MÄLARDALEN UNIVERSITY
SCHOOL OF INNOVATION DESIGN AND ENGINEERING
VÄSTERÅS, SWEDEN

Thesis for degree of bachelor's in computer science 15.0 hp

FINDING BAD SMELLS IN NATURAL LANGUAGE TEST SPECIFICATIONS USING NALABS

Anas Aboradan
Aan19017@studen.mdu.se

Josef Landing
Jlg19004@student.mdh.se

Examiner: Wasif Afzal
Wasif.afzal@mdu.se
MÄLARDALENS UNIVERSITY, VÄSTERÅS, SWEDEN

Supervisor: Eduard Enoiu
Eduard.paul.enoiu@mdu.se
MÄLARDALENS UNIVERSITY, VÄSTERÅS, SWEDEN

2022-06-13

Abstract

Tests are important artifacts in the software development process. Testing activities such as test automation, test maintenance, and test suite optimization mainly rely on an in-depth understanding of test specifications. The manual process of writing test specifications in natural language can create many different quality issues such as ambiguous, incomplete, redundant, or inconsistent test cases. Nowadays, the concept of test smells is proposed by several researchers to be used as indicators of low-quality attributes in test specifications. Quality assurance processes for test specifications often rely on manual reviews to detect these smells. The manual process of detecting these smells is considered time consuming and costly. However, there is currently no work that implements a comprehensive quality model that classifies and identifies these smells by using a systematic strategy. As a result, there is a need for machine-supported analytical measures that decrease the time and effort needed to detect these smells manually, especially when it comes to reviewing and validating large test specifications.
This study aims to investigate which natural language smell metrics implemented in NALABS can be found in test specifications and to measure the sufficiency of those smell metrics. It also aims to extend these smell metrics by exploring, proposing, or combining with new bad smell metrics to cover more aspects of natural language test quality. The results of the study show that the smell metrics exists in real-world test specifications and can uncover many potential quality issues by assisting test designers in identifying certain types of quality issues pertaining to for example the understandability and complexity of test specifications. Moreover, the results show that the list of smell metrics implemented in NALABS is incomplete and can be extended to cover more aspects of test quality.

Table of Contents

List of Figures

List of Tables

1. Introduction

Nowadays, testing is considered as one of the most used methods for determining if a software system satisfies its requirements. The main purpose of testing is to execute a software system in many ways to ensure that the system under test works as expected and is free from errors [1]. A natural language test specification is a common way of describing a set of test cases or test requirements that is needed to create a comprehensive test strategy [2]. The process of writing test specifications manually can create many different quality issues, such as ambiguous, incomplete, redundant, or inconsistent test cases [1], [3].

Quality assurance of test specifications is still largely performed through manual reviews [3], [4], and there is currently no work that implements a comprehensive quality model that classifies and identifies potential quality issues in test specifications by using a machine-supported analytical measure [4], [5]. Therefore, manual reviews are often the only option to check natural language test specifications for quality issues [3], [5]. On the other hand, using machine-supported analytical measures for the evaluation and improvement of natural language test specification quality is very uncommon [5].

Several papers use the concept of bad smells as indicators to identify poorly written natural language test artifacts [1], [3], [4]. B. Hauptmann et el. [4], claim that their work is the first to study test smells in natural-language system tests. They proposed a list of bad smells based on their experience of working in the software testing area. Enoiu and Rajkovic [3] extended a list of bad smells that are used to detect defects in requirements specifications to other metrics related to complexity. According to the authors [3], their proposed smells can be used to detect bad smells in both natural language requirements and test specifications.

This work mostly focused on combining existing natural language test smells introduced by several papers in a single index of quality. The purpose was to create a comprehensive quality model that makes manual quality assurance of test specifications significantly faster and more comfortable.

Our research objective was first to investigate whether the automatic analysis NALABS[1] of natural language bad smells [3] can be applied to detect potential quality issues in test specifications. To reach this objective we conducted an exploratory case study to verify the existence of these smell metrics in the real-world test specifications and to measure the sufficiency of these smell metrics. Secondly, our goal was to investigate the literature to explore, propose or combine the current list of smell metrics [3] with new bad smell metrics to cover more aspects of natural language test quality.

The results of this work show that the natural language smell metrics implemented in NALABS can uncover many potential quality issues in test specifications by providing pointers to certain locations that should be inspected for defects. In addition, this work could combine the natural language test smells [3] with other smell metrics in a single index of natural language test quality [1], [4].

NALABS is mostly directed to companies that are interested in such machine-supported analytical measures that can work as complement to their manual reviews to discover defects in natural language test artifacts in an early stage of its quality assurance process. Companies can use NALABS practically to improve the quality assurance of their test artifacts, and they will have the knowledge that is needed to adapt and improve it.

The study starts with the background (Chapter 2) where we introduce all necessary concepts that the reader needs to understand the study. Then comes the related work (Chapter 3), in this section we present several papers that have worked with similar problems. After that, comes (Chapter 4, 5), we present the problem that the study will address, and the methods we will use to achieve the study purpose. We also discuss the ethical aspects in (Chapter 6) and describe in detail what we perform and how to answer the study's research questions in (Chapter 7, 8). The results of the study are analyzed and presented in (Chapter 9). Then we discuss the threats to validity of the study's result in (Chapter 10) Finally, we discuss the complete work in (Chapter 11), draw the conclusions in (Chapter 12), and suggest one future work related to our study in (Chapter 13).

[1] NALABS is a desktop WPF applications that depend on standard .NET packages and can be accessed at: eduardenoiu/NALABS: NALABS is a requirement quality checker for natural language requirements. It uses a set of bad smells to indicate problematic requirements. (github.com)

2. Background

The background section describes all the concepts that the reader needs to have a better understanding of our study. In this section we will cover requirements engineering in general, and software testing and its stages in detail. At the end of the section, we will cover the concepts of smells and their usage in the software testing context.

2.1. Requirements Engineering

Requirements engineering is a branch of systems and software engineering that contains all project activities related to determining a product's required attributes and capabilities [6]. The initial stage of every software project is requirements engineering. This step is critical because requirements engineering ensures that the product specification fulfills the customers' wishes. Errors in requirements engineering may result in project failure or need for correction in later phases, which is far more expensive than correction during the requirements engineering phase [7].

R.R. Young [8] defines a requirement as an initial attribute in a system, and it is a statement that specifies the characteristics, capacity, or quality aspects of a system for it to be valuable and useful to a client or a user. I. Sommerville and P. Sawyer [9] define it as a set of specifications that describe what should be implemented in a product to achieve its purpose.

The process of documenting the project requirements is known as requirements specification. This process aims to produce a document that contains all the project requirements in a manageable, sharable, and structured manner [6]. Requirements specification documents are often written by developers, requirements engineers or user/customer in natural language, and sometimes it can contain a combination of graphs, symbols, and diagrams [10]. The document will afterwards be used as a baseline and a guideline for all participants in the development of a product [11]. Therefore, the requirements specification document should describe all the projects functional and non-functional requirements and must be validated before it is delivered to those who will design and build the product [6]. During the process of validating the requirements specification, the engineer should ensure that the requirements specification follow quality standards and on a large scale is free from ambiguity, conflicts, and omissions [9].

2.2. Software Testing

Software testing is a method for determining if a software system meets its requirements, and to discover in which situations the behavior of the software is incorrect. Furthermore, software testing aims to ensure that the software system is free from defects and works as expected. In practice, software testing often includes at least one test for each requirement in the requirements document and is typically performed either manually or automatically [12, pp. 206].

2.2.1. Test Specifications

A natural language test specification is used to describe a collection of test cases or test requirements that are necessary to implement a comprehensive test strategy [2], [13]. For test cases, the test specifications are used to describe each test case step as an action and its corresponding expected result as well as test data to be used [1], [2], [4]. One example of a test case is manual system tests (*as shown in* **Table 1**) [1], [4]. In the test requirement the test specifications are used to describe a set of requirements that a set of test cases must cover and satisfy [13]. The testers use the test specification to create test scripts for automated testing or for performing manual tests [2]. Therefore, the authors of [2] claim that a high-quality test specification is required to ensure that the testers who perform the test cases, must understand, implement, and execute the test cases exactly as the test designers intended.

2.2.2. Manual and Automated Testing

A manual test is a test written in natural language and is carried out by a human without any significant tool support. The testers execute a test using artificial test data and compare the test result to their expectations [12, s 210]. In other words, all inputs, output analysis, and assessments are done manually in this test [14]. B. Hauptmann et al. [14] define a manual test as a series of commands, including input and output data that are used to accomplish a specific task with a software system, and to confirm the correctness of the system's behavior. A manual test usually consists of several steps written down in a table (*as shown in* **Table 1**). Each row in this table represents one step that has two parts: a description of an action and its corresponding expected outcome. The action description explains what the testers must do to complete a certain task with the software system. It may also include necessary input data and relevant background information. The expected outcome explains how the testers should validate the system’s response. If necessary, it also provides the relevant output data.

The testers must perform all steps in the test sequentially, beginning from the first step to the last step. After all test steps have been executed and the expected results have been confirmed, a software system have successfully passed the test.

Table 1. *Example of manual system test (test case).*

	Action	Expected result
1.	Click on the login button.	The system shall ask for username and password.
2.	Enter a random username and password.	The system shall respond that the username or password is incorrect.
3.	Enter the correct username and password.	The user shall be logged into the system.
4.	…	…

In contrast to a manual test, an automated test, refer to the process of automatically running test cases [14], [12, pp. 210]. This requires the use of test scripts that can be executed automatically without manual interaction by a human [1]. Manual tests are more often used than automated tests because automated tests are considered expensive and do not pay off in all situations [14]. For instance, automated tests cannot be used to test the appearance of for example a graphical user interface [12, pp. 210].

2.2.3. Test Suite

From a practical point of view, a test suite is a collection of test cases that are often written in natural language [4]. Any execution of the test cases in the test suite should result in a test judgment indicating whether the software system passed or failed the test cases [15].

2.2.4. Stages of Software Testing

According to Ian Sommerville [12, pp. 210] software testing can be divided into three categories: development testing, release testing, and user testing.

- **Development Testing**

The development testing is carried out in three levels and is performed by the developers [12, pp. 210].

1. **Unit Testing** is the first level of development testing. The test is concerned with testing individual units of a system. The purpose is to ensure that each unit such as method, class or object in the software works as intended [12, pp. 211], [16].
2. **Component/Integration Testing** is the second level of development testing**.** At this level of development testing, individual units are integrated in a planned manner using an integration plan. The purpose is to test the integrated units as composite components. This level is designed to detect faults in the internal interaction of integrated units, and the test mainly focuses on testing component interfaces such as parameter passing between units [12, pp. 216], [16].
3. **System Testing** is the last level of development testing and is concerned with testing an entire system based on its requirements. The testing includes several activities such as functional testing and performance testing. The aim of functional testing is to ensure that the system functions meet its behavioral description, while performance testing aims to test response time and resource utilization of a system [12, pp. 219].

- **Release Testing**

The practice of testing a specific release of a system that is designed for usage outside of the development team is known as release testing. In other words, the release testing is designed for users and customers. The main aim of release testing is to ensure that the system meets its functional and non-functional requirements, and that the system is ready to be put in general use. In case the system satisfies its requirements, it can be delivered as a product to the consumer. As a result, release testing should show that the system performs as expected in terms of functionality, performance, and reliability, and that it does not fail during normal use [12, pp. 224].

- **User Testing**

End-users or potential end-users testing a system in their own environment is referred to as user testing. The user can be the customer of the system and can carry out what is commonly referred to as “acceptance testing”. Acceptance testing aims to give the customer a way to formally evaluate the system to determine whether the system can be accepted or if it needs additional development [12, pp. 224].

2.3. The Concept of Smells in Software Engineering

Fowler and Beck [17] introduced in their book the concept of bad code smells to determine when quality of code is low, and refactoring is needed. According to Fowler and Beck, bad code smells usually arise as the result of incorrect implementation of coding concepts, such as applying urgent fixes or making suboptimal decisions. Fowler and Beck [17] listed 22 bad smells in their refactoring book, along with corresponding refactoring techniques. To detect these bad smells in code can be difficult according to Fowler and Beck [17], but there is certain concrete, visible symptoms such as *hard coded values, large classes, long methods, lazy classes, or duplicated code* that one can look for to detect these smells. Furthermore, they mention that the process of detecting these smells in the code sometimes requires specific domain knowledge and experience. Zhang et al. [18] conducted a comprehensive examination of the state of the art in code smells. Since the concept of code smells turns out to be a concrete symptom, it has been transferred by several researchers to be used in context of quality for other various artifacts. For instance, B. Hauptmann et al. [4] has used the notion of smells through the term smells for natural language system tests, and H. Femmer et al. [19] used it in requirements specifications as requirements smells.

2.4. Natural Language Test Smells

Nowadays, the concept of test smells is proposed by several researchers to be used as indicators of low quality in natural language test specifications. For instance, B. Hauptmann [1], identifies *test clones* as a smell that increases time and effort needed to maintain test cases, while B. Hauptmann et al. [4], identify a long test step and a test step that contains ambiguous words as smells that make the understandability of a test step's intention difficult. Therefore, the authors of [1], [4], claim that natural language test specifications that contains smells such as *test clones*, *hard-coded values* or *long test steps* can have a negative impact on the implementation, maintenance, and execution of test cases during the testing life cycle. To detect these smells in natural language test specifications, several detection mechanisms are used. As an example, B. Hauptmann et al. [4], use a list of keywords[2] to identify ambiguity in test steps.

[2] For example, *similar, better,* and *having in mind etc.*

3. Related Works

In this section, we will give an overview about several interesting papers that we discovered during our research. The section contains two subsections. In the first subsection we are going to cover several papers that work with detecting smells in natural language specifications. In the second subsection we are going to describe one machine-supported analytical measure NALABS and the smell metrics implemented in it.

3.1. Detecting Bad Smells in Natural Language

The attempts to improve the quality assurance of natural language artifacts by using machine supported analytical measures has been discovered in several studies. The first group of papers we discovered was from D. Falessi et al [20] and B. Hauptmann et al. [14]. D. Falessi et al [20] used several natural language processing (NLP) techniques to detect similarities between requirements specifications. B. Hauptmann et al. [14] used another technique to detect similar parts of test cases, because they found that a high degree of clones between test cases can have a negative impact on the cost for maintaining and executing them.

The second group of papers we discovered was from B. Hauptmann et al. [4] and H. Femmer et al. [19]. The authors of these papers transferred the concept of code smells to be used in the context of quality for two different artifacts. B. Hauptmann et al. [4] implemented a set of smell metrics based on their experience of working in the software testing area. According to the author [4] their proposed smell metrics can be used to detect defects in natural language system tests . H. Femmer et al. [19] also implemented another set of smell metrics to detect defects in requirements specifications. The purpose of both smell metrics is to detect defects in natural language artifacts in an early stage of its quality assurance process.

