On the Probability Distribution of Faults in Complex Software Systems

Tihana Galinac Grbac*

Faculty of Engineering, University of Rijeka, Vukovarska 58, HR-51000 Rijeka, Croatia

Darko Huljenić

Ericsson Nikola Tesla, Krapinska 45, HR-10000 Zagreb, Croatia

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 distribu-

*Corresponding author

Email addresses: tihana.galinac@riteh.hr (Tihana Galinac Grbac), darko.huljenic@ericsson.com (Darko Huljenić)

tion 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.

Keywords: software fault distribution, probability distribution, non-linear regression, complex software system, empirical research

---

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, 4, 5, 6, 7, 2, 8, 9, 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, 9, 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, 12, 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 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] and [15]. These are the Pareto, Weibull, lognormal, double Pareto, and Yule-Simon distributions. As in [14] and [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] and [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

Table 1: Project characteristics

Proj.	Size [kLOC]	Modification size [kLOC]	No. of modules	No. of FT faults	No. of ST faults	No. of SI faults	Total no. of faults
P1	2787	736	485	1630	3500	404	5534
P2	1793	258	241	835	1693	1051	3579
P3	2880	273	295	942	1603	1529	4074
P4	1637	265	158	1673	3093	1052	5818

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

Table 2: Module size distribution

LOC	P1	P2	P3	P4
≤ 5000	307	116	120	45
$5001 - 10000$	114	74	90	42
$10001 - 20000$	42	37	53	55
$20000 - 30000$	11	6	10	11
$30000 - 40000$	6	3	9	3
> 40001	5	5	13	2
Total	485	251	295	158

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.

For the purpose of this analysis, we searched in the database for all TRs reported during function test, system test and system integration test on software modules modified or impacted in the projects. Based on the TR answer code, we included in our analysis only TRs that need correction, and eliminated the duplicates. This was possible, since the TR answer code provides all this information and is easily accessible in the database. Observe that, besides all modules that are modified in a project, we also consider here those modules that are not modified but contain faults detected in a project. This is in accordance with the original studies [14] and [15]. However, in [10], where the same projects are considered, only modified modules with at least one fault are taken into account. This explains differences in project characteristics between this study and [10].

The outcome of the data collection in a project are the total and modified size and the number of faults for every software module that was modified or impacted in that project. The number of software modules, the total size, the size of modification, the number of faults in every verification phase and the total number of faults are given in Table 1 for each project. The module–size distribution is given in Table 2.

3. Probability Distributions

Let X be the random variable counting the number of faults in a software module. The empirical data from each of the four considered projects contains a sample for X. The goal is to study the probability distribution of X based on these four empirical samples.

We fit the empirical samples for X, as in [13, 14, 15], to the complementary cumulative distribution function (CCDF) of a probability distribution, also known as the survivor function and the reliability function. It is defined at the value x as $P(X \geq x)$, that is, the probability that the underlying random variable X takes the value greater than or equal to x. As explained in [13, 15], the CCDF is the key diagram in practice, equivalent to the Alberg diagram, which is the most common diagram used in software engineering. They give a precise relationship between the CCDF and Alberg diagram. For a detailed discussion of these issues see [16].

The probability density function (PDF) of a probability distribution will be denoted by $p(x)$. It is defined as the derivative of the cumulative distribution function (CDF), which amounts to the negative derivative of the CCDF. That is,

$$p(x) = -\frac{d}{dx}P(X \geq x). \tag{1}$$

We use PDF to observe the power-law in the tail of considered probability distribution. The exponent α of the power-law tail is defined by the condition

$$p(x) \propto x^{-\alpha} \tag{2}$$

for sufficiently large x. The power-law is best observed in the log-log scale, since the above condition becomes a straight line

$$\ln p(x) = \ln C - \alpha \ln x, \tag{3}$$

where $C > 0$ is the constant of proportionality.

In what follows we recall the formulas for PDF and CCDF of the probability distributions considered in this paper. This material is well-known, but we include it for completeness and to fix the notation for distribution parameters. The notation for parameters of some probability distributions below are adjusted compared to [15], so that the parameter α corresponds to the power-law exponent in the tail.

3.1. Weibull Distribution

The Weibull distribution is a continuous probability distribution supported in $x > 0$. Its CCDF is given as

$$P(X \geq x) = \exp\left(-\left(\frac{x}{\gamma}\right)^{\beta}\right), \quad x > 0, \tag{4}$$

where $\beta > 0$ and $\gamma > 0$. Then, the PDF is

$$p(x) = \frac{\beta}{\gamma} \cdot \left(\frac{x}{\gamma}\right)^{\beta-1} \cdot \exp\left(-\left(\frac{x}{\gamma}\right)^{\beta}\right), \quad x > 0. \tag{5}$$

In log-log scale the PDF becomes

$$\ln p(x) = \ln(\beta/\gamma^{\beta}) - (1-\beta)\ln x - \frac{1}{\gamma^{\beta}} e^{\beta \ln x}, \tag{6}$$

which contains a power-law with exponent $\alpha = 1 - \beta$, but having an extra exponential term in $\ln x$ which dominates the behavior.

