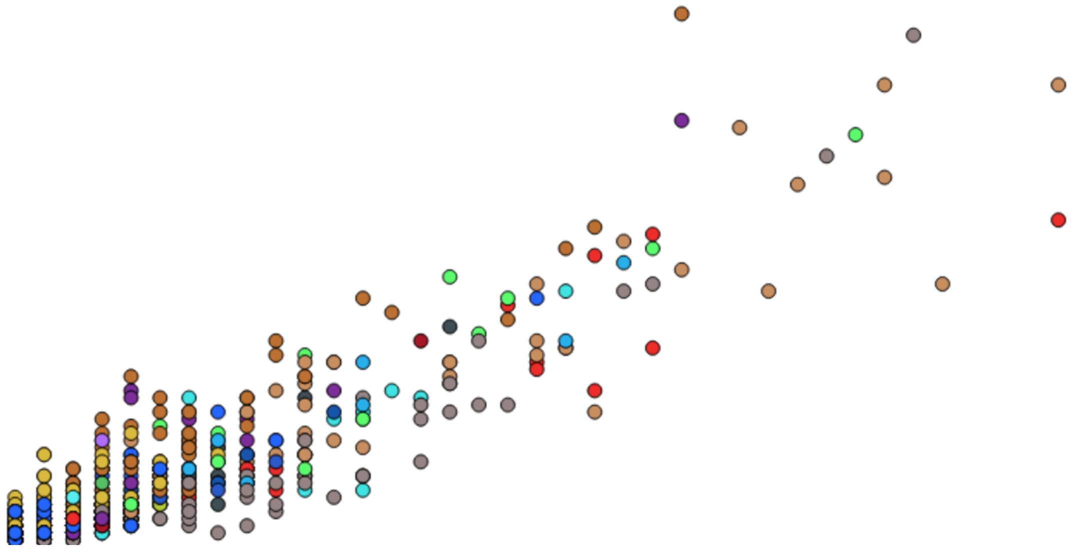
UNIVERSITY OF GOTHENBURG



How Multiple Contributors Reduce Software Quality

A Quantitative Analysis on a Large Telecommunications Company

Bachelor of Science Thesis in Software Engineering and Management

ALE LOTSTRÖM
DANI HODOVIC

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden, June 2015

How Multiple Contributors Reduce Software Quality
A Quantitative Analysis on a Large Telecommunications Company

ALE LOTSTRÖM
DANI HODOVIC

Examiner: MORGAN ERICSSON

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover:
The linear relationship between the number of contributors and the number of defects in a file (page 6).

Department of Computer Science and Engineering
Gothenburg, Sweden, June 2015

Abstract

Since the origin of agile methods and open source development, code ownership tends to be more widely distributed over multiple contributors than before. The question is to what extent a component is affected when several developers contribute to it. Does several contributors provide better solutions than a sole developer or does multiple contributors generate additional defects? Although some previous research results point to the fact that more contributors do expose a project to higher risk, surprisingly little has been done to validate this hypothesis. An answer to this question is highly relevant since it provides organizations with the option to adjust development teams and contribution levels accordingly, in order to assure software quality. By empirically studying a large data set from a proprietary telecommunications company, we examine the relationship between the number of contributors and the number of defects in closed source industrial projects. In addition, we are the first to investigate the effect that multiple contributors have on defect density and defect severity. We find the correlation between contributors and defect density to be statistically significant and we find that the number of contributors and the number of defects has a near perfect positive relationship.

1. Introduction

Defects cost the software industry substantial amounts of money each year. In the US alone, the cost is 59.5 billion USD annually [1]. There is no definitive factor that causes defects, but plenty of previous research indicated that human factors such as developer expertise play a big role [2, 3, 4, 5, 6]. Some researchers have specifically investigated how different compositions of developers and organizational structures can affect the quality of software [7, 8]. As organizations scale and software development teams grow, one can assume that the human factor becomes even more evident.

Fred Brooks [9] famously stated that "adding manpower to a late software project makes it later" after unsuccessfully increasing the number of developers in hopes of finishing a project on time. Other research states that when many developers collaborate on a component, there is a possibility that it becomes victim of unfocused contributions [10, 11]. For instance, a lack of communication between contributors could likely lead to confusion of who is responsible for what, causing the interaction of commits to be less optimal [10]. There is also a chance that a contributor changes something without proper feedback from other developers, which is especially common in Open Source Software (OSS) [11]. Yet, OSS is known to be secure because of the fact that many developers interact with the components, finding and fixing faults that might not have been found by a sole developer [12, 13, 14]. So far, not many studies have examined the effect the amount of contributors has on proprietary software quality, which is the main purpose of this paper. In order to do this, we analyze how the number of contributors affect the number of defects, defect density and defect severity on a file level. The research is conducted within a division at Ericsson, a large telecommunications company that produces distributed, fault-tolerant, soft-real-time, non-stop systems. We conduct a statistical analysis on a large data set ranging over 10 years, containing thousands of source files produced by hundreds of developers.

