who had some familiarity with the game already, and many focused on the first portion of the game.

Fig. 6. Extracted development process from *Slow down bull* game.

Table 2
Context table filled (partialy).

	g1	g2	g3	g4	g5	g6	g7	g8	g9	gN
v01	1	1	1	1	1	1	1	1	1	…
v02	0	1	1	1	1	1	0	0	1	…
v03	0	0	1	0	0	0	0	0	0	…
v04	0	0	0	0	0	0	0	0	0	…
v05	0	0	0	0	0	0	1	0	0	…
v06	1	0	0	0	0	1	0	0	1	…
v07	0	1	1	1	1	1	0	1	1	…
v08	1	0	0	0	0	0	1	0	0	…
v09	1	1	0	0	1	1	0	1	1	…
vN	…	…	…	…	…	…	…	…	…	…

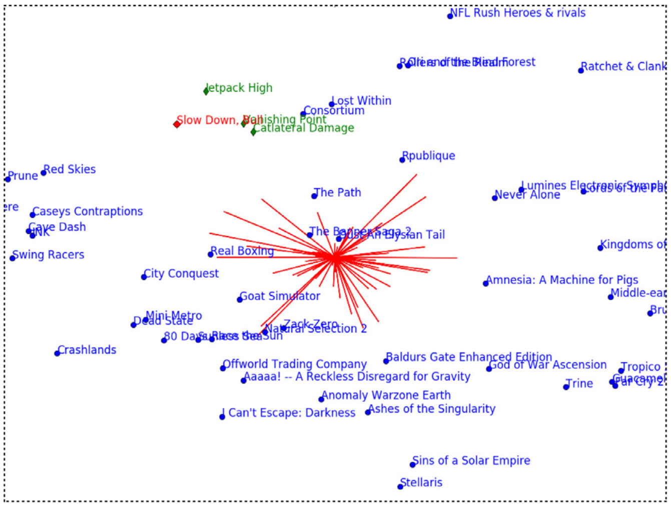


Fig. 7. PCA biplot using the game *Slow down, Bull* as input project context.

prototyping is linked to the games *Catlateral Damage* and *Jetpack High*.

Fig. 9 shows the production phase. The process flow starts with an activity to choose a feature or a bug, depending on the iteration type (component delivery or fixing a problem). As input, this activity receives a list of game properties from the preproduction phase. This list is similar to *sprint backlog* used in agile methods like *Scrum* [19].

The process flow continues with the *development iterations loop*. This element *development iterations loop* marks the beginning of the productions activities. Two quotes, gathered from the postmortems of games *Vanishing Point* and *Catlateral Damage* are highlighted. The premise is similar to *sprints* in agile methods in which the process occurs in short cycles with continuous deliveries of complete features. At the end of each iteration, a new, complete component must be aggregated to the existent game. The artifact *feature delivery* represents this component.

The technical activities happen inside each iteration. There are four described activities, all of them connected with at least one citation from a game: *polish and refinements, meetings, refactoring the development,* and *design tasks*.

According to the development flow of the generated process, after the delivery loop of components, the *testing* activity will start, which is crucial to the success or failure of a game. This activity may happen in many ways, for example, the team can have a play testing laboratory with users, a specialized testing team, automated tests, users testing, beta testing, among others. Only one project, *Vanishing Point* mentions explicitly testing.

After the end of the iterations, with the game tested, the flow arrives on a condition to test whether all the iterations are completed. If positive, the next activity is *quality assurance* that suggests to have the game reviewed by an external entity, either a publisher or a platform (*Sony, Steam, Microsoft*) to identify possible nonconformities with standards required by such publisher or platform. Then, the final *build* is realized, fulfilling the last requirements, to obtain and publish a *gold master*.

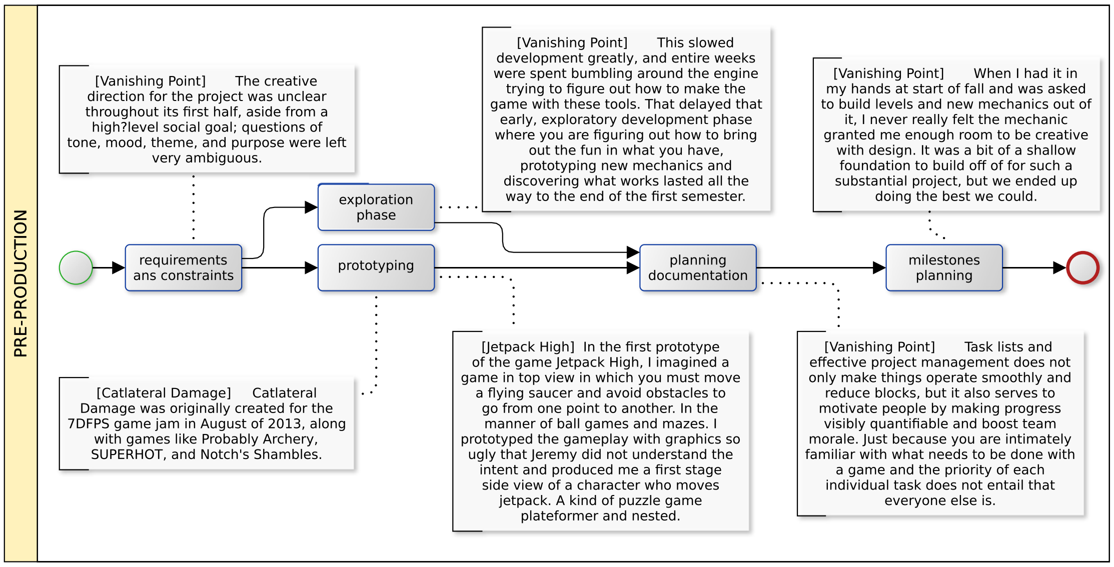
5. Evaluation

To assess our recommendation system, we performed three types of evaluations:

A. **Quantitative**, by measuring Correctness and Coverage of the recommended elements in Sections 5.1;
B. **Qualitative**, by asking feedback from the game developers on the recommended process in Sections 5.2;
C. **Case study**, by applying a recommended process in a real world project in Sections 5.3.

5.1. Quantitative evaluation

To quantitatively assess the recommendation system, we chose four projects which we extracted from postmortems and validated it with



[Vanishing Point] The creative direction for the project was unclear throughout its first half, aside from a high?level social goal; questions of tone, mood, theme, and purpose were left very ambiguous.

[Vanishing Point] This slowed development greatly, and entire weeks were spent bumbling around the engine trying to figure out how to make the game with these tools. That delayed that early, exploratory development phase where you are figuring out how to bring out the fun in what you have, prototyping new mechanics and discovering what works lasted all the way to the end of the first semester.

