How Is Video Game Development Different from Software Development in Open Source?

Luca Pascarella[1], Fabio Palomba[2], Massimiliano Di Penta[3], Alberto Bacchelli[2]
[1]Delft University of Technology, The Netherlands — [2]University of Zurich, Switzerland — [3]University of Sannio, Italy
l.pascarella@tudelft.nl,palomba@ifi.uzh.ch,dipenta@unisannio.it,bacchelli@ifi.uzh.ch

ABSTRACT

Recent research has provided evidence that, in the industrial context, developing video games diverges from developing software systems in other domains, such as office suites and system utilities.

In this paper, we consider video game development in the open source system (OSS) context. Specifically, we investigate how developers contribute to video games vs. non-games by working on different kinds of artifacts, how they handle malfunctions, and how they perceive the development process of their projects. To this purpose, we conducted a mixed, qualitative and quantitative study on a broad suite of 60 OSS projects. Our results confirm the existence of significant differences between game and non-game development, in terms of how project resources are organized and in the diversity of developers' specializations. Moreover, game developers responding to our survey perceive more difficulties than other developers when reusing code as well as performing automated testing, and they lack a clear overview of their system's requirements.

KEYWORDS

Video Games; Mining Software Repositories; Empirical Studies

ACM Reference Format:
Luca Pascarella[1], Fabio Palomba[2], Massimiliano Di Penta[3], Alberto Bacchelli[2]. 2018. How Is Video Game Development Different from Software Development in Open Source?. In *Proceedings of MSR '18: 15th International Conference on Mining Software Repositories , Gothenburg, Sweden, May 28–29, 2018 (MSR '18),* 11 pages.
https://doi.org/10.1145/3196398.3196423

1 INTRODUCTION

In the last decades, several human activities (*e.g.*, financial transactions, methods of defense, healthcare, and scientific research) have started to rely more and more on software systems to run efficiently [20, 46]. Entertainment activities have also followed this trend and video games are one of its most prominent outcomes [21]. Nowadays, the video game industry has reached an estimated yearly revenue of more than $90 billion dollars [45].

Despite being a domain of software systems and being so successful, video games (from hereon, *games*) have attracted the interest of software engineering researchers only in the last decade. For example, among the first is the work by Tschang [48] and Tschang & Szczypula [49] who hypothesized that game development requires developers with uncommon knowledge. In the same vein, Kultima & Alha conducted interviews reporting how game developers perceive differences in the development of their projects [28] and Ampatzoglou & Stamelos provided an overview of the concerns in software engineering for games, indicating that the game domain had received little attention from software engineering research [4].

Murphy-Hill *et al.* are the first who conducted a "broad-based empirical study to explicitly contrast traditional software engineering and ... game development" [33] in an industrial context. Murphy-Hill *et al.* conducted interviews with software engineers expert in game development from different companies, followed by a survey sent to selected developers at Microsoft [2]. They found that game developers perceive their development process to follow Agile methodologies more often, to require a more diverse team, and to require better communication skills with non-engineers, compared to non-game developers. Murphy-Hill *et al.* discussed compelling implications of these findings for software engineering research, practice, and education, thus also highlighting the importance of conducting this kind of studies.

In this paper, we continue on the line of research on game development, by shifting our focus to open source software (OSS) systems. We investigate how developing OSS games is different from developing non-game OSS systems, such as system utilities and office suites. The know-how about practical solutions adopted by video game developers is crucial to lead future research aimed at improving the quality of software and at increasing developers' productivity with practical support. From a high-level perspective, our work is in the direction of increasing our empirical understanding of how and to what extent software development is influenced by its target (*e.g.*, games, office suites, and system utilities) and how research should be tailored accordingly.

Specifically, we conducted an exploratory investigation aimed at reproducing the findings of Murphy-Hill *et al.* [33] and complementing it through the mining of OSS repositories and the analysis of the perception of OSS developers. We (1) mined 30 OSS games and 30 traditional OSS systems, (2) analyzed how developers commit versioned resources, (3) measured the authorship and ownership for specific categories, (4) analyzed the diversity in malfunctions, and (5) challenged our findings with a survey involving 81 respondents among the most productive developers of the chosen projects.

Our results show that developers of OSS games tend to diverge from strict software engineering rules. They autonomously split into teams specialized in specific tasks, yet they manage to collaborate with each other to achieve common goals. Moreover, the investigation suggests that preventing malfunctions is an even

MSR '18, May 28–29, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5716-6/18/05…$15.00
https://doi.org/10.1145/3196398.3196418

harder task in this domain, due to the difficulty in automating testing. Finally, our results, on the one hand, confirm some findings achieved about game development in industry [33], such as less clear requirements, while on the other hand highlights some differences that were not found to be significant previously, such as the likelihood that developed code is included in future releases and the difficulty in writing automated (unit) tests. Overall, our evidence indicates that the domain specific issues that game development poses encompass both the industrial vs. OSS context.

2 RESEARCH GOALS AND SUBJECT

This section defines the goal of our empirical study in terms of research questions and the subject systems we consider.

2.1 Research Questions

The final *goal* of our study is to explore the (potential) differences between the development practices for games vs. non-game systems in OSS. A great source of inspiration for us comes from the work by Murphy-Hill *et al.* [33], who conducted a similarly targeted study in the industrial context. We complement their work and hope to be able to generalize it, by (1) considering the OSS context, thus reaching a different set of developers and projects, and by (2) conducting quantitative analysis on software repositories.

We start our investigation by focusing on the software repositories. In particular, we structure our first research question around one of the six findings that was confirmed in both interviews and survey of the study by Murphy-Hill *et al.*: "Game development requires a more diverse team" [33]. Therefore, we investigate the diversity of resources that exist in game vs. non-game source code repositories, how prominent each type of resource is in the two domains, and whether game developers show a higher degree of specialization on certain resources, thus indicating the existence of a more diverse team. This leads to our first research question:

RQ_1: How do developers of OSS games vs. OSS non-game systems contribute to their projects?

We continue our quantitative investigation exploiting software repositories by focusing on *testing* and *malfunctions*. In fact, the sample of developers surveyed and interviewed by Murphy-Hill *et al.* [33] perceived testing as another significant difference in game vs. non-game software development; testing was reported to be more difficult when developing games because of, *e.g.*, the high coupling with the user interface, the size of the state space, and the inherent non-determinism. It is reasonable to think that this difference may lead to how malfunctions manifest themselves in games. We investigate this aspect in the repositories of the selected systems, by analyzing which faults occur in games vs. non-games. This leads to our second research question:

RQ_2: How do developers of OSS games vs. OSS non-game systems prevent and handle malfunctions in their projects?

Finally, we conclude our investigation by turning to OSS developers to gather their opinions on the development process in the projects to which they are contributing. We do this by means of an online survey. Our aim is to compare and contrast the perceptions of OSS developers to those of the industrial developers sampled by Murphy-Hill *et al.* [33], in order to understand what are the themes that are common across these two different settings and what, instead, differs. This leads to our last research question:

RQ_3: How do developers of OSS games vs. OSS non-game systems perceive the development process of their projects?

