



Mälardalen University
School of Innovation Design and Engineering
Västerås, Sweden

Thesis for the Degree of Master of Science in Software Engineering
30.0 credits

EFFECTIVENESS OF FAULT PREDICTION

Albi Dode
ade15002@student.mdh.se

Examiner: Jan Carlsson
jan.carlson@mdh.se
Mälardalen University, Västerås, Sweden

Supervisor: Adnan Causevic
adnan.causevic@mdh.se
Mälardalen University, Västerås, Sweden

Company supervisor: Xuan Xu,
Accedo Broadband AB, Sweden

May 11, 2018

Acknowledgements

*Personally, it was both a challenge and good experience for me in the professional aspect. A thanks go to my supervisor Adnan Causevic and my examiner Jan Carlson. **Special thanks go to my family. Their belief in my success was a crucial motivation for me to continue with the hard work and finalize this thesis.***

Abstract

The research community in software engineering is trying to find a way on how to achieve the goal of having a fault-free software. The industry that will use a near fault-free software will have it easier to lower the costs of maintenance and the versions of delivered software will be more qualitative. In this case, fault prediction can be used in order to achieve the above objectives. Fully applied fault prediction is not yet achieved on an industrial scale. There is some progress attained in the field during recent years. But knowing and understanding what available tools and algorithms regarding fault prediction can give is yet a goal to be achieved by the industry. In this thesis, two fault prediction algorithms and several metrics combinations are tested in an industrial and open source project.

The main goal is to understand how much fault prediction is integrated and effective in a continuous delivery environment using real case scenarios. The manually collected data, from several versions and in different time periods were applied using two already present algorithms: Naive Bayes and Clustering. As a result, while the usage of this prediction depends on the company needs, further research in the field can be extended.

Keywords: *software engineering, fault prediction, fault prediction algorithms*

Table of Contents

1	Introduction	5
1.1	Problem Formulation	5
1.2	Research Objective	5
1.3	Contribution	6
1.4	Chapters Overview	6
2	Background	7
2.1	State-of-Art	7
2.2	State-of-Practice	7
2.3	Motivation	8
3	Methodology	9
3.1	Research Question 1 (RQ1)	9
3.1.1	Project	9
3.1.2	Tools and Results	10
3.2	Research Question 2 (RQ2)	10
4	Related work	11
4.1	Literature Evaluation and Selection	11
4.2	Existing Studies	11
4.3	Innovations	12
5	An overview of the experimental setup	16
5.1	Collecting Metrics	16
5.1.1	Selecting the Proper Metrics	17
5.1.2	Metrics	18
5.2	Models	18
5.3	Mathematical means for results interpretation	19
5.3.1	Anova, ROC Curve and Area Under ROC curve	19
6	Used Fault Prediction Algorithms	20
6.1	Clustering using k-means	20
6.1.1	Distance Metrics	20
6.1.2	Finding the optimal K number	21
6.2	Naive Bayes	21
6.2.1	Disadvantages	22
7	Discussion	23
7.1	Metrics challenges and reasoning	23
7.1.1	What is industry looking for ?	24
7.1.2	Why is industry not always fully relying on fault prediction ?	25
8	Experimental results and conclusions	26
8.1	Experiments related to RQ2	26
8.2	Experiments related to RQ1	35
9	Answers to research questions	39
10	Future research and suggestions	41
10.0.1	Research Community	41
10.0.2	Company suggestion	41
	References	45

List of abbreviations

ACM Association for Computing Machinery

API Application Programming Interface

AUC Area Under the Curve

CD Continuous Delivery

IEEE Institute of Electrical and Electronics Engineers

LOC Lines of Code

OOP Object Oriented Programming

QA Quality Assurance

ROC Receiver Operating characteristic Curve

SSE Sum of Squares Error

TP True Positive

FP False Positive

1 Introduction

The code used for software programs is required to be of high quality. It undergoes lots of processes to find the faults. A fault free software can be achieved with the help of testing and Quality Assurance (QA) team's skills. Since the teams have started adopting continuous delivery for software development, time is considered as one of the decisive factor [1]. They tend to achieve a fault free software in minimal time without exceeding the deadline. But, the faults in the system tends to reappear resulting in the unexpected behavior of the software.

Several levels of testing are done to attain code that is free from faults. Relying on finding and fixing the faults is not enough. They tend to reappear in the system. The reappearance of faults depends on the code complexity, reuse of code and change due to increasing requirements. Fault prediction is considered of great help for the software companies into achieving a high level of quality for their products. Research community is interested on giving its contribution about finding how fault prediction can be done.

The interest towards the topic of fault prediction starts with Akyiama in 1971 [2]. As per definition [3], fault prediction does the identification of possible faulty code. For doing the prediction before the testing process starts, it relies on historical code data and code properties. Additionally, Catal in his literature review and current trends for fault prediction [4], states that fault prediction is still in the emerging phase and more research challenges exist in this field. The testing process commonly includes logging, collecting and analyzing of fault data. Illes et al. [5] and Derehad et al.[6] pointed out that fault data plays a major role in the process of fault prediction. This process helps in improving the software testing process of fault findings in less time. By identifying a method to analyze fault data, the testing and QA team will have better fault discovery ratio. It also helps in identifying the module or file in which the fault is going to most likely reappear.

1.1 Problem Formulation

Faults are present in the software due to abnormal conditions when a certain requirement is not achieved. The presence of these faults in code is a significant problem because it can lead to the code's inconsistency and unexpected behavior with regard to the functional requirements. In order to overcome such problem, the QA team needs to know in advance where the fault might reside and whether it will reappear. Fault prediction as a process can help to indicate possible faults inside the code and as such to help in removing them before the product release. This process will not only improve the code but also will save time for the code review.

In order to address the question on how to find and remove faults in less time, the fault reporting has to be detailed. In continuous delivery environments, ongoing testing should produce enough details for a fully working fault repository. After that, the problem of finding the right fault prediction tools and methods arises. In this thesis are investigated two existing algorithms based on an approach which makes use of several tools and metrics combination.

The main idea behind this approach is to find a near optimal fault prediction model, build upon continuous delivery metrics. Moreover, it is important to find a model which gives reasonable prediction results faster than manual software quality investigations regardless of the project type in study. Earlier the fault detection, means more time spent on other part of the system.

1.2 Research Objective

To propose and evaluate an approach in order to predict the fault appearance. Reaching of this objective is connected with the research questions as follows:

1. Which selected algorithm gives better predictions? (RQ1)
 - (a) Select appropriate fault prediction algorithms that can be applied in both type of projects using correlated metrics.
 - (b) Evaluate the accuracy of generated results, by checking whether the existing faults are identified by fault prediction algorithms.

2. Which are the most correlated metrics? (RQ2)
 - (a) Select appropriate metrics combination
 - (b) Evaluate the accuracy of generated results

1.3 Contribution

The main contribution of this thesis is summarized as:

- Building a new set of continuous delivery metrics combination which can be used with different prediction algorithms on different projects type.

1.4 Chapters Overview

The rest of this thesis is organized as follow. Section 2 includes state-of-art and background. Section 3 states the methodology that it was followed in this study. Section 4 summarizes the findings of similar previous studies. Section 5 presents the overview of used technical aspects. Details about the used algorithms are given in Section 6. Section 7 is the discussions section. Section 8 and Section 9 show the results and findings. Section 10 summarizes the suggestions.

2 Background

Code suffers from changes, which not necessarily do solve faults, but might also introduce new ones. As such, the faulty software is not released to the market. In order to verify the software, testing comes in place. Studying Lientz [7] the reader gets to know that a lot of effort is spent during this phase. Testing activities are necessary as they give more insight information on where a fault is going to happen or if the code is going to be faulty.

According to Pandey et al. [8], quality of testing is influenced by the working environment as well. Nowadays, more companies started adopting continuous delivery. As such, testing needs to be done frequently. Whenever a new code is uploaded to a common repository, manual and automated tests are performed to achieve a higher quality of code.

Knowing if a code will be really faulty or not is closely connected with the results produced by code investigation coming from previous fault data and the type of fault prediction algorithm used. If a fault is detected early in the developing process that will be the real result coming from the prediction algorithm. Having such a predictor which predicts real possible faults means that the rest of the time can be spent on other sectors of the code which will increase the quality of the product. Achieving this state of real prediction quality requires certain parameters coming from continuous examinations and right testing.

Catal and Diri state that fault prediction is a part of QA activities [9]. The data produced by their tests can be used in such a way that it can predict possible future faults appearance. The necessary data for building an algorithm can not be easily found. It needs a vast information coming from the logs of code. This information needs to be enriched with data frequently. The needed variables are called *fault metric*. For making this work, a data repository is needed. It will allow recovery and study of past versions of the code, allowing for the determination of changes made with each successive release. Throughout the scientific work, it is agreed that a fault database which records previously detected faults is needed.

2.1 State-of-Art

Many of the research contributors have focused their interest on the needed variables that can make a fault prediction algorithm work. Based on that, they conducted their research towards predicting where the fault is going to be, or if file/module is going to be faulty and how many faults there will be. What Hall et al. were trying to bring out as one of the findings is the fact that when data variables are combined in a proper mathematical model better results are achieved [10]. According to Rashid et al [11] having related information improves the programs. Based on that, supervised and unsupervised fault prediction algorithms were made. Supervised includes artificial intelligence techniques such as logistic regression, bayesian model, discriminant analysis, classification tree, neural network, Support Vector Machine, Case Based Reasoning. In the case when training data are in absence, unsupervised techniques take place, such as clustering.

2.2 State-of-Practice

Testing as a process plays a major role in the software industry. It is necessary to test a software as the discovery of faults can be made possible. Fault prediction helps in this case as before hand it will say if the code is going to be faulty or not. However the effectiveness of fault prediction can be questioned, so the motivation for this thesis would be to utilize different available algorithms in a continuous delivery environment and see their effectiveness.

Before a software is released a series of testing are done. They can be manual or automatic testing. In the manual form, the test cases are performed by the tester manually. As a state-of-practice, software companies use a fault reporting system where the reporting of the fault and steps for reproducing it are written there. In this way the stakeholders are notified for the fault and also the developers can report their fixes. Nearly the same steps

are reproduced even for the automatic testing, but in this case, testing is done via specific scripts.

Another thing to be pointed out is that the number of test cases keeps updating time after time. There are added cases to be tested in a specific software or in all the software. Moreover keeping up with the requirements in a timely manner is not as easy as a reported, as fault might require more time to be solved than previously thought. Sometimes the faults are tagged as blocked due to third party dependencies. The solution to that type of fault is given at a later time.

2.3 Motivation

The main motivation of this thesis was on finding an answer about how to achieve more from testing data. Software testing, as part of Continuous Delivery (CD) helps in finding faults before the software is released to the market. As such, by fully using the benefits of testing, the other processes of software development become more qualitative. The usage of a fair fault prediction algorithm in a continuous delivery environment is the final thesis goal. By reading several articles, it can be seen that the interest for the fault prediction topic during the last years is increasing. Even though the number of contributions is big, the same number holds true even for unsolved issues and future improvements that can be done in the field. It is interesting how the evolution was done from simple metrics to more complex algorithms in recent years. Most of the work is based on data gathered from a long period after release. The type of data that certain algorithms require is different from algorithm to algorithm. Some rely on testing metrics, others on process quality data. Finding the right set of data for achieving higher fault prediction performance is yet to be achieved. For the above-mentioned reason, there is a need for a study which deals with continuous delivery environment and which tells the real result of the prediction.

The work for the industrial project was done at Accedo Broadband AB. The reasons behind this decision were:

- (a) To study fault prediction in a continuous delivery industrial environment.
- (b) Point out which theoretical algorithm fits best in a company dealing with continuous delivery.
- (c) Fetch complex data from the project which fit best for the fault prediction model.
- (d) Result taken from an industrial company can be further extended by the research community.

3 Methodology

In this thesis, qualitative and quantitative research designs were adopted to achieve better results. In order to get more insight information for the fault prediction, the literature study methodology was conducted. A careful study in this field will give more insights on what are the current trends in fault prediction and how they can help to our solution. In addition, an highlighting of related work and existing approaches with respect to the fault prediction models was done. Close attention is paid to finding innovative tools and algorithms dealing with fault prediction.

Literature review, Eclipse open-source project and an industrial project at Aceedo Broadband AB were thought to be used in this thesis. The case study is done in order to see in a real case what is happening and also trying to find explanations on the raised questions. As in our case, a comparison of two different algorithm results was made. Based on Robert Stake suggestions [12], the triangulation method was used which includes the usage of more than one data source, data collection method, and different theories.

3.1 Research Question 1 (RQ1)

For answering this question (Section 1.2), the below mentioned steps were followed, as it is important to understand what are the available algorithms and tools that can be used in our case study (Section 4). Later, the algorithms were applied and the predicted results were compared in order to see how the results turned to be in reality. The steps followed are explained in more details below.

- (a) Find related projects: Find projects which have a big number of faults.
- (b) Collect data: Data collection based on the used metrics (as in section 5.2).
- (c) Apply algorithms: Find easy to use fault prediction algorithms (as in section 6).
- (d) Interpret findings: Try to give an answer to the research question (as in section 9).

