CHALMERS
UNIVERSITY OF TECHNOLOGY

Implementation and Evaluation of an Automatic Recommender for Integration

Linlin Wang

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

Master's thesis 2015:06

Implementation and Evaluation of an Automatic Recommender for Integration Test Cases

LINLIN WANG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2015

Implementation and Evaluation of an Automatic Recommender for Integration Test Cases
Linlin Wang

© Linlin Wang, 2015.

Supervisor: Eric Knauss, Computer Science and Engineering
Ola Soder, Axis Communications
Examiner: Miroslaw Staron, Computer Science and Engineering

Master's Thesis 2015:06
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Implementation and Evaluation of an Automatic Recommender for Integration Test Cases
LINLIN WANG
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

Continuous integration promises advantages by enabling software developing organizations to deliver new functions faster. However, implementing continuous integration, especially in large software development organizations, is challenging because of organizational, social, and technical reasons. One of the technical challenges is the ability to rapidly prioritize the test cases which can be executed quickly and trigger the most failures as early as possible. This thesis propose an automatic recommender based on mining correlations between outcome of test and source code changes. The information retrieval measures recall, precision and f-measure, as well as Matthews correlation coefficient(MCC), as the priority metric in determining this correlations. The founding of this correlations can be used to select and execute the recommended test cases instead of a full regression test case, in order to support the decision processes about which test case that should be executed during the integration cycles to get as short feedback loops as possible.

Keywords: continuous integration, continuous deployment, data mining

Acknowledgements

I would like to thank my supervisor Dr Eric Knauss for his invaluable ideas and enthusiastic feedback and my examiner Dr Miroslaw Staron. I would also like to thank Axis communication, especially to my industry supervisor Ola Säder, for giving me the opportunity to conduct this study based on the generously provided historical dataset. Special thanks to Dr Riccardo Scandariato for helpful suggestions.

Linlin Wang, Gothenburg, 06,2015

Contents

Nomenclature **xi**

1 Introduction **1**

2 Background and Related Work **3**
2.1 Background . . . 3
2.2 Related Work . . . 4
2.2.1 Techniques . . . 4
2.2.1.1 Data Mining . . . 4
2.2.1.2 Mining Software Repositories (MSR) . . . 4
2.2.2 Test case selection and prioritization . . . 5

3 Methods **7**
3.1 Research Purpose . . . 7
3.2 Research Questions . . . 7
3.3 Research Methodology . . . 7
3.3.1 Data structure . . . 9
3.3.2 Measurements definition . . . 10
3.3.3 Confusion matrix . . . 12
3.4 Data Analysis . . . 12
3.4.1 Data mining tool: WEKA . . . 12
3.4.2 Data preparation . . . 13
3.4.2.1 ARFF file . . . 13
3.4.2.2 Convert instance . . . 13

4 Case Study **17**
4.1 Axis Communications . . . 17

5 Result and Discussion **19**
5.1 Classifier identification . . . 19
5.2 RQ 1 - Correlation between changes of software artifacts and the outcome of tests . . . 19
5.3 RQ 2 - Test cases selection and prioritization . . . 23

6 Conclusion **27**
6.1 Future work . . . 27

Bibliography **29**

Nomenclature

ARFF Attribute Relation File Format

ETL Extract, Transformation, Load

KDD Knowledge Discovery in Databases

MCC Matthews correlation coefficient

MSR Mining Software Repositories

PCC Pearson's correlation coefficient

SD Streamline Development

SVD Singular Value Decomposition

WEKA Waikato Environment for Knowledge Analysis

1
Introduction

Software organizations nowadays face a market with demands of rapidly changing requirements and pressure to release high quality software faster and more often. Continuous integration as an agile practice, increases the frequency of software releases and shortens the feedback cycle [1]. Software testing as the final "quality gate" for a product and plays an important role in the software lifecycle [2]. It is an activity to identify correctness, completeness as well as software quality, which compass the divers developing phases in continuous integration.
Integration testing is a highly resource demanding and time consuming activity, which calls for being conducted as effective and efficient as possible. The naïve approach to effective testing is to execute as many tests as possible. However, as software evolves, the set of tests tends to grow and not all tests will be equally effective to identify integration problems, some might even become irrelevant over time. Thus, this maximal testing approach leads to the execution of more tests than really necessary, which leads to increased cost and waste of other critical resources [4].
Reducing the number of test cases is one approach for increased efficiency. However, the task of selecting an effective test suite for each integration cycle to achieve the goal of efficient testing is challenging [3]. Software-developing organizations are struggling to find an efficient way of software testing that delivers quality software products that satisfy the requirements, needs, and expectations of stakeholders and avoids costly mistakes and oversights, and at the same time reduces cycle time (e.g. time to market, time to deploy a bug fix, time to give feedback to developers).
The goal of this study is to implement and evaluate an automatic recommender which will help to select a suitable set of functional regression tests on system level that balances the need to find integration problems with the need to execute the tests quickly enough to support the fast pace of continuous integration. The approach of this study is based on the result of mining historical data about source code changes and the results of test case execution. This will lead to a statistical model of (partially hidden) dependencies, represented by a contingency table of test case execution results and changed software parts that can be visualized using the notion of a heatmap [5, 6]. The statistical model can support the decision processes about which test cases that should be executed during the integration cycles to get as short feedback loops as possible.
To evaluate the automatic recommender, this study focuses on the historical data provided by cross-functional teams from large organizations working with large software products. The case study involves two companies from Sweden; both develop embedded software using different flavors of agile software development.

This study builds on previous work, by Eric Knauss et al.[21], who explored how test selection can support continuous integration in large software developing organizations (currently in submission [21]). This study focus on further advancing that work by identifying a suitable data mining approach to fulfill the needs established in the previous work.
The *contribution* of this study is two-fold: First, to define test case selection as a problem related to mining software repositories (here: version control management systems and test result databases). Second, based on the experience with applying our approach in a company, implement an automatic recommender to share challenges and possible mitigation strategies.

2

Background and Related Work

This section provides the background of this study and a review of related literature. The background is introduced, followed by a related work section that consists of related techniques and test case selection and prioritization.

2.1 Background

Nilsson et al. [19] developed a Continuous Integration Visualisation Technique (CIViT) that provides an overview of end-to-end testing activities. The CIViT model serves as a solution to the lack of a holistic, end-to-end understanding of the testing activities and their periodicity in organizations. The model has proven particularly useful as a basis for discussion, which help to identify problems of current testing activities, regarding scope and periodicity, and to reason about suitable measures and to identify how to best improve their testing activities towards continuous integration. In this paper, we propose complementing the CIViT model with automated test case recommendation, to support decision making in the following two scenarios:
Scenario 1 *(Continuous Integration)* — Integration tests are run at various occasions through-out the development cycle [19]. The reason for running those tests however changes during the development cycle. The closer to the release, the more important it is to run all tests and to make sure that no integration problems remain unnoticed. Finally, when delivering a release to a customer, the software should ideally be without errors. Earlier in the development cycle, however, there are different needs. Giving developers quick feedback about integration problems they may have introduced is often more important. When the software product evolves over time, it becomes more and more time consuming to run all tests, and a delay in feedback to the developers become an impediment for the anticipated advantages of continuous integration. Automatic test recommendation can help to select only those tests that are most likely to give information about successful integration, thus allow to balance the tradeoff between giving fast feedback now vs. giving complete feedback later.
Scenario 2 *(Continuous Deployment)* — Continuous deployment allows to deliver new features to the customer continuously. Normally, such a deployment to the customer should be thoroughly tested. However, in some cases, it would be good to deliver a change very fast (i.e. within hours or minutes), for example if it is urgently expected by a customer. This is only possible if the amount of testing is reduced for this fast-track deployment. Automated test recommendation supports fixing a

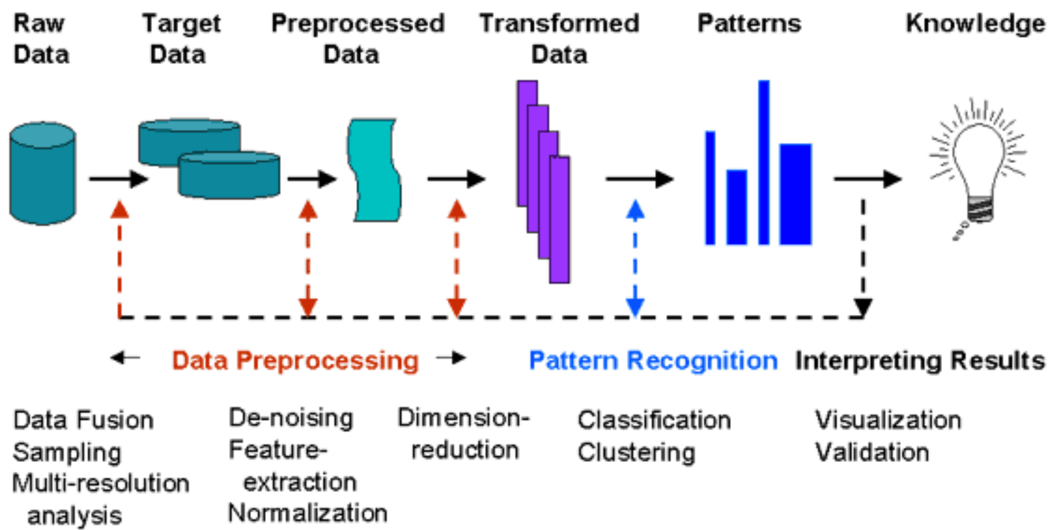


Figure 2.1: Data mining: process of KDD [29].

change request with high priority, deploying it to the customer within hours, and by this balancing value of fast delivery with the risk of introducing problems.

