Information and Software Technology 152 (2022) 107035

Contents lists available at ScienceDirect

Information and Software Technology

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

Different, Really! A comparison of Highly-Configurable Systems and Single Systems

Raphael Pereira de Oliveira [a,*], Paulo Anselmo da Mota Silveira Neto [b], Qi Hong Chen [d], Eduardo Santana de Almeida [c], Iftekhar Ahmed [d]

[a] *Department of Information Systems, Federal University of Sergipe, Brazil*
[b] *Computer Science Department, Federal Rural University of Pernambuco, Brazil*
[c] *Institute of Computing, Federal University of Bahia (IC-UFBA)*
[d] *Department of Informatics, University of California, Irvine, United States*

ARTICLE INFO

Keywords:
Highly-Configurable Systems (HCS)
Mining software repository (MSR)
Survey
Single System (SS)

ABSTRACT

Context: The development of systems that handle configuration options according to a specific environment is considered a hard activity. These kind of systems, Highly-Configurable Systems (HCS) are perceived by researchers and developers as complex and difficult to maintain due to the necessity of handling variation points. Although this perception is reported in the literature, no prior study investigated the differences between HCS and Single Systems (SS).
Objective: This study investigated similarities and differences between HCS and SS using well known metrics from the literature according to three different perspectives: *product perspective* (bug-proneness, complexity, and change size); *process perspective* (number of contributors, number of core developers, and accidental contributors); and *people perspective* (contributor retention and number of paid contributors).
Method: To perform this comparison, we collected data from two surveys and from a mining study (within 15,769 releases of 124 GitHub projects written in C).
Results: In general, we identified that for the majority of the metrics, the perception of practitioners and researchers about HCS and SS is different from our mining results.
Conclusion: The identification of similarities and differences of HCS and SS will help to initiate a discussion and further research in this direction.

1. Introduction

Modern software systems allow developers to configure the system according to a particular market niche [1]. Among many techniques, one way of doing so is through handling the variability in configuration using *features* [2], to handle the needs of a particular user. These kind of systems are referred to *Highly-Configurable Systems (HCS)* [3], allowing a large amount of variability configurations according to the user's needs and encompass large sets of features. Systems that do not allow any kind of variability configuration, or allow a small amount of variability configuration are referred to as *Single Systems (SS)*.

The ability to accommodate a large number of configurations requires additional implementations known as variation points, *i.e.*, elements in code, to manage variability and to facilitate the derivation of different configurations [4]. Due to the presence of variation points in HCS, researchers and developers alike perceive that HCS is more complex and challenging to maintain [5–7]. Such perception is highlighted in the following examples collected from literature: *"bug-finding is a time-consuming and tedious task in the presence of variability"* [5]; *"variability specifications and realizations tend to erode in the sense that they become overly complex"* [6].

However, there are numerous examples of long-held beliefs that proved to be incorrect or outdated when actual evidence was collected through empirical analysis [8,9]. Given that, and the prominence of HCS, it is well past time for investigating the difference between HCS and SS to answer questions, such as:

- RQ1. Are HCS more bug-prone than SS?
- RQ2. Is code complexity in HCS higher than SS?
- RQ3. Is code change size in SS bigger than in HCS?
- RQ4. Is the number of core developers in SS bigger than in HCS?
- RQ5. Is the number of code contributors in HCS bigger than in SS?

* Corresponding author.
E-mail address: raphael.oliveira@academico.ufs.br (R.P. de Oliveira).

https://doi.org/10.1016/j.infsof.2022.107035
Received 22 September 2021; Received in revised form 31 May 2022; Accepted 31 July 2022
Available online 5 August 2022

- RQ6. Is the number of accidental contributors in SS bigger than in HCS?
- RQ7. Is the developer retention higher in HCS than in SS?
- RQ8. Is there more paid contributors in SS than in HCS?

Identifying the similarities and differences between HCS and SS will help to initiate a discussion and further research. Observing a similarity of both types of systems would help us transfer insights, methods, and tools that have been proven useful for SS to HCS and were not considered for transfer due to the perceived differences. On the other hand, identifying differences would help us to design new techniques addressing the difference.

The overarching goal of this work is to cover the gap in research by understanding the similarities and differences between HCS and SS using well known metrics collected from literature pertaining to three different dimensions: the *product*, the *process* and, the *people*. According to Majumber et al. [10], product perspective is related to the product itself (code), process perspective is related to the software development process characteristics and, people perspective is related to resources required within the software development. We investigate bug-proneness, complexity, and change size which are relevant to the product dimension. Next, we explored process related factors, such as number of contributors, number of core developers and accidental contributors. Finally, we investigate the people perspective by analyzing contributor retention and number of paid contributors.

We started by inviting 100 practitioners and 100 researchers to participate in our formative survey. In our survey, we asked respondents to provide their opinion regarding differences between HCS and SS along the three different perspectives (product, process, and people). Based on their responses, we identified 8 metrics that participants anticipated to have difference between HCS and SS.

Next, we conducted a mining software repository study based on 15,769 releases of 124 GitHub projects (written in C). These projects were selected based on a list provided by Medeiros et al. [11], and based on the GitHub Stars [12]. All of the projects were classified into HCS or SS according to the number of `#ifdef` directives. These directives allow to deal with conditional compilation in C, which enable the programmer to include or exclude parts of the code base by providing a corresponding configuration [13].

We collected information regarding the 8 metrics identified through the formative survey for each one of the 15,769 releases. Moreover, we investigated the differences in order to find statistical significant results. Finally, we conducted a large-scale survey where we surveyed 45 practitioners and 50 researchers and asked them about their opinion regarding differences between HCS and SS in terms of the 8 metrics. We compare and contrast our findings identified through the large-scale survey and mining to show how belief and evidence differs when it comes to differentiate between HCS and SS.

Overall, the paper makes the following contributions:

- We report on the first large-scale analysis of similarities and differences between HCS and SS using 8 metrics across 124 different applications spanning over 15,769 releases.
- We present the findings of a survey involving 45 software practitioners and 50 researchers from 24 countries across 5 continents, which shed light on how practitioners and researchers perceive the similarities and differences between HCS and SS.
- We highlight the differences between the belief and actual evidence pertaining to the similarities and differences of HCS and SS.
- Based on our results, we outline implications for developers and researchers.

The remainder of the paper is structured as follows. Section 2 provides an overview of related work. Section 3 discusses our methodology, with Section 4 presenting our findings. Section 5 places our results in the broader context of work to date and outlines the implications for developers and researchers. Section 6 details threats to validity. Finally, Section 7 concludes with a summary of the key findings and an outlook at our future work.

2. Related work

The implementation of variability within HCS is mostly implemented using `#ifdef` directives in C language. Medeiros et al. [11] studied the `#ifdef` directives in C. Thus, they propose a catalog of refactorings to cope with `#ifdef's`. To validate their catalog, they selected 63 HCS written in C from the GitHub repository. `#ifdef's` directives allow to deal with conditional compilation and they handle variability according to the HCS definition. However, according to authors, these directives may have a negative impact on code understanding and maintainability.

Mining HCS repositories was the focus of several work. Lotufo et al. [14] and Israeli and Feitelson [15] were the first ones to analyze the evolution of the Linux kernel. Lotufo et al. [14] investigated the evolution of the Linux kernel feature model between revisions 12 and 32, a period extending over almost 5 years. They analyzed the model size, depth of leaves, constraints, and branching factor for each revision and identified six categories of reasons for changes in the Linux model. Israeli and Feitelson [15] investigated 810 versions of the Linux kernel, released over a period of 14 years, to characterize its evolution using Lehman's laws of software evolution [16]. Passos et al. [17] identified four evolution patterns in commits found in the Linux kernel repository. Next, they extended this work [18] inspecting over 500 commits relative to the addition and removal of features, spanning four years of Linux kernel development. The new catalog included 13 high-level evolution patterns capturing the coevolution of the Linux kernel variability model, Makefiles, and C source code. The work was later extended [19] improving the dataset and pattern identification process and identifying new evolution patterns. Passos et al. [20] analyzed the source code to understand feature scattering during eight years of the Linux kernel evolution and extended it with interviews and a survey with developers [21].

Different aspects of SS have been extensively researched by many researchers [22–25]. We were motivated to investigate product, process, and people following other works in SS, such as [10,26]. Majumder et al. [10] identified hero developers by mining data from more than 1100 GitHub projects. They grouped the analysis into 3 perspectives: product, process, and personnel. Product was represented by metrics related to code, process was related to software development metrics, and personnel metrics related to resource needed in the software development. Rahman et al. [26] analyzed the applicability and efficacy of software process and code metrics according to several perspectives.

In spite of decades of research, to the best of our knowledge, no prior work has analyzed the differences between HCS and SS. This paper aims to fill that gap by conducting the first study that provides empirical evidence regarding the differences between HCS and SS using a mixed-method approach combining mining of software repositories and surveys.

3. Methodology

Our study consisted of two parts: surveys and mining study of software repositories. The goal of the first survey was to get insights into how practitioners and researchers perceive the differences between HCS and SS along three different perspectives (product, process, and people). We then checked these perceptions with empirical evidence gathered from HCS and SS open-source projects followed by another round of survey of a larger sample size of practitioners and researchers for triangulating the findings.

3.1. Survey

Protocol. We used the similar protocol for both surveys. For the first survey, we created a 5-min survey where three questions were asked to collect developers and researchers opinion regarding differences between HCS and SS along three different perspectives (product, process, and people). Based on the Majumder et al. [10] work, we defined three *(yes or no)* questions:

- Is there any difference between HCS and SS related to product metrics (bug-proneness, complexity, and change size)?
- Is there any difference between HCS and SS related to process metrics (number of contributors, number of core developers, and accidental contributors)?
- Is there any difference between HCS and SS related to people metrics (contributor retention and number of paid contributors)?

Based on the responses of the first survey, we confirmed 8 metrics that participants anticipated to have difference between HCS and SS along the product, process, and people perspectives.

For the second survey,[1] we created a 10-min survey designed to compare and contrast our findings identified through a large-scale mining to show how belief and evidence differs when it comes to differentiate between HCS and SS. The second survey was composed of 8 closed questions on a 5-point Likert scale, from Strongly Disagree to Strongly Agree and short free-form text for any additional comment. The survey also collected demographic information from respondents. For the design of the surveys, we followed the Kitchenham and Pfleeger's guidelines for personal opinion surveys [27].

We piloted both surveys with three researchers (2 Ph.D.s and 1 Ph.D. student) with experience in the area to get feedback on the questions and their corresponding answers, difficulties faced to answer the survey and time to finish it. As these pilot respondents were experts in the area, we also would like to know if we were asking the right questions. We conducted several iterations of the survey and rephrased some questions and removed others to make the survey easier to understand and answer. Another concern in this stage was also to ensure that the participants could finish the surveys in 5-min and 10-min respectively. The responses from the pilot survey were used solely to improve the questions and these responses were not included in the final results. We kept the survey anonymous but in the end the respondents could inform their email to receive a summary of the study.