3.2. NALABS Tool: Detecting Bad Smells in Natural Language Requirements and Test Specifications

Enoiu and Rajkovic [3] found that a set of bad smell metrics that some studies use to detect defects in specifications written in natural language are restricted and mostly focuses on maintainability attributes. Therefore, they implemented another tool

NALABS[3] that extends a set of requirement-based smells with metrics related to complexity. According to the authors [3] the smell metrics they extended can be used to detect quality issues in both natural language requirements and test specifications. The authors [3] claim that the smell metrics they extended have been successfully used to detect quality issues in natural language requirements specifications, but the smell metrics remain to be tested on test specifications. In addition, these metrics may not be complete to capture all quality aspects of natural language test specifications, and therefore there is a need to extend these metrics by exploring, proposing, or combining with new bad smell measures to cover more aspects of quality. The following metrics are implemented in NALABS to automatically detect defects in natural language specifications:

- **Vagueness**

Vagueness refers to a metric used to measure properties that add extra complexity to natural language specifications by making it ambiguous and difficult to understand. The authors [3] has suggested a list of keywords that indicate vagueness in natural language specifications. **List of keywords**: *"May", "could", "has to", "have to", "might", "will", "should have"," must have", "all the other", "all other", "based on, some", "appropriate", "as a", "as an", "a minimum", "up to", "adequate", "as applicable", "be able to", "be capable", "but not limited to", "capability of", "capability to", "effective, normal".*

- **Referenceability**

A specification that contains a reference to another document that must be read in order to understand the specification contains a bad smell. This is usually an indication of nesting in the specifications. The author divides keywords that indicate referencing into two categories. The first category is called *NR1* and includes keywords such as *"specified by reference", "see the reference*" etc. The second category is called *NR2* and includes keywords such as *"Figure", "Table", "for example"* and *"Note"* [3].

[3] NALABS is a desktop WPF applications that depend on standard .NET packages and can be accessed on Github on this link: eduardenoiu/NALABS: NALABS is a requirement quality checker for natural language requirements. It uses a set of bad smells to indicate problematic requirements. (github.com)

- **Optionality**

Optional words provide the developers with a wide range of interpretations in order to satisfy the specified claims, therefore their usage in natural language specifications is generally not recommended [3]. **List of keywords**: *"can", "may"* and *"optionally".*

- **Subjectivity**

This metric is used to measure personal opinions or feelings in sentences. The author suggests a list of keywords that indicate subjectivity in natural language specifications. **List of keywords:** *"similar", "better", "similarly", "worse", "having in mind", "take into account", "take into consideration", "as possible"* [3].

- **Weakness**

A word or a phrase that leaves room for multiple ways of interpretations by developers is considered a bad smell in natural language specifications. **List of keywords:** *"adequate", "as appropriate", "be able to", "be capable of", "capability of", "capability to", "effective", "as required", "normal", "provide for", "timely", "easy to"* [3].

- **Readability**

To measure the readability of natural language specifications, the author [3] decided to use automated readability index (ARI) over other readability indexes such as Flesch reading ease index in order to keep the implementation of the readability metric simple. ARI can be calculated by using the formula WS + 9 × SW, where WS is the average number of words per sentence and SW is the average number of letters per word.

- **Conjunctions**

The complexity metric can be measured using different factors. One method that the authors use is counting the number of occurrences of conjunctions. The authors [3] found that some conjunctions are often used to show relations between actions, and they claim that the usage of these words often adds logical complexity to the sentence. The authors also use the number of words as a measure of the specification size. **List of keywords: "***and", "after", "although", "as long as", "before", "but", "else", "if", "in order"," in case", "nor", "or", "otherwise", "once", "since", "then", "though", "till", "unless", "until", "when", "whenever", "where", "whereas", "wherever", "while", "yet"* [3].

Two additional requirements-based metrics are also implemented in NALABS which are the *imperative* and *continuous* metrics, but they are tailored to detect defects in requirements specifications. In our study, we find that it would be interesting to include these metrics to see if they also can help to improve quality assurance of test specifications.

- **Imperatives**

The author [21] recommend using the imperative "shall" instead of other imperatives. They divide imperative words into two categories, but since the two categories introduced by them, include overlapping words, we decided to combine them into one category. This is because we did not find any difference between the two categories. **List of keywords:** *"shall", "must, "is required to", "are applicable", "responsible for", "will", "should", "could", "would", "can", "may".*

- **Continuances**

The use of the words listed by the continuance metric is considered by [21] as an indicator of complexity and excessive details in a sentence. Therefore, this metric is used by [21] to measure the complexity of the specification. **List of keywords:** *"below", "as follows", "following", "listed, in particular", "support", "and".*

As a second step in the same direction proposed by the authors [3], we will first investigate which natural language bad smell metrics [3], [21] exists in real-world test specifications. In addition, to measure the sufficiency of each smell metric that appear in the test specifications, and to see in which way these metrics can help to improve the quality attributes of test specifications. Secondly, we will try to extend these smell metrics by exploring, proposing, or combining with new natural language test smells. We will refer to all metrics implemented in NALABS as ***natural language bad smells***.

4. Problem formulation

Tests are important artifacts in the software development process. Testing activities such as test automation, test maintenance, and test suite optimization mainly rely on an In-depth understanding of the test specifications [14]. Test specifications are usually written by one or several test designers manually. The manual process of writing test specifications in natural language can create many different quality issues, such as ambiguous, redundant, or inconsistent test specifications [1], [3], [14]. One reason for that is because the test designers who write them do not always have software engineering best practices in mind or these do not fully understand what needs to be tested or how to test it [4]. As a result, these quality issues make the understandability of test specifications difficult, which in turn can lead to testers interpreting and executing the test cases differently [1], [4]. Furthermore, it increases the cost for maintaining the test cases [14], and it becomes impractical when it comes to reviewing and validating a large test specification manually [22]. Consequently, there is a need for machine-supported analytical measures that can work as a complement to manual reviews by highlighting certain types of potential quality issues in test specifications during its quality assurance process [4].

4.1. Research questions

To achieve the aim of the study we have formulated two research questions.

RQ 1: Which natural language bad smell metrics implemented in NALABS can be applied to detect potential quality issues in test specifications?

RQ 2: Are there additional smells that can be implemented in NALABS to cover more aspects of natural language test specification quality?

5. Method

In order to answer the aforementioned research questions, two scientific methods are used in this thesis. First, we prefer to rely on the case study research method over other scientific methods such as a controlled experiment to answer **RQ1.** This is because, we need to assess the NALABS tool in a practical setting under realistic conditions. We follow the guidelines from Runeson and Höst [23] to conduct a case study.

To answer **RQ2**, we perform a literature study of relevant scientific studies to explore if there are any studies that introduce more natural language test smells that cover more aspects of test artifact quality. We follow the guidelines from C. Wohlin [24] to conduct a literature study.

6. Ethical and Societal Considerations

When developing new technology, engineers must ensure that the developed technology will be used to benefit society, and not to harm it. For instance, that the developed technology will not be used to harm people physically or mentally [25, pp. 241-243]. In our thesis, the tool we evaluated and improved does not have any negative impact on society since the work aimed to improve quality assurance of natural language test specifications that do not contain any personal data or data that can be used to harm people. Most of the data that was needed to answer **RQ1** was collected from open-source repositories, and there was no need to save it locally on a password-protected computer. When it came to the data that we needed to collect from companies or reviewers, we saved it on a local password protected computer and will be removed at the end of this work. We did not perform any interviews with reviewers or companies, all communication in this work occurred via E-mail. Personal data that might be connected to any individuals or companies was anonymized or not included at all in the report.

7. Case study design

7.1. Case Study Objective

With conducting the case study, we first aim to verify the existence of natural language smells [3], [21] in real world test specifications and to understand how widespread they are. However, we cannot rely on the number of findings to determine which bad smells can be applied, because if a large number of findings are irrelevant to quality attributes of test specifications, it will hinder quality assurance more than it will help. Therefore, we will perform manual assessments of the findings to quantify the quality of each smell metric implemented in NALABS.

7.2. Data Collection Procedures and Data Analysis

7.2.1. Collection of Test Artifacts

The test artifacts were collected from multiple resources. Two of the collected test artifacts were provided by two different companies. We searched for the remaining test artifacts in multiple open-source repositories by using keywords such as “test suite”, “natural language test suite\cases”, “system test” etc. We used open-source repositories such as *GitHub*, *Zendo* and *NLRP benchmark*. In some of these repositories we were not able to find any test artifacts related to our thesis. During the process of collecting the test artifacts that are needed to answer **RQ1**, we found that the test artifacts manifest themselves in different writing styles. What is interesting about this is that some test artifacts only contain a set of test cases such as manual system tests or unit system test, while others only contain test requirements. The last style we found contains a mixture of both, test requirements, and a set of test cases (*See appendices sections 15.1 and 15.3 for examples*). We decided to include all writing styles we found in our study. We checked every test artifact manually to ensure that it contains enough test cases or test requirements that tests or describes different functionality of the system in order to get a reasonable amount of quantitative data, (*See* **Table 2**).

Table 2. *An overview of the characteristics of the test artifacts that were collected.*

Test artifact	#Tests cases	#Words	Source
Test suite & Test Requirements A	5	674	Provided by Company
Test Requirements B	—	3153	Github
Test suite C	17	2113	Github
Test suite D	44	825	Github
Test suite E	27	450	Github
Test Requirements F	—	2057	Provided by Company
Test suite & Test Requirements G	16	3891	Github
Test suite & Test Requirements H	13	3215	Github
Total	122 tests	16 378 words	—

7.2.2. Using NALABS on the Collected Test Artifacts

We applied NALABS on all collected test artifacts, which produced a list of highlighted findings that are categorized according to its smell metric, (*as shown in* **Figure 1**). This process provided us a reasonable number of findings of the natural language smell metrics to be analyzed.

RCM

File Edit View

Id	Text	NW	NC	NV	Optionali	Subjectivi	NR	NR2	Weakness	Imperativ	Imperativ	Continuat	ARI
	Verify by trying to re-download the file after entering invalid Batch No. field.	13	1	0	0	0	0	0	0	0	0	0	54.08
	Verify by trying to re-download the file after leaving the Batch No. field blank.	14	1	0	0	0	0	0	0	0	0	0	51.21
	Verify by re-exporting the same file again on the same path.	11	0	0	0	0	0	0	0	0	0	0	51.91
#0005	Check for the file name downloaded. (In case if the file name is static while download).	16	3	0	0	0	0	0	0	0	0	0	49.06
		0	0	0	0	0	0	0	0	0	0	0	NaN
	Download the file and check for generation of the Batch No.	11	2	0	0	0	0	0	0	0	0	2	51.09
	Verify by re-exporting the same file again on the same path.	11	0	0	0	0	0	0	0	0	0	0	51.91
	Verify by downloading the file after entering the already generated batch no.	12	1	0	0	0	0	0	0	0	0	0	61.50
#0006	Verify by entering the filename to download the file. (In case if user is required to input file name while downloading the file).	23	3	0	0	0	0	0	0	1	0	0	53.76

Figure 1. *shows a part of the list of detected smells by NALABS for test suite D. The detected findings are highlighted and categorized according to its metrics.*

7.2.3. Manual Quantification of The Findings

With the manual reviews of each finding detected by the smell metrics implemented in NALABS, we could quantify the appearance of each smell metric in the test artifacts. This helped us to verify the existence of the smell metrics in the test specifications. By using the number of words as a measure of the test artifacts size, we could put the number of findings in each test artifact in relation to its size. This in turn helped us to understand how widespread the natural language bad smells are in the test artifacts.

7.2.4. Manual Classification of The Findings

In order to quantify the quality of each smell metric implemented in NALABS, we decided to use the metric *precision* over other metrics such as *recall*. Because in our study, we are only interested in a metric that can measure the number of correct positive findings out of all the findings that are labeled by each smell metric as positive findings, and not in a metric such as *recall* that can measure the number of missed positive findings by each smell metric [26].

The metric *precision* is defined as follows:

$$\text{Precision} = \frac{\text{no .of True Positives}}{\text{no .of True Positives} + \text{no .of False Positives}}$$

- **A true positive finding** is an instance of a bad smell, e.g., the finding indicates an actual quality issue in the test specification.[4]
- **A false positive finding** is an instance of a bad smell but does not indicate a quality issue in its context.[5]

When it comes to the *ARI metric,* we used the readability index measures introduced by Lehner, F [27] to quantify the number of *true positive* findings. The author considers a text that have an ARI-score of around 50 as a simple text, around 60 as a medium difficult text, and a score over 70 as a difficult text. We decided to quantify and classify all text that have a score over 70.

[4] For example, if the automated analysis of NALABS classified the word "normal" in the sentence "All inputs in normal state" as a vague or weak word, we considered the finding as a true positive, because the designer does not describe what the normal state is.

[5] For example, if the automated analysis of NALABS classified the word "and" in the sentence "Read the policy creation and check the initial letter cap" as a conjunction or continuance, we considered it as a false positive finding because, it does not lead to unnecessary logical complexity or excessive detail in the sentence.

Sometimes, some of the keywords listed by one metric overlap with other metrics[6], in these cases we quantified and classified the finding for all metrics the finding belongs to. In addition, we ignored all findings that are irrelevant to the test area such as cover page, test plan identifier, references, date, table of contents etc.[7]

[6] For example, the verb "may" belong to the *vagueness*, *imperative* and *optionality* metrics.
[7] For example, if the automated analysis of NALABS classified the word "may" in the sentence "Project Schedule i.e., by May 20th 200" as an optionality smell, we considered the finding as irrelevant.

7.2.5. Validity of Manual Classification

Since there is sometimes a need of domain knowledge in the testing area to determine if a finding is in fact a quality issue, we can not only rely on our own experience to make a final decision about the result. Therefore, before we started to classify all the findings, we decided to classify a random subset of findings to be sent to expert reviewers that have domain knowledge in the software testing area in order to investigate how the reviewers would react to the findings.

We selected 6 subsets from random test artifacts. Each subset contains a minimum of 10 findings from at least 2 different smell metrics as far as the artifact allowed *(as shown in* **Figure 2**). All subsets are grouped into a *word document,* and each finding in the document is commented as a *true* or *false positive* finding according to our classification. The number of findings the document contains is limited as far as the reviewers' time allowed. In case the participants did not agree with the classification of a finding, we asked them to offer us input on why they would label it differently. After all participants responded, we analyzed and compared their inputs with each other and finally to our classifications.