2.2 Related Work

2.2.1 Techniques

2.2.1.1 Data Mining

Data mining is a automatic or (more usually) semiautomatic process of discovering patterns in data stored either in databases, data warehouses, or other information repositories[28]. It is an essential step in the process of Knowledge Discovery from Database(KDD)[29] (see Figure 2.1).
The opportunities for data mining increase as the data growing in size and machines that can undertake the searching become commonplace. In practice, prediction and description are tend to be two high-level primary goals of data mining, which can be achieved using a variety of particular data mining approaches. Association rules and frequent patterns, classification, clustering, test mining are four commonly used techniques in software engineering field [30].

2.2.1.2 Mining Software Repositories (MSR)

Software projects and system continue to grow in size and complexity, changes to its source code occurs.These changes are done incrementally over the lifetime of a project by its various developers. Source control system as CSV record the history of changes to the source code of the software system and stored in a source repos-

itory. Software repositories such as source control repositories, bug repositories, archived communications, deployment logs and code repositories that hold a wealth of valuable information and provide a unique view of the actual evolutionary path taken to realize a software system. The field of Mining Software Repositories(MSR) is by analyzing the rich data available in software repositories to uncover information and assist software developments[23]. Typical MSR process consists of a data preparation phase and a data analysis phase. Here data preparation refers to as ETL : Extract, Transformation, and Load [27]. In this thesis both phases of MSR process will be included and with the main focus on the data analysis phase. Gall et al. [24] presented that software repositories can support developers pointing out hidden code dependencies in order to change legacy systems. Graves [25] indicated that using software change history can support estimating the error probabilities to ease the evolution of reliable software system. In Chen et al. [26] authors show that CVS logs serve as a useful source of information that can assist developers in understanding large systems. Research by Ahmed [22] has demonstrated the value of mining software repositories in assisting managers and developers in performing a variety of software development, maintenance, and management activities.

2.2.2 Test case selection and prioritization

A wide variety of approaches have been developed for rendering reuse more cost-effective via regression test selection [7, 8] and test case prioritization [9, 10, 11, 12]. Yoo and Harman [13] provided a recent survey.
Kim and Porter [14] investigate several regression test selection techniques and proposed a test case selection prioritization approach based on historical test execution data. Their minimization techniques focus on testing parts of the program that have changed since the last testing session and their experimental results suggested that historical fault information is valuable for improving the effectiveness of the regression testing process in the long term. However, they did not consider continuous integration and it is unclear whether this focus on parts of the code that have changed will allow to uncover hidden dependencies during integration testing.
Marculescu et al. [17] propose using a set of predefined fitness functions to assess the suitability of test suite in a potentially vast search space. The approach we proposes in this study is inspired by this generic approach and aim at defining our fitness functions based on source code changes, of which we present here a preliminary version.
Arts et al. [15] and Derrick et al. [16] use formal methods to automatically derive minimal failing test suites based on formally defined sets of properties. Both approaches are based on reusing the existing test suite and studying its sensitivity to capture source code changes to assess the test suite's quality. The property-based testing implied by these approaches however requires a formalization of what should be tested.
Engström et al. [18] provide examples of how EBSE can be applied in industry. In our case, by looking into changes of software artifacts and correlating those with test failures, we take into account not only test quality but also hidden technical dependencies in the software. Generally, each component has been tested thoroughly

before integration. If integration errors occur, then because two or more components were changed in a way that causes problems when these components are put together.
Felderer and Schieferdecker [35] present a taxonomy of risk-based testing that provides a framework to compare risk-based approaches to support test selection. In our study, we focus on the test case prioritization and selection which is one of the risk-based testing specifics in test processes.
Adorf et al.[36] defined a Bayes risk(BR) decision criterion for test selection. They presented a BR-predictor,which takes cost factors, further risk aspects and prior probabilities into account, to guide decision making of a given quality task. Their predictors recommend a risk-based selection of quality assurance tasks. And then prioritize the selected tasks by considering the respective risk decrements by using BR criterion.

3

Methods

This section describes the purpose and approach of this study. It starts with describing research purposes. Then continues with the research questions, and explaining the method for constructing the statistical model and suggesting the selection of test cases. This section ends with data analysis.

3.1 Research Purpose

The main purpose of this study is to understand what extent test failure prediction is possible and what extent this is useful in the test selection for continuous integration, in order to predict which test cases will fail/change based on a list of recent changes. To prioritize the selected suitable test cases on system level could be executed in a short enough period of time to support continuous integration, as the recommendation for the data provider will be another result of this study.

3.2 Research Questions

The background and related works, together with the purpose of this thesis work, lead us to the research questions as follows:

RQ1: How strongly do changes of software artifacts correlate with the outcome of tests on the level of functional integration testing?

Based on the results of research question 1, we plan then to investigate how our automatic integration test recommender will support continuous integration and deployment. We thus phrase our second research question as follows:

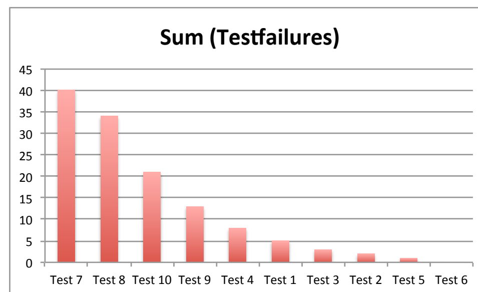
RQ2: How well the recommendation could be for test cases selection and prioritization?

3.3 Research Methodology

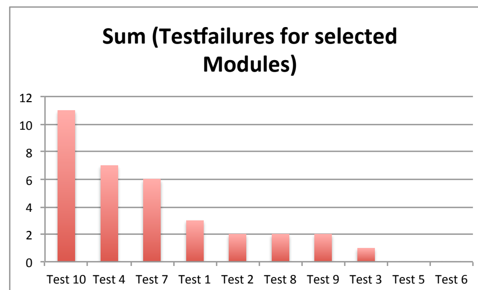
In our approach we propose to use historical data from automatic integration tests in continuous integration or continuous deployment environments to automatically recommend how to prioritize and select tests. In such a scenario we assume that whenever changes to a module are integrated, the whole software product is build and integration tests are automatically executed. The historical analysis takes two

Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10	
0	0	1	3	0	0	2	1	0	3	**Module 1**
0	0	0	0	1	0	0	0	0	1	Module 2
1	0	0	0	0	0	6	2	1	1	Module 3
1	0	2	1	0	0	6	4	1	0	Module 4
0	1	0	4	0	0	1	0	2	4	**Module 5**
1	0	0	0	0	0	1	7	2	3	Module 6
0	1	0	0	0	0	3	1	0	4	**Module 7**
1	0	0	0	0	0	8	5	1	1	Module 8
0	0	0	0	0	0	10	7	3	1	Module 9
1	0	0	0	0	0	3	7	3	3	Module 10
5	2	3	8	1	0	40	34	13	21	Sum
0	2	1	7	0	0	6	2	2	11	Selected Modules

(a) Example heatmap, relating test failures and module changes per build.



(b) Tests, sorted by the amount of total recorded test failures.



(c) Tests, sorted by the amount of recorded failures for selected modules (bold in Fig. 3.1 (a)).

Figure 3.1: Illustrative example of approach: Using historical data on integration test failure and module changes to recommend integration tests.

inputs – the list of source code changes (e.g. results from diff in the source code repository per build) and the results of test case execution (e.g. a list of test cases executed and the results of the execution per build). The method creates a contingency table which shows how often a test case fail if there is a source code change in that particular build (see example in Figure 3.1(a), visualized as a heatmap for better readability [21]).
This contingency table shows which test cases are the most sensitive connected with changes in which source code modules. The sum of test failures can be interpreted as test efficiency: The higher the sum of failures per test and module, the more efficient is this test to identify integration problems in the given module.
In first experiments, we applied a very simple algorithm for test case recommendation, based on two principles:

Principle 1: Tests that recently failed are likely to fail again

In the simplified example in Figure 3.1, we would start by sorting the tests by the amount of failures recorded in our database (see Figure 3.1(b)). If, with respect to our two application scenarios, an organization would have time to run three out of ten integration tests, we would recommend to run Tests 7, 8, and 10.

