The Perfect Test Mix

DANIEL JENDEBERG
The Perfect Test Mix

Daniel Jendeberg

Master of Science Thesis INDEK 2017:111
KTH Industrial Engineering and Management
Industrial Management
SE-100 44 STOCKHOLM
Den Perfekta Testfördelningen

av

Daniel Jendeberg
Abstract
Software development is getting more and more important today because all types of businesses use specialized software. The software used is often central to the business and it needs to always be functional. Therefore, the importance of software testing is becoming more relevant than ever. During the development projects, testing usually takes up the most resources and these resources are divided, within the test portfolio, between different kind of testing. Testing resource allocation is a constant challenge in terms of optimizing resource usage. In this thesis, the relation between automatic and manual testing will be studied to see if it is possible to optimize project results or cost in relation to these projects. To study the effects of test portfolio management on test project results, a study of historical test project allocation and release bug data was conducted at Nasdaq. The data were then statistically analyzed and correlated with an interview with a test manager. The findings show that certain allocations seem to optimize either the result or the cost of the testing projects. Further, the findings show a new type of testing that is neither automatic nor manual, this semi-automated testing needs further research.

Key-words: Software testing, Software development, Portfolio management, Test management, Test portfolio management, automatic testing, manual testing
Sammanfattning


Nyckelord: Mjukvara, Mjukvaruutveckling, Mjukvarutestning, Portfolio management, testning, test management, test portfolio management, automatisk testning, manuell testning.
## Table of Contents

1. Background ......................................................................................................................... 1
   1.1 Introduction ...................................................................................................................... 1
   1.2 Problem formulation ........................................................................................................ 2
      1.2.1 Problem statement .................................................................................................... 2
      1.2.2 Purpose .................................................................................................................... 3
      1.2.3 Research questions ................................................................................................. 3
      1.2.4 Delimitations .......................................................................................................... 4
      1.2.5 Ethical aspects ........................................................................................................... 5

2. Theoretical framework ........................................................................................................ 6
   2.1 Software testing ............................................................................................................... 6
      2.1.1 Testing process ........................................................................................................ 6
      2.1.2 Testing strategies ..................................................................................................... 8
   2.2 Manual and automatic testing ....................................................................................... 9
   2.3 Test portfolio management .............................................................................................. 10
      2.3.1 Portfolio management and resource allocation ....................................................... 10
      2.3.2 Resource allocation within software testing ............................................................. 11
   2.4 Sustainability ................................................................................................................ 12
      2.4.1 Managerial sustainability ......................................................................................... 12
      2.4.2 Software sustainability and “Green IT” .................................................................. 13
   2.5 Thesis theoretical positioning ....................................................................................... 15
   2.6 Definition of concepts .................................................................................................... 15
3. Method .......................................................................................................................... 17
  3.1 Research design ........................................................................................................ 17
  3.2 Part 1 .......................................................................................................................... 18
    3.2.1 Qualitative methodology ....................................................................................... 18
    3.2.2 Semi-structured interviews ................................................................................ 19
    3.2.3 Thematic analysis ................................................................................................. 19
  3.3 Part 2 .......................................................................................................................... 20
    3.3.1 Quantitative methodology ................................................................................... 20
    3.3.2 Data gathering ...................................................................................................... 20
    3.3.3 Simple quantitative analysis ............................................................................... 22
  3.4 Authors relation to case ............................................................................................. 23

4. Result ............................................................................................................................ 24
  4.1 Interview ..................................................................................................................... 24
  4.2 Project release data .................................................................................................... 25
    4.2.1 Resource allocations per release ......................................................................... 25
    4.2.2 Software bugs ....................................................................................................... 26
  4.3 Correlation between software bugs and allocations .................................................. 30

5. Discussion ...................................................................................................................... 32
  5.1 Part 1 – Thematic analysis .......................................................................................... 32
  5.2 Part 2 – Statistical analysis ........................................................................................ 33
    5.2.1 Issues in productions ......................................................................................... 33
    5.2.2 The golden ratio ................................................................................................. 33
5.3 Semi-automatic testing ..........................................................................................34
5.4 Triangulation of Part 1, 2 and Theory .................................................................35
5.5 Effect on sustainability ..........................................................................................35
5.6 Sources of error .....................................................................................................36
  5.6.1 Dialog interpretations .......................................................................................36
  5.6.2 Data gathering and analysis ............................................................................36
  5.6.3 Estimation of testing success ..........................................................................37
  5.6.4 Authors connection to company ....................................................................38
6. Conclusion and Recommendations ........................................................................39
  6.1 Research questions .............................................................................................39
  6.2 Managerial Implications ....................................................................................40
  6.3 Recommendations for “golden ratio” ..................................................................41
  6.4 Interpretation of recommendations ....................................................................41
  6.5 Further research .................................................................................................42
References ..................................................................................................................43
Appendix A – Tables of information ........................................................................45
  A1. Table of figures ....................................................................................................45
  A2. Table of tables ....................................................................................................46
1. Background

1.1 Introduction

The need for computer technologies is growing in today’s digitalized and global economic climate, and it is likely that the growth doesn’t stop here. What is needed to keep in mind is that “With great power comes great responsibility” (Lee & Ditko, 1962), in other words with the growth of software technology comes further needs to ensure software reliability. One such reliability issue is the occurrence of unintentional or faulty behavior in the software, so called software bugs. These software bugs can for example cause issues with faulty behavior or even leave security breaches, so called vulnerabilities, in the system. “Software is complex and humans are fallible, so vulnerabilities are bound to occur” (Wilson et al. 2016). As systems grow more complex there will be even harder to detect and fix all bugs in a system, therefore research considering how to prevent software bugs is getting more and more important.

Software testing is the verification of software functionality, i.e. thoroughly testing the software to minimize the occurrence of software bugs. Software testing is not as easy as it sounds, since completely verifying the functionality of software is a complex task to say the least. Software testing usually makes up the single most expensive part of a software development project (Yamada, 1995; Wang et al, 2010). As software systems grow more sophisticated and complex so does also the bugs of these systems. This increases the need for sophisticated testing of the system and complex testing to make sure that the software products do not suffer from critical issues that make it into production (Wang et al, 2010).

How to test and verify the reliability, stability and safety of a large and complex system of software create a problem. Given the practical and economical limitation of any software project it will be even harder to know how to manage the quality assurance and testing of software project. Software testing is getting increasingly important as well as increasingly difficult,
because of the increasing complexity of software. Older research discusses and provide a view of holistic test coverage, where the tester might use code review as a method of testing (Basili & Selby, 1987). This view of finding every bug and error is not applicable any more in the increasingly complex software system of today especially given the economical limitations of testing projects. Today research is getting more and more related to efficiency of testing (Ramler & Wolfmaier, 2006; Wong et al, 1997; Wang, & Yao, 2010; Amland, 2000). Since many studies of testing are still are looking at the individual software module and modular testing this thesis will attempt to hold a more systems view of the software and attempt to look at the entire product and its functionality.