Table 1: Overview of the analyzed software systems

	Projects	Genre	Language(s)	LOC	Contributors
Game Software System	0 A.D.	3D game	JavaScript, C, C++	5.57M	16
	Arx Libertatis	3D game	C, C++	118k	3
	AssaultCube	3D game	C, C++	126k	5
	Battle for Wesnoth	2D game	C++	701k	67
	Blender 3D	3D game	C, C++, Python	1.83M	98
	Cataclysm-DDA	2D game	C++	251k	168
	Chaotic Rage	3D game	C++	84k	3
	Cyberdojo	2D game	Objective-C and C	263k	8
	Dolphin-emu	3D game	C, C++	1.23M	109
	Dungeon Crawl Stone Soup	2D game	C, C++	372k	57
	FlightGear	3D game	C++	838k	23
	FreeSpace	3D game	C, C++	1.13M	8
	Frogatto	2D game	Python, Ruby, C++	7.4k	9
	Hedgewars	2D game	Pascal, Lua, C, C++	165k	17
	MAME	2D game	C++	9.14M	164
	ManaPlus	2D game	C++	270k	5
	MegaGlest	3D game	C++	304k	8
	Minetest	3D game	C, C++	203k	163
	Multitheftauto	3D game	C, C++	2.73M	24
	Oolite	3D game	C, Objective-C	263k	8
	OpenArena	3D game	C	500k	6
	OpenClonk	2D game	C++	360k	16
	OpenDungeons	2D game	C++	73.7k	6
	OpenMW	3D game	C++	226k	48
	OpenSimulator	3D game	C#	1.28M	8
	Orxonox	3D game	Lua, C, C++	9.28M	12
	Pioneer	3D game	C++	479k	30
	SuperTuxKart	3D game	C, C++	732k	29
	Thousand Parsec	2D game	C++	3.7k	2
	Warzone 2100	3D game	C, C++	667k	22
Non-game Software System	Calligra Suite	Office suite	C, C++	1.15M	18
	Chromium	Browser	C, C++	18.1M	2,695
	Cppcheck	Utility	C++	241k	61
	Doxygen	Doc system	C++	275k	38
	Firebird	Database	C, C++	1.27M	33
	GIMP	Image editor	C	833k	71
	Gparted	System utility	C, C++	44.5k	36
	Iptables	System utility	C	52.8k	27
	K3b	Utility	C++	106k	17
	Kate	Text editor	C++	403k	56
	KiCad	CAD	C, C++	872k	74
	Ktorrent	Application	C++	102k	17
	Libre Office	Office suite	Java, C++	9.15M	234
	Mbed	SDK	C	3.82M	187
	MongoDB	Database	C++	1.28M	133
	MySQL	Database	JavaScript, C, C++	3.35M	132
	Node.js	Framework	C, C++	3.87M	868
	Notepad++	Text editor	C++	338k	64
	Open-Xchang	Email	Java, C	3.33M	53
	OpenSSL	Library	C	488k	124
	OpenVPN	System utility	C	291k	28
	OpenWrt	Embedded Kernel	C	842k	205
	PowerDNS	DNS deamon	C++	216k	58
	Programmer's Notepad	Text editor	C, C++	1.35M	2
	Scilab	Scientific	C, C++	2.29M	18
	Sumatra PDF	PDF viewer	C, C++	569k	11
	Synergy	Application	C, C++	113k	5
	TortoiseGit	Version system	C, C++	368k	7
	Umbrello	UML Modeller	C++	264k	8
	VLC	Media player	C, C++	628k	82

2.2 Selection Of The Subject Systems

To conduct our study, we consider the most popular programming languages used for desktop video game applications (*i.e.*, C, C++, and Objective-C [14]) and on projects whose source code is publicly available *i.e.*, open-source software (OSS) projects. To select a representative sample of game and non-game systems we rely on *OpenHub* [1], which is an online platform that indexes open-source projects, providing basic information (*e.g.*, application domain) as well as data on developers' activities (*e.g.*, number of commits) and statistics on popularity (*e.g.*, number of stargazers). We use OpenHub to select heterogeneous projects having different characteristics in terms of (i) development environment, (ii) number of contributors, and (iii) project size, thus mitigating some threats to external validity. We only consider active projects to be able to get responses to our online survey.

Thus, using OpenHub, we rank projects by popularity and select 30 desktop video game systems and 30 heterogeneous non-game applications. Table 1 reports the chosen projects describing genre, programming language(s), size in LOC, and number of contributors.

3 RQ_1 – DEVELOPMENT ACTIVITIES

Our first research question aims at studying how game and non-game developers contribute to their projects.

3.1 RQ_1 - Research Method

To answer RQ_1, we start by classifying the resources (*i.e.*, files) contained in a repository into categories that reasonably require different expertise and specialization (*e.g.*, source code files and images). Subsequently, we investigate differences between game vs. non-games in terms of (1) the prevalence of the categories (we expect non-games to have less multimedia files, audio, video, and images), (2) the specialization of contributors (we expect games to have more diverse and specialized contributors, as emerging from the study of Murphy-Hill *et al.* [33]) considering authorship as well as ownership, and (3) the evolution of categories.

Classifying the resources. Given the number of investigated projects, a manual categorization of all the files is prohibitively expensive. For this reason, we apply a two-step approach: (i) first, we execute an iterative *content analysis* approach [30] on a subset of six projects to let categories of files emerge and identify features that can be used to automatically classify files; (ii) second, we devise a tool to automatically categorize the files of the studied projects based on the identified features.

In the first step, the first author of this paper (a software engineering researcher with ten years of programming experience) analyzed the files contained in three game and three non-game systems. The task was to analyze the path of each file and identify the file category, also considering the directory organization as an additional clue, how emerged from a study of Jones *et al.* where computer users use "divide and conquer" problem decomposition [25]. Generally, the researcher was able to identify keywords in the path that can be used to discriminate the type of file are identified (*e.g.*, the presence of `src` likely indicates that the file is a source file); in the case of ambiguities (*e.g.*, the term `image` may be related to pictures, but also to high-level system models, such as UML diagrams), the researcher also considered the extension of the file. With this approach, the researcher was able to classify the vast majority ($\simeq$ 95%) of files in the six analyzed systems. Table 2 shows the resulting list of categories. The columns 'Keywords in File Path' and 'File extension' report the specific keywords used to automatically categorize files. The *'Discarded'* category collects the files for which it is not possible to assign any of the other categories identified.

Table 2: Overview of the categories of resources

	Category	Description	Keywords in file path	File extension
Development	**Code**	Source code files, such as sources, headers, assembly files	source, src, tool, include, etc.	cpp, cc, h,hpp, in
	Utility	Scripts, makefiles, build configurations, etc.	util, test, src, source, include, build, comp, etc.	py, pl, js,lua, mk,cmake, m4
	Library	Archives and libraries	lib, data, os, arch, etc.	a, so, lib, dll, so, zip, rar, 7z, gz, bz2
	Language	Translation-related files	language, lng, i18n, translation.	po, pot, i18n, txt, xml
	Docs	Documentation	doc, man, license, guide, package	tex, txt, html, htm, xml, css, pdf, jpg, png, ico, gif
Multimedia	**Audio**	Audio files		wav, ogg, mp3, dsp
	Image	Image files	image, icon, model, scenery, texture, graphic, planet, font, etc.	png, rgb, ttf, cfg, map, jpg, gif, ico, svg, dds, xcf, 3ds, txf, eff
	Data	Domain modeling files or configurations files.	image, icon, model, scenery, texture, graphic, planet, etc.	properties, xml, canvas, effects, in, commands, electrical, extensions, desktop
Other	**Misc.**	General purpose configurations	misc, other, tool, install, etc.	xml, conf, list, cfg, txt, ocm, lo
	No ext.	Files with no extensions		
	Discarded	Every discarded resource not in the above categories		

The process was repeated to ensure the completeness of the categories. The iterations terminated when the category *'Discarded'* contained less than 5% of all the considered files. The output consisted of ten file categories, which could be grouped into three higher-level categories: 'development', 'multimedia', and 'other' (first column in Table 2).

