A Replicated Quantitative Analysis of Fault Distributions in Complex Software Systems

Carina Andersson and Per Runeson, *Senior Member*, *IEEE*

Abstract—To contribute to the body of empirical research on fault distributions during development of complex software systems, a replication of a study of Fenton and Ohlsson is conducted. The hypotheses from the original study are investigated using data taken from an environment that differs in terms of system size, project duration, and programming language. We have investigated four sets of hypotheses on data from three successive telecommunications projects: 1) the Pareto principle, that is, a small number of modules contain a majority of the faults (in the replication, the Pareto principle is confirmed), 2) fault persistence between test phases (a high fault incidence in function testing is shown to imply the same in system testing, as well as prerelease versus postrelease fault incidence), 3) the relation between number of faults and lines of code (the size relation from the original study could be neither confirmed nor disproved in the replication), and 4) fault density similarities across test phases and projects (in the replication study, fault densities are confirmed to be similar across projects). Through this replication study, we have contributed to what is known on fault distributions, which seem to be stable across environments.

Index Terms—Empirical research, replication, software fault distributions.

1 Introduction

The analysis of fault distributions in the development of complex software systems is an area of interest in both practical and scientific terms. From a practical point of view, it can be applied to project and quality management. For the researcher, such analysis contributes to furthering our empirical knowledge in relation to software engineering, which is urgently needed for the development of the field. This paper reports on a quantitative analysis of fault distributions in three different development projects. The analysis, based on extensive data collection, provides empirical evidence and helps build empirical knowledge rather than rely on anecdotal or intuitive arguments.

Relatively few studies have empirically investigated quantitative issues with fault distributions in the development of complex software systems. Fenton and Ohlsson [8] published an elaborate quantitative analysis of fault distributions. They present a study examining faults from two consecutive releases of a telecommunication switching system. The study questions the conventional wisdom about fault distributions in the software engineering domain, and although very carefully conducted and reported, it is a single case study that cannot claim to be generally applicable. Replication is required in other environments to provide a better understanding of the general nature of the results.

Although empirical studies often mention the need for replications in their concluding chapters, few such replication studies are actually conducted. It has been advocated [24] that conducting or participating in families of research studies, rather than relying on single studies, would increase the reliability of results. By replicating a study, more generalizable results may be achieved rather than the isolated findings that come from a single study [15]. In the field of software engineering experimentation, replication studies exist (in the area of defect detection techniques [28], for example), but replicated case studies are rare [30]. In the few replication studies that have been conducted in the field, opinions differ as to whether replicated experiments should be similar or not. With respect to the reuse of experimental materials, Miller [17] states that "although, from a simple replication point of view, this seems attractive, from a meta-analysis point of view, this is undesirable, as it creates strong correlations between the two studies." Pickard et al., on the other hand, [25] state, "The greater the degree of similarity between the studies, the more confidence you can have in the results of a meta-analysis." Yin defines two kinds of case study replication: the case may be selected to predict similar results (literal replication) or to predict contrasting results but for predictable reasons (theoretical replication) [32, p. 47].

The goal of our study is to replicate Fenton and Ohlsson's study [8] and to analyze their hypotheses on a different type of system within a different development context in a literal replication. Given the nature of the phenomenon under investigation, the hypotheses are qualitatively analyzed rather than formally tested by using statistical hypothesis tests. The original hypotheses concern four areas, each of which is examined in this replication study. It is important to continue developing the body of research on these areas through further analysis. We would tend to side with Miller on how replication should be conducted [17], and the context of our study allowed us to examine a case with a different size, system type, and development process. Our replication is literal, as we set the

• *The authors are with the Department of Computer Science, Lund University, Box 118, SE-221 00 Lund, Sweden.*
E-mail: {carina.andersson, per.runeson}@telecom.lth.se.

Manuscript received 1 May 2006; revised 5 Dec. 2006; accepted 9 Feb. 2007; published online 23 Feb. 2007.
Recommended for acceptance by K. Kanoun.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-0099-0506.
Digital Object Identifier no. 10.1109/TSE.2007.1005.

same hypotheses despite these variations. Where possible, the measurements remain the same, although certain measurements are not identical, given the context of the study. In addition, it has been argued that deliberate variations in the conditions of the research study increase external validity and hence allow for a higher degree of generalization [15]. In conducting our replication of the Fenton and Ohlsson study, we extend the scope of the results to a new environment and investigate the range of conditions under which their findings hold.

The paper is organized as follows: Section 2 presents related work on fault distributions. The hypotheses from the original study of Fenton and Ohlsson are listed in Section 3. The investigated systems are presented in Section 4. Our results are presented in Section 5, together with discussions of the results from the original Fenton and Ohlsson study and also with related studies when applicable. Section 5.8 discusses the results, whereas Section 6 presents the conclusions that can be drawn from replicating the original study.

2 Background

Fenton and Ohlsson [8] investigated the distribution of faults within two releases of a system, focusing on four main hypotheses that were investigated though not formally tested. First, it was shown that some software modules have a greater concentration of faults than others, following the Pareto principle [13]. Second, they examined the persistence of faults across test phases. One hypothesis states that if many faults in a given module are detected by the function test, then that module will also have many faults in the system test. Strong support for this hypothesis would enable predictions of fault-prone modules. However, the original case study provides only limited support for this hypothesis. A second perspective on fault persistence in modules concerns whether many faults detected before system release would imply many postrelease faults in site tests and operation. This hypothesis is strongly rejected. The third area investigated by Fenton and Ohlsson is whether the size and complexity of each module have an impact on the number of faults and whether these parameters could function as predictors. The area is divided into several hypotheses, each of which looks more closely at size metrics as predictors for the absolute number of faults and fault density, whereas faults are separated into prerelease and postrelease faults. The hypothesis concerning complexity compares the prediction ability of complexity measures and size metrics. Neither hypothesis is strongly supported in the case study although there is limited support for size metrics as a predictor of the number of prerelease faults. The fourth area investigates fault densities in more detail by using a benchmarking strategy. It is shown that the fault densities of corresponding test phases are similar for the two releases of the system and that, to a certain extent, the fault densities are similar to the findings of other studies.

Fenton and Ohlsson argue that too few studies in this field have been published to enable proper investigation. Here, we contribute by expanding the available empirical research with findings from a study examining three software systems. The software systems are developed by an organization whose characteristics differ substantially from the organization studied by Fenton and Ohlsson, and the results, therefore, provide a new perspective on the hypotheses under investigation.

A similar study of Ostrand and Weyuker examines fault data from 13 releases of an inventory tracking system [21], written mostly in Java. Their study aims to find ways to identify fault-prone files, and the analysis is divided into four categories. First, the distribution of faults over software modules is examined. Here, strong evidence for the Pareto principle is found, stating that a small number of modules are responsible for a large share of faults. Second, the effect of the modules' size on fault density is examined. The results show no evidence to support the conventional wisdom that larger modules have higher fault densities. Third, fault persistence is examined. Unfortunately, no conclusions on the predictive ability of fault concentration during testing for faults after release could be drawn due to the low number of faults detected after release. Finally, the fault proneness of a newly written code compared to a code written earlier is examined. This analysis confirmed the intuitive "knowledge" that fault density for new modules is higher than for older ones. The study of Ostrand and Weyuker [21] is extended in a second study, in which the original data set of 13 releases is expanded to include four further releases [22]. The second study focuses on using previous releases to predict the number of faults that can be expected in each release and their location within the system.

In addition to the studies of Fenton and Ohlsson and of Ostrand and Weyuker, who take a broad view of quantitative issues, other studies investigating fault distributions have been conducted. However, most studies focus on only one or a small number of the issues on which Fenton and Ohlsson formulated their hypotheses. Two studies in this field were conducted more than 20 years ago. Adams' study [1], consisting of data from nine software products, shows that a small number of faults cause the most common failures. Only when these "most-often-failure-causing" faults are removed will reliability improve significantly. Basili and Perricone [3] analyzed fault data from a software project, starting its life cycle in the mid-1970s. The investigation focused on comparing fault distributions between modified and newly developed software modules.

