Is My Project's Truck Factor Low? Theoretical and Empirical Considerations About the Truck Factor Threshold

Marco Torchiano
Politecnico di Torino
Torino, Italy
marco.torchiano@polito.it

Filippo Ricca
DISI, Universitá di Genova
Genova, Italy
filippo.ricca@disi.unige.it

Alessandro Marchetto
FBK-irst
Trento, Italy
marchetto@fbk.eu

ABSTRACT

The Truck Factor is a simple way, proposed by the agile community, to measure the system's knowledge distribution in a team of developers. It can be used to highlight potential project problems due to the inadequate distribution of the system knowledge. Notwithstanding its relevance, only few studies investigated the Truck Factor and proposed ways to efficiently measure, evaluate and use it. In particular, the effective use of the Truck Factor is limited by the lack of reliable thresholds.

In this preliminary paper, we present a theoretical model concerning the Truck Factor and, in particular, we investigate its use to define the maximum achievable Truck Factor value in a project. The relevance of such a value concerns the definition of a reliable threshold for the Truck Factor. Furthermore in the paper, we document an experiment in which we apply the proposed model to real software projects with the aim of comparing the maximum achievable value of the Truck Factor with the unique threshold proposed in literature. The preliminary outcome we achieved shows that the existing threshold has some limitations and problems.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*Process metrics*

General Terms

Management, Theory, Experimentation

Keywords

Truck factor, Code ownership, OSS

1. INTRODUCTION

The "collective code ownership" of a system is the way proposed by the Extreme Programming (XP) [3] community to distribute the system knowledge in the team. Distributing such a knowledge is particularly relevant to avoid situations in which all the knowledge of the system is in the hands of few developers only. Such contributors, who are the only ones who know certain critical parts of a system are often called Heroes [9]. It is well recognized that the presence of Heroes and, more in general, a low amount of spread of knowledge within a team can increase the risk of project failure, especially whether some developers (e.g., the Heroes) leave the project, e.g., for another company, project or for a vacation.

The Truck Factor[1] (TF) is a simple measurement proposed several years ago to have an idea about the code knowledge distribution within a team of developers. More precisely, the TF of a project is defined as "*the number of developers on a team who have to be hit with a truck (i.e., to go on vacation, to become ill, or to leave the company for another) before the project is in serious trouble*". Hence, ideally, to avoid potential problems the TF of a project should be as high as possible (as supposed by the XP principle of collective code ownership [3]).

Despite its strong and evident relevance for team leaders and project managers (a low TF should alert them!), only few works/studies tried to understand how to measure and use the TF in practice. In the best of our knowledge, Zazworka et al. [11] was the first and only who proposed a way to compute the TF using information about the code ownership coming from code repositories.

In particular, the use of the TF is limited also for the absence of an reliable threshold able to indicate that a project run the risk to fail because the system knowledge is in the hands of few developers. In other words, what is almost completely missing in the literature is a reliable threshold under which the responsible manager of the project should be alerted. It is well-recognized by the scientific community that the effective use of metrics is limited by the lack of meaningful and reliable thresholds [1] and this is particularly true for the TF. Indeed, without a clear and shared threshold (or a way to compute it) is really difficult to interpret and use a given metric. So far, to the best of our knowledge, Siddharta Govindaray[2] has been the only one that tried to identify/propose, based on his experience, a threshold able to identify a low TF. Defining a metric threshold based on expert opinions and a limited number of observations is a

[1] http://www.agileadvice.com/archives/2005/05/truck_factor.html

[2] He is the founder of Silver Stripe Software Pvt Ltd, whose flagship product Silver Catalyst is a tool for Agile Project Management

good alternative when no other ways (e.g., empirical or theoretical) have been investigated.

In this paper, we present a theoretical model on the TF including several variables (e.g., Team size) to identify the maximum achievable TF value. This value concerns the threshold selection for the TF, thus it is a meaningful value for this measurement. Moreover, we present an experiment based on real open-source projects that we performed to evaluate the model and compare the maximum achievable TF value with the Govindaray threshold. The achieved results show that the Govindaray proposal seems to have some limitations. In particular, it seems to be reasonable only with particular values of the analyzed variables. This outcome suggests the need of further investigation and experimentation in this context.