Respondents. We followed a two step approach to recruit survey respondents: first, we identified the average of commits per open source projects (HCS and SS) selected for the study (Section 3.2) and contacted the contributors that had more commits than this average. It was important to not select accidental contributors with few contributions in the project whose participation in the study could not bring any benefit in terms of experience and perceptions. In total, we selected 769 emails contacts from contributors. Next, we invited researchers who published papers from 2014–2020 in important conferences (ICSE, FSE/ESEC, ASE, SPLC) and journals (IEEE TSE, ACM TOSEM, EMSE). Regarding conferences, research and industry/practice tracks were considered. All the authors of the selected papers were contacted. In total, we selected 195 papers and 489 emails contacts excluding the duplicated ones.

Contact all potential respondents within a survey is a difficult task [28]. Moreover, selecting potential respondents may raise ethical concerns, such as contacting developers on GitHub [29] using their email addresses, which were not there for this purpose. Thus, to mitigate this threat, we did not selected all the potential respondents.

Of the 769 email contacts from contributors, we randomly selected 100 emails and out of 489 email contacts from researchers, we randomly selected 100 emails. We sent out our initial survey invitations to these 200 respondents. Out of which 54 responded and all emails were successfully delivered. This resulted in a 27% response rate.

For the second survey, we sent invitations to all 1258 contacts. Of the 1258 invitations, 176 of these were not successfully delivered and we received 27 automatic reply notifying the respondent's absence. One author replied saying that he worked in the study as a statistician expert and he did not have background to answer the survey. In total, 1054 invites were delivered and we received 108 responses (10.2% response rate) which we considered satisfactory, since other important studies in the field had reported response rates between 5.7% [21] and 7.9% [11].

From the responses of the second survey, we disqualified 13 responses without actual data (without responses to any of the survey questions of interest to the study despite responding to basic demographics questions, such as role of the respondent), leading to 95 valid responses that were considered. The respondents spread out in 24 countries across 5 continents. The top three countries where the respondents come from are Germany, Brazil, and United States. The professional experience of these 95 respondents vary from 1 years to 40 years, with an average of 14.93 years and median of 12 years. 17.9% of the respondents have a Bachelor's degree, 77.9% have an advanced degree, i.e., Master's or Ph.D., and the other ones have graduated high school, trade/technical school, associate degree, and no degree (1% each). 45 respondents are practitioners and 50 researchers.

Data Analysis. We collected the ratings that our respondents provided for each question for the second survey. Next, we used for ratings the Likert scores from 1 (Strongly Disagree) to 5 (Strongly Agree) and computed the average Likert score of each statement related to product, process, and people perspectives. In addition, we extracted comments that the survey respondents gave to explain the reason why they strongly disagree/disagree or agree/strongly agree with each perspective (Table 1).

3.2. Repository mining

Selecting and filtering projects. We started with 63 HCS projects used by Medeiros et al. [11] in their study, which, according to the authors, were selected based on: 40 projects used in previous studies [39–44], covering different sizes; and 23 most active projects using preprocessor directives in Github (a project was considered active based on its higher number of pull requests opened and closed). Out of these 63 projects, we excluded projects with less than 2 releases or projects without GitHub. Since our analysis is based on GitHub data, we kept only projects that have GitHub. Thus, we selected 44 projects from Medeiros et al. [11]. Moreover, we select the top 900 GitHub projects (written in C) based on the stars that the project has at GitHub. According to Valente et al. [12], the number of stars that a project has at GitHub is useful to measure its popularity. Thus, we selected only projects with more than 1000 stars. From these 900 GitHub projects, we identified HCS and SS candidates. The HCS projects were selected based on: the number of releases must be greater or equal to 2; the number of `#ifdef` directives must be greater than 5; the year of the last commit must be greater or equal to 2021; the project documentation should be written in English; and the project should not be in the Medeiros et al. [11] list. At this point we identified 484 HCS Projects. The SS projects were selected based on: the number of releases must be greater or equal to 2; the number of `#ifdef` directives must be smaller or equal to 5; the year of the last commit must be greater or equal to 2021; the project documentation should be written in English; and the project should not be a tutorial. Thus, we identified 92 SS Projects. In order to have the same amount of HCS projects, we selected all the 44 HCS projects from Medeiros et al. [11] and selected the top 48 HCS projects (from the 484 HCS identified). With 92 HCS projects and 92 SS projects, we started to collect data for each one of the metrics from Table 1. However, some projects did not have available some of the metrics. Bug-proneness and complexity were not available for some SS projects. Thus, we excluded these projects and ended up with 62

[1] https://github.com/raphaufs/hcsSS/.

Table 1
Metrics used in the analysis.

Perspective	Metric	Definition	Ref.
Product	Bug-Proneness	Number of bug-fix commits divided by total number of commits	[30]
	Code complexity	Cyclomatic complexity of code	[31,32]
	Change size	Number of modifications in a commit (insertions and deletions)	[33]
Process	Number of core developers	Number of GitHub developers doing most of the work in a project	[34]
	Number of code contributors	Number of GitHub contributors in a project	[35]
	Number of accidental contributors	Number of GitHub contributors with one contribution in a project	[36]
People	Developer retention	Number of contributors working longer in a project	[37]
	Paid contributors	Number of paid contributors	[38]

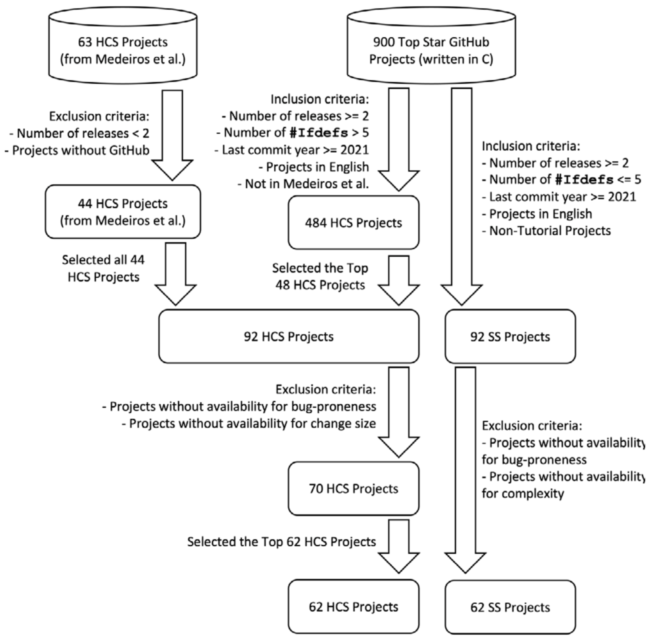

63 HCS Projects (from Medeiros et al.)
900 Top Star GitHub Projects (written in C)
Exclusion criteria: - Number of releases < 2 - Projects without GitHub
Inclusion criteria: - Number of releases >= 2 - Number of #Ifdefs > 5 - Last commit year >= 2021 - Projects in English - Not in Medeiros et al.
Inclusion criteria: - Number of releases >= 2 - Number of #Ifdefs <= 5 - Last commit year >= 2021 - Projects in English - Non-Tutorial Projects
44 HCS Projects (from Medeiros et al.)
484 HCS Projects
Selected all 44 HCS Projects
Selected the Top 48 HCS Projects
92 HCS Projects
92 SS Projects
Exclusion criteria: - Projects without availability for bug-proneness - Projects without availability for change size
Exclusion criteria: - Projects without availability for bug-proneness - Projects without availability for complexity
70 HCS Projects
Selected the Top 62 HCS Projects
62 HCS Projects
62 SS Projects

Fig. 1. Selecting and filtering projects for mining.

SS projects. The same happened for HCS projects when we started to collect data. Bug-proneness and change size were not available for some HCS projects. We removed these projects resulting in 70 HCS projects. In order to have the same amount of SS projects, we selected the top 62 HCS projects. This whole process is shown in Fig. 1. The final list of projects is presented in Table 2 for HCS projects and in Table 3 for SS projects. Next, we present how each metric was identified and measured for the repository mining process. All the scripts created for collecting the metrics are available on-line.[2]

Measuring Bug-Proneness. The Bug-Proneness metric was measured using a python script which identifies the bug-fixing commits within a GitHub Project. It considers the active files in the repository and counts the number of bug-fix commits based on the presence of specific keywords in the commit message. We adopt a method similar to that used in previous studies [45–49] to identify the commits. Keywords like *fix, defect, error, bug, issue, mistake, incorrect, fault, and flaw* and their variations were considered by the python script. Bug-Proneness metric is calculated by dividing the total number of bug-fix commits by total number of commits. Thus, a bug-proneness close to one indicates a higher number of problems within the project. A bug-proneness equal to zero indicates no problem at all. Tables 2 and 3 show the average of bug commits for all the identified releases in HCS and SS projects, respectively.

Measuring Code Complexity. Code Complexity in this paper refers to the McCabe Cyclomatic Complexity [31,32]. Code Complexity metric was evaluated using the KernelHaven[3] tool for HCS and the Understand tool[4] for SS. KernelHaven tool calculates the cyclomatic complexity including the complexity within variation points for HCS projects. On the other hand, for SS the Understand tool allowed to obtain the McCabe Cyclomatic Complexity. A project cyclomatic complexity is considerable bad if its value is higher than 4 [50]. Tables 2 and 3 show the complexity average for all the identified releases within the HCS and SS projects, respectively.

Measuring Change size. To identify the Change Size metric, we used a python script to collect, for each release, the number of insertions and deletions within the code. The total number of modifications (insertions plus deletions) within a source code represents this metric. This metric was statistically normalized to adjust the different scales found. Thus, if the value of this metric is close to one, it means a higher number of modifications. If it is close to zero, it represents a low level of modifications. Tables 2 and 3 show the Change Size average for all the identified releases in HCS and SS projects, respectively.

Measuring Number of Core Developers. To calculate the number of core developers we created a python script. We used the number of commits in the code base as a criterion to decide if a developer is core or non-core member in the project [34,51]. Open source contribution follows a power law, where 20% of contributors are responsible for 80% of the contributions [34]. We follow the same rule to identify the core and non-core developers. We consider a developer as *core* if the developer is in the top 20% of developers in that project (calculated by the number of commits authored). Otherwise the developer is *non-core*. The higher the value obtained for this metric the higher is the number of developers within these 20%. Tables 2 and 3 show the Number of Core Developers average for all the identified releases within the HCS and SS projects, respectively.

Measuring Number of Code Contributors. To count the number of code contributors we built a python script to access all contributors that had one or more commits per release for each project and summed the total number of contributors that had one or more commits per release. Tables 2 and 3 show the average Number of Code Contributors for all the identified releases in HCS and SS projects, respectively.