1.2 Problem formulation

1.2.1 Problem statement

The problematic aspects of software bugs and issues are well known and have been studied for decades. This thesis acknowledges these problems and looks to solve, or minimize, the computer technological phenomenon through utilizing portfolio management strategies to see if there is a difference in performance depending on how resources are allocated. The case will be examined at Nasdaq Inc. that has a strong testing (quality assurance) organization separate from the software development and have kept track of both the resource allocation in their different projects for a long time as well as the number of bugs found in post development states.

Software testing is often divided into two categories, the manual and automatic testing. This thesis will analyze the two categories as part of the test portfolio management strategy. It will also examine how a specific test portfolio management strategy of resource allocation between the different categories will affect the outcome of testing in a complex software system. The problem is not limited to Nasdaq Inc., it is a common issue in software development and testing.
1.2.2 Purpose

The purpose of this thesis is to examine the effects of different resource allocation strategies on software testing in complex systems.

The purpose is the ground for a predictive study (Blomkvist & Hallin, 2015, p. 25), predicting the behavior of testing performance based on test portfolio management. Since the relation between the two is not clear in previous studies and the field has not developed any best practices for resource allocation within test portfolio management the study will also be exploratory (Blomkvist & Hallin, 2014, p.24). In the scope of the exploratory study this thesis attempts to find previous practices at the company and describe how the decision process of test manager have been handled previously.

1.2.3 Research questions

The main research question to be examined in this thesis is:

RQ: How do test portfolio management strategies affect the outcome of test processes and do these strategies have a measurable effect on the number of bugs found in post development?

To answer the main research question (RQ) the following sub questions will need to be examined:

SQ1: Is there a correlation between resource allocation and bugs found in post development?

SQ2: Could portfolio management strategies be applied to testing of software to minimize bugs or resource usage?

SQ3: Is management susceptible to changes in test portfolio management?

SQ1 is the question that a quantitative analysis will focus upon; priority will be to mathematically model the answer. The answer to this question will then be followed up by SQ2 which will be
determined from SQ1 with the help of qualitative interviews to look at different allocation problems that managers face within the organization under study. SQ3 will be solely based on qualitative interviews to see if the managers have room in the project planning process to apply the knowledge from this thesis to further the software testing that is conducted, the answer to SQ3 will have implications for how managers at Nasdaq Inc. should use the recommendation in this thesis.

1.2.4 Delimitations

1.2.4.1 Theoretical
The first theoretical delimitation is that this thesis will only be looking at literature written in English or Swedish, since those are the language that the author knows well enough to understand the publications. Further this thesis will focus primarily on literature within resource allocation, portfolio management, resource allocation problems within software testing, as well as literature within software testing. This delimitation means that some aspect of the theory that could have affected the result might have been left out.

1.2.4.2 Practical
Nasdaq, in its current form is the result of the merger of the Sweden based OM and the American Nasdaq. Both Nasdaq and OM started out as pioneers within the field of electronic trading. Technology remains a large part of the business that Nasdaq conducts today, not only do they run 25 stock exchanges completely but over 70 exchanges from 50 countries have trusted Nasdaq technology and are running it (Nasdaq, 2017). Today Nasdaq Nordic, which is not a legal entity but the name used for Nasdaq’s business in the Nordic and Baltic countries, runs several exchanges including the Stockholm Exchange.

Nasdaq which is one of the largest suppliers of electronic trading technology in the world and the reliability of their systems need to be high. For example, on their largest stock exchange they
trade approximately 2 billion stocks daily, which is approximately 70’000 stocks per second in the 8 hours the exchange is open for trading. The technical systems that handle trading and post trade operations is complex and needs to be both reliable and secure, therefore it is important to verify the functionality of the software.

This thesis is delimitated to the operations of Nasdaq Inc. and will not look at the functions of any other company. This thesis will also be delimited to the software development and testing of products related to the TSN customer (Trade Systems Nordic) which leaves a fixed set of deliveries and releases of software. This delimitation is in place because of the set time limit and TSN is one of the customers that historically have had the most releases per year, the customer is also located in-house which improves the ability to access post development data and processes.

Furthermore, the data used for the statistical analysis will be the time and software issues reported to Nasdaq Inc., and therefore might be object to biases from the company as well as human error. The project deliveries will be analyzed from the amount of time reported for each project and might in some cases differ depending on of who has worked on the project and how many people the time is split between. These delimitations will also affect the reliability of the data gathered and will be brought back in the discussion as sources of error.

1.2.5 Ethical aspects
This thesis will be utilizing the code of ethics from the Swedish Research Council that states the following four areas of rules (Vetenskapsrådet, 2002):

- Information requirements.
  The researcher is required to inform those who are affected about the research objectives.
- Consent requirements.
  The researcher is required to inform participants that they are free to choose to what extent they are participating.
• Confidentiality requirements.
The researcher is, to the best of their ability, required to attempt to give the highest degree of confidentiality and store personal data so that unauthorized personal can’t access it.

• Usage requirements.
The researcher is required to only use the data gathered for the expressed purpose of the research and is not allowed to use the data for anything else outside of the research.

2. Theoretical framework

2.1 Software testing

2.1.1 Testing process
At a large software company that produces complex system, testing is a central part of the business; approximately taking 40-60% of the development resources and is often considered the most crucial and expensive phase of the project (Yamada, 1995; Wang et al, 2010). Though testing is often referred to as one thing, software testing is divided into different areas that falls under different people’s profession. The different categories of testing that is the responsibility of the development organization are: Module Testing (Unit Testing), Functional Testing (Integration Testing), and System Testing (Yamada et al, 1995; Myers et al, 2004). There are further testing categories, for example the Acceptance Test, that is done by the end-user to check that the program meets the objectives set before development, but this type of testing is outside the scope of this thesis since we will focus on the firm that develops the software. In other words, from the view of the development organization’s testing they should find all (or most) bugs possible prior to the Acceptance Test.
Figure 1: The various stages of software development and where the corresponding testing stages appear (Myers et al, 2004, page 94).

The previous figure demonstrates the software development process from a high-level perspective and shows where and when the various types of testing takes place (Authors note: the size of the different stages has no correlation to the resources they take to complete). Viewing the
Acceptance test as not part of the software development project itself helps give the view of it being the delivery and every bug found after that should be considered found after launch.

2.1.2 Testing strategies

2.1.2.1 Outdated research
Old research within software testing state that code reviewing and reading is at least as good as functional testing, if not better, at finding bugs (Basili & Selby, 1987). Although this may be true, the idea that testers should read through the entire code of a single process is impossible in practice, let alone to code review an entire software system. This concludes that older research is outdated within this area and its usability is limited in the current large complex software system since the length of the code is too long to read and the interaction between systems might create new unintentional interactions.

