A Second Replicated Quantitative Analysis of Fault Distributions in Complex Software Systems

Galinac Grbac, Tihana; Runeson, Per; Huljenic, Darko

Published in:
IEEE Transactions on Software Engineering

DOI:
10.1109/TSE.2012.46

2013

A Second Replicated Quantitative Analysis of Fault Distributions in Complex Software Systems

Tihana Galinac Grbac, Per Runeson, and Darko Huljenić,

August 15, 2013

Abstract

Background. Software engineering is in search for general principles that apply across contexts, for example, to help guiding software quality assurance. Fenton and Ohlsson presented such observations on fault distributions, which have been replicated once. *Objectives.* We aimed to replicate their study again to assess the robustness of the findings in a new environment, five years later. *Method.* We conducted a literal replication, collecting defect data from five consecutive releases of a large software system in the telecommunications domain, and conducted the same analysis as in the original study. *Results.* The replication confirms results on un-evenly distributed faults over modules, and that fault proneness distributions persist over test phases. Size measures are not useful as predictors of fault proneness, while fault densities are of the same order of magnitude across releases and contexts. *Conclusions.* This replication confirms that the un-even distribution of defects motivates un-even distribution of quality assurance efforts, although predictors for such distribution of efforts are not sufficiently precise.

1 Introduction

Software engineering is a relatively young discipline, and most of the decisions within the software development life-cycle are still based on common beliefs that have no or limited empirical foundation. Therefore, the empirical approach is crucial for the transformation of software engineering into a mature discipline. This can be achieved only by performing and publishing empirical studies of high quality and replicating them in subsequent studies [36]. Replications provide software engineering, as in all kinds of science, a mechanism to collect evidence for the sustained characteristics of the studied phenomenon.

Numerous data have been published related to the software fault distributions, but the first systematically reported study was performed by Fenton and Ohlsson [10], although it in part is rooted back in Basili and Perricone's study from the late 1970's [3]. Fenton and Ohlsson's study consolidates hypotheses that are related to the Pareto principle of fault distributions within the system, using basic measures to identify faulty system modules and benchmarking of fault data. The study is replicated by Andersson and Runeson [2]. The results obtained are not fully compatible, although they corroborate most of the original findings. Hence, there is a motivation for further replications.

Replications may be *literal* or *theoretical* [34, 36]. Literal replications attempt to follow the original procedures as close as possible, while theoretical replications vary one or more conditions in the settings, for a certain purpose. It is debated whether literal or theoretical replications is preferable [24, 17]. Juristo and Vegas even question whether any replication in software engineering may be classified as literal, since so many factors may vary in the complex setting of an empirical study in software engineering [15].

This paper is a second replication of the original study [10], which we have attempted to be as literal as possible. We follow as much as possible the original and the previous replication studies, to make comparisons possible, and thus contributing to synthesized knowledge. We measure the same variables and analyze the same hypotheses. However, there are certain differences in the measurements, that we could not repeat in our study, resulting in one hypothesis fewer than the original study. Additionally, the difference in time of over a decade, including the introduction of incremental work practices are factors that may have an impact of the outcome of the replication, even though it is conducted within the same multinational corporation.

The first replication study [2] was published seven years after the original study, and studied a different type of system; consumer products versus switching systems in the original study, although both in the telecommunications domain. In this, the second replication study, we performed quantitative analyses of data from five consecutive releases of an evolutionary developed large scale software system for telecommunication applications. We followed recommendations by Miller [24] to change some elements compared to the previous studies to check the stability of the results. As suggested by Miller, we also stick to the same report structure as much as possible, and we even kept the same headings to allow easy reference to the original and first replicated studies. In the rest of the paper we will refer to these papers as the *original study* [10], the *previous replication study* [2], and *this replication study* when referring to the replication study described in this paper.

The paper is organized as follows. In Section 2 we present an overview of the original study and the previous replication, briefly describe their differences and results obtained, as well as some subsequent work regarding the stated hypotheses. In Section 3 we describe the details of the context in which the data for this replication was collected. In Section 4 we repeat the hypotheses from previous studies that we analyze in this replication. The results obtained for each hypothesis are presented in separate subsections of Section 5. We also include a subsection with an overview of the studies addressing some of our hypothesis in the case of open source software. Finally, in Section 6 we discuss the results with respect to the original study and the previous replication, and conclude the paper in Section 7.

2 Background

In the original study [10], the hypothesis were analyzed in four groups. i) Hypotheses related to the Pareto principle of fault distribution, ii) Hypotheses related to persistence of faults, iii) Hypotheses about effects of module size and complexity on fault proneness, and iv) Hypotheses about the quality in terms of fault densities.

The first group of hypotheses has been motivated by the application of the Pareto principle (also known as the 20–80 rule), a well–known quality control principle [14] on software faults distribution. In software engineering, the principle is used to identify percentages of software modules, labeled as fault prone for consequent quality assurance activities [28]. Furthermore, since the Pareto principle is used to predict these 20% of most fault prone modules, it becomes a necessity to investigate the proportion of the total system size these modules take, so the quality assurance decisions would be meaningful. That is, if these software modules constituted the majority of software system size, then the Pareto principle applied on fault distribution would not be helpful. In such case, all the theory related to the classification models as in Lessman et al. [21] and Runeson et al. [33] would be useless. The results obtained by testing this group of hypotheses are consistent between the original and previous replication study. The hypothesis that the small number of modules contain most of the defects was also confirmed in many other studies [4], [6], [16], [27], [28], [9], and no support has been obtained for the hypothesis that the modules where the majority of faults is discovered constitute most of the system size. These results have been repeated for the pre-release and post-release faults separately.

Another widely used approach to identify quality assurance activities in later verification phases is based on the relationship between number of faults from consecutive verification phases. These relationships are studied in the second group of hypotheses. The usually applied principle is that higher incidence of faults in earlier verification phases implies higher incidence of faults in the subsequent verification phases [4]. The original and the previous replication studies give contrary results on this hypothesis. Limited or no support was found in the original study, while in the previous replication these hypotheses were supported, or even strongly supported. The authors of the previous replication argued that some modules are more central from the architectural point of view, and consequently more fault prone during their whole life–cycle. In the original study it is stressed that the causal relationship may be affected by other factors. One example is the testing effort, which data was not available in the original study, and neither in the previous replication study. Contrary results were also obtained in other studies [4], [29]. Wu et al. [40] identified that the review and testing efforts spent on the projects under study differed very much and could be the cause of such varying results. Therefore, more replications of this study, taking other possible influencing factors into account, are necessary, but left for future work.

The size–defect relationship, studied by the third group of hypotheses, has been widely used to address the product quality problem; see for example Koru et al. [18],[19]. The main reason for using static code attributes, such as the size and complexity, in fault prone module predictions lies in the absence of other metrics, and the simplicity of their automated collection. However, their prediction ability should be extensively validated and understood in a wider context before it can be accepted as a generally valid principle. Observing the NASA data sets from the PROMISE database[1], it was identified that the best static code attributes used in the fault prediction varies from data set to data set [23], and that attribute prediction ability greatly depends on the severity of faults [38]. Further investigation on these data sets has indicated that the simple classifiers suffice to model that relationship [21], although better predictions are obtained for a combination of attributes than using a single attribute.

The original study analyzed metrics for fault prediction, size metrics such as lines of code (LOC) and complexity metrics such as McCabe's cyclomatic complexity, and the SigFF metric which measures the number of communication signals between modules. For size as a fault predictor metric, no support or limited support is found in the original study. On the other hand, the previous replicated study found some support for that hypothesis but not consistently across the analyzed projects. For the fault density as fault predictor no support has been found in the original and the previous replication, but some other studies confirmed this hypothesis [13], [29]. Note that there is a methodological threat related to this predictor, see further discussion in Subsection 5.3. The hypothesis testing of complexity metric as a defect predictor has found no support or some weak support in the original study. In the previous replication study this hypothesis has not been tested due to lack of data required to perform the analysis.

Finally, the fourth group of hypotheses is related to benchmarking in terms of defect densities. As indicated in the original study, the main motivation is to engage collecting and publishing the data that would enable better intra and inter company comparisons, help in engaging the prediction based decisions and serve in building software engineering research [39]. Although some data, for example fault densities, are known within the company, they are rarely published. Still, the software engineering community lacks good prediction models and the selection of predictors may vary across the environments, thus leaving limited proof for adequacy of any prediction model in a wider context [5]. The accuracy of predictions is highly related to the level of similarity between the environments used in the prediction (process, data and domain) and could be improved by using quantified and validated environment for prediction approach [42]. The hypothesis that the fault densities are consistent between releases of projects had limited support in the original study, but was supported in the previous replication study. The hypothesis that the fault densities are similar in similar environments is confirmed for both studies.

