Automatic analysis for continuous integration test failures

Xuhui Zhao

2019
Abstract

CI (Continuous Integration) is a software development practice which became more and more popular in last decade. Ericsson followed the trends and used CI several years. Because of the complexity of RBS (Radio Base Station) software few levels of CI have been implemented there. In RCS (RBS Control System) module CI there are many automatic JCAT (Java Common Auto Tester) test loops running every day and some of them failed. This thesis tries to find a way to classify these test failures automatically, so efficiency and lead time can be improved.

Two methods are presented and investigated in this report, rule matching and machine learning. After analysis and comparisons rule matching approach is selected because it does not require huge effort in the initial phase and rule matched data can be used as labeled data for machine learning. This approach requires manual work to add new rules continuously but with correctly defined rules the accuracy is 100%, if the rule is general it can classify one type of issue including the ones which never happen before.

One analysis system is designed and implemented, and only small update is required to the result report block of the CI flow. One matching example is showed and according to estimation this method could save many man hours every year.
# Table of contents

1 Introduction .................................................................................................................. 1
  1.1 Background ............................................................................................................. 1
  1.2 Aim .......................................................................................................................... 2
  1.3 Specific goals ......................................................................................................... 2
  1.4 Used tools .............................................................................................................. 2
  1.5 Thesis outline ........................................................................................................ 3
2 Theory ........................................................................................................................... 4
  2.1 Rule matching ......................................................................................................... 4
  2.2 Machine learning ................................................................................................. 7
3 Process and results ....................................................................................................... 9
  3.1 Process .................................................................................................................... 9
  3.2 Analyze recent failures ......................................................................................... 9
  3.3 Method selection .................................................................................................... 11
  3.4 System implementation .......................................................................................... 12
    3.4.1 Analysis system ............................................................................................... 12
    3.4.2 Update to the CI flows .................................................................................... 17
  3.5 Result ...................................................................................................................... 18
4 Discussion ..................................................................................................................... 20
5 Conclusions .................................................................................................................. 21
6 Future work .................................................................................................................. 22
References ...................................................................................................................... 23
List of figures

Figure 1 Applications of rule(pattern) matching .................................................. 4
Figure 2 Overview of string-matching algorithms .................................................. 5
Figure 3 Rule adding and matching process .......................................................... 6
Figure 4 Overview of Machine Learning Algorithms ............................................. 7
Figure 5 The workflow of the project .................................................................... 9
Figure 6 Architecture of the analysis system ....................................................... 13
Figure 7 Block diagram of log analysis module .................................................... 14
Figure 8 Sequence diagram of the analysis request procedure ............................. 15
Figure 9 Sequence diagram of the analysis result query procedure .................... 15
Figure 10 Rule management page ........................................................................ 16
Figure 11 Overall Technologies used at RCS CI JCAT Test ............................... 17
Figure 12 RCS CI JCAT Test Jenkins Flow Overview ........................................ 18
Figure 13 One rule matching example ................................................................. 19
List of tables

Table 1 Comparisons of rule matching and machine learning approaches .................. 11
List of Abbreviations

A3C       Asynchronous Advantage Actor-Critic Algorithm
BMH       Boyer Moore Horspool algorithm
CI        Continuous Integration
DQN       Deep Q Network
ES        Elasticsearch
G2        Generation 2
JCAT      Java Common Auto Tester
KMP       Knuth Morris Pratt algorithm
LDA       Latent Dirichlet Allocation
LSA       Latent Semantic Analysis
MJE       MSRAN JCAT Extension
MSRBS     Multi Standard Radio Base Station
MSRCS     Multi Standard RBS Control System
PCA       Principal Component Analysis
PV-DBOW   Paragraph Vector with Distributed Bag of Words
PV-DM     Paragraph Vector with Distributed Memory
RBS       Radio Base Station
RCS       RBS Control System
RPC       Remote Procedure Call
SARSA     State–action–reward–state–action
SBC       Source Baseline Check
SBT       Source Baseline Test
SDC       Source Delivery Check
SVD       Singular Value Decomposition
TFIDF     Term Frequency-Inverse Document Frequency
VNF       Virtual Network Function
VNFM      Virtual Network Functions Manager
VRAN      Virtual Radio Access Network
XFT       Cross Function Team
1 Introduction

