Bachelor Thesis
180 credits

Evaluation of a prioritization algorithm for test suite generation

Utvärdering av en prioriterings algoritm för test-svitsgenerering

Adnan Begovic
Viktor Holmelin
Abstract

Software is created to solve some defined problem. In this process incremental steps are usually taken towards the complete product. Once a piece of software has been written or changed, it has to be verified that the software is still functional and performs as expected. This verification is usually done through regression testing. As the number of tests to run in a regression test suite increase, the longer time it takes to execute. Possibly undesirable long time for the developers of the software. Not all regression tests interact with the new or changed piece of software and are thus less useful from a testing perspective. If it is possible to know how to arrange the order of regression tests so that the tests interacting with the new software come first in the test suite, then the expected time until feedback to the developer that something is broken can be reduced. This reduction in time makes for a smoother work flow for the developer.

This thesis investigates a method for predicting which tests to run first given the changes made to the software.

The underlying idea of the method is to use historical test data and changes made to software in order to compute a correlation between tests and software. This idea is tested in a hypothesis test to determine if it has any predictive power. This choice of method is made as it is non-intrusive and does not requiring any instrumentation of the software but only knowledge of the historical test data.

Possible alternative methods are limited to what kind of data there exists and determined through data exploration. Methods for measuring performance and quality is determined through a combination of a literature survey and the desires of the stakeholder.

The result indicates that this is a feasible method.

Keywords: Regression Test Prioritization, Average Precentage Fault Detection (APFD).
Acknowledgments

We would like to thank our supervisor Blerim Emruli, and our teacher Magnus Krampell for giving us valuable feedback throughout the work of this thesis. We would also like to express our gratitude to Axis Communications AB for giving us the opportunity to conduct our thesis at their amazing office; it has been a very informative and exciting journey.
Contents

1 Introduction ........................................... 1
   1.1 Background ........................................ 3
   1.2 Research aim and research questions .............. 5
      1.2.1 Research aim .................................. 5
      1.2.2 Research question .............................. 6
   1.3 Thesis scope and limiting factors ................. 6

2 Theoretical background .................................. 7
   2.1 Convention on classification ....................... 7
   2.2 Summary statistics of classifications ............ 7
      2.2.1 Precision and recall ........................... 8
      2.2.2 Accuracy ....................................... 8
      2.2.3 F-measure ...................................... 8
      2.2.4 Matthews correlation coefficient (MCC) ...... 9
   2.3 Average percentage of fault detection ............ 9
      2.3.1 Example on shortcoming of APFD .............. 10
   2.4 Random variables .................................. 11
   2.5 Boxplot ............................................ 12
   2.6 Hypothesis testing ................................ 12

3 Related work .......................................... 16
   3.1 Test Selection Based on Historical Test Data .... 16
   3.2 Automated System Level Regression Test Prioritization Using Multiple Factors ........................................... 17

4 Method .................................................. 18
   4.1 Controlled experiment ............................. 18
      4.1.1 Theory ........................................... 18
      4.1.2 Hypothesis ...................................... 18
      4.1.3 Model and samples .............................. 18
      4.1.4 Experiment ..................................... 19
      4.1.5 Hypothesis testing ............................. 19
   4.2 System development ................................ 19
      4.2.1 Construct a conceptual framework ........... 20
      4.2.2 Develop a system architecture ............... 20
      4.2.3 Analyze and design the system ............... 21
      4.2.4 Build the prototype system ................... 21
      4.2.5 Observe and evaluate the system ............. 21
5 Result

5.1 Controlled experiment ........................................... 22
   5.1.1 Theory ....................................................... 22
   5.1.2 Hypothesis .................................................... 26
   5.1.3 Model and samples ........................................... 26
   5.1.4 Experiment .................................................... 27
   5.1.5 Hypothesis testing .......................................... 28

5.2 System development .................................................. 31
   5.2.1 Construct a conceptual framework ...................... 31
   5.2.2 Develop a system architecture .......................... 33
   5.2.3 Analyze and design the system ......................... 34
   5.2.4 Build the prototype system ............................... 34
   5.2.5 Observe and evaluate the system ....................... 34

6 Discussion .............................................................. 36
   6.1 Result .......................................................... 36
   6.2 Threats to validity ............................................ 37

7 Conclusion and future work .......................................... 38
   7.1 Answering the research questions ......................... 38
   7.2 Contribution ..................................................... 38
   7.3 Future work ...................................................... 39

References ............................................................... 40

Appendices ............................................................. 42

A Full confusion matrix .............................................. 42
1 Introduction

Software development has taken a direction towards a more Agile method, where the continuous cycle of writing and testing software is an integral part of the development process to better handle changing requirements. This cycle of writing and testing software can be seen as having constant feedback through regression tests.

The process of re-running tests for previously developed and tested software is called regression testing. Regression testing is performed whenever any changes have been made to the software in order to find bugs which might have been introduced, or manifested itself, with the changes to the code[1].

A classical problem within regression testing is that test suites tend to grow in size as software evolves, which would eventually lead to expensive test suite executions[2]. This problem becomes even more obvious in agile software development because of the quick software deliveries.

The agile test pyramid is often used to convey how expensive it could be to write and execute regression tests at different test levels. Typical test levels such as unit, integration, and system levels are presented in Figure 1.

![Agile test pyramid](image)

**Figure 1:** Agile test pyramid

The agile test pyramid shows e.g. that the number of automated unit tests is much larger and faster to execute, compared to the automated functional tests at system level[3].

Automated regression testing can consist of re-running all, or only a selection of regression
tests in a given test suite. Re-running all regression tests for small development projects might not be much of an issue as the feedback from the tests is prompt. However, as the code base grows the duration of time until the result of the test suite is delivered can grow into hours. This is not desirable from the perspective of developers as this delay creates a jerkiness and reduced productivity[4].

*Test suite minimization*, *Test case selection*, and *Test case prioritization* are three different approaches, that could be applied and combined in order to generate a more efficient test suite. Test suite minimization: removes all the redundant tests in the test suite. Test case selection: identifies the tests that are relevant to some set of recent changes. Test case prioritization: re-orders the tests to maximize early fault detection in the test suite [2].

The outline of this thesis is as follow. In 1.1 Background the problem is given a framing. Following in section 1.2 Research question the aim of the thesis is presented. Section 2 Theory provide a brief theoretical background and is followed by 3 Related work presenting closely related work. In 4 Method a presentation to how the work of the thesis was undertaken is explained. Section 5 Result provide the result which are then discussed further in 6 Discussion. Finally in 7 Conclusion and future work is presented.
1.1 Background

Axis Communication AB\(^1\) runs automated test suites several times a day and at night to verify that their software systems still works as expected.

One day a group of test automation engineers realized that their automated test suites of functional tests had grown in size. The duration of running a full test suite was now too expensive to perform during the day since the feedback from the tests was delivered after several hours. In order to provide more rapid feedback to the developers, the number of executed tests had to be reduced. The main concern with this approach was the lack of data on how much of an impact a reduction in executed tests would have on the quality of the software. The first approach to ensure the quality of the software was to run the entire test suite during the night while a subset of relevant tests was run during the day. At this point, the selection and prioritization of the tests were done manually based on testers knowledge and experiences of the system under test. However, they wanted to implement an even smarter and faster way of selecting and prioritizing the tests by simply automating the process.