[Vanishing Point] When I had it in my hands at start of fall and was asked to build levels and new mechanics out of it, I never really felt the mechanic granted me enough room to be creative with design. It was a bit of a shallow foundation to build off of for such a substantial project, but we ended up doing the best we could.

[Catlateral Damage] Catlateral Damage was originally created for the 7DFPS game jam in August of 2013, along with games like Probably Archery, SUPERHOT, and Notch's Shambles.

[Jetpack High] In the first prototype of the game Jetpack High, I imagined a game in top view in which you must move a flying saucer and avoid obstacles to go from one point to another. In the manner of ball games and mazes. I prototyped the gameplay with graphics so ugly that Jeremy did not understand the intent and produced me a first stage side view of a character who moves jetpack. A kind of puzzle game plateformer and nested.

[Vanishing Point] Task lists and effective project management does not only make things operate smoothly and reduce blocks, but it also serves to motivate people by making progress visibly quantifiable and boost team morale. Just because you are intimately familiar with what needs to be done with a game and the priority of each individual task does not entail that everyone else is.

Fig. 8. Preproduction phase of the recommended process for the game *Slow down, Bull.*

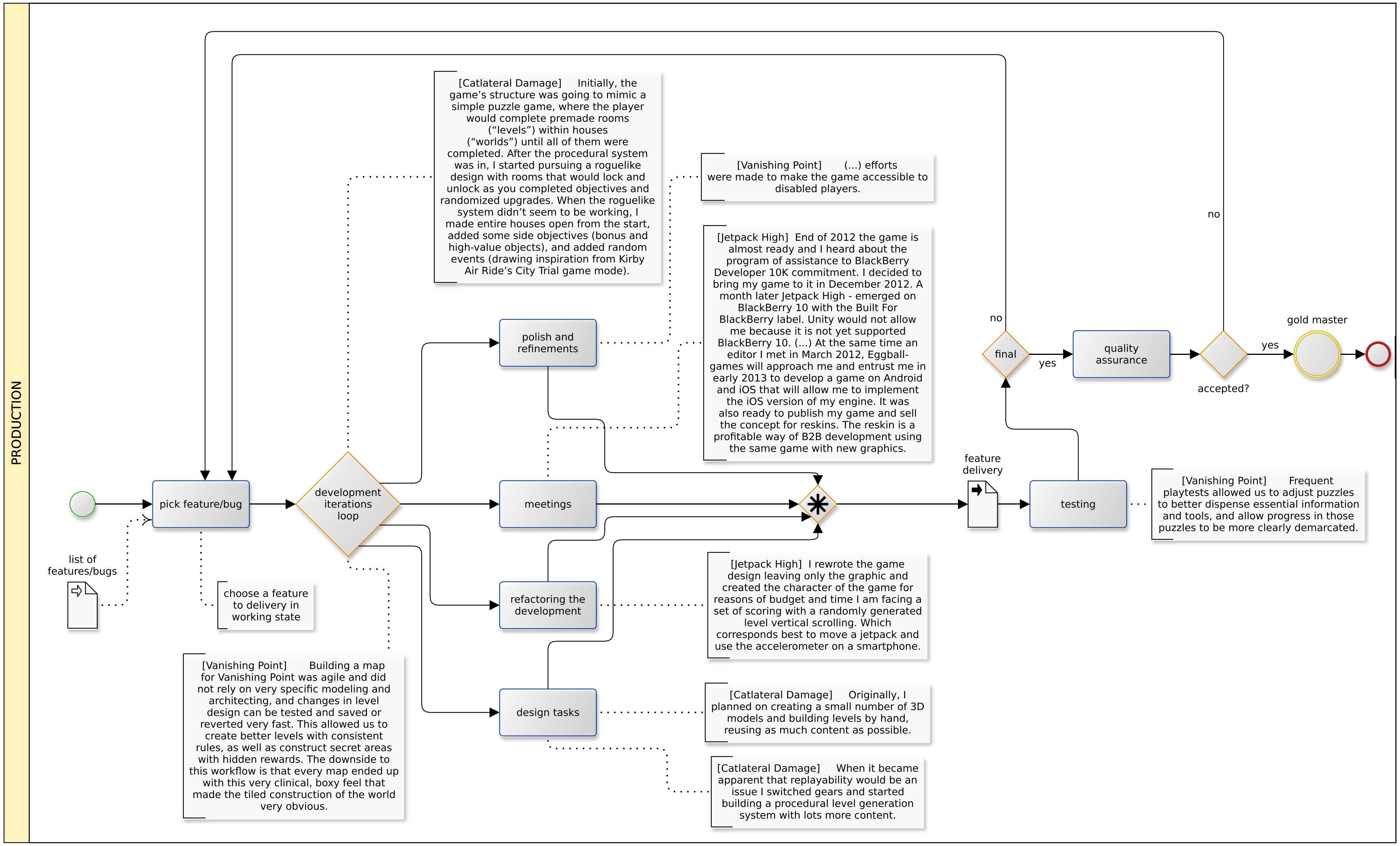

PRODUCTION
list of features/bugs
pick feature/bug
choose a feature to delivery in working state
development iterations loop
polish and refinements
meetings
refactoring the development
design tasks
feature delivery
testing
final
no
yes
quality assurance
accepted?
no
yes
gold master
[Catlateral Damage] Initially, the game's structure was going to mimic a simple puzzle game, where the player would complete premade rooms ("levels") within houses ("worlds") until all of them were completed. After the procedural system was in, I started pursuing a roguelike design with rooms that would lock and unlock as you completed objectives and randomized upgrades. When the roguelike system didn't seem to be working, I made entire houses open from the start, added some side objectives (bonus and high-value objects), and added random events (drawing inspiration from Kirby Air Ride's City Trial game mode).
[Vanishing Point] (...) efforts were made to make the game accessible to disabled players.
[Jetpack High] End of 2012 the game is almost ready and I heard about the program of assistance to BlackBerry Developer 10K commitment. I decided to bring my game to it in December 2012. A month later Jetpack High - emerged on BlackBerry 10 with the Built For BlackBerry label. Unity would not allow me because it is not yet supported BlackBerry 10. (...) At the same time an editor I met in March 2012, Eggball-games will approach me and entrust me in early 2013 to develop a game on Android and iOS that will allow me to implement the iOS version of my engine. It was also ready to publish my game and sell the concept for reskins. The reskin is a profitable way of B2B development using the same game with new graphics.
[Vanishing Point] Frequent playtests allowed us to adjust puzzles to better dispense essential information and tools, and allow progress in those puzzles to be more clearly demarcated.
[Jetpack High] I rewrote the game design leaving only the graphic and created the character of the game for reasons of budget and time I am facing a set of scoring with a randomly generated level vertical scrolling. Which corresponds best to move a jetpack and use the accelerometer on a smartphone.
[Catlateral Damage] Originally, I planned on creating a small number of 3D models and building levels by hand, reusing as much content as possible.
[Catlateral Damage] When it became apparent that replayability would be an issue I switched gears and started building a procedural level generation system with lots more content.
[Vanishing Point] Building a map for Vanishing Point was agile and did not rely on very specific modeling and architecting, and changes in level design can be tested and saved or reverted very fast. This allowed us to create better levels with consistent rules, as well as construct secret areas with hidden rewards. The downside to this workflow is that every map ended up with this very clinical, boxy feel that made the tiled construction of the world very obvious.