This paper is organized as follows. Section 2 summarizes relevant notions and related works. Section 3 introduces the theoretical model for the TF. Section 4 presents the data used in the experiment and collected from real projects while Section 5 analyzes these data and uses them to compare the maximum achievable TF value with the Govindaray threshold. Finally Section 6 discusses the results and Section 7 concludes the paper.

2. BACKGROUND

2.1 Code Ownership

The code ownership [7] identifies the policy used by team managers to control the responsibility of developers with respect to the system code development and management.

To monitor, analyze and thus change the code ownership policy during the software life-cycle, adequate information and tools are needed. The major source of information for computing and evaluating code ownership is, often, the code repository and the activities (i.e., *commits*) performed by developers (i.e., *committers*) on code files that have been tracked in log files (e.g., SVN, CVS). These log files mainly contain the list of activities performed by developers on the code and, for each activity, some additional information, e.g., the version number of the system in which the activity has been executed. Different approaches have been proposed to measure the code ownership by analyzing these repositories. For instance, Weyuker et al. [10] propose to compute the ownership at file level while Girba et al. [6] propose a measure based on the percentage of source code lines modified by a specific developer.

Several works (e.g., [4]) analyze code ownership and developer activities tracked by code repositories for automatically identifying experts for a particular portion or aspect of a system. These recommender systems are often used to assign a developer (i.e., the more suitable) to a change request. These tools make the implicit assumption that a code change performed by a developer indicates the knowledge of the involved system portion for that developer. Fritz et al. [4] perform an analysis of developer activities comparing them with the actual knowledge of code achieved by developers. Their analysis confirms that the frequency of the interaction of developers with the code mainly indicates which portions of the system/code a given developer known.

The assumption underlying such recommender systems concerning the connection between code ownership and developer activities has been also adopted in our work, as well as by Zazworka et al. [11].

2.2 Computing TF

To the best of our knowledge, the unique approach able to compute the TF has been proposed by Zazworka et al. [11]. They applied that approach to five small projects written by students for understanding whether (or not) XP projects have higher TF than non-XP projects. That preliminary experiment provides the first evidence that non-XP projects have significant lower (i.e., worse) TFs than XP projects.

The key idea underlying their approach is that a file in the repository is considered collectively owned by all the developers who worked on it. The assumption is that developers who edited the file have knowledge about it. In this way, for each file 'f' of a project, the *developer set* for 'f' consists of the set of developers that did at least one commit on 'f'.

To better explain their proposal we consider some files of Jfreechart (one of the projects used in our experimental study) as an example. Table 1 and Table 2 show respectively details about the four considered files (i.e., SWTUtils.java, DefaultKeyedValues.java, ChartColor.java and PolarChartPanel.java) and their TF computation.

File (.java)	Dev.Set
SWTUtils	{mungady, nenry}
DefaultKeyedValues	{mungady, taqua}
ChartColor	{mungady}
PolarChartPanel	{mungady}

Table 1: Jfreechart example

For example, columns "File" and "Dev.Set" of Table 1 show that the set of developers that did at least one commit on the file "SWTUtils" is {mungady, nenry}. The first rows of Table 2 present the possible sets of developers potentially "hit by a truck". In particular, the second row of that Table indicates how many developers would be missing (from 0 to 3), while the third row details exactly who. In each Table cell, the sign "+" means that the remaining developers (i.e., those developers in the Dev.Set that have not been "hit by a truck") have the knowledge of the corresponding file reported in the first column. Instead, the sign "-" means that the developers "hit by a truck" were the only to know that file. The penultimate row of the Table reports the preserved file coverage (or residual knowledge) measured as percentage, precisely, the number of files known by the remaining developers divided by the total number of files in the project * 100. Finally, the last row reports the minimum file coverage per number of missing developers. Let us consider, for example, the scenario in which the developer *mungady* leaves the project (see the third column of Table 2). When losing *mungady* the knowledge of "ChartColor" and "PolarChartPanel" is lost. This implies that in such a case the remaining file coverage corresponds to 50% (i.e., 2/4*100).