Once we identified the categories and the discriminating features (keywords in paths and extensions), we created a Python script to parse each file of the remaining repositories and automatically classify them into the categories.

Determining the specialization of authors and owners. To measure the extent to which *authors* are specialized in the categories we identified, we mine the commits performed by each developer over the history of a project to identify the set of files each developer worked on the most.

In addition, we consider the role of *ownership*. In particular, we take into account the findings by Bird *et al.* [7], who reported that even if a file may be committed by many *authors*, it is most of the times touched by a single *author*. In other words, the ownership

of an author on a file describes the degree of responsibility of a certain contributor on that file. Thus, we want to measure what extent it is possible to identify *owners* specialized in contributing to single file categories. To this purpose, we divided the analysis into two steps. In the first step we re-implement the approach by Bird *et al.* [7]: For every file, we (1) count the number of changes, (2) associate an author to each change, (3) rank the authors by contribution frequency, and (4) discard files without a unique author contribution. The cutting threshold is experimentally defined to consider only files in which an author contributes more than 75%. In the second step, we count the number of files each developer mainly contributed to, thus marking each developer's ownership.

Evolution of file categories over time. To understand how the development activities performed on each of the identified categories evolve over time we adopt two complementary strategies to split the time. The first strategy splits each project's history into fixed intervals of three months, while the second strategy splits the history by release. The latter strategy leads to intervals of different duration, yet better accounts for the actual volume of performed development, since OSS projects tend to be more erratic in the amount of work developers produce in a given time period [19].

With both strategies, for each file category f_c identified, we compute the frequency of commits that modified a file belonging to f_c for every snapshot taken in the time window considered.

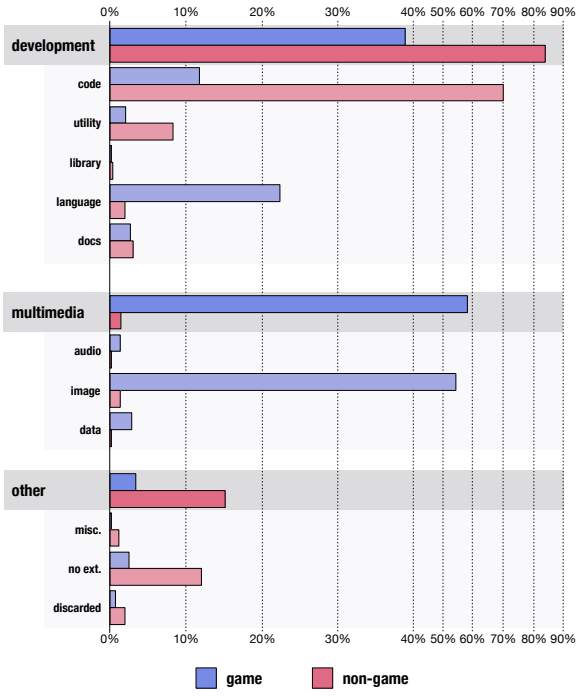

Figure 1: Distribution of files in categories, by domain.

3.2 RQ_1 - Results

We present the results according to different angles we investigated: (i) how systems' files are distributed across the different kinds of categories in games vs. non-games, (ii) how contributors are specialized in changing and owning files in these categories, and (iii) how categories evolve over time.

Distribution of files across categories. Figure 1 details how files are distributed in (grouped) categories (grouped categories show the sum of the values for the inner categories). On average non-game projects have 80% of files related to the *Development* category, while a lower percentage (15%) of files related to the *Other* category and an even lower percentage (less than 5%) related to *Multimedia*. Conversely, game projects have only 30% of files on average belonging to the *Development* category. Furthermore, there is a remarkable amount of files belonging to *Multimedia* (up to 70% for OpenClonk project). This result gives an initial indication that the resources game developers deal with are observably different from those worked on by non-game developers, and regard the management of audio, images, and game scenarios.

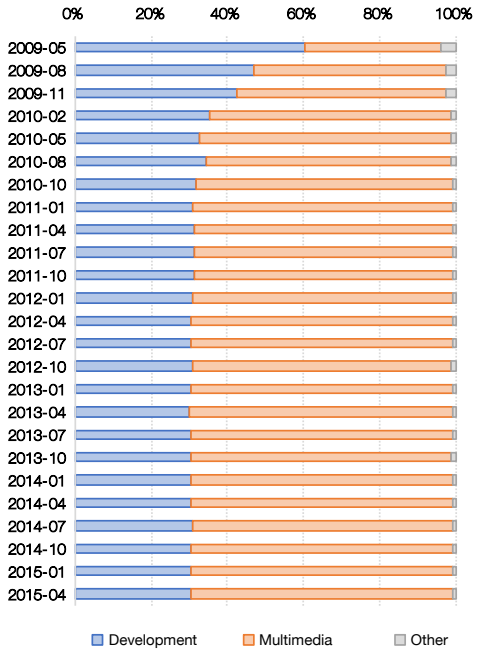

Figure 2: Evolution of changes to files in grouped categories, for *OpenClonk*, a game project, in 3-month time windows.

Specialization of authors and owners. We observe the specialization in terms of authorship by counting the number of changes in each file category; since this analysis implies creating a change graph for every developer of every project, we limit the analysis to the most active developers. The results do not highlight particular trends. For example, one of the most active developers of the game Chaotic Rage had contributed with more than 2,000 commits (at the time of analysis), but the contributions are spread over each category without highlighting a specialization. Similar discussion for the two top contributors of the VLC Media Player project, who performed more than 12,000 and 6,000 commits, respectively, touching all categories of files.

Conversely, we obtain different results when considering specializations in terms of ownership. In fact, in games, we could identify

specific sub-teams working on different files belonging to non-development categories. For example, in Battle for Wesnoth four developers mainly focused their activities on *Multimedia*. Results are consistent when considering the other games. Instead, when analyzing the non-games, we clearly identify only sub-team specialized on categories related to code files. This result corroborates the statement that game development is based on a more diverse team than non-game software development since different game developers are in charge of the evolution of different aspects of the game.

Evolution of the categories. In addition to analyzing the files in the latest version of the project and the history of changes to compute authorship and ownership, we also explore whether observable trends of changes occur in different categories of files. In fact, one of the interviewees in the study by Murphy-Hill *et al.* stated: "[In general software development,] you might be building Word or Excel or something like that [and use] some zip... tool or something. But you're building... video games, you are sometimes building the resource compiling tools or tools that are intended to extract 3D assets from other software like Maya or Max and then convert it into a native format that then your engine can load and render and process. So the tool pipeline is incredibly important to video game development and it's probably I would say almost larger than the game itself." [33], but this was not confirmed by the differences developers' perceptions in their survey. We turn to the source code repositories to look at this in OSS systems; if the aforementioned statement was true, we would expect to see more work on the source code in the first period of development, and then more activity on the game scenarios/data (*e.g.*, committing more changes belonging to *Audio*, *Image*, and *Data*).

