Optimizing Quality Assurance Process through Test Data Analysis

ELLEN NIKLASSON & JOHANNA MALMBORG
BACHELOR´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

Optimizing Quality Assurance Process through Test Data Analysis

Bachelor's Thesis

By

Ellen Niklasson & Johanna Malmborg

Department of Electrical and Information Technology
Faculty of Engineering, LTH, Lund University
SE-221 00 Lund, Sweden

LUND UNIVERSITY

2025

Abstract

This thesis presents an approach to utilize historical test execution data for improving the test process within a Quality Assurance (QA) department. The thesis research introduces two custom-built JavaScript algorithms, createTestRun and updateTestRun, designed to optimize the selection of test cases and effectively reduce redundancy in between test runs. The algorithms were developed by the thesis authors and operate on data retrieved from an existing test management API, where the createTestRun algorithm automatically creates test runs and the updateTestRun algorithm updates an existing test run based on the execution history of other test runs. The data collected are filtered and prioritized based on when they were previously executed or approved. After evaluation of the developed algorithms, it is concluded that the research demonstrates how targeted test selection and filtering historical data can significantly reduce the test scope. The result shows a promising growth in time savings, allocation of resources and offering a scalable solution to large-scale test runs.

Keywords: Regression testing, Historical data, JavaScript, API, Test Optimization

Sammanfattning

Detta examensarbete syftar till att utvärdera om användningen av historisk testdata kan bidra till en förbättrad testprocess. I rapporten presenteras två JavaScript-algoritmer utvecklade av uppsatts-författarna, createTestRun och updateTestRun, som är specifikt utvecklade för att optimera urvalet av testfall och minimera redundans mellan testkörningar. Algoritmernas bygger på historisk data som hämtats via befintliga APIer, där createTestRun automatiskt skapar nya testrundor och updateTestRun uppdaterar en befintlig testrunda baserat på andra testrundors testkörningar. Dessa algoritmer karakteriseras av en kombination av filtrering och prioritering, där testfall rangordnas baserat på deras tidigare exekveringshistorik och godkännandestatus. Vid utvärdering av algoritmerna kan det konstateras att en statistikbaserad selektion av testfall kan leda till en betydande minskning av testrundans storlek. Resultatet är lovande for tidsbesparing, resursallokering och erbjuder en skalbar lösning för storskaliga testrundor.

Nyckelord: Regressionstestning, Historisk data, JavaScript, API, Testoptimering

Acknowledgments

We would like to extend our gratitude to the company for giving us the opportunity to collaborate with them through the duration of this thesis. Their guidance and expertise in the subject were invaluable in helping us reach our goals.

We would also like to express our appreciation to our supervisors, Christin Swahn and Zahra Fathnejad for their support and constructive feedback throughout the entire process.

Ellen Niklasson and Johanna Malmborg

Contents

Preface

In this thesis work, Johanna and Ellen collaborated on all the aspects of the thesis. However, as shown in table 1, the workload was divided in a way that maximized efficiency.

Table 1. The table outlines an estimate of division of work in percent for different tasks.

	Ellen Niklasson	Johanna Malmborg
Development of createTestRun	40	60
Development of updateTestRun	80	20
Testing	40	60
Analys	60	40
Report	50	50
Presentation	20	80
Poster	80	20

1. Introduction

Software development is a key building block for today’s society. The users do not only expect the applications to work, but rather to exceed expectations. With profound expectations, the software provided to the end-user must be assured of its performance, putting a higher demand on Quality Assurance.

1.1. Background

The company was founded in 1984 with the ambition to develop network devices, e.g. printers, storage- and scanning servers. This was the starting point of ThinServer Technology, today known as Internet of Things (IoT). After several years of developing network devices, they launched the first network/IP camera in 1996.

Today, the company has grown into a global leader within the field and has thousands of employees worldwide. The company is organized in several different departments, one of which is the Quality Assurance department (QA). This department ensures the quality of their products through extensive testing.

The tests are registered and processed through an in-house developed application. This application contains a bundle of different tests, both scripted and exploratory ones. When a test is done, its result, date and executer are registered in the application and can be shown by everyone that has access to the page.