To identify the TF, a target threshold (e.g., 50%) that represents the critical file coverage for a project has to be defined. Then, considering the minimum[3] file coverage (i.e., the worst case, where the set of developers with the most exclusive knowledge leaves) computed for set of missing developers, it is possible to plot a curve and identify the actual TF for the project. Indeed, the TF can be deduced from this plot by finding the intersection of the selected threshold with the curve; if the obtained value is not an integer the point on

[3]Authors in [11] have also considered the best case and the average.

	TF: number of missing developers							
File (.java)	0	1 {mungady}	1 {taqua}	1 {nenry}	2 {mungady, taqua}	2 {mungady, nenry}	2 {nenry, taqua}	3 {mungady, taqua, nenry}
SWTUtils	+	+	+	+	+	-	+	-
DefaultKeyedValues	+	+	+	+	-	+	+	-
ChartColor	+	-	+	+	-	-	+	-
PolarChartPanel	+	-	+	+	-	-	+	-
File Coverage %	100	50	100	100	25	25	100	0
Min. File Coverage %	100	50			25			0

Table 2: Truck Factor example

the left must be selected. Figure 1 shows, for instance, the plot obtained for the Jfreechart example. In the example, considering a file coverage threshold of 50%, the TF is 1.

Figure 1: Truck factor chart

2.3 Thresholds

In the TF computation thresholds are fundamental. As we have seen in the approach proposed by Zazworka et al., the TF strictly depend on a threshold that represents the critical file coverage for the target project. To compute the TF such a threshold, given in percentage, has to be defined. In other words, the TF is a parametric measurement depending on a threshold X (i.e., TFX%).

Moreover, to establish whether the TF is high or low we need another fundamental threshold. Without it, we can not infer/understand whether or not the target project has low TF and thus the risk to get in trouble.

Siddharta Govindaray on his blog[4] proposed some thresholds. He claims that: "*Small teams of under 10 people usually target a TF of 4-5 for most parts of the system (that is around 40-50% of the team). Larger teams will probably target a TF of around 8 (which would probably be around 20-25% of the team). This means that should a couple of critical people go on vacation or leave the company, there are enough people in the team who can cover for them.*" Therefore, one can make the assumption that for a *small team* (i.e., ≤ 10 developers) the TF can be considered low if it is $\leq 40\%$ of the team size. Instead, for a *large team* (i.e., > 10 developers) the TF can be considered low if it is $\leq 20\%$ of the team size.

The problem with that threshold is that we can not trust it. It appeared on a non reliable source of information (a blog post) and it is not clear how it has been deduced by Govindaray (expert opinion or by measurement data from a representative set of projects?). As a consequence, we have no guarantee that that threshold can be safely used.

[4]http://siddhi.blogspot.com/2005/06/truck-factor.html

3. THEORETICAL MODEL

In this section we present a theoretical model with the goal of identifying **the maximum possible TF** (i.e., the upper limit for TF) in function of an identified set of relevant variables (e.g., Team size). In this paper, our model will be mainly used to reason about the validity of the Govindaray threshold but it is important to highlight that it is more general and useful. For example, it could be used with any proposed threshold.

For the model construction, we use the following variables:

- **T**: remaining file coverage (or residual knowledge) threshold for the TF
- **N**: number of source files
- **n**: number of developers (or Team size)
- **C**: developer's coverage, the average number of files known by one developer
- **x**: number of missing developers (i.e., developers hit with a truck)

An important factor in computing the TF is represented by the distribution of C_i, i.e., the amount of knowledge of each individual developer. The most favourable condition can be found when all developers have the same amount (i.e., same number of known files not same files!) of knowledge as we can deduce from the following *reductio ad absurdum*: let us suppose, keeping constant the average knowledge, that one developer has a larger knowledge than others; when such a developer is removed from the team, the ensuing knowledge reduction is larger than removing an average developer, therefore her contribution to achieving the TF is greater, i.e., the TF is lower.

In the following, with the goal of identifying the maximum possible TF, we assume that all developers have equal knowledge, that correspond to the average $\bar{C}$, which will be simply denoted as C. In addition we assume the knowledge of the files to be uniformly distributed among the developers.

