Information and Software Technology 58 (2015) 250–258

Contents lists available at ScienceDirect

Information and Software Technology

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

On the probability distribution of faults in complex software systems

Tihana Galinac Grbac [a,*], Darko Huljenić [b]

[a] University of Rijeka, Vukovarska 58, HR-51000 Rijeka, Croatia
[b] Ericsson Nikola Tesla, Krapinska 45, HR-10000 Zagreb, Croatia

ARTICLE INFO

Article history:
Received 4 February 2014
Received in revised form 16 June 2014
Accepted 18 June 2014
Available online 10 July 2014

Keywords:
Software fault distribution
Probability distribution
Non-linear regression
Complex software system
Empirical research

ABSTRACT

Context: There are several empirical principles related to the distribution of faults in a software system (e.g. the Pareto principle) widely applied in practice and thoroughly studied in the software engineering research providing evidence in their favor. However, the knowledge of the underlying probability distribution of faults, that would enable a systematic approach and refinement of these principles, is still quite limited.

Objective: In this paper we study the probability distribution of faults detected during verification in four consecutive releases of a large-scale complex software system for the telecommunication exchanges. This is the first such study analyzing closed software system, replicating two previous studies for open source software.

Method: We take into consideration the Weibull, lognormal, double Pareto, Pareto, and Yule–Simon probability distributions, and investigate how well these distributions fit our empirical fault data using the non-linear regression.

Results: The results indicate that the double Pareto distribution is the most likely choice for the underlying probability distribution. This is *not* consistent with the previous studies on open source software.

Conclusion: The study shows that understanding the probability distribution of faults in complex software systems is more complicated than previously thought. Comparison with previous studies shows that the fault distribution strongly depends on the environment, and only further replications would make it possible to build up a general theory for a given context.

1. Introduction

The knowledge of fault distributions in large-scale complex software systems is very important for planning the quality assurance activities. There are several empirical principles, widely applied in software development practice, related to the distribution of faults. For example, the Pareto principle [1,2], also known as the 20–80 rule, is one of the most popular among them. It states that a majority of faults (80%) in a software system is contained in a minority of software modules (20%). There is a lot of empirical evidence in favor of this principle [3–7,2,8–10]. Another example is the principle that the minority of modules containing the majority of faults confines not too large portion of the system size. Empirical evidence for this principle is obtained in [8–10].

All such principles ultimately depend on the underlying probability distribution of faults in a software system. However, the converse is not true, that is, the fulfillment of a certain principle does not determine the probability distribution uniquely. For example, there are several distributions, besides the Pareto distribution, that would result in the Pareto principle. In other words, the empirical evidence in favor of some principle does not imply information on the probability distribution, and, indeed, our knowledge on the probability distribution of faults in software systems is still quite limited.

Recently a lot of attention is put to the more general problem of determining probability distributions of various metrics in software engineering (see e.g. [11–13]). The final goal of all these works, as well as this paper, is to refine the empirical principles used in software engineering practice, and possibly even use the precise knowledge of probability distributions to predict the behavior of future releases of a complex software system.

The knowledge of the most appropriate probability distribution fitting the empirical fault data in complex software systems would enable more systematic approach and refinement of the Pareto principle and other related principles used in the software development practice. This line of thought is pursued in works of Zhang [14] and

* Corresponding author.
E-mail addresses: tihana.galinac@riteh.hr (T. Galinac Grbac), darko.huljenic@ericsson.com (D. Huljenić).

http://dx.doi.org/10.1016/j.infsof.2014.06.014
0950-5849/

Concas et al. [15]. Both papers study the fault data for the open source Eclipse system using the non-linear regression for fitting.

Zhang [14] compares how the Pareto and Weibull distributions fit the data, and conclude that the Weibull distribution is significantly better. As explained above, this is not contradictory to the Pareto principle itself, since the Pareto principle does not imply the underlying probability distribution.

Concas et al. [15] consider the Weibull, lognormal, double Pareto, and Yule–Simon distributions. The results reveal that, for the Eclipse system, the Yule–Simon distribution provides better fit than the others. The Weibull, lognormal, and double Pareto distributions are quite close, although the Weibull distribution is the worst in all five considered system releases. The authors argue further in favor of the Yule–Simon, but also lognormal and double Pareto distributions, since they all have a generative model, unlike the Weibull distribution.