Fig. 9. Recommended process production phase for the game *Slow down, Bull*.

Table 3
Project context of the game *Slow down, Bull*. The *false* values were omitted.

#	Context	g10 values
p01	Agile	1
p02	Prototyping	1
p06	Outsourcing: Work	1
p08	Pre-Production: Long	1
p09	Post-Production: Normal	1
p13	Testing: In-Ouse Qa Team	1
p19	Size: < =5	1
p22	Type: Single	1
p24	Distributed	1
p27	Developer Type: Third-Party	1
p28	Indie	1
p29	Funding: External	1
p33	Publisher: Self (same Developer)	1
p39	Intelectual Property: New Ip	1
p42	Engine: Off-The-Shelf	1
p46	Pc: Windows	1
p57	Genre: Puzzle	1
p59	Mode: Single-Player	1

Table 4
Recommendation-centric dimensions.

Dimensions	Description
Correctness	How close are the recommendations to a set of recommendations that are assumed to be correct?
Coverage	To what extent does the recommendation system cover a set of items or user space?
Diversity	How diverse (dissimilar) are the recommended items in a list?
Confidence	How confident is the recommendation system in its recommendations?

Table 5
Confusion matrix used to calculate correctness [33].

	Recommended	Not Recommended
Used	True Positives (TP)	False Negatives (FN)
Not Used	False Positives (FP)	True Negatives (TN)

their respective developers. We extracted their contexts and generated processes using our recommendation system. We compared if each recommended element is present in the extracted (validated) process. We used four measures to asses our recommendation system quantitatively [33], shown in Table 4.

Recommendation-centric dimensions primarily assess the recommendations generated by the recommendation system itself: their coverage, correctness, diversity, and level of confidence [33]. We chose two metrics to evaluate our recommendation system: **Correctness** and **Coverage**. *Diversity* does not apply to our system because our dataset is composed by unique elements: each element stored in the database is unique. Also, *confidence* requires observable variables with users and we received feedback from developers, as in Section 5.2.

5.1.1. Correctness metric

The objective of the correctness metric is to assess how useful are the results without being overwhelming with unwanted items [33]. With this metric, we measure **how close the recommended process is compared to the expected, ideal process**, and already validated process.

There are different means of evaluating the recommended processes, like *Predicting User Ratings* and *Ranking Items*. However, we chose **Recommending Interesting Items**, which tests if the recommendation system provide process elements that users may like to use.

We fed the recommender system with data from a project context which project we already validated, a process extracted from a post-mortem verified by its authors and–or developers. The confusion matrix shown in Table 5 compares each process element of the **recommended process** test set with the elements from the **extracted process** (training set).

For example, consider a recommended process with an element *team* and an extracted process that contains this element, we have a true positive (TP) element. Otherwise, if the recommended element is not contained in the extracted process, then we have a false positive (FN) element. Then, we benchmarked our recommendation system by calculating some metrics[7]:

- **Precision**: the percentage of the elements predicted to be relevant that are actually relevant.

$$precision = \frac{TP}{TP + FP}$$

- **Recall**: the percentage of the elements that are actually relevant that are predicted to be relevant.

$$recall = \frac{TP}{TP + FN}$$

- **Accuracy**: the percentage of the elements available that are either correctly recommended or correctly not recommended.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- **False positive rate**: the percentage of the irrelevant elements that are predicted to be relevant.

$$falsePositiveRate = \frac{FP}{FP + TN}$$

- **False negative rate**: the percentage of the elements predicted to be irrelevant that are actually relevant.

$$falseNegativeRate = \frac{FN}{FN + TN}$$

- **Specificity**: a heuristic measure [34] of the likelihood of the relevance of an element given its relation to other elements, some of which are known to be relevant.

$$specificity = \frac{TN}{FP + TN}$$

- **F-measure**: a measure combining precision and recall.

$$F - measure = 2x\frac{precisionXrecall}{precision + recall}$$

5.1.2. Coverage metric

Coverage refers to the proportion of available information for which recommendations can be made. It can be calculated as catalog coverage (about elements) or prediction coverage (about developers) [33]. We use the former because we want to assess the recommended processes. This metric can be calculated as the proportion of elements (process elements) that we can be recommended:

[7] Except Specificity, which is a heuristic measure, all other values are in [0..1].

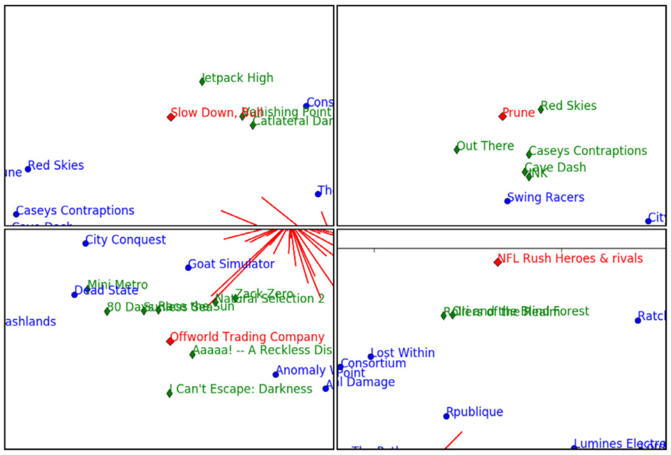


Fig. 10. *Biplots PCAs* from analyzed projects showing their similarity. From left to right: #1, #2, #3, #4.

$$catalogCoverage = \frac{|Sr|}{|Sa|}$$

The elements available for recommendation may not all be of interest to a developer so we used a previous approach [35], which uses weighted catalog coverage for balancing the decrease in coverage by usefulness for developers:

$$weightedCatalogCoverage = \frac{|Sr \cap Ss|}{|Ss|}$$

where:

- *Sr*: is set of process elements recommended;
- *Sa*: is the set of all available elements;
- *Ss*: is the set of elements useful to developers (validated elements).