Given the above assumptions, we can compute the probability that a given file (f) is known by a specific developer (d) as:

$$P(d, f) = \frac{C}{N}$$

since the distribution of knowledge is uniform both for developers and files, such probability is independent of both.

In order to keep the considerations more general and independent of the absolute number of files in the system, we introduce the derived measure **knowledge ratio** $KR = C/N$, being the (average) proportion of system files known by a single developer. Therefore the average number of developers that has a knowledge of a given file is:

$$K = n \cdot KR \tag{1}$$

When $K \geq 1$ (assuming a uniform distribution of knowledge across files) each file is known by at least one developer. When K is smaller than one, $K \cdot N$ is the number of files known by the developers, the proportion of files known by the development team can be obtained dividing by the total number of files (N) that gives K.

We expect in a realistic and consistent case to have a condition where each file is known to at least one developer; such condition can be expressed as:

$$KR \cdot n \geq 1 \tag{2}$$

For instance considering $KR = 10\%$ (i.e., each developer knows on average 10% of the system's files) a realistic case would require a team sizes of at least 10 developers: if each developer knows a different 10% of the system's files, then ten developers together know the whole system.

Now, let us bring into the equation the TF, which hypothesizes a certain number of developers to be removed from the project. If we remove x developers from the team then the proportion of files known to some developer becomes:

$$K_{-x} = (n - x) \cdot KR$$

When the number of developers removed from the team is equal to the TF, i.e. $x = TF$, by definition we have that $K_{-TF} \leq T$, that is:

$$(n - TF) \cdot KR \leq T$$

Since the TF must be an integer value:

$$TF = floor^{5}\left(n - \frac{T}{KR}\right) \tag{3}$$

Let us now consider the case when not all developers posses the same amount of knowledge about the system, in particular we focus on the case when one developer has a higher knowledge than the average an one as a lower knowledge, more in detail

- one developer has knowledge ratio: $KR_{+} = KR \cdot (1+\beta)$
- one developer has knowledge ratio: $KR_{-} = KR \cdot (1-\beta)$
- the remaining ones have the average knowledge KR.

Therefore we can rewrite equation 1 for the knowledge of the system with one pair of developers deviating by β from the average as:

$$\begin{aligned} K^{\beta 1} &= (n-2) \cdot KR + KR_{+} + KR_{-} \\ &= (n-2) \cdot KR + (1+\beta) \cdot KR + (1-\beta) \cdot KR \end{aligned} \tag{4}$$

[5]floor(x) is the largest integer not greater than x

Where the first term corresponds to the average developers, the second term to the above-the-average developer and the latter to the below-the-average developer.

When considering the removal of one developer from the team, the worst case (i.e., the case with the highest loss of knowledge) correspond to removing the above-the-average developer. Since when computing the TF, we always have to consider the worst case, the corresponding knowledge after removing a developer is:

$$\begin{aligned} K_{-1}^{\beta,1} &= (n-2) \cdot KR + (1-\beta) \cdot KR \\ &= (n-1) \cdot KR - \beta \cdot KR \\ &= K_{-1} - \beta \cdot KR \end{aligned} \tag{5}$$

The general formula for the knowledge level with one deviant pair of developers is thus:

$$K_{-x}^{\beta,1} = K_{-x} - \beta \cdot KR$$

By induction, we know that in the case of $d_d \geq x$ deviant developers pairs we have

$$K_{-x}^{\beta,d_d} = K_{-x} - x \cdot \beta \cdot KR$$

If we assume the case of half developers knowing $(1 - \beta)KR$ and the other half knowing $(1+\beta)KR$ of the system, the corresponding standard deviation is $\sigma = \beta \cdot KR$[6]. It is also easy to deduce how the above configuration of developers' knowledge is the most favorable case among those with the same standard deviation.