In the 1990s and from 2000 onward, a second set of studies was published. Möller and Paulish report on the effects of modification and reuse on fault density in a control system developed in high-level programming languages and in assembly language [19]. They show the relationship between fault rates and programming languages, with more favorable results for high-level languages. Graves et al. also investigated the change history to predict fault rates [10] (in a subsystem of a telephone system) and found that the number of times that a module changes acts as an effective predictor. On the other hand, the size and complexity of the modules were poor predictors of faults. Pfleeger and Hatton present data on the number of changes made in an air-traffic-control information system

project [23]. In addition, they investigated the fault densities of the system before and after delivery. The analysis shows that as many as 10 times the number of faults was reported before release as compared to after. A number of studies have examined the relation between module size and fault density [3], [19], [31]. Hatton has reviewed a number of studies, focusing on the size of the components and proportionality to reliability [11]. The results of the studies, including [3] and [19], imply that decomposing a system into small components does not increase reliability. The opposite is argued by El Emam et al., who, in a study of object-oriented systems, showed that as class size increases, so will fault proneness [6]. Denaro and Pezzé [7] used the code and fault database of the Apache Web server to investigate module fault proneness. This open source project also found that a small number of modules account for a large number of faults.

3 Study Design

3.1 Hypotheses

Fenton and Ohlsson listed a number of hypotheses and empirically analyzed whether these hypotheses could be supported or rejected. The four categories of questions addressed are listed below, each with formulated hypotheses. The hypotheses remain the same in the replication study to provide results that can be combined and compared to the results from the original study:

- The Pareto principle of fault distribution: hypotheses relating to how faults are distributed over different modules and the conventional wisdom that a small percentage of modules will contain a large percentage of faults detected.

  1a. A small number of modules contain most of the faults detected during prerelease testing.
  1b. If a small number of modules contain most of the prerelease 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 postrelease testing.
  2b. If a small number of modules contain most of the postrelease faults, then it is because these modules constitute most of the code size.

- Persistence of faults: hypotheses relating to how modules with a high fault concentration in early testing tend to have a high fault concentration in later testing.

  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 prerelease testing implies a higher incidence of faults in postrelease testing and use.

- Effects of module size and complexity on fault proneness: hypotheses about metrics potentially suitable for fault prediction.

  5. Simple size metrics such as Lines of Code (LOC) are good predictors of fault and failure prone modules.
  6. Complexity metrics are better predictors than simple size metrics of fault and failure-prone modules (due to lack of data, this hypothesis is not included in our replication study).

- Quality in terms of fault densities: hypotheses relating to fault densities separated for each test phase, and prerelease and postrelease testing.

  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 (benchmarking figures from other studies).

We have addressed these hypotheses and analyzed them by using new empirical data. The data originate from three software systems and not from successive releases of the same system, as in the original study by Fenton and Ohlsson. However, the systems investigated in the replication study relate to each other in a product-line context and thus have some common components.

Our analyses of the hypotheses differ slightly from the original study. We have applied statistical data analysis techniques, whereas the original study relied primarily on descriptive statistics. Hypothesis 6, concerning whether complexity metrics are useful as predictors of fault proneness, is not discussed in this study, as we did not have access to data on the complexity of each module. Hypothesis 7 concerns fault densities in subsequent releases of a system. As we have one single release of each system, the hypothesis is not comparable between the studies. However, the systems are similar, and it is our opinion that the hypothesis concerning fault density of software systems developed subsequently is of equal interest.

3.2 Data Analysis

The primary method used for data analysis is graphical techniques. Scatter plots are used to show the variation between specific variables for the modules involved. The Alberg diagram [20], suggested by Fenton and Ohlsson, is used to order the modules with respect to the variable concerned. In addition to the graphical technique used by Fenton and Ohlsson, we have applied statistical data analysis. To test whether two sets of data are related and, if so, their degree of linear dependency, the Pearson product-moment correlation coefficient [9] is used.

Even though a statistical analysis is performed within the case study, we do not generalize to a population through statistical inference but to a theoretical proposition through analytical generalization [32, p. 10]. Any case to be studied is selected with a specific purpose in mind rather than with a random sample from a population. In this study, we have chosen replication cases, which we expect will give similar results (although some key characteristics are different). The collected data originate from a single organization; thus, the statistical inference may be generalized to that

TABLE 1
Distribution of Modules by Size

Original study			Replication study			
LOC	Release *n*	Release *n+1*	LOC	Project 1	Project 2	Project 3
<1000	23	26	<10 000	12	30	25
1001-2000	58	85	10 001-30 000	14	28	29
2001-3000	37	73	30 001-50 000	8	14	16
3001-4000	15	38	50 001-70 000	6	6	3
4001-5000	6	16	70 001-90 000	1	3	4
5001-6000	0	6	>90 000	4	9	13
>6000	1	2				
Total	140	246	Total	45	90	90

organization but not to a wider community. However, when the results are combined with findings from other studies we may gradually find much stronger empirical evidence for the theoretical propositions.

4 System Description

The hypotheses are investigated using empirical data from a large telecommunications company. The organization develops consumer products, with a substantial proportion of its functionality implemented in software. The development process is characterized by iterative development, with component releases in small iterations, and product line architecture [4]. The products are implemented on a hardware and software platform delivered by a subcontractor. Within the organization, a software platform is shared among different products, thus following a product line concept. Software modules may appear in different products: either as general components or as versions of the component tailored to the specific product. Although the organization has several development sites worldwide, the development carried out in the chosen projects are localized to one site. ST is partly carried out at a different site.

We analyzed data from three separate software development projects. Two of the projects are application projects developing consumer products. One project is an internal platform project where the final product is used as a platform in other consumer products. We refer to the projects as *Project 1*, *Project 2*, and *Project 3*. Projects 1 and 3 are the application-type projects, whereas Project 2 is the platform project. A project runs for approximately one year. The data from Project 1 were collected retrospectively from the problem-reporting database, whereas the data for Projects 2 and 3 were collected while the projects were running. Each project consists of 10-12 groups of developers, where each group has a responsibility for a specific set of features in the system. A feature is implemented in one to several modules.

Much of the code is written in C, but a minor part of the modules contain files written in Java. A subset of modules in each project is included in the analysis: 45, 90, and 90 modules, respectively, from the three projects (approximately half of the total number of modules in each project). The samples are limited to the modules for which data could be collected automatically, such as LOC. The subset includes modules from each existing group of developers. For our purposes, we refer to a module as a collection of several files, and compared to the original study, the largest modules in the replication study are more than 10 times the size of those in the original study (see Table 1). Regarding system size, the modules involved contain a total of roughly 1,500, 2,500, and 3,500 LOC, respectively, for the three projects. The distribution of modules by size is presented in Table 1.

As mentioned in Section 3, the complexity measure of each module was not available for analysis.

As in the original study, the dependent variable is the number of faults. Each fault is located in a module; that is, if the cause of a detected failure was corrected by modifying n modules, then it is counted as n distinct faults. Change proposals are filed as change requests and not as failures and hence excluded from the analysis. Duplicate failures that are detected more than once were also excluded. More information on the data collection process can be found in [2].