3.2. Lognormal Distribution

The lognormal distribution is a continuous probability distribution supported in $x > 0$. It is characterized by the fact that its natural logarithm is the normal distribution. The mean μ and variance σ^2 of the normal distribution so obtained are the parameters of the lognormal distribution. Thus, its PDF function is given as

$$p(x) = \frac{1}{x\sqrt{2\pi\sigma^2}} \cdot \exp\left(-\frac{(\ln x - \mu)^2}{2\sigma^2}\right), \quad x > 0, \tag{7}$$

where μ is real and $\sigma > 0$. For the CCDF, we have to integrate

$$P(X \geq x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \int_x^{+\infty} \exp\left(-\frac{(\ln t - \mu)^2}{2\sigma^2}\right) \frac{dt}{t}. \tag{8}$$

Making the substitution for $\ln t$, and scaling to the standard normal distribution (with $\mu = 0$ and $\sigma = 1$) yields the CCDF in the form

$$P(X \geq x) = 1 - \Phi\left(\frac{\ln x - \mu}{\sigma}\right), \quad x > 0, \tag{9}$$

where Φ is the CDF of the standard normal distribution. In the log-log scale the lognormal PDF becomes

$$\ln p(x) = -\frac{\mu^2}{2\sigma^2} - \ln\sqrt{2\pi\sigma^2} - \left(1 - \frac{\mu}{\sigma^2}\right)\ln x - \frac{1}{2\sigma^2}(\ln x)^2 . \qquad (10)$$

This is close to a power-law distribution, having an extra quadratic term in $\ln x$. The coefficient of the linear term is the power-law exponent $\alpha = 1 - \mu/\sigma^2$.

3.3. Pareto Distribution

The Pareto distribution was originally introduced in [17] to describe the distribution of income and wealth in a society. It is a continuous probability distribution which is mainly used to fit the tails of other distributions. In particular, its support is in $x \geq x_m$, where x_m is certain positive value of the random variable, usually the beginning of the tail. The Pareto CCDF for $x > 0$ is given as

$$P(X \geq x) = \begin{cases} 1, & \text{for } 0 < x < x_m, \\ (x_m/x)^\beta, & \text{for } x \geq x_m, \end{cases} \qquad (11)$$

and the PDF

$$p(x) = \begin{cases} 0, & \text{for } 0 < x < x_m, \\ \beta x_m^\beta \cdot x^{-(1+\beta)}, & \text{for } x \geq x_m, \end{cases} \qquad (12)$$

where $\beta > 0$. Clearly, this is a power-law distribution with the exponent $\alpha = 1 + \beta$.

We would like to mention here a possible pitfall when fitting the Pareto distribution. Using incorrectly the function $(x_m/x)^\beta$ for all $x > 0$, instead of the CCDF given by (11), would lead to significantly lower goodness-of-fit (see Section 4). The problem is that this function is not a CCDF of a probability distribution, since it grows to infinity as x approaches zero. In particular, the error of estimate would increase significantly for observations near zero.

Another possibility is to restrict the fitting of the Pareto distribution only to tail, and discard the body of the distribution. In this approach, the parameter x_m is estimated as the beginning of the tail, and then the fitting is performed only in the region $x \geq x_m$. For more details see [18]. This approach is taken in [13], *but not in* [15], which we replicate here. Fitting only the tail certainly increases the goodness-of-fit, as it considers only the

region $x \geq x_m$, but loses all information about the body of the distribution, i.e., $x < x_m$. However, as explained in [13], it is sufficient to make estimates of some distribution parameters in future releases, such as the maximal number of faults in a software unit.

We are making a close replication of [15], in which the approach is to fit the faults data to the CCDF given by (11) in the full range of a distribution, i.e. not fitting only the tail. Hence, our approach is the same as in [15].

3.4. Double Pareto Distribution

The idea behind the double Pareto class of probability distributions is to find a distribution that fits well the body of empirical distribution and at the same time the power-law tail. As in [15], we use the form of the double Pareto distribution developed in [19]. Note that the definition of a power-tail exponent in [19] is a bit different than what we use in this paper. We adjusted the notation for the double Pareto distribution parameters accordingly.

This double Pareto distribution is a continuous probability distribution supported in $0 < x \leq x_M$, where x_M is the maximal possible value of the underlying random variable. The double Pareto CCDF for $x > 0$ is given as

$$P(X \geq x) = \begin{cases} 1 - \left[\frac{1+(x_M/t)^{-\beta}}{1+(x/t)^{-\beta}}\right]^{\gamma/\beta}, & \text{for } 0 < x \leq x_M, \\ 0, & \text{for } x > x_M, \end{cases} \tag{13}$$

and the PDF as

$$p(x) = \begin{cases} \frac{\gamma}{t} \cdot \frac{\left[1+(x_M/t)^{-\beta}\right]^{\gamma/\beta}}{\left[1+(x/t)^{-\beta}\right]^{1+\gamma/\beta}} \cdot \left(\frac{x}{t}\right)^{-(1+\beta)}, & \text{for } 0 < x \leq x_M, \\ 0, & \text{for } x > x_M, \end{cases} \tag{14}$$

where $0 < t < x_M$, $\beta > 0$ and $\gamma > 0$.

The parameter t determines the crossover point $x = t$, at which the distribution changes its behavior from the power-law in the body to the power-law in the tail.

The power-law in the tail is governed by the term $(x/t)^{-(1+\beta)}$. Indeed, for x sufficiently large (not exceeding x_M) with respect to t, we have x/t large, so that the fraction appearing in (14) is approximately equal to 1. Thus, the power-law exponent in the tail is $\alpha = 1 + \beta$.

For the power-law in the body of the distribution, we consider x sufficiently small with respect to t, so that x/t is close to zero. For such x

the asymptotic behavior of the fraction in equation (14) is proportional to $(x/t)^{\beta+\gamma}$. Multiplying by the last term $(x/t)^{-(1+\beta)}$ shows that the body follows the power-law with exponent $\gamma - 1$.

3.5. Yule-Simon Distribution

The Yule-Simon distribution is the outcome of the so-called preferential attachment model, developed to explain the power-law in tails of certain empirical distributions. Originally it was developed in [23] to describe distributions of species and genera in the theory of evolution, and later applied in [21] to the distribution of words in books. It is a discrete probability distribution supported in the non-negative integers.