It is also possible to create test runs, which are groups of tests that a team should prioritize or execute. When a test run is created, the test leader selects relevant test cases. The test leader currently performs this task manually, reviewing every test case, checking for the properties they are looking for, and then use their judgment to pick the relevant ones, which requires significant time and effort.

A possible solution for this problem is to fetch and analyze test data that can be used to generate varied and meaningful test runs in the future. The statistics from old test runs could be used to filter and prioritize tests based on when they were previously executed or approved. This would facilitate the workload for both test leaders and employees who run the tests, making the test runs more varied and meaningful.

Statistics will be compiled based on multiple years of test data which have been stored in the test management database, leveraging pre-existing APIs. Extracting relevant information from API responses will be done to manage raw data.

Through efficient testing processes, it is anticipated to improve product quality as various types of errors are detected earlier. Consequently, the thesis will support the company's overarching quality objectives and long-term vision.

1.2. Purpose

The purpose of this thesis is to improve the test processes and quality assurance practices within the QA department by processing and analyzing historical test data. The company has a tangible need for this enhancement, which is why this topic was selected. The company recognizes the potential benefits of utilizing data more efficiently in the planning and prioritization process, resulting in time- and resource savings.

1.3. Reason for thesis

This thesis was chosen based on interests in learning more about how testing works in a real-life company. One of the authors already had some insights from working as a tester, which gave the opportunity to conduct a more in-depth analysis of this subject and help the company with analyzing what could be improved with their system.

Furthermore, the thesis could potentially serve as inspiration for other companies with similar quality assurance practices. This since the findings can be applicable in different contexts where testing and prioritizing are vital.

1.4. Objective

The goal of the thesis is to analyze historical test data to generate a selected test run, combining inputs from recent test creations and historical test outcomes. Hopefully this will reduce redundant work and part of the test execution could be spent on running test cases that may be more important to execute. The thesis will serve as a proof-of-concept regarding how the fetched data can improve the QA-departments quality assurance process and how it can be used to create variation in the test runs.

1.5. Research Questions

Following questions will be answered through the thesis:

1. How are the test runs created now?
2. How much time does it take to create a test run?
3. Which existing APIs are most suitable for collecting relevant data?
4. What data are essential for achieving the objective of this thesis?
5. How can the analyzed data be presented to provide actionable insights for the QA-department?
6. Are test runs updated when another team in the QA-department pass a test for the same release?

1.6. Limitation

The algorithm for creating a test run shall be developed specifically for the team collaborating on this thesis and may not be directly applicable or useful for other teams within the QA department.

2. Technical background

The technical background is meant to give a greater understanding of technical systems and tools used while working towards the thesis goals. This chapter will provide an overview of the various technical systems, tools, and concepts employed throughout this thesis, including embedded systems, testing methodologies, Application Programming Interfaces (APIs), and programming languages.

2.1. Embedded systems

An embedded system is a combination of both hardware and software, working together through a microprocessor- or microcontroller. These usually provide a special function and is a part of a larger electrical system. [1] The QA department, which is the subject of this thesis, exclusively tests embedded systems, rendering them a significant aspect of this research.

2.2. Testing

Testing is a key element of the development cycle, detecting bugs and errors. There are therefore a lot of different test approaches, but this section will focus on scripted and exploratory tests performed with regression testing, as they are the primary sources of historical data used in this thesis.

2.2.1. Regression testing

Regression testing is a testing technique where software is re-tested after code modifications to ensure it continues to work as intended. Although it helps to ensure that new errors are detected, it is a costly approach, particularly with large test runs. Various techniques exist for selecting test cases, including prioritization and minimization. Minimization is a technique where redundant test cases are excluded based on their functional coverage, whereas prioritization orders test cases by priority of execution. Selecting either too few or too many test cases can lead to undetected faults, making minimization and prioritization an essential aspect of regression testing. [2]

2.2.2. Scripted

Scripted testing means that the test follows a step-by-step procedure to ensure that every necessary feature undergo execution during the test. The structured and clear nature of the scripted test often help the executer to determine if the requirements are met or not. It can also be a good alternative when performing complex test cases. However, there are also some disadvantages to scripted test cases. If no tests are designed to verify a specific feature, it is unlikely that any bug or error in that feature will be detected. [3]

2.2.3. Exploratory

Exploratory testing is using the approach of exploring without clear instructions and let the executer base the test on their experience, expertise and knowledge. With this test strategy a wide variety of bugs or errors are usually detected. [4]

2.3. APIs

API (Application Programming Interface) is combined protocols and/or rules that make it possible for different programs to communicate. It allows them to share data, services or functionality, making it easier to build new applications by leveraging existing ones. With APIs, it is possible to only share relevant information and therefore keep sensitive information secured and hidden from external sources.