The most general formula for the upper limit of the TF given a developer knowledge ratio with mean value $\overline{KR}$ and normalized deviation σ_{KR} is:

$$TF(\overline{KR}, \sigma_{KR}) = floor\left(\frac{KR}{KR + \sigma_{KR}} \cdot \left(n - \frac{T}{KR}\right)\right) \tag{6}$$

By comparison with equation 3 we can observe that with respect to the ideal case (with null standard deviation) the real case is reduced by a factor of :

$$\frac{KR}{KR + \sigma_{KR}}$$

For instance, in a concrete case with 10 developers with an average knowledge $\overline{KR} = 10\%$ and standard deviation $\sigma_{KR} = 5\%$, assuming a residual knowledge threshold of 50%, the upper limit for TF is 3. Compared to a TF of 5 yield by equation 3 we have a reduction of 2 (closest approximation of 1/3).

4. COLLECTED DATA

In order to validate our findings we collected data from a series of 20 open-source projects published in different repositories. The features of the analyzed projects are presented in Table 3. The projects are the same used in [8] for different purposes.

The projects come from two different repositories: Sourceforge and Google Code. The smallest software (testability) counts 7 KLOC of code, while the largest (`mantisbt`) is made up of 3.3 MLOC. The number of files of the project range from 69 to 3097. Projects team sizes range from 2 to 38 developers.

[6]Therefore the normalized standard deviation is simply β

	Repository	Software	Files	LOCs	Developers	Revisions	TF60%	KR	σ_{KR}	Gov.Threshold
1	Google	EC-fantastici4	141	15460	4	203	1	0.48	0.26	**1***
2	Google	closure-compiler	536	123660	4	200	1	0.41	0.41	**1***
3	Google	v8	982	324042	32	4556	5	0.13	0.18	8
4	Google	testability	126	7282	4	151	1	0.54	0.31	**1***
5	Google	qsbmac	431	177440	2	1027	0	0.65	0.20	0
6	Google	jpcsp	672	355728	20	1504	0	0.09	0.14	5
7	Google	choscommerce	69	11016	4	224	0	0.48	0.33	1
8	Google	keycar	669	83936	4	465	0	0.29	0.33	1
9	Sourceforge	cppunit	378	22646	6	582	0	0.23	0.33	2
10	Sourceforge	gtk-gnutella	723	248697	15	17424	1	0.17	0.29	4
11	Sourceforge	httpunit	354	39232	3	1062	1	0.61	0.43	**1***
12	Sourceforge	jfreechart	1063	149727	3	2272	0	0.34	0.47	1
13	Sourceforge	mantisbt	641	3399759	38	5752	7	0.18	0.26	10
14	Sourceforge	openvrml	476	74004	2	4141	0	0.56	0.44	0
15	Sourceforge	pdfcreator	193	49834	2	676	0	0.55	0.44	0
16	Sourceforge	phpwiki	537	118815	15	7370	1	0.17	0.29	4
17	Sourceforge	tora	703	198512	17	3523	2	0.20	0.26	4
18	Sourceforge	winmerge	862	172366	16	7149	1	0.15	0.18	4
19	Sourceforge	wvware	93	32258	2	45	0	0.55	0.44	0
20	Sourceforge	kompozer	3097	3287803	2	225	0	0.50	0.50	0

Table 3: Sampled open source projects

The TF for each project has been computed using an implementation of the algorithm presented in Section 2.2. We have adopted $T = 60\%$ (a reasonable value). The last column of Table 3 reports the threshold computed according to Govindaray: a "*" indicates projects where the computed TF is greater or equal to the Govindaray threshold. It happens in only four cases out of 20 (i.e., 20%). More precisely, in four cases the TF is equal to the threshold and in none the TF is greater to the threshold. That means that all the considered projects have low TF.

Moreover, for each project, we analyzed the number of files known to each individual developer and computed the average knowledge ratio (KR) and the corresponding dispersion in terms of standard deviation (σ_{KR}). The relationship between average knowledge ratio and standard deviation is presented in Figure 2. Projects are represented by bubbles whose size is (log-)proportional to the relative team sizes.

We can observe that the two "clouds" of points seem to be separated. In particular, large projects exhibit smaller average developers knowledge while small projects typically show larger knowledge ratios. We actually expected this kind of distribution, whose cause can be explained easily: in large teams the knowledge of the system can be divided among a larger number of developers, thus each developer knows a smaller proportion of files. The opposite happens for small teams: the few developers must know a larger share of the system.