Measuring Number of Accidental Contributors. We also built a python script to collect accidental contributors. We consider an Accidental Contributor, a developer who has only one contribution (commit) in a release [36]. Tables 2 and 3 display the average of Accidental Contributors for all the identified releases within HCS and SS projects, respectively.

Measuring Developer retention. To identify the developer retention, we created a python script. A developer retention is defined as a developer that has a commit within the last 180 days and had at least one commit before these 180 days. The choice of the threshold is motivated by previous studies [37,52]. Tables 2 and 3 show the average of Developer Retention for all the identified releases in HCS and SS projects, respectively.

Measuring Paid Contributor. We also created a python script to collect paid contributors. We used the email address to identify if a

[2] https://github.com/raphaufs/hcsSS/.

[3] https://github.com/KernelHaven/KernelHaven.
[4] https://scitools.com/.

Table 2
Summary of analyzed data for the 62 HCS projects.

Highly-Configurable systems											
#	Project	GitHub	Rel[a]	Bug[b]	Compl[c]	CSize[d]	CDev[e]	CCont[f]	ACont[g]	DRet[h]	PCont[i]
1	akimd/bison	https://github.com/akimd/bison	132	0.15	4.70	0.03323	1.3409	37.31	0.00	1.00	0.05948
2	alliedmodders/amxmodx	https://github.com/alliedmodders/amxmodx	27	0.23	5.42	0.07857	1.3704	9.96	0.00	3.00	0.02602
3	allinurl/goaccess	https://github.com/allinurl/goaccess	40	0.03	3.73	0.00633	1.1250	62.00	0.00	4.00	0.11524
4	angband/angband	https://github.com/angband/angband	550	0.14	0.07	0.09468	0.0291	0.20	0.01	0.38	0.01859
5	arut/nginx-rtmp-module	https://github.com/arut/nginx-rtmp-module	660	0.04	6.77	0.01353	0.1712	0.76	0.00	1.00	0.01859
6	asfadmin/ASF/MapReady	https://github.com/asfadmin/ASF/MapReady	15	0.18	5.74	0.05959	1.2000	29.73	0.00	1.00	0.06134
7	balabit/syslog-ng	https://github.com/balabit/syslog-ng	110	0.22	0.40	0.06340	0.2182	1.18	0.37	2.51	0.04275
8	bilibili/ijkplayer	https://github.com/bilibili/ijkplayer	79	0.10	2.93	0.01947	1.0759	33.01	0.00	4.00	0.01859
9	cherokee/webserver	https://github.com/cherokee/webserver	19	0.11	4.37	0.05189	0.9474	16.95	0.00	2.00	0.04833
10	Cisco-Talos/clamav-devel	https://github.com/Cisco-Talos/clamav-devel	148	0.19	1.14	0.56894	0.4122	1.17	0.01	1.26	0.01115
11	ClusterLabs/pacemaker	https://github.com/ClusterLabs/pacemaker	137	0.09	5.53	0.24288	1.1022	99.77	0.00	11.00	0.23606
12	collectd/collectd	https://github.com/collectd/collectd	162	0.11	5.21	0.06123	1.2593	135.78	0.00	4.00	0.52230
13	curl/curl	https://github.com/curl/curl	199	0.20	1.45	0.11370	0.2663	0.65	0.05	1.33	0.00929
14	emscripten-core/emscripten	https://github.com/emscripten-core/emscripten	397	0.15	0.58	0.57188	0.0000	18.01	3.96	46.06	0.14870
15	FFmpeg/FFmpeg	https://github.com/FFmpeg/FFmpeg	349	0.18	0.13	0.37779	0.0029	11.43	3.37	11.23	0.75465
16	freedesktop/xorg-xserver	https://github.com/freedesktop/xorg-xserver	488	0.11	0.82	0.16184	0.2398	0.72	0.18	0.90	0.01115
17	FreeRADIUS/freeradius-server	https://github.com/FreeRADIUS/freeradius-server	105	0.15	6.33	0.14075	1.5048	74.93	0.00	4.00	0.18216
18	gentilkiwi/mimikatz	https://github.com/gentilkiwi/mimikatz	12	0.10	5.11	0.00018	0.9167	5.50	0.00	3.00	0.00558
19	Genymobile/scrcpy	https://github.com/Genymobile/scrcpy	27	0.16	2.90	0.00838	1.5926	40.96	29.19	3.00	0.01301
20	git/git	https://github.com/git/git	827	0.23	5.06	0.25504	1.0834	952.23	3.34	1.00	0.14126
21	gitGNU/gnu/bash	https://github.com/gitGNU/gnu/bash	27	0.00	4.46	0.33907	1.0741	2.33	0.00	1.00	0.00558
22	glennrp/libpng	https://github.com/glennrp/libpng	1616	0.02	5.21	0.03466	1.0223	7.19	0.00	4.00	0.02974
23	GNOME/gnumeric	https://github.com/GNOME/gnumeric	274	0.24	3.69	0.76856	0.3029	310.55	0.00	6.00	0.53160
24	GNOME/libsoup	https://github.com/GNOME/libsoup	309	0.37	0.56	0.02598	0.6181	2.13	0.79	2.80	0.04833
25	GNOME/libxml2	https://github.com/GNOME/libxml2	193	0.48	6.23	0.21681	0.9948	102.84	0.00	1.00	0.10595
26	GNOME/totem	https://github.com/GNOME/totem	252	0.29	2.83	0.11246	1.0079	382.05	0.00	7.00	0.73606
27	gnuplot/gnuplot	https://github.com/gnuplot/gnuplot	53	0.08	6.24	0.05798	1.4528	127.85	0.00	2.00	0.03160
28	hashcat/hashcat	https://github.com/hashcat/hashcat	27	0.11	6.49	0.36284	1.8519	61.48	0.00	10.00	0.12825
29	hexchat/hexchat	https://github.com/hexchat/hexchat	23	0.13	3.65	0.10603	1.6087	90.57	0.00	2.00	0.07993
30	iovisor/bcc	https://github.com/iovisor/bcc	34	0.19	3.89	0.10939	0.5294	245.68	0.00	7.00	0.22862
31	irssi/irssi	https://github.com/irssi/irssi	72	0.14	1.57	0.02555	0.3750	1.81	0.39	1.88	0.00372
32	jarun/nnn	https://github.com/jarun/nnn	36	0.03	1.34	0.00190	1.3056	66.58	0.00	6.00	0.06320
33	krb5/krb5	https://github.com/krb5/krb5	302	0.15	0.13	0.16274	0.0298	0.31	0.03	0.32	0.02602
34	libuv/libuv	https://github.com/libuv/libuv	231	0.24	4.12	0.04523	0.0000	31.36	14.15	57.04	0.11152
35	MapServer/MapServer	https://github.com/MapServer/MapServer	188	0.12	7.44	0.11736	0.9521	43.33	0.00	6.00	0.14684
36	memcached/memcached	https://github.com/memcached/memcached	104	0.27	4.63	0.01552	1.1538	136.42	0.00	1.00	0.08922
37	micropython/micropython	https://github.com/micropython/micropython	52	0.19	3.09	0.20307	1.9615	147.48	0.00	5.00	0.09108
38	mirror/busybox	https://github.com/mirror/busybox	156	0.22	6.12	0.10574	1.4936	109.86	0.00	2.00	0.05019
39	mpv-player/mpv	https://github.com/mpv-player/mpv	81	0.20	3.02	0.06684	0.0000	98.26	30.36	813.38	0.14126
40	netdata/netdata	https://github.com/netdata/netdata	58	0.51	1.00	0.05698	0.0690	0.31	2.53	11.59	0.07807
41	nginx/nginx	https://github.com/nginx/nginx	548	0.27	0.00	0.02844	1.0328	14.34	0.00	9.00	0.10409
42	obsproject/obs-studio	https://github.com/obsproject/obs-studio	159	0.19	1.25	0.08610	0.0000	26.01	14.57	50.85	0.11152
43	OpenSC/OpenSC	https://github.com/OpenSC/OpenSC	39	0.28	0.98	0.03493	1.6923	6.41	0.92	9.33	0.02788
44	openssl/openssl	https://github.com/openssl/openssl	339	0.16	0.49	0.25062	0.3304	0.71	0.01	0.92	0.01859
45	opentx/opentx	https://github.com/opentx/opentx	132	0.20	1.36	0.21161	2.2424	11.96	1.83	22.35	0.02230
46	OpenVPN/openvpn	https://github.com/OpenVPN/openvpn	73	0.24	0.97	0.05107	0.2055	0.21	0.00	0.21	0.00186
47	openwrt/openwrt	https://github.com/openwrt/openwrt	43	0.19	3.79	0.49328	0.8140	606.21	0.00	26.00	0.12825
48	ossec/ossec-hids	https://github.com/ossec/ossec-hids	44	0.33	0.96	0.05833	0.1136	0.25	0.11	0.57	0.00000
49	php/php-src	https://github.com/php/php-src	1179	0.22	6.66	1.00000	0.0611	546.50	0.00	61.00	1.00000
50	qmk/qmk/firmware	https://github.com/qmk/qmk/firmware	1457	0.28	3.07	0.72270	0.0096	1389.33	0.00	9.00	0.40706
51	radareorg/radare2	https://github.com/radareorg/radare2	101	0.21	5.53	0.29210	1.0297	622.98	0.00	10.00	0.10781
52	redis/redis	https://github.com/redis/redis	266	0.17	4.12	0.08450	1.2218	183.14	115.52	4.00	0.35130
53	robertdavidgraham/masscan	https://github.com/robertdavidgraham/masscan	12	0.15	5.33	0.01243	1.6667	23.42	0.00	3.00	0.04833
54	robol/MPSolve	https://github.com/robol/MPSolve	11	0.28	3.46	0.01468	0.6364	5.45	2.55	7.64	0.01859
55	sleuthkit/sleuthkit	https://github.com/sleuthkit/sleuthkit	48	0.26	1.83	0.17065	0.2083	0.21	0.00	0.21	0.00186
56	sqlite/sqlite	https://github.com/sqlite/sqlite	137	0.15	4.27	0.18190	1.0146	29.94	0.00	4.00	0.06691
57	TauLabs/TauLabs	https://github.com/TauLabs/TauLabs	10	0.11	3.01	0.40632	0.9000	160.30	0.00	2.00	0.22305
58	timescale/timescaledb	https://github.com/timescale/timescaledb	87	0.24	2.58	0.06926	0.4138	53.39	0.00	8.00	0.05204
59	tmux/tmux	https://github.com/tmux/tmux	34	0.12	5.09	0.00704	1.3235	24.82	0.00	6.00	0.08550
60	unbit/uwsgi	https://github.com/unbit/uwsgi	123	0.26	6.00	0.02965	0.7967	116.54	0.00	3.00	0.29740
61	ventoy/Ventoy	https://github.com/ventoy/Ventoy	69	0.06	5.29	0.01656	1.1884	27.70	0.00	2.00	0.00186
62	wiredtiger/wiredtiger	https://github.com/wiredtiger/wiredtiger	479	0.17	0.59	0.07850	0.2171	1.36	0.27	1.92	0.02974
		HCS Totals	13,961	0.18	3.43	0.00785	0.5510	286.66	3.30	16.48	0.13350

[a]Total # of Releases.
[b]Bug-Proneness Average.
[c]Code Complexity Average.
[d]Change Size Average.
[e]Core Developers Average.
[f]Code Contributors Average.
[g]Accidental Contributors Average.
[h]No. of Developer Retention.
[i]Paid Contributors Average.