We computed correctness and coverage of the recommendation system by comparing the similarity between the **extracted** processes, validated by their authors, and the **recommended** processes, generated by the recommendation system. We used four different project contexts, resulting in an array of similar projects. Fig. 10 shows the resulting *byplots*.

1. The game project **Prune** is similar to *Red Skies, Out There, INK, Cave Dash, Caseys Contraptions*;
2. The game project **Offworld Trading Company** is similar to *Mini Metro, Sunless Sea, Race the Sun, I Cant Escape: Darkness, 80 Days, Natural Selection 2, Aaaaa! – A Reckless Disregard for Gravity, Zack Zero*;
3. The game project **NFL Rush Heroes & rivals** is similar to *Ori and the Blind Forest, Rollers of the Realm*;
4. The game project **Slow Down, Bull** is similar to *Catlateral Damage, Jetpack High, Vanishing Point.*

We created a script to compare each element from each recommended process, simplified in the pseudo-code listed in Algorithm 1. We compared if the recommended elements were in the extracted processes, which we considered as correct, because they were reviewed by the postmortems authors. If the elements recommended by the recommendation system were in the extracted processes, then they received a *True Positive (TP)*. Otherwise, if the recommended elements were not part of the extracted processes, then they received a *False Positive (FP)*[8]. If process elements were in the extracted processes but

```
for $gameTested \in gamesTested[]$ do
    $similarGames[] \leftarrow getSimilarGamesPCA(gameTested)$
    $elemGameTested[] \leftarrow getAllElements(gameTeste)$
    $elemRecommended[] \leftarrow getAllElements(similarGames)$
    $elemNotRecommended[] \leftarrow getAllElementsExcept(elemGameTested + elemRecommended)$
if $elemRecommended[i] \in elemGameTested[]$ then
    $truePositive[] \leftarrow elemRecommended[i]$
else
    $falsePositive[] \leftarrow elemRecommended[i]$
end if
if $elemNotRecommended[i] \in elemGameTested[]$ then
    $falseNegative[] \leftarrow elemNotRecommended[i]$
else
    $trueNegative[] \leftarrow elemNotRecommended[i]$
end if
end for
```

Algorithm 1. Pseudo-code to compare the process elements.

[8] Even if an element is not part of the extracted process, it still can make sense and be useful for the developers [36].

Table 6
Confusion matrix with aggregated data.

	T. Positive	F. Positive	F. Negative	T. Negative
#1	32	44	285	540
#2	23	105	153	620
#3	10	32	171	692
#4	14	19	358	505

not recommended, they received a *False Negative (FN)*. Finally, if process elements were not part of the recommended process nor in the extracted process, then they received *True Negative (TN)*.

Table 6 shows the confusion matrix with aggregated data and Table 7 shows the metrics results. The high number of *false negatives* and low number of *true positives* can be improved by increasing the range of similarity: by adding more similar projects. Adding more projects would also increase *recall*, given that recall was low. However, allowing for a longer list of recommendations improves *recall* but is likely to reduce *precision* and improving *precision* may decrease recall [37].

Although *accuracy* and *specificity* rates were good, all others values were low. The recommendation system reached a precision above 40% in two projects. We used *PCA* to find similarity and will consider other algorithms in future work. We can also improve our recommendation system by improving the context variables and samples number.

Table 7 shows the *coverage* metric results with the low values in *Catalog* and *Weighted Catalog*, which could be improved if more elements where recommended. Increasing *coverage* may increase *correctness* as well.

5.2. Qualitative evaluation

We performed the qualitative validation using a set of questions in a on-line form that we sent to project authors and–or developers. For each one of the chosen project, that is, project #1 to #4 already used in the quantitative validation, we asked recipients to analyze both the **extracted** and the **recommended** processes. We also asked them to highlight the elements that were wrong or misplaced. Then, we asked then about the viability of the recommended process:

1. "In general, is this new workflow similar to what you used developing the game?"
2. "Which process elements does not make sense?"
3. "Do you add something important to this workflow?"
4. "If you began a new video game project similar to [GAME], what would be the feasibility of using this new process? (answer 5 to "more feasible" and 1 to "unfeasible")"

In general, **all respondents agreed with the recommended processes**. They found that the processes were similar to the processes that they used during their game development. One developer stated that the activities flow is similar to what was actually done:

"Yes, this workflow looks similar to what we did with *[game title]*, in that we did a lot of iteration with our feature development."

Table 8
Dimensions: User-centric.

Dimensions	Description
Trustworthiness	How trustworthy are the recommendations?
Novelty	How successful is the recommendation system in recommending items that are new or unknown to users?
Serendipity	To what extent has the system succeeded in providing surprising yet beneficial recommendations?
Utility	What is the value gained from this recommendation for users?
Risk	How much user risk is associated in accepting each recommendation?

In another example, the postmortem author and game developer answered about the generated process viability, stating that the most useful contribution may be during the project planning, but not for creation of any production pipeline.

"(...) this might be a useful thing to look at in the beginning, but I would not use it to create a production pipeline because the circumstances have almost assuredly changed since."

Still, the postmortem' author pointed out that the recommended process is interesting for analysis, but not something to be used practically to build a set of production activities. The developers claimed that each project demands a different process based on the game design choice.

"I think the tool that you describe has some really interesting implications from an analytical standpoint, but it is not something I would use in a practical sense, like I wouldn't use it to develop processes or pipelines from. In practice, every game requires different processes to make, and in fact the process used to make a game is as much a part of its design as the actual game. (...) The reasons for variation in process are vast and constantly changing. Even in big studios you tend to end up with processes that are perfect for making the game you were working on the year before (...) However, I still think it is valid work from an analytical standpoint, like as something to consider when having the conversation about dev process for the next game."

5.3. Case study

We performed a case study using qualitative metrics from user-centric dimensions to assess if the recommendation system fulfills the needs of developers [33]. We use the metrics in Table 8.

We interviewed with an team that is currently developing a game. We asked questions to measure these user-centric dimensions. Firstly, we gathered the project context and generated a new process using the recommendations system. Then, we presented the recommended processes to the team members and asked them to answer the following statements using a five-point Likert scale:

1. "(Trustworthiness) The recommendation is similar compared to my project."
2. "(Novelty) The recommendation is new to me (regardless its usefulness)."

Table 7
Correctness and coverage results.