As far as the dispersion of knowledge is concerned, we observe smaller standard deviation values for large projects and larger dispersion for small projects. Also this experimental result is reasonable because in small projects Heroes are more common in percentage [9].

5. ANALYSIS

In this section we compare the theoretical maximum TF with the Govindaray threshold. We conduct the comparison in two different ways:

- first considering a very favourable condition, i.e. when all developers know almost the same proportion of the system files ($\sigma_{KR} = 0.1$)
- second analyzing how the maximum threshold is influenced by varying dispersion of developers' knowledge

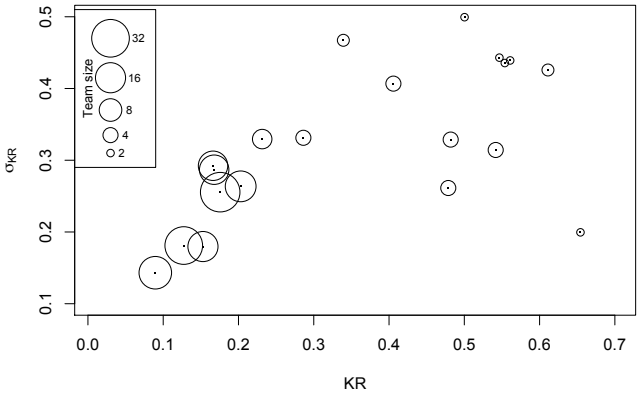

Figure 2: Developer average knowledge and dispersion

All the values of TF are computed with reference to a minimum system knowledge threshold $T = 60\%$.

5.1 Maximum Threshold

We can observe in Figure 3 how the maximum TF varies as a function of team size according to equation 6. We assumed a small dispersion of knowledge, in particular $\sigma_{KR} = 0.1$, which is smaller than any value we found in real software projects, as can be observed in Figure 2.

Each oblique dashed line represents a different average knowledge ratio (KR) of developers, where the actual values are represented by the different markers. While lines cross the whole diagram, points are reported only for consistently realistic values, i.e. when the average knowledge allows the whole system to be known, according to the condition expressed in equation 2. For instance considering $KR = 10\%$ (the "×" symbols) we have a consistent case for team sizes of 10 or more developers.

The Govindaray threshold is represented by the continuous line; since the threshold must be compared to an integer value, its value has been rounded to the closest integer, as a consequence the line appears stepped.

We can observe how, in many cases, the threshold is very close to, if not even above, the maximum achievable TF. For instance, considering the case of $KR = 5\%$ (the "△" symbol) we can notice that for teams of 20 developers, the maximum TF is 2, which is considerably smaller than the Govindaray threshold (i.e., 5). Even for large values the maximum TF remains below the threshold. To reach the TF threshold value, each of the 20 developers should known 7.5% of the system on average (the "+" symbols). To understand the practical implication of this, we could consider a system with 400 K files (similar to `jpcsp` in Table 3), $KR = 5\%$ means that each of the 20 developers knows on average 20 K files, while $KR = 7.5\%$ corresponds to 30 K files. In practice to achieve a sufficient TF each developers would need to get acquainted with additional 10 K files.

We observe, on the other side, that for large team sizes and greater knowledge ratios the threshold is lower than the maximum possible value. For instance, considering a KR=20% and $\sigma_{KR} = 0.1$, for a team size of $n = 16$ the maximum TF is 8 and the threshold is 4. Such a positive case implies that each file is know, on average, by 3.2 ($= KR \cdot n$) developers. Unfortunately, at least in our sample, this is an uncommon condition.