Test step	Expected result
Check for the file name downloaded. (**In case if** the file name is static **while** download).	System **should** allow user to enter file name **while** exporting file. (In this case, max characters **should** be fixed to set for file name download).
Verify by entering the filename to download the file. (**In case if** user **is required to** input file name **while** downloading the file).	System **should** download the data as per the Business req. (Here, Company Name column **should** download the values up to 100 characters only).
Check for the capturing of the errors in the log file **whenever** file encounters any error **while** being exported.	System **should** download the data as per the Business req. (Here, Company Name column **should** download the values up to 100 characters only).
Verify the field length of the characters downloaded in the file. (E.g. As per Business req. it is mentioned that data in column 1, e.g. **Should** be company Name **and should** be upto 100 characters).	System **should** capture the proper logs **and** in detail so that user **can** understand about the errors encountered **while** file being exported.

Figure 2. *shows one of the random subsets we selected from a manual system test from test suite D. The detected findings are highlighted and categorized according to its smell metric. Red words belong to the imperative metric, purple words belong to the conjunction metric and the turquoise words belong to the optionality metric.*

8. Investigating Bad Smells in Natural Language Test Specifications

We have reviewed articles and papers that introduce or propose new natural language test smells that can be implemented in NALABS in order to improve quality attributes of test specification. We searched for articles in different available databases such as *Google scholar, ACM, IEEE,* and *Science direct*. We used keywords such as "test artifact quality assurance", "natural language test", "natural language test smells", "test quality", "natural language test quality", "natural language test requirement", "natural language test specification", "test suite quality", and "test cases quality". During our research we found that there are a limited number of papers covering test smells in natural language test specifications. We did not perform a systematic literature study, instead we checked the references for all the papers that we discovered and reviewed that covered natural language test quality, imitating a snowballing approach in order to discover any other interesting papers.

The majority of the papers and articles that we discovered introduce different bad code smells, and they often overlap with each other. We were unable to go through all the discovered papers, instead we focused on the most relevant papers for our study. We decided to go through some of these papers and tried to find any code smells that can be applied on natural language test specifications. Many of these papers and articles were discarded, because we found that almost all the code smell metrics introduced by these papers are tailored to code related aspects and cannot be applied on natural language test specifications.

We started by reviewing articles that discuss the quality attributes of natural language test artifacts. The first relevant article we discovered was from B. Hauptmann et al. [14], that focused on the detection of clones in test artifacts. Two other interesting papers that we discovered were [28], [29] that just discussed the quality attributes of test artifacts. However, the authors of these papers did not introduce any bad smell metrics, instead they focused on the quality characteristics of test artifacts in general which helped us to capture different ideas and aspects on how the quality assurance of test artifacts can be improved by using NALABS. By checking the references in the article [28] we discovered another interesting paper done by B. Hauptman et al. [4].

By reviewing the paper by B. Hauptman et al. [4], we found that the authors proposed seven natural language test smells. Some of these metrics overlap with the natural language metrics implemented in NALABS [3] such as the *subjectivity and conjunctions* metrics, but the authors used different names for these metrics and focused on different quality aspects. Later another interesting paper was discovered by B. Hauptmann [1] which introduced six more natural language test smells. One of these smells is *Hard-Coded Test Data* which overlaps with the paper from B. Hauptman et al. [4]. The bad smells proposed in both papers [4], [1], are tailored to detect natural language test smells in test specifications for manual system tests (*i.e., test cases*). In addition, some of the bad smells that they proposed are defined as a list of keywords, the authors did not mention that this list of keywords can also be applied to detect smells in test specifications for test requirements.

During our attempt to transfer some code smells into natural language test smells, we just found that one code smell introduced by three papers [30], [31], [32] can be applied on test specifications. We confirmed our transformation by manually reviewing if the smell appears in the test artifacts we analyzed. The table below shows a brief overview of the natural language smell metrics we found with its corresponding description and paper.

Table 3. *An overview of the discovered natural language test smells.*

Smell name	Description	Paper
Hard-Coded Values	When a test specification contains "magic numbers" or strings that indicate test data or names of user interface elements.	[4],[1]
Long Test Steps	A test step is very long.	[4]
Conditional Tests	A test steps description contains conditional logic which is phrased in natural language.	[4]
Badly Structured Test Suite	The test suite's structure does not follow the structure of the tested functionality.	[4]
Tests Clones	Many tests in the suite share large similar parts introduced by (copy paste).	[4],[14],[1]
Ambiguous Tests	Test steps that are written in ambiguous ways leave room for multiple ways of interpretations.	[4]
Inconsistent Wording	For the same domain concept, several names are used in the test suite, e.g., the test suite does not use its domain concepts in a consistent way.	[4]
Branches in Test Flow	The test flow contains branching or alternate flows that are manifested in the test steps' text as conditions.	[1]
Merged Test Steps	Several independent tasks or actions are combined to one single test step.	[1]
Complicated or Bloated Phrases	A test step contains unnecessary or redundant information such as rationales or side information that is not needed to understand the test step.	[1]
Ambiguous Phrases	Ambiguous phrases in the test specification leave room for multiple ways of interpretations.	[1]
Dependent Test	The degree of dependence between one test case and other test cases in the same test suite.	[30],[31],[32]

9. Results

The result section is divided into two subsections. In the first subsection we are going to analyze the result of manual quantification of findings, manual assessment of findings by reviewers and the result of evaluating the natural language smell metrics. At the end of the first subsection, we provide an answer to **RQ1**. In the second subsection, we are first going to describe several natural language test smells that we discovered during our research. Then we are going to propose a set of natural language smells that provides an answer to **RQ2**.

9.1. Result RQ 1

9.1.1. Manual Quantification of Findings

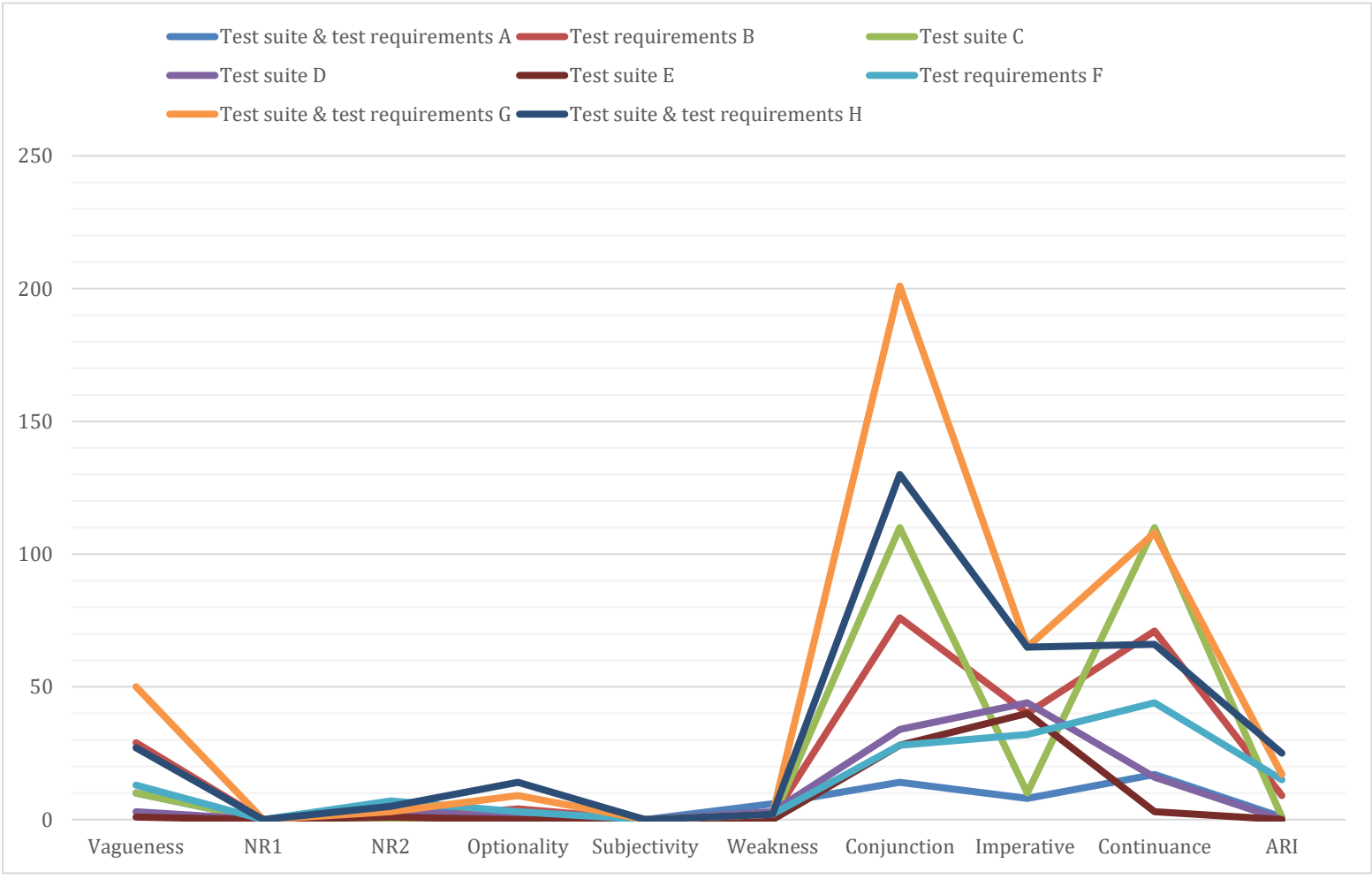

Figure 3. *Shows the number of findings for each bad smell metric in every test artifact.*

The result of the analysis of eight test artifacts shows that the number of findings of each smell metric differ between the test artifacts, and that not all smell metrics exist in the test artifacts we analyzed. Furthermore, the appearance of one smell metric in the test artifacts does not mean that all words listed by that metric exist in the test artifacts. This is because in some instances we were not able to find certain words belonging to that metric in any of the eight-test artifacts we analyzed[8].

As **Figure 3** shows, we were not able to find any appearances of the metrics *subjectivity* and *NR1* in any of the test artifacts. In addition, when it comes to the *NR2, Weakness, ARI*, and *optionality* metrics, we found that the appearances of these metrics are rare. For instance, we were not able to detect a large number of findings of the *optionality* metric. The largest number of findings for the metric *optionality* is found in *test artifact H,* which has 14 findings out of 3215 words. *Weakness* is the metric that we found the least findings of. The largest number of findings of the *weakness* metric is found in *test artifact A* which contains 6 findings out of 674 words (*See the result of analyzing test artifacts A and H by using NALABS in the Appendices sections 15.2 and 15.4*). Despite the small number of findings of the *weakness*, *NR2*, and *optionality* metrics, we send some of these findings to reviewers to make sure if the appearances of these metrics can lead to any quality issue in the test specifications.

The result of the manual quantification of findings also shows that the appearance of the metrics *conjunctions, continuous imperatives*, and *vagueness* are common in the test specifications, because they have a large number of findings compared to other smell metrics such as *weakness* and *optionality*. On the other hand, these metrics appeared in all test artifacts we analyzed. In addition to that, as **Figure 3** shows, the greatest number of findings that we found in each test artifacts are from one of these metrics. Therefore, we included a large number of findings for these metrics in the reviewees' document.

We also analyzed the number of findings in each test artifact relative to its size (*as shown in* **Table 4**). The number of findings in the table does not include the number of findings for the *ARI metric*, since it does not make sense to put the number of *ARI* findings in relation to the artifact's size (number of words). **Table 4** shows that the

[8] For example, we were not able to find the word "optionally" in the metric *optionality* in any of the test artifacts. The same holds for the words "adequate", "be capable of", "timely" or "easy to" in the metric *weakness*.

number of findings per 100 words vary between the test artifacts. The test artifacts that produce the greatest number of findings per 100 words are the test suites C, D and E that only contain a set of test cases. We also discovered that when the test artifacts contain a mixture of a test suite and test requirements the number of findings per 100 words decreases. We confirm our result by analyzing the test artifacts that only contain test requirements. The result shows that the test artifacts that only contain test specifications in terms of test requirements produce the least number of findings per 100 words compared to other test artifacts that contain a set of test cases or a mixture of test cases and test requirements.

Table 4. *shows the number of findings per 100 words for each test artifact.*

Test artifact	#Words	#Findings	#Findings per 100 words
Test suite & Test Requirements A	674	63	9.3
Test Requirements B	3153	221	7.0
Test suite C	2113	240	11.3
Test suite D	825	104	12.6
Test suite E	450	72	16
Test Requirements F	2057	129	6.2
Test suite & Test Requirements G	3891	438	11.2
Test suite & Test Requirements H	3215	309	9.6

As a result, the test artifacts that contain a test specification for test cases such as manual system tests produce a higher number of findings per 100 words compared to test artifacts that only contain a test specification for test requirements.

9.1.2. Manual Assessment of Findings by Reviewers

The goal of the manual assessment of findings by reviewers is to receive their feedback regarding if the quality issues cause by the smell metrics [3], [21] are relevant to quality attributes of test specifications. We got responses from two experts in the software testing area at Mälardalen University. The reviewers responded on every finding in the document. Furthermore, both reviewers left feedback as to why they consider the findings as relevant or irrelevant (*as shown in* **Table 5**).

Table 5. *some examples of the reviewers' answers.*

Smell metric	Text	Feedback R 1	Feedback R 2
Weakness, Vagueness	Be true before the start of testing. When the simulation is activated (By default setting), most of the inputs for the safety functions are simulated to the normal state (To the safety state).	*False positive*. The meaning is clarified immediately after	*False positive*. Normal state is explained in brackets.
Vagueness	Prerequisite for test: As inputs below. All related systems have to be ready and in the normal state (in the safety state) to allow test.	*False positive.* It is analogous to must.	*True positive.* If including" have to be ready".
Optionality	Test that each user can only have one job function.	*False positive.* Not optionality.	*False positive*. Not optionality
Vagueness	Test that each job function has proper access & privileges based on the job function.	*True positive*. Here are the access and privileges on the job function defined?	*False positive.*
Continuance	The functions to be tested are listed in Section 5.1 of this document.	*True positive.* It makes the understandability of the specification difficult.	*False positive.* I would consider it not a quality issue to reference other places in a document, considering what the alternative would be.
Conjunction	To check if the system is compatible with company's browser standards.	*True positive*. What compatible exactly means?	*False positive.* Not a conjunction
Conjunction	Check for the file name downloaded. (In case if the file name is static while download).	*False positive.* This is the precondition that needs to be valid for the test to have sense	*True positive.* Complex test step.
Conjunction	Verify by entering the filename to download the file. (In case if user is required to input file name while downloading the file).	*False positive.*	*True positive*. Complex test step.

Imperative	System should allow user to enter file name while exporting file. (In this case, max characters should be fixed to set for file name download).	*True positive*. It is not clear how relevant is this test. If it is critical, "shall" is better	*False positive.* The verb "should" does not have to be "shall".
NR2	Test: This case tests the safety function outputs when the safety function inputs are set as in the Inputs table below.	*False positive.* Not reference.	*False positive.* Not reference.