Table 3
Summary of analyzed data for the 62 SS projects.

Single systems												
#	Project	GitHub	Rel[a]	Bug[b]	Compl[c]	CSize[d]	CDev[e]	CCont[f]	ACont[g]	DRet[h]	PCont[i]
1	adafruit/Adafruit-GFX-Library	https://github.com/adafruit/Adafruit-GFX-Library	81	0.12	12.92	0.04514	1.3580	35.60	0.00	2.00	0.14595
2	altdesktop/playerctl	https://github.com/altdesktop/playerctl	18	0.24	3.93	0.00541	1.0000	10.72	0.00	2.00	0.07568
3	AltraMayor/f3	https://github.com/AltraMayor/f3	17	0.53	2.98	0.00272	0.9412	6.06	0.00	2.00	0.07027
4	ardagnir/athame	https://github.com/ardagnir/athame	14	0.20	8.12	0.13233	1.0000	6.14	0.00	1.00	0.00541
5	atc1441/ATC/MiThermometer	https://github.com/atc1441/ATC/MiThermometer	51	0.01	2.72	0.00228	0.9804	16.67	0.00	0.00	0.03243
6	basil00/Divert	https://github.com/basil00/Divert	20	0.29	8.90	0.01613	0.9500	2.80	0.00	2.00	0.01081
7	benhoyt/inih	https://github.com/benhoyt/inih	25	0.25	2.83	0.00038	1.6800	12.88	0.00	1.00	0.01081
8	billziss-gh/sshfs-win	https://github.com/billziss-gh/sshfs-win	46	0.01	4.73	0.00046	0.1957	0.76	0.22	3.00	0.01081
9	billziss-gh/winfsp	https://github.com/billziss-gh/winfsp	61	0.07	4.15	0.08273	0.9836	9.00	1.77	2.00	0.05946
10	canonical/dqlite	https://github.com/canonical/dqlite	35	0.04	2.39	0.02911	0.9714	9.57	6.51	3.00	0.04324
11	citusdata/cstore/fdw	https://github.com/citusdata/cstore/fdw	14	0.15	2.96	0.00202	1.5000	10.43	0.00	2.00	0.05405
12	citusdata/pg/cron	https://github.com/citusdata/pg/cron	14	0.15	4.23	0.00215	1.8571	9.43	0.00	4.00	0.06486
13	danielfrg/word2vec	https://github.com/danielfrg/word2vec	11	0.00	12.27	0.02616	2.0000	15.73	9.55	1.00	0.03784
14	datatheorem/TrustKit	https://github.com/datatheorem/TrustKit	30	0.05	2.82	0.04061	1.4333	18.23	0.00	1.00	0.14054
15	eclipse/mosquitto	https://github.com/eclipse/mosquitto	63	0.16	5.20	0.05736	1.0000	69.59	40.11	4.00	0.15676
16	eradman/entr	https://github.com/eradman/entr	43	0.05	3.04	0.00157	1.2093	5.12	0.00	1.00	0.02162
17	ExistentialAudio/BlackHole	https://github.com/ExistentialAudio/BlackHole	11	0.03	7.65	0.00123	1.8182	5.91	2.09	3.00	0.01081
18	F5OEO/rpitx	https://github.com/F5OEO/rpitx	3	0.15	3.02	0.00412	1.3333	6.33	3.67	1.00	0.01622
19	FeralInteractive/gamemode	https://github.com/FeralInteractive/gamemode	11	0.21	3.74	0.00859	1.0000	22.36	13.82	2.00	0.06486
20	gamelinux/passivedns	https://github.com/gamelinux/passivedns	18	0.08	6.05	0.00300	1.0556	6.61	0.00	1.00	0.08649
21	gnif/LookingGlass	https://github.com/gnif/LookingGlass	35	0.12	4.86	0.04907	1.0571	28.77	19.34	1.00	0.02162
22	google/brotli	https://github.com/google/brotli	17	0.29	5.23	0.08163	1.6471	44.00	24.88	2.00	0.14054
23	haad/proxychains	https://github.com/haad/proxychains	6	0.29	6.31	0.00253	1.1667	13.83	5.33	1.00	0.07568
24	HardySimpson/zlog	https://github.com/HardySimpson/zlog	31	0.26	4.41	0.01832	0.9677	5.90	0.00	1.00	0.12432
25	henrypp/memreduct	https://github.com/henrypp/memreduct	11	0.38	1.09	0.00756	2.1818	11.27	0.00	1.00	0.02703
26	henrypp/simplewall	https://github.com/henrypp/simplewall	123	0.04	1.52	0.04404	1.4065	8.44	3.10	5.00	0.07027
27	hoytech/vmtouch	https://github.com/hoytech/vmtouch	11	0.07	6.37	0.00169	1.0909	12.18	0.00	1.00	0.06486
28	icholy/ttygif	https://github.com/icholy/ttygif	7	0.14	2.50	0.00062	1.8571	26.00	18.00	1.00	0.05946
29	inotify-tools/inotify-tools	https://github.com/inotify-tools/inotify-tools	15	0.20	5.10	0.00794	1.3333	28.67	19.33	0.00	0.05946
30	iovisor/gobpf	https://github.com/iovisor/gobpf	3	0.21	2.71	0.00026	1.0000	59.33	0.00	2.00	0.17297
31	jedisct1/minisign	https://github.com/jedisct1/minisign	11	0.04	3.96	0.00177	0.9091	4.91	0.00	3.00	0.04865
32	jhawthorn/fzy	https://github.com/jhawthorn/fzy	11	0.02	2.34	0.00329	0.9091	7.09	4.09	2.00	0.05405
33	jorisvink/kore	https://github.com/jorisvink/kore	35	0.10	4.01	0.01175	0.9714	34.31	18.09	1.00	0.09730
34	jpmens/jo	https://github.com/jpmens/jo	14	0.15	4.61	0.00329	1.0000	16.57	9.43	1.00	0.10811
35	karlstav/cava	https://github.com/karlstav/cava	22	0.26	9.22	0.00565	1.9091	24.77	9.45	3.00	0.16216
36	kornelski/pngquant	https://github.com/kornelski/pngquant	81	0.08	4.60	0.00722	1.6543	27.85	12.91	2.00	0.10270
37	krallin/tini	https://github.com/krallin/tini	38	0.09	4.87	0.00270	0.9737	5.45	4.29	1.00	0.03784
38	leahneukirchen/nq	https://github.com/leahneukirchen/nq	8	0.13	9.94	0.00017	1.0000	4.38	1.88	1.00	0.00541
39	lxc/lxc	https://github.com/lxc/lxc	123	0.22	5.14	0.15005	0.4309	256.61	142.30	0.00	1.00000
40	mintty/wsltty	https://github.com/mintty/wsltty	59	0.17	2.40	0.00085	1.1186	6.02	3.03	1.00	0.01622
41	muennich/sxiv	https://github.com/muennich/sxiv	26	0.07	5.14	0.00474	1.1154	13.23	0.00	1.00	0.08649
42	netblue30/firejail	https://github.com/netblue30/firejail	59	0.23	8.10	0.12479	0.9831	130.15	64.08	9.00	0.92973
43	philippe44/AirConnect	https://github.com/philippe44/AirConnect	21	0.02	5.08	0.00339	1.9048	10.33	5.00	2.00	0.02162
44	PromyLOPh/pianobar	https://github.com/PromyLOPh/pianobar	30	0.15	5.34	0.00967	2.6000	25.33	0.00	1.00	0.04865
45	rbsec/sslscan	https://github.com/rbsec/sslscan	55	0.12	14.37	0.00841	1.5091	29.58	0.00	3.00	0.17838
46	RedBeardLab/rediSQL	https://github.com/RedBeardLab/rediSQL	30	0.00	2.63	0.53773	1.9333	9.03	0.00	4.00	0.02162
47	stlink-org/stlink	https://github.com/stlink-org/stlink	12	0.17	4.12	0.02862	0.5833	151.33	90.17	5.00	0.62703
48	swaywm/sway	https://github.com/swaywm/sway	81	0.14	4.74	0.05753	0.9877	202.42	90.54	4.00	0.77297
49	Syllo/nvtop	https://github.com/Syllo/nvtop	12	0.15	4.69	0.00654	1.2500	6.67	4.17	2.00	0.02162
50	symisc/sod	https://github.com/symisc/sod	3	0.33	2.97	0.00180	0.6667	0.67	0.00	1.00	0.00000
51	Sysinternals/ProcDump-for-Linux	https://github.com/Sysinternals/ProcDump-for-Linux	6	0.34	5.02	0.00124	1.5000	8.83	5.50	1.00	0.02703
52	taviso/ctftool	https://github.com/taviso/ctftool	5	0.00	3.44	0.00016	0.8000	1.00	0.00	0.00	0.00000
53	tideways/php-xhprof-extension	https://github.com/tideways/php-xhprof-extension	44	0.01	2.47	0.00968	0.9773	42.25	0.00	2.00	0.20541
54	uber/h3	https://github.com/uber/h3	29	0.12	3.40	1.00000	1.1724	24.24	8.07	1.00	0.08649
55	ultrajson/ultrajson	https://github.com/ultrajson/ultrajson	19	0.27	3.95	0.30531	0.3158	61.63	34.63	2.00	0.22703
56	ValdikSS/GoodbyeDPI	https://github.com/ValdikSS/GoodbyeDPI	27	0.03	5.48	0.00535	0.9630	3.67	1.81	1.00	0.01081
57	visit1985/mdp	https://github.com/visit1985/mdp	24	0.14	6.47	0.00259	1.1250	21.00	14.88	1.00	0.10811
58	vozlt/nginx-module-vts	https://github.com/vozlt/nginx-module-vts	20	0.39	5.30	0.02255	1.2000	3.05	0.20	2.00	0.01081
59	wg/wrk	https://github.com/wg/wrk	21	0.04	4.76	0.10335	1.0000	1.00	0.05	1.00	0.00541
60	Xfennec/progress	https://github.com/Xfennec/progress	19	0.07	6.24	0.00204	1.2105	14.68	7.42	2.00	0.07568
61	yaoweibin/nginx/tcp/proxy/module	https://github.com/yaoweibin/nginx/tcp/proxy/module	12	0.23	19.08	0.02021	0.9167	3.17	0.00	1.00	0.03243
62	zauonlok/renderer	https://github.com/zauonlok/renderer	6	0.02	2.14	0.35463	1.0000	2.00	0.00	1.00	0.00000
		SS Totals	1808	0.13	5.19	0.00514	1.1571	47.62	21.49	2.22	0.11395

[a]Total # of Releases.
[b]Bug-Proneness Average.
[c]Code Complexity Average.
[d]Change Size Average.
[e]Core Developers Average.
[f]Code Contributors Average.
[g]Accidental Contributors Average.
[h]No. of Developer Retention.
[i]Paid Contributors Average.

developer is a paid contributor or not. We excluded email addresses belonging to the following domains, such as: @gmail, @googlemail, @local, @me.com, @instance-1, @github, @live, @hotmail, @yahoo, @none. Finally, we manually checked all the remained mails in order to have only company mails. The values for this metric were statistically normalized to adjust the different scales. Thus, the closer this metric is to one, the higher is the number of paid contributors. Otherwise, if this metric is zero, it means that the project does not have paid contributors. Tables 2 and 3 show the average of paid contributors for all the identified releases within HCS and SS projects, respectively.