Principle 2: There are hidden dependencies between modules and tests that can be exploited to improve recommendations

To further improve the recommendations, we also aim at taking into account the sensitivity of the integration tests to specific modules. In our example, if we know that Module 1, 5, and 7 were recently changed, we could prioritize the tests only by the test failures recorded for these specific modules (see Figure 3.1(c)). Thus, if the organization in the example would again aim to run 3 tests, we would now recommend Tests 10, 4, and 7.
In preliminary evaluation, this simple algorithm yields mixed results. First, while the accuracy for recommending tests for many builds is very promising, in between there is a number of builds for which our recommendations are bad. Second, while the overall accuracy quickly improves with each build for which we can analyze test failures and module changes, at some point the performance decreases again, presumably once too much data is considered.
In the remainder of this section, we will present a more suitable schema than the heatmap for analyzing the historical data, define how we plan to measure accuracy of integration tests recommendation can be measured, and then how we plan to address them in our experimental setup.

3.3.1 Data structure

The historical data consist of a list of source code changes and the results of test case execution for each build. To allow systematic analysis of the historical data, we suggest a schema with modules and test cases in the columns and build dates as rows. For each build date and module, table shows whether the module changed

in that particular build. For each date and test case, table shows the result (not executed, failed, succeeded). For example as shown in Table 3.1.

3.3.2 Measurements definition

In order to find the optimal test case we use he information retrieval measures recall, precision, f-measure, and Matthews correlation coefficient(MCC). These measure is based on four categories of results:

1. *True positives(TP):* The set of tests that are both recommended by the recommender system and failed according to the ground truth.
2. *False positives(FP):* The set of tests that is recommended but did not fail according to the ground truth.
3. *True negatives(TN):* The set of tests that are not recommended and that did not fail according to ground truth.
4. *False negatives(FN):* The set of tests that are not recommended but should have been, as they failed according to the ground truth.

Recall indicates the percentage of the tests that were recommended and failed with respect to the ground truth. A high recall (close to 1) is important, because otherwise tests that would have failed will be omitted.

$$\text{recall} = \frac{\mid \text{TP} \mid}{\mid \text{TP} \mid + \mid \text{FN} \mid} \tag{3.1}$$

Precision indicates the percentage of the tests that were recommended which actually failed. A high precision (close to 1) corresponds to a relative speedup of testing by eliminating the need to run tests that do not provide new knowledge about the quality of the system.

$$\text{precision} = \frac{\mid \text{TP} \mid}{\mid \text{TP} \mid + \mid \text{FP} \mid} \tag{3.2}$$

Recall and precision relate to each other. The easiest way to get a high recall is to simply recommend all tests. In that case, all tests that could fail are selected but the precision is minimal and no execution time is saved.

Table 3.1: Proposed structure of historical data about source code changes and test failures for automatic integration test case recommendation.

Date	M_1	M_2	...	M_n	T_1	T_2	...	T_m
2015/01/03	nc	nc	...	c	s	f	...	n
2015/01/04	c	ig	...	c	f	f	...	s

For M = Module: nc = No changes, c = Changes, ig = not presented

For T = Test Case: s = Succeeded, f = Failed, n = not executed

Under the assumption that both precision and recall are equally important, it makes sense to compute the f-measure (the geometric mean of recall and precision), defined as:

$$\text{f-measure} = \frac{2 * \text{recall} * \text{precision}}{\text{recall} + \text{precision}} \tag{3.3}$$

The *f-measure* allows comparing the overall performance and this allows to choose the optimal set of test cases to run given the changed source code modules and the contingency table from historical analysis.
The Matthews correlation coefficient(MCC), as a contingency matrix method of calculating coefficient(PCC) [38], is in essence a correlation coefficient between the observed and predicted classifications. It returns a value between -1 and +1. According to Rumsey [37], a coefficient that is

- +1.0 indicates a perfect prediction which tests that are recommended by the recommender system and failed according to the ground truth.
- +0.70 or higher indicates a very strong positive relationship.
- +0.40 to +0.69 indicates strong positive relationship.
- +0.20 to +0.39 indicates moderate positive relationship.
- +0.19 to -0.19 indicates no or negligible relationship.
- -0.20 to -0.29 indicates weak negative relationship.
- -0.30 to -0.39 indicates moderate negative relationship.
- -0.40 to -0.69 indicates strong negative relationship.
- -0.70 or lower indicates very strong negative relationship.
- -1.0 indicates total disagreement between prediction and observation.

If any of the four sums in the denominator is zero, the denominator can be arbitrarily set to one; this result can be shown to be the correct limiting value.

$$\text{MCC} = \frac{TP * TN - FP * FN}{((TP + FP)(TP + FN)(TN + FP)(TN + FN))^{1/2}} \tag{3.4}$$

For the continuous integration this measure is crucial as it enables automated selection of test cases. Together with the ability of automatically adjust the contingency table (along with each integration/test cycle), this approach reduces the effort for test planning and increases the chances of finding defects already during the integration.

3.3.3 Confusion matrix

The confusion matrix contains information about actual and predicted classifications done by a classification system. It reports how good the classifications is in terms of what it gets right and what it gets wrong. Weka as the classification system for this study is presented in Section 3.4.1.
Table 3.2 shows a confusion matrix for a three class classifier. Each column of the matrix represents the instances in a predicted class, while each row represents the instances in an actual class. The positive value that the classifier has predicted as positive is true positive(TP); positive value that the classifier predicted as negative is false negative(FN), which is the sum of values in the corresponding row excluding TP; negative value that classifier predicted as negative is true negative (TN), which is the sum of all columns and rows for the certain class excluding the class's column and row; and negative value the classifier predicted as positive is false positive(FP), which is the sum of values in the corresponding column excluding TP. For instance, in Figure 3.2, AA is the true positive value of class A; (AB+AC) is the false negative value of class A; (BB+BC+CB+CC) is the true negative value of class A; (BA+CA) is the false positive value of class A.

		Predicted		
		A	B	C
Actual	A	AA	AB	AC
	B	BA	BB	BC
	C	CA	CB	CC

Table 3.2: Confusion matrix for a three class classifier

3.4 Data Analysis

3.4.1 Data mining tool: WEKA

The workbench WEKA, stands for Waikato Environment for Knowledge Analysis, provides a uniform interface to many different learning algorithms, along with data pre-processing methods. It provides three main ways to work on mining problem: the Explore,the Experimenter and the KnowledgeFlow.
The Explorer (Figure 3.2) for applying transformation and algorithms to the data, Which consist of 6 different tabs: *Preprocess* for loading and manipulate the data being acted on, *Classify* for operating the Classification and regression algorithms, *Cluster* for learning Cluster algorithms for the data, *Associate* for learning association rules algorithms for the data, *Select attribute* for learning attribute selection algorithms and select the most relevant attributes in the data, and *Visualize* for visualizing the relationship between attributes. The Experimenter for controlling experiments, which allows user to design algorithms, run experiments and analyze result that are statistically significant over multiple runs. The knowledge Flow for graphically designing a pipeline for the state problem, which includes the loading and transforming of input data, running of algorithms and the presentation of results.

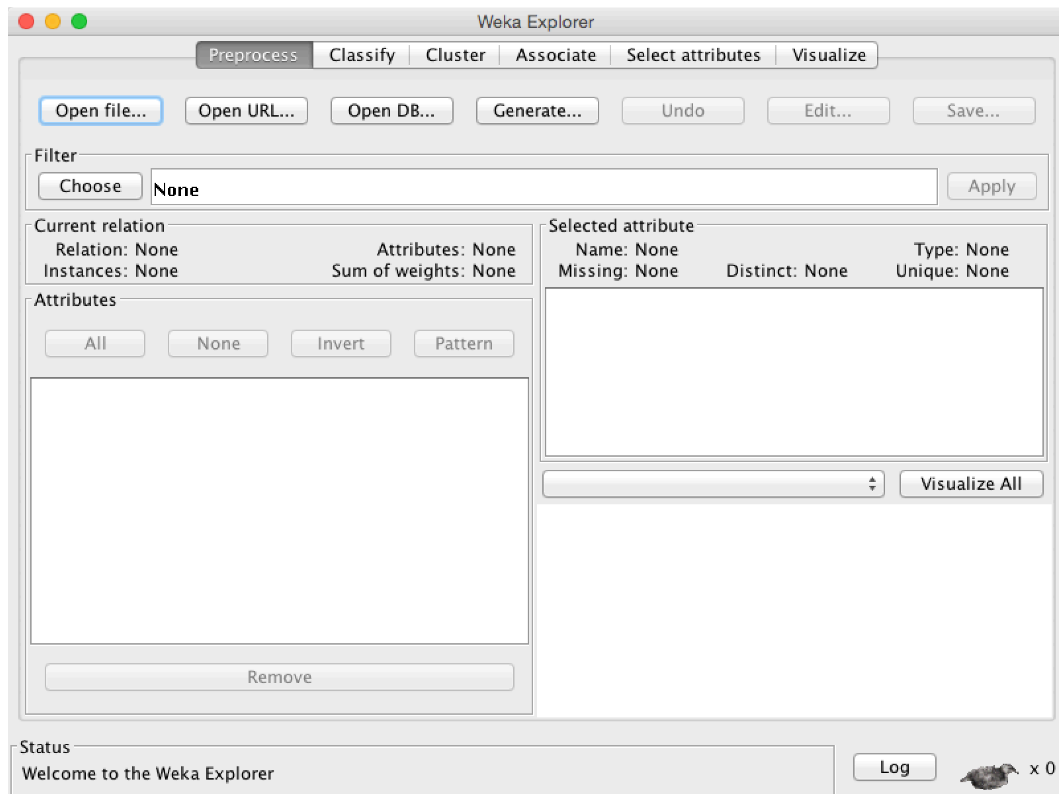


Figure 3.2: Weka explorer interface

3.4.2 Data preparation

3.4.2.1 ARFF file

In order to operate with Weka, an *ARFF file* is needed. It is a standard way of representing dataset which involves two distinct sections: Header and Data. The header of the ARFF file contains the name of the relation, a list of the attributes and their types. The data section contains the data declaration line and the actual instance lines.Attribute values for each instance are delimited by commas. They must appear in the order that they were declared in the header section. Figure3.3 shows an example file in arff form.

