On the distribution of software faults in evolution of complex systems

Tihana Galinac Grbac
Faculty of Engineering, University of Rijeka
Rijeka, Croatia
tihana.galinac@riteh.hr

Goran Mauša
Faculty of Engineering, University of Rijeka
Rijeka, Croatia
goran.mausa@riteh.hr

ABSTRACT

Complex software systems and systems of systems have become essential in the modern human society, making their reliability one of the crucial problems in software engineering. As such systems are developed as a sequence of releases, it is important to understand the reliability behavior during their evolution. There are many empirical principles regarding the distribution of faults within system structure. All these principles are implied by the underlying probability distribution of faults. The aim of this paper is to find the probability distribution that best fits the empirical fault data from 21 versions of two evolutionary developed open source systems, and study how this distribution changes during system evolution.

CCS CONCEPTS

• **Software and its engineering → Software system structures**; *Software reliability*;

KEYWORDS

Systems of Systems, complex systems, fault distributions, system structure

ACM Reference Format:
Tihana Galinac Grbac and Goran Mauša. 2016. On the distribution of software faults in evolution of complex systems. In *SiSoS@ECSA 2016: International Colloquium on Software-intensive Systems-of-Systems at 10th European Conference on Software Architecture (ECSA Volume 3), November 29, 2016, Copenhagen, Denmark.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3175731.3176181

1 INTRODUCTION

Global trend of proactive introducing information and communication technologies in various application domains such as medicine, smart homes, smart cities, automotive etc. and integrating these systems and their applications and services opens new innovative opportunities in all these domains. This integration leads to composing Systems of Systems (SoS). Systems of systems are usually mission critical, supporting numerous people in their everyday activities. That is why their reliable and failure free operation becomes so important. Because of their importance in supporting humans in their everyday activities usually they are integrating functionality of variety of other systems thus they gradually become complex. From stand–alone systems these systems become networked and tend to grow to systems of systems. Although this evolution trend is evident, we still do not have mature discipline for engineering such systems and their evolution. New modelling approaches that are able to simulate system properties already during design of these systems, such as modelling of system growth in relation to its reliability, are the key research directions engineering these Systems of Systems that can help humans to further evolve these systems maintaining its key quality attributes such as reliability.

Integration of various systems is implemented in their complex software. Software is an abstract representation of system that hides complex details of system behaviour. Integrating abstractions of various systems needs deep understanding of system behaviour and understanding of consequences of modeling this abstract interactions. The system modeling and programming activity is human intellectual activity. How humans model software is based on their logical reasoning in solving complex problems by decomposing it into smaller, simpler functional units, blocks, modules, classes. Complex system behaviour is then realised by dynamic interaction of those simpler functional units. These simpler functional units are developed by numerous developers, development teams, communities usually worldwide. Humans are error prone and limited in understanding consequences of their design activities on whole complex system behaviour. Therefore, software systems that are composed from such simpler units are usually prone to failures and huge amount of software development and verification resources are spent to implement their reliable behaviour in operation. Composition of such simpler functional units forms the system structure. Existing system design principles guide us how to define system structure, how to model its elementary structure elements and their interconnections. However, we lack some formal approach that could simulate consequences of our design decisions and that is especially important while building such Systems of Systems. System structure hides some properties that may be used for their modelling and modelling of their behaviour. Here in this paper our focus is on modelling fault distributions over the software structure. In our previous work we found that empirical Pareto principle is valid for industrial context and identified fault probability distributions across industrial system releases that seem consistent [6, 7]. Moreover, we found that system structure evolves during the system evolution and this evolution path is determined from the initial system structure, [12]. Furthermore, we found some hidden topological structure properties that may be used for such modeling [13].