A prioritization algorithm was eventually developed to maximize early fault detection in the suites. The prioritization algorithm is called Suite Builder, and it uses historical regression test data such as test time duration and failure rate to generate a more efficient order of functional tests before the test suite execution. More specifically, an overview of the implementation is presented graphically in Figure 2.

![Figure 2: Overview of the current prioritization algorithm](image)

**Collector** finds and retrieves all the information about regression tests that have been previously executed. The retrieved information from the historical regression test data are test time duration and failure rate for each test.

**Suite creator** uses the retrieved information from the Collector to re-create test suites from the previous run, by filtering the suite on some specific factor like for example failure

\(^1\)Axis Communications AB is a Swedish IT-company with expertise in network video surveillance and security solutions.\([5]\)
rate. Before the suites are re-created, the user needs to select the importance of each factor by assigning a priority weight. The value of the priority weight can be selected in a range between $1 - 10^{p_w}$, where $1^{p_w}$ means low and $10^{p_w}$ means high importance.

After the suites have been re-created by the Suite Creator, the individual prioritizers will give each test a weight equidistantly spaced in the range $0 - 100^{w}$. The first test in suite will have a max weight i.e. $100^{w}$, and the following tests will get weights assigned according to the following formula: $\text{weightPerTest} = \maxWeight - \text{roundValue}\left(\frac{\text{currentWeight}}{\text{amountOfTests}}\right)$. This means that if the $\text{amountsOfTests}$ is greater than the $\text{currentWeight}$, then multiple tests will have the same weights.

- **Failure rate prioritizer** assigns weights based on the highest failure rate of tests in the suite.
- **Time sorted prioritizer** assigns weights based on the shortest duration of tests in the suite.
- **Static list prioritizer** assigns weights based on the static suite ordering\(^2\) of tests.

**Priority merger** calculates the *weighted average* of all the tests that have been assigned a weight by the individual prioritizers.

---

\(^2\)no factor filtering
The weighted average of e.g. TC3 in Table 1 and Figure 3 is calculated by the following formula:

\[
\frac{(98_{pw} - 1_{pw}) + (50_{pw} - 10_{pw}) + (90_{pw} - 1_{pw})}{(1 + 1 + 10)_{pw}} = 57
\]

**Table 1: Priority merger**

<table>
<thead>
<tr>
<th>pw</th>
<th>TC1w</th>
<th>TC2w</th>
<th>TC3w</th>
<th>TC4w</th>
<th>TC5w</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static list prioritizer</td>
<td>1</td>
<td>100</td>
<td>99</td>
<td>98</td>
<td>97</td>
</tr>
<tr>
<td>Failure rate prioritizer</td>
<td>10</td>
<td>1</td>
<td>100</td>
<td>50</td>
<td>-</td>
</tr>
<tr>
<td>Time sorted prioritizer</td>
<td>1</td>
<td>-</td>
<td>75</td>
<td>90</td>
<td>65</td>
</tr>
<tr>
<td>Weighted average</td>
<td>10</td>
<td>98</td>
<td>57</td>
<td>81</td>
<td>74</td>
</tr>
</tbody>
</table>

Axis has managed to developed a simple prioritization algorithm which seems to work, but the performance of the current prioritization algorithm needs to be evaluated and improved if possible. Their goal is now to implement an alternative method for prioritizing a test suite and compare it with the current prioritization algorithm. A secondary goal would be to design and evaluate new algorithms for test selection.

### 1.2 Research aim and research questions

#### 1.2.1 Research aim

The problem experienced at Axis was presented in section 1.1. There has to be some factual basis on which to evaluate a decision if an prioritization algorithm has performed any good. One such measure is the average percentage of faults detected (APFD) [6]. But the usefulness of a measure is very dependent on the situation. Our aim with this thesis is to provide, for the stakeholder³, a suitable measurement metric. In addition we test the viability for using computed correlation between tests and software as an strategy for ordering a test suite given which software has changed.

This thesis aims to provide a better understanding of the performance and quality of regression test suites while maintaining a balance between objectiveness and the need of the stakeholder.

To make a statement that two orderings of tests give the same APFD a hypothesis testing scenario can be used. As is convention the null hypothesis is that the two orderings give the same APFD. This null hypothesis is then rejected given a predetermined significance if the average APFD of the two orderings are different enough. Which in turn will give credence to the method of ordering tests in a test suite.

³Axis Communications AB
1.2.2 Research question

The research question was formulated to both take a step closer to achieving the research aim but also to limit the scope to a reasonable width. The question is closely related to the null hypothesis

**RQ** Does the computed correlation between tests and software using historical data provide the ability to predict tests of interest given changes to software?

The research question is further broken down into more manageable questions.

**Q1** How do we measure the performance of an test suite prioritization algorithm?

**Q2** How do we determine if there is a measurable effect of the test suite prioritization algorithm?

**Q3** How do we implement the calculation of the correlation?

The combined answer to the questions will provide further insight as well as having practical importance in indicating the quality of a test suite.

1.3 Thesis scope and limiting factors

Due to the time duration it takes to run an entire test suite and the sparsity of failures within the regression tests, the evaluation is done on historical data. It is assumed that the regression test suite was built with the intention to find the failures as quickly as possible. Furthermore, it is also expected that the tests run did find the faults if they were present. This will give a reasonable efficiency score to beat as well as a completeness of the test cases.
2 Theoretical background

This section gives an overview of the theoretical background and also an explanation on the conventions used.

2.1 Convention on classification

This section provides definitions and explanation of the classifications of test results in a test suite. With classification it is meant if a test is classified as will fail or not. The label “positive” on a test is used to indicate a test which failed.

- **True positive**: Test outcome is a failure, and it is predicted to be a failure.
- **False positive**: Test outcome is not a failure, but it is predicted to be a failure.
- **False negative**: Test outcome is a failure, but it is predicted to not be a failure.
- **True negative**: Test outcome is not a failure, and it is predicted to not be a failure.

These classifications are usually arranged in a grid form as in Figure 4 called a confusion matrix with the count of each classification as the value of the cells.

![Confusion Matrix](image)

**Figure 4:** Arrangement of predicted and actual outcome, and the names given to the specific combinations. This is the convention used for the confusion matrix in this thesis.

2.2 Summary statistics of classifications

From the numbers in the confusion matrix different summary statistics can be compute, each of them capture different aspects of the data. However, the usefulness of each of them depend on the situation and the desire of the end user. One very important thing to note is the possibility for imbalance in the data, i.e., when one of the groups contain many more instances than the other. This imbalance can lead to one summary statistics indicating a very good result while it may be a very bad result using a different summary statistics.
For example, assume the number of true negative far outnumber the other classes, i.e. most tests do not fail. Making a prediction that all tests will pass will result in a high accuracy while the recall will be really low. The explanation to why this is so follow in the section below.