1.1 Background

Ericsson have implemented CI (Continuous Integration) several years. With this practice developers commit to a shared repository frequently and then automatic build (usually include some unit test) will be triggered, if the build finish successfully it will trigger automatic test. Since each commit only include small change, bugs can be addressed much quicker. In case the changes have to be reverted due to bugs only a small number of changes are lost because of frequent integrations [1]. RBS (Radio Base Station) is one of Ericsson’s most important products, it transmits/receives wireless signal to/from mobile phones. RCS (RBS Control System) is one of its major software blocks, like the operating system of a computer, it manages the resources and provides services to applications. At RCS development projects designers from XFTs (Cross Function Teams) work with new features at their feature branches or local branches. When they are ready they must pass SDC (Source Delivery Check) before they can merge to the main track, then SBC (Source Baseline Check) will be automatically triggered, in case bugs are found during the test their change will be reverted. This kind of test require fast feedback, so only few very important testcases are run and the duration of the test are limited to about one hour. Except the SBC triggered based on commits, there is also SBT (Source Baseline Test) running every night (when there are not many commits so SBT can share the resources with SBC) for extended test scope on the software which have passed the basic test at next integration level, the duration of SBT are much longer, for example about ten hours. All the SBC and SBT test loops require physical or virtual resources. There are limited resources due to cost, if many commits arrive in short period some of them will be queued there waiting for resources, in that case some commit might get feedback after long time. To improve this situation the test for some commits might be aborted manually, then the following test will include more than one commit. In ideal case the CI should be fully automated which means in case of test failures the commit will be automatically blocked, but, due to few different reasons the gating at RCS CI is still manual, few people keep monitoring the test result and block the suspected commit if they think it introduced some issues. Problematic commit might stay at RCS baseline for a while before it is figured out and blocked, but it will not slip to next CI level because the delivery step to next CI level is manual also.
Both SBC and SBT at RCS CI have two different kind of loops, one based on Erlang which contains many unit level testcases and one based on JCAT (Java Common Auto Tester, one of Ericsson’s most preferred Test Automation Frameworks) which mainly contains subsystem level testcases. This project focus on the test failures of the JCAT test loops running at RCS module CI main track. Few engineers analyze the logs of failed test every day and take actions accordingly, for example, retrigger test loop if failure due to test environment issues. One framework could be created to perform analysis and take actions automatically in order to improve efficiency and reduce lead time. The framework must have high extensibility so further improvement can be easily added.

1.2 Aim

The aim of this thesis is to evaluate and propose suitable method which can automat the analysis in some level, so the time consumed every day for analyzing the failure logs can be reduced.

1.3 Specific goals

The objective of the project consists of the following sub-goals:

- Analyze recent RCS JCAT test failures and identify the main categories of the failures.
- Investigate and propose suitable method which can be used for automatic analysis.
- Implement the proposed method.

1.4 Used tools

- Jenkins.
- Pipeline.
- Job DSL.
- Groovy.
- Elasticsearch.
1.5 Thesis outline

Chapter 2 contains the theory related to this thesis, which includes two methods that can be used for automatic analysis, one is rule matching and the other one is machine learning. For rule matching a description, an overview of string-matching algorithms and a process are presented. For machine learning an overview of algorithms with focus on classification is given, furthermore two methods for numeric representation of logs, Bag-of-words and Doc2vec are also described.

Chapter 3 starts with the workflow of the project. Then detailed description of all the steps are given. The first one is to analyze recent failures, four main categories of failures are identified. Method selection is the second step and rule matching approach is selected. The last step is system implementation, a description of the analysis system and required update to the CI flows are given. The result is showed at the end of this chapter and according to estimation it might be able to save more than 789 man hours per year.