SiSoS@ECSA 2016, November 29, 2016, Copenhagen, Denmark
© 2016 Association for Computing Machinery.
ACM ISBN 978-1-4503-6399-0/16/11…$15.00
https://doi.org/10.1145/3175731.3176181

System reliability is usually modeled as probability of failure during system operation. There are numerous reliability models that aim to model system reliability and the most popular ones are Reliability Growth Models [9, 11]. We may also observe failures as manifestation of processed faults that are implemented in software program by human mistake while programming. So, modelling software fault distributions across the system structure is another approach to modelling system faultiness that is in some relation to system reliability that is our goal to investigate in wider context of our research. We may relate this modelling to modelling human intellect for modeling system abstractions. Interesting finding from previous work is that distribution of programming faults across the system units follows Power Law distributions [3, 6, 16]. This fact has been empirically validated for open source and industrial systems. Complex systems are not developed in one project but they evolve to complex system through number of incremental modifications implemented in numerous sequential system releases. Here, it is important to stress that empirical evidence shows that fault behaviour can be modeled by Power Law distributions. That implies that fault distributions in one system release can not be modeled as a consequence of random incremental independent modification and the probability of fault with finite variance but exhibit dependencies with fault behaviour of previous system releases. Moreover, software structure evolution has also some hidden evolutionary path. The modeling of system fault distributions in evolving complex software systems is still unexplored area of system and software engineering discipline and this paper has contribution in that direction.

In modeling complex software system behaviour we have limited understanding of real underlying phenomena. The main reason lies in limited number of empirical studies and very weak accessibility of information from these systems and their development environments. Empirical studies from system engineering discipline are rare, usually performed on open source systems with weak generalisation to industrial systems and especially with weak analytical interaction to other modeling approaches. Importance of this study is exactly to add on empirical evidence about fault distributions in evolving complex software system structures, interconnecting empirical and analytical studies from open source and industrial environments and discussing conclusions from various modelling approaches.

The first step in studying the fault behavior in complex systems is understanding their distribution among software components. This knowledge can be crucial in modeling system reliability, and also in development process for early planning of development activities, such as testing effort and system architecture. The idea is to find the probability distribution that best fits empirical fault data for various complex systems, and study how such distribution changes over system evolution.

Although the software fault behavior is an important problem that was studied extensively resulting in many empirically verified principles (e.g. the Pareto principle [4], [2], [7]), there are only a couple of studies that approach the problem by looking at the underlying probability distribution, which gives rise to those principles. The work [6] considers four releases of the closed source industrial complex software system for the telecommunications domain. On the other hand, there are two papers [16] and [3] considering the open source Eclipse project. The former considers three, and the latter five consecutive versions. Although two of these versions coincide, it is not possible to compare the results, because the considered families of distributions are different, and the distribution was not studied with respect to the same software components. In [16] the the distribution of faults was studied among packages of the Eclipse system, while in [3] it was studied among the files of the Eclipse system.

In this paper we focus on empirical study of fault distributions in large scale open source complex systems in evolution. We perform empirical study on Eclipse projects JDT and PDE and their 11 and 10 releases respectively. We discuss applicability of probability distributions (Weibull, Lognormal, Pareto, Double Pareto and Yule Simon) to model fault behaviour in these two open source projects in relation to previously performed studies in the same open source context. Furthermore, we discuss result obtained in this study in relation to results obtained in our previous study while modelling the same probability distributions to the large scale telecommunication system in evolution from an industrial context [6]. Here it is important to stress that the same experimenter has been involved in both studies, in both data collection and modelling activities, that provides unique opportunity to better understand the differences of these two contexts and possible threats to validity of this approach. Furthermore, we discuss these results in relation to our previous studies using other reliability modelling approaches and structural analysis performed on the same datasets from industrial and open source development environment. That discussions and obtained conclusions are the main contributions of this paper. Here we present just overview of our key findings due to space limitation. The results obtained are directly contributing to the software and systems engineering discipline with conclusions about evolutionary behaviour of software fault and failure distributions across the software structure and time. The main results are identified benefits and limitations of these modelling approaches across the open source and industrial contexts.