2.2.1 Precision and recall

Precision summarize how many of those classified as positive indeed were positive, i.e., the relative numbers of the cells in the first row of the confusion matrix.

\[
\text{Precision} = \frac{TP}{TP + FP} \tag{1}
\]

Recall summarize how many of the actual positive were classified correctly, i.e., the relative numbers of the cells in the first column of the confusion matrix.

\[
\text{Recall} = \frac{TP}{TP + FN} \tag{2}
\]

2.2.2 Accuracy

Accuracy summarizes how many were correctly classified, i.e., the relative numbers of the cells on the diagonal to the total number in the confusion matrix.

\[
\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}
\]

As stated previously, accuracy is susceptible to imbalanced data in particular when many of the instances are actually negative. For example, if there are 1000 instances of which 100 are actually positive and classifying every instance as negative gives an accuracy of

\[
\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} = \frac{0+900}{0+0+100+900} = \frac{900}{1000} = 0.9 \quad (1.0 \text{ is maximum possible accuracy})
\]

On the other hand, recall will be 0 and precision will be undefined. Good accuracy but rather useless classifier as it makes no prediction on which tests will fail.

2.2.3 F-measure

F-measure is the harmonic mean of precision and recall
\[ F\text{-}measure = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \]  

(4)

It is a more complete summary than precision and recall, yet it does not account for the number of true negative.

2.2.4 Matthews correlation coefficient (MCC)

Matthews correlation coefficient [7] use all the information in the confusion matrix to compute the summary statistics making it less susceptible to imbalanced data.

\[ MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}} \]  

(5)

One thing to note is that MCC gives a value in the range \([-1, 1]\) whereas the other presented summary statistics give a value in the range \([0, 1]\).

2.3 Average percentage of fault detection

Average percentage of fault detection (APFD) [8] is a way of measuring how well the ordering of tests has been in a test suite.

The measure of the APFD is the area under the curve as shown by the blue area in Figure 5. On the horizontal axis is the normalized progress through the test suite consisting of seven tests in total. As an example, two of the tests will fail, and this is indicated by a red dot, as opposed to a green dot for a passing test. On the vertical axis is the normalized number of failing tests.
Figure 5: Graphical presentation of the computation of the average percentage of fault detection as the area under the curve. On the horizontal axis is the normalized test index, on the vertical axis is the normalized failing test index. Red dot indicate failing tests and green passing tests.

The APFD could be thought of as the cumulative distribution function of faults in the test suite. The tests in a test suite could either pass or fail and the APFD measures how early on in the test suite the failing tests are ordered. In Figure 6 a few different orderings are shown ordered worse to better from left to right. The better the ordering the closer the APFD will be to 1.0. If the APFD value is 0.5, it is no better then a randomized test (equal precentage for a failing test to occur at any place in the test suite).

Figure 6: Depiction of the same test suite but with different orderings. The better the ordering is, the earlier on the failing tests are ordered.

One shortcoming of the APFD is if a selection is made and a comparison is made between suites of a different number of tests. An example is provided in the subsection that follow.

2.3.1 Example on shortcoming of APFD

For example, a selection is made on the same set as in Figure 5 to only include the first five tests of the suite but the ordering is the same. In Figure 7 this is depicted. The APFD of the test suite with only five tests would be worse than a random order, yet it would finish before the test suite including all seven tests and detect all the regressions as well.
2.4 Random variables

As an example the percentage to get the number 3 on an ordinary six sided die is $1/6$. Or, the percentage that the random variable $x$ is 3 is $1/6$. A bit more precise is to say that the discrete random variable $x$ is uniformly distributed with percentage mass function $1/6$.

Another example is a coin flip. Let the coin be loaded so the percentage to land heads up, $P(\text{head}) = p$, is different from tails up. This is a Bernoulli distribution. If a number of identical coins are flipped, the number of heads up will follow a binomial distribution\cite{9}. That is $P(\#\text{head} = k) = \binom{n}{k}p^k(1-p)^{n-k}$.

For continuous random variables it is similar. For example let $x$ be uniformly distributed over the interval $[0, 1]$, the percentage that $x$ will be in the interval $0.5$ to $0.6$ is $0.1$. For any interval $a$ to $b$ the percentage will be $b - a$. If $x$ is uniformly distributed over a given interval $[C, D]$, $x$ is said to have a percentage density function given by $1/(D - C)$ for $x \in [C, D]$ and 0 otherwise. The percentage that $x$ will be in an interval $[a, b]$ is given by the integral of the percentage density function over the given interval. If $[a, b] \subseteq [C, D]$, the percentage that $x$ will be in the interval is $\int_a^b \frac{1}{D-C} \, dx = \frac{b-a}{D-C}$.

Instead of being uniformly distributed, the density function could look differently. One such common function is $f(x) = \frac{1}{\sqrt{2\pi}\sigma^2}e^{-\frac{1}{2}(x-\mu)^2/\sigma^2}$ called the Normal distribution\cite{9}, (or bell curve, or Gaussian distribution). The parameter $\mu$ gives the central location of the curve, and $\sigma$ the width. The notation to indicate that a random variable follow a Normal distribution is $n \sim N(\mu, \sigma^2)$. This function appear in many places. One such particular instance is in the binomial distribution when $n$ grows large. This is established by the central limit theorem (CLT).
2.5 Boxplot

A boxplot [10] is one way to graphically summarize a dataset. Figure 8 show on top a histogram of 100 samples from a normal distribution with mean 0 and standard deviation 1. The vertical line inside the box indicates the median value, which in this case is 0.08 (sample mean is −0.02). The box extends from the lower to the upper quartile of the data, i.e., the two middle quartiles of all data points (sample standard deviation is 0.97). The whiskers extend to the data point less than 1.5 times the width of the box outside the box. The value 1.5 is chosen by convention. If it were set to infinity, the whiskers would indicate the minimum and maximum data point. If there are any points outside the whiskers, they show up as circles.

Figure 8: Histogram and boxplot of 100 samples from a normal distribution with mean 0 and standard deviation 1.

2.6 Hypothesis testing

A test statistic is a measure of a statistical property [11] [12]. Continuing the example above about coin throws an example of test statistic could be the number of observed heads in a number of throws. The models of the experiment is that the number of observed heads would follow a binomial distribution. It is usually assumed that a coin throw will result in tail with the same percentage that it will result in a tail. This assumption could be stated as a null hypothesis[13] ”The expected percentage for our coin to land heads up is 0.5”. As alternative hypothesis we could take that the expected percentage would be greater than 0.5. The test statistic would then be the average number of heads observed, the sample mean. We would like to be able to reject the null hypothesis by say 95% confidence, or the percentage of the outcome of our experiment would only be 5% is the null hypothesis were correct. Assume that when we perform out experiment of flipping the coin 1000 times, we observe it to land heads up 550 times. Our gut feeling would be that the coin is not fair. But that is not testing the hypothesis. To test the hypothesis we compute the percentage
for this event using the model, so \( P(k \geq 550|H_0) \approx 0.0007 \). So we can reject the null hypothesis\([14]\).

The outcome of the hypothesis test can lead us to make two types of errors,

*Type 1 error*: Reject \( H_0 \) when it is actually true.

\[
\alpha = P(\text{Type 1 error}) = P(\text{Reject } H_0 | H_0 \text{ is true}) \tag{6}
\]

This is controlled by the significance level that we chose for our test.