3.1.1 Project

Projects will be selected according to some criteria depicted below:

- Project's start period: Project whose first release date was at least 6 months old was chosen in order to collect as much data as possible. An analyzation of the selected projects was done in order to manually gather as much as possible data from a certain time period. The impact of CD can be seen in the fact that they have more tickets and faults in Jira, SonarQube, and Github. Also, from the file's time-stamps, we could see that the work integration is done in a frequent way, according to the Agile manifesto.
- Number of faults per category: Since the projects have several modules, the module of the project which had more faults in frequent times was selected.
- Number of releases: The project which release followed CD was selected. More frequent releases means more solved faults, so our algorithms can work better with more information in hand.
- Project's programming language: The selected projects were coded in Java. As Java is Object Oriented Programming (OOP) more data can be collected. This is due to the dependencies that open source tools have. Freely available metrics collection tools work best with OOP code rather than other programming languages.

3.1.2 Tools and Results

In order to answer at RQ1 (Section 1.2) a data set is needed to be built. Building that data set requires a collection of metrics. The collection can be done by using certain tools due to the necessity of particular metrics. In order to pick the right tools we followed some criteria.

- Is the tool free or commercial?
- What type of metrics it can collect?
- Is the data collection automatic or manual?
- Does it have programming language dependency?

To apply the algorithms and interpret the findings, the following steps were followed:

- Checking the tool's results.
- Comparing the predicted results with the results obtained from the data collection.

3.2 Research Question 2 (RQ2)

This question (Section 1.2) mainly deals with finding the correlated metrics and then used for building the data set that the algorithms use. For this reason, the following activities were conducted:

- (a) From the literature review, see what type of metrics have been used and which are new (from section 4.2).
- (b) For each of them, find more explanation regarding advantages, disadvantages, and correlation.
- (c) Considering the used code, which metrics can be collected - There are metrics which can measure the impact of CD and help in the process of forecasting needed changes and help in the decisions needed to be taken for achieving it. Metrics were selected according to a detailed state-of-art review and on what fits best with the nature of the studied projects, and their correlation impact.
- (d) Several correlation experiments, as listed below, were conducted in order to see from the vast metrics combinations, which are the highly correlated and what combination to use. These experiments and combinations for correlation are not shown in details during this thesis:

Table 1: Other correlation experiments

Collected/Tested period	Focus	Metrics combination
Between the sprints	Project and process	Contributors,faults,commits,interest,last commit
Weekly	Static	Effort, nr.faults,loc,functions,classes
Monthly	Project and Process	New lines added, nr of faults, component, repeated
Releases	Project and Process + Static	Fault priority,type,component,contributors,related

The full list of metrics and their explanation is shown at section 5.1.2. The analysis for these experiments was the same as shown in the section 8.1. It was noticed that during these experiments, related to RQ2 (Section 1.2), results showed a Receiver Operating characteristic Curve (ROC) area and True Positive (TP) slightly just above 0.55, Sum of Squares Error (SSE) and incorrectly classifications were too high. Many metrics needed to be substituted with others which hold more importance. Also, another influential factor was the fact that the collected information in some experiments was not enough to make the test run. The most correlated metrics among these different tests and periods seemed to be all of static metrics (number of functions, classes, loc), commits and effort. The majority of them and their tests are shown and used in section 8.1, in more details. To be noticed, is the fact that in section 5.2, the metrics combination came as a more close collaboration with the QA team.

4 Related work

4.1 Literature Evaluation and Selection

For finding our related papers in the field, a search from the Institute of Electrical and Electronics Engineers (IEEE), Association for Computing Machinery (ACM), Science Direct databases was performed. In total, there were found 193 materials related to our field of study, including proceedings, articles, books. The research was done using keywords such as fault prediction, prediction model and also used bibliography on the found materials.

Our review differs from existing reviews in the following ways:

- (a) Time frames: Our review is contemporary as it mostly includes studies published from 2000-2017.
- (b) Innovation: This thesis focuses in new tools, algorithms.

4.2 Existing Studies

The research community is focused on achieving good results on finding the way towards the ideal fault prediction. In order to get reliable results, continuous research should be made. This is the case of Ostrand et al. [13], [14] which required nearly 10 years to achieve good results in the field of fault prediction. They had to study both agile and waterfall methodology in order to be able to collect relevant metrics for their study which was based only on files versions from different releases. Staying on the metrics studies, Hassan [15] demonstrate that complexity metrics should be the ones to be collected in cases when files properties are studied. In difference from Ostrand et al. [14], Hassan states that the complexity metrics are dependent only on the complexity of changes that a file has not to any other reason.

In details, related to the importance of metrics and the time when they should be collected, it was written by Khoshgoftaar et al [16]. Insights are given on the impact that the metrics type can bring on the findings related to the fault prediction methodology. They found different results when different metrics were used compared to the case when it was opted for only one type of metrics to be used. Metrics and their information are considered important and should not be only of one type. Following the same study nature, Jiang et al. [17] and Li and Reformat [18] concluded the same thing, that a right combination of different metrics should be found in order for a model to work.

The right combination of metrics depends on how the data collection is achieved. Catal and Diri [9] point out that if a project has or has not faulty labeled data, semi-supervised approaches should be followed. But this approach is not so much of popularity. In his later studies, Catal [19] gives as a reasoning that the data collection can be made only on the modules where the majority of faults reside at. Yet, they warn researchers to pay attention to the results that were achieved in different releases. Maybe Catal [19] was influenced by the paper of Bell et al. [20] which was published only 1 year before. Bell et al. [20] give details on how metrics can be collected from previous releases. From the code history repositories, they were able to collect metrics coming from different stages of the code lifecycle. Singh and Salaria [21] and Kaur et al. [22] state that only if repositories are available we can apply models which rely on early life cycle metrics. For this, they invite researchers to compare results from different life cycle stages and algorithms. About the code data storage, Kaur et al. [23] needed to study six algorithms and models to give as a finding that if faults were stored in advance the results can change. In conclusion, if releases are stored with details, we can make categorization of modules and also we can try the labeling of the files predicted as faulty or not. Papers of the same year, 2016, such as Liu et al [24], Rathore and Kumar [25], Tahvili et al [26] tell the same thing. They add to the fact that feature selection process from these repositories is the next concern in the field. They try to argue this with related math formulas and state that this selection should be based on the nature of the data set created and if a previous data pre-processing has been done.

Staying on the code data repositories importance, five years after Kaur et al. [22] paper was published, Mahajan, Gupta and Bedi [27] argued by comparing different fault prediction algorithm's results and fault prediction techniques, that only with big repositories we can do relevant comparisons and have meaningful fault prediction results. Having a repository, it means having different metrics to choose from and as such different models can be built [28]. In recent years, several studies were conducted having in focus what is an effective size of a code repository related to fault prediction. Abaei and Salamat [29] try to understand results they got from both a small and large data set. Studying a repository, done by White et al [30], also shows the evolution that a code has suffered is important. Comparing code evolution, by following Erturk and Sezer [31] and strategies coming from Aupy et al. [32] and Adeline et al. [33], we can understand when is the right time of doing a prediction related to the repository size.

4.3 Innovations

In the presented tables it was seen a trend that tools are either built from scratch or some commercial statistical tools are used instead. This might be a reason for the non-success of the proposed fault prediction algorithm. The necessity for such tools is becoming an emerging problem since yet there is no tool that can be applied with all the possible algorithms. Only recently the research community is investing in this aspect, even though the fault prediction is not a new topic. We can understand the need for a tool, not just by analyzing the available papers, but also with the emerging number of projects, mentioned in the related work section, which aim is to only built a tool.

Ostrand et al. [13] tool can be named among the pioneers in the field of fault prediction. The perfection of this tool required 10 years. Compared to what a commercial company could have been done with 10 years makes this tool and the algorithm which comes with it inferior. But its superiority is seen in the simplicity that the prediction done is based on a simple math formula. By carefully analyzing the steps needed for producing this tool and algorithm, it can be understood the developing difficulties.

From our studied sources, this is the only case when a group of authors write in a scientific way and briefly touches the myth that large industrial study cases cannot be accessed by academic researchers. Seeing this tool requirements from an industrial prospect, it brings out the fact that a good team structure, which perfectly manages the changes in a code, and a version control system (data repository) are needed. In this side, the tool and algorithm behind it, cannot be applied in small teams for example or even big teams which usually do not have in place such systems.

Difficulties in the usage have also the innovative algorithm presented by Liu at al. [42] which is a genetic algorithm requiring supervised learning. A supervised learning means to require a data set from previous faults as a way to train the model, achieving it is something that can not be easily done by all the companies and their team structure. But a company can use one of the novel combinations that Suresh in [40] proposes, fewer test executions and more focus on test dependencies brought up in the discussion only certain metrics related to fault prediction. This was not enough to produce reliable results.

Even in their novel algorithm, the authors Abaei and Selamat [45] just briefly mention one particular bad aspect of testing the effectiveness of fault prediction. During their testings, they had to repeat the process three times just to be sure everything was working right. This can not happen in an industrial environment where time is crucial. Plus during these extra repetitions. In reality, conditions tend to change between times. The article also does not make any precision about the needed metrics, they just say they follow the rules of imposed by the commercial tools they used. The industry needs to know what is needed to make the algorithm work. In the same path are also the authors Diamantopoulos et al. [36] and Scanniello et al. [37].

A very recent novel algorithm presented by Okutan et al. [43], from 2016, tries to study the prediction of faults derived by focusing on how many time code copying has happened. It is

Table 2: Available tools from researchers and industry - Related to metrics gathering

Name	Developer	Year	Granularity	Output
Visualize code	scitools.com	2012	File	Complexity
Prest	Ekrem Kocaguneli, Ayse Tosun, Ayse Bener	2009	File	Complexity
Source Meter	sourcemeeter.com	2014	Different	Cohesion, Complexity, Coupling metrics, Documentation, Inheritance, Size, Duplicates, Rule violations
SmartTrace	BASHAR NASSAR	2016	System	Design metrics component pass/failed test cases, component interaction
McCabe IQ	mccabe.com	1990	Flow graph	Design metrics and McCabe Cyclomatic Complexity
CKJM	dmst.aueb.gr	2005	File	Method metrics, Class relations, Cohesion, Coupling
Understand c++/java	scitools.com	2005	method+class	Complexity
Locmeter	locmetrics.com	2006	File	McCabe VG complexity, LOC
Git	github.com	2009	Different	Reports
Jira	jira.com	2002	Ticket system	Reports
Sonar Qube	sonarqube.org	2006	File	Reports
Jenkins	jenkins.io	2011	File	Reports

Table 3: Available tools from researchers and industry - Related to fault prediction

Name	Requirements
Ruby(based on lacheis)	Eclipse plug in only java; Method-level metrics Halstead metrics, McCabe metrics Cyclomatic complexity Lines of code Class metrics ChidamberKemerer metrics, Coupling, LOC
Levenberg-Marquardt (LM)	Class metrics, Coupling, Cohesion, Cyclomatic complexity, LOC.
Eclipse plugin-elaine	Files with most fault density
WEKA, R, PYTHON, SPSS, MATLAB	Only for existing algorithms (clustering, time series, linear regression, etc) - Extensions only via external libraries or modify API.

Table 4: Innovative algorithms

Author	Data-set tested	Metrics	Math	Year
[34]	NASA MDP dataset named PC1	lines of code, McCabe, Halstead	Nave Bayes+Fuzzy	2015
[35]	Tel Com, Eclipse, Apache	mutation metrics	Nave Bayes, Logistic Regression, J48, Random Forest	2016
[14]	AT&T files	LOC, file age, prior faults, change status, programming language, release	Negative binomial regression	2013
[36]	Eclipse	change metrics , temporal metrics, people metrics, churn metrics , binary metric	Genetic Algorithm	2015
[37]	Xerces,Xalan, Velocity,Synapse,POI ,Lucene,Jedit,ANT	Class metrics, Coupling, Cohesion, Lines of Code	BorderFlow algorithm	2013
[38]	NASA	LOC, cyclomatic complexity	k-means and Neural-Gas clustering algorithms	2007
[39]	Eclipse,Nasa	Features ranked from size,similarity	no info	2014
[40]	Apache	Chidamber and Kemerer metrics suite	Genetic algorithm,Particle swarm optimization, Neural network	2015
[41]	Medical Imaging System dataset	Coding lines,comments,Halstead ,Jensen estimator ,McCabe cyclomatic	Euclidean distance	2016
[42]	no info	genetic algorithm, back propagation neural network	no info	2016
[43]	no info	vector machine	no info	2016
[44]	proomise,camel, tomcat, poi, xalan, jedit, velocity, ant, lucene, synapse ,ivy	euclidian distance	no info	2015
[45]	NASA, TURKEY Telecom	class,cohesion, coupling, complexity	fuzzy clustering	2015
[46]	NASA	method metrics	no info	2013

a topic which is new in the field. Maybe the idea was taken from the algorithms which deal with cross-project learning, but there are no references to any of it. Still, an explanation for it can be that even in industry, as the algorithm was tested in large open source projects, also, there is the tendency to copy paste code between files. In this perspective, the algorithm will have success into being implemented as the copying of code is a widespread phenomenon. Still, focusing only on one aspect and one metrics, bring biased predictions as previously stated in the related work section.