	Correctness							Coverage				
	Precision	Recall	Accuracy	FP Rate	FN Rate	Specificity	F-Measure	Sr	Sa	Ss	Catalog	W. Catalog
#1	42,11%	10,09%	63,49%	7,53%	34,55%	92,47%	16,28%	76	913	317	8,32%	10,09%
#2	17,97%	13,07%	71,37%	14,48%	19,79%	85,52%	15,13%	128	913	176	14,02%	13,07%
#3	23,81%	5,52%	77,57%	4,42%	19,81%	95,58%	8,97%	42	913	181	4,60%	5,52%
#4	42,42%	3,76%	57,92%	3,63%	41,48%	96,37%	6,91%	33	913	372	3,61%	3,76%

Table 9
Context values from the project used in case study. The *false* values were omitted.

#	Context	Values
v01	Agile	1
v02	Prototyping	1
v03	Performance Optimization	1
v04	Tools Development	1
v05	Outsourcing Assets	1
v07	Pre-Production Short	1
v11	Reuse Code	1
v12	Reuse Assets	1
v13	Testing In-Ouse Qa Team	1
v20	Size 5 - 25	1
v23	Type Collaborative	1
v27	Developer Type Third-Party	1
v30	Funding Self	1
v33	Publisher Self (same Developer)	1
v39	Intelectual Property New Ip	1
v42	Engine Off-The-Shelf	1
v46	Pc Windows	1
v55	Genre Simulation	1
v56	Genre Strategy	1
v59	Mode Single-Player	1

3. "(Serendipity) The recommendation is surprisingly good for my project."
4. "(Utility) The recommendation is useful to my project."
5. "(Risk) It would be risky to use the recommendation in my project (considering other practices already settled)."

We chose a simulation-like video game project that uses *Unity* engine and has been in development for more than three years. We asked the lead developers to fill the context form according to the project context, shown in Table 9.

We appended the context values in the context table, executed the recommendation system, and obtained a list of similar projects: *Ashes of the Singularity, Baldurs Gate Enhanced Edition, Natural Selection 2, Anomaly Warzone Earth and Aaaaa! – A Reckless Disregard for Gravity.*

A total of **31 elements** were recommended and so, for each one of them, we asked the development team to fill the statements, comparing the recommended elements with their actual activities. Table 10 shows the recommended elements and Fig. 11 shows the results.

The **Trustworthiness** metric had 50% of elements considered **similar** to what was used or applied in the development process. 28.13% of the elements were not considered similar while 21.88% were neutral.

test team *"We have a third party company who perform the tests before it reach the final client (...) the testers have access to the bug database."*

refactoring the development *"It happens often as the development adopt an evolutionary approach (...)"*

retrospective meeting *"(...) retrospective meetings occur regularly with development team and the final client."*

general team details *"These (recommendations) do not applied in our context: The team had not the proper knowledge in the beginning (...) nor*

Table 10
Recommended elements for the context used in the case study project.

Element	Description
test team	Small team with focused on development, not marketing
small team	Beta test group from community. Beta testers with access to bug database. Outsourcing test company to check devices from different platforms
general team details	Lead designer no present. Lack of artists. Experienced team who already worked together. No need to research, team have know-hall about tasks to complete. Full focused with no business meetings.
outsourcing	Outsourcing experienced people with technical expertise. Outsourcing PR / marketing. Outsourcing code and assets.
horizontal development	Follow a higher principle. Everyone may vet new ideas
development problems	Keep the pace and not "crunch" (work overtime) Lack of proper pipeline. Hard to solve things by yourself. Cutting or reworking features.
development process details	Lack of process or structure. Just code without planning. List of main features instead of a design document. Share the development with the audience. No schedule, nor milestones, nor meetings nor design document nor technical plan Development flow
engine and tools	Use a tool for build distribution. Learn everything from scratch.
infrastructure	Self made engine and–or tools. Engine and–or legacy code limiting improvements.
project focus	Focus on the team strength.
scope	Changing scope. Developing self engine.
concept	Artists, level designers, programmers, and animators working together. Set of design principles. Brainstorming ideas based in a goal. Align thinking before prototyping phase. Research similar games. Stick with the game concept since the beginning.
brainstorming features	Brainstorming ideas finding "fun factor".
business tasks	Market study. Selling directly to customers. Pre-order program. Marketing strategy
design tasks	Procedurally generated instead of hand craft levels. AI specialists.
development iterations loop	Digital distribution allowing testers give feedback quickly and often. Heavy tested by non designers. Schedule iterations with buffers, that is, double or triple the time required for a task. Heavily focused on iterations and constant improvements in the game. Legacy problems being solved as the development goes on.
exploration phase	Prototyping a game without polishing, trying to find a good game play. Heavily prototyping and testing iterations. Avoid making experiments in production phase.
in-house tools development	Creating tools to aid developers with a new technology and avoid unknown bugs. Tools allowing non coders to change parameters and experiment with the game engine. The main goal is to create a new engine. Creating an IDE. Creating an Engine with experienced team. Creating tools to automatize tasks and processes.
milestones planning	Pre-order program to raise money allowing the team continue the development. Early access feedback.
multi-player construction	Specialist in multi-player development.
pitch	Pitch the game concept.
planning documentation	Production plan with buffers (more time) after each milestone. Cutting features, making less but better.
polish and refinements	Improving performance
professional feedback	Getting technical feedback.
prototyping	Simple prototyping to find "fun factor". Game being developed in parallel with the engine.
refactoring the development	Time expended with re-designs. Refactoring old code.
requirements and constraints	Game design documentation not updated. Define a list of features and tools. Port UI to other platform.
retrospective meeting	Research target audience.
testing	Beta test group. Multi-platform testing. Delivery builds quickly to test team. Long time beta testing the game. Get feedback from QA with volunteers. Feedback from testers not used properly.
users feedback	Working directly with gamers community using a on-line message board. Game shipped for one platform first.
vertical slice	Community working together creating assets.