By analyzing the reviewers' documents and comparing them with each other, we found that some of the same findings are considered by one reviewer as *true positives,* while the other reviewer considers them as *false positives* (*See* **Table 6**).

Table 6. *Shows the individual precision by each reviewer for every smell metric.*

	Vagueness	**NR2**	**Optionality**	**Weakness**	**Conjunction**	**Imperative**	**Continuance**
#Findings	9	5	3	4	34	21	16
R1	77%	0%	66%	50%	23%	57%	31%
R2	55%	0%	0%	50%	14%	0%	12%

Table 5 shows that, the similarity between the reviewers occurs when it is easy to determine if a finding causes a defect, and the differences occurs when it becomes difficult to determine if a finding in fact causes a defect. This is because, different reviewers have different experience and criteria to judge a finding. That is, the classification of one finding is sometimes dependent on the human reading it, and how they give their subjective review. Therefore, we found 23 conflicts between the reviewers out of 92 findings that the document contains.

9.1.3. Evaluation of Natural Language Smell Metrics

During the process of determine which writing style produces a higher number of *true positive* findings, we found that the number of correct findings differ strongly between the same writing style as well as between the different writing styles. For instance, the number of findings in *test artifact G* is 455 (*including ARI findings*), we classify 116 findings as *true positives* which resulted in a *precision* of 25.3%. In comparison, in the *test artifact H*, we classify 186 findings as *true positives* out of 334, which resulted in a *precision* of 55.5%. Therefore, we decided to calculate the *precision* for the total number of findings for each smell metric (*See* **Table 7**).

Table 7. *The overall result of the precision for each smell metric.*

Smell metric	#Findings	#Perceived true positives	#Perceived false positives	Precision
Vagueness	142	66	76	46.5%
NR1	—	—	—	—
NR2	25	3	22	12%
Optionality	33	22	11	66,5%
Subjectivity	—	—	—	—
Weakness	16	10	6	63%
Conjunction	621	318	303	51,2%
Imperative	304	93	211	30.5%
Continuance	435	132	303	30.1%
ARI	68	37	31	52.1%
Total	**1644**	**681**	**963**	**41,4%**

During, the manual assessment of the metric *vagueness* we found that the using of vague words in a test specification can lead to quality issues in terms of understandability and complexity. We classified some of these findings as *true positives because* we discovered that using words listed by this metric in some instances make the understandability of the test specification ambiguous.

When it comes to the metric *NR2*, we classify most of the findings as *false positives*, because most of the findings indicate that the test artifacts contain tables. Using tables in test artifacts is common because they usually contain necessary input and output data for the specific test, which in turn does not indicate any quality issue in the test artifacts. However, some findings of this metric are classified as correct findings since

they indicate nesting in the test artifacts. This in turn makes the readability of the test specification difficult.

As **Table 7** shows, there are twice as many *true positive* findings as *false positive* findings for the *optionality* metric. This is because, in some instances we found that the designers who write the tests use optional words in a way that does not leave any room for multiple ways of interpretations. In other cases, the using of optional words lead to different expectations by the reader. It may lead to testers executing test cases differently depending on their perception. The same holds for the weakness metric.

An almost equal amount of *true* and *false positive* findings is also found in the *conjunction* metric. In many cases, we found that the use of conjunctions in the test specification adds more logical complexity to a sentence or a test step by adding information that is unnecessary to understand it.

When it comes to the *imperative* metric, in most cases we did not find that changing imperative words to "shall" would help to improve the quality of the test specification. But in some cases, there is a need to change some imperative words to "shall". As an example, we found that the using of different imperative words in the same sentence or test step may lead to a misunderstanding of the test specification, in the case where different imperatives exist in the same sentence.

In the metric *continuance* we classify most of the findings as *false positives*, because in most cases, using words listed in this metric does not lead to quality issues in the test specification. However, we found that in some instances their usage is an indicator of complexity and excessive details in a test step. In these cases, the test step may have to be split into two smaller parts to improve the understandability of the test step.

When it comes to ARI, we found that in most cases the ARI-score is between 40-70, and in some cases the score is over 70, (*See the result of analyzing test artifacts A and H by using NALABS in the Appendices sections 15.2 and 15.4*). We classify an ARI-finding that have an ARI-score over 70 as *true positive* when we found that, the readability of a sentence or a test step mostly depends on the experience of the reader to understand it. In addition, when the sentence or test step indicate a high level of difficulty in terms of readability.

Table 8 shows a subset of findings that are detected by the natural language smell metrics implemented in NALABS. We classified all the findings in the table as *true* or *false positives* and describe their corresponding quality issue.

Table 8. *Shows some examples that are detected by the smell metrics implemented in NALABS.*

Test specifications	Smell metric	Class-Ification	Quality issue
Can the user easily identify the settings icon?	Optionality	True	Different testers have different criteria.
That is one user can only give one update.	Optionality	False	"Can only give" does not leave any room for multiple ways of interpreting.
This load target can be determined via log queries and should be updated quarterly as needed.	Optionality	True	It does not specify how.
Check whether labels float upward or not when the text field is in focus or filled? Check whether Sign-Up Button is present or not.	Conjunction	True	"Or not" Here does not add any meaning to the text, it just adds more logical complexity to the sentence.
Read the Premium payment and check the initial letter cap. Check all the fields and buttons displayed on the Registration Page.	Conjunction, Continuous	False	It does not add any complexity or excessive details to the sentence.
Internet connectivity should be available and registration page must be loaded.	Imperative	True	Seem to indicate a difference in priority or importance of the test pre-condition.
Initial latter should be caped. All Buttons and form Fields should be displayed properly. Standard font, text color and color coding should be there.	Imperative	False	Using of verb "shall" does not make the sentence clearer here.
Touch on the "Symptoms" feature took us to a screen where all the symptoms of COVID-19 were listed - like Fever, Dry cough, Headache, Breathless, tired etc., with images.	Continuous	True	Test step contains excessive detail.
Touch on the "Preventions" feature took us to a screen where all the preventions were listed – like stay home if sick, cover cough, cough on your elbows, clean and disinfect, avoid close contact, cover mouth and nose etc., with images.	Conjunction, Continuous	True	*Complex test step. Include excessive details.*
Touch on the "Feedback Us" Feature took user to another screen where user must select one option out of 5 (5 emoticon) after selecting user must tap on submit to successfully submitting the feedback.	Conjunction	True	*Complex and long test step*. The step must be split into smaller test steps in order to determine

			which functionality of the test step failed.
This shall be input data the program may be presented, but that will not produce any meaningful output.	Vagueness	True	It is difficult to understand the test intention.
The safety functions that are not tested have to be simulated by the internal simulation.	Vagueness	False	It does not make the sentence difficult or complex to understand.
Both [SYSTEM] numbers are changed simultaneously.	ARI Score = 72	False	It is not difficult to read the sentence.
In ideal scenario we can have more details in the report giving a tracking ID to each test, developer who developed it, date on which issue was identified, steps followed to find a bug, screenshots of these steps, Priority to be resolved, etc.	ARI Score = 85	True	It is difficult to read the sentence.
It will display number of new cases coming in on daily bases, a linear & logarithmic curve of graph showing total cases form last one year, graph of daily new cases, daily deaths, population of the country vs corona affected count, a table of all the countries with the counts etc.	ARI Score = 91	True	It is difficult to read the sentence.
Just after submitting the review, user will be able to see one pop up saying "your review is submitted".	Weakness	false	It does not leave room for multiple interpretations.
The safety function isn't in the normal state because 0 isn't allowed as the safety identity number. All inputs in normal state.	Weakness	True	Difficult to understand the test intention. The designer does not describe what the normal state is, which leave room for multiple ways of interpretations.
If affected user has added his/her location on the "Add Location" features, then only he/she will get the notification (i.e., you Entered the affected place or you leave the affected place)	NR2	True	Additional reading is required to understand the test step.
The safety function inputs are set as in the Inputs table.	NR2	False	It does not indicate any quality issue.
Touch "Feedback Us" feature available on the home screen of the application, after touching it should redirect it to a new screen from which user has to select a rating and then click on submit button. After Submitting the feedback, it should show in Firebase Database.	Conjunction, Continuous	True	*Complex and long test step*. The step must be split into smaller test steps in order to determine which functionality of the test step failed.

***Answer to RQ1*:** The result of the manual quantification shows that most of the natural language smell metrics implemented in NALABS exists in the real-world test specifications. The most common findings for the smell metrics are *conjunctions, continuance, imperative, vagueness, ARI, optionality, NR2 and weakness* respectively. No findings for the smell metrics *NR1 and subjectivity* have been found. The result of the manual assessment of the findings shows that the appearance of one finding listed by any smell metric does not necessarily lead to a quality issue in the test specifications. Therefore, the findings of these metrics must be judged manually by reviewers to determine if a finding is in fact a quality issue.

We consider the metrics *weakness, optionality and ARI* that have a small number of findings and a *precision* over 50% as reasonable indicators of low quality in the test specifications. Because if the most findings of these metrics are considered by other reviewers as *false positives*, it will not hinder quality assurance significantly by increasing the time and effort needed to inspect them manually.

When it comes to the metrics *continuous, conjunctions, imperative and vagueness* that have a large number of findings, we cannot only rely on the number of *true positive* findings to determine which metrics are sufficient to be applied. Despite the *conjunction* metric having a *precision* over 50%. Because, we have had limited control over the number of *true positive* findings due to the metric's subjectivity. This means that, if a large number of findings are considered by other reviewers as *false positives*, it will hinder quality assurance by increasing the time and effort needed to inspect them manually. Consequently, these smell metrics can only improve the quality assurance by providing pointers to certain locations that may need to be inspected for defects. In other words, it is up to reviewers if they would inspect them.

Finally, we do not consider the metric NR2 as a reasonable indicator of low quality in test specifications because most of the findings do not lead to any defects in the test specifications.

9.2. Results RQ2

This section is divided into two subsections. In the first subsection we are going to cover all the natural language test smells we discovered during our research. Every part in this subsection describes one test smell with its corresponding quality issue and detection mechanism. In the second subsection we are going to propose a set of natural language smells that will provide an answer to **RQ2**.

9.2.1. Discovered Natural Language Test Smells

Smell - Hard-coded values:

Hard-coded values are a common problematic property when it comes to the maintenance of test cases. Because if a test case contains several hard-coded values, it becomes difficult to find out where to perform changes if these values must be changed. To detect this smell in natural language test cases, B. Hauptmann et al. [4] calculate the number of Hard-coded values relative to the number of words in the test case. Firstly, they calculate the number of words in the test case. Then they calculate the number of hard-coded values in the test case. They consider a word as hard-coded values if the word is in quotation marks or just consists of numbers. The test case contains a smell if the number of hard-coded values in the test case text relative to the number of words in the test case text is more than 10%.

Smell – Long test steps:

Many papers have identified the understandability of a test step as an important quality attribute [4], [28], [29]. A very long test step makes it difficult for testers to understand the step's intention. One measure of the understandability of the test step is only found in one paper [4]. The authors use the number of words in the test step as a measure of the step's understandability. If the number of words in an action or its expected result consists of more than 50 words, the test case contains a smell.

Smell – Conditional tests:

The using of conditional words in a test case specification makes the test case very complex, and it becomes very difficult for testers to understand the intention of the test case. According to [4] complex tests cases are more likely to have errors. Therefore, the authors count the overall occurrences of keywords that indicate conditions in the test specification. A test case contains a conditional smell if its text contains at least one of the following keywords. **List of keywords:** *"if", "whether","depending", "when", "in case".*

Smell - Branches in Test Flow

Test procedures must be established and be deterministically executable in order for the outcomes of the test runs to be understandable and comparable. As a result, test case descriptions should not include any branching logic, such as indeterministic case differentiation or optional parts. However, because of the nature of natural language, it becomes easy for the testers to ignore branching logic that are phrased in the test case descriptions. The metric that detects branches in test flow is found in one paper [1]. The author suggests a list of keywords that may indicate a branch in the test case flow.

List of keywords: *"if", "whether", "depending", "when".* The same list of keywords is used by [4] in the *smell - conditional tests.*

Smell – Badly Structured Test Suite

Another important quality attribute of a test suite is the structure of its test cases. Well-structured test cases have a positive impact on the efficiency [28] and maintenance [14] of the test cases. This is because, a tester who is familiar with the structure of the test cases and the essential components of the system interface perform the test cases more quickly, and they are aware of potential problems. Furthermore, if the test cases follow the same structure, it becomes easy to find out where to perform changes during maintenance of the test cases to ensure that the changes are made consistently [14]. Badly structured test suites can impact the understandability of test cases negatively because, it becomes difficult for testers to understand the functionality that the test intends to verify [4].

A mechanism to detect badly structured test cases in natural language test suites has only been found in one paper [4]. The authors of this paper suggest a detection mechanism based on natural language processing techniques. Firstly, they remove all *stop words*[9] from all test cases text. The authors then normalize the remaining words by reducing them to their word stem. Thereafter, they use the *term frequency-inverse document frequency* (TF-IDF) metric to determine the most dominant concepts of the remaining words for every test case. They presume that a test suite is organized from the beginning in folders and subfolders in a hierarchical manner. In addition, they assume that test cases that verify the same functionality should share the same domain concepts and should not be in different folders. If there is any test case that shares the

[9] *e.g.*, *a*, *and*, or *how*.

same dominant concept with other test cases that are located in different folders, the test contains a smell [4].

Smell – Test clones

Many papers have identified redundancies in test cases as an indicator of low quality of test artifacts, since redundancies in test cases impair their understandability and maintainability [14], [4], [1]. Test cases that share large similar parts that are not identical make it hard for testers to distinguish between the test cases, and it becomes difficult to understand the test's intention. Furthermore, it increases the effort and the time needed to maintain duplicate parts of the test cases, since it becomes hard to find out where maintenance must be performed [14], [4]. The definition of a test clone in test cases has been found in three papers [14], [4], [1]. For the simplicity of implementation in NALABS, we decided to use the definition proposed by B. Hauptmann et al. [4]. A test case contains a smell if it contains at least one test clone that fulfills the following definition.

"*a test clone is a substring of a test with at least 30 words appearing at least twice in a test suite. To find clones which differ slightly (e. g., because of inconsistent typo fixes), clones are allowed to have minor variations such that the difference (the gap) accounts for less than 10% of the length of the clone*".

Smell – Ambiguous Tests

A couple of papers consider ambiguity in a test case description as a low-quality attribute of a test artifact [4], [1]. Since test cases are written in natural language, it is easy to write them in an ambiguous way that leaves room for multiple ways of interpretations. This causes the testers to have different expectations, which may result in different test results in case a test case is executed by different testers. The metric that measures the ambiguity in test case descriptions is found in one paper [4]. A test case contains a smell if it contains at least one of the following keywords.