The total number of faults is in the same range as in the original study (see Table 2). The percentage distributions of the number of faults detected by each test activity are given in Table 3. For *release* $n+1$, the percentage distributions of faults show similarities to the distributions investigated in the replication study. In the replication study, data are collected during the function test, the system test, the customer-acceptance test, and the first couple of months of operation. Faults detected during the function test and the system test are, as in the original study, grouped as prerelease faults, whereas faults detected during the customer-acceptance test and the first months of operation are combined as postrelease faults. This latter group of faults is not separable in this study, as in the study of Fenton and Ohlsson, which combined faults from the "first

TABLE 2
Project Characteristics

		Project type	Sample size (# modules)	# faults
Original study	Release *n*	Application	140	1669
	Release *n+1*		246	3646
Replication study	Project 1	Application	45	1558
	Project 2	Platform	90	4045
	Project 3	Application	90	2419

TABLE 3
Percentage Distribution of Faults per Testing Activity

		Prerelease faults		Postrelease faults
		Function test	System test	Site and Operation
Original Study	Release n	55%	41%	4%
	Release n+1	63%	28%	9%
Replication study	Project 1	72%	23%	5%
	Project 2	67%	29%	4%
	Project 3	62%	28%	10%

26 weeks of site tests" and "approximately the first year of operation after the site tests."

5 Data Analysis and Results

In this section, we analyze each hypothesis and compare the results to related work. The original study of Fenton and Ohlsson is primarily used for comparison, but other studies are referred to where appropriate.

5.1 Pareto Principle Regarding Prerelease Faults

The first category of hypotheses deals with the theory that a small number of modules in a system contain a substantial part of the faults. The Pareto principle was originally formulated by the Italian economist Vilfredo Pareto [26] and was later generalized by Juran [13], suggesting that most of the results in any context are determined by a small number of causes. The principle has been applied to various contexts and is often referred to as the 20-80 rule, in this context suggesting that 20 percent of the modules contain 80 percent of the faults. We, however, are investigating the general principle and do not hypothesize any specific percentage distribution.

Four hypotheses concerning the Pareto principle have been examined in this study, two of which concern prerelease faults.

Hypothesis 1a. *A small number of modules contain the faults that cause most of the failures discovered during prerelease testing.*

The first hypothesis focuses on prerelease faults, that is, faults detected during the function test and the system test. Fig. 1 shows the percentage of the accumulated number of faults when the modules are ordered by decreasing number of faults. Fig. 1 indicates that 20 percent of the modules in Project 1 are responsible for 63 percent of the faults discovered during prerelease testing. For Project 2 and Project 3, the results are almost identical: 20 percent of the modules are responsible for 70 percent of the faults identified during prerelease testing.

These findings are somewhat stronger than those reported in the original study (20 percent, 60 percent) [8], as well as in the studies of Kaaniche and Kanoun (38 percent, 80 percent) [14], Denaro and Pezzè (20 percent, 62 percent) [7], and Munson and Khoshgoftaar (20 percent, 65 percent) [18]. Still, some studies reveal an even stronger

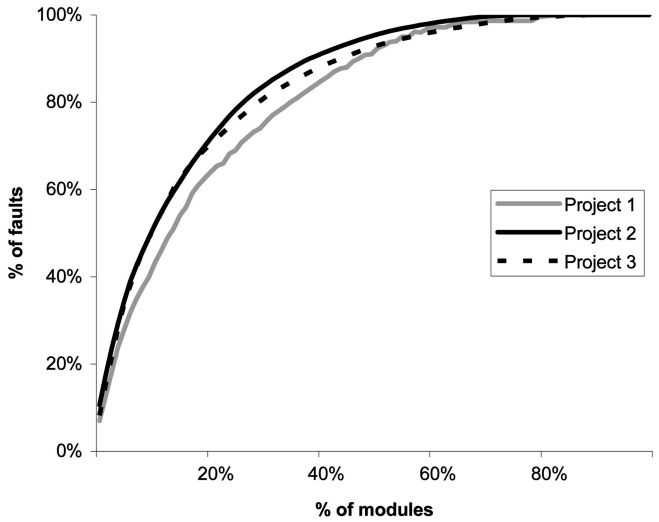

Fig. 1. Alberg diagram showing the percentage of modules versus the percentage of prerelease faults for the projects.

focus on fewer modules. Compton and Withrow found a 12 percent, 75 percent ratio in their study [5]. Hence, our results are within the range of what has been observed before.

Hypothesis 1b. *If a small number of modules contain most of the faults discovered during prerelease testing, then this is simply because those modules constitute most of the code size.*

As discussed in the original study, strong evidence for Hypothesis 1a might be explained if those modules containing a majority of the faults constitute a large share of the total system size. This was also the case in the Compton and Withrow study, where 12 percent of the modules accounted for 63 percent of the LOC [5].

As in the case in the original study, we found no evidence to support this hypothesis (see Table 4). In Project 1, the 20 percent of the modules responsible for 63 percent of the faults made up 38 percent of the code, whereas in Project 2, the 20 percent of the modules responsible for 70 percent of the faults made up 25 percent of the code. In Project 3, the 20 percent of the modules responsible for 70 percent of the faults made up 39 percent of the code.

The results for Hypotheses 1a and 1b show that we have a large proportion of faults in a smaller proportion of the modules. The difference is not that considerable when comparing the proportion of faults to its share of the code base. Still, from a practical perspective, the first hypothesis is significant in that it indicates that there is a set of modules that is more fault prone than the rest.

5.2 Pareto Principle Regarding Postrelease Faults

Hypothesis 2 concerns postrelease faults, that is, faults detected in the customer-acceptance test and during the first months of operation, suggesting that the Pareto principle also holds for these faults.

Hypothesis 2a. *A small number of modules contain the faults that cause the most postrelease failures.*

TABLE 4
Percentage Distribution of Prerelease Faults over Components Related to Size

		Share of modules	Share of prerelease faults	Share of system size
Original Study	Release n	20%	60%	30%
	Release n+1		"almost identical to release n"	
Replication study	Project 1	20%	63%	38%
	Project 2	20%	70%	25%
	Project 3	20%	70%	39%

Fig. 2 illustrates for the three projects the percentage of accumulated number of postrelease faults when the modules are sorted in decreasing order with respect to fault content. In Project 1, 20 percent of the modules are responsible for 87 percent of the postrelease detected faults, and the findings for Project 2 are almost identical. In Project 3, 20 percent of the modules are responsible for 80 percent of the postrelease faults. Compared to Hypothesis 1a, we have even stronger evidence to support the hypothesis, given the large number of postrelease faults in a small proportion of modules.

In the original study, Fenton and Ohlsson examined the proportion of postrelease faults contained in 10 percent of the modules. The original data and the replicated data are shown in Table 5.

The original study provided evidence for this hypothesis. Ten percent of the modules are responsible for 100 percent and 80 percent of the postrelease faults for the two releases, respectively. In our study, the postrelease faults are more unevenly distributed over the system. Furthermore, in our study, the customer-acceptance test faults and only the first months of operation belong to this category, whereas in the original study, the postrelease faults include one year of operation. Thus, our observation is more limited than that of the original study.

Hypothesis 2b. *If a small number of modules contain most of the postrelease faults, then this is simply because those modules constitute most of the code size.*

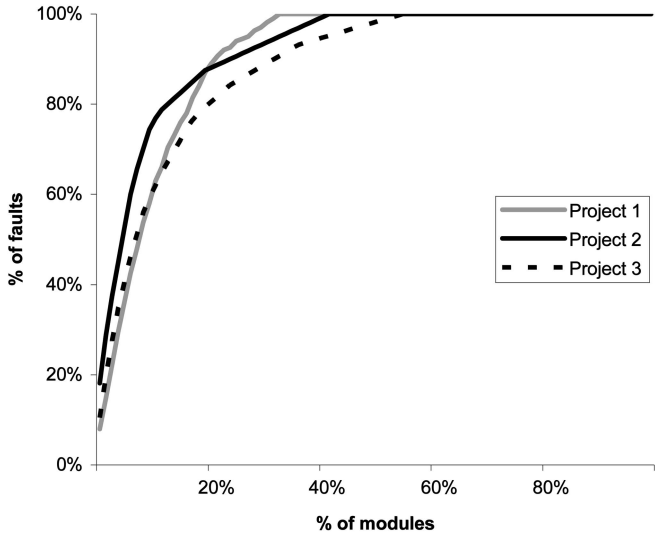

Fig. 2. Alberg diagram showing the percentage of modules versus the percentage of postrelease faults for the projects.