Studying the same hypotheses and the variability of obtained results in a wider context helps us to better understand phenomena under analysis. In the original and previous replication studies, these groups of hypotheses were

[1] http://promisedata.org

analyzed for closed source industrial telecommunication software. Some of the hypotheses have also been analyzed by other researchers in other closed source contexts [40] and on Open Source Software (OSS) [9], [6], [25], [43], [35]. OSS is very different compared to commercial software, although in some aspects, for example, size, complexity, coupling and cohesion, OSS programs can be more similar to commercial software than other OSS programs[32]. Also, the impact of programming language (procedural vs. object–oriented) or chosen entity for analysis (e.g. module, component, file, etc) could also have impact on the results. This kind of analysis could help us to better understand variations in obtained results when testing the hypothesis in wider context.

3 Context of the study

The context of the study is described following the recommendations by Petersen and Wohlin [30]. The differences to the original and the previous replicated studies are highlighted, following the recommendations by Miller [24].

3.1 Product

The data used in this analysis are empirical data obtained from five projects developing sequential releases of a complex large scale telecommunication product, that we denote Rel n, Rel n+1, Rel n+2, Rel n+3, and Rel n+4. The product provides functionality for the Mobile Switching Centre (MSC), a functional node within Third Generation (3G) Core network, and is built on Ericsson's proprietary AXE telephone exchange. The product developed evolutionary in a sequence of releases over more than 30 years and is shared among different products, following the product line concept. It is installed within hundreds of telecom exchanges worldwide. During the evolution, the product line has been split into several product lines to evolve separately, thus forming the product line family. The product is composed of more than 1000 software units that are reused among product lines, either as common or modified units. Furthermore, each product from the product line serves a number of customers, satisfying a number of customer specific requirements. Therefore, each fault that is detected late could have serious consequences to a number of products that all serve to the telecommunication network with high real-time and reliability demands. Because of all these facts, Ericsson gives significant attention to the quality assurance activities and improvements, that is also evident from other publications (see for example [10], [2], [26], [11].) The analysis in the original and the previous replication study is also performed in the telecommunication domain and products were also part of the product line.

In this study, the analyzed part of the software product is the application part (e.g. signalling, traffic control, charging) written in the proprietary Programming Language for EXchanges (PLEX). The programming language used in the analyzed product of the original study was not specified, although the description indicates it was an earlier version of the PLEX language. In the

Table 1: Project characteristics

Study	Project	Project type	No units	No faults
Orig. study	Rel n	Application	140	1669
	Rel n+1		246	3646
Prev. repl. study	Proj 1	Application	45	1558
	Proj 2	Platform	90	4045
	Proj 3	Application	90	2419
This repl. study	Rel n	Application	302	5369
	Rel n+1		179	3563
	Rel n+2		216	4025
	Rel n+3		71	1523
	Rel n+4		71	4037

previous replication majority of code was written in C and some parts in Java. The original study analyzed two successive releases of the application part of the switching system, and the previous replication analyzed two application and one platform release of different consumer products which were sharing some common components, see Table 1.

The total number of units for the five releases included in this study are 302, 179, 217, 71, and 71, respectively, and the total number of faults included in the analysis is 5369, 3563, 4032, 1523, and 4037, respectively. These are larger numbers than in the original study and in the previous replication. Table 2 summarizes the distribution of modules in the analyzed sample by size in the original, the previous replication and this replication study[2]. Note that the module is a general term and is defined in accordance to the context of analysis. In our study, the term module is equivalent to the software unit that is a smallest self contained administrative unit of software product. The largest modules in this replication study are approximately more than five times larger than in the original study, and twice the size of the largest module in the previous replication.

[2]Basili and Perricones's study on IBM 360 data 1977–1980 categorize module sizes in ranges of 50 LOC, having 10 modules out of 370 bigger than 400 LOC in Fortran [3].

Table 2: Distribution of modules by size. Note that the previous replication study has different programming languages from the original and this replication study.

Original study			Previous replication study				This replication study					
LOC	Rel n	Rel n+1	LOC	P1	P2	P3	LOC	R n	R n+1	R n+2	R n+3	R n+4
≤1000	23	26	≤10 000	12	30	25	≤5 000	167	78	72	22	14
1001-2000	58	85	10 001-30 000	14	28	29	5 001-10 000	83	55	76	25	14
2001-3000	37	73	30 001-50 000	8	14	16	10 001-20 000	36	34	41	21	30
3001-4000	15	38	50 001-70 000	6	6	3	20 001-30 000	6	4	6	3	8
4001-5000	6	16	70 001-90 000	1	3	4	30 001-40 000	5	3	8	0	3
5001-6000	0	6	>90 000	4	9	13	>40 001	5	5	13	0	2
>6000	1	2										
Total	140	246	Total	43	90	90	Total	302	179	217	71	71

3.2 Organization and Process

The product development unit under study is globally distributed and during the observed period, the organization of the unit changed dramatically several times. The number of involved local development centers (LDC) distributed across the world, have varied during the five projects and consisted of eleven, five, four, five and four LDCs, respectively. That is different in comparison to the original study where more than 20 design centers were involved in the product development, and to the previous replication where projects involved only one development site.

The software development process has evolved, during the years of product evolution, from the traditional waterfall process and is now characterized by being incremental, iterative and focused on feature development. A number of quality assurance and verification activities are integrated throughout the entire software development process. Design, implementation and maintenance are performed by local development centers (LDC) having defined responsibilities over the particular modules forming the product. Remote LDCs use common development environment, processes, tools and databases. Local implementation may vary because of cultural differences and local habits. All unit verification activities and function testing (FT) is performed locally, but within simulated environments before modules are integrated into the system. At a defined milestone, the modules are delivered to the system integration and verification centre that is responsible for integration of modules into the system and performing the System test (ST). Again, at the next milestone the product is delivered to the network integration and verification organization, where the system is integrated into network and Site tests (SI) are performed. At the end, finally, the product is launched into operation (OP). As in the original study, we will follow the notation for the faults that are found during FT and ST testing as *pre-release* faults, and faults found during SI testing as *post-release* faults.

4 Study design

4.1 Hypotheses

In this study we re-use the hypotheses from the original and the previous replication studies. This enables us to additionally verify the original experiment, that has already been verified in the previous replication study, and to explore the hypotheses in another environment, thus investigating the compatibility of the results. In this way we provide further generalization of the results obtained in the original and previous replication study. The hypotheses were grouped into four groups as mentioned in Section 2. The numbering is the same as in the original study.

G1. *Hypotheses related to the Pareto principle of fault distribution*

1a A small number of modules contain most of the faults detected during pre-release testing.

1b If a small number of modules contain most of the pre-release faults, then it is because these modules constitute most of the code size.

2a A small number of modules contain most of the faults detected during post-release testing.

2b If a small number of modules contain most of the post-release faults, then it is because these modules constitute most of the code size.

G2. *Hypotheses related to the persistence of faults*

3 A higher incidence of faults in function testing (FT) implies a higher incidence of faults in system testing (ST).

4 A higher incidence of faults in pre-release testing implies a higher incidence of faults in post-release testing and use.

G3. *Hypotheses about effects of module size and complexity on fault proneness*

5a Smaller modules are less likely to be fault–prone than larger ones.

5b Size metrics are good predictors of pre-release faults in a module.

5c Size metrics are good predictors of post-release faults in a module.

5d Size metrics are good predictors of a module's pre-release fault density.

5e Size metrics are good predictors of a module's post-release fault density.

6 Complexity metrics are better predictors than simple size metrics of fault and fault-prone modules (due to lack of data, this hypothesis is not included in our replication study).

G4. *Hypotheses about the quality in terms of fault densities*

7 Fault densities at corresponding phases of testing and operation remain roughly constant between subsequent major releases of a software system.

8 Software systems produced in similar environments have broadly similar fault densities at similar testing and operational phases.

4.2 Data collection

As in the original study and the first replication, the study procedures do not intervene with the projects but only passively collect data from several sources. The dependent variable is the number of faults and the independent variable is the lines of code (LOC), as in the original and the previous replication studies.

Information about modules is collected from quality reports, which are spreadsheets reporting results of verification activities in the project. The following information is reported for each module: module name, identity and revision, modified and total size of code, number of faults detected during unit verification and FT. Similar to the previous studies, only modules that have been modified within the project are included. Note that these modules are not all the modules composing the final system.

The underlying faults causing the failures experienced during the ST, SI and OP, are reported in the form of the so called Trouble Reports (TR), and are addressed to a particular location in the software product (e.g. particular module). If the failure is related to faults in several locations in the system, then one TR is issued per location. On the other hand, this single fault could be the cause of a number of failures, resulting in several duplicate TRs, which later have to be removed. We collected all TRs reported on the modules listed in the Quality reports of each analyzed project. In our sample, all duplicate TRs were excluded and thus every fault is counted only once. Also, there are faults that are wrongly assigned to modules and rerouted to another module, and faults that are still not answered. In our analysis we included only TRs with 'Status=Finished' meaning that all activities regarding this TR are completed. Every TR also has the 'Answer code', which contains different values encoding the action taken on that fault. In our analysis we included only TRs that were classified as correction is needed in the source code and has to be solved; all other TRs were excluded from the analysis to avoid noise in the data. Each testing activity has its own reference, that is stored as the 'Market reference' of the TR. This allows us to collect TRs related to ST and SI testing separately. In our study, we did not have the opportunity to collect the OP faults. Hence, the post-release faults refer only to the SI faults.