2.1.2.2 Regression testing
Regression testing is the act of testing functionality that should act in a specified way to determine that new functionality does not introduce bugs in old functionality (Wong et al, 1997). This is important for all software to determine that the system remains stable and that current functionality that should not be affected by the latest changes is not affected. One issue with regression testing is that to fully regression test a system one would have to test all functionality of every part of an entire system, which could get very time-consuming in large and complex systems. Therefore, other strategies have been developed, such as selective regression testing, to minimize the time it takes to test the system while keeping the risk of instability to a minimum (Leung & White, 1991). These strategies could be utilized to minimize the amount of resources needed for regression testing and these resources can be reallocated in accordance with strategies from section 2.3.1 Portfolio management and resource allocation.
2.2 Manual and automatic testing

In software testing, there are two distinctly different ways to conduct testing, automated and manual. Both types of testing are done to verify the functionality of the system and the difference between the two is that manual testing is conducted by a human, usually through some sort of GUI, while automatic testing is conducted by the computer through an API, Application Program Interface (Ramler & Wolfmaier, 2006). Furthermore, the differences between the two gets clearer when looking at the benefits, since manual tests are conducted by a human there occurs on the spot reasoning and the manual tests has a flexibility that automated tests don’t have. The benefits of automated tests can run faster, conduct more operations per time unit and can do so more accurately than manual tests.

Ramler and Wolfmaier (2006) have conducted a study where they look at the economic aspects of manual and automated testing. They developed a model to showcase when to automate and when to keep the testing manual by showing the cost of testing as linear functions of how many times the test is run.

Figure 2: The cost of testing per number of test runs with the cost of manual and automated testing (Ramler & Wolfmaier, 2006).
As seen in figure 2 the cost of testing is different between automated and manual testing. The initial cost of automated testing, $V_a$, is usually larger than the initial cost of manual testing, $V_m$. After the initial cost the two lines are linear and dependent on the number of test runs, and manual testing has a higher coefficient. The higher coefficient for manual testing shows that the cost of running the test is higher and it shows that automation of tests is better the more the test will be run. Ramler and Wolfmaier (2006) use this mathematical model to show when tests should be automated.

2.3 Test portfolio management
2.3.1 Portfolio management and resource allocation
Portfolio management is the processes of managing the project portfolio and constitutes of maximizing the project benefits from a fixed set of resources. For example, this involves Go/Kill decisions and resource allocations and reallocations within the project (Cooper et al., 2001; Cooper et al., 2001). The use of portfolio management is widely spread in today’s business environment, and according to Cooper et al. (2001), the most popular method to use is financial calculations to determine resource allocations. The calculations used could, for example, be NPV (Net Present Value) or ROI (Return on Investment) with the goal to maximize benefits from the used resources, which in turn determines where to allocate resources. The implications for management of a test portfolio would be to manage the resource allocation between different projects within software testing, and determine where resources would most benefit the testing of the product. Examples of division within software testing projects could be automated and manual testing, which is the case in most literature (Ramlıer & Wolfmaier, 2006). In this thesis, we will look at the test portfolio, the portfolio of test projects, as a value creating portfolio and therefore the ideas from Cooper et al. (2001) will be valuable to explain the work conducted in the portfolio (Ramlıer & Wolfmaier, 2006). Both Cooper et al. (2001), and Ramler and Wolfmaier
(2006) discuss resource allocations and optimization of company resources, which is an integral part of the test portfolio management.

2.3.2 Resource allocation within software testing

There has been a lot of research committed to the area of module testing and the optimal algorithms for resource allocation in such projects, but there are only a few that try to find out how to commit resources to testing in larger projects where the complex testing can take many forms. In large complex projects with long event based chains of operation, exhaustive testing will be harder than just testing a set of n different interactions (Kuhn et al., 2004). An issue with the current research is the single objective optimization, where the researchers are only trying to optimize the testing in relation to one objective. In the real world there are several different objectives that can be focused on optimizing. This objective of software testing may vary, for example minimizing cost, minimizing actual software faults or minimizing the number of software faults detected by end-users (Wang et al, 2010). The argument can be made that testing is single objective focused, but that narrow-minded view diminishes the actual complexity of testing. Another point that is missed is in the current research is the difference between manual and automatic testing. In theory, automated tests are better at testing the reliability of software (Myers et al, 2004), but it is easier to simulate user behavior through manual testing and manual testing is required to examine a GUI (Graphical User Interface). In this thesis, the view of software testing is dual objective based, optimizing the effectivity of resources and minimizing the number of faults in post development. Further, this thesis also views software testing as value providing (Ramler et al. 2006). This view promotes a view of being sufficiently exhaustive in testing as well as using the resource in the most effective way.
2.4 Sustainability

2.4.1 Managerial sustainability

Figure 3: The three sustainability aspects and their intersections represented as three circles (Sikdar, 2003).

There are three aspects of sustainability; Economical, Sociological, and Environmental (Sikdar, 2003; Silvius & Schipper, 2010). These three aspects all play into making development sustainable and needs to be viewed separately in the analysis of this case. Further, Silvius and Schipper (2010) states:

“Sustainability in projects and project management is about integrating economical, environmental and social aspects in the management and delivery of projects.” Since this thesis focuses on test portfolio management the project management aspects of sustainability have to be considered.
2.4.2 Software sustainability and “Green IT”

Within the scope of this thesis the sustainability of software, software systems and IT systems need to be considered as well. The highest levels of environmental issues come from the creation of the electronical devices used (Agarwal et al., 2012). It is hard to connect software to tangible environmental issues, as is shown by Agrawal et al. (2012) saying:

“The general perception about a software is that it is automatically green and software hardly has an environmental impact”

But in this instance there are examples of things that can be considered when developing software, for example the electricity consumption (Agarwal et al., 2012). To explain what green software is Agrawal et al. (2012) calls it “simple”, which basically means that short and fast programs are more environmentally friendly since they limit the need for disk space. So in other words, shorter execution times limit electricity needs and shorter code needs less disk space and therefore less manufacturing. It is important that these problems are considered to limit the environmental impact of software, and therefore Agrawal et al. (2012) has created a list of nine concrete environmentally preserving actions to be taken in software projects:

a) To get the project team to use online forms and questionnaires

b) To create a standard set of ‘Environment Requirements’ and list to be added to the non functional requirement criteria in agreement with the client and tune the requirements based on the project being undertaken.

c) To define quantitative metrics that can be used as acceptance criteria for environmental feasibility of the project and to have a clear objective definition of environmental metrics as related to the technology in question needs to be defined, agreed, validated and used as a standard practice.
d) To reduce unwanted redundancy of data which would let more storage be available for future applications resulting in lesser hardware, which along with the environment aspects would be cheaper and easier to maintain. Duplication of data occurs in many systems where the same data is stored in different storages for different usages. On example is OSS and BI system where some duplication of data occurs due to the nurture of the problem. The design needs to look at such aspects and try to reduce unwanted redundancy. The requirement to devise smarter software to exploit parallel, multiprocessor architectures will inevitably mean more efficient use of hardware resources and better coding techniques all round.

e) To develop environment friendly codes. The developers should be taught to use the ‘environment friendly’ best coding practices. The challenge here is the bringing the efficiency into the code so that lesser system resources are utilized but the software should not get less performant in terms of processing speed, latency etc.

f) To define ‘green’ test cases with sample values, benchmark tests and feeding back the results of the tests to further improve the metrics over several iterations.

g) To Use system monitoring tools and benchmark the utilization of resources.

h) To focus on using the existing infrastructure as opposed to buying new hardware. In many companies most of the server capacities are grossly underutilized and yet new hardware is purchased. Over years, companies have purchased hardware on an ad hoc per requirement basis which has led to a huge bank of servers and storages. These not only consume more power and have massive cooling and maintenance requirements, an analysis will show that such a huge bank of hardware is not needed. Consolidation of multiple servers/storages into a fewer number of these can be planned and implemented.

i) To use of virtualized hardware/clouds could be an architectural step that would lead to shared reusable layer of hardware that can be commissioned and decommissioned in a matter of hours
and would lead to lesser power utilization and reduce ‘hardware garbage’. Also the use of ‘hosted service’ within an organisation would ensure that a large number of users can use a given software with considerably lesser energy utilization requirements.”