*Type 2 error*: Accept \( H_0 \) when an alternate hypothesis \( H_A \) is true.

\[
\beta = P(\text{Type 2 error}) = P(\text{Accept } H_0 | H_A \text{ is true}) \tag{7}
\]

This is generally phrased through the power \( \pi_T \) of the test statistic \( T \), which is defined as the complement of the percentage of the type 2 error

\[
\pi_T = 1 - \beta = P(\text{Reject } H_0 | H_A \text{ is true}) \tag{8}
\]

The parameters of the power are the significance level \( \alpha \), the alternative hypothesis \( H_A \), and since we use the asymptotics of the CLT in the test statistic \( T \), it also depends on number of elements \( n \) used for the mean. So we write

\[
\pi_T(\alpha, H_A, n) \equiv P(\text{Reject } H_0 \text{ at level } \alpha | H_A \text{ is true, size of data is } n) \tag{9}
\]

As part of the methodology we should assess the ability of the hypothesis test to reject the null hypothesis \( H_0 \) if the alternative hypothesis \( H_A \) is true. The power of the test expresses this percentage in terms of the parameters of the alternative hypothesis and the size of the data gathered.

We will therefore graph the power of test as a function of the alternative parameters and data size, to find in which regions the test is useful. If we calculate other test statistics we can compare their power and use the most appropriate test in that region.

The null hypothesis \( H_0 \) is that the APFD \( S \) of our method has the same mean and variance as the APFD \( S_R \) of the random model. If we obtained the estimates of \( E(S_R) \) and \( V(S_R) \) by simulating the APFD, \( H_0 \) states that \( E(S) = E(S_R) = \mu_0 \) and \( V(S) = V(S_R) = \sigma^2_0 \). Thus under \( H_0 \) we consider the statistic
\[ T = \frac{\bar{S} - \mu_0}{\sigma_0 / \sqrt{n}} \]  

(10)

where \( \bar{S} \) is the mean of the APFD of our model and \( n \) is the number of APFD samples. By the CLT under \( H_0 \) we have

\[ T \xrightarrow{D} N(0, 1) \]  

(11)

The percentage to reject \( H_0 \) at the significance level \( \alpha \) is expressed as \( P(T \geq z_\alpha) \), where \( z_\alpha \) is the number such that \( \Phi(z_\alpha) = 1 - \alpha \) and \( \Phi(x) \) is the cumulative distribution of a standard Gaussian.

But under \( H_A \) we have \( E(S) = \mu \) and \( V(S) = \sigma^2 \) so

\[ T = \frac{\bar{S} - \mu_0}{\sigma_0 / \sqrt{n}} = \frac{\bar{S} - \mu}{\sigma / \sqrt{n}} \cdot \frac{\sigma - \mu_0}{\sigma_0 / \sqrt{n}} \sim N\left( \frac{\mu - \mu_0}{\sigma_0 / \sqrt{n}}, \frac{\sigma^2}{\sigma_0^2} \right) \]  

(12)

This changes the percentage of rejecting \( H_0 \) to

\[ P(T \geq z_\alpha) = P\left( \frac{T - \mu - \mu_0}{\sigma / \sigma_0} \geq \frac{z_\alpha}{\sigma / \sigma_0} \right) = 1 - \Phi\left( \frac{z_\alpha}{\sigma / \sigma_0} \right) \]  

(13)

so we have that

\[ \pi_T(\alpha, H_A(\mu, \sigma), n) = 1 - \Phi\left( \frac{z_\alpha}{\sigma / \sigma_0} \right) \]  

(14)

This percentage to reject the null hypothesis given that the alternative hypothesis is true is plotted in Figure 9 for some values of \( \mu_0 \) and \( \sigma_0 \).
If the variance of the two sample populations is different there is a need for a modification of the t-test [15] in equation 10, to which Welch’s t-test[16]. is an approximation.

The t-statistic is computed as

$$t = \frac{\bar{S}_1 + \bar{S}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

(15)

Where $\bar{S}$ is the sample mean APFD, $s^2$ the sample variance of the APFD and $n$ the number of samples.

The number of degrees of freedom is

$$\nu = \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\frac{s_1^4}{n_1^2n_1} + \frac{s_2^4}{n_2^2n_2}}$$

(16)

If the computed statistic in equation 15 is above the threshold values as found in for example Numerical recipes 3rd edition: The art of scientific computing[9] or “NIST/SE-MATECH Engineering Statistics Handbook”[14], then the null hypothesis can be rejected.
3 Related work

In this chapter articles closely related to the thesis are reviewed. At the end of each section, comments on how the paper relates to this thesis are presented.

3.1 Test Selection Based on Historical Test Data

In this paper [17], Dunn-Ekelund presents a prototype program which continuously parses and analyzes historical regression test data, to create a weighted correlation between modified code packages and related test packages. The correlation between code packages and test packages are then used to make a recommended subset of regression tests with the intent to reduce the time duration of a given test suite.

The primary purpose of this work is to find a way to minimize the time spent running a big regression test suite for an Axis-product. Running the whole test suite for an Axis-product without test-selection can approximately take seven hours to run where most tests are not correlated with the code changes made.

Other existing selection techniques are very good at finding which function affects which tests ([2] for a general overview, [18] for a description of a system monitoring changes to software source code to make a selection of tests to run, [19] look at changes at the intermediate level, or Java byte code, for making a selection, [20] for using the relation between objects in object oriented programming to identify affected classes), but the techniques usually involve static analysis which can be very costly to perform on a company’s large and complex software.

What especially makes Ekelund’s selection technique different from other technologies is that he performs the correlation between code and tests on a package level, rather than going into details with static analysis. The correlation between code packages and test packages is done while running the code and inspecting the results in real-time (dynamic analysis).

Ekelund performed the test selection on real historical regression test data, but to determine whether the correlations found are correct he had to generate simulated code packages and test packages that had names that correspond to each other. The accuracy of the test selection was then evaluated on simulated data with a single measurement called F-measure (see F-measure, section 2.2.3. The evolution results eventually showed that running only relevant regression tests could reduce the time duration spent running a test suite by a factor of 20.

Comments

Ekelund’s research project was also conducted at Axis Communications AB which means

---

4 A test package is a collection of tests. A correlation between a modified code package and a related test package is a simultaneous change to the code package and change in test verdict
that our test environment is similar. The historical test data for different Axis-camera products are still available but are now even bigger.

Axis current Suite Builder does not select a subset of tests to run. The current Suite Builder only prioritizes based on historical test failure and duration. Ekelund’s promising results have convinced us that a selection technique in addition to the current prioritizer can improve the Suite Builder.

3.2 Automated System Level Regression Test Prioritization Using Multiple Factors

This paper by Strandberg, Afzal, Ostrand, et al. introduces an automated tool called SuiteBuilder developed by Westermo Research and Development AB [21].

Westermo experienced that their nightly regression tests did not finish in time, until the next workday, due to the big regression test suites. Westermo realized that they had to do something about the time-consuming execution because the big regression test suites would eventually get even larger over time, as more and more test cases would be implemented.

Therefore, the SuiteBuilder tool was developed to reduce the time spent on nightly regression tests and to quickly identify critical bugs while not excluding any important test cases.

The SuiteBuilder tool consists of individual prioritizers that each focuses on prioritizing test cases according to a specific factor such as fault detection, time duration, interval since last executed, or based on revisions of the code tested. The separate priorities are then combined with a priority merger which generates a more cost-effective regression test suite.

Westermo stated that their experimental evaluation of the SuiteBuilder tool showed significant improvements in regression testing results.