A new way of thinking for the code used in web development is presented by Chatterjee et al. [44]. The focus was on the math behind it rather than technical data explanation. Plus, the authors did not mention clearly what metrics are needed but only include some explanations about different responses they got from a web page such as error 404 or error 300. Their job is innovative as they are among the few ones which have brought up fault prediction on the web programming side.

The missing previous data were also the starting point for Seliva to conduct a research on the topic [38]. Instead of hiring an expert on labeling fault, not fault, the semi-supervised learning seems to be a good solution. Testing is done in NASA sets but the result is yet to be explored in an industry.

In their study, Kakkar et al. [34] go for a hybrid algorithm between k-means and neural network, but it requires 22 metrics. The accuracy achieves an impressive number of 97 % but no details are given on the way how the study was conducted. In this way, this algorithm does not fulfill the requirements that an industry is looking for a simple working algorithm.

Authors Bowes et al. [35] present in their article a novel algorithm derived from mutation testing. The given details about the tools used make it easy to read compared to previous similar contributions. The tools used are not being fully related to fault prediction but to metric collection offer a wider range of collectible metrics.

Published in 2013, Catal [46] still it is innovative as his contribution deals with the fact what happens when fault data for previous versions are not present. This is an algorithm which should be considered for the companies which have this type of problem when the proposed supervised algorithms can not be applied. And the special property of this algorithm proposal is that there it is not needed human interaction so everything is automated. This algorithm can be considered good as it is among the few ones that rely on threshold value changes. The authors go with an innovative algorithm for doing the removing of irrelevant information. In this way, the data set trained has only what is needed. They make use of Symmetrical uncertainty which has not been seen to be used anywhere else in our studied references. The same concept, that of learning what is needed or not, holds true for an algorithm based on a modified clustering version: to learn on its way the needed information. That's the work of Yin et al. [41] and Chen et al. [39].

5 An overview of the experimental setup

In this section, the focus is on building the data set model with the project under test. Focusing on certain metrics is done according to several coding and testing behavior that Eclipse and Accedo Broadband AB follows. It will include also some technical explanations about the metrics used, tools for collecting such metrics and explaining the mathematical terminologies used in this paper.

5.1 Collecting Metrics

For the metric collection process, the steps defined on the Methodology section 1.2) are carefully considered. In this study, it was opted for choosing OOP projects in order to benefit from several freely available tools supporting the data collection process. On the other hand, the available tools for non-OOP projects offered a much narrower spectrum of our necessary metrics. The data were collected from various modules of these company's projects code: Eclipse and Accedo Broadband AB. Jira and SonarQube were used for the Accedo's project data collection. For Eclipse, data were collected using these freely available websites:

- <https://bugs.eclipse.org/>
- <https://github.com/eclipse/>
- <https://git.eclipse.org/r/#/q/status:open>

For some of these modules the data collection process covered periods of a few days, while for some other modules the data collection covered periods of up to two months. This was due to the fact that the fault generation rate was varying depending on the nature of different modules. During this process, it was noticed that there existed a connection between faults being marked as severe ones, static metrics and times that fault has been reported. As such, we manually spotted those files and analyzed their code and gathered our metrics using freely available tools such as SourceMeter and Prest. The used projects were carefully monitored for the modules with most faults reports. At Accedo Broadband AB, the data collection process also included semi-formal interviews in order to understand the trends and management of faults. For some tools, it was necessary the help of QA employees. Here it was noticed that there was a connection between time spent on solving a fault and the number of commits and code changes that module had. This process was done during one month period. This period was necessary in order to see the progress that the fault suffered and to build such a fault database with necessary faults to start the work with. Tools utilized here were: WEKA, R programming language and SPSS in order to check the prediction results. Tool dependencies on other mathematical libraries were used separately in order to better analyze the data. There were made several metrics combinations in regarding RQ2 (section 1.2)), as shown in 5.2, and as such it was in this step that the period for data collection was found to be more than one week. A validation of the results of those combinations with the manual investigation and the obtained results is shown. Attention was paid while collecting the data:

- Carefully investigated what were those faulty components, which project used objected oriented programming language, in order to access code metrics.
- See for available combinations in data sets, if possible, which have not been previously studied by other research in the field.
- Check if faults were having missing information.
- Assure that information reported was recent (not more than 1 year old).

5.1.1 Selecting the Proper Metrics

The first considerations were regarding what type of metrics can be collected. As being part of the research question, software metrics got a lot of attention. In different literature, they are categorized into two types: pre and post release metrics. The definition given to these types is that in pre-release the metrics are needed to prevent faults reach the customer and in post-release generally with customer feedback. Due to the nature of our study, more focus was in both of them. Considering Halstead et al. [47], for this case, static metrics were praised as being the right one to begin work with. These metrics can give us more insights about the code properties. There can be static code metrics or another type, the right balance has to be found as what information continuous delivery can give in relation to fault prediction. Furthermore, Nagappan Nachiappan and Thomas Ball [48] state that even for large systems, static code attributes are faster to collect. But it also added the fact that they can find certain faults, not all the varieties of them. In this phase, here will try to search for the right balance between metrics available and collectible. After that, there was an agreement regarding the types of attributes granularity and where to focus the fault prediction at. In the end, it was decided to first start at the module level.

In Chidamber's article[49] were proposed some object oriented metrics which can be easily used. The process of standardizing those metrics seems to be ongoing. The variations stay within the context of measuring the code properties. The need for new type of metrics was requested by the fact that the so far used metrics did not fully reflect the object oriented perspective of the code [50]. As a result, future researchers need not be biased and must avoid presenting metrics that lack theoretical foundations and which are too general.

By analyzing what properties are included in static code analysis, it is obvious that these type of metrics help in better understanding the code complexity. Instead of a just manual review, code analyze is done thanks to manual and automated tools. In this way, the information can be taken from different levels such as a file, folder, module level. This is what makes it preferable and easy to get. As Catal has mentioned several times in his findings [19], the usage of static code is advised. It is seen that their usefulness is praised both by researchers and industries together [51].

A file is considered to be the building unit for the modules and the functionality of the project. Also, the majority of software that helped during metric collection worked best in file based data collection. A factor that influenced this decision is that faults reside in file or files. In this way, we collected, thanks to Jira, Bugzilla, and SonarQube, the static metrics (first three metrics in the list below).

Project and process type of metrics are considered as not easy to be collected because they require manual collection work. Both they say a lot about the situation of the project over time. By citing Futong and Tingting [52] we see that it is advised to include some metrics which can count the level of commitment that the team is having towards the project. It also helps in fulfilling the requirements that the ISO 9001:2015 has regarding the responsibilities aspects [53]. The ISO 9001:2015 has in focus a satisfied customer and continuous improvement, which are the basis of Agile but also for continuous delivery in our case. A good paper on process metrics is that of Biljana [54] where these type of metrics are marked as a good evaluator for the post-delivery stage. This paper goes on the same path with the paper from Ali and Borstler [55] where it is stated that process metrics are a good measure of the level of experience and time measurements.

5.1.2 Metrics

This list encapsulates all the metrics used, also in those experiments presented in table 1, three from static nature and twelve from process and project:

- Number of classes: Number of classes contained in a faulty file of the studied project's faulty module.
- Number of functions: Number of functions contained in a faulty class on the same faulty file of the studied project's faulty module.
- Number of statements(LOC+comments): Number of statements contained in a faulty file of the studied project's faulty module.
- Number of faults: Actual faults a project's file had.
- Effort: Time between last appearing of the fault and till solving it.
- LOC: Lines of code in a faulty file.
- Contributors: Number of developers changing the code and doing proper commits, after fault has been solved, to the code repository.
- Time since last commit: When the file was last changed due to fault, but not necessary committed. This as fault might be still faulty with minor bugs.
- Number of commits: Total number of commits needed for solving that faulty file.
- Number of lines added/removed: Changes in lines of code for the faulty file, due to fault solving tentativeness.
- Repeated: If fault has reappeared in that file.
- Files related: Number of different files which the faulty file depends or makes use of.
- Priority: Fault ranking according to the severity and consequences to the overall system performance as low, high or blocking.
- Type: Type of fault as reported by the QA team.
- Interest: How many people were seeing the fault report in Jira/Bugzilla, but not necessarily involved into solving it.

5.2 Models

In this section, the experiments based on the above metrics are shown. Models differ due to their project type nature and to the availability of certain metrics. Some metrics, such as: time since last commit were not used and the combinations differ. Later, based on these we found the best correlation combination and applied the algorithms with the respective metrics. The models for correlation built are as below:

Table 5: Models

Project type	Focus	Metrics
Accedo+Eclipse	Process	Added,removed,files,interest,related,repeated
Accedo+Eclipse	Static+Project	Loc,effort(years),modified,issues,contributors,commits
Accedo+Eclipse	Static+Process	Classes,effort,faults,statements,loc,files,functions

5.3 Mathematical means for results interpretation

5.3.1 Anova, ROC Curve and Area Under ROC curve

Anova is used for making the analysis of variance. It is a statistical method for comparing groups (in our case of variable) according to their means and try to find differences among them [56]. Such variables, are the ones which are measured in different time intervals (as we did with weekly, monthly, sprint wise) 1. From this, we get the F value which represents the variability among groups and inside the group. When F has the value 1, in this case, it means that there are no differences among our studied groups. Another important value is p which is the percentage of error for giving a conclusion that there is a mean difference when in fact there is not. If $p \leq 0.05$ it means that there is a significant mean difference among some of the means.

Both ROC and Area Under the Curve (AUC) serve as techniques for evaluating a model's classification [57]. The accuracy of ROC can be understood via the AUC numerical values, which is another way of ROC's values interpretation. The ROC points in the graph are in the form of a curve which passes from points 0,0 to 1,1. If the results are closer to 1 and in the upper part of the left area, it means better predictive model. This value is composed by TP and False Positive (FP) represented [58], where the Y axis has the values from TP and x -axis from the FP.

Theoretically, we have a more accurate graph if the curve is closer to the left side and in the top direction. If the prediction relies on 0,0 and 1,1 that shows no valuable information. As such, the points in this region should be higher than 0.5 in order to be considered. Values between 0.75 and 1 are more preferred for critical systems, but as Catal [19] says the risk of false alarms in this region is also high. Below is an image to show a simple ROC graph.

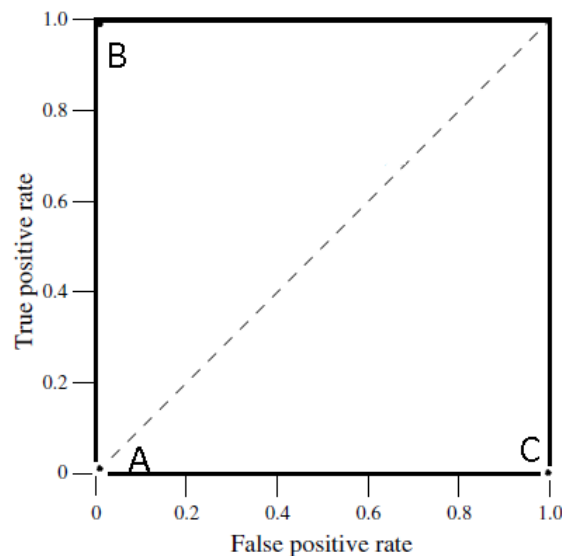


Figure 1: ROC example graph

Some main properties of this graph's points can be summarized as:

- A=(0,0) or the point where no positive classification is done
- B=(0,1) perfect classification
- C=(1,0) commits false positive and gains true positive
- Majority of points it is better to be on the side of the graph where the ration of TP is higher but FP is lower.
- If point reside on the right upper side have the tendency to have false positives.

6 Used Fault Prediction Algorithms

Fault prediction models, based on given algorithms, will be constructed and their output will be further analyzed. In this thesis, Naive Bayes and Clustering approaches will be used. With the given collected data sets, the accuracy of prediction in a continuous delivery environment will be tested given certain metrics. The work done will base the evaluations on the data interpretations taken from the WEKA tool, SPSS and R programming language. In this way, the training of the model will depend on the collected metrics during different development phases.

6.1 Clustering using k-means

Part of the unsupervised learning algorithms, clustering algorithm groups data in such a way that patterns can be defined out from the data properties. In this way, every data attribute is needed during the data analyzing. They are needed for calculating the cluster's centroids distance. That distance will show if those points are closer to each other and similar, they are under the same cluster.

By reading the findings of Ganguli [59], an introduction was made to the fact that with clustering we can get more information regarding our research question number 2: metrics (Section 1.2). They conducted a study where different clusters having different metrics were compared respectively to their error rate prediction. The findings there seem to be a perfect fit with one of our goals. Another reason why we chose clustering is the fact that [59] shows the existence of a tendency for the majority of faults present on certain files. This was enough evidence that clustering of faults based on properties can be done. As said in a conference [60] K-means is found to be efficient. Among the researchers that have used this algorithm, in the conference it was mentioned that the speed of calculations seems to be an advantage among other properties for this algorithm. All these citations made us consider working with it.