Figure 2 is an exemplification of the results we obtained through this analysis. The Figure shows the trend in changes for OpenClonk, a game project, considering the observation interval of 3 months. In the first interval (about 18 months) developers focus more on the source code and project development artifacts (*e.g.*, build automation) and only after they spend effort on improving the game scenarios (*e.g.*, game worlds and audio). These results are consistent across all games and also considering a release-by-release observation interval. Conversely, this trend does not hold for non-games, even considering the broad heterogeneity of the selected projects: In this case, the vast majority of changes fall in the *Development* category in every time period, while other types of changes are a minority. This result seems to corroborate the statement by the interviewee in the study of Murphy-Hill *et al.* [33], at least in the OSS context: The development of games vs. non-games is different and follows different trends in terms of the files modified during the system evolution.

Result 1: Developing games involves activities on more diverse resources than developing non-game systems. The evolution is different: In games a ramp-up period of changes on source code is followed by a steady work on other categories of files; in non-games the evolution consistently involves the same categories. The teams are different: Only in games we could identify specialized owners who apply changes to non-code related categories, thus highlighting the presence of more diverse expertise.

4 RQ_2 – MALFUNCTION HANDLING

The second research question aims at investigating the ways developers in the two domains of projects handle malfunctions.

Table 3: Overview of bug categories considered in our study.

Category	Bug description	Search keywords/phrases
Algorithm	Algorithmic or logical errors	algorithm, logic, rendering, calcula, procedure, problem solving, math, stack size, bench script, mistake, defect
Concurrency	Multi-threading or multi-processing related issues	race condition, synchronization error, deadlock
Failure	Crash or hang	reboot, crash, hang, restart, fault, return failure, segfault, dump, executable file, error message, segmentation, stable, exception, not run, not start
Graphic	Graphic issues such as overlap or rendering problems	graphic, resize, overlap, render, shadow, gui object, frame, ground, window, zoom, water, weapon
Memory	Incorrect memory handling	memory leak, null pointer, heap overflow, buffer overflow, dangling pointer, double free, segmentation fault, buf, memleak, memory leak, overflow, alloc
Performance	Correctly runs with delayed response	optimization, performance, slow, fast, busy
Programming	Generic programming errors	exception handling, error handling, type error, typo, compilation error, copy-paste error, pasting, refactoring, missing switch case, missing check, faulty initialization, default value, match error, compil, autotools, build, undefined pointer, syntax error, instruction, 64bit, overloaded function, translation, engine, not iniatializ
Security	Exposure to dangerous attackers	buffer overflow, security, password, auth, ssl, exploit, injection, aes, 3des, rc4, access
Unknown	Not part of the above categories	

4.1 RQ_2 - Research Method

To understand how malfunctions (*i.e.*, faults) are treated in game vs. non-game systems in OSS, we combined mining project repositories and surveying expert developers. In this subsection, we report the mining method, while the survey part is presented in Section 5.1, together with the other questions asked to developers.

Categories of malfunctions. The categories of malfunctions analyzed are reported in Table 3, together with a short description and the keywords that can be used for assigning a fault to that category. Generally, the categories reflect the ones defined by Ray *et al.* [41], who studied the effect of programming languages on software quality and proposed a catalog of malfunctions that may appear in a software project. We add a new category, *i.e.*, *'graphic'*, to better distinguish issues related to the user interface (potentially prominent in games) from other types of malfunctions (*e.g.*, performance bottlenecks). Despite the same keywords may belong to different categories (*e.g.*, memory leak may be part of *Graphic* and *Memory*) we prevent this issue by defining a specific set of keywords for each category.

Identification and classification of malfunction fixes. Before categorizing malfunctions according to the taxonomy in Table 3, we (i) identify commits reporting faults and (ii) link them,

where possible, to the issue tracker to extract the note explaining the type of fault.

To identify commits reporting faults, we follow an approach similar to that proposed by Mockus and Votta [32] and Fischer *et al.* [16], which was also followed in the studies by Ray *et al.* [40, 41]. This approach looks in the commit message for the existence of specific keywords (*e.g.*, 'error', 'bug', 'fix', 'issue', 'mistake', 'incorrect', 'fault', 'defect', and 'flaw') and accordingly mark the commit as fault repairing. Since the commit message may contain only a short description of the fault fixing activity performed by a developer (*e.g.*, "fix bug #276") rather than presenting the information needed to properly categorize it, we use the issue tracker to retrieve more detailed data on faults. To this aim, we use the explicit references contained in a commit message to identify the issue related to it. Afterwards, we extract the message note of the issue and apply the categorization using the keywords reported in Table 3. However, for projects lacking issue trackers we classify the commit messages only. The set of keywords belonging to each category was iteratively improved: Starting from an initial set of keywords, the first author of this paper manually checked whether the malfunctions that are not possible to categorize contain additional keywords that can be used to assign it to one of the categories considered. This refinement reduced the risk of misclassification.

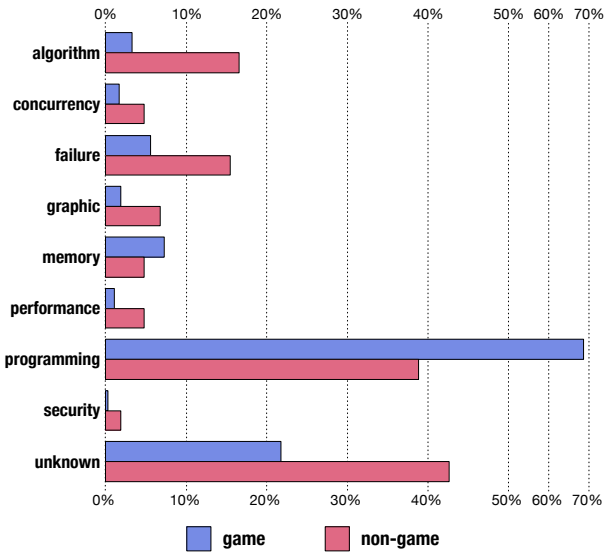

Figure 3: Distribution of faults, by domain.

4.2 RQ$_2$ - Results

Figure 3 reports the comparative distribution of malfunctions. The first observation is that the category *Graphic* related to game projects absorbs about three times more malfunctions with respect to traditional software. However, this may be only a confirmation that systems with a massive use of the Graphical User Interface (GUI) tends to have more problems in the components implementing it than any other system. Similarly, the result might be also reflect the higher number of changes performed by developers on files involving the *Multimedia* category, *i.e.*, the higher the number of changes, the higher the likelihood to introduce faults [24]. Games have also fewer problems in components related to *Programming* than non-games. Also, in this case, this might be due to the lower activity performed on such components. Perhaps more interesting, the number of faults are spread in several categories when considering games, while traditional systems reach up to 62% of the problems related to the *Programming* category. The result for *Security* is quite unexpected: We observe a higher number of security issues in games rather than non-games. This possibly reveals the lack of awareness of game developers with respect to vulnerabilities or, even worse, the lack of testing activities in games: the latter conjecture is supported by (i) the higher number of *Failures* than traditional systems and (ii) the absence of test cases in the considered projects. Another observation is that in the case of games we have a minor capability to classify malfunctions using the same set of keywords used for traditional software. Indeed, the *Unknown* category for games is about one third higher than the same category in the opposite software domain.

Result 2: Malfunctions are differently distributed in games and non-games; while faults in games are more spread across different categories, in non-games faults are mainly related to the *Programming* category. We also confirm that games have more problem of *Graphic* than non-game systems.

5 RQ$_3$ – DEVELOPERS' PERCEPTIONS

The third research question of our study aims at gathering the developers' point of view concerning the development process followed to develop the systems with which they are involved, with the aim of exposing any different perceptions in game vs. non-game developers, for aspects that could not be captured through mining of repositories.