The remainder of this paper is organized as follows. Section 2 describes the context of the study. Section 3 presents analytical model. Section 4 describes evaluation model. Section 5 presents our detailed results, followed by Section 6 in which we discuss possible threats to validity. Section 7 further discusses the results and provides some insights into future work. Finally, Section 8 concludes the paper.

2 CONTEXT OF THE STUDY

This study is performed within wider research scope of research project *Evolving Software Systems: Analysis and Innovative Approaches for Smart Management (EVOSOFT)*[1] aiming to understand fault distributions and effects of different modeling approaches in different evolving complex system development contexts. Since the datasets are crucial for proper empirical research, part of our efforts were devoted to develop proper and systematic data collection procedures that would allow us to minimise data collection threats to validity and maximise generalisation ability of results obtained on these datasets. We are especially interested in transparency of conclusions between industrial and open source software systems,

[1]`http://www.seiplab.riteh.uniri.hr/?pageid=712&lang=en`

since data from open source systems are more accessible to research communities and consequently more used in empirical research studies. Furthermore, we are also experimenting with different modeling approaches to the same datasets thus aiming to understand relations among them in this particular context of fault and failure distributions in the large scale complex software systems.

The datasets consist of measurements performed on functional software element that is elementary building block in software structure. In case of industrial system this functional software unit for which the faults are counted is so called software unit, that is the smallest self contained software artifact forming the system and can be replaced independently from the other system during the system operation. The same unit is the smallest administrative unit of complex system structure that is represented in various industrial software development repositories. On the other hand, in case of the open source software, we analysed Java code and this functional unit that made elementary building block in software structure for which we collected number of faults is Java class (excluding inner classes). Note that in previous studies on fault probability distributions from open source contexts the structural unit for fault count data was package in [16] and file in [3]. The measurements were measuring the number of faults detected in prerelease for each functional unit. Prerelease faults are the faults reported before official system release to network integration test in case of telecommunication system from closed industrial context and, for open source community these are the faults that are reported before system is released to the open community. We count only faults, locations in software functional units, where the system failure reported from the testing phase is located in the particular functional software unit as fault and corrected. Note that here one has to relate system failure occurrence with fault in software code in particular functional software unit. This relation is not clear and developer knowledge is needed to find this relation. If this relation is not captured in the repository (that is usually the case) then it is hard for researcher to identify these links. Moreover, there may be several faults that may lead to one system failure and vice versa, several failures may occurred because of one single fault in software code. Note, that during the numerous testing activities duplicate failures that are caused by the same fault in software code may be reported. All duplicates are removed for both datasets obtained in both contexts.

Datasets for software fault distributions are sensitive on data collection procedure because it combines data from two separated software lifecycle phases (and different software development repositories), i.e. from software program structures that are artifacts from software development phase (obtained from code repositories e.g. Git) and system failures as artifacts from system verification phase (obtained from failure repositories, e.g. Bugzila). It is not trivial to find relation among software structure elements where faults are corrected and system failures that are caused by these faults. Usually this link is not captured and maintained within software development repositories. More information about datasets reliability and systematic data collection procedures is published in [10]. Part of datasets used in this study are freely available[2].

As we already mentioned we study two different development contexts: an industrial closed source large scale software system from telecommunication domain and its four releases during its evolution and two open source software systems JDT and PDE of Eclipse development community with its 11 and 10 consecutive releases, respectively. In both cases the first author of this study was directly involved in data collection procedure that provides better understanding of underlying effects that data collection may impose and thus represent threats to study validity. Furthermore, the obtained conclusions on both datasets are more transparent to the data collection bias.

Industrial software (IS) is a large scale telecommunication EVOSOFT that evolved during more than 30 years, with several millions lines of code, developed in globally distributed organization. The software is developed by development teams that are working in globally distributed local design centers. Development methodology is based on Waterfall model but with functional team organisation and feature driven planing in incremental system delivery and iterative software unit delivery. We have data collected for four consecutive industrial projects from Mobile switching Center software system of Ericsson, and have published several studies using these datasets [6, 7].