3.3. Statistical analysis

After collecting the data for the experiment, we used descriptive statistics, box plots, and statistical tests to perform the analysis. As is usual, in all the tests, we accepted a probability of 5% of committing a Type-I Error [53], i.e., rejecting the null hypothesis when it is actually true. The data analysis was carried out by considering the following steps:

- *Survey and Mining:* We first carried out a descriptive study of the measures for the dependent variables.
- *Survey:* Within the survey data, we applied the Fisher's Test [54] to identify significant difference between the researchers and practitioners answers. A Fisher's Test *p-value* less than 0.05 represents a significant difference between researchers and practitioners.
- *Mining:* We analyzed the characteristics of the data in order to determine which test would be most appropriate to test our hypotheses. For testing normality, we applied the Kolmogorov–Smirnov Test [55]. This test is recommended to check for normality within large samples (more than 5000). Since in our analysis we have 15,769 releases from the mining repository process, we applied this test for normality.
- *Mining:* When the results of the aforementioned test (Kolmogorov–Smirnov Test) shows a non normal distribution (*p-value* less than 0.05), we applied the Wilcoxon Test [56] to identify significant statistical difference between HCS and SS.
- *Mining:* Our null hypothesis states that there is no difference between HCS and SS projects. Our alternative hypothesis states that there is a difference between HCS and SS projects. Since Kolmogorov–Smirnov Test shown a non normal distribution for all variables, we applied only the Wilcoxon Test to test our hypotheses. A Wilcoxon Test result less than 0.05 (*p-value* less than 0.05) shows a significant difference between HCS and SS projects. If *p-value* result is greater or equal to 0.05, it means that there is no difference between HCS and SS projects.
- *Mining:* Furthermore, the statistical significance of the Wilcoxon Test results were complemented with the magnitude of their effects. Since our data is non-parametric we used the Cliff's δ to measure the effect size [57]. Its result (delta estimate) shows how the two means differentiate from each other (small, medium, or large difference).

4. Results

In this section, we present the results from the survey and repository mining process grouped by each perspective. Table 4 shows a summary of the results obtained in this study.

Each one of the statements used in the survey with practitioners and researchers is present in the first column of Table 4. These statements were randomly created in order to avoid bias within the survey responses and in order to find differences between HCS and SS according to the investigated metrics. Thus, some statements are in favor of HCS projects and other are in favor of SS projects.

The second column of Table 4 presents the average Likert score and the Fisher's test results of the 10 statements organized by practitioners and researchers answers. Since we used a scale of 5 points, a likert value could be interpreted as: a value close to zero represents a disagreement with the statement; a value of exact three is neutral; and a value close to five represents an agreement with the statement. Moreover, a Fisher's Test result less than 0.05 shows a significant difference between practitioners and researchers answers, i.e. statement *No. of Accidental contributors in SS is bigger than HCS*.

The third column Table 4 shows the results of the Wilcoxon Test and the results of the Cliff's δ for the repository mining processes according to each statement. A Wilcoxon Test *p-value* greater or equal to 0.05 represents no statistical difference between HCS and SS projects, i.e. statement No. of code contributors in HCS is bigger than SS. And a Cliff's δ value considered large, shows a large distance between the means of HCS and SS projects, i.e. statement *No. of core developers in SS is bigger than HCS*.

Finally, we compared the results from the survey with the results from the mining process for each statement, last column of Table 4. If the results are equals we state our finding as the *Same*. If the results are different, we state our finding as the *Opposite*. For example, both practitioners and researchers disagree with the statement *Code change size of SS is bigger than HCS*. However, the Wilcoxon Test shows that SS projects have more change size than HCS projects. Thus, our finding is the *Opposite*.

For most of the results, we found a difference between HCS and SS projects, except for the number of code contributors. Moreover, the majority of our findings shown the opposite between the survey (perception) and the mining (evidence) analysis, except for bug-proneness and developer retention. Fig. 2 shows the box plots of mean values for each metric used in this analysis.

4.1. Product

4.1.1. Bug-proneness

Survey results. Many respondents disagree or are neutral considering that the number of bugs in SS is higher than in HCS. 14, 31, 33 respondents strongly disagree, disagree, and are neutral with this statement, respectively. The average Likert score for this statement is 2.60 (i.e., between "disagree" and "neutral"). The following are some comments that support or refute the statement:

✓"*An HCS fits more requirements than an SS, and thus more verification activities are conducted with more diverse test scenario. As parts of the HCS are shared between many usage scenarios, they are more thoroughly verified*".

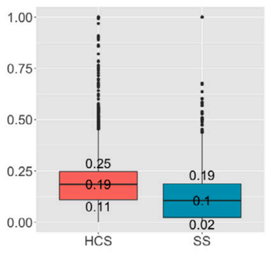
✗"*HCS have additional complexity introduced by variation points, without a systematic method to deal with variation points, more bugs are likely to appear... As a side effect of introducing feature model based product line engineering they measured a significant reduction of newly introduced bugs*".

From the previous comments, we note that participants realize the particularities promoted by variability in HCS. However, this perception was not enough to have a high evaluation rate (agree or strongly agree). In general, they disagree that the number of bugs in SS is higher than HCS. We also ran the Fisher's exact test to check the answers provided by practitioners and researchers and no difference between categories was identified (p-value = 0.5982).

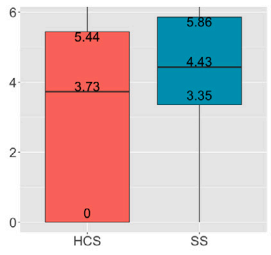
Repository mining results. We applied Wilcoxon test to identify differences between HCS and SS bug fix commits. The results show that the HCS median (0.19) was higher than the SS median (0.10) (Wilcoxon test, p-value 2.2e−16), although, the Cliff's δ results presented a "small" effect size (0.30).

Table 4
Survey and repository mining analyses results.

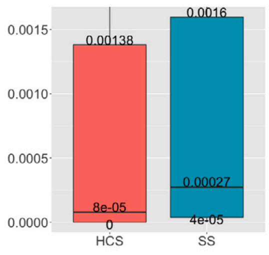
Statement	Survey				Repository mining			Finding (Survey vs. Mining)
	Likert scale		Fisher's test		Wilcoxon test			
	Pract.	Resear.	p-value	Result	p-value	Result	Cliff's δ	
No. of bugs in SS is higher than HCS	2.56	2.64	0.5982	Pract. = Resear.	2.2e−16	HCS > SS	0.30 [0.27,0.33] (small)	Same
Code complexity in HCS is higher than SS	3.82	4.04	0.3417	Pract. = Resear.	2.2e−16	SS > HCS	−0.21 [−0.24,−0.19] (small)	Opposite
Code change size of SS is bigger than HCS	2.71	2.80	0.6042	Pract. = Resear.	2.2e−16	SS > HCS	−0.19 [−0.21,−0.17] (small)	Opposite
No. of core dev. in SS is bigger than HCS	2.58	2.78	0.1774	Pract. = Resear.	2.2e−16	SS > HCS	−0.49 [−0.51,−0.48] (large)	Opposite
No. of code contr. in HCS is bigger than SS	3.33	3.28	0.8126	Pract. = Resear.	0.3619	SS = HCS	−0.01 [−0.03,−0.005] (negligible)	Opposite
No. of Acci. contr. in SS is bigger than HCS	2.76	2.90	1.12e−02	Pract. <> Resear.	2.2e−16	SS > HCS	−0.44 [−0.46,−0.42] (medium)	Opposite
Dev. retention is higher in HCS than SS	3.11	3.34	0.3225	Pract. = Resear.	2.2e−16	HCS > SS	0.17 [0.15,0.18] (small)	Same
There are more paid contr. in SS than HCS	2.91	2.92	0.7194	Pract. = Resear.	2.2e−16	SS > HCS	−0.16 [−0.18,−0.14] (small)	Opposite



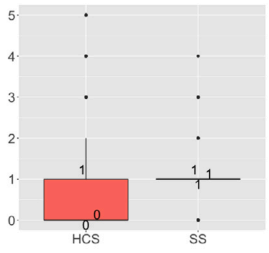
(a) Bug-Proneness



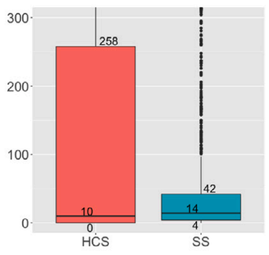
(b) Code Complexity



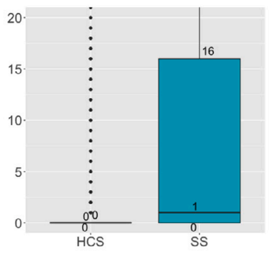
(c) Change Size



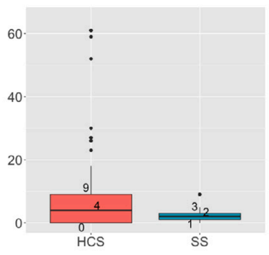
(d) No. of Core Developers



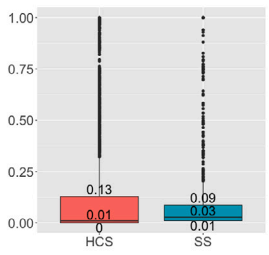
(e) No. of Code Contributors



(f) No. of Acciden. Contrib.



(g) Developer Retention



(h) Paid Contributors

Fig. 2. Box plots of repository mining results.

Finding. Although the survey answers do not show any significant difference between researchers and practitioners opinions related to number of bugs in HCS and SS, both tending to disagree with the statement (Number of bugs in SS is higher than HCS). Moreover, the repository mining statistical analysis presented a significant difference between HCS and SS, where the number of bugs in HCS indeed is higher than SS. This indicates the same results between the perception (survey) and evidence (mining).