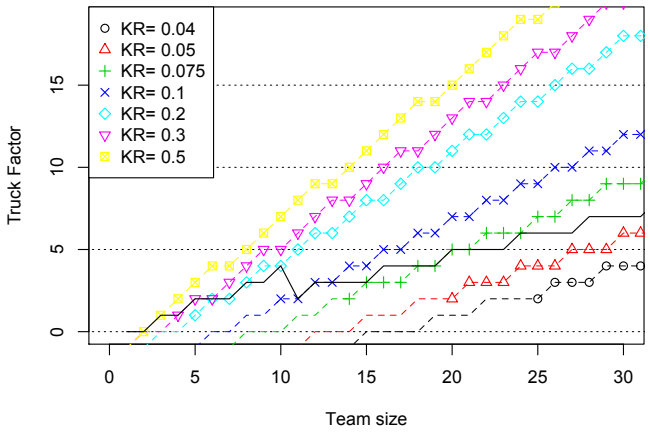

Figure 3: Variation of TF as function of team size for different KR (@ $\sigma_{KR} = 0.1$).

5.2 Knowledge Dispersion

The previous comparison considered only a fixed and relatively small dispersion of the developers knowledge distribution ($\sigma_{KR} = 0.1$). In practical cases we face teams where a wide range of knowledge is present as can be observed in Figure 2.

Figure 4 presents the maximum TF limits for different team sizes considering different dispersion levels of the developer knowledge (σ_{KR}). We consider in this example a fixed average knowledge ratio $KR = 20\%$. As in the previous section's diagram only the points corresponding to consistent cases (see equation 2) are reported.

The topmost line represents the case of $\sigma_{KR} = 0$, which correspond the ideal and most favorable case when all developers know exactly the same amount of files. The bottommost line corresponds to a case with $\sigma_{KR} = 0.5$; in practice this latter case could happen in a ten developers team when we have: two developers knowing just 1% of the system, five 2%, one 5%, one 35%, and one 60%. Such significant variability although striking is quite common in actual projects (see Table 3). The Govindaray threshold is represented by the continuous line.

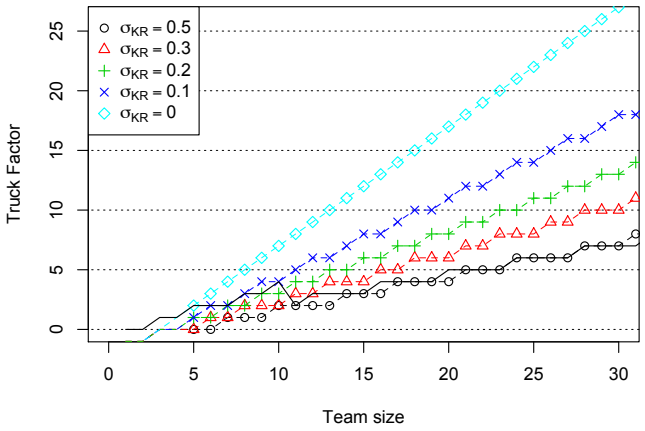

Figure 4: Variation of TF as function of team size for different KR dispersion (@ $KR = 20\%$).

We can observe a condition close to that of an actual project (see e.g., project `gtk-knutella` in Table 3 having $KR = 0.17$, $\sigma_{KR} = 0.29$, and 15 developers): the "△" symbol for team size 15 represents the maximum possible TF (i.e., 4) and is just one unit greater than the Govindaray threshold (i.e., 3). To be in a safe zone, where the maximum TF is twice the threshold, the team ought to reduce the dispersion by $^1/_3$, thus achieving a $\sigma_{KR} = 0.2$ (the "+" symbol) that would grants a maximum TF of 6.

6. DISCUSSION

From our analysis we can summarize two main findings:

- comparing the Govindaray threshold to the theoretical maximum TF at different level of system knowledge (Figure 3) we observe that in several cases – very close to real projects conditions – the threshold appears either above the maximum or just barely below

- the same conclusions can be drawn comparing the threshold with the maximum TF at varying knowledge dispersion values (Figure 4)

The two findings consistently indicate that most real projects have too low a TF, which means they are exposed to a high risk of failure in case some developers abandon the team. The threshold appears to be applicable consistently only for projects where the typical developer knows at least half of the system. This appears to be close to having all developers in the project being Heroes.

In addition it appears that the effort required to achieve a sufficient threshold, by distributing the knowledge among the developers, is very high (e.g. each developer acquiring some kind of knowledge of 10.000 new files!).