Comments

Westermo's solution with SuiteBuilder was used as inspiration for Axis SuiteBuilder. However, due to slightly different conditions on information available, the prioritizers used in Axis solution were limited to be based on failure rate and test duration. This limitation is not necessarily inherent, but rather a pragmatic choice on which prioritizers to implement first.
4 Method

A combination of two methods are used. The first part will follow the scientific method of a controlled experiment to determine how it is possible to measure and test if there is a difference of the alternative prioritizing methods. The second method will be used to develop a prototype of the alternative prioritizer on which measurements can be made for use in the controlled experiment of detecting a difference.

4.1 Controlled experiment

The controlled experiment as described by Easterbrook, Singer, Storey, et al. in [22] is used as the template for one part of the method. The method will help to provide guidelines for how to undertake the study and break it down into more manageable sub problems. The controlled experiment provide a framework for varying a variable and observe its effect. The goal is to be able to test the hypothesis, as in [12], that the variable does not provide any observable difference other than what could be expected from chance.

4.1.1 Theory

An underlying theory is developed explaining how and why the particular phenomena of correlation between changes in code packages and changes in test result express itself making it possible to make predictions. This theory also dictate what to observe and how allowing for an interpretation. As the aim is a quantitative measure of how a particular variable affect the outcome a explicit theory is needed as a lens through which to observe the experiment.

This is explained in section 5.1.1.

4.1.2 Hypothesis

To conduct a controlled experiment a testable hypothesis based on the theory is presented. It is desirable to present a hypothesis which is only dependent on a single parameter in the model. This hypothesis together with the underlying theory guide the steps of the experiment design. It is also stated under which conditions the hypothesis will be rejected.

This is presented in section 5.1.2

4.1.3 Model and samples

Using the theory a model is constructed which enable the collection of samples in the experiment. The samples are to be used in testing the hypothesis and thus have to be
related to the theory and hypothesis.
This is presented in section 5.1.3

4.1.4 Experiment

Using the model the experiment is conducted collecting the necessary data. To perform the experiment the prototype from section 4.2 is needed.
This is described in section 5.1.4

4.1.5 Hypothesis testing

The data collected in the experiment is used to test the validity, or refute, the stated hypothesis. The processing and computation using the data is presented and related to the hypothesis.
This is presented in section 5.1.5

4.2 System development

The choice of method for developing the system was the one described by Nunamaker Jr, Chen, and Purdin in “Systems development in information systems research” [23]. Figure 10 show an overview of the stages involved in the method. This will provide a path to go, in a methodical manner, from the requirements on the system to the finished prototype by clearly delineated steps in the process. It also allows for an iterative approach to solving the problem as the requirements are likely to change and become better formulated with more detailed domain knowledge. The iterative approach will allow for adapting to better understanding of how the interaction with already existing external components will be.
4.2.1 Construct a conceptual framework

The first step is to construct a conceptual framework for the system. For this to be possible, the problem to be solved and the conditions under which to solve it, have to be clearly stated. Also important is to survey what other approaches have already been taken to solve similar problems. This gives a framework within which to discuss different approaches to the problem.

This is presented in 5.2.1

4.2.2 Develop a system architecture

The external components impose rules and limitations on the interactions of the designed system with its surroundings, limiting and governing the system architecture in relation
to the external components. The data processing pipeline can then be constructed and logical components separated to support ease of changing or extending the pipeline.

This is presented in 5.2.2

4.2.3 Analyze and design the system

This part of analysis was mostly related to what kind of data already existed in the database concerning the historical test results. A thorough understanding of the data in the database provides a clearer path to how to extract and process the data. Building blocks already exist within Python packages leaving us to spend most energy on designing the system by careful analysis.

This is presented in 5.2.3

4.2.4 Build the prototype system

Utilizing the analysis and design from previous steps in the design process this step is concerned with actually putting the building blocks in place and realizing the system. This step gives important feedback to the previous steps through the implementation of the ideas.

This is presented in 5.2.4

4.2.5 Observe and evaluate the system

Finally the system is evaluated by computing a metric on the test suite. This gives an objective measure on how the test suite would perform. In addition this can also be used to compare different implementation strategies.

This is presented in 5.2.5
5 Result

Here the results are presented. The ordering of the subsections is based on content. That is, the development of the prototype system is broken out into it separate sub-section separate from the controlled experiment. A chronological ordering would be to let section 5.2 follow immediately after 5.1.3.

5.1 Controlled experiment

In *Testing statistical hypotheses* [12] a general outline for the steps of testing a hypothesis is given. For this to be possible a null hypothesis and its alternative hypothesis have to be formulated addressing the research problem. The goal is then to evaluate the truth of this hypothesis. Evaluation is usually done by rejecting the null hypothesis stating that no measurable difference exist.

The assumptions being made about the sample used for the test have to be considered and stated. Is is these assumptions which make it possible to perform the computations in a reliable manner.

A choice of test statistic has to be made. It is this test statistic which is used in evaluating the hypothesis.

The test statistic is to be evaluated in an appropriate test related to the test statistic.

Is the null hypothesis is correct, the statistical distribution of the test statistic has to be derived

A significance level, or threshold, of the test has to be determined. If the percentage of the test statistic falls below this threshold the null hypothesis will be rejected.

From the observations the observed value of the test statistic is computed.

The decision is made to reject the null hypothesis or nor based on the observed test statistic and the computed threshold value.

5.1.1 Theory

To form a theory, or understanding, a bit more of an in depth investigation of the relation between tests and software is needed. As explained in section 4.1.1 the goal is to form a theory. This theory is used for forming an interpretation of the observed results from varying the threshold.

In section 1.1 an overview was given on the current solution but an example is beneficial.
As an example assume for simplicity that the camera system only has three functions: physically moving the camera, detect motion in the field of view and sending a notification by email. Together with this functionality let there exist two tests: move the camera to a location and feeding in data simulating motion. The expected outcome of moving the camera would be for it be where it was told to be, and for feeding in data simulating motion would be an email notification. The outcome of running a test would be classified as either pass or fail. Assume that previously all tests have passed and there is some change made to the software handling movement of the camera. Let the outcome of the test moving the camera be a fail but the other test pass. This event could be recorded as a single +1 in a three by two matrix with rows corresponding to functionality and columns to tests. Assume the software developers have insight and make a guess that the change in the software caused the test to fail. So a change is made to the software handing movement trying to fix the problem and the tests are ran again. Still the problem persists and the outcome is a fail on the movement test. Yet another change is made and now all the test passes. So this event could be recorded as a +1 in the matrix. Note that there was no recording made when the test did not change outcome! So the matrix now contain a single 2 in the cell corresponding to the software controlling the movement and test of moving the camera. That there is a connection between the software controlling the motion of the camera and the test of the motion may seem obvious. Yet, this 2 could be taken as an indication that there is a correlation between the software controlling the motion and the test of the motion. The relation between the software sending email notification and detecting motion and the other test is could be a bit more intricate and a two to one relation could be imagined. However, it is also easy to imagine there to exist a one to many relation, i.e., one piece of software related to many tests. As the number of software pieces and tests grows it starts becoming intractable to remember all correlations between software and tests.