On the other hand, in this study we use datasets of JDT and PDE Eclipse open source development project. The development methodology in open source community is different from closed industrial development community, with much less standards, and administrative documentation. Open source community is more Agile in sense of formalizing the project, its realisation and system evolution. From that point of view, we expect that the evolution trends observed in these two development contexts by using different modeling approaches should reflect these differences. Eclipse development community has defined to use within its development project (such as Eclipse PDE, JDT, MyLin, etc) the open source code and bug repositories (e.g. GIT and Bugzilla). Eclipse open source community projects are chosen because there is large body of knowledge based on empirical studies exactly in this context and usually with week documenting of data collection procedures that is crucial importance for validity of empirical studies especially in this particular case when datasets have to be linked by researcher and this process is not so straight forward in software development repositories. Furthermore, the Eclipse projects were selected because their satisfactory repository of both, bug reports and commit changes that can create dataset appropriate for our analysis. Moreover, we wanted to analyze projects that are as large as possible, so we choose the projects with the greatest number of versions. Within the student projects at the Faculty of Engineering, University of Rijeka, we have developed number of tools that automatise data collection procedure as published in [10], [12] and datasets that are partially available on our web page. All the experiments using these datasets are performed using licences of Matlab and Statistica.

The datasets used in this study are presented in Table 1.

[2]http://www.seiplab.riteh.uniri.hr/?page_id=834

Table 1: The Eclipse datasets: basic statistics

Project	nr. of software units	total faults	nr. of faults in 20% most faulty units	% of faults in 20% most faulty units
PDE 2.0	576	242	242	100
PDE 2.1	761	231	231	100
PDE 3.0	881	584	485	83
PDE 3.1	1108	733	636	87
PDE 3.2	1351	1124	608	54
PDE 3.3	1713	1888	1231	65
PDE 3.4	2144	2087	1904	91
PDE 3.5	2297	2144	1861	87
PDE3.6	2412	756	648	86
PDE 3.7	2404	819	658	80
PDE 3.8	3522	68	68	100
JDT 2.0	2397	3754	2976	79
JDT 2.1	2743	2147	1818	85
JDT 3.0	3420	4491	3758	84
JDT 3.1	3883	4212	3716	88
JDT 3.2	2233	2317	1936	84
JDT 3.3	4821	2545	2361	93
JDT 3.4	4932	1764	1764	100
JDT 3.5	4395	935	935	100
JDT 3.6	4392	690	690	100
JDT 3.7	4415	715	715	100
JDT 3.8	4444	709	709	100

3 PROBABILITY DISTRIBUTIONS

In the previous empirical studies on fault distributions it is reported that the most appropriate probability distribution fitting the empirical fault data are Yule–Simon, lognormal and double Pareto. These distributions have generative model and this can be explained by evolving trend of fault distributions through system evolution. Motivated by this findings here we analyse like in previous work Pareto, Weibull, lognormal, double Pareto, and Yule–Simon distributions for modeling software fault distributions across the software system modules.

Let X be the random variable counting the number of faults in a software module. It is a discrete random variable taking values in the set $\mathbb{N}_0$ of non-negative integers. Given a sample of software modules, let f_k denote the relative frequency of modules with k faults, for $k \in \mathbb{N}_0$. The empirical distribution of X on that sample is given by its probability mass function PMF

$$p : \mathbb{N}_0 \to [0, 1], \tag{1}$$

given by the assignment $p(k) = f_k$, for $k \in \mathbb{N}_0$. The goal of the paper is to find the best theoretical distribution fitting the empirical distribution of X for samples described in Sect. 2.

As in the previous works [3] and [6], instead of fitting the PMF, we fit the complementary cumulative distribution function CCDF, which is in fact an equivalent problem, as explained in [3]. For a discrete random variable X with the PMF p defined on $\mathbb{N}_0$, the CCDF is given as

$$P(X > x) = \sum_{k>x} p(k) = \sum_{k>x} f_k,$$

where $x \in \mathbb{N}_0$. However, most of the families of probability distributions that we consider are actually continuous. Nevertheless, they will be used to fit the discrete random variable X, viewing the empirical values of CCDF as the values of the CCDF for the continuous random variable at integer arguments.