5.1 RQ$_3$ - Research Method

To answer our third research question, we created an online survey and asked developers of the considered systems to participate. As recommended by Flanigan *et al.* [17], we limit common issues possibly affecting the response rate by keeping the survey short, respecting the anonymity of participants, and preventing our influence in the answers. This part of the study is intended to be a replication of the study by Murphy-Hill *et al.* [33], but in the OSS context. In addition, the survey aims at challenging the results of the first two research questions.

Protocol. We created an anonymous online survey (requiring approximately 10 minutes to be filled out), extensively inspired by that of Murphy-Hill *et al.* [33], to assess differences in developers' perception of games vs. non-game software development.

The survey is organized into five sections, each of them composed of three or four statements that developers are requested to rate using 5-levels Likert Scale [35] ranging from 'Strongly Disagree' to 'Strongly Agree'. The first part aims at gathering demographic information on the expertise of participants as well as their background. The second part asks the participant opinions on software design aspects such as the facility to reuse source code or skills

required to develop the underlying software system. The third part investigates the development processes asking opinions on simplicity to define requirements and common support tools adopted. The fourth part examines the developer organizations and required skills (*e.g.*, is creativity a requirement?), while the last part emphasizes the bug prevention and the testing procedure. In addition, we left a free field to collect suggestions, opinions, and experiences.

Participants. To gather responses from developers who are actually experts of the subject systems, we considered the top ten contributors of each project. We selected the name and email address of these developers by mining the 60 studied software repositories. When the email address of the contributor was not reported or the service returned a delivery error, we selected the next expert developer of the project, to try to achieve at least ten working email addresses for each project, for a total of 600 invitations. We received answers from 45 game developers (15% resp. rate) and 36 non-game ones (12% resp. rate); the response rate is above the suggested minimum response rate for survey studies [5].

5.2 RQ_3 - Results

Figure 4 summarizes the results for our survey, by reporting for each statement a pair of bars: the first one refers to answers from game developers, the second one to answers from non-game developers. Each bar chart depicts the number of answers for each possible rate (*i.e.*, the 5-points Likert scale). Figure 4 also reports results of the statistical comparison, performed using the Wilcoxon rank sum test [12], between the distributions of response values (mapped on the 1-5 ordinal scale) for games and non-games. Note that since we test the null hypothesis *"there is no significant difference between games and non games"* multiple times, we adjust the p-values using the Benjamini-Hochberg correction [6]. Such a correction procedure adjusts p-values by ranking them in ascending order and then multiplying each p-value by the total number of p-values divided by the rank. Finally, we also report the Cliff's delta effect size [23].

Generally, the results tend to confirm what found by Murphy-Hill *et al.* [33] in the industrial context: Developers perceive the development of games to be different than non-game development. Looking at Figure 4 we see that: (1) *difficulties in reusing their code (Q1).* Murphy-Hill *et al.* found that game developers have more difficulties in reusing code than non-game developers, although the difference was not statistically significant. In our case, the difference is even less evident. This could possibly depend on the OSS context. (2) Game developers have a *less clear view of the project requirements (Q5).* This result can be explained with the findings by Kultima and Alha [28], who showed that requirements in game projects are imposed by end users that are more prone to call for new requirements. (3) Games are *tested manually by external testers (Q10),* as much as non-games. This result is aligned with the findings reported by Murphy-Hill *et al.*, in which the difference between games and non-games was not significant in their survey nor in ours. (4) Game developers perceive *less pressure on the evolution of the software architecture (Q6).* This is in line with the results reported by Murphy-Hill *et al.*: The motivation is that game developers do not spend time in evolving the architecture of one-off games that will be not updated anymore. (5) Game developers have

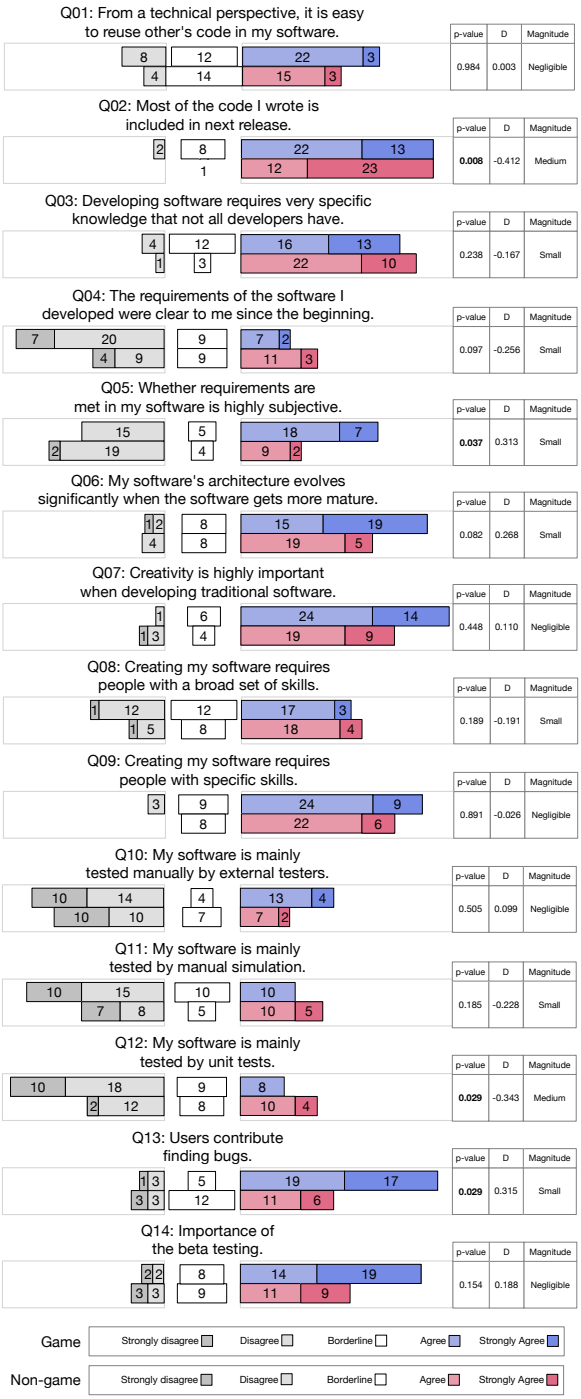

Figure 4: Likert Distributions of survey results.

difficulties in performing unit tests (Q12). This can be due to the characteristics of games as well as the massive presence of GUI-related components that are more difficult to test [42]. This is significant in our OSS study, but it was not for the industrial context investigated of Murphy-Hill *et al.* [33].

The results of our $\mathbf{RQ}_1$ about the presence of specialized sub-teams composing game systems seem aligned with the developers' perception. For instance, one of the game developer surveyed stated: "I should note that most of my experience comes from modding rather than developing games from scratch. Therefore I'm usually bound to the engine and things available. This includes reading a lot of code to understand how things work. Furthermore, it's usually small teams, therefore it helps a lot to be able to tackle problems in multiple divisions (graphics, actual code, ...)." In other words, this comment suggests that building a game from scratch may be complex, however, the organization of specialized sub-teams facilitates developers in designing and developing a game.

A similar idea comes from another game developer: "You can kind of draw a continuum between cloning games and creating completely new games. The first one requires relatively little iteration and understanding of game design, and the second one requires a lot of iteration and understanding of game design."