By thoroughly analyzing data scraped from version control systems, we aim to answer the following questions:

RQ1: *How does the amount of contributors affect the number of defects in software development?*

RQ2: *What is the correlation between defect density and the number of contributors?*

RQ3: *What is the correlation between defect severity and the number of contributors?*

Using Pearson's R, we find a very strong correlation (0.92) between the number of contributors and the number of defects. We believe the strength of the correlation may be affected by other confounding factors, but that the number of contributors is most likely the primary reason. In comparison, we find that the correlation between the number of contributors and defect density is only 0.29, but increases as files grow in size. Furthermore, we find that the ratio of defect severity remains roughly the same when there is an increase in contributors.

Our research contributes to the software industry by providing insight in how the number of contributors and defects correlate. If organizations are aware of how additional developers affect product quality, they may reconsider their development structure and can account for potential threats in their decision making processes.

This paper is structured as follows: Section II addresses previous research related to our subject. Section III contains definitions of our metrics and common terminology used throughout the paper. Section IV introduces our posed hypothesis while Section V explains our data collection and analysis methods. Sections VI and VII cover our results and discussion. Section IX addresses potential threats to validity and Section X acts as a summary and conclusion.

2. Related Work

In software engineering research, data mining and tool-driven approaches are common when analyzing source code and there has been several studies addressing how the quality of a component is affected by the developers contributing to it. Most research that has been done was primarily conducted on OSS projects, often focusing on developer expertise, rather than the number of contributors. Furthermore, many of the prior studies are mostly drawn towards improving defect prediction models and not many have tried to empirically quantify the actual effect that the number of contributors has on code quality.

Meneely and Williams [11] studied the relationship between security vulnerabilities and the number of developers working on the Linux kernel. They found the likeliness of a security vulnerability to increase sixteenfold if more than nine developers contributed to a source file. Their research methodology and used metrics are very similar to ours with the difference that we use defects in order to estimate the amount of defects in a file while they study whole source code files labeled as either "vulnerable" or "neutral", depending on if the file requires patching. We investigate if similar results can be found in proprietary software.

Nagappan et al. [15] provides some results on proprietary project by extracting data from Windows 7 and Windows Vista. They noticed that binaries with less minor contributors and more major contributors contained fewer defects. The way they define code ownership is by looking at the number of commits a single developer made in relation to the total number of commits. Even though contributions often vary in size, it is still a reliable metric. Overall, our research approach is similar but we differ in the sense that we study contributors and defects on a file level. They were unable to trace defects back to particular files and count defects by pre-release and post-release failures of entire products. There are several other studies [7, 16, 17], also from Microsoft, that all examine concepts affecting quality such as organizational structures and distributed development. All the metrics are somewhat related to our research but again, none of these specifically study contributors and defects at a file level. Also, none of the studies from Microsoft examine defect density as we do. In addition to Windows Vista, Bird et al. [18] also investigated the effects of ownership in Eclipse and Firefox. They found that high proportions of ownership and low amounts of minor contributors generated less defects across all three projects. Interestingly, their results are similar among each other even though the three projects are developed using different organizational models. Shin et al. [19] also studied multiple releases of Firefox along with the RHEL4 kernel. Specifically, they examined the relationship of developer activity, code churn and complexity with known security vulnerabilities within those systems. They found statistically significant correlations between the security vulnerabilities and all the three types of metrics. This study uses a different set of metrics than ours and like most other related research, the results are derived purely using from data from OSS projects. For all OSS studies, results are not necessarily applicable to closed source projects because of assumed differences in team structures and communication efforts.

A number of studies [2, 3, 4, 5, 6] examine the impact developer experience and expertise has on software quality. They all confirm that developers with an in depth understanding of the domain, system applications and components are less likely to induce additional faults. Izquierdo-Cortazar et al. [20] also studied developer experience and its relation to the ratio of bug inducing. In contrast to the other studies, they found no statistical significance stating that inexperienced developers are more likely to introduce bugs. This study was performed at Mozilla, using mainly bug fixing commits and bug seeding commits as metrics. We are not examining data on developer expertise in this paper but we do address it as part of the discussion as a confounding factor.

Lastly, Pinzger et al. [10] uses contribution network models to predict future failures of systems. They found that central components sharing many contributors are far more error prone than components with less contributors. Similarly to defect prediction, our research aims to help organizations understand what could cause additional defects.

Judging by the previous studies, we definitely see some solid research efforts with interesting results on several fronts. However, there is an obvious lack of quantitative studies regarding defects and contributors in large industrial project. Evidently, most prior research are done on OSS projects and a large focus is on developer expertise, prediction models, organizational processes and source code metrics etc. In addition, the research that do exist does not address either defect density nor defect severity. Furthermore, we are the first to empirically quantify the effect the number of contributors has on both defect density and defect severity, enabling us to fill a gap in the research community.

3. Metrics & Terminology

Several metrics are used to address our research questions and they act as the main points of interest in our data collection. Because such a large number of observable characteristics exists in software development projects, we use a focused top-down approach when conducting our measurements. Our metrics have been chosen in accordance with industry standards and the Goal/Question/Metric Paradigm [21, 22]. Description of common terminology, used metrics and the reasoning behind them is presented on the following page (Table 1).