For the continuous random variable X, let p be the probability density function PDF, defined on $\mathbb{R}$ or some of its subsets, usually $\mathbb{R}_{>0}$. Then the CCDF $P(X > x)$ of X is related to the PDF p by

$$p(x) = -\frac{d}{dx} P(X > x),$$

where x is in the domain of p. The PDF and CCDF for the probability distributions considered in this paper are given in Table 2. For more details on the considered probability distributions, we refer to the previous work [6] and [3], and the references therein.

4 NON-LINEAR REGRESSION FIT

The fitting to empirical data is done in the same way as in the previous works [3, 6, 16]. More precisely, the fitting of the empirical distribution's CCDF for the random variable X, counting the number of faults in software modules of the 21 open source software development projects described in Sect. 2, was conducted using the *MATLAB* curve fitting tool. This tool uses the non-linear regression to estimate the parameters of the family of probability distributions in which the model for the empirical data is sought. The non-linear regression in *MATLAB* is based on numerical algorithms for minimizing the sum square error

$$SS_{\text{err}} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2, \tag{2}$$

where y_i and $\hat{y}_i$ are the actual and fitted values of ith observation, and n is the number of observations, with respect to the possible values of the unknown parameters. The estimated parameters are the values of parameters for which the minimal sum square error is obtained. The families of possible models considered in this paper are the families of probability distributions listed in Sect. 3. These are the same as in [3] and [6].

As in the previous works, we take the adjusted coefficient of determination R^2 and the standard error of estimate S_e as measures of goodness-of-fit. The adjusted R^2 is adjusted to the degrees of freedom of the distribution. It is defined as

$$R^2 = 1 - \frac{(n-1)SS_{\text{err}}}{(n-m-1)SS_{\text{tot}}}, \tag{3}$$

Table 2: PDF and CCDF for the considered probability distributions

Name	Type	PDF/PMF	CCDF
Weibull	cont.	$\frac{\beta}{\gamma} \cdot \left(\frac{x}{\gamma}\right)^{\beta-1} \cdot \exp\left(-\left(\frac{x}{\gamma}\right)^{\beta}\right), \quad x > 0$	$\exp\left(-\left(\frac{x}{\gamma}\right)^{\beta}\right), \quad x > 0$
Lognormal[a]	cont.	$\frac{1}{x\sqrt{2\pi\sigma^2}} \cdot \exp\left(-\frac{(\ln x-\mu)^2}{2\sigma^2}\right), \quad x > 0$	$1 - \Phi\left(\frac{\ln x-\mu}{\sigma}\right), \quad x > 0$
Pareto	cont.	$\begin{cases} 0 \\ \beta x_m^{\beta} \cdot x^{-(1+\beta)} \end{cases}$	$\begin{array}{ll} 1, & \text{for } 0 < x < x_m, \\ (x_m/x)^{\beta}, & \text{for } x \geq x_m, \end{array}$
Double Pareto	cont.	$\begin{cases} \frac{\gamma}{t} \cdot \frac{\left[1+(x_M/t)^{-\beta}\right]^{\gamma/\beta}}{\left[1+(x/t)^{-\beta}\right]^{1+\gamma/\beta}} \cdot \left(\frac{x}{t}\right)^{-(1+\beta)} \\ 0 \end{cases}$	$\begin{array}{ll} 1 - \left[\frac{1+(x_M/t)^{-\beta}}{1+(x/t)^{-\beta}}\right]^{\gamma/\beta}, & \text{for } 0 < x \leq x_M, \\ 0, & \text{for } x > x_M, \end{array}$
Yule–Simon[b]	disc.	$p(k) = p(0) \cdot \frac{B(c+k,\alpha)}{B(c,\alpha)}, \quad k \in \mathbb{N}_0$	$P(X > x) = p(0) \cdot \frac{\sum_{k>x} B(c+k,\alpha)}{B(c,\alpha)}$