As per algorithm definition, the data set will have k - clusters, taken as an input value and based on the used distance measure the grouping will be done. The cluster will have data points which the mean squared distance between the cluster's center is minimal. In a short summary, this process includes a well-defined centroid. The cluster's distance is measured as the distance between centroids. To better summarize the algorithm steps:

- (a) Define K as total number of clusters.
- (b) Cluster's centers are randomly selected.
- (c) In a loop, for above points find Euclidean distance between instances. Put instance to the cluster with the smallest distance. Re-calculate this steps.
- (d) Repeat till convergence or till threshold.

6.1.1 Distance Metrics

Euclidean distance is among the important parameters needed for building the cluster. The clustering process itself relies on it and on the properties it gets from the points. In the end, this distance will help on the process of minimization of the distance between the point and centroid. In contradiction to many literature and research that have used it, Bouhmala [61] has found that Euclidean distance is not a real estimator of the cluster attributes. This fact should be considered for further research as said in the article, which is a recent one from 2016. In relation to our study, we will go with Euclidean as till now no new improvements have been done to it.

An interesting fact to be considered for the validity of our results should be the standardization of values. In his work, Bouhmala [61] brings out this concept as many metrics have a different numerical scale. The benefit of normalization is emphasized also by Mohamad and Dauda [62]. There it is argued that it is a must. As even for this process there is no standard to use, the normalization process is left to be done by the software of the tools we are using.

6.1.2 Finding the optimal K number

As the rest of the calculations depend on the random number of K and the fact that a convergence must be achieved in order for the calculations to stop, results are not the same in each iteration. The process of iteration is also dependent on the changes that the mean value suffers. In summary, beforehand, the number of clusters must be known. The iterative process of reaching a convergence between centroids is considered a minus. After this is done then we have the right number of clusters. But considering the number of clusters as known is not a real scenario.

There exist different suggestions regarding this problem. If we go with what our tools give us than one option is cross-validation. With it, we divide our data-set once in 10 pieces (10 is by default, but that number can be changed). As a next step, the training data-set is formed out of those 9 pieces and the last one is used for testing. This process is repeated 10 times and in each time a different segmentation division is done. In the end, we average the results in which every data one time is used for testing and the rest of times 9, for training. What we get as an output result is the eleventh time of algorithm running. In this finale time, it runs on the whole data-set and here the results are evaluated and errors estimated. This is a suggested case when the data-set is not big [63].

In their article, the authors [64] state that if we want to play with the k number, a good domain knowledge must be known beforehand. A method that gives the possibility to play with the number of clusters is elbow method. The differences that the increase of number k brings to the sum of squared errors is the main idea of this function. For better visualization, here it is needed to plot a graph and see that after some k number the sum of squared errors seems to be constant. That is the right number k. In the tool that is used, attention should be paid to the sum of squared errors. As previously mentioned, elbow method is closely related to SSE. The comparison of this value in different clusters gives an idea about the right number of K. Mathematically SSE calculates the distance of points in a cluster with the clusters centroid. Ideally, SSE should be small as the clusters increase. In reality, in our case, it turned out difficult to use the tool's option, as the k number was not easy to be deduced from the graph.

Another variable which impacts the output is the seed value. Its property is connected with the initial centers of centroids. As initial centers are randomly selected, the seed value acts as the initializer for the random generation. The seed value is also connected with the converging of the cluster. This comes from the fact that the centers that were initially set from the seed can produce some non-true results.

6.2 Naive Bayes

Naive Bayes is defined to be a simple supervised algorithm. As it relies on the Bayes theorem, its calculations are based on probabilities. Based on a citation at Sankar, Kannan and Jennifer [65], it was understandable that among different machine learning algorithms there is not much of difference in prediction results. Also, Naive Bayes requires no optimization. Following these two facts, it was opted to work with Nave Bayes, as it is an algorithm which calculations are based on probability and Baye's theorem. Part of supervised learning family of algorithms, it is considered as easy to be trained due to the fact that events are unrelated and independent among them. In this thesis, where the data set is built from scratch, Nave Bayes is good as it can work with fewer data.

Using collected metrics, a prediction is made taking in consideration the modules posterior probabilities. As we will deal with prediction, this is a good case as we get to know what is the probability of encountering something given the knowledge of something happened before. These assumptions have a property the fact that predictors are independent of each other. If we compare it with clustering, here there are no iterative parameters or processes. The probability nature of this algorithm makes it take into consideration independently the class and the feature in a study. That is a reason why it is considered stable. Before starting with explaining the formula, some definitions are given:

- Frequency table : Table representation for the times a data has occurred.
- Likelihood table : Table representation where the probability of the data is predicted given the happening of the class.

Math behind

$$P(c | x) = \frac{P(x | c) P(c)}{P(x)}$$

What this formula is finding is the posterior probability. $P\{x | c\}$ is the value of the likelihood ($P\{c | x\}$) that x will appear while belonging to class c. $P\{c\}$ is the value for the prior probability. From the formula we can distinguish the independence property, X does not depend on other values. The rest of the steps for the algorithm execution is as follows:

- (a) Given target build frequency table.
- (b) Transform it as a likelihood table.
- (c) Prediction is based on the value having the biggest posterior probability.

$$P\{c | x\} > P\{d | x\}$$

categorize as c else categorize as d

6.2.1 Disadvantages

The biggest disadvantage topic for Naive Bayes is data property. As previously mentioned, this type of classifier is very sensitive to the metric's independence. The metrics, for this reason, need to be carefully chosen. Another influential factor is the normalization of values. We have to overcome one of the main problems that of conditional metrics. The metrics in a study should be less and independent. Naive Bayes imposes also some limitations to the type of data we need to consider, that of categorical data. If we want to use mixed type, in the available tools is not yet available. For this reason, our tests will be done what our tools found by us.

7 Discussion

7.1 Metrics challenges and reasoning

During our first presentation with the studied projects, there were noticed some behaviours regarding the fault repository. Types of faults were reported by the QA engineers as solved, closed, not-replicate, re-opened, etc. Studying the re-opened faults was considered as an indicator of the quality that the previous version of the project had and as an indicator for the re-work in the code part. The priority served as understanding the maturity that a team has when marking faults as major, low or critical based on the consequences that a fault might have on the module. These markings were done based on the team experience and structure. In this way, the time that it is needed to solve the fault or the chances that it will re-appear can be understood. We had to consider also what possibilities the fault reporting software offered. Reporting a fault was done according to the fields that Jira, Bugzilla offered even though the fault reporter might want to include more or less information regarding it. During this process, it was seen that more effort was put in the comments section rather than other fields.

Continuing with the fault labeling, the importance of the status metric stands on the fact that the fixing process for a fault is on pause as it can depend on another unresolved fault. There is another possibility that the discovery of this one has higher chances to make related files also marked as faulty. Especially in the Eclipse project, it was noticed that these factors were associated with notes for the corresponding component, product, operating system, hardware. From the fault tickets, we had to consider the life cycle that a fault undergoes. Some of them were re-opened. Later in the process, we also came across to the dependent (values are dependent on other ones) and independent metrics. While working on Jira the inclusion of the fault severity, dependent variable, became a must due to a better understanding of the fault categorization.

The main question during metric collection process was: "What metrics combination to do?". According to Andersson and Runeson [66], size metrics alone are not good predictors. A study done by Denaro et al. [67] suggests that object oriented metrics and static ones have the same performance in large data sets. We need to combine them with other metrics to see if for small data sets there might be an increase in performance. The file and the module in which the fault resides on is due to the fact that in the majority of cases that part of the file/module is more complex than other files. We got the idea after reading the work of Nachi and Tom [68] which have found a correlation between faults and complexity, but there were not given the exact metrics to be studied on. For the project and process metrics, it was seen that we can pick a certain metric that could be easily found in all of the reported files. In the case of open source project, it was seen a variation in the number of commits and those who did the commit, so it was thought to be included.

There were also considerations regarding the dependency and independence of the metrics selected in order not to violate the requirements of the used algorithms. The selection was done according to the studied sources and also to the meaning that the metrics hold. A better indicator for this case is the Pearson R correlation factor. It shows what value between a range of -1 and +1 the relation resides. If the value is closer to +1 it means that if one metric increases even the other one will increase. The independent variables were those under the static code class and dependent were those of process and project class. There were many reasons for doing this classification, such as seeing the commitment proneness and the findings for a possible dependency between this metric and static code as suggested by Lu et al. [69]. The above reasoning served as a good starting point for proceeding with experiments as: first collect metrics, see their correlation and then apply algorithm.

7.1.1 What is industry looking for ?

It is important to bring the industry and scientific research community on the same path in order to utilize more the fault prediction methods. We came with this section while doing this thesis at the company and the difficulties we came across. More collaborations can be made in this field. Some suggestions given can ease this process:

- **A cross-platform tool:** The tools available in the market do not have this ability. The given tools require a lot of dependencies which are time-consuming for configuring. The commercial tools at a certain point have bypassed this problem with the installation wizard, instead, those from the research community require to be functional only if some programming framework are fully present in the machine.
- **Libraries:** This concerns only the tools coming from the research community. The tool itself might have superior functionality compared to commercial ones, but when it comes to mathematical interpretation, the majority of tools are built from MATLAB or WEKA Application Programming Interface (API) calls, so they are not independent stand-alone tools.
- **Graphical utility:** Only commercial tools such as MATLAB or WEKA offer this feature as inbuilt. For scripts that are built from Python or R programming language, a user needs to have pre-installed some libraries which come with their dependencies. In this way, the user needs to know before hand what to install.
- **API extensibility:** This is closely related to the two previously mentioned points. We take for example WEKA popularity. It offers the API calls to its services. That is a feature not seen on other similar tools or scripts. It means, that if a developer wants to integrate the tool functionality, without API will be difficult. This goes in close with the fact that a tool can be or not be able to be adapted to new future algorithms that might come.
- **Metric understanding:** The majority of researchers do explain metrics, but the meaning is not perceived equally. For example, in the file density tool by Ostrand et al. [14] when developer metric was introduced in both pre and post release phases, the complexity of the code was taken as a point of referring, so it does not necessarily mean the competency of a developer towards the code.
- **Metrics settings:** This option should be present in those tools which mathematical knowledge resides on various combinations that certain values can have. We can bring as an example the case of k number in clustering. Setting different values for k reside in different outputs. Tools which have this option tend to be more popular.
- **Easy to read output:** As the expected results from fault prediction algorithms is expected to be numbers, the format in which those are presented can play a role in the popularity of the tool.
- **How the tool is represented and maintained:** Certain tools have a short user manual or it is easy to contact the support team. Concerning to the maintenance part, some tools have only one version and are not maintained anymore. This means that they were built only for one purpose: testing the algorithm presented in the article or the tool itself did not present interest in the research community.
- **Automation:** The majority of tools require human interaction. They lack this feature. Seeing that the industry wants most of the data in one output, it would be a suggestion that for example time scheduling of metric collection can be done as an option. This would increase the tools in the market.
- **Proper handling of data:** Industry has an enormous quantity of data. The proposed tool should be able to handle the linking of code commits with the reporting of bugs. Plus, only relevant data should be saved in already existing bug tracking systems. There might be inconsistencies between what an already in hand tool can offer to the new fault prediction one.

7.1.2 Why is industry not always fully relying on fault prediction ?

Fault prediction algorithms seems to be a promising topic, yet industrial adoption seems to be far away. A reason for this can be the way how a company works. It might be the case that the nature of continuous delivery, as it follows Agile principles, requires constant internal updates and reviews. The right integration of daily scrum meetings makes possible that the daily routine is focused on a common goal. Having brief and short daily meetings makes the team know what today's problem is or was inherited from yesterday. When the team has a common view of the problems or other topics related to the continuous delivery environment, adapting to a new way of thinking might be difficult [6].

The connection between industry and research community seems to be questioned by the fact that fewer companies are letting researchers use their data sets of faults. By this reasoning, the impact that a data set and its produced numbers would be difficult to be understood in a matter of time by the industrial team. Relying on numbers might be a new concept for the team. The difficulty rises when the team must have a common understanding of what those numbers represent. Each and every metric plays a role in the applied fault prediction algorithm, but it might be the case that the company does not collect certain metrics anymore due to lack of some metric collection software. A reason for that can be the fact of having different understandings on what a metric can be. Taking for example number of submissions. Making too many submissions or even less it does not mean that the submitted code is of good quality or even the complexity of a code. Introducing and implementing prediction concept might be time-consuming in a team where continuity of other works is primary.

Keeping the focus during a short time period it is not that easy. Each tester or developer has his/her own way of working. There is yet not a tool which says that in this exact line there will be this percentage of having a possible fault in a near future. Even if it was true, this can interfere with the way how the team will work. Instead of focusing on delivering a working product, the team will be more focused on how to not have a fault notification, considering the case when the team has full trust on the fault prediction algorithm. Pair programming or further team meetings could be considered as not necessary since the regulator would be the fault prediction tool rather than team members. For this reason, the concept of team collaboration might be questioned. Related to this topic, face-to-face communication between team members seems more reasonable. In this way, the team stays away from misunderstandings that might arise by blindly following the tool [70]. The tools that the research community is offering to the industry still are not able to know the reason of why a certain bug is happening. This can help into the understanding and prediction of future faults of same nature. Something that tools, in general, are missing (besides Prest and Weka) is the possibility to export the result in different formats so it can be further elaborated. In this sense, industry, or precisely the QA or dev team, might have it difficult to manually export.