4. Hypotheses

With regards to our main research question (RQ1), we investigate if the number of contributors affects the number of defects per file. Specifically, we examine if having more than one contributor to a file results in an increase or decrease in faults. There are indeed arguments for both tendencies. On the one hand, one can assume that an increase in contributors makes it harder to coordinate and organize committed code, which could have a negative impact on quality. Additionally, an increase in contributors might also complicate communication efforts, potentially increasing the amount of defects. Furthermore, there is also a possibility that the level of expertise of an added contributor has an impact on the file. As found in other studies [2, 3, 4, 5, 6], it is likely that a generalist will induce additional faulty code. On the other hand, it is also a possibility that an additional expert could potentially lower the amount of faults in a file. Thus, we investigate whether the effects balance each other out or not. Furthermore, there is a chance that an increase in contributors simply raises the amount of found bugs and does not increase the actual amount, meaning that a sole developer is less likely to discover many defects.

Null Hypothesis 1 *There is no difference in the number of defects based on the number of contributors.*

Alternative Hypothesis 1 *There is a difference in the number of defects based on the number of contributors.*

Secondly, we examine the effect that the number of contributors have on defect density (RQ2). Despite the fact that previous research [10, 11, 15] emphasizes that multiple contributors induce more defects, they provide no answer as to whether or not defect density is affected. As such, more contributors could equal more defects, but also larger files, which would not result in lower product quality when size is taken into consideration. Therefore, defect density is a far more reliable measure of product quality than purely looking at the number of defects as it accounts for the size factor of files [27, 28]. If there is no correlation between contributors and defect density it would highly question previous research results.

Null Hypothesis 2 *There is no difference in defect density based on the number of contributors.*

Alternative Hypothesis 2 *There is a difference in defect density based on the number of contributors.*

Finally, we examine if there is a change in the ratio of defect severity when the number of contributors to a file increases (RQ3). Specifically, we investigate how the number of critical faults (Class A) increases relative to Class B and Class C faults. If there is a larger ratio of Class A faults, added contributors could cause significantly reduced quality. Testing this is relevant as one can argue that a ratio increase of Class C faults is more forgiving and does not affect quality as much. However, we deem it most likely that the distribution of defect severity remain roughly the same, and even if the ratio changes, we would likely see an increase or decrease of all severity types.

Null Hypothesis 3 *There is no difference in defect severity ratio based on the number of contributors.*

Alternative Hypothesis 3 *There is a difference in defect severity ratio based on the number of contributors.*

5. Data Collection & Analysis

The data set we use in the study originates from data mining tools built internally at Ericsson. The tools parse revision control history, usually from the master branch, and store the results in a SQL database. The database contains data commit entries from "normal" feature development and commits that are intended to fix defects.

Because we look at defect density as part of our paper, we have decided to exclude any other file type than C from our analysis. The reason for this is that there is no definitive way of measuring size in different programming languages. If we were to include multiple file types in our study it would introduce risks of ambiguity in the data set as size measurements across programming languages is out of this scope for this study. Since

Table 1: Metrics & Terminology

Keyword	**Definition**	**Usage**
Contributor(s)	A developer that contributes to a file is labeled a contributor. We calculate the number of developers per file based on distinct entries from feature development data and bug fixing data.	Used when estimating the number of developers per file.
Defects	Entries in the version control system tracked as bug fixes. Also known as Trouble Reports (TRs). Gives an estimate count on defects per file when aggregated.	Used when estimating the number of defects per file.
Lines of Code (LOC)	Number of lines in a file. For source code this represents the code including comments. For arbitrary files this represents all text.	Used a size measure when calculating the defect density for a file.
Defect Severity	There are four different severity tags attached to most of the defects: Improvements, Class A, Class B and Class C faults. Improvements are typically changes that are not fixing bugs and therefore we exclude that tag when testing severity. Class A (e.g causing service unavailability or process restarts) is the most critical type of fault while Class B (e.g simple failures or disturbances) is of medium severity. Class C (e.g spelling faults and incorrect printouts) is the least critical type of fault.	Used when filtering out data and retrieving valid defects.
Defect Density	The ratio of defects to size. Defects are generally counted through the number of TRs [23]. Size is commonly expressed in Functional Points, cyclomatic complexity or LOC, and it often serves as a measurement of product quality [24, 23, 25, 22]. In terms of defects we only track the ones that indicate an actual failure occurred, this means that we only include defects of severity A, B, C.	Used for estimating if an increase in contributors causes a higher defect density.
Cyclomatic Complexity	The number of conditional statements in a file. This can only be measured in source code files.	Used as an alternative measure for size of files. We divide the number of defects with the number of conditional statements for each file.
Effective Complexity	Effective complexity is another measurement that gives a relative complexity of a file based on the complexity of the functions[26]. This is a better complexity measure of a file than cyclomatic complexity as it cannot be affected by file size as much as cyclomatic complexity is.	Indicates how complex a file is. Used as a supplementary metric when looking at correlation matrices.

the majority of the files at the target company are written in C, with a smaller portion being written in C++, we chose C files as the source of analysis.