[a] Φ is the CDF of the standard normal distribution.
[b] $B(a, b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$ is the beta function.

where m is the number of parameters in the fitting function, and

$$SS_{\text{tot}} = \sum_{i=1}^{n}(y_i - \bar{y})^2 \tag{4}$$

is the total sum of squares, where $\bar{y}$ is the mean of the observed data. The standard error of estimate S_{e} is defined as

$$S_{\text{e}} = \sqrt{\frac{SS_{\text{err}}}{n - m}}. \tag{5}$$

5 RESULTS

We omit here to provide the detailed results of distribution fitting with estimated parameters of best fit for each distribution and the obtained measures R^2 and S_{e} for goodness-of-fit due to limit in page numbers. Instead we just report an overview of the obtained results in Figure 1 and discuss them and compare to the previous work. In the figure, for each release of the analysed projects PDE and JDT, we provide measures R^2 for the goodness-of-fit of the distributions with estimated parameters.

The best fit for all the analysed project releases, in both cases PDE and JDT, is as follows: Yule-Simon, Pareto, Double Pareto, Lognormal, and Weibull. Only in versions PDE 3.2 and PDE 3.7 this order is a bit different, where Pareto slightly outfits the Yule–Simon distribution. The difference in goodness-of-fit in these projects is so small that we may safely conclude that the Yule–Simon distribution provides the best fit for the considered projects.

This is in accordance with earlier findings in [3], and confirms that the Yule–Simon distribution is a good choice for fitting the fault distributions in open source software systems. Since Yule–Simon distribution has a generative model, this is a promising distribution for early predictions of fault distributions in open source software projects, since the distribution parameters may be estimated from the project properties and early performance.

6 VALIDITY

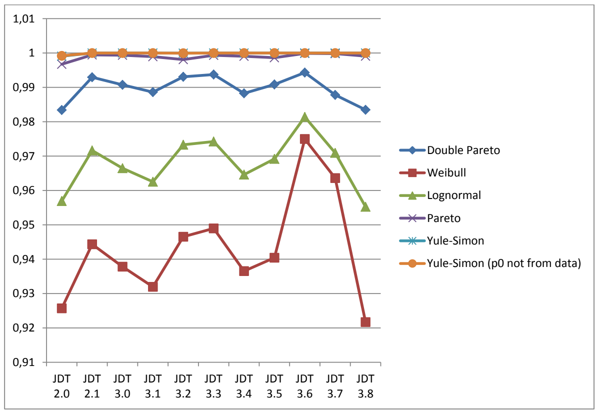
Threats to validity are addressing problem to what extent the results are biased [14]. To discuss construct validity we explain how the probability fault distributions may be used in modelling of complex software systems and for planing verification activities. As we may observe the best fit probability distribution is the same for all analysed releases in evolution and distribution parameters do not change much from release to release. This is an indicator that the probability distribution may be used during software evolution as guiding principle for planing verification activities that is in line with findings from empirical studies investigating Pareto principle that is widely used in practice [2, 4, 7]. However, when we compare best fit analytical distributions between open source and industrial software we may observe some differences in best fit model. The cause of such deviation may be some other factors from the development context that may influence the results and we did not involve them into this study. For example, industrial software projects are more systematic in planning the impacts and system evolution. However, these additional factors have to be further investigated. In this study the datasets are per Java class and we obtained similar result as in previous studies on open source systems [3, 16] although their datasets were per package and file. This may imply that the probability distributions are not affected by granularity of system (class, file, package) used in the analysis. Furthermore, our datasets are obtained by following systematically developed data collection procedure [10]. As we may observe, the similar results are obtained in previous studies in open source environment. On the other hand, the same experimenter has been involved into data collection activity in industrial software for which we obtained different results, see Table 3. The only difference in data collection was that in industrial software systems linking procedure between faults and software modules was inherently implemented within software development repositories and is performed by software developer implementing correction for particular fault. Thus this dataset is more reliable compared to open source projects where linking is artificially made by researcher after the project is finished.