There might be challenges for the company during an fault prediction algorithm implementation. A certain algorithm which relies on changes that a code might suffer might point out false positive notification. Changes to the code were due to a new functionality being added. The developer, in this case, might postpone the commit and so the deadline is not met. This can be a scenario during the first phases of the project when the data set of that algorithm is not fully trained. Finding the balance between reality on prediction is yet to be further studied. These aspects are not foreseen in any of the studied algorithm articles.

The introduction of certain algorithms into the market might be blocked by the time when a company decides to introduce fault prediction in its working environment. If it is pre-release phase than algorithms requiring pre-release metrics can be applied. That is also influenced by the fact if the company is correctly making use of control version history repositories or fault data sets. Even if the company decides to use a certain algorithm, fault prediction can not be applied in the spot as collecting the metrics might require time. For example, Jira, a fault tracking system, does not have the fault prediction metrics collection ability yet, but its reports show some insights [13].

8 Experimental results and conclusions

In this paper, we aimed a further utilization of the testing data while using them for feeding the fault prediction algorithm. We posed the research questions and explored ways to answer them. Our work was focused on analyzing two different projects where we collected the data and used two fault prediction algorithms Clustering and Naive Bayes. Feeding the algorithm with data was done in a period varying from two weeks to 3 months. Based on the models we proposed in the Models section 5.2, here we will show our best values we got based on the most correlated metrics applied to the prediction algorithms. After selecting positive correlation metrics values, we will show the algorithms values with screen shots taken from the respective tools we used.

By making use of free open source tools, the data collected so far, first were divided into different metrics combinations sets as shown in section 5.1.2 and tested for their correlation. Metrics adjustments and combinations were done accordingly and partially based on the findings of the correlation impact metrics on different models studied by Jiarpakdee et al. [71]. After that, metrics having the best correlation and the best value were tested with Weka and R tools with respective algorithms. Between each and every metric used here, it was seen how they do impact each-other values. The produced forecast was checked if true.

8.1 Experiments related to RQ2

In the beginning, after the criteria presented for answering RQ2 were fulfilled (section 3), manual data collection was done. Some subsets were created, using the software SourceMeter in regards to the correlation, and in different time periods their impact was tested using the respective algorithms via Weka, SPSS, and R. A way of seeing the effectiveness and validity of the prediction with certain metrics is plotting the ROC curves and finding AUC values. We will consider for usage in the prediction algorithm some of these metrics, which are correlated and have a correlation value bigger than 0.5, in our prediction algorithms. Here we can see the values of true and false positive predictions made by the algorithms. Instead, Anova values are a type of testing used among statistical data.

As correlation includes the relationship between two metrics, finding the quantitative relation involves the usage of Pearson's correlation formula. From the obtained results we see that there exists a positive correlation, which means if one value increases the correlated one also increases. For detecting how correlated our collected data were, the time of the study was 3 months. In this period, there were also subsets of experiments which are not shown here as their results did not have any impact and were not positively correlated (section 3). Between each and every metric used here, it was seen how they do impact each-other values. This is closely related to the answer of RQ2 as with performing correlation, redundant metrics do not serve as input when the prediction algorithm is tested. From the experiments, we see values in between -1 and 1 . In some cases, the values are closer to 1 or -1 and that respectively means that the correlation is strongly positive or strongly negative. A good example can be the positive correlation between removed lines of code and added lines of code. The selected metrics were used also because they passed the requirements that a metric needs to be used in Pearson R. The data values were collected in a continuous way and the relationship among metrics is linear. These results served as an indicator to use these values later with the respective algorithms.

		added	removed	files	interest	related	repeated
added	Pearson Correlation	1	,812**	,867**	-,193	-,210	,269
	Sig. (2-tailed)		,000	,000	,365	,326	,205
	N	24	24	24	24	24	24
removed	Pearson Correlation	,812**	1	,981**	-,121	-,084	,136
	Sig. (2-tailed)	,000		,000	,573	,697	,526
	N	24	24	24	24	24	24
files	Pearson Correlation	,867**	,981**	1	-,167	-,117	,174
	Sig. (2-tailed)	,000	,000		,436	,587	,416
	N	24	24	24	24	24	24
interest	Pearson Correlation	-,193	-,121	-,167	1	,240	-,111
	Sig. (2-tailed)	,365	,573	,436		,259	,607
	N	24	24	24	24	24	24
related	Pearson Correlation	-,210	-,084	-,117	,240	1	-,405*
	Sig. (2-tailed)	,326	,697	,587	,259		,050
	N	24	24	24	24	24	24
repeated	Pearson Correlation	,269	,136	,174	-,111	-,405*	1
	Sig. (2-tailed)	,205	,526	,416	,607	,050	
	N	24	24	24	24	24	24

Figure 2: Correlation: process metrics

The first correlation experiment was conducted in order to check from the proposed set of metrics which one fits best for the final set to be used. Based on these results, the aim is to understand and check that there is a relation between changes a file suffers (removed and added code) and the chance of that file to be faulty. The correlation proves that there is a chance of faultiness in files which are mostly changed. The result of most interest here is that out of five selected metrics, three of them represent a strong positive linear relationship and those three might be used in the set to be tested with the prediction algorithm. The conclusion derives from the fact that three combinations were closer to +1, which means a strongly positive relation:

- removed-added
- removed-files
- files-added

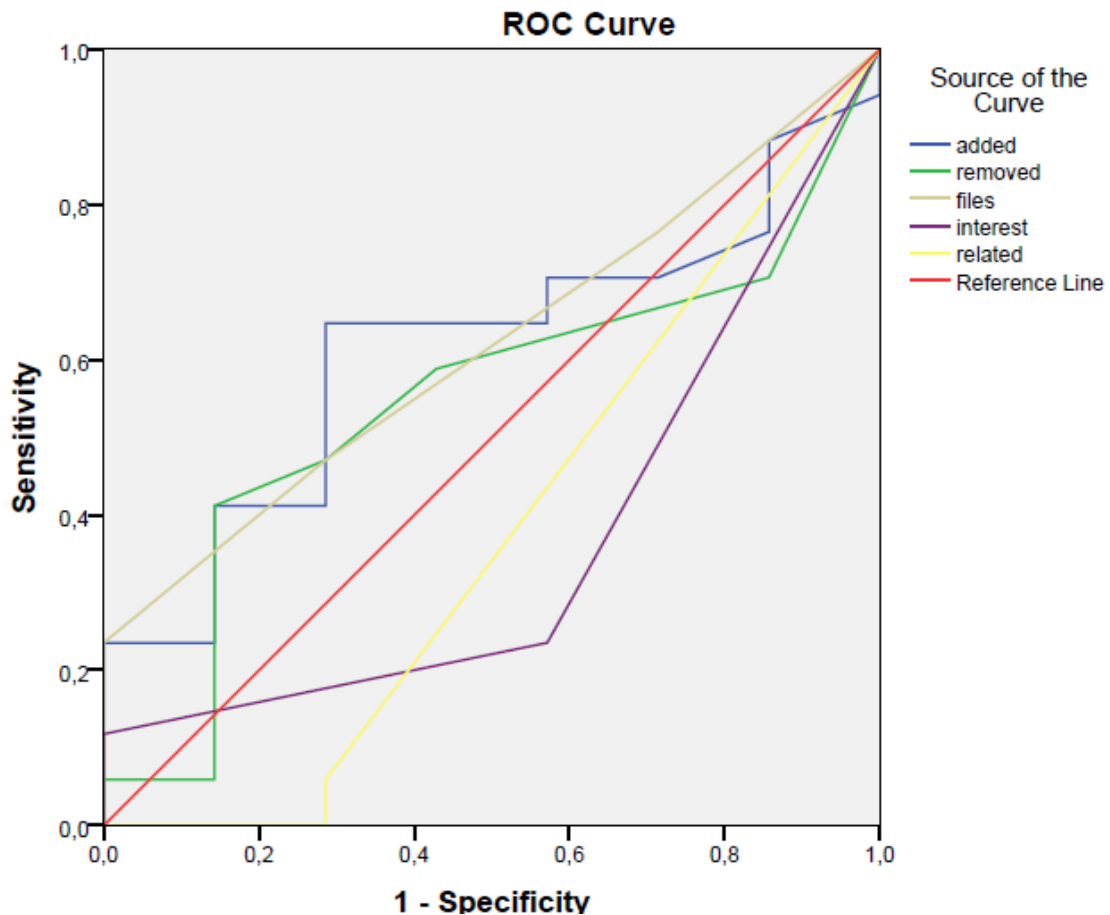


Figure 3: ROC graph of the metrics used in above experiment

In figure 3 is presented a graph from SPSS tool which shows the values from the previous correlation experiment. As we said, The sensitivity axis represents the probability of detection and the 1-specificity axis shows the probability of false detection. Sensitivity or the true positive rate shows how many true predictions can be made depending on the amount of data. The sensitivity value should be higher as it means less misses in prediction.

Considering the point (0.4;0.5) from removed and files metrics, that combination has the tendency to guess nearly half of the predictions as positive and half negative. The added, removed, files metrics lines go up and down within the middle region which makes that metrics performance to be considered as nearly random. In this graph, there are some other interesting points such as (0.3;0.5) where the removed, files, added, number of commits metrics meet, (0.8;0.8) files and added metrics meet. So, it is seen that those metrics appear in different combinations and have a high rate of identifying positives. As we explained previously, in the ROC curve, the left uppermost part of the graph is important. Only files metrics seem to stay in that region, other metrics tend to slightly be on the other side for some time.

Area Under the Curve

Test Result Variable(s)	Area
added	,613
removed	,546
files	,618
interest	,366
related	,378

Figure 4: AUC of the previous correlation experiment

The values indicate that this combination has 3 values higher than 0.5, which means nearly a good classifier. Area under curve in the figure 4 is the interpretation of the fig.3 graph ROC. AUC was done in order to have a better picture of the previous results and see in numbers how good a prediction can be, given the metrics which have good correlation. The conclusion is that given the metrics, the prediction will be slightly above average 0.5, as also the SSE (shown in Anova) has high values, but still, those were the best metrics we can give to the algorithm as input.

		Sum of Squares	df	Mean Square	F	Sig.
added	Between Groups	146186,096	1	146186,096	1,709	,205
	Within Groups	1881489,529	22	85522,251		
	Total	2027675,625	23			
removed	Between Groups	19386,723	1	19386,723	,415	,526
	Within Groups	1028782,235	22	46762,829		
	Total	1048168,958	23			
files	Between Groups	153,447	1	153,447	,687	,416
	Within Groups	4910,387	22	223,199		
	Total	5063,833	23			
interest	Between Groups	,359	1	,359	,272	,607
	Within Groups	28,975	22	1,317		
	Total	29,333	23			
related	Between Groups	1,303	1	1,303	4,307	,050
	Within Groups	6,655	22	,303		
	Total	7,958	23			

Figure 5: Anova of previous experiment

The result shows high SSE values. With Anova experiment it is proven that the mean deviance is big. This proves that it is not a so good predictor. The F value is of more interest. It shows how that specific metric is distributed from the mean of the group it resides in. Still, the F low values do not make it right due to very high SSE values. Only files, related and interest metrics seem to have good SSE values. The file metric, as concluded in fig.4 AUC correlation experiment, will be later included in the fault prediction experiments.

		loc	years	modified	issues	contributors	commits
loc	Pearson Correlation	1	-.516**	.944**	.801**	.539**	.821**
	Sig. (2-tailed)		.000	.000	.000	.000	.000
	N	54	54	54	54	54	54
years	Pearson Correlation	-.516**	1	-.547**	-.454**	-.622**	-.604**
	Sig. (2-tailed)	.000		.000	.001	.000	.000
	N	54	54	54	54	54	54
modified	Pearson Correlation	.944**	-.547**	1	.817**	.655**	.896**
	Sig. (2-tailed)	.000	.000		.000	.000	.000
	N	54	54	54	54	54	54
issues	Pearson Correlation	.801**	-.454**	.817**	1	.437**	.737**
	Sig. (2-tailed)	.000	.001	.000		.001	.000
	N	54	54	54	54	54	54
contributors	Pearson Correlation	.539**	-.622**	.655**	.437**	1	.804**
	Sig. (2-tailed)	.000	.000	.000	.001		.000
	N	54	54	54	54	54	54
commits	Pearson Correlation	.821**	-.604**	.896**	.737**	.804**	1
	Sig. (2-tailed)	.000	.000	.000	.000	.000	
	N	54	54	54	54	54	54