Motivated by these two papers on the Eclipse system, and the importance of finding the most appropriate distribution, we study the probability distribution of faults in a very different context, that is, in four consecutive releases of the large-scale complex software system for telecommunication exchanges. We consider all distributions appearing in [14,15]. These are the Pareto, Weibull, lognormal, double Pareto, and Yule–Simon distributions. As in [14,15], the fitting method is the non-linear regression for the fault data in the form of the complementary cumulative distribution function (CCDF) of the random variable counting the number of faults in a software module.

In reporting the results of non-linear regression we follow closely the exposition of [15] to simplify the comparison. Additionally to the goodness-of-fit measures, we report the distribution parameters obtained for the best fits. These are not reported in [15], and we can compare only to the Pareto and Weibull distribution fits of [14]. We provide such detailed results, so that the replications of this study for other software systems could be easily compared.

It turns out that the results are different from those of [14,15], which can be explained by a very different context. In our study the double Pareto distribution is the best fit to the fault count data. The lognormal distribution is the second best, followed closely by the Yule–Simon distribution, which is even slightly better in one of the projects. Only then comes the Weibull distribution, while the Pareto distribution is worse than others. However, the Pareto distribution performs much better than reported by [14]. We hope that this study will become a source of several replications in both similar and different contexts, so that the most appropriate probability distributions of faults could be identified in different types of software systems and development environments.

The paper is organized as follows. In Section 2 the context of the study and the fault data are described in detail. Section 3 recalls the considered probability distributions. The results of the non-linear regression fit are reported in Section 4. Section 7 concludes the paper.

2. Context of the study

We describe in this section the context of this study including the software system, the development organization, the software development process, and the data collection performed for the purpose of this study.

2.1. Software system

The study is undertaken on a sequence of four development projects, denoted P1, P2, P3, P4, developing consecutive releases of the same software product. The considered projects are the same as in the study [10], except that the last two releases, denoted by Rel $n+3$ and Rel $n+4$, are combined into project P4. The reason is that the size of datasets from these two releases is too small for a reliable fitting, and they are indeed two subprojects of the same development project.

The software product is the software system for the Mobile Switching Center (MSC), that is, a functional node in the Third Generation (3G) telecommunication network. The MSC node is built on Ericsson's AXE exchange that has evolved for more than 30 years and is in operation in hundreds of exchanges all over the world.

The system is coded in Ericsson's in-house Programming Language for EXchanges (PLEX). It is a large-scale software system with several millions lines of code (LOC). The software system architecture is modular, involving more than 1000 software modules.

2.2. Organization

The development organization is a globally distributed Ericsson's unit with long experience in software development for AXE exchange. The number of involved development units varied during the projects. A typical software development project involves more then 300 developers world-wide and lasts for one to two years.

2.3. Development and verification process

The software development process has evolved over the years from the traditional waterfall model by introducing the incremental and iterative delivery and feature development. In this study we concentrate on the faults detected during the testing part of verification process, which consists of the function test (FT), system test (ST) and system integration test (SI). The essential difference is in the system coverage under the test. The function test covers functional environment, that is, only software modules responsible for the functional execution and is very often executed in the simulated system environment. The system test covers essential system environment for function integration, often executed on the test plants, and the system integration test that covers all deployment environment executed on the test plants.

The fault handling process consists of collection of trouble reports (TR) issued whenever the failure occurs. It is very precise and contains all the information required for fault analysis and fault decision process. The same process is used during the software verification and during the system in operation. The fault handling process is a standard Ericsson's process. TRs are stored in a database, which can be easily searched.

For every failure that occurs during verification, one or more TRs are issued. This is because there could be one or more faults in the code responsible for the same failure. Hence, a TR is issued for each location in the code (software module) that could contain the fault causing the failure. These TRs are answered, and an answer code is attached to the TR. The answer code indicates whether the fault really exists and should be corrected, and whether the fault is already corrected as a consequence of another TR. Duplication of TRs could happen due to parallel testing activities and since the same fault could be a reason for several failures. More precisely, for every fault in the code there is exactly one TR with the answer code saying the fault should be corrected. Only these TRs are included into our analysis and all duplicates were excluded.

2.4. Data collection

As a result of the standard TR handling process all relevant data regarding TR collection, analyzing and answering are stored in the database.