List of keywords: *"similar", "better", "similarly", "worse", "having in mind", "take into account", "take into consideration", "clear", "easy", "strong", "good, bad", "efficient", "useful", "significant", "adequate", "fast", "recent, far", "close".*

Smell - Ambiguous Phrases

Ambiguous phrases in test steps lead to the same quality issue as mentioned in the previous smell *ambiguous tests*, the difference between the smells *ambiguous tests* and *ambiguous phrases* is that the author of [1] searches for ambiguous phrases in test steps instead of ambiguous words [4]. B. Hauptmann [1], combines two techniques to detect ambiguous phrases in test steps which are *word list* and *text patterns*. Firstly, he detects all phrases that contain words that are likely to be ambiguous in their interpretation by using a list of keywords.

List of keywords: *"should", "most", "any", "more or appropriate".*

Thereafter, he uses a list of anti-patterns such as "most recent ..." or "more than (NOUN)". A test case contains a smell if it contains at least one detected phrase that does not match any of the listed anti patterns.

Smell - Inconsistent Wording

Inconsistent wording is considered as another common problematic quality attribute when it comes to understanding test case descriptions. The smell arises when the designers who write the test specification do not use domain concepts in a consistent way e.g., several names are used for the same domain concept in the same test suite. The measure of inconsistent wording in test suites is found in [4].

Similarly, to the first two steps in the smell *badly structured test suite*, the authors first remove all stop words from the text and normalize the remaining words to their word stem. Secondly, all words that have the same meaning are grouped together. Afterwards they calculate the most frequently used synonym in every group. Thereafter they go through every test step word by word to determine for each word if there is a more often used synonym for that word in the test suite. The test case contains a smell if there is at least one word that does not use the most frequently used synonym in its group [4].

Smell – Merged test steps

A couple of papers have identified the simplicity of test steps as one of the most important quality attributes [28], [1]. H.K.V. Tran [28], found that a good test case is a case that is comprised of steps that are well connected and do not include any unnecessary information. According to B. Hauptmann [1], a single test step should be clear and not include multiple independent tasks. Because when a test step consists of many independent test steps, it becomes hard to locate reasons for failed test cases since it is not clear which part of the test step failed. Furthermore, when a test step is

comprised of multiple tasks, it becomes difficult to understand the test step's intention. To detect merged test step, B. Hauptmann [1] calculates the number of words for each step in the test cases. A test case contains a smell if at least one of its steps consists of more than 130 words.

Smell - Complicated or Bloated Phrases

A complicated or bloated phrase in a test step can also have a negative effect on its maintainability and understandability. B. Hauptmann [1], identifies a complicated or bloated phrase as a phrase that is bloated up with unnecessary information. As mentioned before, a couple of papers have identified simplicity of test steps as an important quality attribute [28], [1]. Unnecessary information in a test step makes it difficult for testers to grasp what they should do [1]. To detect complicated or bloated phrases in the test steps, B. Hauptmann [1] calculates the number of words for each sentence in the test steps. A test case contains a smell if at least one of its test steps contains a sentence that consists of more than 45 words.

Smell - Dependent Test

One of the most important bad code smells is dependencies between test cases. A couple of papers found that dependencies between tests can have a negative effect on the execution of test cases [30],[31]. This is because a successful execution of one test case is dependent on the successful execution of other test cases in the test suite. That is, the test cases can only be run as a part of a collection of test cases in the test suite, not on its own. Dependencies between test cases can also mean that the test cases share a static field, stream, or file etc. This in turn can lead to one test case failing when it should not [32]. By manually checking the appearance of this smell in the eight test artifacts we analyzed, we found that there is a high degree of dependencies between test cases in *the test suite A*[10] (*See Appendices section 15.1*). No other dependencies between test cases have been found in the remaining artifacts. We suggest a list of keywords that indicate potential dependencies between test cases according to our findings.

List of keywords: *Completed test case + any number, completed tests, completed previous test\tests.*

[10] Example 1: Completed Test case 1 to obtain the normal state in the tested safety function and in all related systems to allow test.
Example 2: Completed previous tests to obtain the necessary state of the tested safety function and all related systems.

9.2.2. Proposed Improvements of NALABS

The result of the literature study shows that there are additional smells that can be implemented in NALABS to cover more aspects of natural language test specification quality. Most of these smells are tailored to improve the quality assurance of test specifications for manual system tests (*i.e., test cases*). Some of the discovered natural language bad smells are already partially implemented in NALABS. For instance, the author [3], use the number of words as a measure of specifications size, while the number of words is used by other authors to measure the complexity of a test step [4], [1]. In addition, the list of words that NALABS use to measure subjectivity in natural language specifications overlaps with the list used by B. Hauptman et al. [4] to detect ambiguity in test steps.

***Answer to RQ2*:** The table below shows all discovered natural language bad smells that we propose to be implemented in NALABS to improve quality assurance of test specifications. The table also specifies which smells are partially implemented and what remains to be implemented in NALABS.

Table 9. *Shows the proposed natural language bad smells that can be merged with the natural language smells implemented in NALABS.*

Smell name	Implemented	Smell name in NALABS	Already implemented	Implementation needs
Hard-Coded Values	No	—	—	See detection mechanism in the smell description above.
Long Test Steps	Partially	Number of words	Word count	Threshold: 50 words per action or its expected result in the test step. Improve GUI to indicate the smell.
Conditional Tests	Partially	Conjunctions	List of keywords: *if, when, in case.*	List of keywords*: Whether, depending.*
Badly Structured Test Suite	No	—	—	See detection mechanism in the smell description above.
Tests Clones	No	—	—	See detection mechanism in the smell description above.
Ambiguous Tests	Partially	Subjectivity Weakness	List of keywords: *similar, better, similarly, worse, having in mind, take into account, take into consideration, adequate.*	List of keywords: *clear, easy, strong, good, bad, efficient, useful, significant, fast, recent, far, close*.
Inconsistent Wording	No	—	—	See detection mechanism in the smell description above.
Branches in Test Flow	Partially	Conjunctions	List of keywords: *if, when.*	List of keywords: *whether, depending.*
Merged Test Steps	Partially	Number of words	Word count	Threshold: 130 words per test step text. Improve GUI to indicate the smell.
Complicated or Bloated Phrases	Partially	Number of words	Word count	Threshold: 45 words per sentence in a test step. Improve GUI to indicate the smell.
Ambiguous Phrases	No	—	—	See detection mechanism in the smell description above.

Dependent Test	No	—	—	List of keywords: *Completed test case + any number, completed tests, completed previous test\tests*

10. Threats to Validity

External validity:

All the test artifacts we analyzed to answer **RQ1** are collected from different companies and open-source repositories, and they are real-world examples. It is possible that, the artifacts are created by different testers who might have used different tools and processes. In addition, the artifacts test different application domains. We consider the external validity threat in our study to be mild. But to mitigate it and reach a more generalizable result, the study has to be repeated. To make the replication of the study possible, we used a publicly available tool NALABS that can be accessed in GitHub[11]. As a result, other researchers can repeat our study by using different test artifacts that test different application domains. The discovered natural language test smells that are used to answer **RQ2** are collected from scientific papers, and all the smells are tested and considered by the authors of these papers as actual bad smells.

Internal validity:

Since the classification of the findings has been performed manually in **RQ1**, it is possible that, the number of *true positive* findings depends on the reviewer's opinion thus introducing bias and subjectivity. To reduce this risk, before we started the manual assessment of the findings, two researchers that have domain knowledge in the software testing area and are not a part of the study team, have performed manual assessments of a subset of findings. This resulted in us forming a common understanding of the smell metrics and their corresponding quality issues. The manual assessment of the findings has been performed on a randomly selected subset of all the smell findings by the reviewers, we think it is possible that this introduces inaccuracy. To double check our understanding of the natural language smell metrics, we performed an individual classification of 20% of all findings. Afterwards, we reviewed and compared our classifications to each other to clarify in which context a finding is an instance of an actual smell. We repeated the classifications until we came to an agreement and gained an in depth understanding of the smell metrics that helped us to classify all the findings.

[11]NALABS can be accessed on GitHub at: eduardenoiu/NALABS: NALABS is a requirement quality checker for natural language requirements. It uses a set of bad smells to indicate problematic requirements. (github.com)

11. Discussion

In this work, we focused on evaluating and improving one machine-supported analytical measure NALABS that can work as a complement to manual reviews by highlighting defects in natural language test artifacts during its quality assurance process. The purpose was to make manual quality assurance of test artifacts significantly faster and more convenient.

In the first step, we investigated whether the already implemented smell metrics in NALABS can be applied to detect defects in natural language test specifications. We performed manual assessments of each smell metric by reviewing its detected findings. The result of the evaluation of the natural language smells shows that, the appearance of one natural language bad smell metric in the test specifications does not always lead to a defect in the test specifications, it sometimes just provides pointers to certain locations that the quality assurance may need to be inspected. We think that the result we reached for **RQ1** is reasonable compared to the result of the study made by H. Femmer et al [19]. The author mentioned that the appearance of their proposed list of smells in requirements artifacts cannot always be considered as an actual defect, and the findings must be judged manually by the context.

According to the claim we made in section 7.1 which was, "*if most of the findings of one smell metric are false positives, it will hinder quality assurance more than it will help*", we evaluated which smell metrics are reasonable to be applied. After manually assessing the *ARI, optionality,* and *weakness* metrics we have found that the appearances of these metrics are rare in the test specifications. In addition, in most cases the appearance of these metrics actually causes defects in the test artifacts. If we consider these metrics as reasonable indicators of low quality and our evaluation is not entirely accurate. It will not significantly hinder quality assurance by increasing the time and effort needed to inspect them manually. In case other reviewers consider a large number of these findings as *false positives*. Therefore, these metrics are considered as reasonable indicators of low quality in test specifications.

For the *conjunctions, imperatives, continuous* and *vagueness* metrics that have a large number of findings, if we consider them as reasonable indicators of low quality and a large number of these findings are considered by other reviewers as *false positives*, it actually happened in the manual assessment of findings by reviewers. For instance, in

the case of the metric imperatives, one reviewer considers 12 findings out of 21 findings as *true positives,* while no findings are considered by the other reviewer as *true positives*. In that way the findings of these metrics will have a negative impact on quality assurance by increasing the time and effort needed to inspect them. On the other hand, if we consider them as unreasonable indicators of low quality due to the large number of *false positives* the metrics produce, it may lead to reviewers stop checking the appearances of these metrics and consequently miss defects caused by them. This means that the test artifact still contains defects. Therefore, we could just consider them as pointers to certain locations that the quality assurance may need to inspect. In other words, it is up to reviewers if they prefer to inspect them. We still recommend refactoring the *true positive* findings of these metrics. This is because, we found that refactoring the *true positive* findings of these metrics could help to improve understandability and decrease complexity of the test specifications.

When it comes to the metrics *NR1* and *subjectivity*, we were not able to find any appearances of these metrics. Therefore, we could not determine if these metrics can be applied. If these metrics does not appear in the test artifacts we analyzed, it does not mean that they do not exist in other real world test artifacts. Consequently, there is a need to analyze other test artifacts that tests other application domains to investigate these smells further.

In the second step, we investigated the literature to explore if there are more bad smells that can cover more aspects of test specifications quality. In our attempt to create a comprehensive quality model we found that, there are limited papers that study the natural language test smells, which confirms the claim made by B. Hauptmann et al [4]. As a result, it hinders our ability to achieve the study goal. Because it is difficult to claim that the combination of the smell metrics introduced by [3] with the smell metrics introduced by [1], [4] will be complete to capture all quality aspects of natural language test specifications. In addition, the limitation of test artifacts provided by companies or those that exist in open-source repositories restricts our attempt to create a sufficient list of keywords that can detect all dependencies between natural language test cases. Therefore, this list has to be extended by analyzing other test artifacts that tests other application domain.

Finally, the combination of the already implemented smells in NALABS and the discovered and proposed smells cannot be applied to detect defects in other artifacts

such as natural language requirements specifications, because most of the discovered smells are tailored to detect smells in natural language test specifications according to the authors of [1], [4].

12. Conclusions

In this work we investigated which natural language test smells implemented in NALABS are sufficient to detect defects in natural language test specifications. The work also focused on extending these smell metrics to cover more aspects of natural language test quality. The purpose was to combine the existing natural language test smells in a single index of quality.

The result of our study shows that most of the natural language smell metrics implemented in NALABS have been found in the natural language test specifications. The most common findings for the smell metrics are *conjunctions, continuance, imperative, vagueness, ARI, optionality, NR2 and weakness* respectively. No findings for the metrics NR1 and subjectivity. The appearance of one finding detected by any smell metric does not necessarily lead to a defect in test specifications. Therefore, the finding still should be judged manually by reviewers. Some of these metrics can uncover many potential quality issues related to for example the understandability and complexity, while other metrics can just provide pointers to certain locations that may need to be inspected for defects, i.e., it is up to the reviewers to inspect them. As a result, the machine-supported analytical measure NALABS can work as a complement to manual reviews to make the manual quality assurance of test specifications faster and more convenient. Our study has also extended the smell metrics implemented in NALABS to cover more aspects of natural language test quality. The extension of natural language test smells is not implemented yet. In addition, the extended list of smell metrics may also not be complete and can possibly be extended further.

13. Future Work

This work mostly focused on combining existing natural language test smells in a single index of quality in order to build a comprehensive quality model. The natural language test smells that we proposed or discovered remains to be implemented in NALABS. In addition, the combined list of discovered and already implemented smells in NALABS may not be sufficient enough to capture all quality aspects of natural language test artifacts. As a future work, we propose to first check the literature and investigate if there are any new bad smells that can cover more aspects of natural language test quality, and finally to implement all smells in NALABS.

14. References

[1] B. Hauptmann, "Reducing System Testing Effort by Focusing on Commonalities in Test Procedures" Ph.D. dissertation, Institut f¨ur Informatik der Technischen Universit¨at M¨unchen, Germany, 2016. [Online]. Available: 1294074.pdf (tum.de)

[2] J. Katharina, T. Matthias and F. Houdek, "Challenges concerning test case specifications in automotive software testing: assessment of frequency and criticality," *Software Quality Journal,* vol. 29, *(1),* pp. 39-100, 2021.

[3] Rajkovic, & Enoiu, E. (2022). NALABS: Detecting Bad Smells in Natural Language Requirements and Test Specifications. [online] Available: (PDF) NALABS: Detecting Bad Smells in Natural Language Requirements and Test Specifications (researchgate.net)

[4] B. Hauptmann, M. Junker, S. Eder, L. Heinemann, R. Vaas and P. Braun, "Hunting for smells in natural language tests," *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 1217-1220, doi: 10.1109/ICSE.2013.6606682.