The modules in the original study "were selected randomly for analysis from the set of modules that were either new or had been modified" [10]. In the previous replication, the "samples are limited to the modules for which [code] data could be collected automatically" [2], thus excluding some special modules. In our analysis we included the modified and new modules that had faults reported. Moreover, we identified two outliers, both in release Rel n+3. We removed these two unusual observations from that sample. These two observations have unusually high amount of faults compared to other testing phases of the same module and compared to other software modules.

4.3 Data analysis techniques

The data analysis techniques used in this replication are the same as the ones used in the previous replication study. These are Alberg diagrams [28], scatter plot diagrams and correlation analysis. The replication analysis is done by basic vote counting (i.e. counting data point for or against the hypotheses [31]), since raw data is not available from the original study, and the few data points (mostly three) does not allow any statistical analyses.

5 Data Analysis and Results

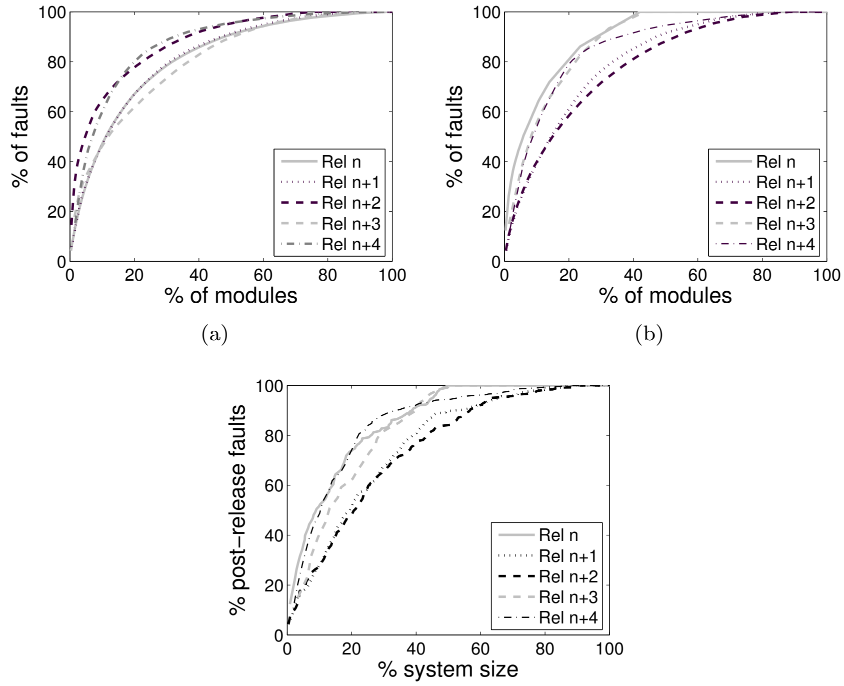
The analysis of the hypotheses stated in Section 4 is performed in the context of the study described in Section 3. The results are discussed for each group of hypotheses in separate subsections, and in relation to the original and previous replication study. Also, where appropriate, the relation to other studies is elaborated.

5.1 Hypotheses related to the Pareto principle of fault distribution

The main idea behind the Pareto principle, also known as 80–20 rule, has been widely used within the software engineering community, mostly in the form that the 20% of the software modules are responsible for 80% of the faults (hypothesis 1a and 2a). However, if these 20% of modules constitute the majority of the system size, its practical application would be meaningless (hypothesis 1b and 2b).

Hypothesis 1a. *A small number of modules contain most of the faults detected during pre-release testing*

The hypothesis 1a, as stated in the original and previous replications, is testing the percentage of modules in relation to the percentage of the pre-release faults. Here, we repeat the same analysis, reporting the results using the Alberg diagram in Figure 1(a). The modules are sorted in decreasing order with respect to the number of pre–release faults, and the hypothesized relationship is represented in the percentage scale. Almost identical graphs are obtained for all analyzed releases. Moreover, these graphs are very similar to the graphs in the original and previous replication. When compared at the point corresponding to 20% of most pre-release fault prone modules, the original study reports that 60% of pre-release faults are located in these modules, the previous replication 63%, 70% and 70% for the three projects, respectively, and this replication 67%, 66%, 78%, 63% and 80% for the five releases, respectively (see Table 3). We consider these results be consistent across the original, previous replication and this replication study.



(a) (b) (c)

Figure 1: Alberg diagram showing the percentage of: (a) modules versus the percentage of pre-release faults, (b) modules versus the percentage of post-release faults, and (c) system size versus the percentage of post-release faults

Hypothesis 1b. *If a small number of modules contain most of the pre-release faults, then it is because these modules constitute most of the code size.*

The analyses performed in the original study and previous replication studies have found no support for this hypothesis. The result obtained in this study is consistent with the previous ones, thus, giving even stronger support for the applicability of Pareto principle as stated in hypothesis 1a. At the point corresponding to 20% of most pre-release fault prone modules, their share in the system size is 30% in the original study, 38%, 25% and 39% in the three projects of the previous replication study, and 32%, 28%, 23%, 26% and 23% in the five releases, respectively, of this replication study. Therefore, we conclude the hypothesis 1b is *not* supported (see Table 3).

Hypothesis 2a. *A small number of modules contain most of the faults detected during post-release testing*

The Alberg diagram is used to analyze this hypothesis (see Figure 1(b)), comparing two observation points in the diagram. At the point corresponding to 10% of the most post-release fault prone modules, the percentages of post-release faults were the following: in the original study 100% and 80%, in the previous replication 63%, 74% and 59% for three projects, and 62%, 39%, 40%, 55% and 53% for the five releases in this replication study. At the point corresponding to 20%, the percentage of the most post-release fault prone modules in the previous replication study is 87%, 88%, and 80%, for the analyzed projects, and in this replication is 81%, 61%, 58%, 76% and 81% for the five sequential releases, respectively (see Table 3). In the original study, the hypothesis was strongly supported already at the 10% level, whereas in this replication study, this hypothesis is confirmed first at 20% of the modules. The previous replication study also gives stronger support to this hypothesis than this second replication study does.

It is important to notice that there are certain differences in how the post-release faults were measured between the original, previous and this replication. In the original study, the post-release faults included all SI and faults from the first *year* of operation. In the previous replication, all SI faults and faults from the first *months* of operation were included, while in this replication *only SI* faults represent the post-release faults. Although we can not bring definite conclusions based only on these three studies, it is indicative that for longer period of operation, the growth of graphs in the Alberg diagram is faster. A possible explanation for this observation could be that the operational faults are concentrated in an even smaller portion of modules than post-release faults in general.

In the original and previous replication studies, stronger support is observed for hypothesis H2a regarding post-release faults than for hypothesis H1a regarding the pre-release faults, although with higher variability for post-release faults. In this study, the support for hypothesis H1a and H2a is very similar, although the variability is again higher for post-release faults.

Table 3: H1: Percentage distribution of pre-release faults over modules related to size; H2: Percentage distribution of post-release faults over modules related to size

			H1a	H1b	H2a	H2b
Study	Project	Share of mod.	Share of pre-r. faults	Share of sys. size	Share of post-r. faults	Share of sys. size
Orig. study	Rel n	10%	-	-	100%	12%
	Rel n+1	10%	-	-	80%	-
	Rel n	20%	60%	30%	-	-
	Rel n+1	almost identical to Rel n			-	-
Prev. repl. study	Proj 1	10%	-	-	63%	19%
	Proj 2	10%	-	-	74%	10%
	Proj 3	10%	-	-	59%	27%
	Proj 1	20%	63%	38%	87%	24%
	Proj 2	20%	70%	25%	88%	22%
	Proj 3	20%	70%	39%	80%	40%
This repl. study	Rel n	10%	-	-	62%	15%
	Rel n+1	10%	-	-	39%	15%
	Rel n+2	10%	-	-	40%	15%
	Rel n+3	10%	-	-	55%	16%
	Rel n+4	10%	-	-	53%	11%
	Rel n	20%	67%	32%	81%	27%
	Rel n+1	20%	66%	28%	61%	27%
	Rel n+2	20%	78%	23%	58%	25%
	Rel n+3	20%	63%	26%	76%	26%
	Rel n+4	20%	80%	23%	81%	22%

Hypothesis 2b. *If a small number of modules contain most of the post-release faults, then it is because these modules constitute most of the code size*

The results for the original, previous and this replication study are given in Table 3. In *neither* of the three studies the hypothesis H2b is supported. However, contrary to the previous studies, in our case the converse hypothesis to H2b is *not* supported either, because 100% of post-release faults were contained in modules that make 50%, 88%, 92%, 50% and 88% of the system size for the five projects, respectively. However, already 80% of post-release faults are contained in 26%, 39%, 43%, 28% and 22% of the system size, respectively. Thus, most of the post-release faults are concentrated in small portion of the system size, but the last few are spread across most of the system. The Alberg diagram of percentage of system size versus the percentage of post-release faults is shown in Figure 1(c). Again, it is worth to mention that the sample of post-release faults in this replication consisted only of SI faults.