Our research primarily revolves around the correlation between contributors and defects per file. We have therefore spent a substantial amount of time aggregating the data set for each file. Table 2 contains data on the files studied.

We gather defect data based on version control commits that are tagged as defect fixes. Not all of the commits are proven to belong to a confirmed defect and some of these commits lack proper defect tags. We have chosen to exclude all of the commits that did not tag a specific defect severity from the analysis. It is important to note that the majority of the defects we found were not defects reported by customers, but defects that were found internally during development and testing phases. Table 3 provides an overview of the defect data.

Table 2: Files.

Property	**Amount**
Total number of files	17130
.c, .cc files	2977 (16%)
.cpp files	572 (3%)
Other (eg. .h, .hpp, .xml, .txt)	13581 (80%)
Files containing known defects	5476 (30%)
Source files containing known defects	1484 (8%)
Files with A, B or C severity tags	4636 (25%)
Files containing improvement tags	3018 (16%)

We collect contributor data from two version control

Table 3: Defects.

Property	Amount
Total number of known defects	10529
Class A defects	1926 (18%)
Class B defects	5357 (51%)
Class C defects	1679 (16%)
Improvements	557 (5%)
Untagged	1010 (10%)

tags, one for normal feature development and one for defect fixes. The two sources contain roughly the same number of contributors (See Table 4), but data from the defect fixing branch is far more reliable. The reason behind this is that the majority of the feature development is done in alternate branches, squashed into one commit and then delivered to the master branch. This means that a lot of commits and their subsequent data is lost, e.g a feature which was developed by 10 developers is delivered as a commit of one developer. Defect fixing data on the other hand is usually fixed directly on the master branch, or developed in very few commits before integrated into the master branch. This results in a higher total number of contributors found fixing defects than what we found in feature development. We have chosen data from the defect fixing tags as the main source for contributor data, but we provide an analysis and results for both data sets.

Table 4: Contributors.

Type	Amount
Contributors (feature development)	461
Contributors (defect fixes)	491

In order estimate defect density for each file we need to identify a size measurement and in this case we have chosen LOC. LOC may not be a valid measurement across languages as they vary in verbosity and abstraction, but in the case of comparison of files within one language it seems perfectly valid. We divide the number of defects a file has with the LOC to get the defect density. As a supplementary metric, we also use cyclomatic complexity as a second size metric to see if the results are aligned with the LOC metric.

Our main method for proving correlation is by using simple linear regression tests. We use the Pearson Product-moment Correlation Coefficient to determine the linear correlation between the independent variable (contributors) with the dependent variable (defects). The Pearson test produces a correlation coefficient which ranges from -1 to 1, where -1 indicates perfect negative correlation and 1 expresses perfect positive correlation while a correlation coefficient of 0 or values close to 0 implies a weak correlation [29]. The Pearson test is not distributionally robust and can be influenced by outliers [30]. We graphically visualize the data using scatter plots in order to manually identify the extensiveness of potential outliers.

We mainly look at only C files data from 2013-2015 because the defect tagging accuracy rose above 50%, implying that the tagging practises improved significantly in the company and consequently providing us with a purer data set. A brief overview of the defect tagging accuracy is displayed in Figure 1, showing a reach of 50% in 2013. Even though we exclude certain data from our analysis, our sample sizes are still large, ensuring quality results.

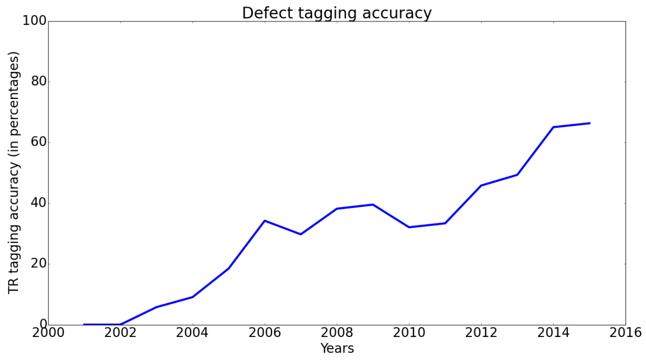


Figure 1: *Defect Tagging Accuracy*

The data mining tools are primarily written in python. In order to easily query the API they provide, we chose to conduct this study using tools from the Python ecosystem. Notably we used R, Numpy and Scipy to calculate the correlation matrices. The charts in this paper were plotted using matplotlib and d3.js.

6. Results

From our statistical tests, we created a matrix (Table 5) containing the correlation coefficients of all our metrics combined. The matrix displays data from the last two years (2013-2015). However, when analyzing all data ranging back from the earliest entry (2001) , we do not experience any dramatic changes in values. From here on, we will refer to contributors from the feature development as FD and contributors from defect fixes as DF.

How does the amount of contributors affect the number of defects in software development?