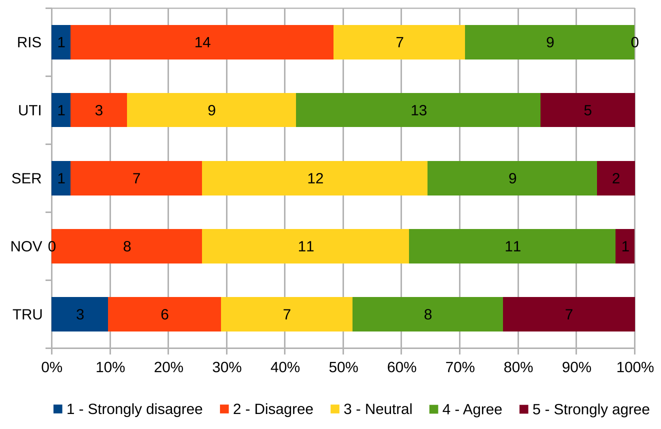


Fig. 11. Case study results.

had been worked together"

The **Novelty** metric had 40.63% of elements considered **new** by the development team. 34.38% were considered already known and 25.00% were neutral.

in-house tools development *"Although these practices were not used (by us) today (...) we found it a good surprise."*

The **Serendipity** metric had 37.50% of elements considered **surprisingly good** for the project. 25% were considered not good and 37.50% were neutral.

engine and tools *"We do not have a build tool (...) most of the techniques applied are learned from scratch."*

The **Utility** metric had 59.38% of elements considered **useful** to be used in the actual process. 12.50% were not considered good and 28.13% were neutral.

The **Risk** metric had 28.13% considered **risky** if added to the project. 46.88% were considered not risky and 25% were neutral. However, among these elements, none received a "5" as a answer.

The team cited a interesting point, as the project had already more than three years of development, many of the activities used were the results of "trials and errors". Using the recommendation system, some of these errors could have been avoided :

"This practices could be used since the beginning, for example, prototyping, test team, bug control, small team, building tools and do less but better."

Generally speaking, the team emphasized the following subset of activities does not applied to their project: user community, gathering resources, marketing, fun factor related practices and any activity multi-player. From the recommendation system, there were two details to consider. Firstly, some elements had details from more than one project, which were merged together. Therefore, a recommended element may contradict itself, showing a activity that makes another impossible. Secondly, the context has some particular elements not contained in the analyzed postmortems. For this reason, a few elements were not considered similar nor useful.

6. Discussion

The use of postmortems to analyze video game projects is not new but used with another goal it this article: to extract the processes used by developers. Even though the intent of postmortems is not to document processes, it is still possible to extract data relevant to the processes and, with many postmortems, assemble activities to create a flow of activities realized by the team.

For this article, we extracted 55 processes for postmortem analysis. These processes document what occurred during the development of the games. Thus, the extracted processes, as well as the extraction system, may serve as a tool to analyze past projects. For example, a developer, who had developed 10 games and wants to make a retrospective of all projects, can use our system to extract 10 processes and generate their graphical representations and make comparisons, group similar projects, and even define metrics and evaluate the processes, projects, and teams.

Something that may help to solve some issues regarding the postmortem analysis and help the developers community, would be a creation of an **experiences repository** where finalized projects reports would be stored using a more rigid structure, describing errors and hits during the project execution.

We did not find a source related to a definition of video game project context. We created a way to define video game project contexts, based on literature, books, and postmortems. We defined 61 video game variables, divided into 6 categories. This set of variables is not immutable but is a basis for future extensions. Moreover, we encourage aggregation or refactoring variables because such operations do not interfere with how the recommending system works.

The recommendation system is the main contribution of this article. The aim of the generated processes is to help the development team in planning a game project, even before the preproduction phase. The resulting processes help developers to make decisions regarding the development phases or serving as a high level process to be followed.

For example, before the development starts, with the game context defined, the context form is filled by game developers. The recommendation system creates a new process using similar projects as a basis. The development team or responsible developer may analyze all the problems that occurred in other, similar games. With this data, it is possible to draw plans that prevent such problems and apply activities that went right.

The system evaluation, although with relatively low measures, results in good acceptance by the game developers. Developers who wrote postmortems acknowledged the similarity of the extracted and recommended processes. The developers related that many of the activities presented by the recommendation system were already part of the processes used by their teams. It emphasizes that the premise of this article, "learning with the past", can be a way to avoid video game development problems.

7. Threats to validity

There are concerns about the generated processes, given that the postmortem main purpose is not describe processes' elements. It took us to carry out the validation of both, the extracted and recommended processes.

As pointed in a feedback, the recommended process cannot define specifics tasks to be followed by developers. Again, it is because the information contained in postmortems does not have such specificity. Nonetheless, we argue that it is the best source of information about development experiences.

Most of the work of creating a process visualization had to have be made manually, which cost much time. Although the process visualization is built mainly by scripts, its flow is entirely made by hand. It happened because we did not stored the meta-data about elements' flow, which can be done in future extensions.

We performed tests using a dataset with 55 samples against 61 contextual properties. This might cause noise in the dataset, that is, *Overfitting* issues [14].

8. Related work

Postmortems was used by researchers as a primary source of information about game development. Petrillo et al. [3,4] extracted, with postmortem analysis, the most common problems faced by developers in game industry. They argued that these problems occur in the traditional software industry as well. Still, the authors stated that game developers suffer mainly from managerial problems, not technical ones.

In a complementary research, Petrillo et al. list the most common practices adopted in game projects. The results showed that, even if informally, game developers are adopting agile practices and, for this reason, implanting agile methods may occur naturally [9].

Another research, done by Washburn et al. [38], regarding development practices and postmortems, analyzed 155 articles extracting the characteristics and pitfalls from the game development. The result was a set of good practices for game developers.

Murphy-Hill et al. [5] realized 14 qualitative interviews and 364 quantitative, with professionals from game development area. The authors highlighted the dubious agile methods adoption, that is, the use of the word “agile” to justify the lack of process. Moreover, Pascarella et al. [39] extended this work assessing how developing games is different from developing non-game systems by performing analysis in both Open Sourced repositories types. They found that developing video games differs from developing traditional software, in many aspects, like: the way resources evolve during the project; the expertise required by the developers; the way the software is tested; how bugs are fixed; how release planning is followed and regarding requirements handling.

O’Hagan et al. did a literature review of 404 articles, from industry and academy, showing a total of 356 software processes, grouped in 23 models, of which 47% were agile and 53% hybrid [40]. Still, O’Hagan et al. made a case study in game industry exploring the role of software process in game development. The results showed that there is no good practices model for game development and suggested to create such model, based on ISO/IEC 29110 [41].

Kanode et al. also explored the role of software engineering in game development. For the authors, game development has unique characteristics and problems, and software engineering practices may help to overcome these difficulties. For example, implanting agile methods in prep-production phase and managing the large amount of game assets [8].

Regarding Recommendation Systems, there are several attempts to provide better tools to developers. For example, recommendation of developments artifacts [42–45], contextual information [46], libraries [47], methods [48], commands [49] and even pull-requests commenter [50].

These related works showed the game industry problems, suggested the use of good practices and provide recommendation of different development artifacts. However, there is no mention, as far as we know, about a implementation and validation of a system to aid developers, recommending a set of activities and practices in the form of a process, based on similar projects context.