[5] B.I.E. Elmar, "Quality Assurance of Test Specifications for Reactive Systems" Ph.D. dissertation, Department, Georg-August-Universität zu Göttingen, Germany, 2010. [Online]. Available: main-final.pdf (uni-goettingen.de)

[6] K. Wiegers and J. Beatty, Software Requirements, Third Edition. One Microsoft Way, Redmond, Washington 98052-6399: Microsoft Press A Division of Microsoft Corporation.

[7] L. Kof, “Natural language processing: Mature enough for requirements documents analysis?” in Natural Language Processing and Information Systems: 10th International Conference on Applications of Natural Language to Information Systems, NLDB 2005, Alicante, Spain, June 15-17, 2005. Proceedings, ser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, vol. 3513, pp. 91,102.

[8] R. R. Young, The Requirements Engineering Handbook. Artech House, 2004

[9] Ian Sommerville and P. Sawyer, REQUIREMENTS ENGINEERING Good Practice Guide. Baffins Lane, Chichester, West Sussex PO19 1UD, England: John Wiley & Sons Ltd, 1997.

[10] N. Fenton and J. Bieman, Software Metrics A Rigorous and Practical Approach THIRD EDITION. 6000 Broken Sound Parkway NW, Suite 300 Boca Raton, FL 33487-2742: CRC Press Taylor & Francis Group, 2015.

[11] M. J. Ali, Metrics for Requirements Engineering. Ume˚a University, Department of Computing Science, 2006.

[12] Sommerville, I., Pieper, K. and Alm, P., n.d. *Software Engineering*. 9th ed. USA. Addison-Wesley. Pearson: 2015

[13] Masri, & Zaraket, F. (2016). Coverage-Based Software Testing. In *Advances in Computers* (Vol. 103, pp. 79–142). Elsevier. Available: Test Requirement - an overview | ScienceDirect Topics

[14] B. Hauptmann, M. Junker, S. Eder, E. Juergens and R. Vaas, "Can clone detection support test comprehension?" *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, 2012, pp. 209-218, doi: 10.1109/ICPC.2012.6240490.

[15] Y. Falcone, J. Fernandez, L. Mounier, and J. Richier. "A compositional testing framework driven by partial specifications". In TESTCOM/FATES 2007, volume 4581 of LNCS, pages 107-122. Springer, 2007.

[16] Y. Singh, Software Testing, Cambridge, UK:Cambridge University press, 2012. Available:https://books.google.se/books?hl=en&lr=&id=HdKwDwAAQBAJ&oi=fnd&pg=PT3&dq=+%22software+testing%22&ots=79gKn3oD9-&sig=1uCP7bRcr48lmLbPYdgLFTAa8IA&redir_esc=y#v=onepage&q=%22software%20testing%22&f=false

[17] M. Fowler and K. Beck, Refactoring: improving the design of existing code. Addison-Wesley, USA, 1999.

[18] M. Zhang, T. Hall, and N. Baddoo. "Code bad smells: a review of current knowledge". Journal of Software Maintenance and Evolution, vol. 23(3), pp.179–202, April. 2011.

[19] H. Femmer, D. M. Fernandez, S. Wagner, and S. Eder, "Rapid quality assurance with requirements smells," Journal of Systems and Software, vol. 123, pp. 190–213, Jan. 2017.

[20] D. Falessi, G. Cantone, and G. Canfora, "Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques," IEEE Transactions on Software Engineering, vol. 39, no. 1, pp. 18–44, 2011.

[21] K. Rajković, "MEASURING THE COMPLEXITY OF NATURAL LANGUAGE REQUIREMENTS IN INDUSTRIAL CONTROL SYSTEMS," Mälardalen university, Västerås, Sweden, Dissertation, 2019. [Online] Available: FULLTEXT01.pdf (diva-portal.org)

[22] A. Viggiato, D. Paas, C. Buzon, and C. Bezemer. "Using natural language processing techniques to improve manual test case descriptions." International Conference on Software Engineering - Software Engineering in Practice (ICSE - SEIP) Track. May. 2022.

[23] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," Empirical software engineering, vol. 14, no. 2, p. 131, 2009

[24] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering", *Proc. 18th Int. Conf. Eval. Assessment Softw. Eng. (EASE)*, 2014.

[25] K. Säfsten, M. Gustavsson, Research methodology-for engineers and other problem-solvers, Studentlitteratur AB Lund, 2019.

[26] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks", *Inf. Process. Manag.*, vol. 45, no. 4, pp. 427-437, Jul. 2009

[27] Lehner, F. "Quality control in software documentation: Measurement of text comprehensibility," Information and Management, vol. 25, pp.133–146, Sep. 1993.

[28] H. K. V. Tran, N. B. Ali, J. Börstler and M. Unterkalmsteiner, "Test-case quality–understanding practitioners' perspectives", *International Conference on Product-Focused Software Process Improvement*, pp. 37-52, November 2019.

[29] H. K. V. Tran, M. Unterkalmsteiner, J. Börstler, and N. bin Ali. "Assessing test artifact quality––A tertiary study". In: Information and Software Technology Volume 139 (2021).

[30] W. Aljedaani, A. Peruma, A. Aljohani, M. Alotaibi, M.W. Mkaouer, A. Ouni, C.D. Newman, A. Ghallab, and S. Ludi. "Test Smell Detection Tools: A Systematic Mapping Study. In Evaluation and Assessment in Software Engineering" (Trondheim, Norway) (EASE 2021). Association for Computing Machinery, New York, NY, USA, pp.170–180. 2021.

[31] M. Kummer, "Categorizing test smells", BSc thesis, *Institute of Computer Science and Applied Mathematics, University of berns*, Switzerland, 2015. [online] Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.696.5180&rep=rep1&type=pdf

[32] M. Waterloo, S. Person and S. Elbaum, "Test Analysis: Searching for Faults in Tests (N)," *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015, pp. 149-154, doi: 10.1109/ASE.2015.37.

15. Appendices

15.1. Test artifact A

1. Test

In this chapter you can find a set of tests for this safety function. A set of the tests have been selected based on the functionality of this safety function, to cover the largest parts of the function.

1. Action before test

To test this safety function correctly, other the safety functions that are not tested have to be simulated by the internal simulation. The internal simulation is controlled from the visualization as shown in the picture below (ANONYMYZED).

The Category 1 Stop Ok must be True before the start of testing. When the simulation is activated (by default setting), most of the inputs for the safety functions are simulated to the normal state (to the safety state). The remaining safety function inputs / variable / Parameters s have to by forced in the beginning of test application.

Table 3 Forced Safety Function Inputs in Test application

Input / variable	Data Type	Description
IN1	BOOL	Applicable for (X), (Y), (Z). In True during all test cases.
IN2	BOOL	In True during all test cases.
IN3	BOOL	In False during all test cases.
IN4	BOOL	In False during all test cases.

2. Test case: 1 – All inputs in normal state

Prerequisite for test: As inputs below. All related systems have to be ready and in the normal state (in the safety state) to allow test.

Test: This case tests the safety function outputs when the safety function inputs are set as in the **Inputs** tables below. The [SYSTEM] numbers are same to reach the safety state.

Expected result: The safety function outputs must be True. The safety function is in the normal state. All inputs are in the safety state and therefore the [SYSTEM] can operate.

[SYSTEM] Control Inputs	Value
v_in.[SYSTEM]1	**15**
v_in.[SYSTEM]2	**15**

Outputs	Expected Value
OUT1	**True**
OUT2	**True**

OUT3	**True**
OUT4	**True**

3. Test case: 2 – Change of [SYSTEM] number

Prerequisite for test: As inputs below. Completed Test case 1 to obtain the normal state in the tested safety function and in all related systems to allow test.

Test: This case tests the safety function outputs when the safety function inputs are set as in the **Inputs** table below. One of the [SYSTEM] numbers is changed.

Expected result: The safety function outputs must be False.

[SYSTEM]Control Inputs	Value
v_in.[SYSTEM]1	**15**
v_in.[SYSTEM]2	**5**

Outputs	Expected Value
OUT1	**False**
OUT2	**False**
OUT3	**False**
OUT4	**False**

4. Test case: 3 – Change of [SYSTEM] number

Prerequisite for test: As inputs below. Completed previous tests to obtain the necessary state of the tested safety function and all related systems.

Test: This case tests the safety function outputs when the safety function inputs are set as in the **Inputs** table below. One of the [SYSTEM] numbers is changed.

Expected result: The safety function outputs must be False.

[SYSTEM]Control Inputs	Value
v_in.[SYSTEM]1	**5**
v_in.[SYSTEM]2	**15**

Outputs	Expected Value
OUT1	**False**
OUT2	**False**
OUT3	**False**
OUT4	**False**

5. Test case: 4 – Change of both [SYSTEM] numbers

Prerequisite for test: As inputs below. Completed previous tests to obtain the necessary state of the tested safety function and all related systems.

Test: This case tests the safety function outputs when the safety function inputs are set as in the **Inputs** table below. Both [SYSTEM] numbers are changed simultaneously.

Expected result: The safety function outputs must be True.

[SYSTEM]Control Inputs	Value
v_in.[SYSTEM]1	**5**

Outputs	Expected Value
OUT1	**True**

v_in.[SYSTEM]2	**5**

OUT2	**True**
OUT3	**True**
OUT4	**True**

6. Test case: 5 – Change of both [SYSTEM] numbers

Prerequisite for test: As inputs below. Completed previous tests to obtain the necessary state of the tested safety function and all related systems.

Test: This case tests the safety function outputs when the safety function inputs are set as in the **Inputs** table below. Both [SYSTEM] numbers are changed to 0 simultaneously.

Expected result: The safety function outputs must be False. The safety function isn't in the normal state because 0 isn't allowed as the safety identity number.

[SYSTEM]Control Inputs	Value
v_in.[SYSTEM]1	**0**
v_in.[SYSTEM]2	**0**

Outputs	Expected Value
OUT1	**False**
OUT2	**False**
OUT3	**False**
OUT4	**False**

15.2. The result of analyzing Test artifact *A* by using NALBAS

Text	NW	NC	NV	Opt	Subj	NR	NR2	Weak	NI1	NI2	Con	ARI
Completed previous tests to obtain the necessary state of the tested safety function and all related systems.	17	2	0	0	0	0	0	0	0	0	2	66
6. Test case: 5 Change of both [SYSTEM] Numbers	9	0	0	0	0	0	0	0	0	0	0	50
Prerequisite for test: As inputs below.	6	0	0	0	0	0	0	0	0	0	1	57
Expected result: The safety function outputs must be True.	9	0	0	0	0	0	0	0	2	2	0	59
One of the [SYSTEM] numbers is changed.	7	0	0	0	0	0	0	0	0	0	0	49
Test: This case tests the safety function outputs when the safety function inputs are set as in the Inputs table below.	21	1	0	0	0	1	0	0	0	0	1	63
Completed previous tests to obtain the necessary state of the tested safety function and all related systems.	17	2	0	0	0	0	0	0	0	0	2	66
Completed previous tests to obtain the necessary state of the tested safety function and all related systems.	17	2	0	0	0	0	0	0	0	0	2	66
Prerequisite for test: As inputs below. All related systems have to be ready and in the normal state (in the safety state) to allow test.	25	2	3	0	0	0	0	2	0	0	3	53
A set of the tests have been selected based on the functionality of this safety function, to cover the largest parts of the function.	24	0	1	0	0	0	0	0	0	0	0	65
Prerequisite for test: As inputs below.	6	0	0	0	0	0	0	0	0	0	1	57
Prerequisite for test: As inputs below.	6	0	0	0	0	0	0	0	0	0	1	57
5. Test case: 4 Change of both [SYSTEM] Numbers	9	0	0	0	0	0	0	0	0	0	0	50
4. Test case: 3 Change of [SYSTEM] number	9	0	0	0	0	0	0	0	0	0	0	57
Test: This case tests the safety function outputs when the safety function inputs are set as in the Inputs table below. One of the [SYSTEM] numbers is changed.	28	1	0	0	0	1	0	0	0	0	1	56
. Completed Test case 1 to obtain the normal state in the tested safety function and in all	23	2	2	0	0	0	0	2	0	0	2	62

related systems to allow test.												
Prerequisite for test: As inputs below.	6	0	0	0	0	0	0	0	0	0	1	57
The remaining safety function inputs / variable / Parameters s have to by forced in the beginning of test application.	20	0	1	0	0	0	0	0	0	0	0	64
Expected result: The safety function outputs must be False.	9	0	0	0	0	0	0	0	2	2	0	60
The safety function is in the normal state. All inputs are in the safety state and therefore the [SYSTEM] can operate.	21	2	2	2	0	0	0	2	0	2	2	53
Expected result: The safety function outputs must be False.	9	0	0	0	0	0	0	0	2	2	0	60
The internal simulation is controlled from the visualization as shown in the picture below (ANONYMYZED).	15	0	0	0	0	0	0	0	0	0	1	69
The safety function isn't in the normal state because 0 isn't allowed as the safety identity number.	17	0	2	0	0	0	0	2	0	0	0	61
Expected result: The safety function outputs must be False.	9	0	0	0	0	0	0	0	2	2	0	60
Both [SYSTEM] numbers are changed to 0 simultaneously.	8	0	0	0	0	0	0	0	0	0	0	60
Test: This case tests the safety function outputs when the safety function inputs are set as in the Inputs table below.	21	1	0	0	0	1	0	0	0	0	1	63
Both [SYSTEM] numbers are changed simultaneously.	6	0	0	0	0	0	0	0	0	0	0	72
Expected result: The safety function outputs must be True.	9	0	0	0	0	0	0	0	2	2	0	59
2. Test case: 1 All inputs in normal state	9	0	2	0	0	0	0	2	0	0	0	45
The Category 1 Stop Ok must be True before the start of testing. When the simulation is activated (by default setting), most of the inputs for the safety functions are simulated to the normal state (to the safety state).	39	2	2	0	0	0	0	2	2	2	0	62
This case tests the safety function outputs when the safety function inputs are set as in the Inputs tables below. The [SYSTEM] numbers are same to reach the safety state.	31	1	0	0	0	0	0	0	0	0	1	58

To test this safety function correctly, other the safety functions that are not tested have to be simulated by the internal simulation.	22	0	1	0	0	0	0	0	0	0	0	68
Table 3 Forced Safety Function Inputs in Test application	9	0	0	0	0	1	0	0	0	0	0	58
In this chapter you can find a set of tests for this safety function.	14	0	0	2	0	0	0	0	0	2	0	50

15.3. Test artifact H

Introduction/Overall Test Objectives

This test case is intended to verify that the following functionality is working successfully:

This are all the main features of the application and our objective is to get them into working efficiently.

- Feature 1 – "Application Sliders on start-up" - User will get some necessity DO's, so that they can follow the same and protect themselves from this virus.
- Feature 2 – "Prevention" - User will get some advice on how to prevent them self from this virus.
- Feature 3 – User should get to know about the symptoms.
- Feature 4 – Total Number of cases - User should get to know about the active/recovered cases.
- Feature 5 – "Add Location" – User should be able to add multiple locations.
- Feature 6 – "App Feedback" – This feature allows user to give us a.