Finally, discussion, conclusions, and ideas for future work are given in the last part of this report.
2 Theory

2.1 Rule matching

Rule matching try to check if there is match with some predefined rules. After one test failure has been investigated and the root cause is figured out, one might be able to say that “this is product (or other) issue because there are this or that traces in the logs”, in that case one rule can be setup with these kinds of traces. If there is another failure later with such traces in the logs, the problem can be identified, and the root cause is known. This method can be used to classify the type of failures. Rules can be called as patterns so sometimes rule matching is called pattern matching. Rule(pattern) matching has been used in many kinds of applications, some of them are showed in Fig. 1.

![Diagram showing applications of rule(pattern) matching](image)

One rule or pattern can simply contain only one string. It can also contain few strings cascaded using logical operators such as logical AND (&&), logical OR (||) and logical NOT (!). Regular expression can also be used to define the rules or patterns. Some applications might use some more complicated patterns such as trees and arrays, or even graphs and point sets [2].
Rule matching uses string-matching algorithms, sometimes they are called string-searching algorithms. There are quite many different algorithms and typically can be divided to two types, exact matching and approximate matching. Figure 2 shows the overview of string-matching algorithms. For exact string-matching popular algorithms are Needleman Wunsch algorithm, Smith Waterman algorithm, KMP (Knuth Morris Pratt algorithm), Dynamic Programming algorithm and BMH (Boyer Moore Horspool algorithm). Fuzzy string searching algorithm, Rabin Karp algorithm and Brute Force algorithm are widely used approximate string-matching algorithms [2,3].

![Figure 2 Overview of string-matching algorithms](image)

There are many search engines online now, like google, bing, baidu etc. The modern search engines usually combine few different search algorithms and use different one for different types of data (text, image, audio, video) [3]. Some of search engines are open source and Elasticsearch is one of the most powerful open source search engines.
Rule matching use prior knowledge of the system, if only very specific rules are added it cannot predict the problem which did not happen before. In case some general rule can be identified it can classify one type of problem which even include some issues that never happened before. For example, when there is “Check Failed: NO New PMD Entries” in the logs it means there was crash on the node, then the failure can be classified as product fault even though it is the first time that this kind of crash happened.

If rules are added when new issues are investigated as shown in Fig. 3 a knowledge base will be built up after some time. It is probably the easiest way for automatic log analysis, and if the rules are added correctly it will be 100% correct for known issues and issues of specific type.

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{figure3.png}
\caption{Rule adding and matching process}
\end{figure}
2.2 Machine learning

Machine Learning predict results based on data. It typically can be divided into three main categories, supervised learning, unsupervised learning and reinforcement learning. Figure 4 shows the overview of the algorithms.

Supervised learning learns by examples. It uses labeled data consisting of a set of samples each include an input object and an output. Usually the data is split into two parts, the training dataset and the validation dataset. The model is trained using the training dataset and then validated using the validation dataset. There are mainly two types of supervised learning, classification predicts the category of an object and regression predicts the specific value. For classification there are quite many algorithms such as Logistic Regression, Naive Bayes, Decision Tree, K-Nearest Neighbors, Support Vector Machine, Random Forest, and Neural Network [4-6].

Many machine learning algorithms need large amount of data to achieve expected percentage of accuracy. In many cases it is not easy or even impossible to fetch enough amount of data. Supervised learning even requires the data to be labeled, which often consumes lots of time or money.
Most of text classification and clustering algorithms require the text input to be represented as a fixed-length vector. Bag-of-words and Doc2vec are two well-known and widely used methods. Bag-of-words simply represent a text as the bag of its words, the ordering of the words is ignored. To be able to use Bag-of-words model a vocabulary is needed, it can be generated from the collection of texts or documents. A good representation of English text typically has at least four or five thousand vocabularies. Except the vocabulary a measure is needed to extract features from the text and the most popular one is TFIDF (Term Frequency-Inverse Document Frequency) [7]. Bag-of-words has few disadvantages. The ordering and semantics of the words are lost. Doc2vec method overcome these weaknesses. Doc2vec method heavily based on a well-known concept word2vec, which takes a large corpus as input and produces a numeric representation of each word, typically of several hundred dimensions. Using this method words with similar meaning are located close to each other in the vector space. Word2vec uses either continuous bag-of-words or continuous skip-gram to produce word embeddings. The continuous bag-of-words model use a simple neural network with a single hidden layer to predict the current word from the surrounding words. But in the end, the neural network trained is not used, instead, the weights of the hidden layer are the things used as the word vectors. The continuous skip-gram model does the inverse and predicts the surrounding words from the current word. Similar as word2vec, Doc2vec also has two models. Paragraph Vector with Distributed Memory (PV-DM) just add a paragraph id to the continuous bag-of-words model. There is another algorithm similar to continuous skip-gram, which is Paragraph Vector with Distributed Bag of Words (PV-DBOW). It is recommended to use both algorithms together even though PV-DM alone usually works quite well for many tasks [8-11].
3 Process and results