9. Conclusions

Building a video game demands a large number of activities and choosing these activities is arduous and demands experience. This article presented an approach and a recommendation system, based on video game development postmortems, to suggest software processes using contexts and similarity degrees. The recommendation system helps video game developers to conduct new projects by suggesting a set of activities in a form of a process. It learns from previous game development experiences within similar contexts. Moreover, provides to video game developers an approach to reflect about the processes used in game industry.

This article thus presents three main contributions. It first describes the creation of a database of game development processes from the analysis of 55 postmortems. The second identifies video game project characteristics, like team attributes and technical details, to model projects contexts that work as input for the recommendation system. The third and final one describes and validates a recommendation system capable of generating processes based on previous projects with similar contexts.

The quantitative results of the validation of the recommendation system showed an average *precision* of 31.58%; good *accuracy* of 67.59%; and excellent *specificity* of 92.48%. However, other metrics had lower outcomes, like *recall* at only 8.11%. These results showed that a better technique must be provided, in order to filter the recommended processes’ elements: provide more elements with more relevance. The former is achievable by adding a larger numbers of postmortems in the database while the latter requires using better classification algorithms, improving the video game context categories, and enriching the elements of the processes.

The qualitative validation provided by video game developers showed that our approach does extract similar process from postmortems as well our recommendation system suggests useful new processes.

Finally, through a case study with a team of developers, we showed that video game development teams may, transparently, improve their software processes during the course of game development. Thus, our recommendation system can recommend video game development processes that help game developers to avoid possible problems.

As future work, we intend to improve our recommendation system by adding more postmortems and analyzing their activities. We will gather more resources and feedbacks from video game developers to improve our model. Moreover, we will improve the method used to generate processes visualization as well as the user interface for the recommendation system. Further, we will add more meta-data in process’ elements, like order and flow. With this we hope cut most of the hand working and provide a completely automated processes recommendation system.

Acknowledgments

We give our thanks to the authors of the postmortems for sharing their cases as well for their feedbacks. This work has been supported by the National Council for Scientific and Technological Development of Brazil.

References

[1] NEWZOO, The global games market reaches $99.6 billion in 2016, mobile generating 37%, 2016, (https://goo.gl/hrDlhL).

[2] A. Gershenfeld, M. Loparco, C. Barajas, Game Plan: the Insider’s Guide to Breaking in and Succeeding in the Computer and Video Game Business, St. Martin’ s Griffin Press, New York, 2003.

[3] F. Petrillo, M. Pimenta, F. Trindade, C. Dietrich, Houston, we have a problem... : a survey of actual problems in computer games development, Proceedings of the 2008 ACM symposium on Applied computing - SAC 08, ACM Press, 2008, http://dx.doi.org/10.1145/1363686.1363854.

[4] F. Petrillo, M. Pimenta, F. Trindade, C. Dietrich, What went wrong? a survey of problems in game development, Comput. Entertain. 7 (1) (2009) 1, http://dx.doi.org/10.1145/1486508.1486521.

[5] E. Murphy-Hill, T. Zimmermann, N. Nagappan, Cowboys, ankle sprains, and keepers of quality: how is video game development different from software development? Proceedings of the 36th International Conference on Software Engineering - ICSE 2014, ACM Press, 2014, http://dx.doi.org/10.1145/2568225.2568226.

[6] C. Politowski, L. Fontoura, F. Petrillo, Y.-G. Guéhéneuc, Are the old days gone?: A survey on actual software engineering processes in video game industry, Proceedings of the 5th International Workshop on Games and Software Engineering, GAS ’16, ACM, New York, NY, USA, 2016, pp. 22–28, http://dx.doi.org/10.1145/2896958.2896960.

[7] J. Weststar, M. Andrei-Gedja, Developer Satisfaction Survey 2015 Industry Trends and Future Outlook Report, Technical Report, International Game Developers Association (IGDA), 2016.

[8] C.M. Kanode, H.M. Haddad, Software engineering challenges in game development, 2009 Sixth International Conference on Information Technology: New Generations, IEEE, 2009, http://dx.doi.org/10.1109/itng.2009.74.

[9] F. Petrillo, M. Pimenta, Is agility out there? Proceedings of the 28th ACM International Conference on Design of Communication - SIGDOC 10, ACM Press, 2010, http://dx.doi.org/10.1145/1878450.1878453.

[10] C. Keith, Agile Game Development with Scrum, first ed., Addison-Wesley Professional, 2010.

[11] C. Politowski, D. de Vargas, L.M. Fontoura, A.A. Foletto, Software engineering