** . Correlation is significant at the 0.01 level (2-tailed).

Figure 6: Correlation: static+project metrics

The second correlation experiment served to test the fact that different type of metrics can give better correlation values. The aim here is to find a good combination of those metrics. The time needed to collect such metrics is bigger than the previous experiment. As the time factor increases, it can be observed that certain metrics of different nature have an impact. In the fault repository, it was observed that if the loc metric is bigger so is the reported number of faults per commit. Here is presented only the best combination taken out of different proposed set of metrics. This experiment differs from the previous one due to the fact that here we test more metrics than in the first one. By looking at the result is noticed that different combined metrics can give good values of correlation. The majority of correlation values is very close to +1, making this set a good candidate to be later used in the prediction. This was in compliance with trying to find the best combination of static and project metrics. Here 5 out of 6 metrics show a strong positive linear relationship. The obtained results show that the above correlation results are close to +1 which means that they are good metrics to be taken into consideration. The interest here is that these combinations will be considered to be used with the prediction algorithms:

- loc-modified-issues(faults)-commits
- modified-issues(faults)-commits
- contributions-commits

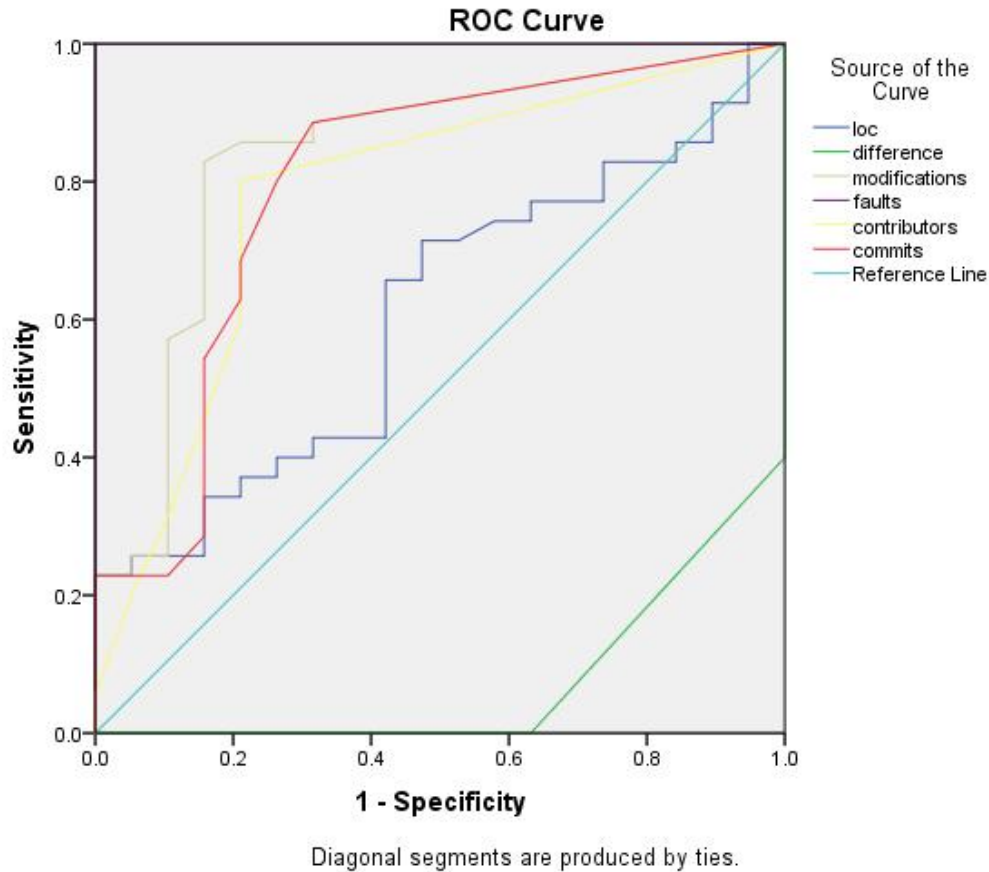


Figure 7: ROC graph of the metrics used in the above experiment

This figure shows the values of second correlation experiment. The sensitivity axis represents the probability of detection and the 1-specificity axis shows the probability of false detection. Sensitivity or the true positive rate shows how many true predictions can be made depending on the amount of data. The sensitivity value should be higher as it means fewer misses in prediction.

Considering the point (0.4;0.4) from LOC, it has the tendency to guess nearly half of the predictions as positive and half negative. Given the LOC information in what class the prediction will be, faulty or not. The LOC lines go up and down within the middle region which makes that metrics performance to be considered as nearly random. In this graph, there are some other interesting points such as (0.1;0.2) where the number of code contributors (effort), number of commits and LOC meet, (0.1;0.3) where contributors and number of modifications meet, (0.3;0.95) where commits and modifications meet. So, it is seen that those metrics appear in different combinations and have a high rate of identifying positives. Only the metric, which holds information about differences in the number of time starting from when the fault was reported till when it is solved, seems not to be of importance. As we explained previously, in the ROC curve, the left uppermost part of the graph is important. The nearly perfect point taken from this graph is that of (0.3;0.95) where commits and modifications meet has a high rate of true positives regarding the classification.

Area Under the Curve

Test Result Variable(s)	Area
loc	.607
difference	.074
modifications	.836
faults	1.000
contributors	.779
commits	.797

Figure 8: AUC of the previous correlation experiment

The values indicate that this combination has a value higher than 0.5, which means nearly a good classifier. In the figure 8 are given areas under ROC curves of the Figure 7. AUC was done in order to have a better picture of the previous results and see in numbers how good a prediction can be, given the metrics which have good correlation. The conclusion is that given the metrics, the prediction will be above average 0.5, but still, those were the best metrics we can give to the algorithm as input.

ANOVA

		Sum of Squares	df	Mean Square	F	Sig.
loc	Between Groups	2053953.912	18	114108.551	56.198	.000
	Within Groups	71066.921	35	2030.483		
	Total	2125020.833	53			
difference	Between Groups	549.208	18	30.512	4.968	.000
	Within Groups	214.940	35	6.141		
	Total	764.148	53			
modifications	Between Groups	135782.516	18	7543.473	51.664	.000
	Within Groups	5110.318	35	146.009		
	Total	140892.833	53			
contributors	Between Groups	84.983	18	4.721	3.888	.000
	Within Groups	42.498	35	1.214		
	Total	127.481	53			
commits	Between Groups	3912.170	18	217.343	19.880	.000
	Within Groups	382.645	35	10.933		
	Total	4294.815	53			

Figure 9: Anova representation of previous values

The result shows high SSE values. With Anova experiment it is proven that the mean deviance is big. This shows that it might not be a so good predictor, regardless of good AUC values above 0.5. The F value is of more interest. It shows how that specific metric is distributed from the mean of the group it resides in. These metrics deserve more interest while included in the final set of prediction algorithm as the metrics commits and contributors are not so common in other studied literature experiments of this nature and they show very good AUC values and less SSE compared to other metrics.

Correlations

		Classes	Effort	Faults	Statements	LOC	Files	Funciones
Classes	Pearson Correlation	1	-.280	-.266	.376	.451 [*]	^b	.513 ^{**}
	Sig. (2-tailed)		.176	.199	.064	.024	.	.009
	N	25	25	25	25	25	25	25
Effort	Pearson Correlation	-.280	1	.998 ^{**}	-.325	-.316	^b	-.315
	Sig. (2-tailed)	.176		.000	.112	.123	.	.125
	N	25	25	25	25	25	25	25
Faults	Pearson Correlation	-.266	.998 ^{**}	1	-.341	-.331	^b	-.327
	Sig. (2-tailed)	.199	.000		.096	.106	.	.110
	N	25	25	25	25	25	25	25
Statements	Pearson Correlation	.376	-.325	-.341	1	.974 ^{**}	^b	.703 ^{**}
	Sig. (2-tailed)	.064	.112	.096		.000	.	.000
	N	25	25	25	25	25	25	25
LOC	Pearson Correlation	.451 [*]	-.316	-.331	.974 ^{**}	1	^b	.789 ^{**}
	Sig. (2-tailed)	.024	.123	.106	.000		.	.000
	N	25	25	25	25	25	25	25
Files	Pearson Correlation	^b	^b	^b	^b	^b	^b	^b
	Sig. (2-tailed)
	N	25	25	25	25	25	25	25
Funciones	Pearson Correlation	.513 ^{**}	-.315	-.327	.703 ^{**}	.789 ^{**}	^b	1
	Sig. (2-tailed)	.009	.125	.110	.000	.000	.	
	N	25	25	25	25	25	25	25

Figure 10: Correlation test: static+process metrics

Continuing on finding the best set of metrics to test, figure 10 has the correlation metrics between static and process combination. The values seem to be less good compared to the combined static and project. Those metrics came as a result that for both types of projects this information was available and especially at the industrial project, in Jira the QA team in the comments mentions these metrics as having an influence for the fault reappearance. Combinations having a correlation higher than 0.5 are 5 out of 7. These combinations will be considered for the prediction algorithm:

- Faults-effort
- Statements-loc
- Statements-function
- Function-loc

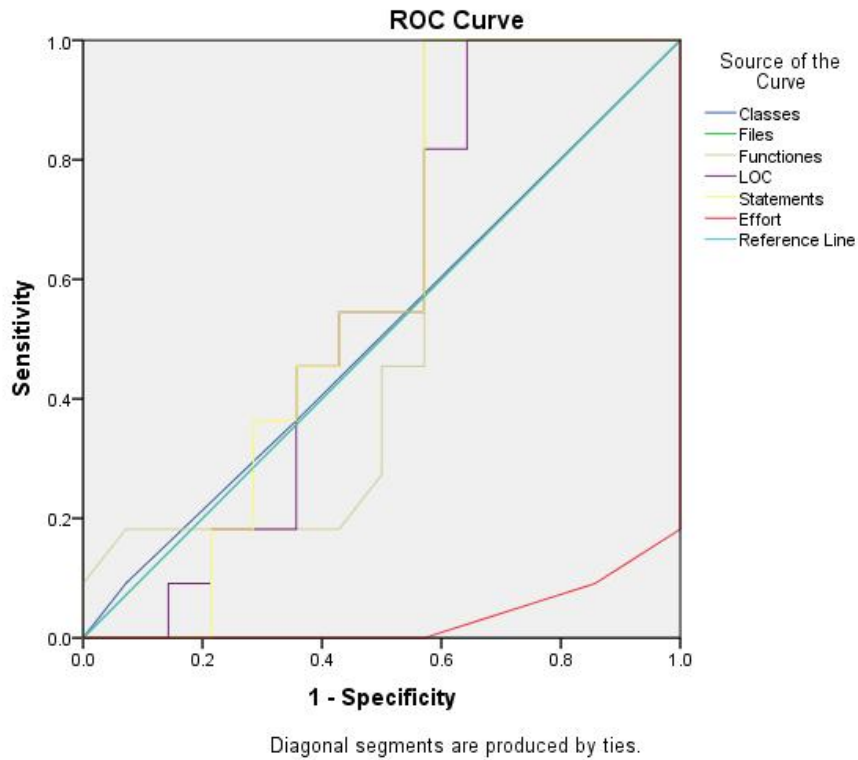


Figure 11: ROC graph of the above experiment

As in the previous correlation we got certain metrics to consider, the values seem to have common points to the left side, so it is proven that we have a fair combination of metrics. It is seen that the points of interest are (0.4;0.4) where statements, functions, files, and LOC meet, (0.6;0.8) where statements, LOC and functions meet, (0.5;1.0) of statements and (0.6;1.0) of LOC. The line of files is in between the reference line, but still to be considered.

Test Result Variable(s)	Area
Classes	.506
Files	.500
Functiones	.552
LOC	.558
Statements	.578
Effort	.032

Figure 12: AUC representation of previous results

Here we see in details that this combination tends to be better as the area values are slightly higher than 0.5. This combination of metrics will be considered as in 10 loc was a good metric and gave also good combinations. So, when loc metric is used with other metrics, the AUC tends to have slightly good values.