5.2 Hypotheses related to the persistence of faults

Planning of later verification activities for a software system is usually based on the results obtained in earlier verification. The widely used principle is that higher incidence of faults in earlier verification implies higher incidence of faults in subsequent verification activities. Therefore, this group of hypotheses is stated to test this principle.

Hypothesis 3. *Higher incidence of faults in FT implies higher incidence of faults in ST*

The scatter plots representing the relation of the FT faults and ST faults on each software module are presented in Figure 2(a) for five releases, respectively. It can be observed that there exist some relationships between FT and ST faults, as was also indicated in the original and previous replication studies using the same plots. As in the previous replication study, the statistically significant correlation (that is, with p–value< 0.05) is identified and confirms the stated hypothesis, meaning that the majority of ST faults are contained in the modules where the majority of FT faults is located. The Pearson correlation coefficient r equals 0.86, 0.82, 0.96, 0.83 and 0.94, for the five releases, respectively, which indicate quite strong correlations, although the highest data point tend to have an un-proportionally high impact on the parameters of the correlation line. Similar results were obtained in the previous replication, in which the coefficient r equals 0.74, 0.84 and 0.68 in the three analyzed projects, respectively.

Figure 3 depicts the Alberg diagrams for accumulated percentage of ST faults, the dotted lines show modules ordered according to the number of FT faults, and the solid line show modules ordered according to the number of ST faults. In the original and the previous replicated study the same diagrams were used in the analysis and the selected point of observation was at 50% of faults detected in the ST. Columns under H3(a) in the Table 4 summarizes the results obtained in the previous studies along with the results obtained in this replication study. The results indicate that 50% of ST faults occurred in the modules that were responsible for 37% and 25% of FT faults in the original

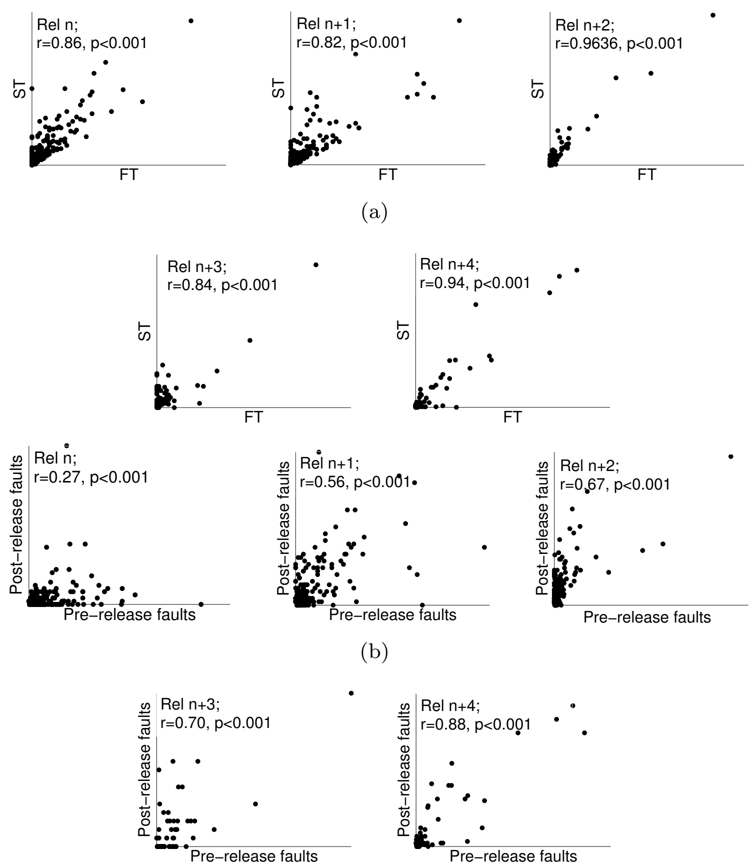


Figure 2: Scatter plot showing relationship: (a) between faults detected in function test and faults detected in system test, (b) between pre-release and post-release faults faults

study, 40%, 39% and 38% in the previous replication, and 54%, 53%, 63%, 62% and 48% in this replication study. The highest concentration of FT faults in the observed modules was obtained in this replication study, thus providing the stronger evidence for this hypothesis than in the previous studies.

Furthermore, the columns under H3(b) in Table 4 summarizes the results obtained for 10% of modules that were the most fault prone in the ST. The the fifth and sixth column of the table list the percentage of faults found in those modules during ST and FT, respectively. We can observe that faults in FT imply faults in ST, but the levels vary for the analyzed projects and consequently, prediction models for number of ST faults based on the number of FT faults must be calibrated for different situations [1]. Note that in all projects under H3(b) in Table 4, the percentage of FT faults is less than the percentage of ST faults for the 10% of the most ST fault prone modules.

Table 4: H3: (a) Percentage distribution of system test faults over function test; (b) Faults in ST for 10% of the most fault prone modules when ordered with respect to fault proneness in the ST and the FT; H4: Percentage of pre-release faults in modules that have no subsequent post-release faults (–PoR)

		H3(a)		H3(b)		H4	
Study	Project	%ST	%FT	%ST	%FT	%FT +ST	%–PoR modules
Orig. study	Rel n	50%	37%	38%	17%	93%	
	Rel n+1	50%	25%	46%	24%	77%	
Prev. repl. study	Proj 1	50%	40%	53%	39%	36%	
	Proj 2	50%	39%	56%	52%	29%	
	Proj 3	50%	38%	56%	39%	13%	
This repl. study	Rel n	50%	54%	47%	43%	26%	XX%
	Rel n+1	50%	53%	46%	38%	10%	XX%
	Rel n+2	50%	63%	62%	40%	3%	XX%
	Rel n+3	50%	62%	43%	49%	25%	XX%
	Rel n+4	50%	48%	60%	58%	2%	XX%

The correlations determined in this study confirm the findings from the previous replication study and lead to the conclusion that this hypothesis is strongly supported. The support for this hypothesis is even stronger than in the previous replication study.

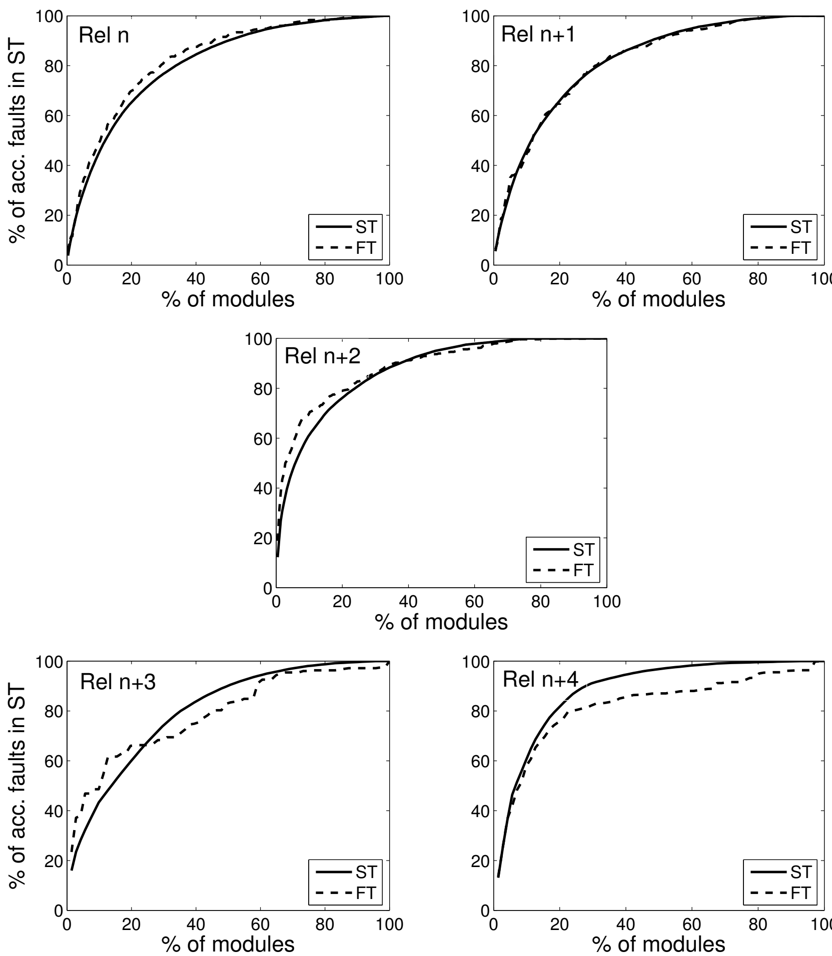


Figure 3: Alberg diagrams showing accumulated percentage of the number of faults in *system test* when modules are ordered with respect to the number of faults in *system test* and *function test*, respectively.