7 DISCUSSION AND FUTURE WORK

In our previous study investigating applicability of empirical Pareto principle we found out that it is generally valid for industrial and open source software systems. However, we did not obtain the same results when analysing analytical fault distributions. It seems that the probability distributions do not behave similarly in industrial and open source software systems. Moreover, we found out that granularity of structure used in the analysis do not effect the best

Table 3: Ranking the probability distributions with respect to their performance in the non-linear regression fitting of the empirical samples for the random variable counting the number of faults in a software module

Rank	This study	Galinac Grbac et al. [6]	Concas et al. [3]	Zhang [16]
1	Yule–Simon	Double Pareto	Yule–Simon	Weibull
2	Pareto	Lognormal	Double Pareto	Pareto
3	Double Pareto	Yule–Simon	Lognormal	–
4	Lognormal	Weibull	Weibull	–
5	Weibull	Pareto	–	–



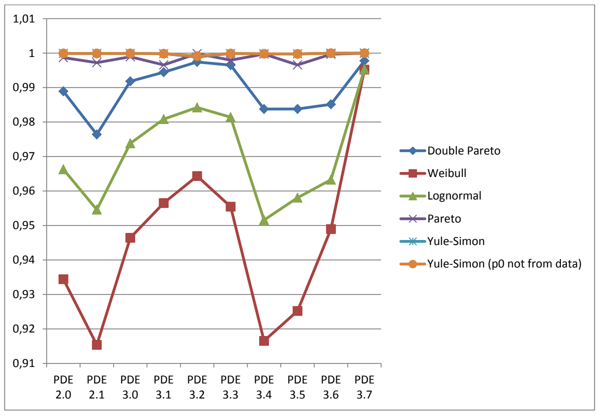


Figure 1: R^2 for non-linear regression fit regression parameter for the double Pareto, Weibull, Lognormal, Pareto and Yule–Simon CCDF distribution to the random variable X in projects JDT and PDE

fit probability distributions and that this property could be generalized for different system granularities. Probably at some higher system granularity levels we may not bring statistical conclusions because of sample size. Moreover, some conclusions from the initial empirical study on fault distributions [4] are confirmed here. The same best fit probability distribution is present in all releases of one software system. This is in line with observation that the fault behaviour does not change much from release to release. Moreover, we obtained similar probability distribution parameters across system releases in open source context. In future work we would analyse the parameters of probability distributions for different verification phases of the industrial software. Empirical study analysing Pareto principle for each verification phase (unit test, system test, function test) has identified that Pareto principle persists across the verification phases and fault distributions are more similar between sequential verification phases [8].

Our previous study investigating software structure evolution has statistically identified that there exists continuous change in software structure during software evolution across releases [12]. For the future work we plan to investigate the correlation between software structure change and changes in probability distribution parameters. That may lead to conclusion how to model software systems and how is software structure affecting the probability fault distributions.

For modelling system reliability there are numerous reliability growth models available [9, 11]. However, it is hard to identify best reliability model early enough, during development project in order to use it for planing of future verification activities. In system evolution, when development projects are overlaping over the same software base this modelling is getting even worse [5]. Furthermore, the best fit reliability growth model of fault data over time are changing as verification time progress [1, 15]. It is interesting for the future work to analyse how the fault distributions are changing as verification progress and its relation to reliability growth models. In that analysis it is interesting to analyse if fault distribution are the system structure property or verification process property. These conclusions may be valuable for early determination of reliability growth model that is crucial in planing verification activities.

8 CONCLUSION

In the conclusion, we compare the results of this study with the results of the previous works. The ranking of the performance of considered probability distributions in this study and the previous ones is given in Table 3.