From the above premises we draw the conclusion that the TF threshold proposed by Govindaray (the only one formulated with sufficient precision) is too high both compared to real projects and to theoretical thresholds.

An alternative interpretation is that, assuming the threshold is substantially correct, most open source project are

very fragile and particularly sensitive to developers abandoning the team. Although, we personally believe less this later explanation.

6.1 Threats to validity

We know that the reliability version control systems' logs can be varying, e.g. the recall of traceability links can be as low as 40% [2]. Therefore it is possible that the file knowledge measures we collected from the analyzed projects are underestimated. Although this might compromise our observation concerning the actual data, all our considerations about the theoretical model remain valid.

We assumed the knowledge relationship between a developer and a file both assumes a binary value and remains constant over time. There have been proposed more sophisticate model taking into consideration interactions in time to compute a degree of knowledge [5]. We believe the discussion presented in this paper may be considered valid as a snapshot of the project in a specific instant in time.

It is possible that the idea of TF is valid only for projects adopting an Agile approach, or more specifically projects enacting the XP practice of collective code ownership. Since no detailed information is available about the development process adopted in the analyzed open source projects, it is very likely none of them strictly adopted Agile principles, and therefore our findings lack construct validity. Although the idea of TF originated in the Agile communities, no one ever explicitly limited its applicability to projects adopting collective code ownership.

7. CONCLUSION

In this preliminary work, we presented the first theoretical model able to infer the maximum possible Truck Factor of a project (i.e., the upper limit for the Truck Factor). We validated it against some data collected from a set of open source projects.

In this paper, the proposed model has been mainly used to reason about the validity and applicability of the Govindaray threshold (in the best of our knowledge, the only one formulated with an adequate precision) but we have to point out that it is more general and has a wider use.

The main outcome of this work is that the Govindaray threshold appears to be practically inconsistent and inapplicable in several cases: it is too high both compared to real projects and to theoretical thresholds. We believe that this preliminary outcome could represent a new starting point for stimulating researchers and developers to work on the Truck Factor and on a new and more reliable threshold.

For the future, we plan to improve the theoretical model making it stronger and precise.

Furthermore, we also plan to perform a larger and systematic experiment for validating our model.

8. REFERENCES

[1] T. Alves, C. Ypma, and J. Visser. Deriving metric thresholds from benchmark data. In *26th IEEE International Conference on Software Maintenance*. IEEE, 2010.

[2] K. Ayari, P. Meshkinfam, G. Antoniol, and M. Di Penta. Threats on building models from cvs and bugzilla repositories: the mozilla case study. In *Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*, CASCON '07, pages 215–228, New York, NY, USA, 2007. ACM.

[3] K. Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 1999.

[4] T. Fritz, G. Murphy, and E.Hill. Does a programmer's activity indicate knowledge of code? In *Joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering (ESEC/FSE)*, pages 341–350, 2007.

[5] T. Fritz, J. Ou, G. C. Murphy, and E. Murphy-Hill. A degree-of-knowledge model to capture source code familiarity. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 385–394, New York, NY, USA, 2010. ACM.

[6] T. Girba, A. Kuhn, M. Seeberger, and S. Ducasse. How developers drive software evolution. In *International Workshop on Principles of Software Evolution (IWPSE)*, pages 113–122, 2005.

[7] M. Nordberg III. Managing code ownership. *IEEE Software*, pages 26–33, 2003.

[8] F. Ricca and A. Marchetto. Are heroes common in floss projects? In *International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2010.

[9] F. Ricca and A. Marchetto. Heroes in FLOSS projects: an explorative study. In *International Working Conference on Reverse Engineering (WCRE)*, pages 155–159. IEEE, 2010.

[10] E. J. Weyuker, T. J. Ostrand, and R. M. Bell. Do too many cooks spoil the broth? using the number of developers to enhance defect prediction models. *Empirical Software Engineering*, 13(5):539–559, 2008.

[11] N. Zazworka, K. Stapel, F. Shull, V. Basili, and K. Schneider. Are developers complying with the process: an xp study. In *Symposium on Empirical Software Engineering and Measurement*. IEEE, 2010.