Hypothesis 4. *A higher incidence of faults in pre-release testing implies higher incidence of faults in post-release*

Strictly speaking, this hypothesis can not be tested in this study, since we do not have data regarding OP faults. We tested this hypothesis conditionally, counting post-release faults only as the faults detected during SI testing. In Figure 2(b) scatter plots for the five analyzed releases are presented. In these plots each dot represents one system module, and its position in the plot is defined by the number of pre-release and post-release faults in that module. From the statistical analysis using Pearson correlation, it can be concluded that there exists statistically significant correlation (that is, p–values< 0.001 between the pre-release and post-release faults in majority of releases. The Pearson correlation coefficients r in the five releases are 0.27, 0.56, 0.67, 0.70 and 0.88, respectively. A possible reason for variability of r may be in the sample size and possible outliers.

Furthermore, we analyzed the modules with no subsequent post-release faults and the percentage of pre-release faults detected in such modules is listed in seventh column of Table 4. The table shows that this replication provides contrary results, compared to the original study. In the modules with no subsequent post-release faults (XX% of the total number, respectively), the percentage of pre-release faults was 26%, 10%, 3%, 25% and 2% for the five releases, respectively. Although, we were testing this hypothesis conditionally (only SI faults were included), the obtained result confirms the hypothesis unconditionally, because any additional OP fault would only lower the percentage. Similar results were obtained in the previous replication as well. The percentages of pre-release faults in the analyzed modules were 36%, 29% and 13% for three different projects, while the original study found *no* evidence to support this hypothesis and could even report evidence to support the converse hypothesis.

Other studies also report inconsistent results. Wu et al. [40] found that 38.4% of not fault-prone modules in testing contain 94.4% of faults in the field and that 61.6% of fault-prone modules in testing contain 5.6% faults in the field. This result was obtained for one project but is not consistent with sequential project under the same conditions, by the same experienced developers in the same environment, and with the same languages and tools. After further investigation performed in that context, the authors report that reason for such results lies in the variation of review and testing effort invested in each project. The previous studies also noticed that there are other factors that could influence the results obtained. For example, the fault density measure is a better measure of verification process then of product quality [11].

5.3 Hypotheses about the effects of module size and complexity on fault proneness

There are fault prediction techniques developed on the basis of various code attributes. The most popular code attributes are size and complexity measures, but also the fault density is used assuming the linear relationship between size and faults. In the original study, this group of hypotheses includes the hypothe-

Table 5: Hypotheses 5a–e. For each hypothesis: correlation coefficients between module size and various fault count measures

Study	Project	Corr. coeff.	5a Total number of faults	5b Pre-release faults	5c Post-release faults	5d Pre-release fault density	5e Post-release fault density
Prev. repl. study	Proj. 1	Pearson	0.38	0.37	0.44	–0.17	–0.15
	Proj. 2	Pearson	0.05	0.05	0.08	–0.22	–0.16
	Proj. 3	Pearson	0.62	0.60	0.65	–0.10	–0.02
This repl. study	Rel n	Pearson	0.30	0.29	0.16	–0.16	–0.07
		Spearman	0.49	0.49	0.3	–0.18	0.14
	Rel n+1	Pearson	0.23	0.22	0.16	–0.19	–0.22
		Spearman	0.42	0.38	0.35	–0.18	–0.013
	Rel n+2	Pearson	0.06	0.02	0.15	–0.10	–0.23
		Spearman	0.39	0.29	0.37	–0.10	–0.15
	Rel n+3	Pearson	0.37	0.35	0.38	–0.10	0.07
		Spearman	0.29	0.27	0.3	–0.23	0.16
	Rel n+4	Pearson	0.08	0.08	0.06	–0.10	–0.16
		Spearman	0.14	0.15	0.099	–0.27	–0.31

sis that simple size metrics, such as Lines Of Code (LOC), are good predictors of fault prone modules, faults and fault density (hypotheses H5a, H5b, H5c, H5d, and H5e), and the hypothesis that the complexity metrics are better predictors than simple size metrics (hypothesis H6). As in the previous replication study, we were not able to test hypothesis H6 because of lack of relevant data.

Hypothesis 5a. *Smaller modules are less likely to be failure-prone than larger ones*

In the previous replication study, hypothesis H5a was analyzed using the correlation between the size of modules and total number of faults. The results from the previous replication and this replication are presented in the fourth column of Table 5. In this replication we did not identify any correlation between the total number of faults and the total volume, as correlation coefficients for all five releases are low. Hence, this hypothesis is *not* supported. In the previous replication study some correlation between the total number of faults and the total LOC had been identified for Project 3. Mohaghegi and Conradi [26] identified weak correlation between total number of faults and the total LOC. The correlation is also studied for the reused and non-reused components separately, and strong correlation is identified for non-reused components.

Hypothesis 5b. *Size metrics are good predictors of pre-release faults in a module* The correlation coefficients between the LOC and pre-release faults are given in the fifth column of Table 5 for each analyzed release. The hypothesis is not supported as the correlation coefficients are low. The same result could be observed from the scatter plots presented in Figure 4(a). In the previous replication, the correlation coefficients were very low and only for Project 3 a moderate correlation is identified. Also, in the original study, no strong evidence has been identified in favor to this hypothesis, and conclusions were based solely on scatter plots, without examining the correlation coefficients. The study performed by Wu et al. [40], also indicates that there is *no* relationship between size and pre-release faults.

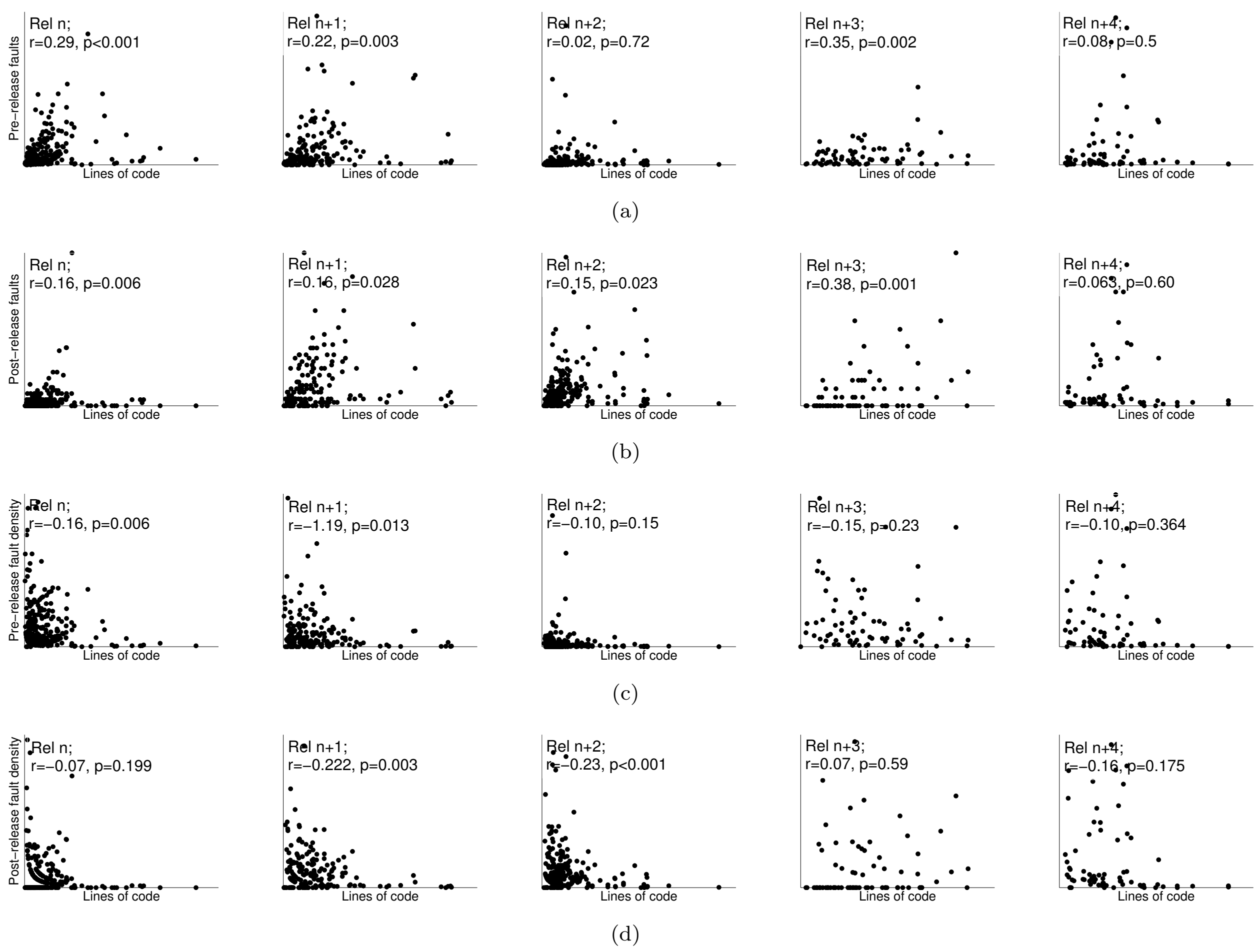


Figure 4: Scatter plot showing relationship between: (a)LOC and pre-release faults, (b)LOC and post-release faults, (c)LOC and pre-release fault density, (d)LOC and post-release fault density

Hypothesis 5c. *Size metrics are good predictors of post-release faults in a module*

Figure 4(b) presents scatter plots of size in LOC versus post-release faults. From this figure, as from the correlation coefficients given in the sixth column of Table 5, no support for this hypothesis is found. Similar results were obtained in the original and previous replication studies, as well as in some other literature [40]. On the other hand, the Alberg diagram in the original study, showing the accumulated number of faults when the modules are ordered with respect to the module size in LOC, reveals that size is a better predictor of fault-prone modules than other considered metrics. In our case, such Alberg diagrams are presented in Figure 5. It shows that in all releases, module size is not a good predictor of fault proneness.