3.4.2.2 Convert instance

For both modules and tests package, the type of attribute is class. Modules package attributes have 3 classes : nc, c and ig. nc denotes a no changed module. c denotes a changed module. ig denotes different types of padding, for example if a package was not used, which will not be considered in this study. Three classes in the tests package are s, f and n, where s denotes a successful test. f denotes a failed test. n denotes a test not be executed. For instance, in the historical dataset, SHA1 hash (packages in Git) or a tag of some sort (packages in CVS) are valid IDs. If changes in the code have occurred then the hash/tag will not be the same in the next test run. In this case, the instance hash code/IDs will be converted to f or c in the arff file(see Figure3.4).

```
@relation arff_example

@attribute Time string
@attribute power string
@attribute axistrapv1 string
@attribute axistrapv2c string
@attributeaxistrapv3 string

@data
LFPP3367STHFUNCSET2130603235436,nc,c,c,nc
LFPP3367STHFUNCSET2130604112118,nc,nc,c,ig
LFPP3367STHFUNCSET2130604221846,nc,ig,ig,c
```

Figure 3.3: Example of arff file

In final arff file, name of modules and tests will be the attributes. And it's corresponding values will be the instances in this case.

```
@relation module_packages

@attribute LFPP3367STHFUNCSET2130603235436 string
@attribute LFPP3367STHFUNCSET2130604112118 string
@attribute LFPP3367STHFUNCSET2130604221846 string
@attribute LFPP3367STHFUNCSET2130605221418 string
@attribute LFPP3367STHFUNCSET2130606221303 string
@attribute LFPP3367STHFUNCSET2130607221255 string
@attribute LFPP3367STHFUNCSET2130608221314 string

@data
f52828de4efaeb18550ef00b084f729e2efc9c80,-2,-2,f52828de4efaeb18550ef00b084f72
9e2efc9c80,c5f5e7dff0356c0ab10a8f8250c247ae8e5cbcf8,c5f5e7dff0356c0ab10a8f825
0c247ae8e5cbcf8,c5f5e7dff0356c0ab10a8f8250c247ae8e5cbcf8
4e83e7dfe601477a361680fd9169d48960107ac3,4e83e7dfe601477a361680fd9169d4896010
7ac3,4e83e7dfe601477a361680fd9169d48960107ac3,79a384cf46f1dc84de67f6bbf9e04ec
428ba9e68,79a384cf46f1dc84de67f6bbf9e04ec428ba9e68,91301434d4828a24dc2ac2535e
9628c11ff4c42a,91301434d4828a24dc2ac2535e9628c11ff4c42a
```

(a) Origenal arff file (Before convert)

```
@relation module_packages_convert

@attribute LFPP3367STHFUNCSET2130603235436 {nc,c,ig}
@attribute LFPP3367STHFUNCSET2130604112118 {nc,c,ig}
@attribute LFPP3367STHFUNCSET2130604221846 {nc,c,ig}
@attribute LFPP3367STHFUNCSET2130605221418 {nc,c,ig}
@attribute LFPP3367STHFUNCSET2130606221303 {nc,c,ig}
@attribute LFPP3367STHFUNCSET2130607221255 {nc,c,ig}
@attribute LFPP3367STHFUNCSET2130608221314 {nc,c,ig}

@data
nc,ig,ig,nc,c,nc,nc
nc,nc,nc,c,nc,c,nc
```

(b) Converted arff file (After convert)

Figure 3.4: Example of convert instances

4 Case Study

This study was conducted with the collaboration of a company called Axis Communications, which will be introduced in the following sections shortly.

4.1 Axis Communications

Axis Communications is an IT company offering network video solutions for professional installations. The company is the global market leader in network video, driving the ongoing shift from analog to digital video surveillance. Axis products and solutions focus on security surveillance and remote monitoring, and are based on innovative, open technology platforms. The agile principles with frequent deliveries to the main branch and empowered software development teams are mainly used on the software development processes in Axis. Cross functional teams that include designers, architects and testers take the responsibility for a large part of the development process. Test selection promises to help with optimizing time and other resources needed for integration testing at certain parts of this process.
Axis provides two dataset files: result file and package file, in *csv* format, which are the log of testing and module execution result of one year long. The result file has tests in the columns and dates as rows. It records the test result with three types: did not execute, failed, and succeeded. There are also minuses in the file which means that there is no data available for some reason. For instance, 1 denotes an unknown error occurred; 2 denotes the test was skipped; 3 denotes the test does not exist. In our study, the different minuses will be treat as the same value. The package file has modules in the columns and dates as rows. It has the hashed key to mark the statue of the module. If the changes in the code have occurred then the hashed code will not be the same in the next test run, which the module will be record as change.
Overview of the data structure in package file as shown in Figure 4.1. The x-axis represents the number of module changes. The y-axis represents the number of modules. There are 933 module instances in the package file and totally 163 changes occurred. 68% which is 631 out of 933 module instances has no change at all, and it is not included in Figure 4.1. 11.7% module instances only changed one time. There is one module instance changed 23 times in total, which has the maximum time of change. Figure 4.2 shows the data structure in result file. The x-axis represents the number of test failures. The y-axis represents the number of tests. There are 919 test instances in the result file and 358 records of test failure in total. 86.7% which is 798 out of 919 test case succeed without any failure, and it is not included

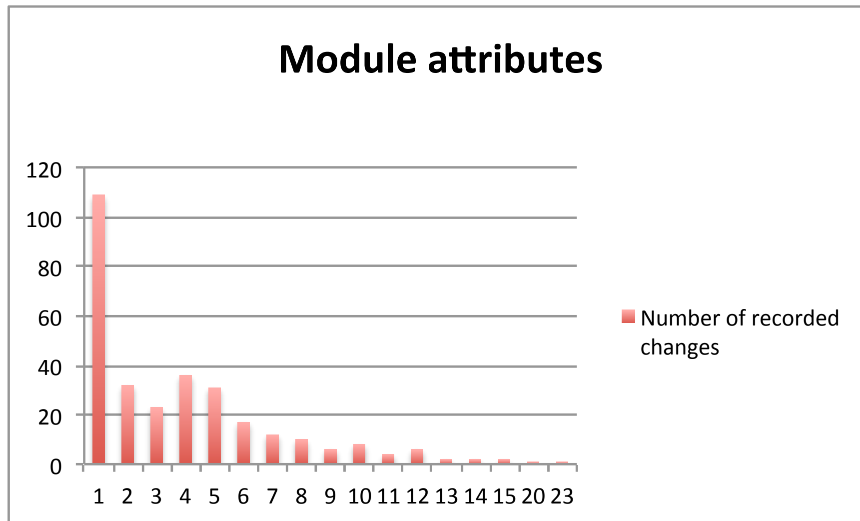


Figure 4.1: Data structure in package file.

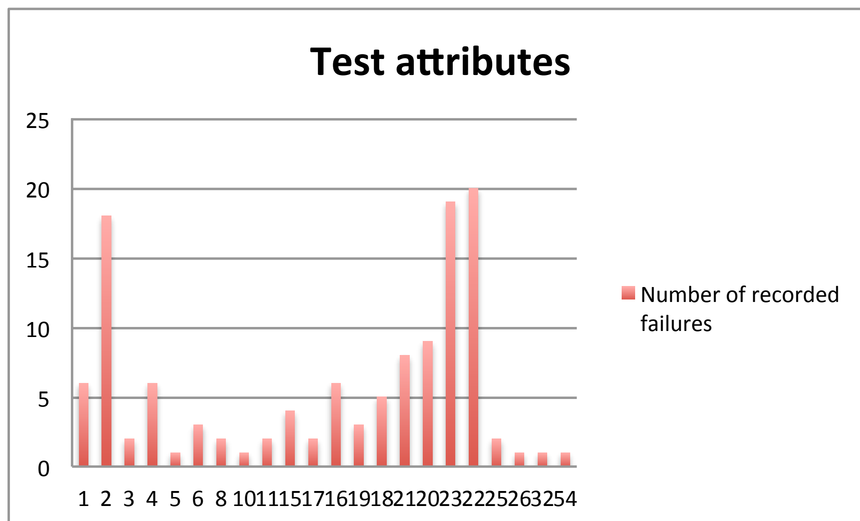


Figure 4.2: Data structure in result file.

in Figure 4.2. 5.5% test instances has less than 10 failures. The test instance that has less than 30 failures are around 2.8%. There is only one test that has maximum number of failure which is 54.