2.5 Thesis theoretical positioning
This thesis will attempt to look at software testing as multi objective based, minimizing the amount of software faults over a set amount of resources. In contrast to previous research the focus will be on the resource allocation for optimizing the software testing as a part of the test portfolio management strategy and the interaction from different types of testing within the portfolio. The research area of test portfolio management as a tool to minimize computer scientific issues and human errors, for example software bugs, has not been thoroughly studied and needs further research. To conclude, this thesis will be positioned between the larger areas of Industrial Management and Computer Science.

2.6 Definition of concepts
Following is a list of terms and concepts used in this thesis that require further explanation since their meaning may wary a bit from the norm. These are the definitions used by the author of this thesis.

Software bug
The term software bug is commonly used to describe a fault in a software system, for example Myers et al. (2011) defines it in their book The art of software testing, as a program that is not meeting the specification contains bugs. In this thesis a bug, or issue, will be defined as a miss in meeting the defined specification and that these bugs are reported in the bug reporting software that the production team has. In other words, bugs that goes unnoticed or bugs that are caught by the testing team will not be considered.
**Fixed issues**
The term fixed issues relate to the software bugs or new features that are being developed or fixed by a software developer.

**Release**
A release or software release is the action of releasing a new set of features or bug fixes, fixed issues, to the end user. The release usually contains several fixed issues that are supposed to service or increase the value of the software.

**Production**
The term production refers to the state of the software after a release, sometimes also known as post-development. In this state, the software is in full use and should function according to specifications, but since that is never the case, bugs are reported by the production team that is overlooking the software.

**Successful testing project**
The vision of any testing project should be a completely bug free release, but as stated earlier that is not practically possible. When looking at the success of testing project, we can gather that a release of software that fixes more bugs than it creates is a successful release. Therefore, when a release is released with more fixed issues than new bugs, it is a successful release since it should increase the theoretical value of the software. This definition will be needed when examining the results in this thesis.
3. Method

3.1 Research design

This thesis is designed according to the ideas of post positivism, that there is a theoretical correct way to do thing, but it might not perfectly map to the practical world due to knowledge biases. Compared to the old notion of positivism, post positivism moves away from the idea of “either/or, or black/white thinking” (Ryan, 2006).

Further the design of this thesis relies on the quantitative analysis to provide numerical data. The qualitative interviews will provide verbal data and this will later be triangulated using theoretical concepts from both managerial and computer scientific literature. Triangulation is originally a geometrical concept, but used as a metaphor within social scientific research to explain the concept of utilizing two or more points of reference within the theoretical framework as well as the own research to preserve the validity of the research (Thurmond, 2001).

The purpose of this thesis is in part exploratory and predictive (Blomkvist & Hallin, 2015, p.24), as further discussed in 1.2.2 Purpose, and therefore the method will also be divided in two. The first part is qualitative research to determine the adaptability and as a pre-study, find employee’s relation to the case and how it might affect them. The qualitative part of this methodology is exploratory, trying to further develop the knowledge around the case and the phenomenon under study. The second part is the quantitative research to determine the relation of test portfolio management to testing performance; this method is used to answer the predictive part of the thesis, were this thesis attempts to guide the managers towards a successful portfolio resource management.
### Table 1: summary on the parts of the thesis.

<table>
<thead>
<tr>
<th>Part 1 – Qualitative</th>
<th>More Qualitative, interested in how the managers of the portfolio work and consider their work as well as what strategic approaches they use.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part 2 – Quantitative</td>
<td>More Quantitative, interested in the data of how resources were finally committed to a projects as well as how successful the projects were.</td>
</tr>
</tbody>
</table>

Table 2: shows the theoretical fields of each part.

<table>
<thead>
<tr>
<th>Part</th>
<th>Portfolio Management</th>
<th>Software Testing Strategies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part 1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Part 2</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Each of the two parts end up in a theoretical field, the interview will have a broader perspective since different questions can be asked. The second part will end up in the Portfolio Management field since it will only regard the project data from the TSN portfolio and through that look at the allocations and success of projects within that portfolio.

#### 3.2 Part 1

**3.2.1 Qualitative methodology**

The qualitative approach in this thesis is formulated as explorative research to create a stable foundation of knowledge to help formulate the origins of the current test portfolio management strategy. In essence, this could be considered a pre study, together with the previous knowledge and observations the author has which is discussed further in 3.4 Authors relation to case. Due to the prior knowledge of the author step to increase the replicability of the thesis and decrease the impact of the authors previous knowledge will be discussed.
3.2.2 Semi-structured interviews

The interview will be conducted with the manager who, historically and currently, has set the resource allocation scope for the test projects for the TSN releases. The interview will feature open ended questions to receive information about current and historical strategies, as well as try to estimate the possibility of implementing new test portfolio management strategies.

The semi-structured interview is deemed fit to get knowledge about the decision processes concerning resource allocation of managers and getting enough information about how they plan projects. Structured interviews would be too strict and might miss some aspects from the decision process. The unstructured interview, or discussion, have been considered but might be too informal and unstructured and might lead to missing areas that needs to be covered.

Since there is only one manager currently in charge of this project, the interview will be conducted with that manager. This is regarded as enough because of the knowledge transfer existing between project managers within the project management office. This interview should not however be viewed as the complete truth of how all projects are allocated at Nasdaq, some of the information gained will be biased so only facts will be considered and opinions will rather be left out from the analysis, and these will be found by correlating with the authors prior knowledge.

3.2.3 Thematic analysis

To analyze the interview, this thesis will use a thematic analysis and use the reoccurring themes discussed in the interview and view this as the basis of analysis in Part 1 of the thesis. The idea of a thematic analysis is rather to find common themes in different ethnographic interviews (Aronson, 1995), but since this thesis doesn’t have enough people to interview, the focus will be with collecting information from the one person and then creating themes in relation to the author’s previous observations at the company. This will be considered enough to get the
required data and correlations to analyze thematically. These themes will then be correlated with theory to help promote the validity of the results, in a triangulation fashion.

3.3 Part 2

3.3.1 Quantitative methodology

The focus of this thesis is the quantitative approach, where data will be gathered and analyzed with mathematics and statistics to figure out guidelines for resource allocation. The focus of data gathering will be to get a large enough set of data to draw conclusions from and the analysis will be focused on creating models for that fit the data.

The problematic aspects with this approach to quantitative methodology are that either the data gathered could be too small to draw generalizable conclusions. Reflections on the reliability of the data will be necessary and conclusions cannot be based solely on the data gathered but needs to be triangulated with the qualitative research as well as with theory.