Limitations/Dependencies/Requirements

1.1 Test Case Limitations

* We can only test one feature at a time, the reason is if we are trying to add multiple location then application start giving both notification (You are in affected zone & You are in a safe location) which is very confusing to the user.
* When user is giving multiple feedback, our app just updates the feedback. That is one user can only give one update.
* For now, only admins can add their location if they are tested positive.

1.2 Test Case Dependencies / Assumptions

* As we are using free version of Firebase, it only allows us to use 5 physical devices to connect at once, that means only five device can give a feedback at once.
* For a user to get a notification, other users (admins) should have added location if they ae affected they only other users can get notification if they enter affected area.
* Our application is dependent on worldometer website, as one of our features show us the total number of corona virus count in united states, that feature is linked with worldometer website, if that website is down or out of service than "Total number" feature will not work.
* There's a feature (Find Location & Add Location), both of these feature are fully dependent on Google Maps, are to use that google map we are using a library "com.google.android.gms.maps.GoogleMap".

1.3 Default Setup

Database – Firebase Real Time database
Tools – Appium (For automation testing; work in progress), Android Studios (Functional Testing)
Libraries – There are various libraries we need to make our application works smoothly without any lag.

Some of those libraries are:

1) Dagger2
2) Location Services
3) Retrofit
4) Glide

1.4 Process Flow

Test Process – Out test process was basically a functional testing where we started testing every feature, we have 6 main features and after testing all those features, we went deep into testing sub-features.

For example – We started our test right after we tap on Application icon, after tapping on the icon there's 3 sec delay and in that delay, we are showing one symbol which says covid 19 and right after 3sec delay, there are 3 basic DO's which user should follow before doing anything.

Since we worked on the development part of this application, we are completely aware of how this application should work, so we wrote test cases for all the scenarios, for all the touch/click on the application and what will the expected output be. And we tested all the test cases and made a note of it whether it passed or failed.

Test Set Up – It is done in the android studio, when we developed it initially, we worked on unit testing where we tested all the individual screen and we also tested after integrating/linking all the screens which was integration testing, but we never wrote test cases back then but now we considered full-fledged test cases.

Also, we are working on automation testing where we are using Appium testing tool to test all the test cases at one time. In this we must write test cases and once we run it, it will give the test results telling how many test cases passed and how many of them failed along with the time taken to execute.

Test Report – Below is our test case report where we have listed all the test cases along with the result either it Passed or Failed the test case along with the steps on how to test and its expected results. In ideal scenario we can have more details in the report giving a tracking ID to each test, developer who developed it, date on which issue was identified, steps followed to find a bug, screenshots of these steps, Priority to be resolved, etc.,
These details will give us the Pass percentage and fail percentage of the test cases executed. Also, these details will be very beneficial to know the current quality of the product or give status report to stake holders of the projects or when is it ready to be released etc.,

2 Test Case Specification

Test case1: To test if user can download the file and open it.

Step Num	Step Description	Path and Action	Test Data	Expected Results	Pass/Fail	Comments
1.0	**Checking if user can access the application and open it.**	Download the file.	Attach the file here.	Application downloaded in the mobile.		At this time, we DO NOT have the application available on play store, so we just have an executable file to install the application.
2.0		Click on the application to open it.	-	Was able to access the application.		Tested both in android & iPhone.
End	**Test case to download and open the application was Passed without any issues.**					

Test case2: To test if it displays a flash screen of COVID-19 as soon as you open.

Step Num	Step Description	Path and Action	Test Data	Expected Results	Pass/Fail	Comments
1.0	**To test if the initial flash screen is displayed as soon as we open the application.**	Click on the application to open it.	-	Application displays an initial flash screen which says COVID-19 as soon as we click on the application to open it.		
End	**Test case to display initial flash screen of the application was Passed without any issues.**					

Test case3: To test initial safety measures screen upon opening the application. (Application just shows few safety measures with images and a short message as soon as you open the application)

Step Num	Step Description	Path and Action	Test Data	Expected Results	Pass/Fail	Comments
1. 0	**Test case to test the initial screen of the application upon opening the application.**	Click on the application to open it, which should show initial screen of safety measures.	-	Application opened in mobile. Displays initial screen - "Keep Distance: Distance means so little, when life means so much" with an image and a short message.		Since this application was developed 8 months ago it just has basic information about COVID (what we knew at that point of time)
2. 0	**To test the second safety measure.**	Swipe left to see the next safety measure. This action should move to next screen with different safety measure.	-	This left swipe moved the screen left and displayed a new screen with a different safety measure - "Wear a mask: Don't die, please buy" with an image and a short message.		
3. 0	**To test the third safety measure.**	Swipe left to see the next safety measure. This action should move to next screen with different safety measure.	-	This left swipe moved the screen left and displayed a new screen with a different safety measure - "Wash your hands: Please wash for at least 20 seconds" with an image and a short message.		
4. 0	**To test "BACK" touch button.**	There will be a "BACK" touch button on second and third screen. Once BACK button is	-	Upon touching the BACK button from 2nd and 3rd screen, it was taken back to 1st and 2nd screen respectively.		

		touched, it should take the user back to previous screen.				
5.0	**To test "FINISH" touch button.**	There is a "FINISH" touch button in the last screen of the safety measures, it should take user to main home screen of the application upon touching it.	-	Upon touching the FINISH button from the last screen, it took us to main home page of the application.		
End	**Test case to test the initial screen of safety measures was Passed successfully.**					

Test case4: To test the symptoms feature of the application (This feature will list major symptoms of COVID-19)

Step Num	Step Description	Path and Action	Test Data	Expected Results	Pass/Fail	Comments
1.0	**Test case to test the symptoms feature of the application.**	Touch "Symptoms" feature available on the home screen of the application, this should take user to a new screen where all the major symptoms of COVID-19 are listed.	-	Touch on the "Symptoms" feature took us to a screen where all the symptoms of COVID-19 was listed - like Fever, Dry cough, Headache, Breathless, tired etc., with images		We have numerous symptoms at this point of time, but the application will list only 5-6 major symptoms which was highlighted last year same time.
2.0	**To test "BACK" touch button.**	BACK touch button should navigate the user back to	-	This took us back to main home screen, tried this serval time by		

		main home screen of the application.		navigation to different screens and it worked fine.		
End	**Test case to test SYMPTOMS feature of the application was Passed successfully.**					

Test case5: To test the Prevention feature of the application (This feature will list how can we Prevent ourselves from getting infected by COVID)

Step Num	Step Description	Path and Action	Test Data	Expected Results	Pass/Fail	Comments
10 1.	**Test case to test the Prevention feature of the application.**	Touch "Preventions" feature available on the home screen of the application, this should take user to a new screen where all the major Preventions of COVID-19 are listed.	-	Touch on the "Preventions" feature took us to a screen where all the preventions was listed - like stay home if sick, cover cough, cough on your elbows, clean and disinfect, avoid close contact, cover mouth and nose etc., with images.		
20 2.	**To test "BACK" touch button.**	BACK touch button should navigate the user back to main home screen of the application.	-	This took us back to main home screen.		
End	**Test case to test PREVENTION feature of the application was Passed successfully.**					

Test case6: To test the admin login

Step Num	Step Description	Path and Action	Test Data	Expected Results	Pass/Fail	Comments

1. 0	**Test case to test admin login feature.**	Touch "Add location" feature on the home screen of the application, this should take user to admin login screen (we have restricted adding location by all users at this point of time).	-	"Add location" touch button took us to admin login screen.	Pass	
2. 0	**To test admin login for various inputs.**	Leave both email and password blanks and try to login.	-	It displayed a small pop-up at the bottom of the screen telling- "Please enter email and Password to login".		
3. 0		Enter only email and leaving the password blanks.	-	Received a pop-up telling- "Please enter password to login".		
4. 0		Enter only password and leave email blanks.	-	Received a pop-up telling- "Please enter email to login".		
5. 0		Enter incorrect email and password.	-	Received a pop-up telling- "Incorrect!!Please enter valid credentials to login".		
6. 0		Enter correct email and password.	-	Was able to login successfully.		
End	**Test case to test admin login feature was Passed successfully.**					

Test case7: To test the Total number feature of the application (Displays the total number of COVID cases in individual countries and across the world)

Step Num	Step Description	Path and Action	Test Data	Expected Results	Pass/Fail	Comments
1. 0	**Test case to test the total number of cases across the world.**	Touch the "Total number" feature from the main screen, this should take user to a new webpage where it displays total number of cases till date.	-	This feature took us to new web page called **world meter** where it had information about total number of corona cases till date, number of deaths, number of recoveries.		
2. 0		Touch countries in the link to see the total number of cases in individual countries across the world.	-	This displayed the number of cases, deaths & recoveries of all the countries across the world.		
3. 0		Further scrolling down.	-	It will display number of new cases coming in on daily bases, a linear & logarithmic curve of graph showing total cases form last one year, graph of daily new cases, daily deaths, population of the country vs corona affected count, a table of all the countries with the counts etc.,		
End	**Test case to test "Total number" feature was Passed successfully.**					

Test case8: To test the add location feature of the application (This feature will let the user add his/her own location)

Step Num	Step Description	Path and Action	Test Data	Expected Results	Pass/Fail	Comments
1.0	**Test case to test the add location feature of the application**	Touch "Add Location" feature available on the home screen of the application, this should take user to a new screen where user is directed to add his/her location manually.	-	Touch on the "Add Location" Feature took us to a screen where user can manually add the location and with that location, this application will show the result like (Active cases, notify about the area is safe or not)		
End	**Test case to test ADD LOCATION feature of the application was Passed successfully.**					

Test case9: To test the find location feature of the application (This feature will let the user add his/her own location)

Step Num	Step Description	Path and Action	Test Data	Expected Results	Pass/Fail	Comments
1.0	**Test case to test the Find location feature of the application**	Touch "Find Location" feature available on the home screen of the application, this should take user to a new screen where user is directed to google maps.	-	Touch on the "Find Location" Feature took user to a screen where user can see his actual location on the google maps.		
2.0	**Test case to test when user hovering around googles maps.**	If affected user has added his/her location on the "Add Location" features, then only he/she will get the	-	Yes, after directing it to google maps screen, notification pop ups are there.		

		notification (i.e., you Entered the affected place or you leave the affected place)				
End	**Test case to test FIND LOCATION feature of the application was Passed successfully.**					

Test case10: To test the Feedback Us feature of the application (This feature will ask user about his/her experience using this app)

Step Num	Step Description	Path and Action	Test Data	Expected Results	Pass/Fail	Comments
1. 0	**Test case to test the "Feedback Us" feature of the application**	Touch "Feedback Us" feature available on the home screen of the application, after touching it should redirect it to a new screen from which user has to select a rating and then click on submit button.	-	Touch on the "Feedback Us" Feature took user to another screen where user must select one option out of 5 (5 emoticon) after selecting user must tap on submit to successfully submitting the feedback.		
2. 0		On second screen, user will see 5 emoticons which will be the rating he/she wanted to give to developer.	-	User was able to select an emoticon.		
03.		After selecting one of the emoticons, user must touch on submit button in order to submit his/her review.	-	User was able to submit his/her review.		
04.		Just after submitting the review, user will be able to see	-	User was able to see the pop-up message.		

		one pop up saying "your review is submitted"				
End	**Test case to test FEEDBACK US feature of the application was Passed successfully.**					

Test case11: To test the Feedback Us confirmation feature of the application (This feature will let the owner know about the feedback given by the users)

Step Num	Step Description	Path and Action	Test Data	Expected Results	Pass/Fail	Comments
1.0	**Test case to test the "Feedback stored in the database" feature of the application**	Touch "Feedback Us" feature available on the home screen of the application, after touching it should redirect it to a new screen from which user has to select a rating and then click on submit button. After Submitting the feedback, it should show in Firebase Database.	-	User feedback was there in the Firebase Database in the form of point, scaling from 0-5.		
End	**Test case to test FIREBASE DATABASE feature of the application was Passed successfully.**					

Test case failed 1 - admin login (crashed)

Step Num	Step Description	Path and Action	Test Data	Expected Results	Pass/Fail	
1.0	**Test case to test admin login feature.**	Touch "Add location" feature on the home screen of the application, this should take user to admin login screen (we have restricted adding location by all users at this	-	"Add location" touch button took us to admin login screen.		

		point of time).				
02.	**To test admin login for various inputs.**	Leave both email and password blanks and try to login.	-	It doesn't work every time, sometime application gets crashed and doesn't show any pop-up text.		
03.		Enter only email and leaving the password blanks.	-	If email does not contain any "@xx.com", instead of throwing a pop-up text "email incorrect", application redirect user to main screen.		
04.		Enter only password and leave email blanks.	-	Received a pop-up telling- "Please enter email to login".		
05.		Enter incorrect email and password.	-	Received a pop-up telling- "Incorrect!!Please enter valid credentials to login".		
06.		Enter correct email and password.	-	Was able to login successfully.		
End	**Test case to test ADMIN LOGIN feature was FAILED.**					

Test case failed 2 - Find location has the location which was already added by the admin

Step Num	Step Description	Path and Action	Test Data	Expected Results	Pass/Fail	
01.	**Test case to test Find Location feature.**	Touch "Find location" feature on the home screen of the application, this should take user to check the location in google maps (we have restricted adding location by	-	"Find location" touch button took us to google maps screen.		

		all users at this point of time).				
02.	**To test find location for multiple location.**	When user is in google maps screen, he/she can only look for the places added by admin.	-	Result should show user's current location and application should tell user about the affected area. Instead of that user can only see the location which was marked by the admin.		
End	**Test case to test FIND LOCATION feature was FAILED.**					

15.4. The result of analyzing Test artifact *H* by using NALBAS

Text	NW	NC	NV	Opt	Subj	NR	NR2	Weak	NI1	NI2	Con	ARI
Touch on the Feedback Us" Feature took user to another screen where user must select one option out of 5 (5 emoticon) after selecting user must tap on submit to successfully submitting the feedback."	34	2	0	0	0	0	0	0	4	4	0	77
Touch Feedback Us" feature available on the home screen of the application, after touching it should redirect it to a new screen from which user has to select a rating and then click on submit button."	36	4	1	0	0	0	0	0	3	3	2	77
Test case to test the Feedback Us" feature of the application"	11	0	0	0	0	0	0	0	0	0	0	53
Yes, after directing it to google maps screen, notification pop ups are there	13	1	0	0	0	0	0	0	0	0	0	58
If affected user has added his/her location on the Add Location" features, then only he/she will get the notification (i.e., you Entered the affected place or you leave the affected place)"	31	3	3	0	0	0	0	0	3	3	0	62
Test case to test when user hovering around google maps.	10	1	0	0	0	0	0	0	0	0	0	52
Touch on the Find Location" Feature took user to a screen where user can see his actual location on the google maps."	22	1	0	2	0	0	1	0	0	2	0	61
Touch Find Location" feature available on the home screen of the application, this should take user to a new screen where user is directed to google maps."	27	1	0	0	0	0	0	0	3	3	0	70
Enter incorrect email and password.	5	2	0	0	0	0	0	0	0	0	2	60
Enter only password and leave email blanks.	7	2	0	0	0	0	0	0	0	0	2	54
Enter only email and leaving the password blanks.	8	2	0	0	0	0	0	0	0	0	2	55
Leave both email and password blanks and try to login.	10	4	0	0	0	0	0	0	0	0	4	50
There is a FINISH" touch button in the last screen of the safety measures, it should take user to main home screen of the application upon touching it."	28	0	0	0	0	0	0	0	3	3	0	68
To test FINISH" touch button."	5	0	0	0	0	0	0	0	0	0	0	51
Upon touching the BACK button from 2nd and 3rd screen, it was taken back to 1st and 2nd screen respectively.	20	4	0	0	0	0	0	0	0	0	4	60