When using APIs, the communication can be explained in a client-server paradigm. The client initiates a request to access resources or services, whereas the server responds with the requested recourse or action. The API itself works as an intermediary handling the communication between them. [5]

2.3.1. REST API

A REST API is built on the functions of HTTP (Hyper Text Transfer Protocol). It uses the common methods as GET, PUT, POST and DELETE to perform operations on resources. REST APIs can be used to create data, content, algorithms and other digital capabilities using web URLs. [6]

2.4. Database

Within the field of computing a database is a collection of data, organized, stored and managed using a Database Management System (DBMS). [7]

2.4.1. MySQL

MySQL is a type of DBMS used for relational databases called Relational Database Management System (RDBMS) that uses Structured Query Language (SQL) for database operations. [8] The historical test data handled in this thesis work, are fetched from a MySQL database.

2.5. Tools

This section describes and explains the tools used for developing the algorithms to generate efficient test runs.

2.5.1. Postman

Postman is a platform created for building and using APIs. The user interface is well planned to make every step of API use simple and effective. For the intent of this thesis, postman was used to make GET calls and visualize the response in a clear manner. [9]

2.5.2. GitLab

GitLab is a web-based DevSecOps platform that can develop, secure and operate software. GitLab provides a Git-repository manager that includes features like wiki, issue-tracking and continuous integration and development pipeline. GitLab was used during the thesis development stage to store and effectively co-program. [10]

2.5.3. JavaScript

JavaScript is a dynamic programming language widely used for web development. Asynchronous operations are supported by JavaScript which allows network requests without blocking code execution making it a good choice for API use. [11]

2.5.4. Node.js

At Node.js webpage they explain that “Node.js is an open-source, cross-platform JavaScript runtime environment that lets developers create servers, web apps, command line tools and scripts.”. [12] This means that it is a software, which source code can be examined and edited by the public, that supports multiple operating systems and can execute code written in JavaScript. [13] For the thesis Node.js was relevant to execute JavaScript code.

2.5.5. Visual Studio Code

Visual Studio is a well-established Integrated Development Environment (IDE). It provides developers with a variety of features such as writing and editing code, debugging and running/deploying code/applications. JavaScript and Node.js is built-in without the need for downloading extension packages. [14]

2.6. Test Management tool

The company utilizes its in-house developed test management tool to organize test runs, track test results, and enable collaboration among QA team members via a web-based user interface. The test management tool also provides a REST API to work with test runs, test cases and test results.

Test cases serve as the foundation of the application, existing in different versions with multiple results. Each result is thoroughly documented, including date, status, firmware release and corresponding test case version.

The application uses tags and feature tags for categorizing test cases which can help to filter test cases in a multitude of ways. The tags are often used to get test cases with specific properties when creating a test run.

3. Method

To reach the thesis goal a clear but flexible plan was concluded to be of great importance. Therefore, the first objective was to create a layout for the plan.

3.1. Process planning

At the beginning of the thesis project, a Gantt chart was developed to outline the estimated timeframe for various tasks (see figure 2 in appendix). To assess the different time needed for each phase, the knowledge from prior projects played a big role. The chart was then used to ensure that the thesis development work remained on schedule. The thesis work generally adhered to the planned phases of elicitation, research, development, testing, user testing, iteration based on feedback, reporting, and presentation. However, testing evolved into an ongoing process throughout the development phase.

3.2. Elicitation and research

To initiate the thesis work, a few days were allocated to gather information that were helpful for the understanding of the task and get familiarized with the needed tools. Furthermore, meetings/interviews with test leaders and testers were conducted to gather information about what kind of parameters, filtering and prioritization that they expected for future test run creations.

3.2.1. Literature studies

During the literature review, considerable study time was spent analyzing the company’s API documentation to gain a deeper understanding of its capabilities and properties.

Relevant literature on related subject that were discussed in section 2.1 were found with LUBsearch, while other technical information was gathered through Google Search.