5
Result and Discussion

In this section, the results of the evaluation are presented and discussed, according to the method proposed in section 3.3. The structure is divided with the aim of clearly answering the research question that presented in section 3.2.

5.1 Classifier identification

RandomForest and NaiveBayes are two classifier for the first experimentation. The Naive Bayes approach is known to handle missing data quite well, as it considers attributes separately both at model construction time and prediction time [20]. Thus, if a data instance has a missing value for an attribute, it can be ignored while preparing the model, and ignored when a probability is calculated for a class value. Compared with Naive Byes approach, RandomForest at most of the time given better result than NaiveBayes, even though it requires far more processing time than NaiveBays. As an example result shown in Figure 5.1, by selecting the same system test attributes as class, the accuracy by RandomForest (Figure 5.1(a)) is higher than the one by NaiveBays (Figure 5.1(b)). For instance, the MCC of class s, in 5.1(a) is 31.4% whereas in 5.1(b) is 10.0%; the f-Measure of class s, in 5.1(a) is 87.3% while in 5.1(b) is 82.6%.
Therefor, we decided to use RandomForest classifier for later experimentation in order to get consistent and marked improvements in accuracy. Classifiers that been used during the study followed the conventional 10-fold cross validation. Data is broken into 10 sets of size n/10. 9 datasets use for training the classifier and 1 for testing. The process repeat 10 times and take a mean accuracy.

5.2 RQ 1 - Correlation between changes of software artifacts and the outcome of tests

In general, we got good prediction result of test success rather than test failure. In other words, changes of software artifacts are stronger correlate with test success than test failure on the level of functional integration testing.
Figure 5.2 displays the overview of the evaluation output. We found among total 919 test case, there are 32% of test cases has no value of any measurements. In the following result, we did not include these 32% of test cases. The result is presented in two scenarios as showing below:

```
Correctly Classified Instances         200               78.4314 %
Incorrectly Classified Instances        55               21.5686 %
Kappa statistic                          0.2541
Mean absolute error                      0.2042
Root mean squared error                  0.3373
Relative absolute error                 80.4634 %
Root relative squared error             95.1418 %
Coverage of cases (0.95 level)          97.2549 %
Mean rel. region size (0.95 level)      60.7843 %
Total Number of Instances              255

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.964    0.758    0.798      0.964    0.873      0.314    0.698     0.843     s
                 0.000    0.000    0.000      0.000    0.000      0.000    0.848     0.375     f
                 0.246    0.040    0.636      0.246    0.354      0.304    0.691     0.465     n
Weighted Avg.    0.784    0.583    0.746      0.784    0.740      0.306    0.699     0.749

=== Confusion Matrix ===

   a   b   c   <-- classified as
 186   0   7 |   a = s
   4   0   1 |   b = f
  43   0  14 |   c = n
```

(a) Result of Random forest classifier.

```
Correctly Classified Instances         183               71.7647 %
Incorrectly Classified Instances        72               28.2353 %
Kappa statistic                          0.1015
Mean absolute error                      0.1882
Root mean squared error                  0.4339
Relative absolute error                 74.179  %
Root relative squared error            122.3808 %
Coverage of cases (0.95 level)          71.7647 %
Mean rel. region size (0.95 level)      33.3333 %
Total Number of Instances              255

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.886    0.806    0.774      0.886    0.826      0.100    0.535     0.772     s
                 0.000    0.000    0.000      0.000    0.000      0.000    0.474     0.022     f
                 0.211    0.111    0.353      0.211    0.264      0.122    0.530     0.265     n
Weighted Avg.    0.718    0.635    0.665      0.718    0.684      0.103    0.532     0.644

=== Confusion Matrix ===

   a   b   c   <-- classified as
 171   0  22 |   a = s
   5   0   0 |   b = f
  45   0  12 |   c = n
```

(b) Result of NaiveBays classifier.

Figure 5.1: Example result of select a system test attribute as the class to apply RandomForest and NaievBayes classifier, with full dataset in the package file.

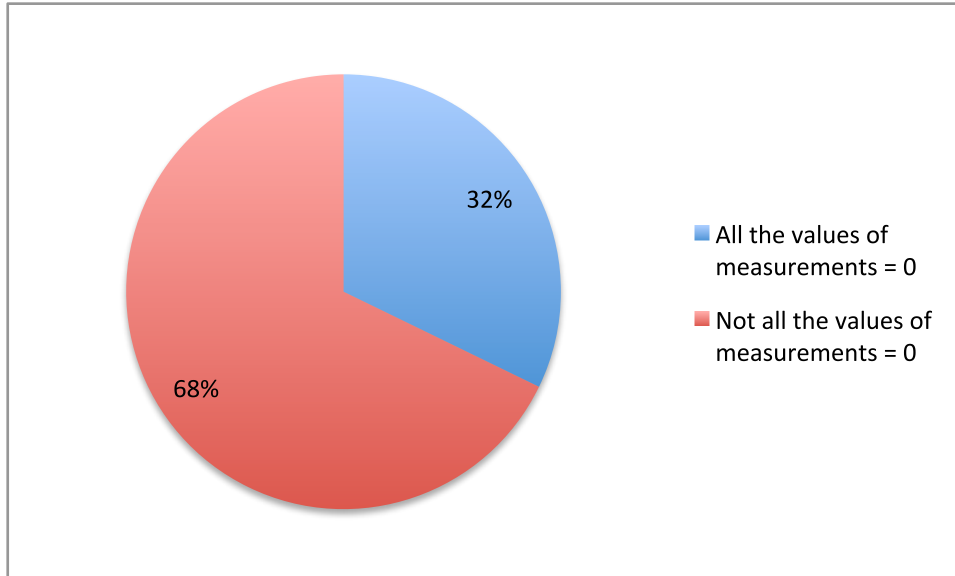


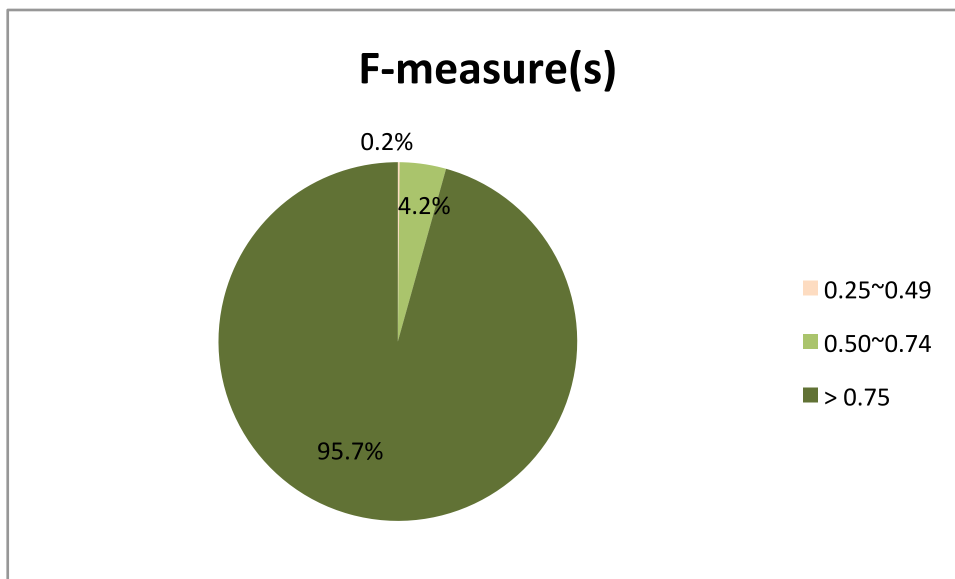
Figure 5.2: Overview of result.