There will be a BACK" touch button on second and third screen. Once BACK button is touched, it should take the user back to previous screen."	26	3	3	0	0	0	0	0	6	6	2	53
Received a pop-up telling- Please enter password to login"."	9	0	0	0	0	0	0	0	0	0	0	61
Add location" touch button took us to admin login screen."	10	0	0	0	0	0	0	0	0	0	0	54
Enter correct email and password.	5	2	0	0	0	0	0	0	0	0	2	57
These are the test cases for our manual testing. Also, we are working on scripts for automation test using Appium tool to complete this is less amount of time. This will also reduce the manual effort and will be more accurate.	41	2	6	0	0	0	0	0	6	6	2	55
To test BACK" touch button."	5	0	0	0	0	0	0	0	0	0	0	48
To test find location for multiple location.	7	0	0	0	0	0	0	0	0	0	0	55
Touch on the Preventions" feature took us to a screen where all the preventions was listed - like stay home if sick, cover cough, cough on your elbows, clean and disinfect, avoid close contact, cover mouth and nose etc., with images."	41	6	0	0	0	0	0	0	0	0	5	63
Touch Preventions" feature available on the home screen of the application, this should take user to a new screen where all the major Preventions of COVID-19 are listed."	28	1	0	0	0	0	0	0	3	3	1	73
Further scrolling down.	3	0	0	0	0	0	0	0	0	0	0	66
Test case to test the Prevention feature of the application.	10	0	0	0	0	0	0	0	0	0	0	55
Test case to test the symptoms feature of the application.	10	0	0	0	0	0	0	0	0	0	0	54
Upon touching the FINISH button from the last screen, it took us to main home page of the application.	19	0	0	0	0	0	0	0	0	0	0	58
Test case to test Find Location feature.	7	0	0	0	0	0	0	0	0	0	0	50

Result should show user's current location and application should tell user about the affected area. Instead of that user can only see the location which was marked by the admin.	30	2	0	2	0	0	1	0	6	8	2	60
When user is in google maps screen, he/she can only look for the places added by admin.	17	1	0	2	0	0	0	0	0	2	0	54
Find location touch button took us to google maps screen."	10	0	0	0	0	0	0	0	0	0	0	54
Touch Find location" feature on the home screen of the application, this should take user to check the location in google maps (we have restricted adding location by all users at this point of time)."	35	0	0	0	0	0	0	0	3	3	0	77
Was able to login successfully.	5	0	0	0	0	0	0	0	0	0	0	53
Enter correct email and password	5	2	0	0	0	0	0	0	0	0	2	55
Received a pop-up telling-Incorrect!!Please enter valid credentials to login"."	10	0	0	0	0	0	0	0	0	0	0	73
Enter incorrect email and password.	5	2	0	0	0	0	0	0	0	0	2	60
Add location" touch button took us to admin login screen."	10	0	0	0	0	0	0	0	0	0	0	54
It doesnâ€™t work every time, sometime application gets crashed and doesnâ€™t show any pop-up text.	15	2	0	0	0	0	0	0	0	0	2	63
Leave both email and password blanks and try to login.	10	4	0	0	0	0	0	0	0	0	4	50
Touch Add location" feature on the home screen of the application, this should take user to admin login screen (we have restricted adding location by all users at this point of time)."	32	0	0	0	0	0	0	0	3	3	0	75
User was able to submit his/her review.	7	0	0	0	0	0	0	0	0	0	0	49
After selecting one of the emoticons, user must touch on submit button in order to submit his/her review.	18	2	0	0	0	0	0	0	2	2	0	62
This feature took us to new web page called world meter where it had information about total number of corona cases till date, number of deaths, number of recoveries.	29	2	0	0	0	0	0	0	0	0	0	71
Touch the Total number" feature from the main screen, this should take user to a new webpage where it displays total number of cases till date"	26	2	0	0	0	0	0	0	3	3	0	66

Received a pop-up telling-Incorrect!!Please enter valid credentials to login"."	10	0	0	0	0	0	0	0	0	0	0	73
For a user to get a notification, other users (admins) should have added location if they ae affected they only other users can get notification if they enter affected area.	30	2	1	2	0	0	0	0	3	5	0	73
Test case to test the Find location feature of the application	11	0	0	0	0	0	0	0	0	0	0	53
Touch on the Add Location" Feature took us to a screen where user can manually add the location and with that location, this application will show the result like (Active cases, notify about the area is safe or not)"	39	4	3	2	0	0	0	0	3	5	2	80
Touch Add Location" feature available on the home screen of the application, this should take user to a new screen where user is directed to add his/her location manually."	29	1	0	0	0	0	0	0	3	3	0	73
Test case to test the add location feature of the application	11	0	0	0	0	0	0	0	0	0	0	52
To test BACK" touch button."	5	0	0	0	0	0	0	0	0	0	0	48
This left swipe moved the screen left and displayed a new screen with a different safety measure - Wash your hands: Please wash for at least 20 seconds" with a image and a short message."	35	4	0	0	0	0	0	0	0	0	4	74
Checking if user can access the application and open it.	10	3	0	2	0	0	0	0	0	2	2	52
These details will give us the Pass percentage and fail percentage of the test cases executed. Also, these details will be very beneficial to know the current quality of the product or give status report to stake holders of the projects or when is it ready to be released etc.,	50	5	6	0	0	0	0	0	6	6	2	66
Also, we are working on automation testing where we are using Appium testing tool to test all the test cases at one time. In this we must write test cases and once we run it, it will give the test results telling how many test cases passed and how many of them failed along with the time taken to execute.	60	6	3	0	0	0	0	0	5	5	4	67
Since we worked on the development part of this application, we are completely aware of how this application	43	3	3	0	0	0	0	0	6	6	2	85

should work, so we wrote test cases for all the scenarios, for all the touch/click on the application and what will the expected output be												
When user is giving multiple feedback, our app just updates the feedback. That is one user can only give one update.	21	1	0	2	0	0	0	0	0	2	0	52
If email does not contain any @xx.com", instead of throwing a pop-up text "email incorrect", application redirect user to main screen."	21	1	0	0	0	0	0	0	0	0	0	60
Received a pop-up telling- Please enter email to login"."	9	0	0	0	0	0	0	0	0	0	0	58
Enter only password and leave email blanks.	7	2	0	0	0	0	0	0	0	0	2	54
Enter only email and leaving the password blanks	8	2	0	0	0	0	0	0	0	0	2	54
User was able to see the pop-up message	8	0	0	0	0	0	1	0	0	0	0	44
Just after submitting the review, user will be able to see one pop up saying your review is submitted""	19	1	5	0	0	0	1	2	3	3	0	59
User was able to select an emoticon	7	0	0	0	0	0	0	0	0	0	0	44
On second screen, user will see 5 emoticons which will be the rating he/she wanted to give to developer.	19	0	6	0	0	0	1	0	6	6	0	59
Test case to test the total number of cases across the world.	12	0	0	0	0	0	0	0	0	0	0	49
Was able to login successfully.	5	0	0	0	0	0	0	0	0	0	0	53
Received a pop-up telling- Please enter email to login"."	9	0	0	0	0	0	0	0	0	0	0	58
Add location" touch button took us to admin login screen."	10	0	0	0	0	0	0	0	0	0	0	54
We can only test one feature at a time, the reason is if we are trying to add multiple location then application start giving both notification (You are in affected zone & You are in a safe location) which is very confusing to the user.	45	2	0	2	0	0	0	0	0	2	0	83
Test case to test admin login feature.	7	0	0	0	0	0	0	0	0	0	0	48
This took us back to main home screen.	8	0	0	0	0	0	0	0	0	0	0	42
BACK touch button should navigate the user back to main home screen of the application.	15	0	0	0	0	0	0	0	3	3	0	58
This took us back to main home screen, tried this serval time by navigation to	21	2	0	0	0	0	0	0	0	0	2	61

different screens and it worked fine												
BACK touch button should navigate the user back to main home screen of the application.	15	0	0	0	0	0	0	0	3	3	0	58
Touch on the Symptoms" feature took us to a screen where all the symptoms of COVID-19 was listed - like Fever, Dry cough, Headache, Breathless, tired etc., with images"	29	1	0	0	0	0	0	0	0	0	1	58
Touch Symptoms" feature available on the home screen of the application, this should take user to a new screen where all the major symptoms of COVID-19 are listed."	28	1	0	0	0	0	0	0	3	3	1	72
Swipe left to see the next safety measure. This action should move to next screen with different safety measure.	19	0	0	0	0	0	1	0	3	3	0	54
To test the third safety measure.	6	0	0	0	0	0	0	0	0	0	0	48
This left swipe moved the screen left and displayed a new screen with a different safety measure - Wear a mask: Do die, please buy" with an image and a short message."	32	4	0	0	0	0	0	0	0	0	4	71
Swipe left to see the next safety measure. This action should move to next screen with different safety measure.	19	0	0	0	0	0	1	0	3	3	0	54
Application displays an initial flash screen which says COVID-19 as soon as we click on the application to open it.	20	0	0	0	0	0	0	0	0	0	0	63
Click on the application to open it.	7	0	0	0	0	0	0	0	0	0	0	45
Click on the application to open it.	7	0	0	0	0	0	0	0	0	0	0	45
Application downloaded in the mobile.	5	0	0	0	0	0	0	0	0	0	0	64
In ideal scenario we can have more details in the report giving a tracking ID to each test, developer who developed it, date on which issue was identified, steps followed to find a bug, screenshots of these steps, Priority to be resolved, etc.,	43	0	0	2	0	0	0	0	0	2	0	85
Test Report â€“ Below is our test case report where we have listed all the test cases along with the result either it Passed or Failed the test case along with the steps on how to test and its expected results.	41	4	0	0	0	0	0	0	0	0	4	77
For example “ We started our test right after we tap on Application icon, after	53	8	0	0	0	1	0	0	3	3	4	92

tapping on the icon theres 3 sec delay and in that delay, we are showing one symbol which says covid 19 and right after 3sec delay, there are 3 bas which user should follow before doing anything.												
Our application is dependent on worldometer website, as one of our features show us the total number of corona virus count in united states, that feature is linked with worldometer website, if that website is down or out of service than â€œTotal numbr feature will not work.	47	2	3	0	0	0	0	0	3	3	0	90
For now, only admins can add their location if they are tested positive.	13	1	0	2	0	0	0	0	0	2	0	54
Test case to test admin login feature.	7	0	0	0	0	0	0	0	0	0	0	48
User feedback was there in the Firebase Database in the form of point, scaling from 0-5.	16	0	0	0	0	0	0	0	0	0	0	57
Touch Feedback Us" feature available on the home screen of the application, after touching it should redirect it to a new screen from which user has to select a rating and then click on submit button. After Submitting the feedback, it should show in Firebase Database."	46	5	1	0	0	0	0	0	6	6	2	67
Test case to test the Feedback stored in the database" feature of the application"	14	0	0	0	0	0	0	0	0	0	0	58
It displayed a small pop-up at the bottom of the screen telling- Please enter email and Password to login"."	19	2	0	0	0	0	0	0	0	0	2	61
It will display number of new cases coming in on daily bases, a linear & logarithmic curve of graph showing total cases form last one year, graph of daily new cases, daily deaths, population of the country vs corona affected count, a table of all the countries with the counts etc.,	51	0	3	0	0	1	0	0	3	3	0	91
This displayed the number of cases, deaths & recoveries of all the countries across the world.	16	0	0	0	0	0	0	0	0	0	0	60
Touch countries in the link to see the total number of cases in individual countries across the world.	18	0	0	0	0	0	1	0	0	0	0	60
Touch Add location" feature on the home screen of the application, this should take user to admin login screen (we have restricted adding location by all users at this point of time)."	32	0	0	0	0	0	0	0	3	3	0	75

To test the second safety measure.	6	0	0	0	0	0	0	0	0	0	0	49
Application opened in mobile. Displays initial screen - Keep Distance: Distance means so little, when life means so much" with an image and a short message."	26	3	0	0	0	0	0	0	0	0	2	58
Click on the application to open it, which should show initial screen of safety measures.	15	0	0	0	0	0	0	0	3	3	0	60
Test case to test the initial screen of the application upon opening the application.	14	0	0	0	0	0	0	0	0	0	0	60
To test if the initial flash screen is displayed as soon as we open the application.	16	1	0	0	0	0	0	0	0	0	0	54
Was able to access the application.	6	0	0	0	0	0	0	0	0	0	0	51
Test Set Up, It is done in the android studio, when we developed it initially, we worked on unit testing where we tested all the individual screen and we also tested after integrating/linking all the screens which was integration testing, but we never wrote test cases back then but now we considered full-fledged test cases.	56	8	0	0	0	0	0	0	0	0	2	99
And we tested all the test cases and made a note of it whether it passed or failed	18	5	0	0	0	1	0	0	0	0	4	50
Out test process was basically a functional testing where we started testing every feature, we have 6 main features and after testing all those features, we went deep into testing sub-features.	31	4	0	0	0	0	0	0	0	0	2	78
Database â€“ Firebase Real Time database Tools â€“ Appium (For automation testing; work in progress), Android Studios (Functional Testing) Libraries â€“ There are various libraries we need to make our application works smoothly without any lag. Some of those libraries are: 1) Dagger2 2) Location Services 3) Retrofit 4) Glide	47	0	1	0	0	0	0	0	0	0	0	74
* Thereâ€™s a feature (Find Location & Add Location), both of these feature are fully dependent on Google Maps, are to use that google map we are using a	30	0	0	0	0	0	0	0	1	0	0	57

library â€œcom.google.android.gms.maps.GooglеMapâ€.												
As we are using free version of Firebase, it only allows us to use 5 physical devices to connect at once, that means only five device can give a feedback at once	32	2	0	2	0	0	0	0	0	2	0	68