3.3.2 Data gathering

The data gathering in this quantitative study will be gathered from secondary data sources, already gathered data. Part of the data was intended to be used in the manner of quality improvement of the organization, this is the data tracking production and software issues, and it can therefore be viewed as accurately depicting the situation. The data about resource allocations from previous projects was gathered to track how much resources are put into each project, and can be viewed as accurate.

The data will be gathered from the systems in place at Nasdaq Inc.’s development, this thesis will have access to historical project data from the reporting system that Nasdaq uses to track resources in projects. There are two systems that will be used in the data gathering: Jira is the task tracking software used by Nasdaq Inc., where software development and verification tasks are reported and assigned to someone how either implements or tests a solution. Clarity is the
software tool used to track resource allocation for all projects at Nasdaq Inc., where each person is responsible to report their own work time.

There are three data lists that will be viewed and gathered:

- **Project resource allocation**, gathered with database queries from the Clarity database.
  - This data will be the reported time for each TSN release. Each release is usually represented by the month and year that it was on the form MMM-YY, i.e. JUN-15, SEP-16 etc. The time for each project is reported under a specific ID that corresponds to, for example, automated testing for a specific release.
  - This data was exported to an Excel file to facilitate access.

- **Development reported solves and issues found prior to the deployment**, gathered with database queries from the Jira database.
  - This data represents the number of issues solved for a specific release, i.e. they are either bugs found prior to a release or it is new functionality that is introduced to the system.

- **IIP (Issues in Production)**, gathered from database queries from the Jira database.
  - This data is gathered to show how many issues, software bugs, was discovered in production. These issues cause problems for the software system at a stage where it is released and therefore affects the customer negatively.

The benefits of this type of data gathering is that the data is already gathered and all that is required is access, which is a compelling argument for this method given the limited time for a master thesis project. The weaknesses are that the data could in theory be taken out of context and since it was not gathered especially for this thesis the data might not give a holistic view of the problem. These weaknesses will have to be addressed and it will be of great importance to connect findings to literature as well as the qualitative study to safeguard the validity.
3.3.2.1 Programming to view retrieve data

In order to sort and view the data in a complete fashion a couple of Visual Basic scripts were written, since the data was in large Excel files with hundreds of rows. Below the scripts can be viewed in pseudo-code for replicability.

i. **Code to retrieve the rows of interest.**
   
   For Each Row in Totals
   
   If (name contains “TSN” && name contains “Test”) {
      
      copy row to Results
   }
   
   Next row

ii. **Code to split time spent into projects.**

   For Each Row in Totals
   
   For Project
   
   If (Row belongs to Project){
      
      Add time to Project
   }Else { Next Project } Next row

After the data was sorted, manual work commenced to create graphs of the relevant data. These graphs will show the allocations of the projects correlation with bugs and fixed issues to determine the success of the projects.

3.3.3 Simple quantitative analysis

This thesis will use the data collected to track project costs, calculated in hours of work, within the test management portfolio and correlate the number of hours used for testing in separate projects and correlate them with the number of hours developed and the outcome of the project.
The number of hours worked on a specific project is stored in a data system called Clarity. The database was then exported to an Excel file to be used in this thesis. The file contains every hour reported by an employee on any project, which made it hard to work with and impossible to analyze manually. For the sake of saving time, a Java-program was written to read the Excel file, extract the relevant information and present it in a comprehensible fashion.

To create an analysis of the data collected, the data had to be correlated with three specific points; the number of hours put into the two project categories (manual and automated test), the number of bug fixes versus the number of IIP’s and the total number of hours tested per fixed issue. By correlating several points of data, the analysis will be deeper and attempt to showcase the outliers in the data. Upon getting this data, this thesis will attempt to make a qualified estimation towards what range of testing the best result is achieved. The estimation will be made in what percentage of testing resources spent on automatic testing receives the best result.

3.4 Authors relation to case
This thesis is performed at the author’s current employer, where the author has been working part time for three years. Therefore, there might be knowledge obtained and observations made that will be from the author’s previous experience at the company. Further, the author is employed as an automatic tester, which could create a bias towards that type of testing as well as having more extensive knowledge within that field than within manual testing. These prior affiliations should be considered when reading and discussing this thesis, and to solve this the author will be keeping as an objective mindset as possible.
4. Result

4.1 Interview

From the interview, it is clear that the resource allocation in the test portfolio is not set in stone, it is more a give and take between test managers. You help someone when they need it and they will do the same for you (Interviewee, author’s translation). So, to begin with, the project has a plan for how the resource allocation should look, but that is not the exact way it will end up.

The testing strategies that are used are risk-based; when a change is made there will be more testing in that system, or closely related systems. When bugs are found, regression testing will be done in the future to see that the same bug doesn’t occur multiple times in different release.

The test manager stated that the organization is very adaptable when it comes to test portfolio management. The project management office exchange knowledge and try to optimize their approach after projects. What is needed to take into consideration is the fact that in a release there are several different components and the testing skills of both manual and automated testers only go so far, in other words it is easier to test the same system that you tested in the last release than to learn a completely new component.

Another point that the interviewee brought forward is that, if possible, the automatic tester will benefit from learning how to test a system manually before developing automatic tests. This view shows a learning curve for automatic tester that means learning both professions rather than starting with atomization of manual testing sequences.

In conclusion of the interview the task manager stated that the PMO and the organization was always in need of new data on how to optimize production, even if the results would only provide a basis for discussion amongst managers.
4.2 Project release data

4.2.1 Resource allocations per release

Below is the resulting set of resources allocated into the projects under study. From left to right the columns represent the total resource expenditure, the release, the automatic resources, the manual resources, the percentages for both automatic and manual testing, and the quote of automatic to manual.