Looking a bit more closely at the test to get a better understanding. The tests are partitioned into different logical areas: storage, network, etc. Each of these areas belongs to a type related to the firmware or external interface. This creates a partitioning of test cases for a given area and type as can be seen in the top of Figure 11 where the lightness indicates the number of test-cases within each particular area. If the percentage for a test to fail would be the same for every test, then the number of failures would follow the same distribution as the number of tests and the bottom part of Figure 11 would look very similar to the top. The situation, however, is a bit more complicated. In Figure 11 the total number of tests are displayed together with the cumulative history of failures. This means if a given test is run for many products this will skew the distribution to a higher propensity for failure, likewise if a test is run for few products the skew will be to lower failure rates.
Figure 11: The top image depict the number of tests per area and type of test with intensity of color representing count, a so called heat map. The bottom image is a heat map for number of failures. The area with most tests are for security which also has the most number of failures. However, one aspect which gets hidden in this representation is that some tests are applicable to all products producing a higher count of failures. Two kinds of tests exist; external interface tests (eit) and firmware functionality tests (fft).

In Figure 12 this skew is shown as the relative number of failures per test area. That is, the number of failures is divided by the number of tests of that area and shown on a logarithmic scale. The two kinds of tests have different symbols. Round symbols for eit and diamonds for fft tests. If there are no failing tests for a particular combination of test type and test area the rate is plotted at zero and colored red. These combinations with no failing tests are impossible to predict based on the historical failure rate. Another implication of this is that the correlation matrix will have some zero entries.
Looking at the matrix from the example this would also increase in size with increasing number of software pieces and test. If many tests have been run, there would have been a +1 with each simultaneous change of software piece and test outcome. This number could then be used as an indicator on the correlation between software pieces and tests. No extra instrumentation on the code is needed, as it is sufficient to use the historical test data of changed test outcomes together with a log on which piece of software changed to construct the matrix.

If we know that there has been a change made to a piece of software, this given information could be used to make inference using the correlation matrix on which tests would most likely change outcome. This is the underlying idea on which we can base a hypothesis on.

This theory also dictate what to observe and how allowing for an interpretation. As the aim is a quantitative measure of how a particular variable affect the outcome a explicit theory is needed as a lens through which to observe the experiment.
5.1.2 Hypothesis

As presented in section 5.1.1 the count of number of times a test has changed outcome simultaneously with change to a piece of software, or correlation, can be used as indicator of the true correlation between test and software. However, to classify a pair, software piece and test, as correlated a decision has to be made on the number. One such simple decision is to base it on a threshold of the correlation count, i.e., if the correlation count is above the threshold they are classified as related, otherwise not. For example, let two particular pairs of test and piece of software have changed simultaneously 5 and 2 times respectively. Then for a threshold of 3 the first pair will be considered correlated but the second will not be considered correlated. Using the threshold as the parameter for the controlled experiment together with the APFD as the related measurable quantity the null hypothesis can be stated.

Using the correlation and a given threshold for predicting the order of the tests we perform no better, as measured by the test statistic APFD, than a random ordering of tests.

This hypothesis will be rejected if the APFD we measure is less than 5% likely to occur\(^5\).

5.1.3 Model and samples

The model is conceptually simple as only the historical test results together with the changes in software packages are required. Both of these resource are accessible through databases. As the data is collected on an evolving code and test base, not all tests are still in existence for example. This number, however, is a few in comparison to the total number thus they were excluded from consideration if they were not still in existence. Two years of historical data, up until 2018, was used as input for computing the correlation. To generate a sample for a date after 2018 the database was queried for the version of the software packages that day and also the software versions if the previous giving a list of software packages which had changes. These changes were then used in conjunction with the compute correlation to compute the ordering of the tests. To be able to compute the APFD the outcome of the tests had to be know. However, as the tests had been run already it was possible to query the historical tests result data and see which tests passed or failed. From this the APFD was computed and recorded as a sample.

This is further explained in section 5.2

\(^5\)This selection of 5% is a bit arbitrary but a common choice. For the situation at hand the choice is not too critical a it is just a initial exploration, but if the approach proves viable it may be good to re-evaluate this choice.
5.1.4 Experiment

The experiment consisted of computing the ordering of tests over a period of a four months using the actual test result to compute the APFD for varying thresholds. The prototype from section 4.2 was used as the simulation vehicle.

Table 2 show the different computed confusion matrices when using the correlation to create a prioritized list. The threshold is such that for a value of 0 there has to have been one change of code package and test verdict for a given build for a correlation to be said to exist. The correlation is computed from the historical data up to the last few firmwares, this is then used to predict how the tests should be ordered. The first column show the threshold, then comes the confusion matrix in the next two columns. To indicate how they are grouped the numbers are colored blue and follow the same convention as in Figure 4. A fuller confusion matrix is in Table 4 in appendix A which present the data in form of mean and standard deviations ”per product”, but also medians.

**Table 2:** Confusion matrices for different thresholds.

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>429 26896</td>
</tr>
<tr>
<td>1</td>
<td>407 17426</td>
</tr>
<tr>
<td>2</td>
<td>347 13411</td>
</tr>
<tr>
<td>3</td>
<td>107 10935</td>
</tr>
<tr>
<td>4</td>
<td>168 8354</td>
</tr>
<tr>
<td>9</td>
<td>131 6360</td>
</tr>
</tbody>
</table>

Figure 13 show in the last column the APFD calculated from the historical test results. The two latest, boxplots from 2018 and onward, correspond to the period in time over which the experiment was conducted to predict the ordering of the tests.
Figure 13: Box plots of first and last failure of the test suites grouped into interval over three months with APFD computed for the same period.

5.1.5 Hypothesis testing

The data collected and summarized in table 2 is tested against data from the null hypothesis. As we assume the null hypothesis to be a random ordering of the test this data can be simulated.

To, as truthfully as possible, replicated the real failure rate the percentage for a test to fail is computed from the confusion matrix

$$p_{\text{failure}} = \frac{TP + FN}{TP + FN + FP + TN}$$  (17)

For a given number of tests in a test suite $n_{\text{tests}}$ the number of failing tests are modeled as a binomial distribution $n_{\text{fail}} \sim \text{Bin}(n_{\text{tests}}, p_{\text{failure}})$. From the $n_{\text{tests}}$ a random selection without replacement of $n_{\text{fail}}$ is made representing the tests which will fail.

This gives a simulated random ordering of tests. Repeating this procedure many times a
The sample average APFD is calculated together with its sample variance, and exactly this data is used for testing the hypothesis. Average APFD becomes 0.49 and standard deviation 0.17 for 30,000 repeated trials.

To determine if the simulated APFDs follow a Gaussian distribution a percentage plot is shown in Figure 14. This is a plot of the observed values against the theoretical quantiles. A normal distribution can be transformed into one with mean 0 and variance 1. So, if the observed values fall on a straight line in the plot, the values are normally distributed. As the APFD has a lower limit of 0 and an upper limit of 1 is will not completely follow a straight line, but for the central part it will be close enough, and we work under the assumption of normality for the distribution. This will allow us to make use of Welch’s t-test as described in section 2.6.

![Probability Plot](image)

**Figure 14:** Percentage plot of simulated APFDs using a random ordering.

In Figure 15 the simulated APFD is plotted together with the one computed using correlation. The solid green line is the sample average of the simulated APFD, and the dotted green line represents two standard deviations. The computed APFD from the predictions is plotted as red dots with error bars indicating two standard deviations. As can be seen, the sample mean computed APFD from the predictions is two standard deviations above the sample mean from the simulated APFD. Yet, the issue remains if this difference is significant enough to be able to reject the null hypothesis. Not much change occurred going from a threshold of three to four, so it was not evaluated for the thresholds five through eight.
Figure 15: The red dots are the results from running the algorithm. Using the data from Table 2 a random test suite is simulated and the APFD is computed and shown as the green lines. The dashed line is $\mu + 2\sigma$. The blue line is computed from the historical data.