3.1 Process

Figure 5 shows the workflow of the project. The first step is to analyze the recent test failures and then identify the main categories of the failures. The second step is to investigate available methods and select a suitable one for automatic analysis. Then the last one is to implement the selected method.

![Figure 5 The workflow of the project](image)

3.2 Analyze recent failures

The perfect CI test flow only fails when the committed change introduces new issues, but in reality, there are other issues except product fault, such as test environment issues and test case issues.
When analyzing the failed JCAT logs, the first thing which need to be identified is if it is a product fault or not, if it is not a product fault is it test environment issue or test code issue. According to recent analysis, the main category of the faults is either test environment or product faults. From later 2018 to earlier 2019 when the Jenkins was migrated to new Jenkins server and architecture there were quite a lot environment related issue. Recently when the new Jenkins is getting stable the majority of failures are caused by product related faults.

Typically, there are below types of failures:

- **Product faults:**
  The committed change might have defects which cause some testcases in SBC to fail, in that case the commit should be reverted. In SBT there are also failures caused by product faults, such as crashes occurred occasionally caused by product defects which have already been delivered to the baseline for some time, in that case a trouble report should be written.

- **Test environment issues:**
  There are quite many kind of test environment issues, for example, sftp server temporarily unavailable, Jenkins server or slave crash due to lack of memory or some other issues, no available nodes to use for few hours. In case of test environment issue the test might need to be rerun to make sure the commit does not introduce any issues.

- **Test code issues:**
  The test code use MJE (MSRAN JCAT Extension) which have new releases quite often because of new features or fault corrections. Time to time the MJE version used need to be uplifted, mainly due to some fault found. There are also defects in the test code, such as dead loop happened in some specific situation.

- **Other unknown issues:**
  There are also issues occur very rarely and not able to be reproduced locally, in that case it might be impossible to know if it is product fault or not. When the issue appears again later with some useful traces or information the root cause might be able to be identified.
In case there is test environment related issue, usually there will be few failures and the failures will continue until the issue is noticed and fixed. In case of product fault, the failures will also continue happening until the issue is figured out and the faulty commit is blocked. There are not many test code related issues but when there is, it usually happens very rarely and is difficult to reproduce, for quite many cases it will fall into unknown category at beginning and might be identified later when some useful information is found.

3.3 Method selection

Both rule matching and machine learning classification algorithms can be used to identify the category of the failure. Table 1 shows the comparisons of these two approaches.

<table>
<thead>
<tr>
<th>Method</th>
<th>Percentage of failures can be classified</th>
<th>Accuracy</th>
<th>Effort needed at initial phase</th>
<th>Manual work needed continuously</th>
<th>Other advantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule matching approach</td>
<td>&lt; 100</td>
<td>100</td>
<td>acceptable</td>
<td>yes</td>
<td>Matched data can be used as labeled data for machine learning</td>
</tr>
<tr>
<td>Machine learning approach</td>
<td>100</td>
<td>&lt; 100</td>
<td>too much</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>

Rule matching is a simple method and easy to implement. With correct rules defined the accuracy is 100% but there might be failures without any matches. It doesn’t require too much effort at the initial phase, but in order to increase the matching rate manual work is needed continuously to add new rules. Rule matching approach also have one advantage which is rule matched data can be used as labeled data for machine learning usage.