The Yule-Simon PDF is given as

$$p_k = p_0 \cdot \frac{B(c+k, \alpha)}{B(c, \alpha)}, \quad k = 0, 1, 2, \ldots, \tag{15}$$

where $c > 0$, $\alpha > 0$, and $0 < p_0 \leq 1$ is the probability that the random variable takes the value zero. Here $B(a, b)$ denotes the Beta Function, which can be expressed in terms of the Gamma Function as

$$B(a, b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}. \tag{16}$$

We only need the values of the Beta Function for $a, b > 0$, and in that range it is well-defined, that is, there are no poles, and also no zeros. The parameter p_0 is in fact determined by c and α by the condition that the total probability $\sum_{j=0}^{\infty} p_j$ should be 1. More precisely,

$$p_0 = \left(\sum_{j=0}^{\infty} \frac{B(c+j, \alpha)}{B(c, \alpha)} \right)^{-1}, \tag{17}$$

so that the Yule-Simon PDF is

$$p_k = \frac{B(c+k, \alpha)}{B(c, \alpha)} \cdot \left(\sum_{j=0}^{\infty} \frac{B(c+j, \alpha)}{B(c, \alpha)} \right)^{-1}, \quad k = 0, 1, 2, \ldots \tag{18}$$

The Yule-Simon CCDF is then given as

$$P(X \geq x) \;=\; \sum_{k \geq x} p_k$$

$$= \quad p_0 \sum_{k \geq x} \frac{B(c+k,\alpha)}{B(c,\alpha)} \tag{19}$$

$$= \quad \left(\sum_{j=0}^{\infty} \frac{B(c+j,\alpha)}{B(c,\alpha)} \right)^{-1} \cdot \sum_{k \geq x} \frac{B(c+k,\alpha)}{B(c,\alpha)}.$$

It can be shown that the parameter α is the power-law exponent in the tail of the Yule-Simon distribution. Indeed, taking the logarithm of equation (15), and writing the Beta Function as in (16), we obtain

$$\ln p_k = C + \ln \Gamma(c+k) - \ln \Gamma(c+k+\alpha), \tag{20}$$

where $C = \ln p_0 + \ln \Gamma(\alpha) - \ln B(c,\alpha)$ is a constant. Applying Stirling's approximation for the logarithm of the Gamma Function gives, for k sufficiently large, that

$$\ln \Gamma(c+k) - \ln \Gamma(c+k+\alpha) \approx$$

$$\approx \quad \alpha - \ln \left(1 + \frac{\alpha}{c+k}\right)^{c+k-1/2} - \alpha \ln(c+k+\alpha)$$

$$\approx \quad -\alpha \ln k. \tag{21}$$

We used the fact that the second term in the first line tends to $-\alpha$ as $k \to \infty$, while the asymptotic of the last term as $k \to \infty$ coincides with that of $-\alpha \ln k$. Thus, we obtained

$$\ln p_k \approx C - \alpha \ln k, \tag{22}$$

for large k, showing that α is the power-law exponent in the tail.

4. Non-linear Regression Fit

For the non-linear regression fit [22] of the empirical CCDF for the random variable X counting the number of faults in software units, we used the *Matlab* curve fitting tool, which is based on the least squares method for fitting. Except for the Yule-Simon distribution, we discard the software units with no faults (as in [15]), since they do not fit zero values. As already explained in Sect. 3.3, we fit the Pareto distribution, following [15], to the full data sample, and *not only the tail*. The obtained CCDFs for double Pareto, lognormal, Weibull and Pareto distributions are presented in log-log

Table 3: Distribution parameters and goodness-of-fit for non-linear regression

Project	Distribution	Parameters	R^2	S_e
P1	Double Pareto	$\beta = 0.7941$ $\gamma = 92.5331$ $x_M = 142.4923$ $t = 0.0126$	0.99901	0.00654
	Weibull	$\gamma = 12.6963$ $\beta = 0.7216$	0.98019	0.02926
	Lognormal	$\mu = 2.0034$ $\sigma = 1.3623$	0.99554	0.01389
	Pareto	$x_m = 2.4695$ $\beta = 0.7436$	0.97077	0.03553
	Yule-Simon (p_0 from data)	$c = 2.0997$ $\alpha = 1.8212$	0.98179	0.02376
	Yule-Simon (p_0 not from data)	$c = 3.5498$ $\alpha = 1.9951$	0.99130	0.01643
P2	Double Pareto	$\beta = 0.7694$ $\gamma = 27.3994$ $x_M = 124.8392$ $t = 0.0649$	0.99759	0.01147
	Weibull	$\gamma = 15.4349$ $\beta = 0.7811$	0.98569	0.02797
	Lognormal	$\mu = 2.2221$ $\sigma = 1.3033$	0.99610	0.01460
	Pareto	$x_m = 2.8309$ $\beta = 0.7036$	0.96176	0.04572
	Yule-Simon (p_0 from data)	$c = 4.1133$ $\alpha = 1.9281$	0.98159	0.02883
	Yule-Simon (p_0 not from data)	$c = 7.8705$ $\alpha = 2.2364$	0.99285	0.01796
P3	Double Pareto	$\beta = 1.5429$ $\gamma = 1.5661$ $x_M = 8377.4032$ $t = 8.5373$	0.99515	0.01789
	Weibull	$\gamma = 13.2804$ $\beta = 0.9589$	0.98436	0.03214
	Lognormal	$\mu = 2.1531$ $\sigma = 1.0944$	0.99444	0.01916
	Pareto	$x_m = 2.8945$ $\beta = 0.7898$	0.95798	0.05268
	Yule-Simon (p_0 from data)	$c = 3.5853$ $\alpha = 2.0078$	0.96888	0.03945
	Yule-Simon (p_0 not from data)	$c = 9.6442$ $\alpha = 2.6396$	0.98967	0.02273
P4	Double Pareto	$\beta = 0.9672$ $\gamma = 2.7293$ $x_M = 2942.2168$ $t = 3.3279$	0.99703	0.01347
	Weibull	$\gamma = 22.2033$ $\beta = 0.7237$	0.97342	0.04029
	Lognormal	$\mu = 2.5494$ $\sigma = 1.4433$	0.99295	0.02075
	Pareto	$x_m = 3.6015$ $\beta = 0.6280$	0.97523	0.03810
	Yule-Simon (p_0 from data)	$c = 33.1215$ $\alpha = 2.9207$	0.98353	0.03100
	Yule-Simon (p_0 not from data)	$c = 12.5872$ $\alpha = 2.1426$	0.99709	0.01302

scale in Fig. 1 for project P1, Fig. 2 for project P2, Fig. 3 for project P3, Fig. 4 for project P4, and the parameter values are listed in Table 3.