<table>
<thead>
<tr>
<th>Total</th>
<th>Release</th>
<th>Auto</th>
<th>Manu</th>
<th>Auto (%)</th>
<th>Manu (%)</th>
<th>Quote</th>
</tr>
</thead>
<tbody>
<tr>
<td>2365,47</td>
<td>201403</td>
<td>199,7</td>
<td>2165,77</td>
<td>8.442%</td>
<td>91,558%</td>
<td>0,092207</td>
</tr>
<tr>
<td>789,55</td>
<td>201406</td>
<td>26,8</td>
<td>762,75</td>
<td>3.394%</td>
<td>96,606%</td>
<td>0,035136</td>
</tr>
<tr>
<td>1233,8</td>
<td>201411</td>
<td>99,8</td>
<td>1134</td>
<td>8.089%</td>
<td>91,911%</td>
<td>0,088007</td>
</tr>
<tr>
<td>1229,35</td>
<td>201504</td>
<td>658,9</td>
<td>570,45</td>
<td>53,597%</td>
<td>46,403%</td>
<td>1,155053</td>
</tr>
<tr>
<td>166,6</td>
<td>201506</td>
<td>37</td>
<td>129,6</td>
<td>22,209%</td>
<td>77,791%</td>
<td>0,285494</td>
</tr>
<tr>
<td>120,25</td>
<td>201509</td>
<td>33</td>
<td>87,25</td>
<td>27,443%</td>
<td>72,557%</td>
<td>0,378223</td>
</tr>
<tr>
<td>184,7</td>
<td>201511</td>
<td>17,7</td>
<td>167</td>
<td>9.583%</td>
<td>90,417%</td>
<td>0,105988</td>
</tr>
<tr>
<td>251,75</td>
<td>201602</td>
<td>47,4</td>
<td>204,35</td>
<td>18,828%</td>
<td>81,172%</td>
<td>0,231955</td>
</tr>
<tr>
<td>303,4</td>
<td>201604</td>
<td>116,2</td>
<td>187,2</td>
<td>38,299%</td>
<td>61,701%</td>
<td>0,620726</td>
</tr>
<tr>
<td>556,7</td>
<td>201606</td>
<td>158,1</td>
<td>398,6</td>
<td>28,399%</td>
<td>71,601%</td>
<td>0,396638</td>
</tr>
<tr>
<td>517,16</td>
<td>201609</td>
<td>312,36</td>
<td>204,8</td>
<td>60,399%</td>
<td>39,601%</td>
<td>1,525195</td>
</tr>
<tr>
<td>471,1</td>
<td>201702</td>
<td>89,4</td>
<td>381,7</td>
<td>18,977%</td>
<td>81,023%</td>
<td>0,234215</td>
</tr>
<tr>
<td>8189,83</td>
<td>1796,36</td>
<td>6393,47</td>
<td>21,934%</td>
<td>78,066%</td>
<td>0,280968</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: The historical release data.
The above graph shows how many resources was spent on automatic and manual testing. As mentioned in the methodology, automatic testing represents creating and maintaining automatic tests as well as developing test tools that can further increase the efficiency of manual testing whilst manual testing represents the creation and running of manual test sequences. To calculate the total amount of testing resources expended, that will be the integral of the two curves.

4.2.2 Software bugs

4.2.2.1 Software defects

Software defects are known software bugs, or other unintentional behavior, that have been reported in the Jira system and they are going to be fixed in a coming release.
Figure 5: shows all unresolved issues for a release, i.e. remaining bugs that were not attended to.

Figure 6: This picture shows the number of issues in production for each release.
In figure 6 shows the number of issues detected in production per release. This is a graph of raw data and the data needs some fixing before it can be perfectly understood. First, the data is total number of issues detected, and this interesting and highly relevant, for the purpose of this thesis it would be more relevant to look at defects detected per a time unit. This would give a better picture of the stability of the release, because of the alternating time span of the releases, of course, more issues will be detected in a release running from November 2014 to April 2015 (5 months) than in a release running from September 2016 to November 2016 (1 month). Second, we would like to correlate this data with the data in the figure 6, which shows the information of data detected per release compared to the size of the release, the number of defects fix compared to the number of defects introduced. Although there are clearer data for what bugs were software related bugs; these are all the issues found in production, the data used later can be subject to the number of reported actual software bugs, and this is because some of the bugs represented in this graph will be bugs caused by hardware that is not under the oversight of the portfolio under observation. As seen in the graph below, a lot of issues in production are not under the responsibility of any sub system and might therefore not be tested in the testing projects.
Figure 7: This pie chart shows the allocation of issues in production relative to systems.

This above chart shows the number of issues that occurred in the system, showing the latest 1300 issues and dividing them into in which subsystem that the issue occurred. The other category can be issues that are allocated outside of a subsystem, for example non-functional issues. This data provides an oversight of where different issues occur, for example resolving issues in one part of the system might be more likely to cause issues than issues in another system. Therefore, this
data should be considered in relation to the number of resolved issues in every system, since the more changes to the code in a system the more likely a bug is to occur.

4.3 Correlation between software bugs and allocations

The below table, table 4, contains the data for how many bugs were found versus how many issues were solved in each of the releases, the releases are here named by 6 figures. The first four figures represent the year and the last two numbers corresponds to the month of that year that the release was taken into production.

<table>
<thead>
<tr>
<th>Release</th>
<th>Fixed issues</th>
<th>New Bugs</th>
<th>Quote (fixed/new bugs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>201403</td>
<td>83</td>
<td>72</td>
<td>0.86747</td>
</tr>
<tr>
<td>201406</td>
<td>98</td>
<td>85</td>
<td>0.867347</td>
</tr>
<tr>
<td>201411</td>
<td>115</td>
<td>97</td>
<td>0.843478</td>
</tr>
<tr>
<td>201504</td>
<td>94</td>
<td>100</td>
<td>1.06383</td>
</tr>
<tr>
<td>201506</td>
<td>64</td>
<td>35</td>
<td>0.546875</td>
</tr>
<tr>
<td>201509</td>
<td>94</td>
<td>76</td>
<td>0.808511</td>
</tr>
<tr>
<td>201511</td>
<td>105</td>
<td>95</td>
<td>0.904762</td>
</tr>
<tr>
<td>201602</td>
<td>37</td>
<td>66</td>
<td>1.783784</td>
</tr>
<tr>
<td>201604</td>
<td>61</td>
<td>50</td>
<td>0.819672</td>
</tr>
<tr>
<td>201606</td>
<td>46</td>
<td>42</td>
<td>0.913043</td>
</tr>
<tr>
<td>201609</td>
<td>12</td>
<td>20</td>
<td>1.666667</td>
</tr>
</tbody>
</table>

Table 4: Shows the number of bugs and fixed issues
Figure 8: Showing the correlation between two ratios, blue is the ratio of fixed bugs

Figure 8 show the correlation between different ratios of resources spent on automatic versus manual tests together with the ratio of new software bugs over fixed issues. The results showcase that the top three results have got a percentage of automatic testing at 30%±8 and the following four have a lower ratio, but several times more testing hours put in. After that, the release of June 2016 has also got a ratio of less introduced bugs than fixed issues as well as an automatic test percentage of within the ratio of 30%±8. Three last points of data, they score above one new bug for every new fixed issue and they are not in the interval of 30%±8, they had 54%, 19%, and 60% of resources spent on automatic testing.
5. Discussion

5.1 Part 1 – Thematic analysis

Portfolio management
In a large company portfolio management is very important, which was shown in the interviews where all the interviewees put a high amount of their time into planning the resource allocations within projects and also spent a lot of their time reallocating assets between projects.

Risk-based testing
The risk-based testing discussed in the interview is closely related to the ideas of Amland (2000); where the idea is to test the component that is at the highest risk of failing. Then ordering the testing based on what would be most likely to fail and what provides the highest damage if it fails. For example, it might be a promising idea to always test the entry of orders in a stock market technology since if there is a software bug there it would affect the entire purpose of the software.

Regression testing
Regression testing was also discussed in the interview, and there are several theories regarding how to regression test software bugs. From the authors observation, the theories about selective regression testing from Leung and White (1991) are applied at the company. For example, automatic test cases that were developed several years ago might not be used currently because the system doesn’t change in that part, but the tests are kept in storage for when that part of the system changes and therefore becomes a risk in the risk-based testing.

Resource “trading”
The idea that unused resources are constantly reorganized and “traded” between different projects is promoted in Computer Scientific testing strategies, such as selective regression testing and
risk-based testing (Leung & White, 1991; Amland, 2000). In management theory, more exactly portfolio management, the reallocation of resources is a central area.