Hypothesis 5d. *Size metrics are good predictors of a module's pre-release fault density*

A number of studies have analyzed this hypothesis. The studies have found that smaller modules have higher fault densities [3], [37], and that larger modules also tend to have higher fault densities, leading to the conclusion that the module has an optimal size regarding the fault density [22], [8]. However, El Emam et al. [8] prescribed this to the mathematical artifact (plotting LOC against 1/LOC). Consequently, this hypothesis, as concluded in the previous replication, represents a methodological threat, but is still reported for the completeness of the replication. Koru et al. proposed an alternative analysis method without the threat [19]. Despite that the measure would overstress such a relation, *no* support was found for this hypothesis in the original study. The contradictory observations have been noticed for the two analyzed releases. In the previous replication study, the correlation analysis indicated a negative correlation but since the association was low the analysis did not support this hypothesis either. Similar results are obtained in this study, see scatter plots in Figure 4(c), the correlation coefficients in Table 5. We may conclude that the linear relationship between size and fault count is not always observable, although some general indication that fault count increases with system size is noticed.

Hypothesis 5e. *Size metrics are good predictors of a module's post-release fault density*

Figure 4(d) presents this relationship for the five releases analyzed in this paper and the correlation coefficients are given in the Table 5. As in the previous hypothesis, the negative correlation is obtained and the relatively small value of the coefficient does *not* support the hypothesis. Finally, we considered the average fault densities of modules of similar size. Such analysis in the original study replicates the same analysis by Basili and Perricone [3]. As in the original study, the fault densities for modules grouped by size in Table 6 do not show any regularity.

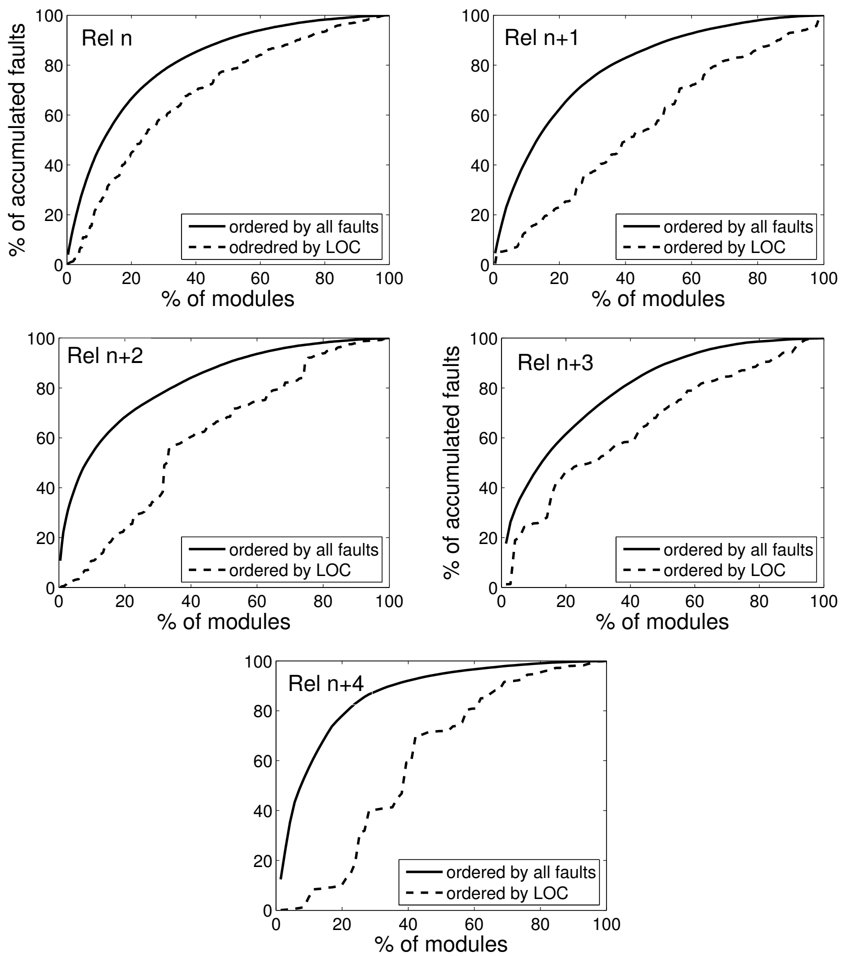


Figure 5: Accumulated percentage of number of faults when modules are ordered with respect to LOC.

Table 6: Average number of faults/fault densities for Release n, n+1, n+2, n+3, n+4

Rel	n	n+1	n+2	n+3	n+4
Size [$kLOC$]	f/FD	f/FD	f/FD	f/FD	f/FD
< 5	167/3.44	78/3.53	72/3.53	22/3.50	14/4.78
$5 - 10$	83/2.97	55/3.59	76/2.94	25/2.45	14/5.22
$10 - 15$	24/3.07	23/2.62	28/1.93	12/1.41	16/7.40
$15 - 20$	12/3.41	11/2.03	13/1.50	9 /2.35	14/6.50
> 20	16/1.05	12/0.73	27/0.54	3/4.58	13/1.04

5.4 Hypotheses about the quality in terms of fault densities

Publishing of benchmarking measures enables definition of standards for guiding the software process improvement activities. In the original and the previous replication study, the only hypothesized benchmarking measure was the fault density: between the subsequent testing phases within the project (H7) and between the same testing phases of different projects (H8). The same analysis is performed in this replication study. Some partial results have already been reported by Galinac Grbac and Huljenić [11].

Hypothesis 7. *Fault densities at corresponding phases of testing and operation remain roughly constant between subsequent major releases of software system*

The fault densities were calculated as the ratio of the total number of faults divided by the total volume of code. This ratio is calculated for each testing phase separately, in our case these are FT, ST and SI, and the results are given in Table 7. Observing the results we can conclude that the fault densities for the phases of testing remain in the same order of magnitude, although it varies up to a factor of four between releases. Moreover, the consistent results are obtained in five analyzed projects, indicating that the process is stable and repeatable. This is an interesting result, since the organization has been changed over the projects. Furthermore, if we compare our results with previous studies, then we can notice that the FT fault densities in releases analyzed in this replication are similar to the FT fault densities of projects analyzed in the previous replication, and ST fault densities are similar to the ST fault densities of the releases analyzed in the original study. As in the previous studies, based solely on observing the results presented in the table, we found support for this hypothesis (see Table 7).

Hypothesis 8. *Software systems produced in similar environments have broadly similar fault densities at similar testing and operational phases*

The order of magnitude decrease in fault density between pre-release and

Table 7: H7: Fault densities at three phases of testing

Study	Project	FT	ST	SI	OP	CAT
Orig.	Rel n	3.49	2.60	0.07	0.20	-
study	Rel n+1	4.15	1.82	0.43	0.20	-
Prev.	Proj 1	0.7	0.2	-	-	0.03
repl.	Proj 2	1.2	0.5	-	-	0.07
study	Proj 3	0.4	0.2	-	-	0.07
	Rel n	0.71	1.7	0.20	-	-
This	Rel n+1	0.55	1.15	0.71	-	-
repl.	Rel n+2	0.38	0.68	0.65	-	-
study	Rel n+3	0.58	1.74	0.20	-	-
	Rel n+4	1.30	2.17	0.83	-	-

post-release faults is observed in all three studies (see Table 8). In our study we considered only SI faults, so that the post-release fault density is probably higher than reported in the table. Overall fault density is in line with best practices reported in other studies [10]. However, comparing the results across different projects using small sample of projects may give misleading conclusions. In previous replication, two application projects looks more similar than the application and platform projects. In this study, the fault densities at similar testing phases of different projects are more similar to one another than compared to other studies. Wu et al. [40] obtained consistent results for fault densities measured in seven projects. The pre-release fault densities are in the range 1.95–10.16 and post-release failure densities 0.013–0.824.

The relationship between fault densities were analyzed across two sequential system releases within product family with respect to the module reuse by Mohagheghi and Conradi [26]. Reused modules had lower fault density then non-reused ones. In that sense, the variation in fault densities between releases might be caused by the level of reusing the software modules. Similar results were obtained by Ostrand and Weyker [29] when analyzing fault density for new and older files, and the result was that the fault densities tend to decrease as the system matures.