Fitting the Yule-Simon distribution is done in two ways. The first way is to estimate p_0 a priori as the relative frequency of software units with no faults in the sample. This approach was taken in [15]. The second way is to express p_0 in terms of parameters c and α, as in equation (17), and then fit to the CCDF given by formula (19). The obtained CCDFs for Yule-Simon distributions are presented in log-log scale in Fig. 5 for project P1, Fig. 6 for project P2, Fig. 7 for project P3, Fig. 8 for project P4, and the values of distribution parameters in Table 3.

As a measure for goodness-of-fit we compute, as in [14], the coefficient of determination R^2 adjusted for the degrees of freedom of the fitting model, and the standard error of estimate S_{e}. Their values are also given in Table 3.

More precisely, let

$$SS_{\mathrm{err}} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{23}$$

be the sum square error, where y_i and $\hat{y}_i$ are the actual and fitted values of ith observation, and n is the number of observations. The least square method used in non-linear regression actually determines the unknown parameters by minimizing this value. Let

$$SS_{\mathrm{tot}} = \sum_{i=1}^{n} (y_i - \bar{y})^2 \tag{24}$$

be the total sum of squares, where $\bar{y}$ is the mean of the observed data.

The adjusted coefficient of determination R^2 is defined as

$$R^2 = 1 - \frac{(n-1)SS_{\mathrm{err}}}{(n-m-1)SS_{\mathrm{tot}}}, \tag{25}$$

where m is the number of parameters in the fitting function.

The standard error of estimate S_{e} is defined as the square root of the sum square error divided by the degrees of freedom. That is,

$$S_{\mathrm{e}} = \sqrt{\frac{SS_{\mathrm{err}}}{n-m}}. \tag{26}$$

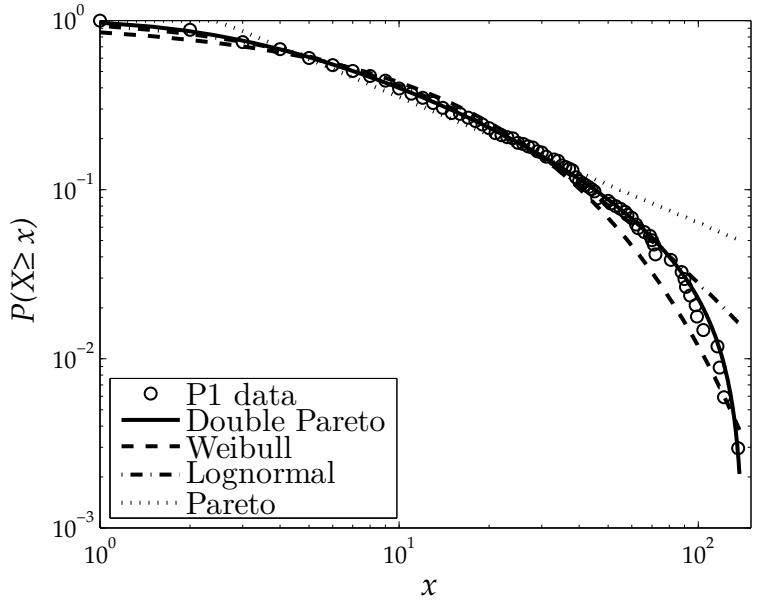

Figure 1: Non-linear regression fits to the double Pareto, Weibull, Lognormal and Pareto CCDF for random variable X counting testing faults in **project P1**

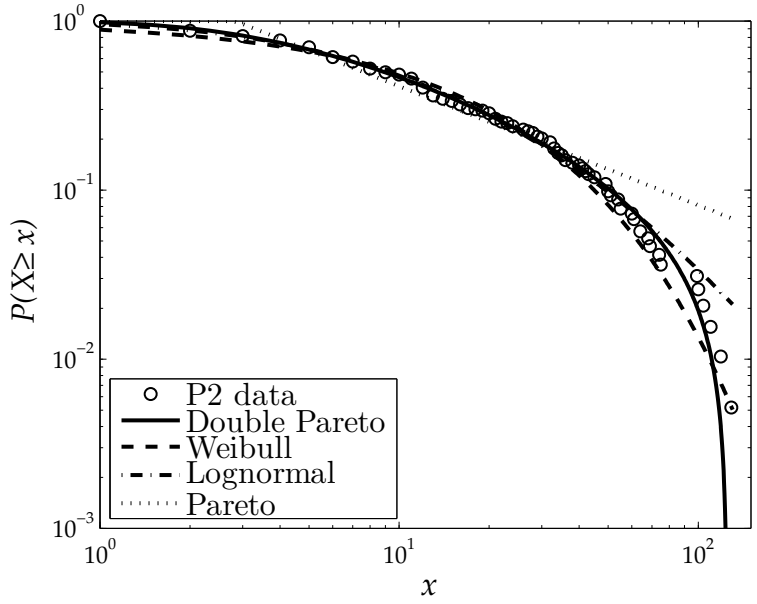

Figure 2: Non-linear regression fits to the double Pareto, Weibull, Lognormal and Pareto CCDF for random variable X counting testing faults in **project P2**