The Gantt chart allocated two weeks for research, yet in practice, only a few days were devoted to this stage. However, technical research continued through each development phase, extending beyond the initially planned timeframe.

3.2.2. Interview/meeting

In the beginning of the thesis process, meetings with our supervisor concluded what kind of filtering's that were expected for the test cases. In the beginning of April, when the first algorithm was finished, another meeting with our supervisor and a test leader was held to gather information regarding what priorities the different filtering functions should have.

Interview questions and answers:

In what order do you believe that the filtering functions should be prioritized?

Test Leader: *That will be decided with time after using your algorithm in real scenarios. You can choose for now, but make it possible for us to change these priorities.*

Company supervisor: *Start with test cases that have failed a couple of times during the latest test runs as priority one.*

Do you have any feedback regarding the algorithm?

Test Leader: *We would like for you to make it possible for the algorithm to create new test runs in Test Management Tool, the* *current print outs are not sufficient. Furthermore, it would be nice if you could create another algorithm that updates current test runs if a test case is passed from another department.*

The second meeting revealed that POST and DELETE permissions for the Test Environment Tool's staging environment were required.

3.2.3. Application testing

While developing an external tool extending the existing system's functionality, it is crucial to be familiar with how the current system is used. To get accustomed with the test management tool used for creating test cases and test runs, tasks were performed. The different tasks included creating test runs based on different tags, features tags and prioritizing different test cases.

3.3. Development of algorithms

After extended information gathering from literature and application testing a broader understanding of the system and handling of historical data was created. The new information helped to brainstorm ideas on how the test process could be improved and were then discussed together with supervisors. Since the QA team uses regression testing as explained in section 2.2.1 today, where the test leader chooses test cases to be added to a test run manually, it was decided together with the test leader and supervisor that the thesis work should include automation of minimization and prioritization.

To achieve this, two algorithms were developed. One handling the creation of a new test run and one updating an already existing test run. The first algorithm is supposed to filter test cases into three different prioritizations based on their failure rate, latest result and execution release or execution date. The second algorithm checks if the test cases have been passed on/after a certain firmware release or date. If the chosen test run contains test cases that meet these criteria, they are supposed to be deleted from the test run.

With the algorithms in mind, the raw data returned by the APIs were analyzed to find relevant properties needed to create the different filtering functions. The analysis was made by reading and understanding the APIs documentation but also by trying the different API calls with Postman.

The developed algorithms were made using VS Code as an IDE and JavaScript as the programming language. JavaScript's built in fetch functions were utilized to be able to retrieve data from the already existing APIs.

3.4. Testing

The produced algorithms were iteratively tested manually throughout the development process. When these were completed, user testing was also done.

3.4.1. Manual testing

Testing started early in the development to ensure seamless integration with the company's APIs. To be able to use the company's APIs, the algorithms needed to include authorization and cookie content in every fetch operation. Initially, this was tested using a randomly selected API endpoint. The resulting JSON output was compared with Postman, verifying a successful connection establishment.

Every function developed within the algorithm underwent rigorous testing, with outputs compared to both Postman and the company's Test Management Tool. Incorrect outputs and/or errors were identified a couple of occasions, prompting targeted code corrections to achieve desired functionality.

When creating test runs in the staging environment of the Test Management Tool, algorithm 1 called createTestRun was used and the result was evaluated by comparing it with print outs in the terminal. Test runs were also created manually, set with a passed test result for some test cases, to then confirm that algorithm 2 called updateTestRun works as intended.

Example of conclusive test:

1. Create a project manually in the staging environment.
2. Create a test run manually in said project.
3. Run algorithm 1 to add test cases to the test run. Verify that the test run is correct by comparing it with the output in the terminal.
4. Create another test run manually in any project. Add test cases manually and choose on which release they should be run on.
5. Mark 3 test cases in the manually created test run as passed for the selected release.
6. Run algorithm 2 and verify that the 3 test cases marked as passed in the manually created test run, now are deleted from test run created with algorithm 1.

3.4.2. User testing

When the two algorithms were finished, a user test was conducted with a test leader. The algorithms along with the configuration file and a README file were provided to the test leader. This made it possible to evaluate if the process of running the algorithms was understandable by reading the step-by-step instructions in the README file and to identify any potential issues that may have been overlooked during manual testing.