Since we found evidence for Hypothesis 2a, it is worthwhile investigating Hypothesis 2b. As in Hypothesis 1b, the support of Hypothesis 2a could be explained if those modules containing a majority of the postrelease faults constitute a large amount of the total system size. However, we did not find evidence for this (see Table 5).

In contrast to the original study, we did not find strong evidence in favor of the converse hypothesis, that is, that most operational failures are caused by faults in a small proportion of the code. In the replicated study, 100 percent of the postrelease faults were contained in modules that make up 32 percent, 41 percent, and 70 percent of the system for the three projects, respectively. To summarize, the original study showed strong evidence for the converse hypothesis, whereas the replication study did not (see Table 6).

5.3 High Incidence of Faults in FT Implies High Incidence of Faults in ST

It has been observed that modules with a high fault incidence in the early phases of testing imply a high fault incidence in later phases of testing. The hypothesis thus suggests that a high detection rate of faults in the function test in a particular module will result in a high detection rate of faults in the system test in the same module.

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

This hypothesis is investigated from a number of different perspectives. Fig. 3 illustrates the relationship between the number of faults detected in the function test

TABLE 5
Percentage Distributions of Postrelease Faults over Components

		Share of modules	Share of faults	Share of system size
Original Study	Release n	10%	100%	12%
	Release n+1	10%	80%	-
Replication study	Project 1	10%	63%	19%
	Project 2	10%	74%	10%
	Project 3	10%	59%	27%
	Project 1	20%	87%	24%
	Project 2	20%	88%	22%
	Project 3	20%	80%	40%

TABLE 6
Percentage Distributions of Postrelease Faults Related to Size

		Share of post-release faults	Share of system size
Original Study	Release n	100%	12%
	Release n+1	78%	10%
Replication study	Project 1	100%	32%
	Project 2	100%	41%
	Project 3	100%	70%

and the system test. Each dot represents a module. Statistical analysis shows a correlation between the number of faults in the function test and the system test. The Pearson correlation coefficient r is used as a measure of association.

The correlation between the number of faults detected in the function test and the number of faults detected in the system test indicates that the most fault-prone modules in the early phase of testing will be fault prone when tested in the system test phase. In the analysis, the modules are ordered by decreasing number of faults in the system test and show that for Project 1, 50 percent of the faults detected in the system test occurred in modules that were responsible for 40 percent of the faults detected in the function test. For Project 2, 50 percent of the faults detected in the system test occurred in modules that were responsible for 39 percent of the faults detected in the function test. For Project 3, 50 percent of the faults detected in the system test occurred in modules that were responsible for 38 percent of the faults detected in the function test. Fig. 4 shows that for each project, these 50 percent of the system test faults are contained in a small number of modules. The corresponding numbers in the original study are that 50 percent of the faults detected in the system test occurred in modules responsible for 37 percent and 25 percent of the faults detected in the function test, respectively, for *releases* n and $n+1$.

This information can be used to predict fault contents over modules. In an ideal situation, the most fault-prone modules in the function test would also be the most fault prone in the system test. If so, the two lines in Fig. 4 would overlap. As evident in Table 7, the overlap is not complete, but for Project 2, up to a certain level, most of the modules are fault prone both in the function test and the system test. Correlation analysis can explain these results, where Project 2 had a particularly high correlation between the number of faults in the function test and the number of faults in the system test. The systematic difference that we can see between the three projects is that the platform project (Project 2) has a higher fault density than the application projects (Projects 1 and 3).

The original study did not present any correlation analysis for this hypothesis. In addition, the proportions were less coherent than in our study; hence, no significant conclusions could be drawn from the data. We interpret the replication data as showing support for the hypothesis to which the findings of Yu et al. [33] lend further support.

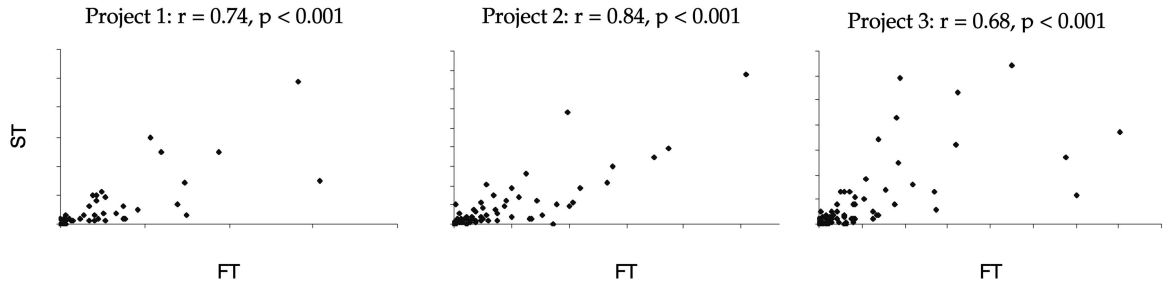

Fig. 3. Scatter plot showing relationship between faults detected in the function test and faults detected in the system test.

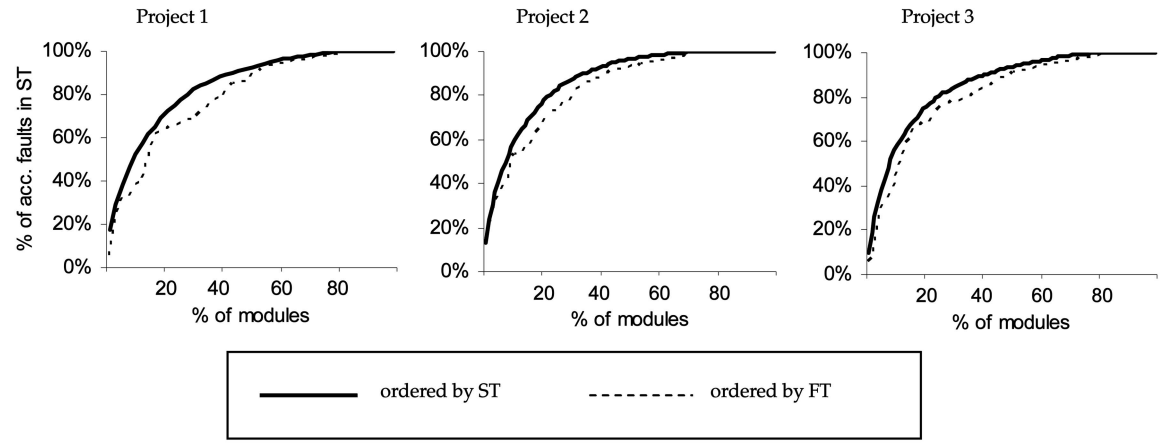

Fig. 4. Accumulated percentage of the number of faults in the system test when modules are ordered with respect to the number of faults in the system test versus the function test.

TABLE 7
Faults in the System Test for 10 Percent of the Most Fault-Prone Modules When Ordered with Respect to Fault Proneness in the System Test and the Function Test, Respectively

		ST	FT
Original study	Release n	38 %	17 %
	Release n+1	46%	24%
Replication study	Project 1	53 %	39 %
	Project 2	56 %	52 %
	Project 3	56 %	39 %

5.4 High Incidence of Prerelease Faults Implies High Incidence of Postrelease Faults

The belief that modules that are fault prone before release will also have a high incidence of faults after release works as a rationale for the fourth hypothesis.