4.1.2. Code complexity

Survey results. Many respondents expressed that code complexity in HCS is higher than SS. 43, 30 respondents agree and strongly agree with this statement and the average Likert score for this is 3.94 (i.e., mostly "agree"). The following are comments that support this statement:

✓*"When HCS are developed using pre-processing directives, i.e., #if and #else, the complexity to develop, understand, maintain the source-code is higher".*

✓*"In theory the complexity should not be higher in HCS compared to SS: an HCS would have its feature highly decoupled making it easily understandable and maintainable. But the reality is the those features are often coupled and the mechanism itself that allows to make the system configurable makes it more complex".*

This statement seems to be like universally supported, at least among the participants whom we surveyed. It also had the highest value in our study. In addition, the Fisher's exact test did not identify any difference between practitioners and researchers (p-value = 0.3417).

Repository mining results. We applied Wilcoxon test to identify differences between HCS and SS code complexity. The results show that the HCS median (3.73) was lower than the SS median (4.43) (Wilcoxon test, p-value 2.2e−16). The statistical significance was confirmed by the Cliff's δ results (−0.21, considered a "small" effect size).

Finding. The survey answers did not show any significant difference between researchers and practitioners opinions related to code complexity in HCS and SS, both tending to agree with the statement (Code complexity in HCS is higher than SS). However, the repository mining statistical analysis shown a significant difference between HCS and SS code complexity, where the complexity in SS is higher than HCS. Thus, our findings were the opposite when comparing the survey and mining repository processes.

4.1.3. Change size

Survey results. Many respondents disagree or are neutral considering that average code change size of SS is bigger than HCS. 32, 34, 21 respondents disagree, are neutral, and agree with this statement, respectively. The average Likert score for this statement is 2.76 (i.e., between "disagree" and "neutral"). We received the following comments that refute the statement or are neutral:

✗*"In my experience, the more configurable a system, the more code it needs to support that".*

✗*"Proper abstraction – that can seem like a boilerplate – could increase the change size and also gave the opportunity to achieve HCS, therefore the change size is somewhat the same".*

↔*"I believe that this highly depends on the coding guidelines for a specific project"*.

We also ran the Fisher's exact test to check the answers provided by practitioners and researchers and no difference between categories was identified (p-value = 0.6042).

Repository mining results. We applied Wilcoxon test to identify differences between HCS and SS in terms of change size. The results show that the SS median (0.00027) was higher than the HCS median (0.00008) (Wilcoxon test, p-value 2.2e−16), although, the Cliff's δ results shown a "small" effect size (−0.19).

Finding. The survey answers did not show any significant difference between researchers and practitioners opinions related to change size in HCS and SS. Both of them tend to disagree with the defined statement for this metric (Code change size of SS is bigger than HCS). However, the repository mining statistical analysis shown a significant difference between HCS and SS, where the change size in SS is higher than HCS. Thus, we found opposite results when comparing the survey answers and the mining repository process.

4.2. Process

4.2.1. Number of core developers

Survey results. Many respondents disagree or are neutral considering that the number of core developers in SS is bigger than HCS. 10, 24, 47 respondents strongly disagree, disagree, and are neutral with this statement, respectively. The average Likert score for this statement is 2.68 (i.e., between "disagree" and "neutral"). The following are some comments that support or refute the statement:

✓*"As the code is not shared, the number of core developers needed to build the corresponding SS to the various usage scenarios of the HCS is bigger. But the proficiency of the developers needed to build an HCS is usually higher than the ones for an SS"*.

✗*"If an app is applicable in more contexts, it will attract a wider audience. Some of that audience will be programmers, thus more programmers on average will get involved compared to an SS project. More programmers means, all other things being equal – such as the rate of conversion from starting contributor to core contributor – there will also be more core contributors over time compared to an SS project"*.

We also ran the Fisher's exact test to check the answers provided by practitioners and researchers and no difference between categories was identified (p-value = 0.1774).

Repository mining results. We applied Wilcoxon test to identify differences between HCS and SS in terms of core developers. The results show that the SS median (1) was higher than the HCS median (0) (Wilcoxon test, p-value 2.2e−16), although. The statistical significance was confirmed by the Cliff's δ results (−0.49, considered a "large" effect size).

Finding. The survey answers do not show any significant difference between researchers and practitioners opinions related to the Number of Core Developers in HCS and SS. However, based on the defined statement for this metric (Number of core developers in SS is bigger than HCS), practitioners and researchers tend to disagree. The repository mining statistical analysis presented a significant difference between HCS and SS, where the number of core developers within SS is higher than HCS. Thus, we found the opposite result when comparing the survey answers and the mining repository process.

4.2.2. Number of code contributors

Survey results. In general, the respondents are neutral or agree that the number of code contributors in HCS is bigger than SS. 43, 31, 8 respondents are neutral, agree, and strongly agree with this statement, respectively. The average Likert score for this statement is 3.31 (i.e., between "neutral" and "agree"). We received the following comments that refute or support the statement:

✗*"I can't say for sure, but I think it is the same"*.

✓*"Number of contributors is related to number of variations"*.

We also ran the Fisher's exact test to check the answers provided by practitioners and researchers and no difference between categories was identified (p-value = 0.8126).

Repository mining results. We applied Wilcoxon test to identify differences between HCS and SS in terms of number of code contributors. The results show that the SS median (14) was higher than the HCS median (10). However, since the Wilcoxon test *p-value* was 0.3619 (greater than 0.05) we did not found a significant difference between HCS and SS code contributors. The statistical significance was confirmed by the Cliff's δ results (−0.01, considered a "negligible" effect size).

Finding. The survey answers did not show any significant difference between researchers and practitioners opinions related to the number of code contributors in HCS and SS. Both of them tend to agree with the defined statement for this metric (Number of code contributors in HCS is bigger than SS). However, the repository mining statistical analysis did not shown a significant difference between the number of code contributors within HCS and SS (HCS code contributors is statistically equals to SS code contributors). Thus, we found an opposite result when comparing the survey answers and the mining repository process.

4.2.3. Number of accidental contributors

Survey results. Many respondents disagree or are neutral considering that the number of accidental contributors in SS is bigger than HCS. 6, 16, 63 respondents strongly disagree, disagree, and are neutral with this statement, respectively. The average Likert score for this statement is 2.83 (i.e., between "disagree" and "neutral"). The following are some comments that support or refute the statement:

✓*"Since SS is emerging product, number of accidental contributors in SS is bigger than HCS"*.

✗*"More use cases → bigger audience → more contributors, including the 1-and-done crowd. However, there is also another factor (which, again, has nothing to do with HCS vs. SS): the skill of the project maintainer in making contributors feel valued, validated, and appreciated. A welcoming and friendly atmosphere that makes the contributor feel like they are making a difference, as well as an active community with frequent feedback and discourse, all contribute to the conversion of contributors from accidental to regulars/core"*.

We also ran the Fisher's exact test to check the answers provided by practitioners and researchers and, in this statement, we found a significant difference between both categories (p-value = 1.12e−02).

Repository mining results. We applied Wilcoxon test to identify differences between HCS and SS in terms of number of accidental contributors. The results show that the SS median (1) was higher than the HCS median (0) (Wilcoxon test, p-value 2.2e−16). The statistical significance was confirmed by the Cliff's δ results (−0.44, considered a "medium" effect size).

Finding. After Analyzing the survey answers, we identified a significant difference between the answers from researchers and practitioners related to the number of accidental contributors in HCS and SS. Although researchers and practitioners answers are different, both of them are below the average the likert scale, meaning that researchers and practitioners disagree with the defined statement for this metric (Number of Accidental contributors in SS is bigger than HCS). The repository mining statistical analysis presented a significant difference between HCS and SS, where the number of accidental contributors within SS is higher than HCS. Thus, we found an opposite result when comparing the survey answers and the mining repository process.

4.3. People

4.3.1. Developer retention

Survey results. In general, the respondents are neutral or agree that the developer retention is higher in HCS than SS. 54, 28, 3 respondents are neutral, agree, and strongly agree with this statement, respectively. The average Likert score for this statement is 3.23 (i.e., between "neutral" and "agree"). We received the following comments that support or refute the statement:

✓"*An HCS is much more interesting to build than an SS, so it might help in building long term teams*".

↔"*This I can't really say. I think there is a balancing act in play between an HCS being more intimidating, but also more applicable so more experienced developers get on board. I think I largely see more in an HCS, but I feel like it is difficult to make a call here simply from my observations*".

We also ran the Fisher's exact test to check the answers provided by practitioners and researchers and we did not find a significant difference between both categories (p-value = 0.3225).

Repository mining results. We applied Wilcoxon test to identify differences between HCS and SS in terms of developer retention. The results show that the HCS median (4) was higher than the SS median (2) (Wilcoxon test, p-value 2.2e−16). The statistical significance was confirmed by the Cliff's δ results (0.17, considered a "small" effect size).

Finding. The survey results did not present any significant difference between researchers and practitioners opinions related to the number of developer retention in HCS and SS. Both of them tend to agree with the defined statement for this metric (Developer retention is higher in HCS than SS). Moreover, the repository mining statistical analysis shown a significant difference between the number of developer retention within HCS and SS, where the number of developer retention in HCS is indeed higher than SS. Thus, we found the same result when comparing the survey answers and the mining repository process.

4.3.2. Paid contributors

Survey results. Many respondents disagree or are neutral considering that there are more paid contributors in SS than HCS. 6, 13, 60 respondents strongly disagree, disagree, and are neutral with this statement, respectively. The average Likert score for this statement is 2.92 (i.e., between "disagree" and "neutral"). The following are some comments that support or refute the statement:

✓"*But the contributors in HCS are usually paid higher wages*".

✗"*General apps that have a more complete feature set will naturally receive more funding from people who need to get a job done. If Blender could only do animation or only modeling, etc., it wouldn't have nearly as much funding as it does with those features plus a ton more, etc*".

We also ran the Fisher's exact test to check the answers provided by practitioners and researchers and we found no difference between both categories (p-value = 0.7194).

Repository mining results. We applied Wilcoxon test to identify differences between HCS and SS in terms of number of paid developers. The results show that the SS median (0.03) was higher than the HCS median (0.01) (Wilcoxon test, p-value 2.2e−16). The Cliff's δ results presented a "small" effect size (−0.16).

Finding. Analyzing the survey answers, we identified that there is no significant difference between the answers from researchers and practitioners related to the number of paid contributors in HCS and SS. Both of them disagree with the defined statement for this metric (There are more paid contributors in SS than HCS). The repository mining statistical analysis presented a significant difference between HCS and SS paid contributors, where the number for SS is higher than HCS. Thus, once again, we found an opposite result when comparing the survey answers and the mining repository process.