5.2 Part 2 – Statistical analysis

5.2.1 Issues in productions

Looking at the data from figure 6, from section 4.2.2.1 Software defects4.2.2 Software bugs, we can see that the number of production issues was generally higher in 2015 than 2016. This is most likely related to the decrease in present software defects which can be seen in figure 5, from the same section. With these two graphs we can see that the number of issues in production is often closely related to the golden ratio discussed below, 5.2.1 Issues in productions. For reference sake, all the software defects from figure 5 were not fixed, that they are not present in that graph any more can also be from redundancy, like when a subsystem is decommissioned.

5.2.2 The golden ratio

According to the data presented in heading 4.3 Correlation between software bugs and allocations, there is a clear indication that in a specific interval, the projects get optimized both in terms of resources spent as well as in bugs that have been let into production. Looking at the data, the interval where testing project has been most successful is 30%±8 percent of automatic testing out of all the testing. That means that the ratio between automatic testing and manual testing has been 30:70 (±8). This interval has been clear in the data gathered, the topmost three projects have all had resource allocations in that range. It could be argued that the following three projects are also successful regarding the few bugs found in production, and their allocations where more towards the manual side (5%±4), but these three projects utilized more than half of the resources spent on testing during these three years, 2014-2016. In other words, to test the release with mostly manual testing is more expensive in regards to time, and this idea is supported by Ramler and Wolfmaier (2006) who in their thesis showed that there is times where tests should be automated instead of run as manual time consuming tasks. Within these resource
heavy projects tests, that probably should have been automated, have probably been run several
times as manual tests, which would have explained why the project ended up being expensive.

5.3 Semi-automatic testing

One issue with this thesis is that the allocations within the testing projects are allocated within the
projects of Automatic and Manual testing, since this is not the entire truth of how the resources
work. There is a third category of testing called test tool development at Nasdaq, but for the sake
of theory we should probably call it semi-automated testing. This category of testing is
unprecedented in previous literature, which focuses on differences between manual and
automatic testing. Semi-automatic testing is the act of programming tools that can be used by the
manual testers and it increase efficiency of the manual testing resources. A typical example for
Nasdaq could be that a manual tester needs to enter several trades in a manual test scenario, this
act of adding several trades might take a fair amount of time, but with a specified test tool, that is
specifically used to find companies and traders that can trade the instruments that the manual
tester needs, the time required for this test might move from minutes to seconds. This testing
category is allocated within automatic testing, since automatic testers have got previous
knowledge from programming. Looking at the figure 2 under heading 2.2 Manual and automatic
testing, the graph from Ramler and Wolfmaier (2006), it is simple to see that a third equation is
needed to fully explain the connections. The third line is hard to add to the graph since we do not
know how much time the semi-automatic development would take, it would have a higher initial
cost than the manual tests but there is no obvious distinguishable correlation to the automatic
testing. So, the area of semi-automatic testing is in need of further research, especially on the
economical side such as the research of Ramler and Wolfmaier (2006) with automatic and
manual testing.
5.4 Triangulation of Part 1, 2 and Theory

From the previous section, 5.2.2 The golden ratio, it can be seen that a specific allocation of resources is indicated to give increased testing performance. This is supported by portfolio management research, for example Cooper et al. (2001), who states that portfolio management is important “to properly and efficiently allocate scarce resources”, which is precisely what this thesis shows: properly allocated resources add to the success of the testing projects. Further, the interview showed that the managers are willing to adapt to and wanting new guidelines for the resource allocation within projects. Considering both the parts of this thesis and theory, it shows compelling evidence that it is possible to use portfolio management to optimize the complex task of testing software.

5.5 Effect on sustainability

The sustainability aspects of this management case can be divided into three categories; Social, Environmental, and Economical.

First, since this thesis objective is to optimize processes the economic benefits are clear (Ramler & Wolfmeier, 2006; Ramler et al., 2006). If there is a way to optimize the resources for a process this either will be beneficial because: If it finishes the process faster, enabling resources to allocate elsewhere without diminishing returns, or the process will be able to be more thorough, making resources more effective. This will be the main effect on sustainability from this case.

Second, this thesis focuses on the allocation of human resources within test portfolio management. Increasing the effectiveness of the resources allocated might be beneficial for the environment as every hour put into a project means that a computer is most likely running, since the testing of software requires a computer to be active no matter what. It will be hard to find a numerical for the amount that this effects the environment, but we know that if we decrease the amount of time on a project this will decrease the environmental footprint by that project. This
goes into the ideas of Agarwal et al. (2012) who stated that “simple” with low execution time is better for the environment, but in this case, it also goes for the manual testing since a computer is used for every task there as well.

Finally, if we discuss the social aspects of the results these are most likely low. By optimizing the number of hours allocated to different projects the argument could be made that this would diminish the position of the employee since the employee might not have enough work to fill their days. This scenario is possible if the organization has a perfect ratio of resources to work needed, although this is not the case at Nasdaq and from both the interviews and the observations from the author, there is clear evidence that they are a bit understaffed. But even though this is not the case at the current object under study does not mean that this will not be a pressing issue when attempting to apply this thesis in a different setting.

5.6 Sources of error

5.6.1 Dialog interpretations
The semi-structured interview where held with one manager at the company, they were recorded and analyzed. To minimize the chance of any answers being misunderstood the results where then checked with the interviewees to see that the author interpreted the answers correctly and to see that the interviewee was satisfied with the depiction of their answers. First after this feedback process the answers where further analyzed and the analytic themes where detected. This feedback loop is believed to minimize the issues that author interpretations of the answers could have as well as minimizing the misunderstanding of questions.

5.6.2 Data gathering and analysis
The data gathered is not specifically gathered for the purpose of this thesis, but is gathered for monitoring the project’s resource allocation and its outcome. Since the purpose of this thesis is to analyze the outcome and internal resource allocations, it is believed that the data will be
sufficient, and since the data is the only historical depiction and is gathered at the time of the project it is deemed accurate enough for this thesis.

The analysis of the data takes into account every release for a specific customer during the past four years, therefore it is deemed enough data to draw some valid conclusions on. The error of the calculations could be significant since there is a relatively small set of data, if looking at the quantitative analysis literature, but it will be enough to give an indication of the how the perfect resource allocation should look.

The sample size of the data is relatively small and this might cause the answer to be off in the project. It would have been beneficial to have access to a greater amount of data, which would have given this thesis a statistical relevance that it is now lacking. To view the result and analysis as statistically valid is wrong, this should be interpreted as a case study of projects within a specific company and the learning outcome from this thesis is that this can be monitored and this is one way to track and evaluate projects within the test management portfolio and that this does probably affect the outcome of test management.