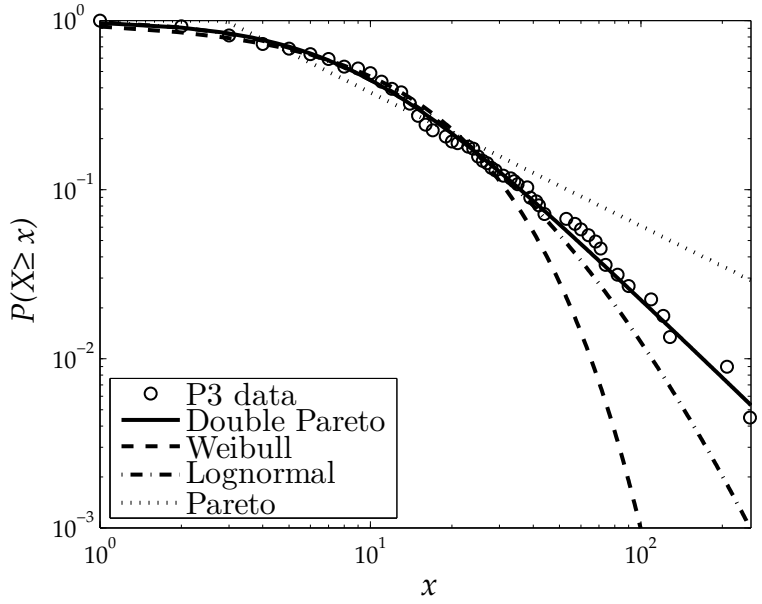

Figure 3: Non-linear regression fits to the double Pareto, Weibull, Lognormal and Pareto CCDF for random variable X counting testing faults in **project P3**

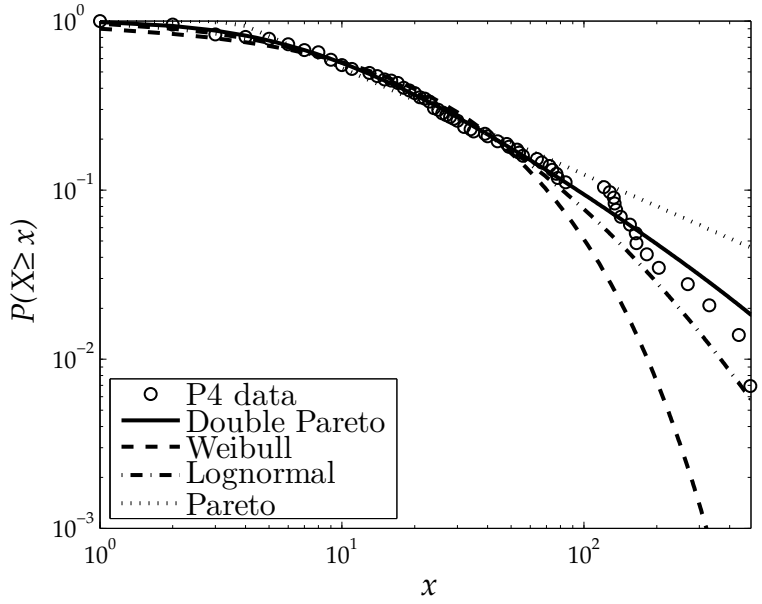

Figure 4: Non-linear regression fits to the double Pareto, Weibull, Lognormal and Pareto CCDF for random variable X counting testing faults in **project P4**

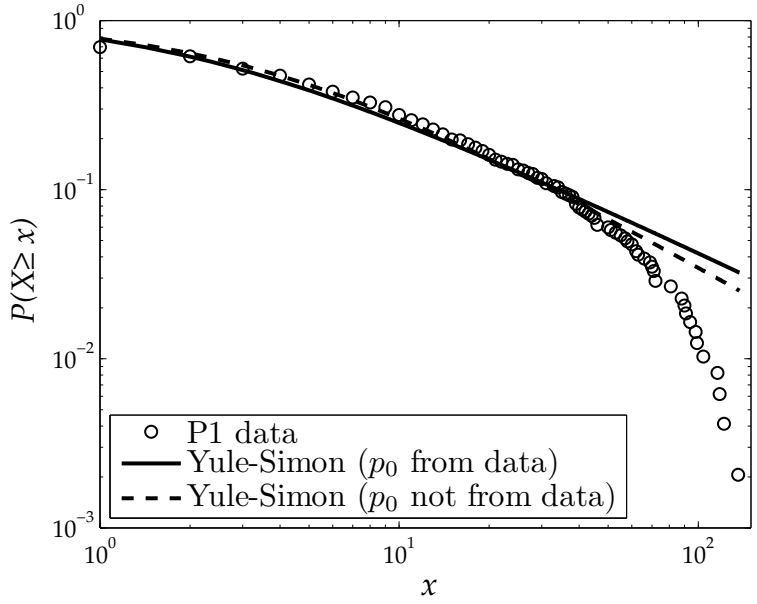

Figure 5: Non-linear regression fits to the Yule-Simon CCDF with and without a priori estimate for p_0 from data for random variable X counting testing faults in **project P1**