5. Discussion

5.1. Product perspective

Our results show that the survey respondents do not agree or are neutral that the number of bugs in SS is higher than HCS. The literature consider that in HCS, features interact in non-trivial ways in order to influence the functionality of each others [58]. When these interactions are unintended, many times they induce bugs that manifest themselves in some configurations but not in others, or that manifest differently in different configurations. One survey participant highlighted it: *"In HCS, there are some additional mechanisms that leads to additional bugs, such as feature interaction bugs"*. Moreover, data from repository mining showed that HCS indeed is more bug-prone than SS. This finding confirms what many researchers argue.

Regarding complexity, survey respondents agree that HCS projects have more complexity than SS ones. However, the mining results revealed the opposite. As highlighted by one of the survey participants, an HCS has more verification activities than an SS, as parts of the HCS are shared among many usage scenarios [59]. Thus, the correlation between more verification activities and less complexity is worth of a further investigation.

Our results showed that, on average, code change size of SS is bigger than HCS. This finding is someway expected since HCS are designed carefully to accommodate common and variable parts. However, the survey respondents disagree or are neutral regarding this aspect. Thus, new studies should be conducted to understand the type of changes in HCS (common part or variability changes), the evolution patterns, and the co-evolution process between source code and other artifacts, such as configuration models.

5.2. Process perspective

Our repository mining result shown that SS have more core developers than HCS. However, the survey respondents agree that HCS have more core developers than SS. Despite HCS systems have less core developers, and are considered less complex than SS, they have more bugs. Further study can help to unearth the underlying reason for such observation.

Regarding code contributors, the repository mining process showed that SS and HCS do not have difference. On the other hand, the survey respondents are neutral or agree that the number of code contributors in HCS is bigger than SS. New studies are required to understand productivity and quality issues besides the role of code contributors.

Based on the repository mining result, we see that the number of accidental contributors in SS is bigger than HCS. It supports other finding from this study which states that SS have more core developers than HCS. Since in SS we have more core developers doing most of the development, it is natural to have developers doing sporadic contributions. The survey respondents also disagree or are neutral that the number of accidental contributors in SS is bigger than HCS.

5.3. People perspective

Regarding developer retention, the data from repository mining showed that it is higher in HCS than SS. Although SS have more core developers, it is hard to peripheral developers stay motivated. Moreover, survey respondents are neutral or agree that developer retention is higher in HCS than SS. According to one participant: *"An HCS is much more interesting to build than an SS, so it might help in building long term teams"*. A dedicated study aimed towards understanding this observation is required to make any conclusive remarks.

Based on the repository mining the number of paid contributors in SS is bigger than HCS. Furthermore, SS also have more core contributors than HCS. This correlation should be investigated in details in order to identify if a high number of paid contributors implies in a high number core developers. On the other hand, survey respondents agree that HCS have more paid contributors than SS.

6. Threats to validity

We have taken care to ensure that our results are unbiased, and have tried to eliminate the effects of random noise, but it is possible that our mitigation strategies may not have been effective. In this section, we discuss the threats to validity for our study.

6.1. External validity

The primary threat to the external validity of this study involves the generalizability of our subjects. We reduce this threat by using 124 different projects from different application domains. However, all of the projects are open source, so we cannot make any claims about how our results might generalize to proprietary projects.

6.2. Construct validity

The set of metrics used to compare SS and HCS were selected based on the formative survey. However, other metrics could have been used for this purpose. Thus, our evaluation is not exhaustive, but we believe that the metrics we used provide a fair assessment of difference between HCS and SS.

For our analysis, we categorized developers into core and non-core groups and for this categorization we had to set a threshold of number of commits in the code base for each developer. It might be the case that some of the developers that were categorized as non-core developers according to our criteria were actually core developers who focus on large contributions rather than frequent contributions, or simply focus on architecture and high-level design (high value contributions).

Further, our keyword-based search for bug fixing commit identification relies on the developers appropriately using keywords in the commit message, that may not always happen and may lead to missed bug fixes.

6.3. Internal validity

The primary threats to the internal validity of this study are possible faults in the implementation of our approach and in the tools that we used to perform the evaluation. We control this threat by using tools validated by other researchers for collecting various metrics. We also extensively test our implementations and verify their results against a smaller program for which we can manually determine the correct results.

It is always possible that the participants misunderstand the survey questions. To mitigate this threat, we conducted a pilot study with three researchers with different experience levels both in OSS and industry. We also conducted a pilot study with survey design experts. We updated the survey based on the findings of these pilot studies. Even after our best effort, it is possible that some of our survey respondents had a poor understanding of the statements for rating. To reduce the impact of this issue, we included an "I don't know" option in the survey and ignored responses marked as such.

7. Conclusions

In this paper, we investigated similarities and differences between HCS and SS using well known metrics from the literature according to different perspectives: *product*, *process*, and *people*. We collected data from two surveys and from a mining study (within 15,769 releases of 124 Github projects). Our results shown that for the majority of the metrics, the perception of practitioners and researchers about HCS and SS is different from our mining findings.

Bug-proneness is perceived by practitioners and researchers to be higher in HCS compared to SS. To corroborate with this finding, or mining study revealed the same. Thus, the more `#ifdef` a code have, the more bugs may be present.

Although complexity, change size, number of core developers, number of code contributors, and number of accidental contributors are perceived by practitioners and researchers to be higher in HCS compared to SS, our mining study found the opposite. We believe that the strengths of HCS approaches, such as testing, may be adapted and applied in the context of SS to improve their general quality related to complexity. Related to code change size, we strengthen the body of evidence that HCS are carefully designed to accommodate variability instead of performing code changes. Despite HCS systems have less core developers, and are considered less complex than SS, they have more bugs. This finding is worth of investigating. The role of contributors in HCS seems to be the same from SS. Although, HCS presented more bugs within the source code.

Developer retention is perceived by practitioners and researchers to be higher in HCS compared to SS. Our mining study also found the same. Although it seems that HCS may be more interesting to build compared to SS, a dedicated study is required to make any conclusive remarks.

Finally, the number of paid contributors are perceived by practitioners and researchers to be higher in HCS compared to SS. However, our mining study found the opposite. Since SS also have more core developers, we recommend a further investigation of the correlation between paid contributors and core developers and then, compare the results with HCS.

The current work here lays out highlighted perspectives, quantitative evidence to clarify existing beliefs about HCS, and highlight the difference between belief and evidence. We hope these results prompt a discussion and further research in the direction as to how utilize the similarity of both types of systems to transfer insights, methods, and tools. Also, identify the differences to design new techniques addressing the difference.

CRediT authorship contribution statement

Raphael Pereira de Oliveira: Conceptualization, Methodology, Validation, Statistics, Results, Analysis, Discussion, Created the scripts for complexity, change size, core developers, code contributors, accidental contributors, and paid contributors. **Paulo Anselmo da Mota Silveira Neto:** Create the bug-proneness script, Running the bug-proneness. **Qi Hong Chen:** Running the analysis of complexity, change size, core developers, code contributors, accidental contributors, and paid contributors. **Eduardo Santana de Almeida:** Conceptualization, Introduction, Related work, Conclusion. **Iftekhar Ahmed:** Conceptualization, Introduction, Running scripts, Conclusion.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work is partially supported by INES (www.ines.org.br), CNPq grant 465614/2014-0, FACEPE grants APQ-0399-1.03/17 and PRONEX APQ/0388-1.03/14. In addition, we would like to thank all the participants in the survey.

References

[1] S. Apel, D. Batory, C. Kstner, G. Saake, Feature-Oriented Software Product Lines: Concepts and Implementation, Springer Publishing Company, Incorporated, 2013.
[2] K. Kang, S. Cohen, J. Hess, W. Novak, A. Peterson, Feature-oriented domain analysis (FODA) feasibility study, Tech. rep., Software Engineering Institute, 1990.
[3] I. Abal, J. Melo, S. Stanciulescu, C. Brabrand, M. Ribeiro, A. Wasowski, Variability bugs in highly configurable systems: A qualitative analysis, ACM Trans. Softw. Eng. Methodol. 26 (3) (2018) 10:1–10:34, http://dx.doi.org/10.1145/3149119.