Hypothesis 4. *A higher incidence of faults in prerelease testing implies a higher incidence of failures in postrelease.*

Fig. 5 illustrates the relationship between the number of prerelease and postrelease faults. Statistical analysis indicates a moderate correlation between the two parameters (see Fig. 5). In contrast to Fenton and Ohlsson who found support for the converse hypothesis, our findings support the original hypothesis. Fenton and Ohlsson showed that almost all of the faults detected in prerelease testing appear in modules that subsequently had no postrelease faults. They observe that 93 percent and 77 percent of the faults in prerelease testing occurred in modules that had no postrelease faults for the two releases, respectively. On the other hand, in the replication study, 36 percent, 29 percent, and only 13 percent of the faults in prerelease testing occur in modules that have no subsequent postrelease faults for the three projects, respectively.

The original study therefore rejects the hypothesis, although noting that the result is remarkable. Compton and Withrow [5], however, found many more postrelease faults in modules that also had prerelease faults. It is clear from the contradictory results that there are factors at work that are not measured in these case studies. For example, if the prerelease testing is sufficiently thorough for certain modules, then one can expect that they will not contribute to postrelease faults; if the prerelease testing is not sufficiently thorough, then postrelease faults may be found in these components.

TABLE 8
For Each Hypothesis: Correlation Coefficients between Module Size and Various Fault Count Measures

	5a. Total number of faults	5b. Pre-release faults	5c. Post-release faults	5d. Pre-release fault-density	5e. Post-release fault-density
Project 1	0.38	0.37	0.44	-0.17	-0.15
Project 2	0.05	0.05	0.08	-0.22	-0.16
Project 3	0.62	0.60	0.65	-0.10	-0.02

5.5 Size Metrics as Fault Predictors

In this section, we will focus on hypotheses concerning the use of size metrics to predict which modules will be fault prone. The proportions of the newly developed code in the three projects are not identical: The system developed in Project 2 contains a larger amount of newly developed or modified modules than the other projects. Unfortunately, an appropriate measure of the amount of modified code was not accessible, since the change history includes modified code in terms of added, changed, and deleted. Thus, we use the total number of LOC for the analyses. The analysis of the effect of module size is separated into five different closely related hypotheses that address different aspects of module size effect.

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

A number of studies have investigated the relationship between module size and the number of faults and have produced counterintuitive results. Conventional wisdom says that smaller modules should result in fewer faults since small modules with less LOC are easier to develop. On the other hand, a large amount of small modules will result in more interface faults spread across the modules.

To analyze Hypothesis 5a, we have correlated the size of the modules measured in LOC to the total number of faults. Column 2 in Table 8 gives the measures of association for the investigated projects.

In the original study, Hypothesis 5a was investigated by showing the number of modules that had a certain number of faults [8]. As in the study by Basili and Perricone [3], a

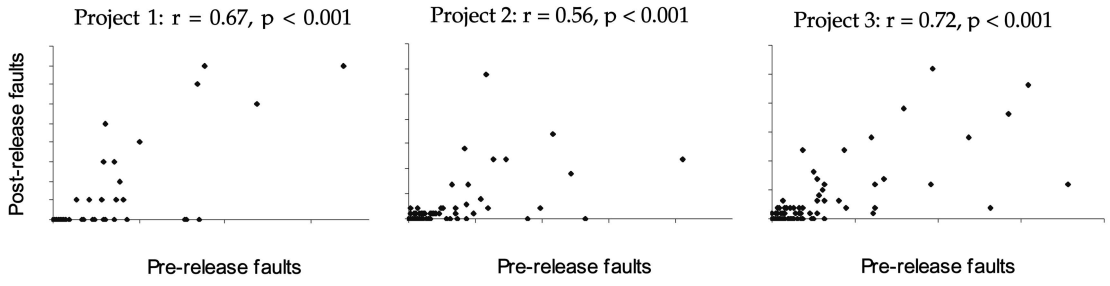

Fig. 5. Scatter plot showing relationship between prerelease faults and postrelease faults.

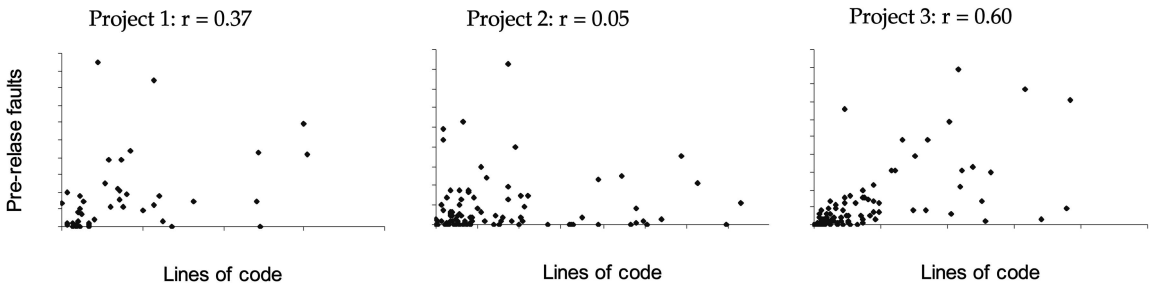

Fig. 6. Scatter plot showing relationship between LOC and prerelease faults.

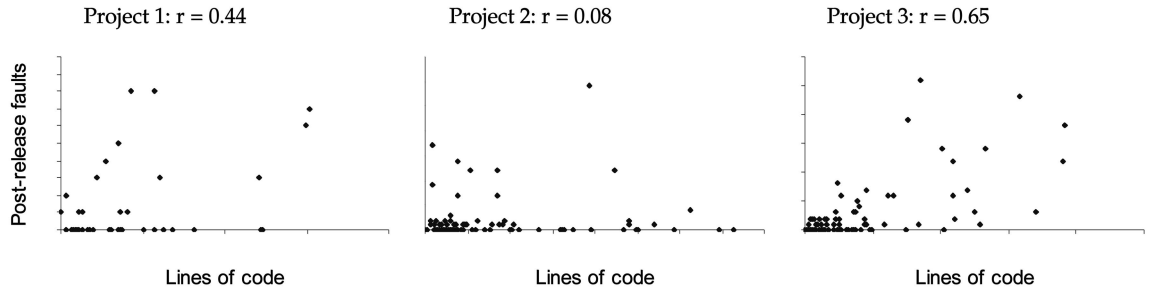

Fig. 7. Scatter plot showing relationship between LOC and postrelease faults.

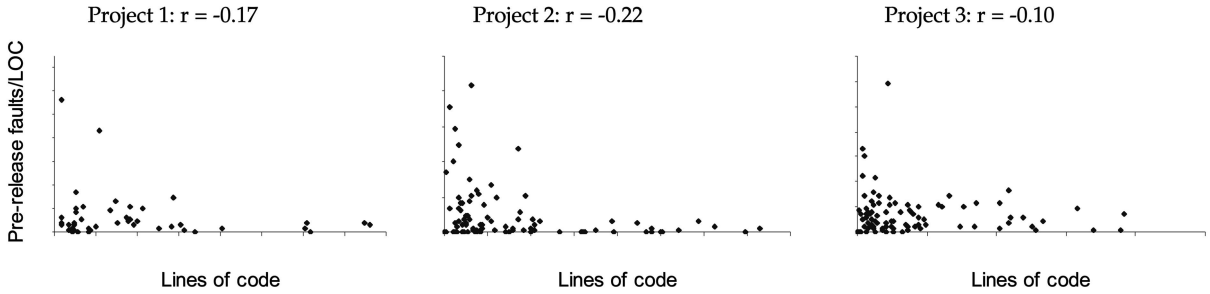

Fig. 8. Scatter plot showing relationship between prerelease fault density and size.

table is used that groups modules according to frequency of faults. Compared to the results in these studies, the data sets in the replication study have a lower proportion of modules with few faults. Although Fenton and Ohlsson's study did not find evidence for the hypothesis, we have found some support although it is not consistent in all the projects involved. Some correlation exists between the total number of faults and the total LOC for Project 3, whereas the correlation coefficients are low for Project 1 and Project 2.