We conclude the analysis of the results by reporting the comment left by a game developer who reports his/her experience as an OSS game programmer: "Video game development is a specific project type and thus needs a specific experience. Without speaking of deadlines that aren't existing in Open source development. The design goals have to be refined every mid-milestone development cycle to the least. All this makes Agile development and extreme programming fit that kind of development well. And with such development management type, a dynamic and proactive team is also a must. Also, on open source projects, whenever the team is unable to commit itself to make compromise and find common acceptable goals, the project is hampered and is threatened to die or stall."

Result 3: Game developers find it difficult to reuse their code, have less pressure on evolving their systems' architecture, and are less able to properly have an overview of the requirements of a project. Furthermore, they report more difficulties in performing automated unit tests than non-game developers.

6 DISCUSSION & IMPLICATIONS

After presenting the results of our study, in this section we focus on (i) a discussion of the results also comparing our findings with those reported by Murphy-Hill *et al.* [33] and on (ii) an analysis of the direct implications that our study has for both practitioners and researchers. Table 4 presents a summary of the comparison between our findings and those of Murphy-Hill *et al.* [33].

6.1 Discussion

Our results highlighted a number of points to be further discussed, and in particular:

The development of games is different. While the differences between the two types of software were already pointed out by Murphy-Hill *et al.* through surveys and interviews, we extended existing knowledge by tracking developers' activities on files. We observed that programmers involved in the two types of software mainly perform activities on different categories of resources, with files related to *Multimedia* mainly modified by game developers and files associated with the *Development* category as the main target of non-game developers.

Table 4: Our findings and those achieved by Murphy-Hill *et al.* [33]. '✓✓' stands for strong differences measured in the study, '✓' for modest differences, '—' for irrelevant/absent differences, and no mark for a not comparable task.

	Murphy-Hill *et al.* (industry)		This study (OSS)	
	Interviews	Survey	Repositories	Survey
Software requirements	✓✓	✓✓		✓✓
Software design	✓	—	✓	✓✓
Software construction, tools, and methods	✓	—	—	—
Software testing and quality	✓	✓	✓✓	✓
Software maintenance	✓	✓	—	✓
Software configuration management	✓	✓	—	✓
Problem solving and skill variety	✓✓	✓✓		✓
Distribution of files			✓✓	
Specialization of authors			✓	
Specialization of owners			✓✓	
Categorization of malfunctions			✓✓	
Planning of releases	✓	✓	—	✓✓

The way resources evolve is different. Differently from previous analyses [33], we could observe the artifacts evolution during the history of software projects. Game developers focus more on the underlying code base during the first development phase, while they have more work later on the maintenance and evolution activities of audio, images, and multimedia-related data. Conversely, non-gamed developers equally evolve the code during the entire software lifecycle. More importantly, our survey let emerge statistically significant differences between the two types of systems with respect to the amount of code usually included in future releases of the system: Game developers report to spend more effort in code that is not directly put into production.

The game teams have more diverse expertise. We could corroborate quantitatively, observing the ways game and non-game developers apply changes on their systems, the qualitative finding reported by Murphy-Hill *et al.*: The teams are different. Specifically, in game systems, we could detect specialized developers who apply changes to non-code related categories; in non-games, instead, we could not identify developers specializing on a specific non-code category.

Game development and testing. Our survey respondents reported that that games are mainly tested by relying on external people that manually exercise the system. While Murphy-Hill *et al.* already identified some alarm signals on the way game developers perform testing, we found statistically significant differences between games and non-games when considering the difficulties of game developers to write automated tests.

Malfunctions are handled differently. Faults are located in different areas of game and non-game systems. Our quantitative analysis on the systems' repositories showed that: On the one hand, games exhibit malfunctions in different file categories; on the other hand, non-games have a notably higher percentage of bug fixes in the *Programming* category.

Less clear release planning. For games, it is less clear whether developed features will be included in the next release than for non-games. This can be due to a more evident proneness to experimenting new features (including changing the game story or users' experience) that may or may not turn out to be successful. Interestingly this seems to be more evident for OSS projects, where we observe significant differences with respect to non-games. This could be because in industrial context there is a stricter or more anticipated release planning.

More difficulties in handling requirements. OSS game developers report to not being able to easily manage the evolution of the requirements. As a consequence, they cannot properly assess whether they have been met or not. Also in this case, this is something common between open source and industrial projects [33], however we identified statistical significant differences between the two domains.

6.2 Implications

The aforementioned findings have a number of implications for researchers, practitioners, and education. The former are called to the definition of a new, *ad-hoc* branch of software engineering, *i.e.*, *game-oriented* software engineering. More specifically:

(1) **Specializing requirement engineering for games.** Given the different audience and different expectations that end users have for games, new methodologies should be investigated to better support the activities of game developers during the management and the evolution of such user-driven requirements.

(2) **Game-specific source code reuse patterns.** Our results reported that developers are not able to reuse the source code of a game, likely because of system-dependent factors that make it hard to move in other projects. This represents an opportunity for researchers with respect to the definition of novel design patterns and methodologies that game developers can adopt to make the source code more extensible and reusable.

(3) **Game-specific fault localization.** Current approaches for detecting faults mainly target source code artifacts [13, 24, 36], while game development calls for different approaches in order to properly support developers in preventively adopt corrective actions.

(4) **Game-specific testing.** The use of manual testing and the challenges with automated testing in game systems highlight the need for a new set of methodologies to ease the developers' ability to identify malfunctions and to enable automatic testing activities for games. Studies could be conducted to investigate new generation record-replay tools, automated test data generators, etc.

Practitioners need to find appropriate ways to handle requirements, possibly closely involving end users during the whole software lifecycle, *e.g.*, by finding ways to ease communication with them as also mandated by Agile methodologies. Similarly, practitioners must be aware that faults can be not only hidden in the source code, but also in other types of resources: they are therefore called to adopt suitable tools supporting their activities.

Finally—confirming previous results of Murphy-Hill *et al.* [33]—our results suggest that game development requires skills that are strongly different from those required for the development of traditional software. Developers are indeed called to be more creative and able to work on a different variety of resources, *e.g.*, domain modeling files. Thus, it would be beneficial for students to have dedicated courses where gathering the required skills to approach game development.

7 THREATS TO VALIDITY

In this section we discuss possible threats to the validity of our study and how we mitigated them.

File and bug classification validity. A first threat might be related to how we identified the file categories in the context of $\mathbf{RQ}_1$. To ensure the comprehensiveness of the taxonomy, we adopted an iterative content analysis approach [30] on six projects of the dataset. However, we cannot exclude the missing analysis of specific file types out of the categories identified. For what concerns fault classification, we rely on a taxonomy by Ray *et al.* [41] and verified that, besides the need for adding one category (Algorithmic faults), such a taxonomy fits with the range of faults found in the analyzed projects.

Survey validity. Although maintaining high response rates is always desirable in a survey, research evidence indicates that open-source developers are massively assailed by interview and survey requests, therefore the response rate is lower than other contexts, *e.g.*, industrial participants [33]. Another considerable factor is that the response ratio decreases year by year how highlighted by Bartel [44]. In our study we tried to mitigate a low response rate typical of the open-source context reaching the most active developers of each project for a total of 600 invitations. Furthermore, as recommended by Flanigan *et al.* [17], we kept the survey as short as possible, so that participants could fill out it in no more than ten minutes. As a result, we obtained a response rate up to 15%, which is satisfactory for a study conducted with open-source subjects and in line with the minimum response rate suggested for survey studies [5].