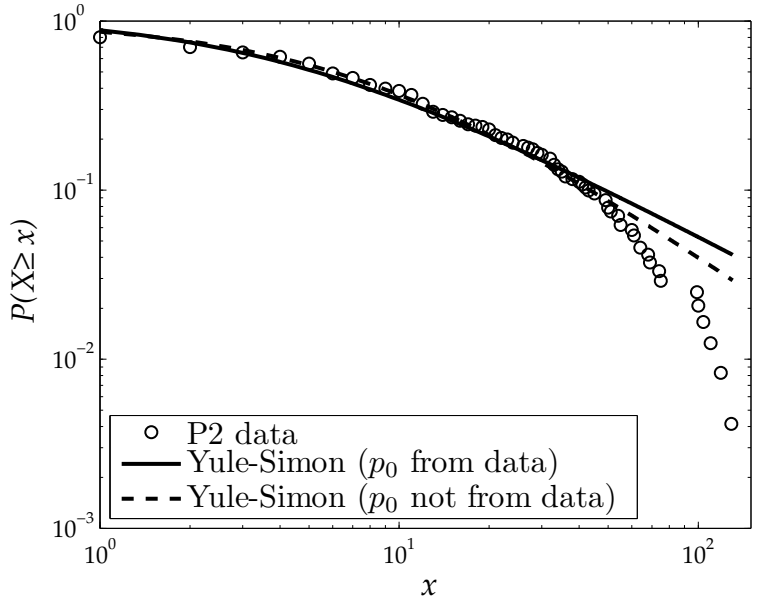

Figure 6: Non-linear regression fits to the Yule-Simon CCDF with and without a priori estimate for p_0 from data for random variable X counting testing faults in **project P2**

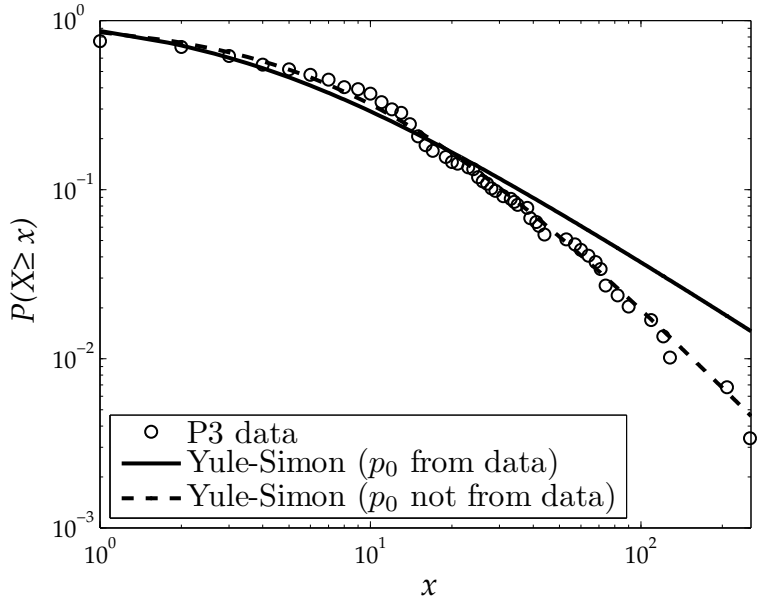

Figure 7: Non-linear regression fits to the Yule-Simon CCDF with and without a priori estimate for p_0 from data for random variable X counting testing faults in **project P3**

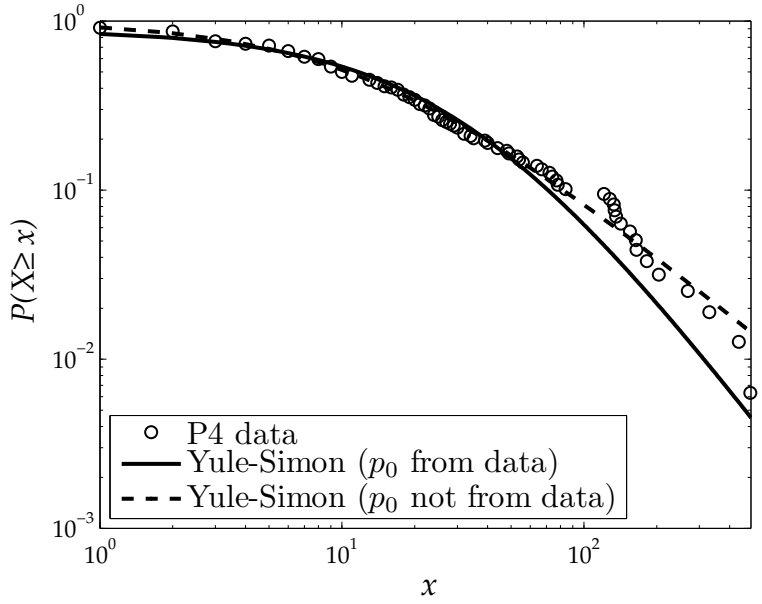

Figure 8: Non-linear regression fits to the Yule-Simon CCDF with and without a priori estimate for p_0 from data for random variable X counting testing faults in **project P4**

Comparing the goodness-of-fit reported in Table 3 shows first of all that the R^2 values for all distributions are very close. The difference of the coefficient R^2 between the best and the worst fit for each project does not exceed 0.04. However, there is a certain tendency that can be observed in all the projects.

The double Pareto distribution has the highest R^2 in projects P1, P2 and P3, while in P4 the R^2 for the Yule-Simon distribution (with p_0 not estimated from the data) is better, but only for negligible 0.00005. Hence, we conclude that the most appropriate probability distribution to fit our empirical data is the double Pareto distribution.

Despite the Yule-Simon outperforming slightly the lognormal distribution in P4 (for 0.004), the lognormal distribution has consistently higher values of R^2 in P1, P2 and P3. Hence, it is more likely that the lognormal is the second best choice for the fit, followed by the Yule-Simon distribution (with p_0 not estimated from the data).

Only then comes the Weibull distribution, with higher values of R^2 than for the Pareto distribution in P1, P2 and P3, although slightly outperformed by the Pareto distribution in P4 (for less than 0.002).

These results are *not consistent* with those of [15], which point to the Yule-Simon distribution as significantly better fit than others. The reason for this inconsistency is certainly a very different context of this study compared to the previous work. It is also interesting to observe that the Weibull and Pareto distributions are much closer than obtained in [14].

5. Discussion and future work

The detailed discussion in Section 4 of the goodness-of-fit using the nonlinear regression for the five considered probability distributions is summarized in Table 4. In the same table we also summarize the results of [15] and [14]. This is a bit simplified view of the results, as there is no quantitative information.