During the user test, it was observed that some necessary information was missing from the README file.

1. Where one finds authentication information.
2. That the test run itself needs to be created manually in the Test Management Tool.
3. Where to find the Test Run id.
4. That there is more than one way to specify the firmware release.
5. How to install node.js.

Additionally, the user test exposed an issue where the execution would occasionally become stuck in the terminal, despite completing all API-calls. This problem was resolved by incorporating a *process.exit()* command.

3.5. Evaluation

To determine if the algorithms improve the creation and efficiency of the test process, a comparison between results from manually created test runs and test runs created with the createTestRun algorithm were made, considering the number of test cases included in the test runs.

Since the process of creating a test run today is based on the test leaders' experience and manual work, evaluating statistics is not sufficient. Interviews and user testing with test leaders were of most importance, providing valuable insights that, when combined with statistical data, formed a solid foundation for evaluation.

3.6. Company collaboration

The development was only conducted at the company office and therefore a lot of communication was in person. However, some of the communication, meetings and quick questions were made over Microsoft Teams. A page including weekly goals and future milestones was shared with the supervisors so they could follow the process closely.

3.7. Source evaluation

Information gathered from reference [1], [2] and [7] are considered reliable due to the academic credibility of the sources, ScienceDirect, IEEE Xplore and NE. These sources are peer reviewed or additionally controlled and fact checked. The reviews are performed by experts in the field, making them a reliable information source.

Within the field of computer science, there are a lot of companies providing their official documentation. While finding descriptions of technical terms and tools these documents are a trustworthy resource. References [5], [6], [9], [10] and [12] are all official documentation collected from company websites. These are maintained by the organizations that create or own the tools or technologies described in the technical background in section 2.

However, not all information can be found on company websites but luckily the tech industry has a lot of well-established educational platforms and media. All the references [8], [11], [13] and [14] come from widely recognized platforms within the programming community. GeeksforGeeks and freeCodeCamp are both maintained by professionals in their fields, specializing in tutorials. InfoWorld is a long-standing tech publication known for professional articles.

References [3] and [4] are from Testsigma, which offer practical insights into the modern software development practices. Although they do not use peer-reviews, these resources are written by industry experts.

3.8. AI Tools

In addition to all information sources, AI were utilized to assist with language and grammar throughout the writing process of this report. The information written in this report and the code that has been developed were not created using AI.

4. Result

The thesis work concluded in the two algorithms with goals described in section 3.3.

4.1. Configuration file

A JSON based configuration file is utilized to add the required parameters to the two distinct algorithms (see figure 3 in appendix). This contains parameters such as authentication credentials to validate user permission. Additionally, this is where one specifies the release or date for the latest execution, applicable tags and failure rate threshold for algorithm 1's prioritization logic. A test run id is also needed for the algorithms to know what test run to add test cases to, or which test run to update. When algorithm 2 is to be run, neither date, tags nor failure rate are required.

4.2. CreateTestRun algorithm

This algorithm retrieves test cases containing chosen tags and adds them to a general map. It utilizes properties such as 'status', 'release' or 'createdAt', which are obtained from the data fetched via REST APIs.

The 'status' property indicates the outcome of a test case execution. The 'release' property specifies the firmware version associated with a test case execution. The 'createdAt' property denotes the date of a test case execution.

During the evaluation. the 'status' property is used to assess the lates 10 results of each test case against the chosen failure rate, if the latest result shows a "failed" status or the absence of a status property (indicating no prior execution). If the test cases meet any of these criteria, they are added to the priority map with different priorities depending on which criteria they satisfy, and then they are deleted from the general map. If fewer than 10 results exist, the failure rate is calculated accordingly.

The ‘status’ property and ‘release’ or ‘createdAt’ property are combined in a filtering function to determine whether a test case requires re-execution or not. This is done by comparing the firmware release of the latest execution with the chosen release, and that the status indicates “passed” as a result for this execution. If both conditions are met, the test case is added to a separate map and removed from the general map. In case a release is missing from the configuration file, the date of the last execution is compared with a chosen date instead.