To test the hypothesis Welsh’s t-test[16] is employed. The t-statistic is computed as

$$t = \frac{\bar{S}_1 + \bar{S}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$  \hspace{1cm} (18)

Where $\bar{S}$ is the sample mean APFD, $s^2$ the sample variance of the APFD and $n$ the number of samples

The number of degrees of freedom is

$$\nu = \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\frac{s_1^4}{n_1^2n_1} + \frac{s_2^4}{n_2^2n_2}}$$  \hspace{1cm} (19)
Table 3: Computed \( t \) and \( \nu \) statistics from equation 18 and 19 for varying thresholds.

<table>
<thead>
<tr>
<th>Threshold</th>
<th>( t )</th>
<th>( \nu )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10.63</td>
<td>185</td>
</tr>
<tr>
<td>1</td>
<td>10.32</td>
<td>191</td>
</tr>
<tr>
<td>2</td>
<td>7.84</td>
<td>196</td>
</tr>
<tr>
<td>3</td>
<td>2.52</td>
<td>198</td>
</tr>
<tr>
<td>4</td>
<td>2.15</td>
<td>198</td>
</tr>
<tr>
<td>9</td>
<td>1.00</td>
<td>197</td>
</tr>
</tbody>
</table>

The critical values for \( t \) and \( \nu \) from “NIST/SEMATECH Engineering Statistics Handbook”[14] (1.960 for \( 1 - \alpha = 0.975 \)) are compare to the result in Table 3. We can not reject the null hypothesis for a threshold of 9. For a threshold of 4 or below, however, we are able to reject the null hypothesis.

5.2 System development

In this section the construction of the prototype system\(^6\) is described as set out in section 4.2.

5.2.1 Construct a conceptual framework

The problem as stated was to find a measure of the effectiveness of a prioritization. A measure of the effectiveness of a given prioritization will make it possible to objectively say if one approach is better than the other. In addition it was a desired to suggest an alternative, and hopefully better, approach to the current implementation. However, our goal for the alternative method is to determine if it does perform an ordering by a hypothesis test. This will help Axis to better utilize their resources.

There exists an extensive database of historical test data. In this historical test data, detailed information is included such as which test it was, software version, product. Everything to pinpoint the combination of software and hardware it was run on, the verdict and duration. The test verdict and duration information in this database is currently leveraged for computing a prioritization of the tests in the test suite. In addition, there exist a code repository (which can be considered a database) with information of which software package version is included in each firmware build for the products. It is hoped that the combination of the information in these two databases will give give a better prioritization that what is currently implemented.

In Figure 16 a conceptual model of the prototype is shown. The requirements are modes

---

\(^6\) The source code is obtainable upon request to the authors
and consist of the ability to make connection querying a database to retrieve information and perform the computations needed to perform the hypothesis testing. With a connection to the databases in the first step it will be possible to retrieve the data. However, it is unlikely that this data directly will be usable but instead it will require some data processing into a format which is useful for the second step. In the second step the computation will take place producing a prioritized test suite. This prioritized test suite will then have to be evaluated in the third and last step. By replacing the computation part with different approaches it will be possible to evaluate the current implementation as well as any other implementation.

![Figure 16: Partitioning of the prototype into three more logically contained parts.](image)

Figure 17 show a decomposition of the first part of measuring the performance. To be able to measure the performance the output from the different prioritization algorithms is computed in a data analysis step. Acquisition of the data will be through database connections. The prioritizers based on failure rate and test duration only look at the historical test data for generating the prioritization. As the historical test data already exist in an Elasticsearch database (database based on the query language Lucen), they will only utilize one database connection. Prioritizing on correlation is a bit more involved. It draws on three different databases, one containing the historical data, one containing the version of the software components and one containing the tests. Software versions are obtained through parsing a log file from each build and is obtained from a tree walk of a directory hierarchy presented through a web interface. Tests are contained in a database and accessed through a standard RestAPI.
5.2.2 Develop a system architecture

The architecture of the system is rather linear. It consists of a pipeline retrieving the data from databases, processing the data from the databases and performing computations and finally present the data.

Requirements on the system is for it to be able to retrieve the necessary information. For this to be possible it has to communicate with three types of databases. One is a standard big name database, one is a file served over http and one with a standard JSON get-API. The three databases are mentioned above containing the accumulated test verdicts and the code repository. As software evolve some test become, for example, obsolete or redundant and should no longer be run. The database containing the tests only contain the tests which are currently considered useful, and it only make sense to include the tests in this database in a test suite for new software versions. Once the information has been retrieved it has to be transformed from JSON in two cases, and one case a text file has to be parsed.

The list of functional requirements is rather short as they are dictated by the type of database connections

1. Retrieve data from Elasticsearch
2. Retrieve data from Rest API
3. Retrieve data from text-file presented in directory hierarchy served over webinterface

The information is then used for computing the summary statics.

5.2.3 Analyze and design the system

![Diagram showing databases connected to tests, software, and test results]

**Figure 18:** The database connection is to multiple databases each with their particular API and content.

5.2.4 Build the prototype system

Building the system consisted of implementing the functionality as a Python script.

5.2.5 Observe and evaluate the system

This thesis is a continuation on *Automated System-Level Regression Test Prioritization in a Nutshell* [21] and *Test Selection Based on Historical Test Data* [17]. However, as the aim is to measure and evaluate a suite prioritization, there is a need to know what other strategies have been used and which of these might apply to our situation. In [17] the evaluation is done through the F-measure which require there to be a selection performed. If there is no selection, other strategies such as time until first failure [24] or code coverage [6] could be used for example.

The requirement on the measurement is that it has to be based on the results from previous tests in the form of pass/fail. That is, it has to be performed without access to the source code or the test code, there can be no need to know the relation between software packages and test packages.

Software and regression testing are well-established fields within computer science field yet there exist specific words which are used in widely different meanings [24]. Instead, we opted to find well-cited survey articles and snowball from there evaluating if the method used in the cited article is evaluated in a, for us, relevant way.

The main entry point is the article by Yoo and Harman *Regression testing minimization, selection and prioritization: a survey* [2] resulting in articles *Prioritizing test cases for regression testing* [6] and *Test case prioritization: A family of empirical studies* [25].
These and many others use the \textit{average percentage of fault detection} (APFD) as metric which also fulfills the requirements we had on a metric. In addition, the APFD plot plays a similar role as a cumulative distribution function in percentage. Depending on the particular need and desire of the end user, different parts may be of different value.
6 Discussion

As expressed in "Exploratory data analysis" [10]

"It is important to understand what you CAN DO before you learn to measure how WELL you seem to have DONE it."

The artifact provide a way for an initial exploration of what can be done. This is demonstrated by rejecting the null hypothesis, i.e., as measured by the APFD the ordering is better than random.

6.1 Result

As presented most tests do not fail and tests which fail do so only a few times so it is unlikely to successfully predict the failure of a test based on historical results. Even if it has previously failed, this gives no information on when the test will fail next time. Some of the tests have never failed so it would be impossible to predict these until they have done so.

As the baseline the historical data is presented in Figure 13, a summary over approximately two years with each box plot correspond to three month worth of test data.