Widely used empirical principles on fault distributions, originally studied in [8], and in further replicated studies [9] and [10], are related to the Pareto principle of fault distribution, persistence of faults across the verification phases, effects of module size and complexity on fault proneness. Although the empirical Pareto principle has been confirmed in number of different contexts and studies it does not imply that the underlying statistical fault distributions are equal. In fact, the results in Table 4 show that,

Table 4: 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	Concas et al. [15]	Zhang [14]
1	Double Pareto	Yule-Simon	Weibull
2	Lognormal	Double Pareto	Pareto
3	Yule-Simon	Lognormal	—
4	Weibull	Weibull	—
5	Pareto	—	—

with the exception of the Yule-Simon distribution, the order of the probability distributions according to the goodness-of-fit coincides in this study and in [15] and [14]. Hence, the main difference between our study and [15] is the position of the Yule-Simon distribution. In the study [15] it is significantly better than all others, while in our study it is only at the third position, although close to the lognormal distribution at the second. However, it is hard to say what exactly is the reason for the inconsistency, as there are many possible influencing factors in such different contexts. It is also possible that the generative model of preferential attachment for the Yule-Simon distribution does not apply in our context, but works well in the context of the study by [15]. All these are interesting problems for further research.

One possible explanation for the difference is that the systems may be in a different stage of equilibration. The software system may be considered as a discrete complex system and studied as a physical system. It is in perfect equilibrium when there are no new faults reported. At that stage the discrete conservation laws may be imposed, just as in the continuous physical systems (e.g. conservation of energy). The theory of conservation laws for software systems was developed by Hatton in [24]. From these basic principles, he predicts the behavior of a complex software system that is close to equilibrium, in particular, the module size and fault distributions. If the systems considered in this study and those in [15] and [14] are not in the same stage of equilibration, it could be the reason for different fault distributions.

To determine how close to equilibrium a given software system is, one may use the reliability growth models [25]. The system is in equilibrium, when the reliability curve stabilizes. However, this should be taken with caution, since every verification phase, and even every verification technique has its own point of equilibrium, determined by the reliability curve of faults detected in a specific verification phase or by a specific technique. So the full

system is in equilibrium only after all verification stages and techniques are stabilized. Hence, an important line of further research is to study the fault distributions in each verification phase. On the other hand, if a system is produced evolutionary in a sequence of releases as the system in this study, it would be interesting to observe the existence of the equilibrium point from the system evolution perspective.

Another factor that could be the reason for different results here compared to the previous studies is the module size distribution of the observed system. The connection between fault distribution and module sizes was already studied in the pioneering work [26]. The linear correlation between faults and size is also one of the hypothesis considered in [8, 9, 10], but it was only partially confirmed. The size distribution for software systems close to equilibrium, as a consequence of the conservation laws was also studied in [24]. As explained in previous section 2.1, the projects in this study are the same as in [10], and a rough distribution of module sizes is given in Table 2. It turns out that the size distribution in project P4 is quite different than in other projects considered here. It has significantly higher portion of larger modules (10–20 kLOC). On the other hand, observe that the order of best fit fault distributions for project P4 is a bit different than for other projects, close to the order reported in [15]. In particular, the Yule-Simon distribution fits the fault data for project P4 equally well as the double Pareto distribution. It could be possible that the generative model of preferential attachment giving the Yule-Simon distribution depends on the size distribution. Unfortunately, only the total system size is reported in [15] and [14], so that we cannot verify this conjecture.

6. Threats to validity

Internal validity refers to a proper demonstration of a casual relation between two variables in a study. In our case, there are two possible threats to internal validity, which are the same as in the original studies [14] and [15]. As we follow the original studies in using the non-linear regression to fit the fault data, the first threat to internal validity is a possibility that the obtained distribution fits do not reflect the underlying distribution of faults and are just obtained by chance. Very high R^2 value assures that this is not the case. Another possibility to verify this would be distribution fitting using the maximum likelihood method, combined with Monte Carlo simulation for evaluating the distribution fits. The second threat to internal

validity is that, even though the distribution is correct, it is not a consequence of its generative model, when such model exists. This is a difficult question, which was neither pursued here, nor in the original studies [14] and [15], except that in the latter the very existence of a generative model for the Yule-Simon distribution was taken as an additional argument in its favor. The same holds for the double Pareto distribution, which is the best fit in this study.

The construct validity refers to whether the particular properties of the samples in a study are measures of general constructs. The question is to which extent the sample projects studied in this paper represent the development project of the considered system. Reducing the threat to construct validity is the reason for using four projects on the same system. Note that during this four projects the development organization have changed as explained in the Section 2.2 although the real faults for each release, represented in figures of Section 4, seem to have visually almost the same behavior. It turns out that the double Pareto distribution is, indeed, the best fit in all the projects. However, the distribution parameters are varying, so that the complete understanding of the construct would require further estimation models for parameter changes. This difficult issue is out of scope of this work. It is possible that the reason is in a different size distribution. The results may also be influenced by the differences in data collection. The data collection for this study is described in Section 2. It relies on the data from trouble reports, which is very precise and linked to the system module at the moment of fault detection and during correction. The data collection for the Eclipse system in the previous study relies on the open bug report database, in which it is not so easy to relate faults with modules, eliminate duplications and false trouble reports.

External validity refers to the generality of the results across different settings including those not considered in a study. This replication study is an attempt to generalize the findings of the original studies on fault distributions [15] and [14]. In this replication study we analyzed the fault distributions in an industrial context, which is quite different from the open source development environment in the original studies. Thus, addressing the threat to the external validity is the main focus of this paper. It turns out that the results are different than in the original studies, and possible reasons for that are discussed in Sect. 5.