5.5 Results in open source projects

Open source software (OSS) is becoming an alternative to closed source software in many applications, and fault data are easily accessible. Consequently, a growing trend in empirical studies in the field of OSS is aiming to approximate industrial software and provide some generalization of results that would provide stronger basis for software engineering community [32]. However, the OSS development process may be very different from a closed source process. The OSS is mostly evolutionary developed and module sizes can vary dramatically

Table 8: H8: Fault densities pre- and post-release

Study	Project	Pre-release	Post-release	All
Orig. study	Rel n	6.09	0.27	6.36
	Rel n+1	5.97	0.63	6.60
Prev. repl. study	Proj 1	0.90	0.03	0.93
	Proj 2	1.70	0.07	1.77
	Proj 3	0.60	0.07	0.67
This repl. study	Rel n	2.41	0.20	2.61
	Rel n+1	1.70	0.71	2.41
	Rel n+2	1.06	0.65	1.71
	Rel n+3	2.32	0.20	2.52
	Rel n+4	3.47	0.83	4.30

between successive product releases, built by a number of developers (often volunteers), with no explicit system level and detailed design, and without any plan and schedule [25]. The majority of testing is left to its users and the development process lacks a systematic approach to quality improvements [20]. The OSS are mostly object-oriented systems written in C++ or Java programming languages. Here we summarize empirical results on OSS related to the four groups of hypotheses.

G1. *Pareto distribution of faults.* This is widely investigated in OSS projects and very consistent results are obtained. Bugzilla data for the Java Development Kit component of Eclipse OSS project: 20–82 [9], Apache server 2.0: 20–60 [6], Eclipse 3.0 for files: 20–63 in pre-release and 20–60 in post–release, and Eclipse 3.0 for packages: 20–60 in pre–release and 20–64 post–release [41].

G2. *Persistence of faults.* The usefulness of early fault data to predict late fault data was also confirmed by OSS data. Zimmermann et al. [43] obtained significant and high correlation coefficients (0.907 for files and 0.921 for packages) meaning that the files/packages having high pre-release defect count will most likely also have high post-release defect counts. The study performed on Eclipse 2.0, 2.1 and 3.0 projects by Shihab et al. [35] again confirms these results, and even found evidence that the pre-release defects are one of the most stable predictors across analyzed releases.

G3. *Effects of module size and complexity on fault proneness.* Using code metrics for defect prediction has been widely investigated and *no* support has been found in favor to this hypothesis.In the Eclipse 3.0 project, the Spearman correlation coefficients for the relationship LOC and pre-release/post-release faults was 0.407/0.420 (for files) and 0.461/0.419 (for packages) [43]. Similar results were obtained for the McCabe complexity measures. The LOC measure is identified as the most significant and most stabile predictor of post-release defects across three Eclipse projects analyzed [35].

G4. *Quality in terms of fault densities.* OSS software quality is investigated

Table 9: Summary of hypotheses in the original study, previous replication and this replication study

Hypothesis	Original Study	Previous replication study	This replication study
1a Few modules contain most faults (pre-release)	Confirmed (20–60)	Confirmed (20–63; 20–70; 20–70)	Confirmed (20–67; 20–66; 20–77; 20–63; 20–80)
1b Few faulty modules constitute most of the size (pre-release)	No support (20–30)	No support (20–38; 20–25; 20–39)	No support (20–32; 20–29; 20–22; 20–26; 20–23)
2a Few modules contain most faults (post-release)	Confirmed (10–80; 10–100)	Confirmed (10–63; 10–74; 10–59)	Confirmed (20–81; 20–61; 20–58; 20–76; 20–81)
2b Few faulty modules constitute most of the size (post-release)	No support; strong evidence of a converse hypothesis (100–12; 60–6)	No support (100–32; 100–41; 100-70)	No support (20–27; 20–27; 20–25; 20–26; 20–22)
3 High fault incidence in FT implies the same in ST	Limited support (50–25; 50–37)	Strong support (50–40; 50–39; 50–38)	Strong support (50–54; 50–53; 50–63; 50–62; 50–48)
4 High fault incidence pre-release implies the same post-release	No support – strongly rejected (93–0; 77–0)	Support (36–0; 29–0; 13–0)	Confirmed (26–0; 10–0; 3–0; 25–0; 2–0)
5a LOC is a good predictor of faults	No support	Varying support	No support
5b LOC is a good predictor of pre-release faults	Limited support	Varying support	No support
5c LOC is a good predictor of post-release faults	No support	Varying support	No support
5d LOC is a good predictor of pre-release fault density	No support	No support	No support
5e LOC is a good predictor of post-release fault density	No support	No support	No support
6 Complexity metrics are good predictors of faults	No (for cyclomatic complexity), some week support for specific metric	N/A	N/A
7 Fault densities are constant between releases or projects	Limited support	Support	Limited support
8 Fault densities are similar in similar environments	Confirmed	Confirmed	Confirmed

a lot in the relation to the quality of the closed software systems, but still with limited empirical basis for making such comparisons. The major OSS software releases, Apache server and Mozilla, were analyzed in relation to five projects developing software from the telecommunication domain [25]. The results indicate that pre-release fault density in the analyzed OSS software is lower than in the commercial software, and the post-release fault densities are higher in OSS then in the commercial software. A replication of this study [7] performed for OSS version of the Unix project, confirmed that the fault densities of OSS system are comparable to the fault densities of commercial systems.

6 Discussion

In this section, we discuss results of the complete study, and their consistency to the original and previous replication summarized in Table 9, having in mind the difference in the context in which the studies are performed.

The three studies were all performed on closed proprietary software in the telecommunication domain. However, the studies differ in having analyzed different products, products of different size, and the analysis covering different sample sizes, with different ranges in module size, programming language, and organization of development work. Although not completely described, we consider the development process having similar characteristics in these environments. The quality assurance process of all three studies could be analyzed and measured in the way that is explained in the original study, so no major differences have been identified in that sense. Still there are substantial differences between the processes, for example, with respect to iterativeness. In light of this, the consistent results across the different constexts may be considered a surprise.

All three studies give consistent answers to the first set of hypothesis (H1a–b, H2a–b). A minority of modules contain a majority of faults. This is true for both pre-release and post-release faults. In neither case, this is caused by an uneven distribution of module sizes, hence it is *not* due to this minority of modules comprising any substantially larger share of the total size. The original study provided limited support for the hypothesis (H3) that fault-prone modules in function test also are fault-prone in system test. Both replications give strong support for the hypothesis, and hence we conclude that the same modules tend to be fault-prone across those test phases. The hypotheses regarding pre- versus post-release (H4) was *not* supported by the original study, but both replications support that the same modules tend to be fault-prone both before and after release. Numerous prediction models have been proposed to identify fault-prone modules based on code metrics, like size and complexity. Neither of these three studies provide any consistent results of such metrics as predictors of fault proneness. At most, size have been shown to explain 40% of the variation. Hence the hypotheses about fault proneness prediction (H5a–e) are rejected, since size is not a sufficient predictive factor. Fault densities, finally (H7–8) are concluded to be of the same magnitude of size between releases and across environments. Still there are variations of a factor of four between releases within the same phase. Prediction models may not be built across different environments, but must be calibrated to specific contexts, and even so be quite error prone.

7 Conclusion

We report a second replication study of Fenton and Ohlsson's study from the late 1990's [10]. The study is as close as it can be more than 10 years later, aiming for a *literal* replication. Further, parts of the study originate from the late 1970's [3]. We also conduct the analysis in light of the first replication, from 2007 [2]. We study four groups of hypotheses, as defined by Fenton and Ohlsson [10]: i) Pareto principle of fault distribution, ii) persistence of faults, iii) effects of module size and complexity on fault proneness, and iv) quality in terms of fault densities.

In conclusion, the Pareto principle is clearly confirmed in this replication, which makes it worthwhile to try to identify fault-prone modules and spend un-evenly distributed efforts on testing different parts of the system. Modules identified to be fault-prone in one phase tend to be so in subsequent phases, paving the way for the first set of candidates to focus on. Size related predictors, on the other hand, are not given any support for being good enough to identify fault-prone modules. Finally, the fault density across releases and environments is of the same magnitude, but still varies a lot with factors not under control in the current studies.

References

[1] C. Andersson and P. Runeson, "A Spiral Process Model for Case Studies on Software Quality Monitoring – Method and Metrics," *Softw. Process Improv. Pract.*, vol. 12, no. 2, pp. 125-140, Mar./Apr. 2007.

[2] C. Andersson and P. Runeson, "A Replicated Quantitative Analysis of Fault Distributions in Complex Software Systems," *IEEE Trans. Softw. Eng.*, vol. 33, no. 5, pp. 273–286, May 2007.

[3] V.R. Basili and B.T. Perricone, "Software Errors and Complexity: an Empirical Investigation," *Commun. ACM*, vol. 27, no. 1, pp. 42–52, Jan. 1984.

[4] B.T. Compton and C. Withrow, "Prediction and Control of ADA Software Defects," *J. Syst. Softw.*, vol. 12, no. 3, pp. 199–207, July 1990.

[5] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating Defect Prediction Approaches: a Benchmark and an Extensive Comparison," *Empirical Softw. Eng.*, vol. 17, no. 4–5, pp. 531–577, Aug. 2012.

[6] G. Denaro and Mauro Pezzè, "An Empirical Evaluation of Fault-Proneness Models," *Proc. 24th Internat. Conf. on Softw. Eng. (ICSE '02)*, pp. 241–251, May 2002.