The APFD does provide a way to measure the effectiveness of a prioritization but it does come with limitations. Each end consumer of a regression test have slightly different criteria and no one measure could probably satisfy them all. Computing the correlation is possible and does provide insight into which tests to run when code packages change.

The hypothesis test does reject the null hypothesis of no effect. So the lucky combination that not every code package change with a given new firmware, it is possible to use the correlation to make a prediction on the ordering of the test cases given which code packages have changed.

Combined with the prototype it is possible to obtain an ordering of tests based on correlation of code change and test result changes without any extra instrumentation. This ordering does also provide a non null effect.
6.2 Threats to validity

One issue which might cause a problem with the validity of the reasoning in this thesis is the selection of the historical data for testing the hypothesis. This can be cast in the more common form of selecting training and testing data in other circumstances. To avoid selection effects the validation is usually done through folds, eg., k-fold cross validation. It might be that the current results are an artifact from this selection. Currently we do no believe this to have too much of an effect. But, nonetheless, it might be beneficial to perform a cross validation to determine if this is a valid assumption.

The quantile plot showed that the APFD is reasonably normally distributed. However, as the APFD is bracketed it is not truely normally distributed. This can have an impact on the hypothesis test, and especially the validity of rejecting the null hypothesis.
7 Conclusion and future work

This section provides a summary of the answers to the research questions and how this thesis contributes to the field. Lastly a short note on what future work may be fruitful is presented.

7.1 Answering the research questions

The research questions were presented in section 1.2.2 after a presentation of the settings and premises under which the questions would be answered. In this section, a summary of the answer is given.

*How do we implement the calculation of the correlation?*

In section 5.2.4 the presentation on how calculation of the correlation was done looking for simultaneous changes to test outcome and software changes.

*How do we measure the performance of an test suite prioritization algorithm?*

In section 2.3 an introduction was given to the APFD and in section 5.1.4 the results were presented.

*How do we determine if there is a measurable effect of the test suite prioritization algorithm?*

In section 5.1.5 the hypothesis testing was presented building on the material from section 2.6.

*Does the computed correlation between tests and software using historical data provide the ability to predict tests of interest given changes to software?*

At the end of section 5.1.5 the result of the hypothesis testing was presented indicating that we could reject the null hypothesis of no effect.

7.2 Contribution

This thesis provides an answer to one particular problem in one type of setting: Is it possible to compute the correlation between code packages and functional test cases, and if so is it useful. If the precondition is already satisfied, or if there is a better and more natural way to recover this correlation, eg., instrumentation of code, this information is valuable when prioritizing tests.
7.3 Future work

Future work could consist of putting the comparison on a more statistical sound base. Going back in time and performing the prioritization of the test and comparing with the actual outcome of the regression test. Also, the computation of the correlation might be served well by a closer investigation on detecting differences in package versions.
References


[16] Bernard L Welch. “The generalization ofstudent’s’ problem when several different population variances are involved”. In: Biometrika 34.1/2 (1947), pp. 28–35.


Appendices

A Full confusion matrix

Table 4 show the different computed confusion matrices when using the correlation to create a prioritized list. The threshold is such that for a value of 0 there has to have been one change of code package and test verdict for a given build for a correlation to be said to exist. The correlation is computed from the historical data up to the last few firmwares, this is then used to predict how the tests should be ordered. The first column show the threshold, then comes the confusion matrix in the next two columns. To indicate how they are grouped the numbers are colored blue and follow the same convention as in Figure 4. Next comes the mean and standard deviations "per product", and last the medians. For example, for a threshold of 0 a prediction is made that ~ 200 tests will fail of which only 3 actually fail, still ~ 2000 tests are predicted not to fail, which is actually also the case.

**Table 4**: Confusion matrices for different thresholds.

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Sum</th>
<th>Mean</th>
<th>(Std)</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>429</td>
<td>26896</td>
<td>3.25 (4.13)</td>
<td>203.76 (34.10)</td>
</tr>
<tr>
<td>158</td>
<td>280565</td>
<td>1.20 (1.45)</td>
<td>2125.49 (236.11)</td>
<td>0.0</td>
</tr>
<tr>
<td>1</td>
<td>407</td>
<td>17426</td>
<td>3.08 (3.83)</td>
<td>132.0 (14.44)</td>
</tr>
<tr>
<td>180</td>
<td>290035</td>
<td>1.36 (1.72)</td>
<td>2197.23 (245.57)</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>347</td>
<td>13411</td>
<td>2.63 (3.02)</td>
<td>101.60 (14.44)</td>
</tr>
<tr>
<td>240</td>
<td>294050</td>
<td>1.82 (2.86)</td>
<td>2227.65 (246.84)</td>
<td>0.5</td>
</tr>
<tr>
<td>3</td>
<td>107</td>
<td>10935</td>
<td>1.39 (1.55)</td>
<td>82.84 (14.63)</td>
</tr>
<tr>
<td>403</td>
<td>296526</td>
<td>3.05 (5.47)</td>
<td>2246.41 (249.44)</td>
<td>1.0</td>
</tr>
<tr>
<td>4</td>
<td>168</td>
<td>8354</td>
<td>1.27 (1.25)</td>
<td>63.29 (5.14)</td>
</tr>
<tr>
<td>419</td>
<td>299107</td>
<td>3.17 (5.44)</td>
<td>2265.96 (253.91)</td>
<td>2.0</td>
</tr>
<tr>
<td>9</td>
<td>131</td>
<td>6360</td>
<td>0.99 (1.01)</td>
<td>48.18 (2.40)</td>
</tr>
<tr>
<td>456</td>
<td>301101</td>
<td>3.45 (5.50)</td>
<td>2201.06 (255.17)</td>
<td>2.0</td>
</tr>
</tbody>
</table>

In Table 5 the average computed summary statistics using different thresholds are shown. Most interesting is the APFD which can be used to compare the result with the one using prioritizers based on testcase failure rate and duration.
Table 5: Summary statistics for different thresholds of correlation

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Precision</th>
<th>Recall</th>
<th>Accuracy</th>
<th>$F_1$</th>
<th>MCC</th>
<th>APFD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0157</td>
<td>0.7308</td>
<td>0.9122</td>
<td>0.0307</td>
<td>0.0987</td>
<td>0.82</td>
</tr>
<tr>
<td>1</td>
<td>0.0228</td>
<td>0.6934</td>
<td>0.9428</td>
<td>0.0442</td>
<td>0.1189</td>
<td>0.82</td>
</tr>
<tr>
<td>2</td>
<td>0.0252</td>
<td>0.5911</td>
<td>0.9557</td>
<td>0.0484</td>
<td>0.1156</td>
<td>0.77</td>
</tr>
<tr>
<td>3</td>
<td>0.0165</td>
<td>0.3135</td>
<td>0.9632</td>
<td>0.0314</td>
<td>0.0650</td>
<td>0.64</td>
</tr>
<tr>
<td>4</td>
<td>0.0197</td>
<td>0.2862</td>
<td>0.9715</td>
<td>0.0369</td>
<td>0.0689</td>
<td>0.63</td>
</tr>
<tr>
<td>9</td>
<td>0.0202</td>
<td>0.2232</td>
<td>0.9779</td>
<td>0.0370</td>
<td>0.0615</td>
<td>0.60</td>
</tr>
</tbody>
</table>