Another filter relies solely on the ‘release’ or ‘createdAt’ property. This priority category is determined by comparing the firmware release of the latest execution with the chosen release. If the execution firmware release is older than the chosen release, the test case is added to the priority map. As with the previous scenario, if a release is missing from the configuration file, the comparison is made with a chosen date instead. The test cases that undergo this part of the algorithm didn’t meet the criteria in the other filtering functions above.

When all test cases have undergone the filtering functions, REST APIs are used to add the test cases from the priority map to a test run and set the priority for every test case. The system applies a hierarchical approach to prioritize test cases based on predefined criteria described above.

Initially, test cases with a failure rate equal to or exceeding the specified threshold (defined in the configuration file) are automatically assigned Priority 1. These high-risk test cases are given top priority for execution.

Next, test cases that pass through the initial filter are subjected to a secondary evaluation, which assesses whether they have successfully passed on or after the selected release or execution date (as specified in the configuration file). If they fulfill these requirements, they are excluded from the test run, as they have already demonstrated success in recent executions.

The remaining test cases are then evaluated based on their latest status. If a test case has a failed or undefined status, it is assigned Priority 2, indicating a need for re-execution or further investigation.

Finally, if a test case has not been executed on or after the selected release or execution date, it is assigned Priority 3. This ensures that test cases with outdated execution histories are revisited and re-evaluated.

The test cases without priority will be printed in the terminal, and a question will pop up asking the user to choose if non prioritized test cases should be included in the test run or not. The test cases that met the criteria in the second filtering function and therefore will be excluded from the test run, will be printed in a separate text file in case the user wants to evaluate the result. If the user answered 'no' on the question regarding adding non prioritized test cases to the test run, these will be printed in this text file as well.

The information gathered could look like the data in tables 2 and 3.

Table 2. Table illustrating an example of status data that is fetched using API-calls

Test Case	Status									
ID 1	Failed	Failed	Passed	Failed	Failed	Passed	Failed	Passed	Passed	Passed
ID 2	Passed	Passed	Passed	Passed	Passed	Failed	Passed	Passed	Passed	Passed
ID 3	Failed	Passed	Passed	Passed	Passed	Passed	Passed	Passed	Failed	Failed
ID 4	--	--	--	--	--	--	--	--	--	--
ID 5	Passed	Failed	Failed	Passed	Passed	Passed	Passed	Passed	Passed	Passed
ID 6	Passed	Failed	--	--	--	--	--	--	--	--

Table 3. Table illustrating an example of the status, release and execution date of the latest test run for each test case

Test Case	Status	Release	Execution date
ID 1	Failed	12	null
ID 2	Passed	12	null
ID 3	Failed	12	null
ID 4	--	--	--
ID 5	Passed	11	null
ID 6	Passed	12	null

Scenario based on table 2 and table 3, with a chosen failure rate of 40 % and a release of 12:

ID 1: Test case with ID 1 would be added to the test run with priority 1 since its failure rate is greater than 40%.

ID 2: Test case with ID 2 would be excluded from the test run, since it passed on the chosen release

ID 3: Test case with ID 3 would be added to the test run with priority 2, since it failed during the last execution but has a failure rate less than 40%.

ID 4: Test case with ID 4 would be added to the test run with priority 2, since it never has been executed

ID 5: Test case with ID 5 would be added to the test run with priority 3, since it hasn't run on release 12 or later

ID 6: Test Case with ID 6 would be added to the test run with priority 1, since its failure rate is greater than 40%

4.3. UpdateTestRun algorithm

The second algorithm was developed to help different teams within the QA department to collaborate. The idea was to see if another team has performed a test case on the latest firmware and therefore could be removed from the other team's test run, to reduce recurrence.

The algorithm retrieves test cases and corresponding results associated with the selected test run, organizing them into categorized maps. Subsequently, it filters out all the test cases already executed during the test run, preventing deletion of existing results, resulting in a map containing only unperformed test cases within the current test run (see figure 1).

Note that if a test case is set as priority 1, this algorithm ensures that only test cases that have received a passed status after the creation date of the chosen test run will be removed.