Scenario 1 : f-measure

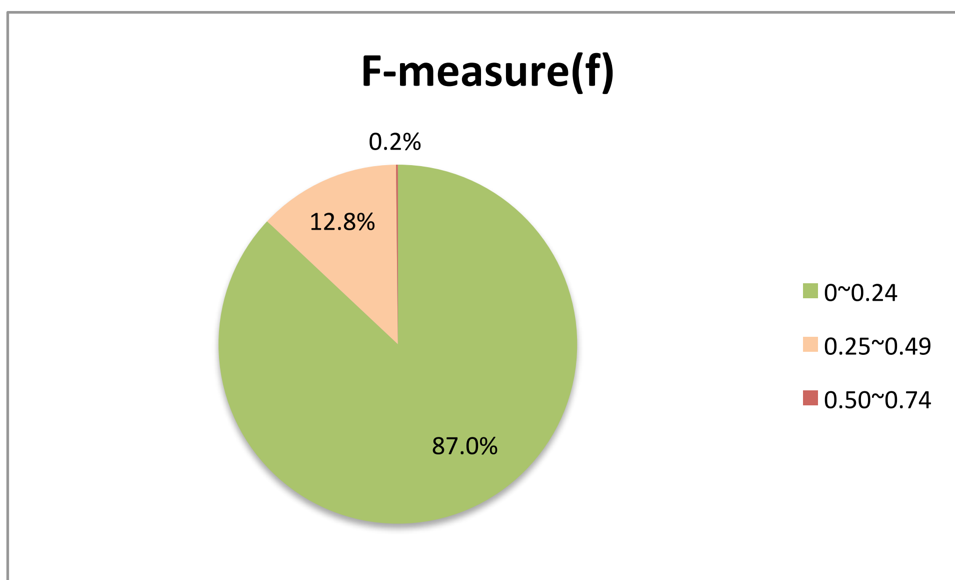
In order to evaluate how good the classifier in predicting test success or failure, we evaluated the 68% (which is 623) of f-measure of class *s* and class *f* which is illustrated in Figure 5.3. An overview of the comparison of f-measure of class *s* and class *f* is shown in Figure 5.4.
By going into details, as shown in Figure 5.3(a), for f-measure of class *s* nearly 96% of test cases are over 0.75; around 4% of test cases are between 0.50 and 0.74; less than 1% of test cases are between 0.25 and 0.49. According to Figure 5.3(b), for f-measure of class *f*, less than 1% of test cases are between 0.50 and 0.74; almost 13% of test cases are between 0.25 and 0.49; 87% of test cases are between 0 and 0.24.

Scenario 2 : MCC

In order to evaluate whether we could trust the prediction, we checked the MCC of class *s* and *f*. An overview of MCC of class *s* and class *f* comparison is shown in Figure 5.6. Figure 5.5 illustrates the comparison in detail.
As can be seen from Figure 5.5, for MCC of class *s*(5.5(a)), almost 31% of test cases are over 0.70; 26.5% of test cases are between 0.40 to o.69; around 42% of test cases are between 0.20 to 0.39; only less than 1% of test cases are between -0.19 to 0.19. Which means that, for the relationship between the predicted test success and actual test success, nearly 31% of test cases have very strong positive relationship; around 42% of test cases have moderate positive relationship; less than 1% of test cases have no or negligible relationship.
Figure 5.5(b) shows the result of MCC of class *f*. For MCC of class *f*, only 1.3% of test cases are between 0.40 and 0.69; almost 12.5% of test cases are between 0.20 and 0.39; around 86% of test cases are between -0.19 and 0.19. This implies that, for relationship between predicted test failure and actual test failure, there are only



(a) f-Measure of class s.



(b) f-Measure of class f.

Figure 5.3: f-Measure of class s and class f.

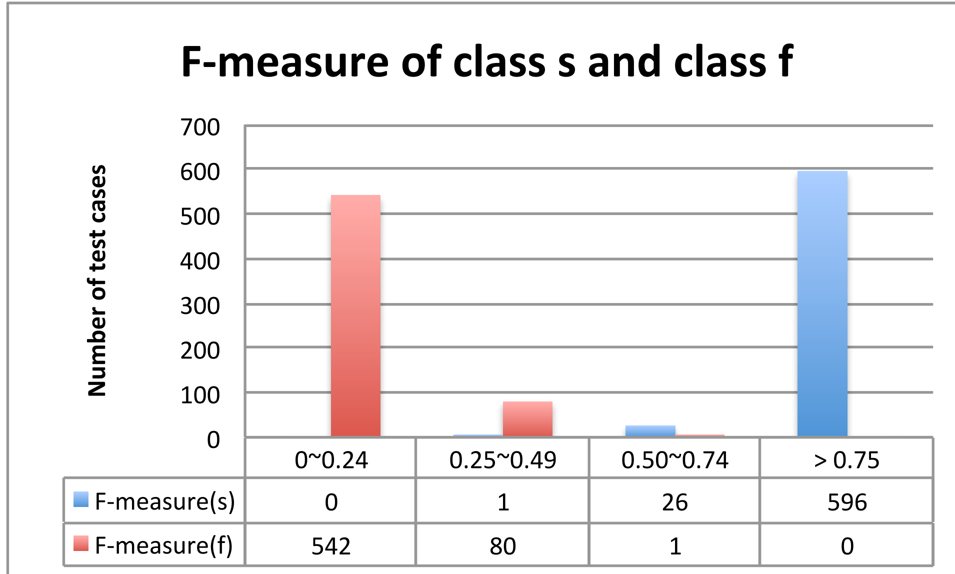


Figure 5.4: Comparison of f-measure of class *s* and class *f*.

1.3% of test cases have the strong positive relationship; 12.5% of test cases have the moderate positive relationship; nearly 86% of test cases have no or negligible relationship.

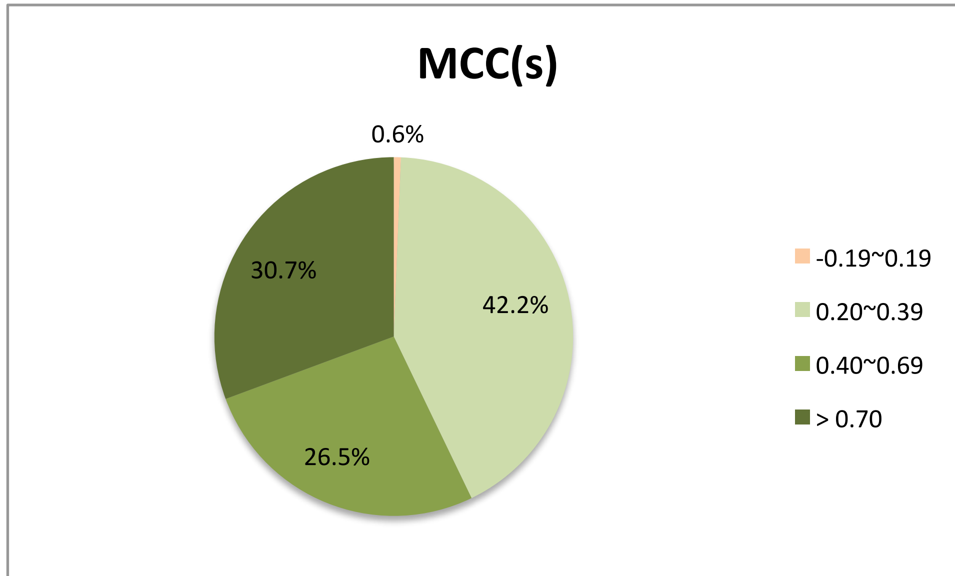
5.3 RQ 2 - Test cases selection and prioritization

This section answers the second research question related to the test cases selection and privatization based on the test effectiveness of module change. We defined three types of test cases Firstly, we defined four types of test cases. Then, we discussed how good our recommendation about test cases selection and prioritization. Table 5.1 shows four types of test cases and the corresponding recommendation for test case selection.

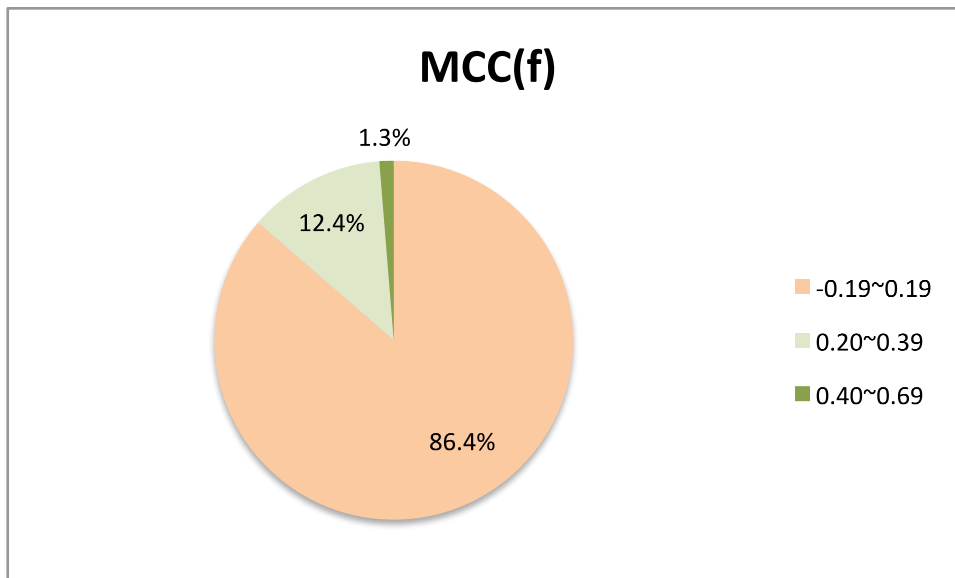
Types of test cases	How to identify	Recommendation strategy
Type I	Mainly ignored	Ignore
Type II	No failure in data	Do not execute
Type III-a	Have failure in data MCC(s) >= 0.4 f-measure(s) >= 0.75	Use the classifier
Type III-b	Have failure in data MCC(s) <0.4	Unknown

Table 5.1: Four types of test cases.