Evidently, we find a strong relationship between the

Table 5: *Pearson Correlation Matrix. (C files 2013-2015)*

-	Contr. FD	Contr. DF	Defects	Defects A	Defects B	Defects C	LOC	Complexity	Def.density(LOC)	Def.density(CX)
Contr. FD	1	0.87	0.77	0.66	0.73	0.68	0.48	0.52	0.25	0.14
Contr. DF	0.87	1	0.92	0.79	0.88	0.79	0.49	0.56	0.29	0.14
Defects	0.77	0.92	1	0.85	0.97	0.84	0.47	0.54	0.30	0.15
Defects A	0.66	0.79	0.85	1	0.74	0.60	0.38	0.45	0.28	0.12
Defects B	0.73	0.88	0.97	0.74	1	0.76	0.44	0.52	0.27	0.13
Defects C	0.68	0.79	0.84	0.60	0.76	1	0.46	0.49	0.27	0.15
LOC	0.48	0.49	0.47	0.38	0.44	0.46	1	0.82	0	0.04
Complexity	0.52	0.56	0.54	0.45	0.52	0.49	0.82	1	0.04	-0.03
Density (LOC)	0.25	0.29	0.30	0.28	0.27	0.27	0	0.04	1	0.29
Density (CX)	0.14	0.14	0.15	0.12	0.13	0.15	0.04	-0.03	0.29	1

number of contributors and the number of defects in a file. Looking at contributors from DF (See figure 2), we note an almost perfect positive relationship between the variables (0.92) and for contributors in FD (Figure 3) a correlation of 0.77. For both metrics we reject our first null hypothesis. Both measurements are statistically significant at a confidence interval of 99%, with p-values smaller than 1.0e-300. The fact that the FD metric is less strong is most likely the result of improper defect tagging culture as discussed earlier. When examining all files versus only C files, the difference in correlation is insignificant. Furthermore, by analyzing this data over time, there is no major difference other then that the median values for both defects and contributors are slightly smaller when only examining data from the last two years. The linear relationship is not notably different and the correlation between defects and contributors persists. Because of the large number of files studied, it is hard to get a good understanding of the full distribution of the files in the graph. Therefore, it is important to note that the majority of the files are worked on by a single developer and contains only one defect. There is also relatively few instances of files where there is only one contributor and more than one defect.

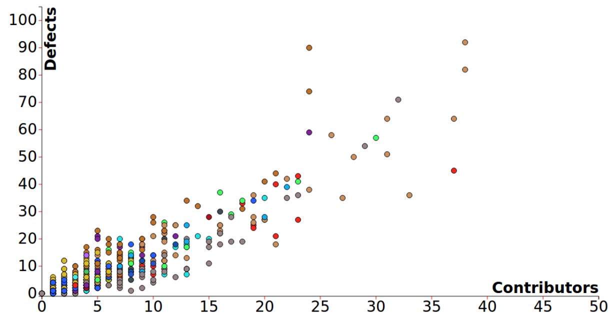


Figure 2: *Contributors (DF) and Defects for C files between 2013-2015*

What is the correlation between defect density and the number of contributors?

Looking at defect density we find that the Pearson value is 0.29 while the scatter plot shows heavy presence of files with few contributors and varying defect density values as seen in Figure 4. The result is statistically significant and we able to reject the null hypothesis at a confidence interval of 99% with a p-value of 4.2e-60. Analyzing the data further, we find that the median LOC value for all C files is 245 while the mean is 585. This indicates that there are many files with a very low LOC count. Surprisingly files with a larger LOC than 10 carry the coefficient 0.37 and files larger than 100 LOC push that value to 0.46. We therefore suspect that these small files significantly decrease the linear correlation. When looking only at files above the median, the correlation increases to 0.57.