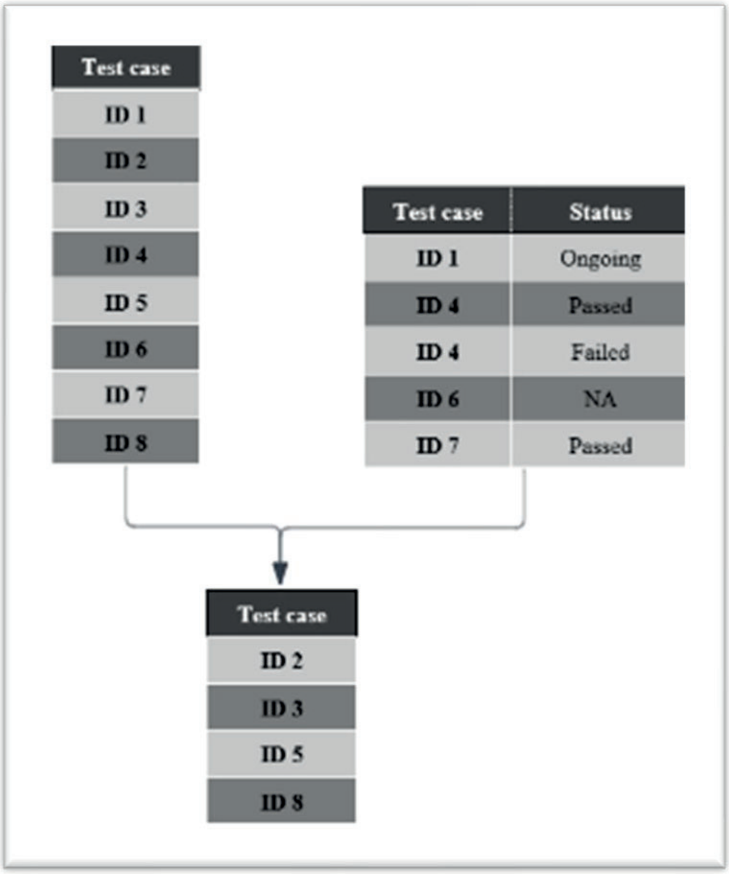

Fig. 1. Illustration of the test case map transformation after excluding test cases with successful results in the same test run.

For each test case, their latest result is retrieved and checked whether it was conducted using the latest firmware and achieved a "passed" status. Every test case that fulfills the conditions will be deleted from the current test run, using DELETE calls.

4.4. Analysis of results

This section assesses the effectiveness of the algorithms in reducing time consumption in comparison to creating test runs manually, but also how the algorithm can affect test execution redundancy, highlighting the observed changes.

4.4.1. Statistics

For analyzing purposes five test runs were created with the createTestRun algorithm given the values: release "12.0.0", failure rate 0.4 (40%) and tags 1-5 in the configuration file. The observed statistics can be found in tables 4-6.

Table 4. Illustrates the number of selected test cases when using the createTestRun algorithm to create a test run.

Tag name	createTestRun algorithm	Possible test cases	Filtered out test cases
Tag 1	861	1983	1122
Tag 2	231	317	86
Tag 1 & Tag 3	207	776	569
Tag 1 & Tag 4	17	89	72
Tag 1 & Tag 5	87	224	137

In table 4 the number of selected test cases in the test run, possible test cases for the chosen tags and test cases filtered out are displayed. The values presented provide insight into the efficiency of the createTestRun algorithm in reducing the test case load in a test run. It is evident that the algorithm is effective in filtering out a significant proportion of test cases, with 56% of test cases filtered out for tag 1 and 26% for tag 2. The three combinations are the most commonly used when creating test runs, resulting in a reduction of test cases with 77%, 80% and 58% respectively.

Table 5. The table displays the number of test cases assigned to different priority levels (1, 2, 3, or no priority) for each tag combination.

Tag name	Priority 1	Priority 2	Priority 3	No priority
Tag 1	101	135	405	220
Tag 2	20	18	151	42
Tag 1 & Tag 3	26	55	36	90
Tag 1 & Tag 4	5	6	2	4
Tag 1 & Tag 5	38	19	3	27

Furthermore, the createTestRun algorithm also include prioritizing test cases depending on their former results. The assignment of different priorities to test cases based on previous results enables the QA department to allocate resources more efficiently, focusing on high-priority test cases first. This approach also provides a scalable solution for very large test runs, making them more manageable (see table 5).

Despite the advantages, there are some flaws with the createTestRun algorithm. While analyzing all test cases within each priority, no priority and priority 2 stands out.