[7] T.T. Dinh-Trong and J.M. Bieman, "The FreeBSD Project: A Replication Case Study of Open Source Development," *IEEE Trans. Softw. Eng.*, vol. 31, no. 6, pp. 481–494, June 2005.

[8] K. El Emam, S. Benlarbi, N. Goel, W. Melo, H. Lounis, and S.N. Rai, "The Optimal Class Size for Object-Oriented Software," *IEEE Trans. Softw. Eng.*, vol. 28, no. 5, pp. 494–509, May 2002.

[9] M. English, C. Exton, I. Rigon, and B. Cleary, Fault Detection and Prediction in an Open-Source Software Project," *Proc. 5th Internat. Conf. on Predictor Models in Softw. Eng. (PROMISE '09)*, pp. 17:1–17:11, May 2009.

[10] N.E. Fenton and N. Ohlsson. "Quantitative Analysis of Faults and Failures in a Complex Software System," *IEEE Trans. Softw. Eng.*, vol. 26, no. 8, pp. 797–814, Aug. 2000.

[11] T. Galinac Grbac and D. Huljenić, "Defect Detection Effectiveness and Product Quality in Global Software Development," *Product-Focused Softw. Process Improv. (PROFES '11)*, *Lect. Notes in Computer Sci.*, vol. 6759, pp. 113–127, June 2011.

[12] O. Gómez, N. Juristo, and S. Vegas, "Replication Types in Experimental Disciplines," *Proc. ACM-IEEE Internat. Symp. on Empirical Softw. Eng. and Measurement (ESEM '10)*, 2010.

[13] L. Hatton, "Reexamining the Fault Density-Component Size Connection," *IEEE Softw.*, vol. 14, no. 2, pp. 89–97, Mar. 1997.

[14] J.M. Juran, *Quality Control Handbook*, New York: McGraw-Hill, 1974.

[15] N. Juristo and S. Vegas, "The Role of Non-Exact Replications in Software Engineering Experiments," *Empirical Softw. Eng.*, vol. 16, no. 3, pp. 295–324, June 2011.

[16] M. Kaâniche and K. Kanoun, "Reliability of a Commercial Telecommunications System," *Proc. 7th Internat. Symp. on Softw. Reliability Eng. (ISSRE '96)*, pp. 207–212, Oct./Nov. 1996.

[17] B.A. Kitchenham, "The Role of Replications in Empirical Software Engineering – a Word of Warning," *Empirical Softw. Eng.*, vol. 13, no. 2, pp. 219–221, Apr. 2008.

[18] A.G. Koru, K. El Emam, D. Zhang, H. Liu, and D. Mathew, "Theory of Relative Defect Proneness," *Empirical Softw. Eng.*, vol. 13, no. 5, pp. 473–498, Oct. 2008.

[19] A.G. Koru, D. Zhang, K. El Emam, and H. Liu, "An Investigation into the Functional Form of the Size-Defect Relationship for Software Modules," *IEEE Trans. Softw. Eng.*, vol. 35, no. 2, pp. 293–304, Mar. 2009.

[20] A.G. Koru, D. Zhang, and H. Liu, "Modeling the Effect of Size on Defect Proneness for Open-Source Software," *Proc. 3rd Internat. Workshop on Predictor Models in Softw. Eng. (PROMISE '07)*, *ICSE 2007 Companion*, pp. 115–124, May 2007.

[21] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings," *IEEE Trans. Softw. Eng.*, vol. 34, no. 4, pp. 485–496, July 2008.

[22] Y.K. Malaiya and J. Denton, "Module Size Distribution and Defect Density," *Proc. 11th Internat. Symp. on Softw. Reliability Eng. (ISSRE '00)*, pp. 62–71, Oct. 2000.

[23] T. Menzies, J. Greenwald, and A. Frank, "Data Mining Static Code Attributes to Learn Defect Predictors," *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 2–13, Jan. 2007.

[24] J. Miller, "Replicating Software Engineering Experiments: a Poisoned Chalice or the Holy Grail," *Inf. Softw. Technol.*, vol. 47, no. 4, pp. 233–244, Mar. 2005.

[25] A. Mockus, R.T. Fielding, and J.D. Herbsleb, "Two Case Studies of Open Source Software Development: Apache and Mozilla," *ACM Trans. Softw. Eng. Methodol.*, vol. 11, no. 3, pp. 309–346, July 2002.

[26] P. Mohagheghi and R. Conradi, "An Empirical Investigation of Software Reuse Benefits in a Large Telecom Product," *ACM Trans. Softw. Eng. Methodol.*, vol. 17, no. 3, pp. 13:1–13:31, June 2008.

[27] J.C. Munson and T.M. Khoshgoftaar, "The Detection of Fault-Prone Programs," *IEEE Trans. Softw. Eng.*, vol. 18, no. 5, pp. 423–433, May 1992.

[28] N. Ohlsson and H. Alberg, "Predicting Fault-Prone Software Modules in Telephone Switches," *IEEE Trans. Softw. Eng.*, vol. 22, no. 12, pp. 886–894, Dec. 1996.

[29] T.J. Ostrand and E.J. Weyuker, "The Distribution of Faults in a Large Industrial Software System," *Proc. ACM SIGSOFT Internat. Symp. on Softw. Testing and Analysis (ISSTA '02)*, pp. 55–64, July 2002.

[30] K. Petersen and C. Wohlin, "Context in Industrial Software Engineering Research," *Proc. 3rd Internat. Symp. on Empirical Softw. Eng. and Measurement (ESEM '09)*, pp. 401–404, Oct. 2009.

[31] L. M. Pickard, B. A. Kitchenham and P. Jones, "Combining Empirical Results in Software Engineering" *Information and Software Technology* vol. 40, no. 14, pp. 811–821, 1998.

[32] B. Robinson and P. Francis, "Improving Industrial Adoption of Software Engineering Research: a Comparison of Open and Closed Source Software," *Proc. ACM-IEEE Internat. Symp. on Empirical Softw. Eng. and Measurement (ESEM '10)*, pp. 21:1–21:10, Sept. 2010.

[33] P. Runeson, M.C. Ohlsson, and C. Wohlin, "A Classification Scheme for Studies on Fault-Prone Components," *Product-Focused Software Process Improvement (PROFES '01)*, *Lect. Notes in Computer Sci.*, vol. 2188, pp.341–355, Sept. 2001.

[34] P. Runeson, M. Höst, A. Rainer, and B. Regnell, *Case Study Research in Software Enginering: Guidelines and Examples*, Wiley, 2012.

[35] E. Shihab, Z.M. Jiang, W.M. Ibrahim, B. Adams, and A.E. Hassan, "Understanding the Impact of Code and Process Metrics on Post-Release Defects: a Case Study on the Eclipse Project," *Proc. ACM-IEEE Internat. Symp. on Empirical Softw. Eng. and Measurement (ESEM '10)*, pp. 4:1–4:10, Sept. 2010.

[36] F.J. Shull, J.C. Carver, S. Vegas, and N. Juristo, "The Role of Replications in Empirical Software Engineering," *Empirical Softw. Eng.*, vol. 13, no. 2, pp. 211–218, Apr. 2008.

[37] R.W. Selby and V.R. Basili, "Analyzing Error-Prone System Structure," *IEEE Trans. Softw. Eng.*, vol. 17, no. 2, pp. 141–152, Feb. 1991.

[38] R.S. Chhillar and Nisha, "Empirical Analysis of Object-Oriented Design Metrics for Predicting High, Medium and Low Severity Faults Using Mallows," *SIGSOFT Softw. Eng. Notes*, vol. 36, no. 6, pp. 1–9, Nov. 2011.

[39] B. Turhan, T. Menzies, A.B. Bener, and J. Di Stefano, "On the Relative Value of Cross-Company and Within-Company Data for Defect Prediction," *Empirical Softw. Eng.*, vol. 14, no. 5, pp. 540–578, Oct. 2009.

[40] S. Wu, Q. Wang, and Y. Yang, "Quantitative Analysis of Faults and Failures with Multiple Releases of Software," *Proc. 2nd ACM-IEEE Internat. Symp. on Empirical Softw. Eng. and Measurement (ESEM '08)*, pp. 198–205, Oct. 2008.

[41] H. Zhang, A. Nelson, and T. Menzies, "On the Value of Learning from Defect Dense Components for Software Defect Prediction," *Proc. 6th Internat. Conf. on Predictive Models in Softw. Eng. (PROMISE '10)*, pp. 14:1–14:9, Sept. 2010.

[42] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-Project Defect Prediction: a Large Scale Experiment on Data vs. Domain vs. Process," *Proc. 7th Joint Meeting European Softw. Eng. Conf. and the ACM SIGSOFT Symp. Foundations of Softw. Eng. (ESEC/FSE '09)*, pp. 91–100, Aug. 2009.

[43] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting Defects for Eclipse," *Proc. 3rd Internat. Workshop on Predictor Models in Softw. Eng. (PROMISE '07)*, pp. 9:1–9:7, May 2007.