processes in game development: a survey about brazilian developers' experiences, XV Simpósio Brasileiro de Jogos e Entretenimento Digital, SBGAMES '16, (2016).
[12] W. Hamann, Goodbye postmortems, hello critical stage analysis, 2003, (https://goo.gl/bbLVWA).
[13] M. Robillard, R. Walker, T. Zimmermann, Recommendation systems for software engineering, IEEE Softw. 27 (4) (2010) 80–86, http://dx.doi.org/10.1109/MS.2009.161.
[14] M.P. Robillard, W. Maalej, R.J. Walker, T. Zimmermann, Recommendation systems in software engineering, Springer Sci. Bus. (2014).
[15] W.S. Humphrey, The software engineering process: Definition and scope, Proceedings of the 4th International Software Process Workshop on Representing and Enacting the Software Process, ISPW '88, ACM, New York, NY, USA, 1988, pp. 82–83, http://dx.doi.org/10.1145/75110.75122.
[16] M. Fowler, The new methodology, Wuhan University J. Nat. Sci. 6 (1) (2001) 12–24, http://dx.doi.org/10.1007/BF03160222.
[17] W.W. Royce, Managing the development of large software systems: Concepts and techniques, Proceedings of the 9th International Conference on Software Engineering, ICSE '87, IEEE Computer Society Press, Los Alamitos, CA, USA, 1987, pp. 328–338. URL http://dl.acm.org/citation.cfm?id=41765.41801 .
[18] C. Larman, V.R. Basili, Iterative and incremental developments. a brief history, Computer 36 (6) (2003) 47–56, http://dx.doi.org/10.1109/MC.2003.1204375.
[19] K. Schwaber, M. Beedle, Agile Software Development with Scrum, Series in agile software development, Pearson Education International, 2002. URL https://books.google.com.br/books?id=lO3XLgAACAAJ .
[20] K. Beck, Extreme Programming Explained: Embrace Change, An Alan R. Apt Book Series, Addison-Wesley, 2000. URL https://books.google.com.br/books?id=G8EL4H4vf7UC .
[21] C. Larman, V. Basili, Iterative and incremental developments. a brief history, Computer 36 (6) (2003) 47–56, http://dx.doi.org/10.1109/mc.2003.1204375.
[22] VersionOne, The 10th State of Agile Report, Technical Report, (2015).
[23] J. Musil, A. Schweda, D. Winkler, S. Biffl, A Survey on the State of the Practice in Video Game Software Development, Technical Report, Technical report, QSE-IFS-10/04, TU Wien, 2010.
[24] A. Fuggetta, E.D. Nitto, Software process, Proceedings of the on Future of Software Engineering - FOSE 2014, ACM Press, 2014, http://dx.doi.org/10.1145/2593882.2593883.
[25] B. Boehm, Balancing agility and discipline: a guide for the perplexed, Software Engineering Research and Applications, Springer Berlin Heidelberg, 2004, p. 1, http://dx.doi.org/10.1007/978-3-540-24675-6_1.
[26] P. Kruchten, Contextualizing agile software development, J. Softw. 25 (4) (2011) 351–361, http://dx.doi.org/10.1002/smr.572.
[27] D. Callele, E. Neufeld, K. Schneider, Requirements engineering and the creative process in the video game industry, 13th IEEE International Conference on Requirements Engineering (RE 05), IEEE, 2005, http://dx.doi.org/10.1109/re.2005.58.
[28] M. Fridley, Postmortem: Kingdoms of amalur: Reckoning, 2013, https://goo.gl/Wkjlh4.
[29] M. Myllyaho, O. Salo, J. Kääriäinen, J. Hyysalo, J. Koskela, A review of small and large post-mortem analysis methods, IEEE France, 17th International Conference Software & Systems Engineering and their Applications, Paris, 2004.
[30] I. Jolliffe, Principal Component Analysis, Wiley Online Library, 2002.
[31] B. Bates, Game Design, Thomson Course Technology, 2004. URL https://books.google.com.br/books?id=f7XFJnGrb3UC .
[32] M. Moore, J. Novak, Game Development Essentials: Game Industry Career Guide, Game development essentials, Delmar/Cengage Learning, 2010. URL https://books.google.com.br/books?id=D-spAQAAMAAJ .
[33] I. Avazpour, T. Pitakrat, L. Grunske, J. Grundy, Dimensions and metrics for evaluating recommendation systems, Recommendation Systems in Software Engineering, Springer, 2014, pp. 245–273.
[34] M.P. Robillard, Topology analysis of software dependencies, ACM Trans. Softw. Eng. Methodol.(TOSEM) 17 (4) (2008) 18.
[35] M. Ge, C. Delgado-Battenfeld, D. Jannach, Beyond accuracy: evaluating recommender systems by coverage and serendipity, Proceedings of the Fourth ACM Conference on Recommender Systems, ACM, 2010, pp. 257–260.
[36] G. Shani, A. Gunawardana, Evaluating recommendation systems, Recommender Systems Handbook, Springer, 2011, pp. 257–297.
[37] F. Salfner, M. Lenk, M. Malek, A survey of online failure prediction methods, ACM Comput. Surveys (CSUR) 42 (3) (2010) 10.
[38] M. Washburn, P. Sathiyanarayanan, M. Nagappan, T. Zimmermann, C. Bird, What went right and what went wrong, Proceedings of the 38th International Conference on Software Engineering Companion - ICSE 16, ACM Press, 2016, http://dx.doi.org/10.1145/2889160.2889253.
[39] L. Pascarella, F. Palomba, M.D. Penta, A. Bacchelli, How Is Video Game Development Different from Software Development in Open Source? (2018). May
[40] A.O. O'Hagan, G. Coleman, R.V. O'Connor, Software development processes for games: a systematic literature review, Systems, Software and Services Process Improvement, Springer, 2014, pp. 182–193.
[41] A.O. O'Hagan, R.V. O'Connor, Towards an understanding of game software development processes: a case study, Communications in Computer and Information Science, Springer International Publishing, 2015, pp. 3–16, http://dx.doi.org/10.1007/978-3-319-24647-5_1.
[42] D. Čubranić, G.C. Murphy, Hipikat: recommending pertinent software development artifacts, Proceedings of the 25th International Conference on Software Engineering, ICSE '03, IEEE Computer Society, Washington, DC, USA, 2003, pp. 408–418. URL http://dl.acm.org/citation.cfm?id=776816.776866 .
[43] R. Holmes, A. Begel, Deep intellisense: a tool for rehydrating evaporated information, Proceedings of the 2008 International Working Conference on Mining Software Repositories, MSR '08, ACM, New York, NY, USA, 2008, pp. 23–26, http://dx.doi.org/10.1145/1370750.1370755.
[44] R. Holmes, R.J. Walker, G.C. Murphy, Strathcona example recommendation tool, SIGSOFT Softw. Eng. Notes 30 (5) (2005) 237–240, http://dx.doi.org/10.1145/1095430.1081744.
[45] T. Zimmermann, P. Weisgerber, S. Diehl, A. Zeller, Mining version histories to guide software changes, Proceedings of the 26th International Conference on Software Engineering, ICSE '04, IEEE Computer Society, Washington, DC, USA, 2004, pp. 563–572. URL http://dl.acm.org/citation.cfm?id=998675.999460 .
[46] L. Ponzanelli, S. Scalabrino, G. Bavota, A. Mocci, R. Oliveto, M. Di Penta, M. Lanza, Supporting software developers with a holistic recommender system, Proceedings of the 39th International Conference on Software Engineering, ICSE '17, IEEE Press, Piscataway, NJ, USA, 2017, pp. 94–105, http://dx.doi.org/10.1109/ICSE.2017.17.
[47] A. Ouni, R.G. Kula, M. Kessentini, T. Ishio, D.M. German, K. Inoue, Search-based software library recommendation using multi-objective optimization, Inf. Softw. Technol. 83 (2017) 55–75.
[48] I. Velásquez, A. Caro, A. Rodríguez, Kontun: a framework for recommendation of authentication schemes and methods, Inf. Softw. Technol. (2017).
[49] M. Gasparic, A. Janes, F. Ricci, G.C. Murphy, T. Gurbanov, A graphical user interface for presenting integrated development environment command recommendations: design, evaluation, and implementation, Inf. Softw. Technol. 92 (2017) 236–255.
[50] J. Jiang, Y. Yang, J. He, X. Blanc, L. Zhang, Who should comment on this pull request? Analyzing attributes for more accurate commenter recommendation in pull-based development, Inf. Softw. Technol. 84 (2017) 48–62.