7. Conclusion

Building a software engineering theory of fault distributions has recently received significant attention. So far, we are aware of empirical principles regarding fault distributions that have been empirically confirmed in different environments. However, knowing the appropriate statistical fault distribution would enable more systematic approach for software engineering management practice. Then we could move a step forward in the software engineering research. One direction would be to a more general level, that is, looking for the underlying processes that generate distributions and how they influence the statistical fault distributions, and thus, finally start to build systematic theory of fault distributions. Another direction would be towards prediction of fault distribution early in development projects, that is, constructing parameter estimation models for the underlying distribution.

This paper contributes to the study of software fault distributions by replicating the studies [15] and [14] in the context of a commercial large-scale complex software system developed by a globally distributed organization using strictly defined development processes. This is very different context than the open source projects considered in the original studies, and it turns out that the results are different. A high-level discussion of some possible factors influencing different fault distribution in this and original studies is provided in Section 5.

However, to determine exactly the factors controlling the fault distribution in complex software systems, it is necessary to gather more information in similar and different contexts, and if possible conduct controlled experiments with the aim to study the fault distributions. We hope that this study will become a source of replications leading to better understanding of the probability distribution of faults in complex software systems. This would provide invaluable insight, and enable more systematic approach and refinement of the empirical principles regarding fault distributions used in the software development practice, as well as possible predictions of system behavior for future releases.

Acknowledgment

The work presented in this paper is supported by the University of Rijeka research grant 13.09.2.2.16.

References

[1] J. Juran, Quality control handbook, McGraw-Hill, New York, 1974.

[2] N. Ohlsson, H. Alberg, Predicting fault-prone software modules in telephone switches, IEEE Trans. Softw. Eng. 22 (1996) no. 12, 886–894.

[3] B. Compton, C. Withrow, Prediction and control of ADA software defects, J. Syst. Softw. 12 (1990) no. 3, 199–207.

[4] G. Denaro, M. Pezzè An empirical evaluation of fault-proneness models, in: Proc. 24th Internat. Conf. on Softw. Eng. (ICSE '02), pp. 241–251.

[5] M. English, C. Exton, I. Rigon, B. Cleary, Fault detection and prediction in an open-source software project, in: Proc. 5th Internat. Conf. on Predictor Models in Softw. Eng. (PROMISE '09), pp. 17:1–17:11.

[6] M. Kaâniche, K. Kanoun, Reliability of a commercial telecommunications system, in: Proc. 7th Internat. Symp. on Softw. Reliability Eng. (ISSRE '96), pp. 207–212.

[7] J. Munson, T. Khoshgoftaar, The detection of fault-prone programs, IEEE Trans. Softw. Eng. 18 (1992) no. 5, 423–433.

[8] N. Fenton, N. Ohlsson, Quantitative analysis of faults and failures in a complex software system, IEEE Trans. Softw. Eng. 26 (2000) no. 8, 797–814.

[9] C. Andersson, P. Runeson, A replicated quantitative analysis of fault distributions in complex software systems, IEEE Trans. Softw. Eng. 33 (2007) no. 5, 273–286.

[10] T. Galinac Grbac, P. Runeson, D. Huljenić, A second replicated quantitative analysis of fault distributions in complex software systems, IEEE Trans. Softw. Eng. 39 (2013) no. 4, 462–476.

[11] G. Concas, M. Marchesi, S. Pinna, N. Serra, Power-laws in a large object-oriented software system, IEEE Trans. Softw. Eng. 33 (2007) no. 10, 687–708.

[12] P. Louridas, D. Spinellis, V. Vlachos, Power laws in software, ACM Trans. Softw. Eng. and Methodology 18 (2008) no. 1, article no. 2.

[13] G. Concas, M. Marchesi, A. Murgia, R. Tonelli, An empirical study of social network metrics in object-oriented software, Advances in Softw. Eng., vol. 2010, Article ID 729826, 21 pages, 2010.

[14] H. Zhang, On the distribution of software faults, IEEE Trans. Softw. Eng. 34 (2008) no. 2, 301–302.

[15] G. Concas, M. Marchesi, A. Murgia, R. Tonelli, I. Turnu, On the distribution of bugs in the Eclipse system, IEEE Trans. Softw. Eng. 37 (2011) no. 6, 872–877.

[16] M. E. J. Newman, Power laws, Pareto distributions and Zipf's law, Contemporary Physics 46 (2005), no. 5, 323–351.

[17] V. Pareto, Cours d'économie politique, F. Rouge, Lausanne, 1897.

[18] A. Clauset, C. R. Shalizi, M. E. J. Newman, Power-law distributions in empirical data, SIAM Review 51 (2009) no. 4, 661–703.

[19] C. Stark, N. Hovius, The characterization of landslide size distributions, Geophysical Research Letters 28 (2001) no. 6, 1091–1094.

[20] G. Yule, A mathematical theory of evolution based on the conclusions of Dr. J. C. Willis, Philosophical Trans. Royal Soc. of London Series B 213 (1925), 21–87.

[21] H. Simon, On a class of skew distribution functions, Biometrika 42 (1955), 425–440.

[22] D. Bates, D. Watts, Nonlinear regression analysis and its applications, John Wiley & Sons, New York, 1988.

[23] G. Yule, A mathematical theory of evolution based on the conclusions of Dr. J. C. Willis, Philosophical Trans. Royal Soc. of London Series B 213 (1925), 21–87.

[24] L. Hatton, Power-Law Distributions of Component Size in General Software Systems IEEE Trans. Softw. Eng. 35 (2009) no. 4, 566–572.

[25] Lyu MR (ed) Handbook of software reliability engineering, McGraw-Hill, New York, 1996.

[26] V.R. Basili and B.T. Perricone, Software Errors and Complexity: an Empirical Investigation, Commun. ACM, 27 (1984), no. 1, 42–52.