Machine learning approach doesn’t require continuous manual work, but at beginning much more effort are needed. To get good accuracy for supervised learning typically the number of samples should be 10 times more than the number of features. To represent English text Bag-of-words usually have more than 4000 dimensions, with doc2vec the dimensions can be decreased to few hundreds, so it means at least few thousands of samples need to be collected and labeled, which is a huge task. Normally the accuracy of machine learning approach is below 100%. 

Table 1 Comparisons of rule matching and machine learning approaches
Rule matching method is selected because machine learning approach request huge effort at initial phase to label data and rule matched data can be used as labeled data for machine learning to use.

3.4 System implementation

3.4.1 Analysis system

3.4.1.1 Overview

The system is designed to be independent to the CI flows and only little update is needed to the CI loops. When the system receives one request with the log URL to analyze, it will answer with the URL to check the analysis result. This concept allows other teams to use the analysis service as it is easy to be implemented for other projects.

Figure 6 shows the architecture of the system, it includes four layers:

- **WEB/External API**
  
  This layer provides the user interface to external system.

- **Service layer**
  
  The service layer includes three log analysis modules, the result analyzer and the RPC (Remote Procedure Call) block.

- **Storage layer**

  Storage layer has three blocks: ES(Elasticsearch), file system and H2 database.

- **Platform**

  The lowest layer is the platform, Docker and Kubernetes are used.
The three log analysis modules are the key components, they have two layers: function analysis layer and data handler layer. Figure 7 shows the block diagram and how it connects and communicates with storage and the result analyzer. Now only the Jenkins log analysis module is implemented since Jenkins logs are the most important ones and it include the JCAT console log which also print many useful node logs.
3.4.1.2 Main procedures

There are two important procedures, the analysis request procedure and the analysis result query procedure.

The sequence diagram of the analysis request procedure is showed in Fig. 8. When the Web/External API receive a request with a log URL, it will send a request to the analysis module with the log URL and get a response back with task id included, then the Web/External API will reply the external system the URL to check the analysis result, at same time the analysis module try to fetch the logs from external storage, after pre-analysis it will save the logs to ES/File system using predefined schema. Analysis module will check with ES/File system if there is any match with defined rules and then perform the analysis, the analysis result will be sent that to result analyzer, in case any action such as retriggering the flow need to be performed an RPC request will be sent and these information will be returned back to the analysis domain to save to the ES/File system.
Since the rules are updated time to time so the analysis result should show the matching result with the latest rules. Figure 9 shows the sequence diagram of the analysis result query procedure. When the analysis result URL is clicked or refreshed a request with the task id will be sent to the analysis module, analysis module will check with ES/File system if there is match according to latest rules and then perform analysis, the result will be sent back to the GUI and showed to the user.
3.4.1.3 Rule management

Rules are saved to H2 database. It can be added, deleted or updated through the “add”, “delete” and “update” buttons on one webpage, Figure 10 shows the snapshot taken at 26th April 2019.

**Figure 10 Rule management page**
3.4.2 Update to the CI flows

Jenkins is used as the CI framework and Pipeline is the key plugin of the Jenkins. Figure 11 shows the overall technologies used at RCS CI JCAT test. Job DSL is used to keep track of the job configuration changes at the repository test-rcs-flow. Both Job DSL and Pipeline script are using Groovy. Flow service is used to reserve nodes and provide configuration files to Jenkins jobs, it is built in Micro Service concept and using SpringBoot as framework. All the data transferred between jobs and service are in JSON format. Configuration files are in YAML format which are more readable, and Velocity is used as templating engine to replace the variables in it.