[4] J. van Gurp, J. Bosch, M. Svahnberg, On the notion of variability in software product lines, in: Proceedings Working IEEE/IFIP Conference on Software Architecture, 2001, pp. 45–54.
[5] S. Schulze, J. Liebig, J. Siegmund, S. Apel, Does the discipline of preprocessor annotations matter?: a controlled experiment, in: ACM SIGPLAN Notices, Vol. 49, ACM, 2013, pp. 65–74.
[6] B. Zhang, M. Becker, Recovar: A solution framework towards reverse engineering variability, in: 2013 4th International Workshop on Product LinE Approaches in Software Engineering (PLEASE), IEEE, 2013, pp. 45–48.
[7] M. Goedicke, K. Pohl, U. Zdun, Domain-specific runtime variability in product line architectures, in: International Conference on Object-Oriented Information Systems, Springer, 2002, pp. 384–396.
[8] P. Devanbu, T. Zimmermann, C. Bird, Belief & evidence in empirical software engineering, in: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), IEEE, 2016, pp. 108–119.
[9] T. Menzies, W. Nichols, F. Shull, L. Layman, Are delayed issues harder to resolve? Revisiting cost-to-fix of defects throughout the lifecycle, Empir. Softw. Eng. 22 (4) (2017) 1903–1935.
[10] S. Majumder, J. Chakraborty, A. Agrawal, T. Menzies, Communication and code dependency effects on software code quality (an empirical analysis of herbsleb hypothesis), J. Syst. Softw. (2022) (Preprint).
[11] F. Medeiros, M. Ribeiro, R. Gheyi, S. Apel, C. Kästner, B. Ferreira, L. Carvalho, B. Fonseca, Discipline matters: Refactoring of preprocessor directives in the #ifdef Hell, IEEE Trans. Softw. Eng. 44 (5) (2018) 453–469.
[12] H. Borges, M.T. Valente, What's in a GitHub star? Understanding repository starring practices in a social coding platform, J. Syst. Softw. 146 (2018) 112–129.
[13] C. Hunsen, B. Zhang, J. Siegmund, C. Kästner, O. Lebenich, M. Becker, S. Apel, Preprocessor-based variability in open-source and industrial software systems: An empirical study, Empir. Softw. Eng. 21 (2) (2016) 449–482.
[14] R. Lotufo, S. She, T. Berger, K. Czarnecki, A. Wasowski, Evolution of the linux kernel variability model, in: Software Product Lines: Going beyond - 14th International Conference, SPLC 2010, September 13-17, 2010. Proceedings, Jeju Island, South Korea, 2010, pp. 136–150, http://dx.doi.org/10.1007/978-3-642-15579-6_10.
[15] A. Israeli, D.G. Feitelson, The linux kernel as a case study in software evolution, J. Syst. Softw. 83 (3) (2010) 485–501.
[16] M.M. Lehman, J.F. Ramil, P. Wernick, D.E. Perry, W.M. Turski, Metrics and laws of software evolution - the nineties view, in: 4th IEEE International Software Metrics Symposium (METRICS 1997), November 5-7, 1997, Albuquerque, NM, USA, 1997, p. 20.
[17] L.T. Passos, K. Czarnecki, A. Wasowski, Towards a catalog of variability evolution patterns: the Linux kernel case, in: 4th International Workshop on Feature-Oriented Software Development, FOSD '12, September 24 - 25, 2012, Dresden, Germany, 2012, pp. 62–69.
[18] L.T. Passos, J. Guo, L. Teixeira, K. Czarnecki, A. Wasowski, P. Borba, Coevolution of variability models and related artifacts: a case study from the Linux kernel, in: 17th International Software Product Line Conference, SPLC 2013, August 26 - 30, 2013, Tokyo, Japan, 2013, pp. 91–100.
[19] L.T. Passos, L. Teixeira, N. Dintzner, S. Apel, A. Wasowski, K. Czarnecki, P. Borba, J. Guo, Coevolution of variability models and related software artifacts - A fresh look at evolution patterns in the Linux kernel, Empir. Softw. Eng. 21 (4) (2016) 1744–1793.
[20] L.T. Passos, J. Padilla, T. Berger, S. Apel, K. Czarnecki, M.T. Valente, Feature scattering in the large: a longitudinal study of Linux kernel device drivers, in: Proceedings of the 14th International Conference on Modularity, MODULARITY 2015, Fort Collins, CO, USA, March 16 - 19, 2015, 2015, pp. 81–92.
[21] L. Passos, R. Queiroz, M. Mukelabai, T. Berger, S. Apel, K. Czarnecki, J.A. Padilla, A study of feature scattering in the Linux Kernel, IEEE Trans. Softw. Eng. Early Access (2018) 1.
[22] E. Engström, P. Runeson, M. Skoglund, A systematic review on regression test selection techniques, Inf. Softw. Technol. 52 (1) (2010) 14–30.
[23] B.A. Kitchenham, What's up with software metrics? - A preliminary mapping study, J. Syst. Softw. 83 (1) (2010) 37–51.
[24] K. Petersen, Measuring and predicting software productivity: A systematic map and review, Inf. Softw. Technol. 53 (4) (2011) 317–343.
[25] T. Sharma, D. Spinellis, A survey on software smells, J. Syst. Softw. 138 (2018) 158–173.
[26] F. Rahman, P.T. Devanbu, How, and why, process metrics are better, in: 35th International Conference on Software Engineering, ICSE '13, May 18-26, 2013, San Francisco, CA, USA, 2013, pp. 432–441.
[27] B.A. Kitchenham, S.L. Pfleeger, Personal opinion surveys, in: Guide To Advanced Empirical Software Engineering, Springer, 2008, pp. 63–92.
[28] S. Baltes, S. Diehl, Worse than spam: Issues in sampling software developers, 2016, pp. 1–6, http://dx.doi.org/10.1145/2961111.2962628.
[29] G. Gousios, The GHTorent dataset and tool suite, in: Proceedings of the 10th Working Conference on Mining Software Repositories, in: MSR '13, IEEE Press, 2013, pp. 233–236.
[30] C. Casalnuovo, Y. Suchak, B. Ray, C. Rubio-González, GitcProc: a tool for processing and classifying GitHub commits, in: Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, Santa Barbara, CA, USA, July 10 - 14, 2017, 2017, pp. 396–399.
[31] S. El-Sharkawy, A. Krafczyk, K. Schmid, MetricHaven: More than 23,000 metrics for measuring quality attributes of software product lines, in: Proceedings of the 23rd International Systems and Software Product Line Conference - Volume B, in: SPLC '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 25–28.
[32] A. Gopal, T. Mukhopadhyay, M.S. Krishnan, The role of software processes and communication in offshore software development, Commun. ACM 45 (4) (2002) 193–200.
[33] N. Nagappan, B. Murphy, V. Basili, The influence of organizational structure on software quality: An empirical case study, in: Proceedings of the 30th International Conference on Software Engineering, in: ICSE '08, Association for Computing Machinery, New York, NY, USA, 2008, pp. 521–530.
[34] A. Mockus, R.T. Fielding, J.D. Herbsleb, Two case studies of open source software development: Apache and Mozilla, ACM Trans. Softw. Eng. Methodol. 11 (3) (2002) 309–346.
[35] D.A.A. Tamburri, F. Palomba, R. Kazman, Exploring community smells in open-source: An automated approach, IEEE Trans. Softw. Eng. (2019) 1, http://dx.doi.org/10.1109/TSE.2019.2901490.
[36] A. Lee, J.C. Carver, A. Bosu, Understanding the impressions, motivations, and barriers of one time code contributors to FLOSS projects: A survey, in: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), 2017, pp. 187–197, http://dx.doi.org/10.1109/ICSE.2017.25.
[37] B. Lin, G. Robles, A. Serebrenik, Developer turnover in global, industrial open source projects: Insights from applying survival analysis, in: Proceedings of the 12th International Conference on Global Software Engineering, in: ICGSE '17, IEEE Press, 2017, pp. 66–75.
[38] D. Riehle, P. Riemer, C. Kolassa, M. Schmidt, Paid vs. Volunteer work in open source, in: 47th Hawaii International Conference on System Sciences, HICSS 2014, January 6-9, 2014, Waikoloa, HI, USA, 2014, pp. 3286–3295.
[39] M.D. Ernst, G.J. Badros, D. Notkin, An empirical analysis of c preprocessor use, IEEE Trans. Softw. Eng. 28 (12) (2002) 1146–1170, http://dx.doi.org/10.1109/TSE.2002.1158288.
[40] J. Liebig, S. Apel, C. Lengauer, C. Kästner, M. Schulze, An analysis of the variability in forty preprocessor-based software product lines, in: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, in: ICSE '10, Association for Computing Machinery, 2010, pp. 105–114.
[41] J. Liebig, C. Kästner, S. Apel, Analyzing the discipline of preprocessor annotations in 30 million lines of C code, in: Proceedings of the Tenth International Conference on Aspect-Oriented Software Development, in: AOSD '11, Association for Computing Machinery, 2011, pp. 191–202.
[42] F. Medeiros, C. Kästner, M. Ribeiro, S. Nadi, R. Gheyi, The love/hate relationship with the C preprocessor: An interview study, in: J.T. Boyland (Ed.), 29th European Conference on Object-Oriented Programming, ECOOP 2015, Vol. 37, in: LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, pp. 495–518.
[43] F. Medeiros, M. Ribeiro, R. Gheyi, Investigating preprocessor-based syntax errors, SIGPLAN Not. 49 (3) (2013) 75–84.
[44] M. Ribeiro, F. Queiroz, P. Borba, T. Tolêdo, C. Brabrand, S. Soares, On the impact of feature dependencies when maintaining preprocessor-based software product lines, SIGPLAN Not. 47 (3) (2011) 23–32.
[45] J. Garcia, Y. Feng, J. Shen, S. Almanee, Y. Xia, Chen, Q. Alfred, A comprehensive study of autonomous vehicle bugs, in: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, in: ICSE '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 385–396, http://dx.doi.org/10.1145/3377811.3380397.
[46] A. Di Franco, H. Guo, C. Rubio-González, A comprehensive study of real-world numerical bug characteristics, in: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), 2017, pp. 509–519, http://dx.doi.org/10.1109/ASE.2017.8115662.
[47] M.J. Islam, G. Nguyen, R. Pan, H. Rajan, A comprehensive study on deep learning bug characteristics, in: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, in: ESEC/FSE 2019, Association for Computing Machinery, New York, NY, USA, 2019, pp. 510–520, http://dx.doi.org/10.1145/3338906.3338955.
[48] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, V. Filkov, Quality and productivity outcomes relating to continuous integration in GitHub, in: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, in: ESEC/FSE 2015, Association for Computing Machinery, New York, NY, USA, 2015, pp. 805–816, http://dx.doi.org/10.1145/2786805.2786850.
[49] Y. Zhang, Y. Chen, S.-C. Cheung, Y. Xiong, L. Zhang, An empirical study on TensorFlow program bugs, in: Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, in: ISSTA 2018, Association for Computing Machinery, New York, NY, USA, 2018, pp. 129–140, http://dx.doi.org/10.1145/3213846.3213866.
[50] M. Bigonha, K. Ferreira, P. Souza, B. Sousa, M. Januário, D. Lima, The usefulness of software metric thresholds for detection of bad smells and fault prediction, Inf. Softw. Technol. 115 (2019) http://dx.doi.org/10.1016/j.infsof.2019.08.005.
[51] I. Ahmed, U.A. Mannan, R. Gopinath, C. Jensen, An empirical study of design degradation: How software projects get worse over time, in: Empirical Software Engineering and Measurement (ESEM), 2015 ACM/IEEE International Symposium on, IEEE, 2015, pp. 1–10.

[52] M. Foucault, M. Palyart, X. Blanc, G.C. Murphy, J.-R. Falleri, Impact of developer turnover on quality in open-source software, in: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, in: ESEC/FSE 2015, Association for Computing Machinery, New York, NY, USA, 2015, pp. 829–841.

[53] M.R. Hess, J.D. Kromrey, Robust confidence intervals for effect sizes: A comparative study of cohen's d and cliff's delta under non-normality and heterogeneous variances. paper presented at the annual meeting of the American educational research association, 2004.

[54] R.A. Fisher, On the interpretation of x 2 from contingency tables, and the calculation of p, J. R. Stat. Soc. 85 (1) (1922) 87–94.

[55] W. Conover, Practical Nonparametric Statistics, third ed., in: Wiley series in probability and statistics, Wiley, New York, NY [u.a.], 1999, URL http://gso.gbv.de/DB=2.1/CMD?ACT=SRCHA&SRT=YOP&IKT=1016&TRM=ppn+24551600X&sourceid=fbw_bibsonomy.

[56] R. Woolson, Wilcoxon signed-rank test, in: Wiley Encyclopedia of Clinical Trials, Wiley Online Library, 2007, pp. 1–3.

[57] B. Kitchenham, L. Madeyski, D. Budgen, J. Keung, P. Brereton, S. Charters, S. Gibbs, A. Pohthong, Robust statistical methods for empirical software engineering, Empir. Softw. Eng. 21 (2017) http://dx.doi.org/10.1007/s10664-016-9437-5.

[58] L.R. Soares, J. Meinicke, S. Nadi, C. Kästner, E.S. de Almeida, Exploring feature interactions without specifications: a controlled experiment, in: E.V. Wyk, T. Rompf (Eds.), Proceedings of the 17th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences, GPCE 2018, November 5-6, 2018, ACM, Boston, MA, USA, 2018, pp. 40–52.

[59] E. Engström, P. Runeson, Software product line testing - A systematic mapping study, Inf. Softw. Technol. 53 (1) (2011) 2–13.