Hypothesis 5b. *Size metrics are good predictors of prerelease faults in a module.*

Fig. 6 illustrates the relationship between module size in the total LOC and the number of faults detected in prerelease testing for the projects. The correlation of the relationship shows a very low value for Project 2. On the other hand, the measure of association for Project 3 gives moderate support for the hypothesis.

This is the single hypothesis related to size metrics for which Fenton and Ohlsson observed some, albeit limited, support in their scatter plots.

Hypothesis 5c. *Size metrics are good predictors of postrelease faults in a module.*

Fig. 7 illustrates the relationship between module size in the total LOC and the number of faults detected in postrelease testing for the projects. As in the case for Hypothesis 5b, the measure of association for Project 2 is very low, showing no evidence to support the hypothesis, whereas the measure of association for Project 3 provides limited support. In the original study, Fenton and Ohlsson did not find any evidence to support the hypothesis.

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

Studying fault density and module size can easily be misleading. The relationship between a variable X and $1/X$ will always be negative, as may be the case between size and fault density [29]. The relationships between size and the number of faults, which were analyzed in Hypotheses 5b and 5c, are more relevant, but for the purposes of replication, we present the analysis of Hypotheses 5d and 5e, although the analysis presents a significant methodological threat.

Fig. 8 illustrates the relationship between the total LOC and prerelease fault density of each module for the projects. The correlation is, as expected, negative. This suggests an inverse relationship between fault density and LOC; that is, we have higher fault densities for smaller modules. However, the measures of association for the projects have low values and do not provide any support for the hypothesis.

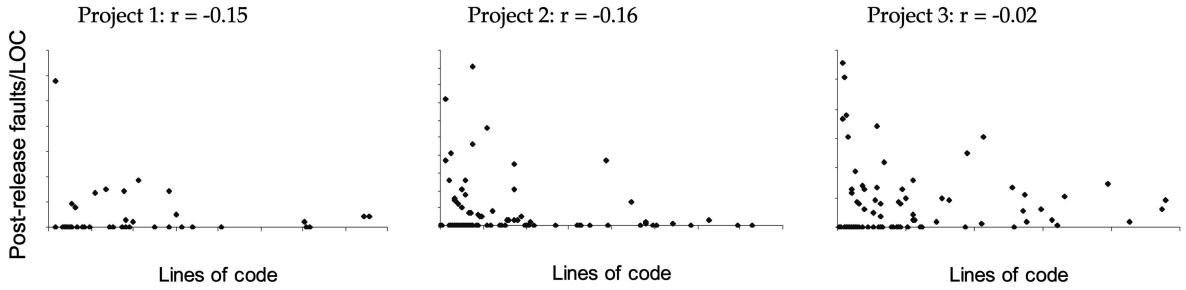

Fig. 9. Scatter plot showing relationship between module postrelease fault density and size.

In the original study, when examining the relationship between fault density and module size, Fenton and Ohlsson did not find any trend at all. The results differ from several other studies, where a relation between fault density and module size is shown. Hatton discusses an optimal module size with higher fault densities for both the smallest and the larger components [11], whereas Möller and Paulish [19], and Ostrand and Weyuker [21] only show that smaller modules have a higher fault density. On the other hand, Fenton and Ohlsson argue that the analysis conducted in some studies is misleading because of the data grouping. By analyzing simple scatter plots instead of comparing average figures for different size intervals, Fenton and Ohlsson do not find any evidence of a relationship between fault density and module size. They considered their results as a confirmation that there is no causal relationship between size and fault density.

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

In Hypothesis 5d, the relationship between module size and fault density was investigated. We shall now consider the relationship with respect to postrelease faults rather than prerelease faults. Fig. 9 illustrates scatter plots for LOC versus postrelease fault density for the projects. As in the case in the previous hypothesis, the measures of association are negative, suggesting that smaller modules have a higher fault density for postrelease faults. However, the values are very low and do not provide any evidence to support the hypothesis. Methodological threats are also inherent in this analysis.

In the original study, no support for the hypothesis was found.

In addition to the above hypotheses concerning size metrics and fault proneness, the ranking ability of LOC is assessed. Fenton and Ohlsson concluded that the LOC metric worked rather well at predicting the most fault-prone modules despite the fact that the analysis did not provide any evidence to support the hypotheses. Our results do not conclusively indicate that LOC works as a predictor for ranking the most fault-prone modules. The Alberg diagrams in Fig. 10 illustrate the LOC ranking ability for the projects. In Project 1, 20 percent of the largest modules are responsible for 26 percent of all faults, whereas in Project 2, 20 percent of the largest modules are responsible for 18 percent of all faults. These two projects do not indicate any particularly strong ranking ability for LOC. However, in Project 3, 20 percent of the largest modules are responsible for 57 percent of all faults, which is a significant portion of the detected faults. The diagrams for Projects 1 and 2 do not indicate strong predictability, and LOC works less well at ranking the most fault-prone modules than in the original study of Fenton and Ohlsson. On the other hand, for the third project, a number of modules are evidently more fault prone than the rest, which indicates a better ranking ability on fault proneness than results from the original study.

Summarizing the analysis of size metrics as a predictor for fault proneness, the results vary, which might be due to

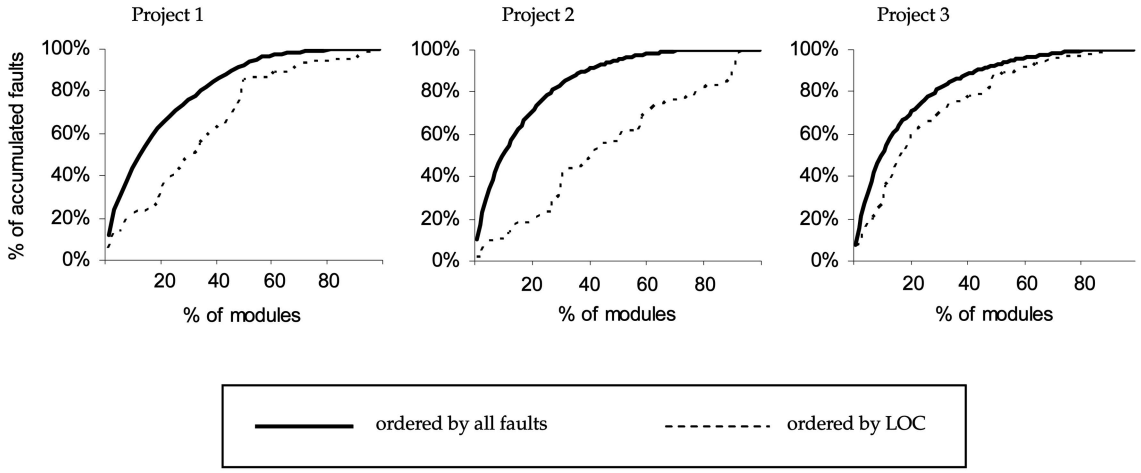

Fig. 10. Accumulated percentage of number of faults when modules are ordered with respect to LOC.

TABLE 9
Fault Densities for the Testing Activities (Faults/Thousand LOC)

	FT	ST	CAT
Project 1	0.7	0.2	0.03
Project 2	1.2	0.5	0.07
Project 3	0.4	0.2	0.07

the variety of conditions. For example, Project 2 is a large platform project not resulting in a final end product as the application projects do. For Project 2, a large proportion of the code is newly developed in comparison to Projects 1 and 3, where more code are reused.

5.6 Fault Densities at Corresponding Test Phases

The purpose of Hypotheses 7 and 8 is to visualize the range of fault densities that could be expected in software development and testing and to foster consensus as to what fault density can be expected under certain circumstances by considering such variables as development environment, testing activities, and programming languages.

We have not investigated the hypotheses on subsequent releases, as done in the original study, but we investigated on the three subsequent projects, each of which delivers a separate system. However, we considered an evaluation of the hypotheses as significant, since the projects are all executed in the same environment over a similar period of time, and they share software components although the ratio of common code is unknown.