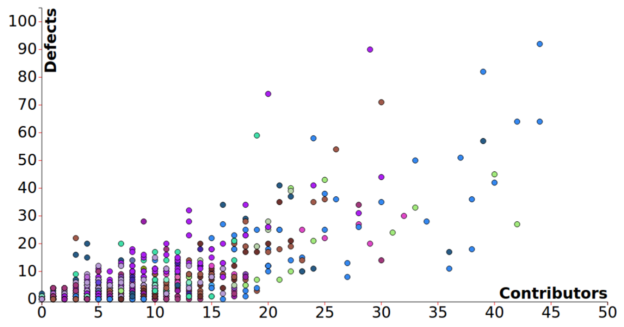


Figure 3: *Contributors (FD) and Defects for C files between 2013-2015*

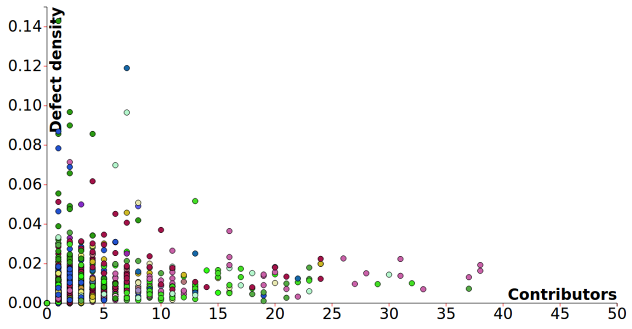


Figure 4: *Contributors (DF) and Defects Density (LOC) for C files*

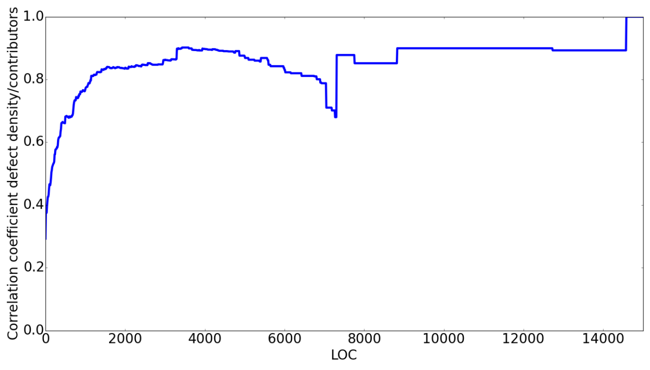


Figure 5: *LOC and the correlation coefficient for Defect Density/Contributors. The steep increase from the coefficient value of 0.28 to 0.40 happens around 10 LOC and is not visible in the plot*

In Figure 5, we demonstrate the relationship between the defect density correlation coefficient and how it increases when files get larger. The correlation coefficient (Y) is derived from files larger than the LOC value at X, e.g the correlation coefficient for X = 500 was calculated for all files with more than 500 LOC. Naturally, filtering the data like this increases the p-value as the sample size becomes smaller (only 31 files larger than 5000 LOC), but when looking only at files with as high as over 8000 LOC, the p-value is only 0.037 and the results still statistically significant. Regardless, the main purpose with this is to present how defect density values are skewed if file sizes are small, explaining the shape of Figure 4. As expected, the correlation coefficient increases when examining larger files. With Pearson's R ranging from 0.74 for files larger than 750 LOC to 0.82 for files larger than 1500 LOC.

What is the correlation between defect severity and the number of contributors?

When calculating Pearson's R for the DF metric with proper severity tags and those without, the resulting coefficients are 0.84 for class C faults, 0.88 for class B faults and 0.85 for class A faults. As shown in Figure 6, the different classes of defect severity all increase in a similar fashion when there is an increase in contributor amount. Therefore, we fail to reject the null hypothesis and state that there is no difference in defect severity ratio based on the number of contributors. Since 51% of the known defects are of Class B, it is logical that multiple contributors add mostly Class B faults compared to Class A and Class C, thus keeping the ratio of overall severity relatively unchanged.

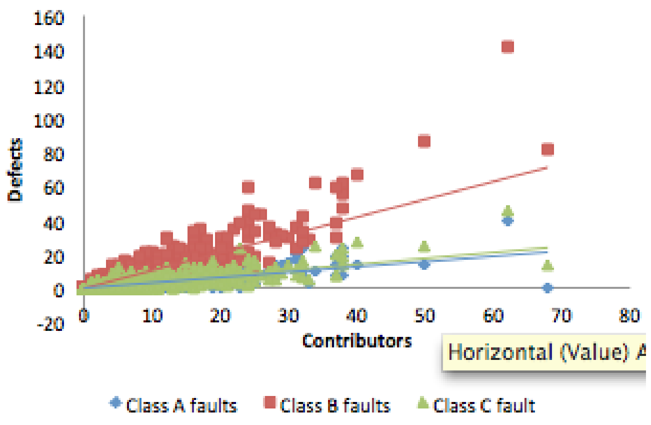


Figure 6: *Contributors (DF) and Defect Severity*

7. Discussion

Due to the statistical significance of our results, we are able to state that an increase in contributors most certainly does increase the number of defects. However, more contributors does not seem to increase defect density as much, mostly because many files have small sizes, showing high defect density for as little as one defect. The difference between the correlation coefficients of contributors/defects and contributors/defect density indicates that there are confounding factors affecting at least one of the correlations. Looking at figure 5, we do indeed see that the correlation between contributors and defect density increases as file sizes grow bigger.

In context, we can go back to Brook's law stating that "Adding manpower to a late software project makes it later" and relate our results to what he found. Bearing in mind that there is no guarantee that the number of defects directly causes a later release of a product, it can still effectively lower the quality of it. One can ask what the "perfect" contributor amount would be in order to assure as few defects as possible, and judging by our results, that number seems to be one. Does this mean that collaboration efforts in software development should be avoided? Much more extensive research is needed to address such a conclusion. However, it is interesting that when we look at files with two contributors, there is already a notable increase in defects.

We can also compare our results to previous research attempts [15, 10, 11] that has indicated that an increase contributors causes a decrease in software quality. Our results clearly supports their findings but on a much finer granularity than binaries or large components as we evidently prove the same hypotheses on a file level. As mentioned, Meneely & Williams [11] found that the likeliness of a security vulnerability increases sixteen fold if more than nine developers contributed to a source file. Notably, we do not see such a significant increase related to a particular contributor amount as we find the increase to be relatively linear. It is also important to note that our use of size metric to estimate defect density could increase the reliability of results compared to the other studies.