Sample validity. A potential threat to validity of a research study conducted on a small sample of subjects is that it could deliver little knowledge. Even if there are historical evidence that shows otherwise (*e.g.*, Flyvbjerg [18] gave many examples where singular entity contributes to important discoveries in physics, economics, and social science) we selected a considerable amount of projects creating a sample of 60 heterogeneous open-source systems. Precisely, we selected 30 open-source games and 30 non-game projects with more than $1,000$ active contributors per month for games and more than $4,000$ active contributors per month for non-games. Moreover, these projects involve a representative population up to 40 millions of source lines for game projects and up to 45 millions of source lines for non-game projects.

Conclusion validity. Whenever appropriate, we support our claims with suitable statistical procedures. While in $\mathbf{RQ}_1$ and $\mathbf{RQ}_2$ we mainly report and discuss results through descriptive statistics, in $\mathbf{RQ}_3$ we corroborate the comparisons depicted with asymmetric stacked bar charts with Wilcoxon sum rank tests (adjusting p-values

using the Benjamini-Hochberg procedure [6]) and Cliff's delta effect size.

8 RELATED WORK

In the past, researchers have studied software development methodologies with the aim of improving the software development process and software quality in general. In the specific area of game development, researchers mainly focused on two aspects, (i) productivity [15, 26, 50] and (ii) social interaction [11, 22, 51].

Video games productivity. In the first direction, Kultima and Alha [28] explored how developers perceive the development process, interviewing 28 expert game programmers. They found that in this kind of software practitioners have different priorities than traditional systems, which are mainly imposed by end-users. This directly impacts on software requirements, enforcing engineers to work more on *non-functional* attributes than *functional*.

Another study aimed at understanding how developers build computer games has been conducted by Stacy and Nandhakumar [47]. Relying on the responses achieved conducting 20 interviews, they found that games are perceived as *"special"* kind of software. Therefore, applying traditional development processes to a game project may be dangerous with the consequence of having a low productivity. Callele *et al.* [10] investigated 50 dead projects, finding the reasons of their failure, discovering the issues that should be addressed with formal processes (*e.g.*, the transition from game design to formal requirements). Cristiano *et al.* [39] investigated the processes of software engineering in game development by analyzing 20 postmortem games. They discovered that iterative processes are the common techniques (such as *Agile* and *Waterfall*) adopted in game development.

Kasurinen *et al.* [27] interviewed 27 game developers to understand what are their expectations on tools aimed at supporting productivity. Even if many developers are concerned about the adaptability of development supporting tools, they are generally pleased with such instruments. Musil *et al.* [34] found that the *Agile* model is the most popular solution to support productivity. Fábio *et al.* [37] confirm the same findings on the *Agile* model.

A different perspective is given in the study by Lewis *et al.* [29], who aimed at producing a hierarchical taxonomy of faults that can appear in a game project. Differently from the taxonomy of malfunctions used in our study, Lewis *et al.* mainly focused on faults occurring in the Graphical User Interface rather than on the application logic. Therefore, we preferred the use of the more comprehensive taxonomy proposed by Ray *et al.* [41].

Lin *et al.* [31] studied the phenomenon of 0-day updates, *i.e.*, updates aimed at fixing bugs immediately after issuing a release, in the context of games present on the Steam platform. As a key result, they found that 0-day updates are more frequent in games having a frequent update release strategy. Fábio *et al.* [38] surveyed game developers to understand the common problems emerged during game development.

Video games as a social interaction. Burger-Helmchen and Cohendet [9] explored the mutual relationship between expert developers of industrial companies and communities of players. They found that games benefit of deep collaborations between firms and end users more than other software. Amin and Cohendet [3] conducted a large study having as object the importance for firms to recognize internal communities and derivable profit when members of different communities share knowledge. Similarly, Brown and Duguid [8] described how internal communities are viewed as suppliers of sense and collective beliefs for employees of firms.

Scacchi [43] investigated how free and open-source (FOSS) software communities interact to develop complex game projects. However, he found that FOSS community tend to ignore modern software engineering processes.

A different perspective is followed in the study by Murphy-Hill *et al.* [33]. They combined two studies to find possible gaps that are present in the development of games. In the first part of the study they interviewed 14 expert game developers, while in the second part they analyzed 364 surveys conducted in Microsoft. The results showed that developers recognize themselves in categories and such categories are defined based on their background. This observation somehow confirms previous findings about the creation of community with specialized background.

Our study has differences and commonalities with the work by Murphy-Hill *et al.* [33]. On the one hand, our qualitative analysis has the goal to compare the findings achieved in an industrial context with those obtained when considering open-source games. On the other hand, our study performs an additional quantitative analysis of 60 projects with the aim of looking at change and bug fixing activities performed by developers and comparing games with other kinds of software.

9 CONCLUSION

Game development has been shown as different with respect to that of non-game development for the industrial context [33]. In this paper, we further investigate this line of research by providing a large-scale empirical analysis on whether and how game developers perform different activities in the OSS context. We started with a software repository mining analysis involving 60 systems, aimed at measuring (i) how developers working in game and traditional software development contribute to their projects and (ii) how they handle malfunctions. Subsequently, we performed a survey targeting a total of 81 developers (45 developing games and 36 working on non-game systems), aimed at evaluating the developers' perception of their development process and activities, and differences between game vs. non-game systems.

Our findings highlight a number of points that can inspire the research community to define a new generation of software engineering tools, which explicitly target games and can help developers with the development, maintenance, and evolution of this special and widespread type of software systems.

ACKNOWLEDGMENTS

Bacchelli and Palomba gratefully acknowledges the support of the Swiss National Science Foundation through the SNF Project No. PP00P2 170529.

REFERENCES

[1] [n. d.]. Black Duck Open Hub. https://www.openhub.net. ([n. d.]). [Online; accessed 30-Jan-2019].

[2] [n. d.]. Microsoft Corporation. https://www.microsoft.com. ([n. d.]). [Online; accessed 30-Jan-2019].

[3] Ash Amin and Patrick Cohendet. 2004. *Architectures of knowledge: Firms, capabilities, and communities*. Oxford University Press on Demand.

[4] Apostolos Ampatzoglou and Ioannis Stamelos. 2010. Software engineering research for computer games: A systematic review. *Information and Software Technology* 52, 9 (2010), 888–901.

[5] Yehuda Baruch. 1999. Response rate in academic studies—A comparative analysis. *Human relations* 52, 4 (1999), 421–438.

[6] Yoav Benjamini and Yosef Hochberg. 1995. Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. *Journal of the Royal Statistical Society. Series B (Methodological)* 57, 1 (1995), 289–300.

[7] Christian Bird, Nachiappan Nagappan, Brendan Murphy, Harald Gall, and Premkumar Devanbu. 2011. Don't touch my code!: examining the effects of ownership on software quality. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, 4–14.

[8] John Seely Brown and Paul Duguid. 2001. Knowledge and organization: A social-practice perspective. *Organization science* 12, 2 (2001), 198–213.

[9] Thierry Burger-Helmchen and Patrick Cohendet. 2011. User communities and social software in the video game industry. *Long Range Planning* 44, 5 (2011), 317–343.

[10] David Callele, Eric Neufeld, and Kevin Schneider. 2005. Requirements engineering and the creative process in the video game industry. In *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*. IEEE, 240–250.