Hypothesis 7. *Fault densities at corresponding phases of testing and operation remain roughly constant between subsequent projects conducted in the same context.*

The results from this study show that fault densities are in the same order of magnitude for the same test activities in the three projects (see Table 9). Project 2 has higher fault densities for the function test and the system test activities, whereas Projects 1 and 3 have similar fault densities for these activities. The results indicate that the development process used in the investigated organization is stable and consistent.

Nevertheless, the fault densities for the test activities for Project 2, which differ from the other projects, could be analyzed further. The project as such is larger than Project 1 but smaller than Project 3 in terms of the total LOC. However, the proportion of the newly developed code is larger in Project 2 than in the other two projects. This probably resulted in a higher number of injected faults.

As shown in Section 4 and Table 3, the test activities have similar distributions for fault detection, regardless of other project-specific characteristics such as size, duration, and number of detected faults. Although test effort, evidently an important parameter in this subject, is excluded from the analysis, the distributions in Table 3 present a view of the test activities' detection capacities, which also suggests that the development process is stable.

The original study provides little support for the hypothesis when investigating fault densities, with only one data point as evidence.

TABLE 10
Fault Densities for Prerelease and Postrelease Faults (Faults/Thousand LOC)

		Pre-release	Post-release	All
Original study	Release n	6.09	0.27	6.36
	Release n+1	5.97	0.63	6.60
Replication study	Project 1	0.9	0.03	0.93
	Project 2	1.7	0.07	1.77
	Project 3	0.6	0.07	0.67

5.7 Fault Densities of Systems from a Similar Environment

Hypothesis 8. *Software systems produced in a similar environment have broadly similar fault densities at similar testing and operation phases.*

In the original study, for Hypothesis 8, Fenton and Ohlsson compared their observed fault densities for prelease and postrelease activities with other published data. They suggested that "software systems developed in a similar environment have broadly similar fault densities at similar testing and operational phases." To that analysis, we add the data for our three projects, as shown in Table 10. It should be noted that a comparison of the absolute figures is not revealing, as different programming languages are used. When comparing the result of the three projects in the case study, we can see that the prerelease fault densities are an order of magnitude higher than postrelease densities, which reflects similar results in the original study and related work [23]. Potential confirmation of the hypothesis depends on what is meant by "similar." Among the three studied projects, we have identified larger dissimilarities between the application projects and the platform project, whereas the two application projects are more closely comparable. Thus, to some extent, the general conclusions drawn in the original study and related work [23] are supported by the higher ratio of prerelease to postrelease fault densities.

5.8 Discussion

In this section, we summarize and interpret the results of the hypotheses under study. The results are reported in Table 11 and commented on below.

It should be noted that confirmed hypotheses where correlations between two metric values are found do not imply causal relationships. The results as such do not explain the observed behavior. For example, detecting a high number of faults could occur simply because these modules have been tested more thoroughly and may not be dependent on the various metrics included in the analysis.

The results regarding the Pareto principle for fault distributions over modules are, to a large extent, the same in the replicated study as in the original study. A small set of modules contains a majority of the faults, both regarding prerelease and postrelease faults. This is in line with other empirical studies [5], [7], [14], [18], [21], although the exact proportions differ from study to study. This result does not occur because these few modules are larger than other

TABLE 11
Summary of Hypotheses in the Original Study [8] (Table 7), This Replication, and Other Empirical Studies

	Hypothesis	Original study	Replication	Other
Pareto principle	1a. Few modules contain most faults (prerelease)	Confirmed (20-60)	Confirmed (20-63; 20-70; 20-70)	(12-75) [5]; (20-62) [7]; (38-80) [14]; (20-65) [18]; Support [21]
	1b. Few faulty modules constitute most of the size (prerelease)	No support (20-30)	No support (20-38; 20-25; 20-39)	(12-63) [5] No support [21]
	2a. Few modules contain most faults (postrelease)	Confirmed (10-80; 10-100)	Confirmed (10-63; 10-74; 10-59)	(38-54) [14]; Support [21]
	2b. Few faulty modules constitute most of the size (postrelease)	No support; strong evidence of a converse hypothesis (100-12; 60-6)	No support (100-32; 100-41; 100-70)	
Faults early-late	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)	Support [33]
	4. High fault incidence prerelease implies the same postrelease	No support – strongly rejected (93-0; 77-0)	Support (36-0; 29-0; 13-0)	Support [5] No support [21]
Faults vs. size	5a. LOC is a good predictor of faults	No support	Varying support	Support converse hypothesis [3]
	5b. LOC is a good predictor of prerelease faults	Limited support	Varying support	
	5c. LOC is a good predictor of postrelease faults	No support	Varying support	
	5d. LOC is a good predictor of prerelease fault density	No support	No support	Confirmed [11], [19], [21],
	5e. LOC is a good predictor of postrelease fault density	No support	No support	
Complexity metrics	6. Complexity metrics are good predictors of faults	No (for cyclomatic complexity), some weak support for specific metric	N/A	
Fault densities	7. Fault densities are constant between releases or projects	Limited support	Support	
	8. Fault densities are similar in similar environments	Confirmed	Confirmed	Confirmed [23]

modules. The original study found support for the converse hypothesis (that is, small modules are more fault prone), but this is not the case in our study. The replication gives additional support to the validity of the Pareto principle for fault distribution over modules.

The next two hypotheses state that a high fault incidence in a module in the function test implies the same in the system test, and a high prerelease fault incidence implies a high postrelease fault incidence. The original study found limited support for the first hypothesis, and no support for the latter. The original report comments on their results as being "remarkable" [8]. In the replication study, we found support for both hypotheses. We found the results of the replication study in line with other studies [5], [33]. One reason for the observation is that there are always more faults to find: Some modules are more central from an architectural point of view, and these modules contain faults during their whole life cycle.

There was no support for using LOC as a predictor for faults in the original study. Only for prerelease faults did they find limited support for a relation between the number of faults and LOC. In the replication study, we find somewhat stronger support, but it is not consistent across the studied projects. For one of the application projects (Project 3), there was a measure of association between faults and the total LOC of around 0.6. For the other application project (Project 1), the measure was around 0.4, whereas for the platform project (Project 2), there was no correlation. Graves et al. [10] argue that the number of times that the code is changed might be a better predictor for how many faults the system will contain than the LOC. The correlation between size and fault density is supported neither by the original nor by the replication study. The differing results also indicate that the hypotheses do not capture the essence of these questions. Factors not taken into account in the studies probably have a larger impact. The validity of the analysis as such is questioned by Rosenberg [29]; hence, it is not surprising that there are no significant results.

Fault densities are hypothesized to be constant between releases and similar across environments. Both hypotheses are supported in the original study and, to some extent, also in the replicated study when investigated on subsequent projects instead of releases of the same product. Among the

three studied projects, we identify larger dissimilarities between the application projects and the platform project, whereas the two application projects are more closely comparable. Studying the percentage distribution of faults in Table 3 shows another view of "similarity." In this case, all three projects are "similar." We interpret the data as follows: The development process used in all three projects very much governs percentage distribution, whereas the technical characteristics and the amount of newly developed code of the platform versus application projects show different behavior in terms of fault density. Therefore, we confirm the hypotheses, although not drawing any conclusions as to whether the similarity should be measured in fault density or as a percentage distribution of faults found in the development process.

In summary, the replicated study differs from the original study with respect to size, type of application, number of faults, and so forth. There is substantial variation between the three studied projects. Nonetheless, we confirm some general statements. This indicates that there are certain underlying phenomena that can be observed and possibly formulated as theories, although on a very general level [34]. These levels of detail do not seem to be sufficiently specific to be effective for predictive purposes.