Furthermore, when comparing proprietary software and OSS, one can speculate in that communication between developers has potential to be better in proprietary software, as development is often less geographically distributed. Logically, this should mean that proprietary software benefits from an increase in contributor amount more than OSS does. However, judging by both our research as well as previous studies [15, 10, 11], it is hard to argue for any benefits coming from an addition of contributors at all. Interestingly enough, it seems to be even more disadvantageous in proprietary software, based on the lack of research on the subject compared to research on OSS, stating that the contributor amount adds security [31, 13, 14].

One big question mark that does arise from our results is whether or not an increase in contributors increases only the number of discovered faults or if the total number of faults actually increases. Because of a large number of contributors, an increase in found faults is known to be a common factor in OSS. For instance, Eric Raymond [12] stated that "given enough eyeballs, all bugs are shallow" when studying the Linux kernel. So does this mean that with more contributors, more defects are just found? It is likely to be true to some extent and perhaps more so in OSS projects, but because of the high level of automatic testing done at Ericsson, we consider it likely that the testing tools discover the same faults regardless of the number of contributors. However, it is also possible that the OSS projects in question, exercise automated testing as well.

As mentioned previously in the paper, a big factor that we have not studied but still need to address is developer expertise, which is known to impact the amount of defects in a component [2, 3, 4]. As a consequence, we have to assume that inexperienced contributors are more likely than experts to introduce bugs to the code in our case as well. However, we can not prove that developers with high ownership generate less faults as there are not many significant outliers in our data. It is obvious in files that have only a sole developer and very few faults but there is also a possibility that a file can have 10 contributors even though one contributor has an ownership of 90%. Since our data set consists of no files with a high number of contributors and a low amount of defects or vice versa, we have to assume that as long as the number of contributors increase, defects will always increase accordingly, regardless of ownership levels.

8. Threats to Validity

It should be noted that our data analysis cannot identify and compensate for confounding factors, such as the file age, complexity, churn rate that may be under-lying reasons that affect the correlation coefficient. We therefore encourage readers to be vary of the confounding factors and not to interpret the results in a literal sense as **correlation does not imply causation**. In particular, we suspect that older files are prone to age-bias, meaning that generations of developers could have contributed to the file and thus increased the contributor count while not currently working on that file. To compensate for this, we have looked at the data in two time intervals: one that accumulates all data from 2001 , and one that accumulates data for the period between 2013-2015. Another limitation is that we can not make any conclusions regarding latent defects that are not found. We have no guarantee that all bugs in the files are found.

All of the data is gathered using an internal tool at the company. Using this type of automated data collection, we have to account for the possibility of errors in the tool itself, the database, and the fact that our queries might be wrong. For this reason, we manually look into all inconsistencies and abnormal values in the data together with a domain expert at Ericsson in order to assess any potential threats.

We conducted the study at a large company with hundreds of employees, which means that there are most likely variations in development behaviour, both between teams within Ericsson but also compared to teams in similar companies. Additionally, we have no way of knowing how well our results are transferable to corporations in other software domains nor how well it translates to companies of other sizes. It primarily affects our study in the way commits are tagged. This is clearly shown in the defect fixes data where some choose to tag their improvement commits while some do not. This affects the defect data to some extent and we deal with it by looking only looking at defects of known severity. On the one hand, we do believe our results are likely to be transferable to any software company developing using similar team structures and development processes. On the other hand, when it comes to defects severity, every organization most likely uses their own definitions of severity classes and there is no guarantee that these results are directly transferable. Furthermore, severity tagging also runs a big risk of being wrongly reported. There is likely many cases where for example a class B fault should have been tagged as either class A or class B per definition. Regardless if our results our applicable or not, we provide enough details of our research methodology so that our study can be replicated at other companies.

9. Conclusion

We have examined how the number of contributors affects software quality at a large, proprietary telecommunications company. By performing bi-variate analysis tests on thousands of files we found that there is indeed a strong correlation between contributors and defects for each file. Similarly, we found that defect density also is correlated to the number of contributors, and that this correlation is stronger the larger the files were. Additionally, we found that the ratio of defect severity remains roughly the same as the amount of contributors to a file increases.

We concur with previous research, showing that the number of contributors has a negative impact on software quality. We provide new insight on how contributors affect defect density and defect severity, which had not been done before. Ultimately, we encourage other researchers to investigate the correlation between contributors, defect density and defect severity in order to produce similar results and provide additional insight on how these variables are correlated.

Acknowledgement

We would like to thank Ericsson AB and in particular Jesper Derehag for providing us with the data set and domain expertise. Furthermore, we thank our research group consisting of Rogardt Heldal, Regina Hebig, Michel Chaudron and Patrizio Pelliccione for helping us with our methodology and supporting us throughout the research process.

References

[1] D. Lo and S.-C. Khoo, "Software specification discovery: A new data mining approach," *NSF NGDM*, 2007.