On datasets, we study software engineering empirical principles, and probability distributions. In the second phase we will examine the findings in relation to other software properties, like examining the effect of software structure on fault distributions and other related local properties including size and modified size. In the third phase we want to investigate possibilities of identifying formal models and innovative approaches for fault distributions that would be useful in simulating software evolution, guiding system design, implementation and verification for the purpose of its smart quality management. Finally, we want to experiment with innovative approaches in real software development environment.

ACKNOWLEDGMENTS

The authors would like to thank to the support of Croatian Science Foundation's funding of the project EVOSOFT (UIP-2014-09-7945) and by the University of Rijeka Research Grant 13.09.2.2.16.

REFERENCES

[1] Carina Andersson. 2007. A Replicated Empirical Study of a Selection Method for Software Reliability Growth Models. *Empirical Softw. Engg.* 12, 2 (April 2007), 161–182.

[2] Carina Andersson and Per Runeson. 2007. A Replicated Quantitative Analysis of Fault Distributions in Complex Software Systems. *IEEE Trans. Softw. Eng.* 33, 5 (May 2007), 273–286.

[3] Giulio Concas, Michele Marchesi, Alessandro Murgia, Roberto Tonelli, and Ivana Turnu. 2011. On the Distribution of Bugs in the Eclipse System. *IEEE Trans. Softw. Eng.* 37, 6 (Nov. 2011), 872–877.

[4] Norman E. Fenton and Niclas Ohlsson. 2000. Quantitative Analysis of Faults and Failures in a Complex Software System. *IEEE Trans. Softw. Eng.* 26, 8 (Aug. 2000), 797–814.

[5] T. Galinac and S. Golubić. 2005. Project overlapping and its influence on the product quality. In *Proceedings of the 8th International Conference on Telecommunications, 2005. ConTEL 2005.*, Vol. 2. 655–662.

[6] Tihana Galinac Grbac and Darko Huljenić. 2015. On the probability distribution of faults in complex software systems. *Inf. Softw. Technol.* 58 (Feb. 2015), 250–258.

[7] Tihana Galinac Grbac, Per Runeson, and Darko Huljenić. 2013. A Second Replicated Quantitative Analysis of Fault Distributions in Complex Software Systems. *IEEE Trans. Softw. Eng.* 39, 4 (April 2013), 462–476.

[8] Tihana Galinac Grbac, Per Runeson, and Darko Huljenić. 2016. A Quantitative Analysis of the Unit Verification Perspective on Fault Distributions in Complex Software Systems: An Operational Replication. *Software Quality Journal* 24, 4 (Dec. 2016), 967–995.

[9] Michael R. Lyu (Ed.). 1996. *Handbook of Software Reliability Engineering.* McGraw-Hill, Inc., Hightstown, NJ, USA.

[10] Goran Mauša and Tihana Galinac Grbac. 2016. A Systematic Data Collection Procedure for Software Defect Prediction. *Computer Science and Information Systems* 13, 1 (2016), 173–197.

[11] John D. Musa. 2004. *Software Reliability Engineering: More Reliable Software Faster and Cheaper.* Authorhouse.

[12] Jean Petrić and Tihana Galinac Grbac. 2014. Software Structure Evolution and Relation to System Defectiveness. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE '14).* ACM, New York, NY, USA, Article 34, 10 pages.

[13] Joao Pita Costa and Tihana Galinac Grbac. to appear. The topological data analysis of time series failure data in software evolution. In *Proceedings of the 8th ACM/SPEC International Conference on Performance Engineering (ICPE '17).*

[14] Per Runeson and Martin Höst. 2009. Guidelines for Conducting and Reporting Case Study Research in Software Engineering. *Empirical Softw. Engg.* 14, 2 (April 2009), 131–164.

[15] C. Stringfellow and A. Amschler Andrews. 2002. An Empirical Method for Selecting Software Reliability Growth Models. *Empirical Softw. Engg.* 7, 4 (Dec. 2002), 319–343.

[16] Hongyu Zhang. 2008. On the Distribution of Software Faults. *IEEE Trans. Softw. Eng.* 34, 2 (2008), 301–302.