[11] Philip A Chan and Terry Rabinowitz. 2006. A cross-sectional analysis of video games and attention deficit hyperactivity disorder symptoms in adolescents. *Annals of General Psychiatry* 5, 1 (2006), 16.

[12] W. J. Conover. 1998. *Practical Nonparametric Statistics* (3rd edition ed.). Wiley.

[13] Dario Di Nucci, Fabio Palomba, Giuseppe De Rosa, Gabriele Bavota, Rocco Oliveto, and Andrea De Lucia. 2018. A developer centered bug prediction model. *IEEE Transactions on Software Engineering* 44, 1 (2018), 5–24.

[14] Nick Diakopoulos and Stephen Cass. 2017. The Top Programming Languages 2017. *IEEE Spectrum* http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2017 (Jul 2017).

[15] Melissa A Federoff. 2002. *Heuristics and usability guidelines for the creation and evaluation of fun in video games*. Ph.D. Dissertation. Indiana University Bloomington.

[16] Michael Fischer, Martin Pinzger, and Harald C. Gall. 2003. Populating a Release History Database from Version Control and Bug Tracking Systems. In *19th International Conference on Software Maintenance (ICSM 2003), The Architecture of Existing Systems, 22-26 September 2003, Amsterdam, The Netherlands*. 23.

[17] Timothy S Flanigan, Emily McFarlane, and Sarah Cook. 2008. Conducting survey research among physicians and other medical professionals: a review of current literature. In *Proceedings of the Survey Research Methods Section, American Statistical Association*, Vol. 1. 4136–47.

[18] Bent Flyvbjerg. 2006. Five misunderstandings about case-study research. *Qualitative inquiry* 12, 2 (2006), 219–245.

[19] Karl Fogel. 2005. *Producing open source software: How to run a successful free software project*. " O'Reilly Media, Inc.".

[20] Joseph F Francois and Kenneth A Reinert. 1997. *Applied methods for trade policy analysis: a handbook*. Cambridge University Press.

[21] Jeanne B Funk. 1993. Reevaluating the impact of video games. *Clinical pediatrics* 32, 2 (1993), 86–90.

[22] Tobias Greitemeyer. 2014. Intense acts of violence during video game play make daily life aggression appear innocuous: A new mechanism why violent video games increase aggression. *Journal of Experimental Social Psychology* 50 (2014), 52–56.

[23] Robert J. Grissom and John J. Kim. 2005. *Effect sizes for research: A broad practical approach* (2nd edition ed.). Lawrence Earlbaum Associates.

[24] Ahmed E Hassan. 2009. Predicting faults using the complexity of code changes. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*. IEEE, 78–88.

[25] William Jones, Ammy Jiranida Phuwanartnurak, Rajdeep Gill, and Harry Bruce. 2005. Don't take my folders away!: organizing personal information to get ghings done. In *CHI'05 extended abstracts on Human factors in computing systems*. ACM, 1505–1508.

[26] Jesper Juul. 2011. *Half-real: Video games between real rules and fictional worlds*. MIT press.

[27] Jussi Kasurinen, Jukka-Pekka Strandén, and Kari Smolander. 2013. What do game developers expect from development and design tools?. In *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*. ACM, 36–41.

[28] Annakaisa Kultima and Kati Alha. 2010. "Hopefully everything I'm doing has to do with innovation": Games industry professionals on innovation in 2009. In *Games Innovations Conference (ICE-GIC), 2010 International IEEE Consumer Electronics Society's*. IEEE, 1–8.

[29] Chris Lewis, Jim Whitehead, and Noah Wardrip-Fruin. 2010. What went wrong: a taxonomy of video game bugs. In *Proceedings of the fifth international conference on the foundations of digital games*. ACM, 108–115.

[30] William Lidwell, Kritina Holden, and Jill Butler. 2010. *Universal Principles of Design, Revised and Updated: 125 Ways to Enhance Usability, Influence Perception, Increase Appeal, Make Better Design Decisions, and Teach through Design* (2nd ed.). Rockport Publishers.

[31] 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.

[32] A Mockus and LG Votta. 2000. Identifying reasons for software changes using historic databases. In *Software Maintenance, 2000. Proceedings. International Conference on*. 120–130.

[33] Emerson Murphy-Hill, Thomas Zimmermann, and Nachiappan Nagappan. 2014. Cowboys, ankle sprains, and keepers of quality: How is video game development different from software development?. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, 1–11.

[34] J Musil, A Schweda, D Winkler, and S Biffl. 2010. A Survey on a State of the Practice in Video Game Development. *Technical report, QSE-IFS-10/04* (2010).

[35] A. N. Oppenheim. 1992. *Questionnaire Design, Interviewing and Attitude Measurement*. Pinter Publishers.

[36] Fabio Palomba, Marco Zanoni, Francesca Arcelli Fontana, Andrea De Lucia, and Rocco Oliveto. 2017. Toward a Smell-aware Bug Prediction Model. *IEEE Transactions on Software Engineering* (2017).

[37] Fabio Petrillo and Marcelo Pimenta. 2010. Is agility out there?: agile practices in game development. In *Proceedings of the 28th ACM International Conference on Design of Communication*. ACM, 9–15.

[38] Fábio Petrillo, Marcelo Pimenta, Francisco Trindade, and Carlos Dietrich. 2009. What went wrong? A survey of problems in game development. *Computers in Entertainment (CIE)* 7, 1 (2009), 13.

[39] 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 *Games and Software Engineering (GAS), 2016 IEEE/ACM 5th International Workshop on*. IEEE, 22–28.

[40] Baishakhi Ray, Vincent Hellendoorn, Saheel Godhane, Zhaopeng Tu, Alberto Bacchelli, and Premkumar Devanbu. 2016. On the naturalness of buggy code. In *Proceedings of the 38th International Conference on Software Engineering*. ACM, 428–439.

[41] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. 2014. A large scale study of programming languages and code quality in github. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 155–165.

[42] Per Runeson. 2006. A survey of unit testing practices. *IEEE software* 23, 4 (2006), 22–29.

[43] Walt Scacchi. 2004. Free and open source development practices in the game community. *IEEE software* 21, 1 (2004), 59–66.

[44] Kim Bartel Sheehan. 2001. E-mail survey response rates: A review. *Journal of Computer-Mediated Communication* 6, 2 (2001), 0–0.

[45] Brendan Sinclair. 2015. Gaming will hit $91.5 billion this year. http://www.gamesindustry.biz/articles/2015-04-22-gaming-will-hit-usd91-5-billion-this-year-newzoo (2015).

[46] Michael G Stabin. 1996. MIRDOSE: personal computer software for internal dose assessment in nuclear medicine. *Journal of Nuclear Medicine* 37, 3 (1996), 538–546.

[47] Patrick Stacey and Joe Nandhakumar. 2009. A temporal perspective of the computer game development process. *Information Systems Journal* 19, 5 (2009), 479–497.

[48] F Ted Tschang. 2007. Balancing the tensions between rationalization and creativity in the video games industry. *Organization Science* 18, 6 (2007), 989–1005.

[49] F Ted Tschang and Janusz Szczypula. 2006. Idea creation, constructivism and evolution as key characteristics in the videogame artifact design process. *European Management Journal* 24, 4 (2006), 270–287.

[50] Michael Washburn Jr, 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*. ACM, 280–289.

[51] Dmitri Williams, Nicole Martins, Mia Consalvo, and James D Ivory. 2009. The virtual census: Representations of gender, race and age in video games. *New Media & Society* 11, 5 (2009), 815–834.