As we mentioned in the beginning of 5.2, among total 919 test case, there are 32% of test cases has value zero for all the measurements. Those test cases are only stated as "ignore" in the log. In other words, those 32% of test cases could be ignored since



(a) MCC of class s.



(b) MCC of class f.

Figure 5.5: MCC of class s and f.

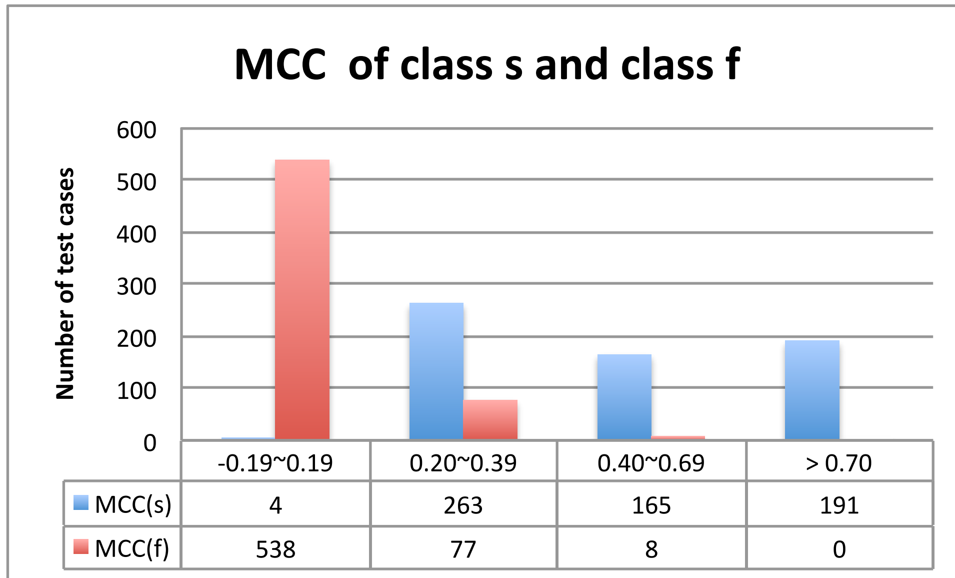


Figure 5.6: Comparison of MCC of class s and class f.

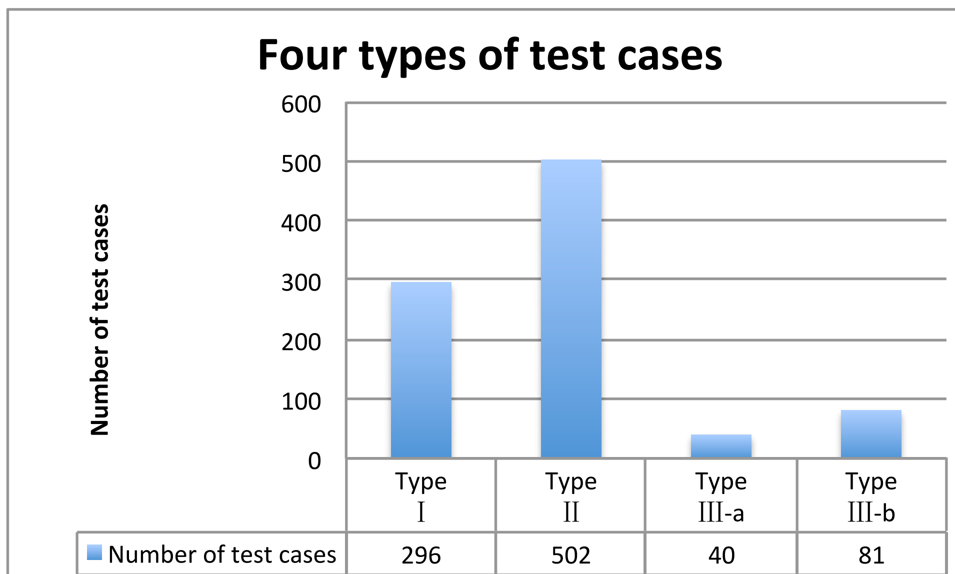


Figure 5.7: Four types of test cases and the corresponding amounts.

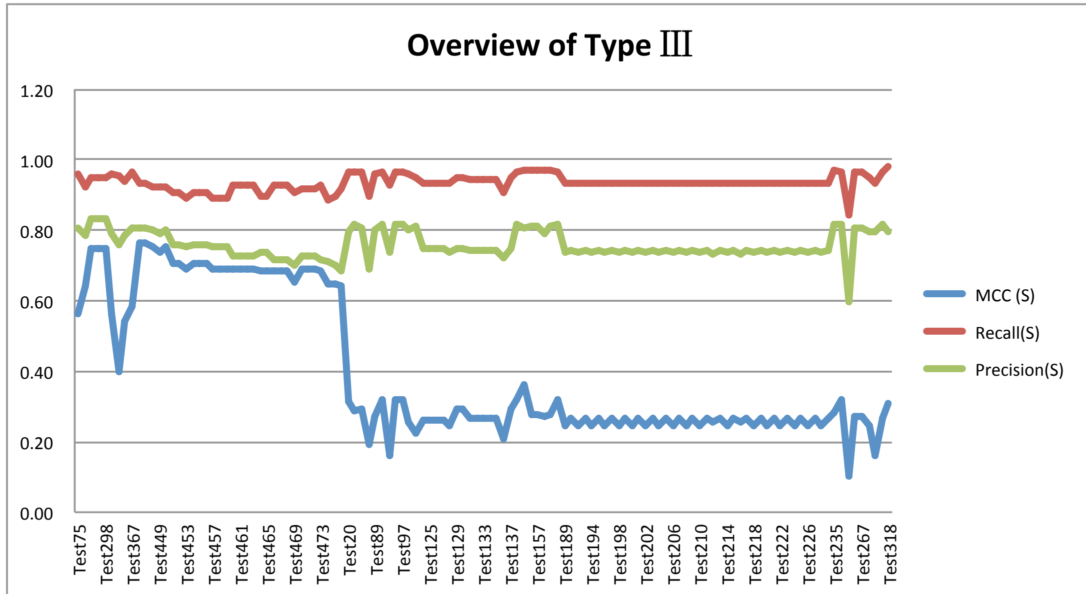


Figure 5.8: Overall recommendation evaluation of TypeIII.

it will not be affected by any changes of module and it has been always stated as "ignore". We define this type of test cases as *Type I*.
By analyzing the 68% of test cases, we found that, there are 502 out of 623 test cases that have no failure in the history log (see Figure 5.7). Those test cases are stated as either "success" or "ignore" but not "fail". We define this type of test cases as *Type II*. This type of test case could always not be executed since it will either be succeed or be ignored once it be executed, and will not be affected by any changes of the module.
We define test cases, which have record failures in data and MCC value of class s is over 0.4 together with f-measure of class s is over 0.75, as *Type III-a*. For those test cases that have record failures in data but the MCC of class s is lower than 0.4 even thought the f-measure of class s is over 0.7, we have not yet figure out how to deal with them. We define this type of test cases as *Type III-b*.
Figure 5.8 shows the recommendation tendency of Type III. The average recall is over 0.90, while precision is around 0.76. Which means that Only 6% of the test cases that fail were not recommended and only 23% of the test cases we did recommend but do not fail.

6
Conclusion

This study set out to find the correlation between the changes of software artifacts and outcome of tests on the level of functional integration testing and what extent this is useful in the test selection for continuous integration. Through a single case study that carried out at Axis Communications, correlation between the changes of software artifacts and outcome of tests were found, four types of test cases were defined for test case selection based on the information retrieval area (f-measure and MCC), and recommendation strategy were conducted.
In relation to RQ1 - correlation between the changes of software artifacts and outcome of tests on the level of functional integration testing, we found out that the changes of software artifacts are more correlated with test success rather than test failure. The quality of the founding is increasing with more dataset take into account for the experiment. However, the quality of the dataset is hard to ensure. For instance, refactoring by the development teams can reduce the impact of hidden technical dependencies on test failure over time.
In relation to RQ2 - test cases selection and prioritization, according to historical log of the test case, the test cases were categorized in four types. Test cases that mainly have recorded as ignored could be ignore as always. Test cases that have no failure record would not be executed. Test cases that have recorded failures in the data and for which the classifications score MCC values for s over 0.4 as well as f-measure of class s over 0.75, could use the classifier. Unfortunately, our classifier was not good enough to deal with the test cases with MCC of class s under 0.4. Given this strategy, nearly 87% of test cases are recommended to not be executed (because they did not fail in history). 13% of test cases can be prioritized based on the data mining approach used in this thesis. Results differ from strong recommendations (based on the MCC) to moderate recommendations, but in average show a moderate to strong relationship to actual test success (avg. MCC = 0.40). In average, the recall is around 0.93, which means that only 7% of the tests that fail will not be recommended, while the precision is around 0.76, which means that out of the recommended test cases, only 24% will not fail.