8.2 Experiments related to RQ1

Here, we used the best correlation metrics to the prediction algorithms per each project.

```

Number of iterations: 6
Within cluster sum of squared errors: 37.845930478325236
Initial starting points (random):
Cluster 0: 10,10,3,43,Minor,30,Implementation,6
Cluster 1: 45,89,3,230,Minor,10,Fragments,1
Cluster 2: 140,140,4,829,Major,45,Fragments,4
Cluster 3: 6,18,2,109,Minor,10,Adapter,2
Missing values globally replaced with mean/mode
Final cluster centroids:

```

Attribute	Cluster#				
	Full Data (49.0)	0 (20.0)	1 (11.0)	2 (10.0)	3 (8.0)
Com.Class	37.551	27.15	40.7273	78.7	7.75
Com.File	56.7959	40.8	67.6364	114	10.375
Com.Function	3.1224	2.95	4.2727	3.4	1.625
LOC	249.6531	141.35	296.1818	556.9	72.375
Type	Major	Major	Minor	Major	Minor
Effort	38.2857	35	27.1818	75.5	15.25
Module	Fragments	Implementation	Fragments	Fragments	Util
NrOfFaults	3.551	2.8	3.7273	6.2	1.875

```

=====
Time taken to build model (full training data) : 0 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      20 ( 41%)
1      11 ( 22%)
2      10 ( 20%)
3       8 ( 16%)

```

Figure 13: Accedo project - Clustering k=4

Accedo project clustering prediction here uses metrics based on the correlated metrics from model 3. From the cluster instances, it is seen that out of 49 instances, 20 reside on cluster 0 (under module implementation). As this module resides on that cluster 0, it means that the prediction properties will be with the same as those of the cluster. As per clustering algorithm property, the question: what properties will the new fault have, it was predicted that cluster 0 (where faulty predicted module resides) will likely have a fault of major type, have 3 functions per class, 141 lines of code per class, etc. By that prediction values, the mean values are shown here. This due to steps of the algorithms which normalize values.

```

Correctly Classified Instances      527          56.973 %
Incorrectly Classified Instances    398          43.027 %
Kappa statistic                    0.2125
Mean absolute error                0.1082
Root mean squared error            0.2348
Relative absolute error            92.1429 %
Root relative squared error        97.1458 %
Total Number of Instances          925
Ignored Class Unknown Instances    75

=== Detailed Accuracy By Class ===

```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.259	0.089	0.240	0.259	0.294	0.191	0.634	0.222	a
	0.089	0.047	0.280	0.089	0.135	0.069	0.580	0.200	b
	0.921	0.595	0.654	0.921	0.765	0.389	0.626	0.568	c
	0.000	0.000	0.000	0.000	0.000	0.000	0.556	0.013	d
	0.000	0.000	0.000	0.000	0.000	0.000	0.508	0.012	e
	0.000	0.000	0.000	0.000	0.000	0.000	0.512	0.049	f
	0.000	0.000	0.000	0.000	0.000	0.000	0.564	0.016	g
	0.273	0.049	0.170	0.273	0.209	0.178	0.855	0.130	h
	0.000	0.000	0.000	0.000	0.000	0.000	0.805	0.025	i
	0.000	0.000	0.000	0.000	0.000	0.000	0.871	0.028	j
	0.000	0.000	0.000	0.000	0.000	0.000	0.610	0.003	k
Weighted Avg.	0.570	0.350	0.464	0.570	0.495	0.260	0.621	0.387	

```

=== Confusion Matrix ===

```

	a	b	c	d	e	f	g	h	i	j	k	<-- classified as
a	36	15	78	0	0	0	0	10	0	0	0	a
b	23	14	106	0	0	0	0	15	0	0	0	b
c	19	10	468	0	0	0	0	11	0	0	0	c
d	2	2	1	0	0	0	0	0	0	0	0	d
e	0	1	10	0	0	0	0	0	0	0	0	e
f	6	0	24	0	0	0	0	7	0	0	0	f
g	4	0	7	0	0	0	0	1	0	0	0	g
h	12	7	5	0	0	0	0	9	0	0	0	h
i	4	1	3	0	0	0	0	0	0	0	0	i
j	0	0	3	0	0	0	0	0	0	0	0	j
k	0	0	1	0	0	0	0	0	0	0	0	k

Figure 14: Accedo project - Naive Bayes

The metrics are the same as in the clustering experiment. This experiment tested if the Naive algorithm would predict the same module or not as in the clustering experiment, given more data in a broader time period, as in an industrial project releases happen more often. The prediction made was that the module c(implementation) will be faulty as the ROC and precision values are better compared with others' modules values and the precision value is closer to +1. As such, we proved that both algorithms predicted the same module, but in Naive with a higher rate of precision (related to the fact that the value of correctly classified instances is higher) which tells about the ratio of how well faults are predicted and that there were no misses.

The trustworthiness of the predictions is given by the number of correctly classified instances which is 57%. Plus, in the confusion matrix, the classification done gives module c as a prediction, as most of the instances point at it, plus from the mean absolute error, which states how much the estimates differ from actual values has low value. It has the best value of TP rate, what fraction of those that are actually positive were predicted positive. Also, the TP (true positive value), recalls are again closer to 1, meaning the prediction made by the algorithm with the given data set resulted true. It is seen that more than half of the instances are correctly classified.

```

Number of iterations: 4
Within cluster sum of squared errors: 4.2626480151064365

Initial starting points (random):

Cluster 0: 1,1,31,225,106,2,20
Cluster 1: 2,1,14,150,72,2,30
Cluster 2: 1,1,0,23,0,38,560

Missing values globally replaced with mean/mode

Final cluster centroids:
Attribute      Full Data      Cluster#
              (25.0)        0           1           2
=====
Classes        1.44           1           2.2222       1
Files          1              1           1            1
Functions      14.36         11.6667     24.3333      0
LOC            153.44        117         263.5556     15
Statements     66.56         52.9167     114.3333     0
Nr Faults      5.32          2.3333      2.1111       21.5
Effort         75            30.8333     25           320
Module         platform      platform     cdt          jdt

Time taken to build model (full training data) : 0.01 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      12 ( 48%)
1       9 ( 36%)
2       4 ( 16%)

```

Figure 15: Eclipse - Clustering k=4

With this experiment, it was seen how the algorithm would behave with open source data and what would be the prediction. The metrics here, as in the previous clustering experiment, serve as an indicator of the file's property which is going to be faulty. So, the numbers in the predicted cluster show that the faulty module will be platform and will have a mean of 11 functions, 2 faults. Considering the metrics used, here we see that there will be higher chances of fault when the number of loc, statements, and effort spent is high in platform module.

```

Correctly Classified Instances: 15          65  †
Incorrectly Classified Instances: 10        35  †
Kappa statistic                0.362
Mean absolute error            0.1145
Root mean squared error        0.1823
Relative absolute error        42  †
Root relative squared error     35  †
Total Number of Instances      23
Ignored Class Unknown Instances 2

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
      0.667    0.100    0.500     0.667   0.571     0.503    0.892    0.600    cdt
      0.000    0.000    0.000     0.000   0.000     0.000    0.548    0.121    swtbot
      0.375    0.000    1.000     0.375   0.545     0.530    0.854    0.780    jdt
      0.857    0.188    0.667     0.857   0.750     0.631    0.911    0.781    platform
      1.000    0.000    1.000     1.000   1.000     1.000    1.000    1.000    pdt
      1.000    0.045    0.500     1.000   0.667     0.691    0.977    0.500    equinox
      1.000    0.136    0.250     1.000   0.400     0.465    0.932    0.250    orion
Weighted Avg.  0.609    0.078    0.692     0.609   0.582     0.536    0.865    0.674

=== Confusion Matrix ===

 a b c d e f g  <-- classified as
2 0 0 1 0 0 0 | a = cdt
0 0 0 0 0 1 1 | b = swtbot
1 0 3 2 0 0 2 | c = jdt
1 0 0 6 0 0 0 | d = platform
0 0 0 0 1 0 0 | e = pdt
0 0 0 0 0 1 0 | f = equinox
0 0 0 0 0 0 1 | g = orion

```

Figure 16: Eclipse project test with Naive Bayes

With this experiment, we wanted to prove the previous clustering prediction. Here the predicted platform's files actually have the same file properties as previously predicted. In previous cluster experiment, the predicted cluster resides on the Naive's predicted module. In this experiment, we have a 65% correctly classification of instances. If we compare this values with the real values, we see that platform module has the most number of faults reported there for the studied period of time (February-March) as in fig.17.

In the confusion matrix, platform module has all of its 6 files predicted correctly. Other faults were predicted to reside on other modules compared to what module they were really faulty. We can see our prediction to be in coherence with the finding of [19] which states that the nearly perfect F-measure should be max 0.75, as above false predictions might come. Other modules might have values as 1, but in the confusion matrix, their predictions were not so trusting worthy, as previously explained. Considering pdt module which is only one time reported as faulty its values cannot be considered as valid, as other modules have more faults reports.

9 Answers to research questions

(a) Which selected algorithm gives better predictions?

- This question was answered by performing the experiments in fig.13 till fig.16 in section 13. There the tests predicted the same module for both of the algorithm. Due to its properties, Clustering gives more of a general view regarding the predicted faulty module properties. As it can be seen in fig 17 the faulty module had 8 new faults. This number was predicted more in precise by the Naive algorithm, as 6 in the confusion matrix. As such, Clustering results can still be trusted but for their precision, rather than as a guide towards the faulty module and can be used as a summarizing of the fault properties. It does not give any detail for its prediction precision, only in what common properties, the faults will reside on. On the other hand, Naive predicts the same modules as faulty like Clustering did, but it gives more precise details on its prediction such as F measures, recall, and others. This helped into understanding the prediction and discarding any false prediction. This option is not available in Clustering. Naive trustworthiness can be seen by its properties, which for our tests were: give prediction error details even with few data, estimate on labeled previous data and more closer to reality predicted number of modules.
- Naive and Clustering algorithm for both Eclipse and Accedo's project showed in reality that the predicted module indeed in the days after the test was performed had faults. The predicted results, in reality, resulted as 3/4 real, for example as in fig.17 platform had 8 new faults, we predicted 6. As it is said by Scaniello et al. [37], the recall ratio was not reported to pass that number in any of the studied papers by them. After we did the correlation experiments and created the data subsets with the best values metrics, fault prediction algorithms were applied. Below are shown the best values taken from it, from Naive (Recall) and Clustering algorithm (ROC). With a recall of between 0.92 and 0.85, as shown in the tables below 6 and 7, our model might have the tendency to miss some faults, but still, it is within the range of a fair good model. While testing the algorithms with metrics combinations, the precision seems to have good values and the prediction resulted as true.

Table 6: Accedo Project Best values

Prediction	Recall	ROC
Module c	0.921	0.626

Table 7: Eclipse Project Best values

Prediction	Recall	ROC
Platform	0.857	0.911

		Status					
		NEW	ASSIGNED	RESOLVED	VERIFIED	CLOSED	Total
Product	Platform	<u>8</u>	<u>1</u>	<u>7</u>	<u>1</u>	<u>5</u>	<u>22</u>
	Total	<u>8</u>	<u>1</u>	<u>7</u>	<u>1</u>	<u>5</u>	<u>22</u>

Figure 17: Total new faults for Feb-March - Eclipse Platform module

(b) Which are the most correlated metrics?

- The results obtained from the experiments 1 till 12 in section 8.1 vary from combination to combination, but static metrics combined with process metrics tend to give better results in terms of correlation and can be tagged as nearly good metrics. As AUC was used as a numerical evaluator for ROC curve, which is the surface under the ROC curve, it was seen that there were 13 out of 15 studied metrics combinations such as: added, removed, files, loc, modifications, number of faults, contributors, commits, number of faults, number of classes, number of faulty files, number of functions and number of statements which reside in values >0.5 and <1 . These metrics, which come model 2 and 3 in 5.2 were closer to value 1 which makes them good metrics to be used in the prediction algorithms.

In the ROC graphs, it was noticed that when the sensitivity increases the specificity decreases and that is normal, but the shifting is not done smoothly. But when used in the fault prediction algorithms, they gave good predictive values, regardless of the project nature, open-source or industrial project. We can see that these values gave good SSE values, in case of clustering experiments, such as fig.13 and fig.15, and more than 57% and 65% of the instances were correctly classified in the case of Naive, fig.14 and fig.16, which is above medium. To be pointed out here, it is the fact that the metrics of contributors and commits, according to our studied literature, seems an unexplored field. Yet, in our study, we tried to include those in the correlation metrics tests.

10 Future research and suggestions

Based on our findings and discussion section, here some suggestions are given for future research on the field. Our suggestions are related to certain aspects of fault prediction, which if extended can help to bypass conceptual problems that might arise. In a later future, new tools that deal with fault reporting or existing ones can implement such prediction model depending from the software nature.

10.0.1 Research Community

The research community should be focused towards models that implement new algorithms. A good help for this can be a new algorithm based on the continuous delivery environment. In this way, the impact of CD metrics can better understand based on the working culture. It has been 40 years since the first publication on fault prediction, nearly 17 since agile manifesto was published and yet CD has been left aside regarding fault prediction aspect. The conduction of more studies on this environment is encouraged since a good algorithm needs time to be published and tested. Maybe the introduction of new metrics and automated tools for building data sets might help or more cooperation with industry since till now only open source projects have been studied more.

10.0.2 Company suggestion

This type of study can be further extended in an industrial context. New metrics and different projects can be added. The building of a fault repository could help in this process. Software companies can extend their collaborations with the research community, by starting so of a real collaboration for understanding better industrial needs in a research context. If these suggestions go well, it can be made possible the development of an automated fault prediction tool and algorithm which:

- has metrics collection based on project properties.
- has API, adjust metrics, export results option.
- reads meta data from the code repository.
- show notifications in real time.