[2] W. Fong Boh, S. A. Slaughter, and J. A. Espinosa, "Learning from experience in software development: A multilevel analysis," *Management Science*, vol. 53, no. 8, pp. 1315–1331, 2007.

[3] B. Curtis, H. Krasner, and N. Iscoe, "A field study of the software design process for large systems," *Communications of the ACM*, vol. 31, no. 11, pp. 1268–1287, 1988.

[4] T. Fritz, G. C. Murphy, and E. Hill, "Does a programmer's activity indicate knowledge of code?," in *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pp. 341–350, ACM, 2007.

[5] A. Mockus and D. M. Weiss, "Predicting risk of software changes," *Bell Labs Technical Journal*, vol. 5, no. 2, pp. 169–180, 2000.

[6] F. Rahman and P. Devanbu, "Ownership, experience and defects: a fine-grained study of authorship," in *Proceedings of the 33rd International Conference on Software Engineering*, pp. 491–500, ACM, 2011.

[7] N. Nagappan, B. Murphy, and V. Basili, "The influence of organizational structure on software quality: an empirical case study," in *Proceedings of the 30th international conference on Software engineering*, pp. 521–530, ACM, 2008.

[8] M. Cataldo, P. A. Wagstrom, J. D. Herbsleb, and K. M. Carley, "Identification of coordination requirements: implications for the design of collaboration and awareness tools," in *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pp. 353–362, ACM, 2006.

[9] F. P. Brooks, *The mythical man-month*, vol. 1995. Addison-Wesley Reading, MA, 1975.

[10] M. Pinzger, N. Nagappan, and B. Murphy, "Can developer-module networks predict failures?," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pp. 2–12, ACM, 2008.

[11] A. Meneely and L. Williams, "Secure open source collaboration: an empirical study of linus' law," in *Proceedings of the 16th ACM conference on Computer and communications security*, pp. 453–462, ACM, 2009.

[12] E. S. Raymond, *The Cathedral & the Bazaar: Musings on linux and open source by an accidental revolutionary.* " O'Reilly Media, Inc.", 2001.

[13] J.-H. Hoepman and B. Jacobs, "Increased security through open source," *Communications of the ACM*, vol. 50, no. 1, pp. 79–83, 2007.

[14] B. Witten, C. Landwehr, and M. Caloyannides, "Does open source improve system security?," *Software, IEEE*, vol. 18, no. 5, pp. 57–61, 2001.

[15] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't touch my code!: examining the effects of ownership on software quality," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pp. 4–14, ACM, 2011.

[16] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy, "Does distributed development affect software quality?: an empirical case study of windows vista," *Communications of the ACM*, vol. 52, no. 8, pp. 85–93, 2009.

[17] C. Bird, "Sociotechnical coordination and collaboration in open source software," in *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, pp. 568–573, IEEE, 2011.

[18] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "An analysis of the effect of code ownership on software quality across windows, eclipse, and firefox,"

[19] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities," *Software Engineering, IEEE Transactions on*, vol. 37, no. 6, pp. 772–787, 2011.

[20] D. Izquierdo-Cortázar, G. Robles, and J. M. González-Barahona, "Do more experienced developers introduce fewer bugs?," in *Open Source Systems: Long-Term Sustainability*, pp. 268–273, Springer, 2012.

[21] R. Van Solingen, V. Basili, G. Caldiera, and H. D. Rombach, "Goal question metric (gqm) approach," *Encyclopedia of Software Engineering*, 2002.

[22] S. H. Kan, *Metrics and models in software quality engineering*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[23] W. A. Florac, "Software quality measurement: A framework for counting problems and defects," tech. rep., DTIC Document, 1992.

[24] M. Sherriff and L. Williams, "Defect density estimation through verification and validation," in *The 6th Annual High Confidence Software and Systems Conference, Lithicum Heights, MD*, pp. 111–117, 2006.

[25] Y. K. Malaiya and J. Denton, "Estimating defect density using test coverage," *Rapport Technique CS-98-104, Colorado State University*, 1998.

[26] V. Antinyan, M. Staron, W. Meding, P. Osterstrom, E. Wikstrom, J. Wranker, A. Henriksson, and J. Hansson, "Identifying risky areas of software code in agile/lean software development: An industrial experience report," in *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*, pp. 154–163, IEEE, 2014.

[27] V. R. Basili, "Quantitative evaluation of software methodology.," tech. rep., DTIC Document, 1985.

[28] V. R. Basili and D. M. Weiss, "A methodology for collecting valid software engineering data," *Software Engineering, IEEE Transactions on*, no. 6, pp. 728–738, 1984.

[29] M. G. Kendall *et al.*, "The advanced theory of statistics.," *The advanced theory of statistics.*, no. 2nd Ed, 1946.

[30] D. Curran-Everett, "Explorations in statistics: correlation," *Advances in physiology education*, vol. 34, no. 4, pp. 186–191, 2010.

[31] E. Raymond, "The cathedral and the bazaar," *Knowledge, Technology & Policy*, vol. 12, no. 3, pp. 23–49, 1999.