6.1 Future work

This case study focused on studying a single company. It would be interesting to involve multiple companies. This in order to see if the results found are unique to the studied company or could apply to other companies as well. Also, for future study, how would the result be if consider the interrelationship of software artifacts and

tests (e.g. if a certain test case fail in response to two or more modules changing simultaneously.). For instance, consider a hidden technical dependency between Module M1 and Module M2. During a refinement of M1, a bug could be introduced that does not yet surface, as M2 does not rely on a correct implementation in this particular case. The integration tests on the first build with the new version of M1 therefore does not produce test failures. Later, M2 is refined and after integration, Test T3 starts to fail. Association Rule Learning promises better ability to discover the relation between modules on different level of continuous integration testing, and how dependencies affect the outcome of tests,According to Witten [20]. Finally, it would also be beneficial to further investigate to take into account business value.

Bibliography

[1] Goodman, D., Elbaz, M.: It's not the pants, it's the people in the pants, learnings from the gap agile transformation what worked, how we did it, and what still puzzles us. In: Agile Conference AGILE 2008, pp. 112-115 (August 2008).

[2] Dustin, E. (2002). Effective Software Testing: 50 Ways to Improve Your Software Testing, Addison-Wesley,Boston.

[3] P. M. Duvall, S. Matyas, and A. Glover, Continuous integration: improving software quality and reducing risk. Pearson Education, 2007.

[4] Dustin, E., Rashka, J., & Paul, J. (1999). Automated software testing: Introduction, Management, and Performance, Addison-Wesley, Boston, MA.

[5] M. Staron, J. Hansson, R. Feldt, A. Henriksson, W. Meding, S. Nilsson, and C. Hoglund, "Measuring and visualizing code stability-a case study at three companies," in Software Measurement and the 2013 Eighth International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2013 Joint Conference of the 23rd International Workshop on. IEEE, 2013, pp. 191-200.

[6] R. Feldt, M. Staron, E. Hult, and T. Liljegren, "Supporting software decision meetings: Heatmaps for visualizing test and code measurements," in Software Engineering and Advanced Applications (SEAA), 2013 39th EUROMICRO Conference on. IEEE, 2013, pp. 62-69

[7] A. Orso, N. Shi, and M. J. Harrold. Scaling regression testing to large software systems. In Proceedings of the 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2004

[8] L. White and B. Robinson. Industrial real-time regression testing and analysis using firewalls. In Proceedings of the International Conference on Software Maintenance, Sept. 2004

[9] S. Yoo, M. Harman, P. Tonella, and A. Susi. Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge. In Proceedings of the Eighteenth International Symposium on Software Testing and Analysis, pages 201-212, 2009

[10] H. Do and G. Rothermel. On the use of mutation faults in empirical assessments of test case prioritization techniques. IEEE Transactions on Software Engineering, 32(9), Sept. 2006.

[11] S. Elbaum, D. Gable, and G. Rothermel. The impact of software evolution on code coverage. In Proceedings of the International Conference on Software Maintenance, pages 169-179, Nov. 2001

[12] M. Staats, P. Loyola, and G. Rothermel. Oracle-centric test case prioritization. In Proceedings of the International Symposium on Software Reliability Engineering, pages 311-320, Nov. 2012.

[13] S. Yoo and M. Harman. Regression testing minimisation, selection and prioritisation: A survey. Software Testing, Verification and Reliability, 22(2), 2012.

[14] J. M. Kim and A. Porter. A history-based test prioritization technique for regression testing in resource constrained environments. In Proceedings of the International Conference on Software Engineering, pages 119-129, May 2002.

[15] T. Arts, J. Hughes, J. Johansson, and U. Wiger, "Testing telecoms software with quviq quickcheck," in Proceedings of the 2006 ACM SIGPLAN workshop on Erlang. ACM, 2006, pp. 2-10.

[16] N. Walkinshaw, K. Bogdanov, J. Derrick, and J. Paris, "Increasing functional coverage by inductive testing: a case study," in Testing Software and Systems. Springer, 2010, pp. 126-141.

[17] B. Marculescu, R. Feldt, and R. Torkar, "Practitioner-oriented visualization in an interactive search-based software test creation tool," in Software Engineering Conference (APSEC, 2013 20th Asia-Pacific. IEEE, 2013, pp. 87-92

[18] E. Engstrom, R. Feldt, and R. Torkar, "Indirect effects in evidential assessment: a case study on regression test technology adoption," in Proceedings of the 2nd international workshop on Evidential assessment of software technologies. ACM, 2012, pp. 15-20.

[19] A. Nilsson, J. Bosch, and C. Berger, "Visualizing testing activities to support continuous integration: A multiple case study," in Agile Processes in Software Engineering and Extreme Programming. Springer, 2014, pp.171-186

[20] Witten, I. H., Frank, E., & Hall, M. A. (2011). Data Mining: Practical Machine Learning Tools and Techniques (3rd ed.). US: Morgan Kaufmann.

[21] E. Knauss, M. Staron and W. Meding and O. Söder, A. Nilsson, and M. Castell, "Supporting Continuous Integration by Code-Churn Based Test Selection," in Workshop on Rapid Continuous Software Engineering (RCoSE '15), Florenz, Italy,2015,in submission.

[22] A.E.Hassan,"Mining Software Repositories to Assist Developers and Support Managers," ICSM 2006: 22nd IEEE International Conference on Software Maintenance, September 24, 2006 - September 27. 2006, Philadelphia, PA, United states: IEEE Computer Society , 2006. 339-342. Print.

[23] A. E. Hassan. The road ahead for mining software repositories. In FoSM: Frontiers of Software Maintenance, pages 48–57, October 2008.

[24] H. Gall, K. Hajek, and M. Jazayeri. Detection of logical coupling based on product release history. In Proceedings of the 14th International Conference on Software Maintenance, Bethesda, Washington D.C., Nov. 1998.

[25] T. L. Graves, A. F. Karr, J. S. Marron, and H. P. Siy. Predicting fault incidence using software change history. IEEE Transactions on Software Engineering, 26(7):653–661, 2000.

[26] A. Chen, E. Chou, J. Wong, A. Y. Yao, Q. Zhang, S. Zhang, and A. Michail. CVSSearch: Searching through source code using CVS comments. In Proceedings of the 17th International Conference on Software Maintenance, pages 364–374, Florence, Italy, 2001.

[27] V. Panos , S. Alkis, and S. Spiros. Conceptual modeling for ETL processes. In DOLAP '02: Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP, pages 14–21, New York, NY, USA, 2002. ACM.

[28] J. Han, M. Kamber, Data Mining: Concepts and Techniques, London:Academic Press, 5, 2001.

[29] Sapphire, Large Scale Data Mining and Pattern Recognition,1999

[30] Q. Taylor,C. Giraud-Carrier, Applications of data mining in software engineering, Int. J. Data Analysis Techniques and Strategies, Vol. 2, No. 3, 2010

[31] B. Livshits and T. Zimmermann, Dynamine: finding common error patterns by mining software revision histories, SIGSOFT Softw. Eng. Notes, vol. 30, no. 5, pp. 296–305, 2005.

[32] J. Sliwerski, T. Zimmermann, and A. Zeller, "When do ´ changes induce fixes?" in MSR '05: Proceedings of the 2005 international workshop on Mining software repositories. New York, NY, USA: ACM, 2005, pp. 1–5.

[33] H. Kim and N. Nachiappan, Empirically Detecting False Test Alarms Using Association Rules. In Companion Proceedings of the 37th International Conference on Software Engineering,2015.

[34] Imola K.Fodor, A survey of dimension reduction techniques, center for Applied Scientific Computing,Lawrence Livermore National Laboratory P.O.Box808,L560, Livermore,CA, 2002.

[35] Felderer M, Schieferdecker I. A taxonomy of risk-based testing. International Journal on Software Tools for Technology Transfer(STTT) 2014; 16(5):559–568.

[36] H. Adorf, M. Felderer, M.Varendorff, R.Breu, A Bayesian Prediction Model for Risk-Based Test Selection.

[37] Deborah J. Rumsey (2011). Statistics For Dummies, 2nd Edition. Canada: Wiley Publishing, Inc.. chapter 10.

[38] POWERS, D.M.W. . (2011). EVALUATION: FROM PRECISION, RECALL AND F-MEASURE TO ROC, INFORMEDNESS, MARKEDNESS & CORRELATION . Journal of Machine Learning Technologies . 2 (Issue 1), p37-63.