References

- [1] T. Muthusamy and K. Seetharaman, “A test case prioritization method with weight factors in regression testing based on measurement metrics,” *International Journal of Advanced Research in Computer Science and Software Engineering Volume*, vol. 3, 2013.
- [2] F. Akiyama, “An example of software system debugging,” in *IFIP Congress (1)*, 1971, pp. 353–359.
- [3] S. S. Rathore and S. Kumar, “A study on software fault prediction techniques,” *Artificial Intelligence Review*, May 2017.
- [4] C. Catal, “Software fault prediction: A literature review and current trends,” *Expert Systems with Applications*, vol. 38, no. 4, pp. 4626–4636, Apr. 2011.
- [5] T. Illes-Seifert and B. Paech, “Exploring the relationship of history characteristics and defect count: an empirical study,” in *Proceedings of the 2008 workshop on Defects in large software systems*, 2008, pp. 11–15.
- [6] J. Derehag, E. Weyuker, T. J. Ostrand, and D. Sundmark, “Transitioning Fault Prediction Models to a New Environment,” in *12th European Dependable Computing Conference (EDCC)*. IEEE, Sep. 2016, pp. 241–248.
- [7] B. P. Lientz, “Issues in software maintenance,” *ACM Comput. Surv.*, vol. 15, no. 3, pp. 271–278, Sep. 1983.
- [8] A. K. Pandey and N. K. Goyal, *Early Software Reliability Prediction*, ser. Studies in Fuzziness and Soft Computing. India: Springer India, 2013, vol. 303.
- [9] C. Catal and B. Diri, “A fault prediction model with limited fault data to improve test process,” in *International Conference on Product Focused Software Process Improvement*. Springer, 2008, pp. 244–257.
- [10] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, “A systematic literature review on fault prediction performance in software engineering,” *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, 2012.
- [11] E. Rashid, S. Patnaik, and V. Bhattacharjee, “Search-based information retrieval and fault prediction with distance functions,” *International Journal of Software Engineering and its Applications*, 2014.
- [12] R. E. Stake, *The art of case study research*. Sage, 1995.
- [13] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, “Predicting the location and number of faults in large software systems,” *IEEE Transactions on Software Engineering*, vol. 31, no. 4, pp. 340–355, 2005.
- [14] T. J. Ostrand and E. J. Weyuker, “Predicting Bugs in Large Industrial Software Systems,” in *Software Engineering*. Springer, 2013, pp. 71–93.
- [15] A. E. Hassan, “Predicting faults using the complexity of code changes,” in *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, 2009, pp. 78–88.
- [16] T. M. Khoshgoftaar and N. Seliya, “Fault prediction modeling for software quality estimation: Comparing commonly used techniques,” *Empirical Software Engineering*, vol. 8, no. 3, pp. 255–283, 2003.
- [17] Y. Jiang, B. Cukic, and T. Menzies, “Fault prediction using early lifecycle data,” in *The 18th IEEE International Symposium on Software Reliability (ISSRE’07)*. IEEE, 2007, pp. 237–246.
- [18] Z. Li and M. Reformat, “A practical method for the software fault-prediction,” in *2007 IEEE International Conference on Information Reuse and Integration*, Aug 2007, pp. 659–666.
- [19] C. Catal, “Performance evaluation metrics for software fault prediction studies,” *Acta Polytechnica Hungarica*, vol. 9, no. 4, pp. 193–206, 2012.

- [20] R. M. Bell, T. J. Ostrand, and E. J. Weyuker, “Does measuring code change improve fault prediction?” in *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*. ACM, 2011, p. 2.
- [21] M. Singh and D. S. Salaria, “Approaches for software fault prediction,” *International Journal of Computer Science and Technology (IJCST)*, vol. 3, no. 4, pp. 419–421, 2012.
- [22] D. Kaur, A. Kaur, S. Gulati, and M. Aggarwal, “A clustering algorithm for software fault prediction,” in *Computer and Communication Technology (ICCCCT), 2010 International Conference on*. IEEE, 2010, pp. 603–607.
- [23] A. Kaur and I. Kaur, “An empirical evaluation of classification algorithms for fault prediction in open source projects,” *Journal of King Saud University - Computer and Information Sciences*, Apr. 2016.
- [24] W. Liu, S. Liu, Q. Gu, J. Chen, X. Chen, and D. Chen, “Empirical studies of a two-stage data preprocessing approach for software fault prediction,” *IEEE Transactions on Reliability*, vol. 65, no. 1, pp. 38–53, 2016.
- [25] S. S. Rathore and S. Kumar, “An empirical study of some software fault prediction techniques for the number of faults prediction,” *Soft Computing*, pp. 1–18, 2016.
- [26] S. Tahvili, W. Afzal, M. Saadatmand, M. Bohlin, D. Sundmark, and S. Larsson, *Towards Earlier Fault Detection by Value-Driven Prioritization of Test Cases Using Fuzzy TOPSIS*. Springer International Publishing, 2016, pp. 745–759.
- [27] R. Mahajan, S. K. Gupta, and R. K. Bedi, “Design of Software Fault Prediction Model Using BR Technique,” *Procedia Computer Science*, vol. 46, pp. 849–858, 2015.
- [28] R. Malhotra, “A systematic review of machine learning techniques for software fault prediction,” *Applied Soft Computing*, vol. 27, pp. 504 – 518, 2015.
- [29] G. Abaei and A. Selamat, “A survey on software fault detection based on different prediction approaches,” *Vietnam Journal of Computer Science*, vol. 1, no. 2, pp. 79–95, May 2014.
- [30] M. White, C. Vendome, M. Linares-Vasquez, and D. Poshyvanyk, “Toward deep learning software repositories,” in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, May 2015, pp. 334–345.
- [31] E. Erturk and E. A. Sezer, “A comparison of some soft computing methods for software fault prediction,” *Expert Systems with Applications*, vol. 42, no. 4, pp. 1872 – 1879, 2015.
- [32] G. Aupy, Y. Robert, F. Vivien, and D. Zaidouni, “Checkpointing algorithms and fault prediction,” *Journal of Parallel and Distributed Computing*, vol. 74, no. 2, pp. 2048–2064, Feb. 2014.
- [33] A. Adline and M. Ramachandran, “Predicting the software fault using the method of genetic algorithm,” *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering (IJAREEIE)*, vol. 3, no. special issue 2, pp. 390–398, 2014.
- [34] A. S. D. Shyna Kakkar, “Software fault prediction using hybrid k-means-feed forward neural network,” *International Journal of Computer Science and Electronics Engineering (IJCSEE)*, vol. 5, pp. 1–6, 2015.
- [35] D. Bowes, T. Hall, M. Harman, Y. Jia, F. Sarro, and F. Wu, “Mutation-aware fault prediction,” in *Proceedings of the 25th International Symposium on Software Testing and Analysis*. ACM, 2016, pp. 330–341.
- [36] T. Diamantopoulos and A. Symeonidis, “Towards interpretable defect-prone component analysis using genetic fuzzy systems,” in *Proceedings of the Fourth International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, ser. RAISE ’15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 32–38.
- [37] G. Scanniello, C. Gravino, A. Marcus, and T. Menzies, “Class level fault prediction using software clustering,” in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*. IEEE, 2013, pp. 640–645.

- [38] N. Seliya and T. M. Khoshgoftaar, "Software quality analysis of unlabeled program modules with semisupervised clustering," *Trans. Sys. Man Cyber. Part A*, vol. 37, no. 2, pp. 201–211, Mar. 2007.
- [39] J. Chen, S. Liu, W. Liu, X. Chen, Q. Gu, and D. Chen, "A Two-Stage Data Preprocessing Approach for Software Fault Prediction," in *Eighth International Conference on Software Security and Reliability (SERE)*. IEEE, Jun. 2014, pp. 20–29.
- [40] Y. Suresh, "Software quality assurance for object-oriented systems using meta-heuristic search techniques," in *Applied and Theoretical Computing and Communication Technology (iCATccT), 2015 International Conference on*. IEEE, 2015, pp. 441–448.
- [41] Q. Yin, R. Luo, and P. Guo, "Software Fault Prediction Framework Based on aiNet Algorithm," in *Seventh International Conference on Computational Intelligence and Security*. IEEE, Dec. 2011, pp. 329–333.
- [42] Q. Liu, F. Zhang, M. Liu, and W. Shen, "A fault prediction method based on modified Genetic Algorithm using BP neural network algorithm," in *Systems, Man, and Cybernetics (SMC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 004614–004619.
- [43] A. Okutan and O. Taner Yildiz, "A novel kernel to predict software defectiveness," *Journal of Systems and Software*, vol. 119, pp. 109–121, Sep. 2016.
- [44] S. Chatterjee and A. Roy, "Novel Algorithms for Web Software Fault Prediction," *Quality and Reliability Engineering International*, vol. 31, no. 8, pp. 1517–1535, Dec. 2015.
- [45] G. Abaei and A. Selamat, "Increasing the Accuracy of Software Fault Prediction Using Majority Ranking Fuzzy Clustering," in *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, R. Lee, Ed. Cham: Springer International Publishing, 2015, vol. 569, pp. 179–193.
- [46] C. Catal and B. Diri, "A fault detection strategy for software projects," *Tehniki vjesnik*, vol. 20, no. 1, pp. 1–7, 2013.
- [47] M. H. Halstead, *Elements of Software Science (Operating and Programming Systems Series)*. New York, NY, USA: Elsevier Science Inc., 1977.
- [48] N. Nagappan and T. Ball, "Static analysis tools as early indicators of pre-release defect density," in *Proceedings of the 27th international conference on Software engineering*. ACM, 2005, pp. 580–586.
- [49] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on software engineering*, vol. 20, no. 6, pp. 476–493, 1994.
- [50] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Transactions on software engineering*, vol. 22, no. 10, pp. 751–761, 1996.
- [51] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE transactions on software engineering*, vol. 33, no. 1, 2007.
- [52] H. Futong and S. Tingting, "Software Project Metrics and Quality Management," in *Ninth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*. IEEE, Oct. 2013, pp. 615–618.
- [53] R. A. Harpster, "How FMEAs can be the cornerstone of ISO 2001: 2015 compliant risk based quality management system," in *Reliability and Maintainability Symposium (RAMS), 2016 Annual*. IEEE, 2016, pp. 1–5.
- [54] B. Stanic, "Static code metrics vs. process metrics for software fault prediction using bayesiannetwork learners," Master's thesis, Mlardalen University, School of Innovation, Design and Engineering, 2015.
- [55] M. J. Ali and J. Borstler, "Metrics for Requirements Engineering," Master's thesis, Umea University, Sweden, 2006.
- [56] T. Fawcett, "Roc graphs: Notes and practical considerations for researchers," HP Laboratories Palo Alto, Tech. Rep., 2004.

- [57] L. Goncalves, A. Subtil, M. R. Oliveira, and P. Z. Bermudez, “ROC curve estimation: An overview,” *REVSTAT Statistical Journal*, vol. 12, no. 1, pp. 1–20, 2014.
- [58] T. Fawcett, “An introduction to roc analysis,” *Pattern Recogn. Lett.*, vol. 27, no. 8, pp. 861–874, Jun. 2006.
- [59] M. Ganguli, *Making use of JSP*. Wiley, 2002.
- [60] T. G. R. J.N.V.R. Swarup Kumar, M. V. Chaitanya, and A. Tejaswi, *A Novel Model for Software Effort Estimation Using Exponential Regression as Firing Interval in Fuzzy Logic*. Springer, 2011.
- [61] N. Bouhmala, “How Good is the Euclidean Distance Metric for the Clustering Problem,” in *5th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI)*. IEEE, Jul. 2016, pp. 312–315.
- [62] I. B. Mohamad and D. Usman, “Standardization and its effects on k-means clustering algorithm,” *Research Journal of Applied Sciences, Engineering and Technology*, vol. 6, no. 17, pp. 3299–3303, 2013.
- [63] P. Smyth, “Clustering Using Monte Carlo Cross-Validation.” in *The Second International Conference on Knowledge Discovery and Data Mining*, vol. 1, 1996, pp. 26–133.
- [64] K. Singh, D. Malik, and N. Sharma, “Evolving limitations in K-means algorithm in data mining and their removal,” *International Journal of Computational Engineering & Management*, vol. 12, pp. 105–109, 2011.
- [65] K. Sankar, K. S., and P. Jennifer, “Prediction of code fault using naive bayes and svm classifiers,” *Middle-East Journal of Scientific Research*, vol. 20, no. 1, pp. 108–113, 2014.
- [66] C. Andersson and P. Runeson, “A Replicated Quantitative Analysis of Fault Distributions in Complex Software Systems,” *IEEE Transactions on Software Engineering*, vol. 33, no. 5, pp. 273–286, May 2007.
- [67] G. Denaro, L. Lavazza, and M. Pezze, “An empirical evaluation of object oriented metrics in industrial setting,” in *The 5th CaberNet Plenary Workshop, Porto Santo, Madeira Archipelago, Portugal*, 2003.
- [68] N. Nagappan, T. Ball, and A. Zeller, “Mining metrics to predict component failures,” in *Proceedings of the 28th International Conference on Software Engineering*, ser. ICSE ’06. New York, NY, USA: ACM, 2006, pp. 452–461.
- [69] H. Lu, Y. Zhou, B. Xu, H. Leung, and L. Chen, “The ability of object-oriented metrics to predict change-proneness: A meta-analysis,” *Empirical Softw. Engg.*, vol. 17, no. 3, pp. 200–242, Jun. 2012.
- [70] A. Hess, P. Diebold, and N. Seyff, “Towards Requirements Communication and Documentation Guidelines for Agile Teams,” in *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*. IEEE, 2017, pp. 415–418.
- [71] J. Jiarpakdee, C. Tantithamthavorn, and A. E. Hassan, “The Impact of Correlated Metrics on Defect Models,” *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, p. 16, 2018.