5.6.3 Estimation of testing success
The data that was gathered to estimate the success for the projects was regarding the different software bugs found in production and how many issues that were fixed by the programmers. This estimate was the best that could be achieved under the circumstances. It is still a strong candidate to estimate the success, but it could be argued that the number of bugs found before production correlated with the number of bugs found in production might be a better estimate of testing success. However, this data is not clearly enough stated and not monitored thoroughly enough to be representative of the environment under study. In future studies, the testing projects should be monitored more closely and it would be beneficial to look at several different estimation for their success before deciding if the projects are successful or not.
5.6.4 Authors connection to company
As previously stated, in section 3.4 Authors relation to case, the author is connected to the company and has partaken in some of the projects under study. This could have caused the author to be biased in his knowledge, although the quantitative results are rather self-explanatory and every manipulation of the numerical data is clearly stated in the method and results. Because of this, this is not considered a problem within the quantitative study. The relation of the author to the company would make the interviews easier to conduct and the previous relationship to the interviewees should be considered since it could cause the interviewees to speak more plainly and it could be easier for the author to get the answers he sought after.
6. Conclusion and Recommendations

6.1 Research questions

Here is a review of the research questions from heading 1.2.3 Research questions.

RQ: How do test portfolio management strategies affect the outcome of test processes and do these strategies have a measurable effect on the number of bugs found in post development?

And here are the sub questions:

SQ1: Is there a correlation between resource allocation and bugs found in post development?

SQ2: Could portfolio management strategies be applied to testing of software to minimize bugs or resource usage?

SQ3: Is management susceptible to changes in test portfolio management?

Viewing the research questions, in light of the analysis we can draw a set of conclusions, starting with the sub questions.

First, looking at sub question one, SQ1; we can from the analysis see that there is a clear indication for a resource allocation that is clearly more stable than the others. This specific allocation is 30%-8 of testing resources spent on automatic testing.

Second looking at sub question two, SQ2; the analysis from part 2, 5.2 Part 2 – Statistical analysis, indicates that since there is a specific allocation it is likely that resource allocations have got an effect on the outcome. This is shown in the analysis, 5.2.2 The golden ratio, and supported by theories, for example from heading 2.3 Test portfolio management and 2.2 Manual and automatic testing (Wang, 2010; Ramler & Wolfmaier, 2006).
Third, looking at sub question three, SQ3; the analysis from part 1, 5.1 Part 1 – Thematic analysis, is indicating that the project management office is adaptable and the applied strategies for resource management are adaptable enough to facilitate new guidelines and discussion material.

So to conclude, these sub questions and answer the main research question, it is indicated that from the analysis and the theoretical material that there is a connection between resource allocations within the test portfolio and that it has a measurable effect on the outcome of the projects. This however does not mean that the indicated allocation is the correct one - the focus of this thesis is the quantitative methods used and the simple statistical analysis made of the projects that indicate gave the results and the analysis. The method is what should be used, researched further, and evolved: to be able to draw more distinct conclusions.

6.2 Managerial Implications

From the qualitative interviews, 5.1 Part 1 – Thematic analysis, we saw a need for more guidelines that can evolve the discussion regard how to allocate resources and this thesis can fill that void of discussion material. The results from the quantitative statistical analysis, 5.2 Part 2 – Statistical analysis, the managerial implications are clear. The golden ratio for projects in the TSN test portfolio have a best practice for resource allocation that increase the effectiveness of the resources spent, either by reducing the need for resources or simply finding more of the issues before production. As discussed, a bit later, in 6.4 Interpretation of recommendations, this thesis is not irrefutable proof that this ratio is the best one and the thesis should not be interpreted as such.

This thesis provides support for further discussion about the how to allocate resources within the test portfolio.
6.3 Recommendations for “golden ratio”

The golden ratio of 30%+-8 of resources spent on automatic testing should be considered when setting the allocations for projects within the portfolio, but only as a basis for discussion. This is because of the low amount of data gathered, as well as being currently only concretely applicable to projects within the TSN test portfolio that contained the projects under review.

The recommendation regarding how to achieve a better portfolio management is that test managers should maintain a close watch on the resources spent on each testing project and keep correlating these with the number of bugs found in production, the more data gathered the better the ratio will be specified. The method used in this thesis is an easy way to analyze how the resource allocations interact within a portfolio. Because of the difference between different portfolios might be bigger than expected in form of resources that are possible to allocate. How closely the portfolio management strategies for different testing projects in different portfolios can mimic each other should be further investigated.

6.4 Interpretation of recommendations

The fully compiled report should be interpreted as a basis for discussion prior to setting the resource allocation within the testing projects for a specific release. It should be obvious that there are of course other problems or issues that can take president over the estimated golden ratio from this thesis, but this thesis could give a valuable hint to the managers. Further, the goal of this thesis was not to find the perfect ratio of automatic testing to manual testing but rather to provide the insight that these allocation choices made by the test managers within the test portfolio could have a significant effect on the outcome of the project and therefore should be closely monitored in order to fully optimize the usage of human resources within the test portfolio.
Further, there might be issues because this thesis looked at the number of fixed issues and the number of introduced bugs. This means that, in this thesis, each fixed issue is regarded to be of the same size and introduce about the same number of bugs. In a perfect world, the number of bugs found by the test project and the number of bugs in production would have been viewed.

6.5 Further research

The need for further research within the field of test portfolio management within software development is clear; this thesis is a first step toward the resource optimization within the testing portfolio. Future research need to include areas of how closely the test management and portfolio management can be transferred within testing portfolios that test closely related products. Methods should be researched to find strong models to measure how productive testing projects are, how these test project’s resource allocation should be measured, and when its resources should be reallocated.

Further, the effects of semi-automatic testing should be researched to see a distinguishable difference from the old view of testing being either automatic or manual. This would be interesting because a change in the view of how testing of software should be done might increase the effectiveness of the entire test portfolio.
References


**Appendix A – Tables of information**

Appendix A contains two tables of all pictures and tables in the report, with their page number and the description of what they display.

**A1. Table of figures**

<table>
<thead>
<tr>
<th>Picture</th>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>8</td>
<td>The different stages of software development and where the corresponding testing stages appear (Myers et al, 2004, page 94).</td>
</tr>
<tr>
<td>Figure 2</td>
<td>11</td>
<td>The cost of testing per number of test runs with the cost of manual and automated testing. (Ramler &amp; Wolfmaier, 2006).</td>
</tr>
<tr>
<td>Figure 3</td>
<td>14</td>
<td>The three sustainability aspects and their intersections represented as three circles (Sikdar, 2003).</td>
</tr>
</tbody>
</table>
Figure 4  32  Plot of manual and automatic resource expenditure for projects from 2014-2016.

Figure 5  33  Shows all unresolved issues for a release, i.e. remaining bugs that were not attended to.

Figure 6  34  This picture shows the number of issues in production for each release.

Figure 7  36  This pie chart shows the allocation of issues in production relative to systems.

Figure 8  38  Showing the correlation between two ratios, blue is the ratio of fixed bugs.

### A2. Table of tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>21</td>
<td>Summary on the parts of the thesis.</td>
</tr>
<tr>
<td>Table 2</td>
<td>22</td>
<td>Shows the theoretical fields of each part.</td>
</tr>
<tr>
<td>Table 3</td>
<td>31</td>
<td>The historical release data.</td>
</tr>
<tr>
<td>Table 4</td>
<td>37</td>
<td>Shows the number of bugs and fixed issues</td>
</tr>
</tbody>
</table>