![Figure 11: Overall Technologies Used at RCS CI JCAT Test](image)

Figure 12 shows the structure of the Jenkins flow. Flow-Portal gets the triggers and then start specific flows according to the test activity received. Then ENV-Setup job will be called to setup the test environment, for MSRCS (Multi Standard RBS Control System) flows few physical MSRBS nodes will be reserved and initialized, for VRAN (Virtual Radio Access Network) flows VNFM (Virtual Network Functions Manager) or VNFM and VNFs (Virtual Network Function) will be instantiated. When test environment is setup successfully Test-Execution job will be called, specific test suites will be executed on the test environment. After that the Env-TearDown job will be called, for MSRCS flows the MSRBS nodes will be rollbacked to the initial state and unreserved, for VRAN flows the VNFM and VNFs will be deleted and all the cloud resource will be released. In case of failures ESI-collect job will collect ESI logs for further analysis. The last job is Result-Report.
which will record the links to the logs and calculate the success rate of flow. The result will be showed on the RCS CI radiator pages.

**Figure 12 RCS CI JCAT Test Jenkins Flow Overview**

Because of the independent design of the analysis system, there are not much update required to the Jenkins flows. The only block which need to be changed is the result-report block. When there are test failures the log URL will be sent to the analysis system and one response with the analysis result link will be returned, that link will be added to the test result database and showed in the detailed result pages which can be accessed from the radiator.

### 3.5 Result

When the analysis system is running and the update to the Jenkins flows have finished, all the test failures will have one link which can show if there is any match with current defined rules. Figure 13 shows one example with one rule matched and the failure can be identified as test code issue.
During the trial period 13 rules had been added and 7.2% of failures were matched. This percentage will increase if more rules are added later. Last year there were 1096 tickets to analyze the test failures and on average around 10 hours were spent per ticket. This means it might be possible to save more than $1096 \times 10 \times 7.2\% = 789.12$ man hours per year which is really good.
4 Discussion

Data is very crucial to many machine learning algorithms. No algorithm can get expected result without right amount of data or if the data is not diverse enough. Natural language processing task typically require more data because of the high dimension and sparsity of data after numerical representation of text. Using bag-of-words method the representation usually has few thousand dimensions. With doc2vec it can be decreased to few hundreds, then to get good percentage of accuracy with a classification algorithm it will be good to have at least thousands of labeled data, which is a huge task.

With rule matching approach new rules need to be added continuously to achieve high matching rate. One knowledge base can be formed if rules are added after analyzing all the problems, with that it will be very useful and save lots of time for daily analysis. Rule matching is hundred percent correct if correct rules are added, general rules can match one type of problem including the issues which have never happened before. Rule matched data can be used as labeled data for machine learning usage, when there are many rules defined machine learning can use matched data as labeled data to predict the issues without any match.

Lots of things need to be considered when retriggering the Jenkins jobs automatically. Such as the usage of physical nodes and the cloud resources, the load of Jenkins servers and slaves. The RPC call for automatically retriggering jobs have been verified successfully but without such information in place the jobs cannot be retriggered automatically.

Together with continuous integration, automatic test is implemented widely in recent years, this thesis tries to move the automation one step further, into the test analysis domain. This work compliance with the target 9.5, goal 9 of the United Nations’ 2030 Agenda for Sustainable Development [12]. The idea and method used might inspire others working in similar fields.
5 Conclusions

In this thesis recent RCS JCAT test failures were analyzed. Product fault, test environment issue and test code issue are identified as the main categories of the failures, unknown fault category is used when the root cause of failure is not able to be figured out when lack of information, they might fall into the other categories later when the fault appears again with more information fetched.

Rule matching and machine learning approaches for automatic analysis are investigated. Machine learning approach is not used because huge effort is required at the initial phase to label the data and in case rule matching is selected rule matched data can be used as labeled data for machine learning. Rule matching approach is selected and implemented, the result shows that it could save some time every day for analyzing logs.
6 Future work

Currently there is another ongoing project which will try to monitor the usage of all resources (physical nodes, cloud resources, load of Jenkins server and slaves), when it is ready it can be used to decide if the failure jobs caused by test environment issues can be retriggere automatically or not.

Later when many rules are defined, the matched data can be used as labeled data for machine learning classification algorithms to predict the problems without any match.
References