6 Conclusions

The motivation for conducting this study was to contribute to the body of empirical research on software fault behavior in large complex software systems. Others have conducted similar investigations, and the same hypotheses have been examined before. Nevertheless, research such as a replication is a cornerstone of science [27]. For case studies, we are not concerned with sampling and statistical inference across studies; rather, cases are specifically selected to investigate whether a theoretical proportion holds under different conditions. Given the relatively basic nature of our understanding of software development processes and associated fault distributions, our strategy is to attempt either confirming or rejecting the structures and behavior that are published in unique studies. Unfortunately, published studies in this area are rare. This may be a result of the substantial effort required to conduct such investigations and perhaps also because replications are not really acknowledged as important research. A replicated evaluation would provide meaningful results for a real-world setting, but the software engineering field still has difficulties in recognizing the benefits [17].

Generalizing the results can only be accomplished on a step-by-step basis. The purpose of the replication is to investigate variations of the original study. Thus, the same hypotheses (bar one) are investigated, but in a different context and development environment. The generalization of the findings can be achieved by gradually expanding the context. The investigated projects in the replication study belong to the same field as the system in the original study (that of telecommunications) although the type of system is different. In the original study, a switching system was developed, and in the replication study, the projects under examination developed consumer products. In addition, the sizes of the systems concerned differ, as do the programming languages used. In the replication study, some parameters are the same, whereas others are not. The variations between the studies are considered important factors in increasing the understanding of fault distributions.

The results obtained in some of the hypotheses are not the same in the original study, as in the replication study. Even for the three projects included in the replication study, the results are not identical. Clearly, the dissimilarities indicate that the variations in the research setting regarding the software system under investigation and its development environment influence the results to a large extent. In these cases, the hypotheses do not sufficiently reflect the underlying factors. Nonetheless, we believe that the results are of interest to a wider community, and we wish to contribute to the empirical and scientific basis of this kind of data. By publishing empirical data that either supports or rejects both conventional wisdom and the results from the original study, more lessons can be learned on software fault distributions.

References

[1] E.N. Adams, "Optimizing Preventive Service of Software Products," *IBM J. Research and Development*, vol. 28, no. 1, pp. 2-14, 1984.

[2] C. Andersson and P. Runeson, "A Spiral Process Model for Case Studies on Software Quality Monitoring—Methods and Metrics," *Software Process: Improvement and Practice*, 2007, to appear, Digital Object Identifier 10.1002/spip.311.

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

[4] J. Bosch, *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach.* Addison-Wesley, 2000.

[5] T.B. Compton and C. Withrow, "Prediction and Control of ADA Software Defects," *J. Systems and Software*, vol. 12, pp. 199-207, 1990.

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

[7] G. Denaro and M. Pezzè, "An Empirical Evaluation of Fault-Proneness Models," *Proc. 24th Int'l Conf. Software Eng. (ICSE '02)*, pp. 241-251, 2002.

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

[9] N.E. Fenton and S.L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach.* Thomson Computer Press, 1996.

[10] T.L. Graves, A.F. Karr, J.S. Marron, and H. Siy, "Predicting Fault Incidence Using Software Change History," *IEEE Trans. Software Eng.*, vol. 26, no. 7, pp. 653-661, July 2000.

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

[12] *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std 610.12-1990, 1990.

[13] J.M. Juran and F.M. Gryna, Jr., *Quality Control Handbook*, fourth ed. McGraw-Hill, 1988.

[14] M. Kaâniche and K. Kanoun, "Reliability of a Commercial Telecommunications System," *Proc. Seventh Int'l Symp. Software Reliability Eng. (ISSRE '96)*, pp. 207-212, 1996.

[15] R.M. Lindsay and A.S.C. Ehrenberg, "The Design of Replicated Studies," *Am. Statistician*, vol. 47, no. 3, pp. 217-228, 1993.

[16] Y.K. Malaiya and J. Denton, "Module Size Distribution and Defect Density," *Proc. 11th Int'l Symp. Software Reliability Eng. (ISSRE '00)*, pp. 62-71, 2000.

[17] J. Miller, "Replicating Software Engineering Experiments: A Poisoned Chalice or the Holy Grail," *Information and Software Technology*, vol. 47, no. 4, pp. 233-244, 2005.

[18] J.C. Munson and T.M. Khoshgoftaar, "The Detection of Fault-Prone Programs," *IEEE Trans. Software Eng.*, vol. 18, no. 5, pp. 423-433, May 1992.
[19] K.-H. Möller and D.J. Paulish, "An Empirical Investigation of Software Fault Distribution," *Proc. First Int'l Software Metrics Symp. (Metrics '93)*, pp. 82-90, 1993.
[20] N. Ohlsson and H. Alberg, "Predicting Fault-Prone Software Modules in Telephone Switches," *IEEE Trans. Software Eng.*, vol. 22, no. 12, pp. 886-894, Dec. 1996.
[21] T.J. Ostrand and E.J. Weyuker, "The Distribution of Faults in a Large Industrial Software System," *Proc. ACM SIGSOFT Int'l Symp. Software Testing and Analysis (ISSTA '02)*, vol. 27, no. 4, pp. 55-65, 2002.
[22] T.J. Ostrand, E.J. Weyuker, and R.M. Bell, "Predicting the Location and Number of Faults in Large Software Systems," *IEEE Trans. Software Eng.*, vol. 31, no. 4, pp. 340-355, Apr. 2005.
[23] S.L. Pfleeger and L. Hatton, "Investigating the Influence of Formal Methods," *Computer*, vol. 30, no. 2, pp. 33-43, Feb. 1997.
[24] S.L. Pfleeger, "Soup or Art? The Role of Evidential Force in Empirical Software Engineering," *IEEE Software*, vol. 22, no. 1, pp. 66-73, Jan./Feb. 2005.
[25] L.M. Pickard, B.A. Kitchenham, and P.W. Jones, "Combining Empirical Results in Software Engineering," *Information and Software Technology*, vol. 40, no. 14, pp. 811-821, 1998.
[26] J.F. Reh, "Pareto's Principle: The 80-20 Rule," *Business Credit*, vol. 107, no. 7, p. 76, 2005.
[27] C. Robson, *Real World Research*, second ed. Blackwell, 2002.
[28] M. Roper, M. Wood, and J. Miller, "An Empirical Evaluation of Defect Detection Techniques," *Information and Software Technology*, vol. 39, no. 11, pp. 763-775, 1997.
[29] J. Rosenberg, "Some Misconceptions about Lines of Code," *Proc. Fourth Int'l Software Metrics Symp. (Metrics '97)*, pp. 137-142, 1997.
[30] P. Runeson, A. Andrews, C. Andersson, T. Berling, and T. Thelin, "What Do We Know about Defect Detection Methods," *IEEE Software*, vol. 23, no. 3, pp. 82-90, May/June 2006.
[31] C. Withrow, "Error Density and Size in Ada Software," *IEEE Software*, vol. 7, no. 1, pp. 26-30, Jan./Feb. 1990.
[32] R.K. Yin, *Case Study Research: Design and Methods*, third ed. Sage, 2003.
[33] T. Yu, V.Y. Shen, and H.E. Dunsmore, "An Analysis of Several Software Defect Models," *IEEE Trans. Software Eng.*, vol. 14, no. 9, pp. 1261-1270, Sept. 1988.
[34] A. Zendler, "A Preliminary Software Engineering Theory as Investigated by Published Experiments," *Empirical Software Eng.*, vol. 6, no. 2, pp. 161-180, 2001.

Carina Andersson received the MSc degree in engineering physics with industrial management in 2001 and the PhD degree in software engineering from Lund University in 2006. She is a research associate in the Department of Computer Science, Lund University, Sweden. Her main research interests are software development verification and validation processes and software quality metrics and models.

Per Runeson is a professor of software engineering at Lund University and is the leader of the Software Engineering Research Group. He currently holds a senior researcher position funded by the Swedish Research Council. His research interests include software development methods and processes, in particular, methods for verification and validation. He serves on the editorial boards of the *Empirical Software Engineering Journal* and several IEEE program committees and is a senior member of the IEEE.