The test cases with no priority have managed to go through each filter without complying with their criteria. After further investigation it is evident that these test cases latest status is something else than "passed" or "failed", hence "ongoing", "NA", "planning", "blocked" etc. As described in section 4.2, the user will be able to see which test cases fall under the no priority category and then choose if these should be included in the test run or not. This ensures a overview for the test leader, who can make an informed decision based on the print outs.

Test cases can get priority 2 by either having their latest status as "failed" or undefined. Testcases that match the undefined criteria have no previous results, meaning they've never been executed, are the ones of interest. When inspecting each test case, it became evident that some test cases are not possible to perform due to lack of instructions and could therefore be considered "garbage" data.

Table 6. The table illustrates the number of test cases never been run and number of "garbage data" test cases per tag combination.

Tag name	Total	Garbage data
Tag 1	36	6
Tag 2	7	4
Tag 1 & Tag 3	11	1
Tag 1 & Tag 4	0	0
Tag 1 & Tag 5	1	0

As previewed in table 6, a potential drawback of the data is the presence of incomplete information so called "garbage" data, with 17% and 57% of the total number of never run test cases being incomplete. However, when using well-established combinations of tags to create test runs, this issue becomes negligible, as these combinations are frequently used and well maintained. Therefore, it is concluded that this limitation has a minimal impact on the collaboration team.

To analyze the updateTestRun algorithm the previously created test runs where used. One week after the creation of the test runs, the updateTestRun algorithm was used on each one. The statistics can be seen in table 7.

Table 7. Illustrates five different test runs before and after updateTestRun is used on each test run.

Tag name	Before	After
Tag 1	861	793
Tag 2	231	228
Tag 1 & Tag 3	207	184
Tag 1 & Tag 4	17	15
Tag 1 & Tag 5	87	73

The analysis of the data in table 7 assesses the impact of the updateTestRun algorithm by counting the number of test cases that are deleted. These deleted test cases had already been performed by another team or in another test run, thereby creating redundancy if also performed in the selected test run. The results show that for Tag 1, 7,9% of test cases could be removed; for Tag 2, 1.3% of test cases could be removed. The combinations of tags yielded the following decreases: Tag 1 and 3 resulted in a 6.3% decrease, Tag 1 and 4 in an 11.8% decrease, and Tag 1 and 5 in a 16.1% decrease.

4.4.2. Expert feedback

To further analyze the algorithms that were developed during the thesis, interviews were conducted with experts that in this case were test leaders from the collaboration team and another team.

In figure 4 (see appendix), an interview with another team's test leader is presented. The interview revealed that, under current parameters, the test leader believes that the algorithms' application is limited. It became evident that if the algorithm's would be updated to compare products and specific test runs instead of release and tags, they would have the potential to reduce time, redundancy and resources for this team as well.

In figure 5 (see appendix), an interview with two test leaders from the thesis collaboration team is presented. This interview revealed that both test leaders believe that the algorithms will save time and redundancy, at least for some combination of tags. They appreciate the ability to prioritize test cases but can see improvement in how the prioritizations are decided. For both the algorithms, the test leaders have some ideas of improvement to make them more user friendly and secure.

4.5. Problems and solutions

During the thesis process, a couple of problems were encountered. It was discovered that older versions of test cases are not archived, despite the latest version being archived. As a result, inactive test cases were being retrieved for the selected tag. This problem was solved by filtering the data by removing the test case id's where the latest version has its "archived" property set to true and then adding the resulting test cases to the general map.

Another problem arose due to the Test Management Tool's capacity to handle a large volume of concurrent PUT requests. This resulted in error messages 502 and 504 when adding test cases and their prioritizations to a test run. To overcome this challenge, a batching solution was implemented, limiting the number of concurrent API requests to 70.

5. Conclusion and discussion

Overall, the findings suggest that the historical data used in createTestRun algorithm and updateTestRun algorithm holds considerable promise for optimizing test case selection and improving test efficiency - thereby improving the QA departments quality assurance process. This is proven by the calculated data shown in section 4.4.1 as well as the interviews presented in 4.4.2 that were conducted with test leaders. The algorithms are effective in reducing the test case load in test runs as shown in table 4 and 7, minimizing redundancy and time that need to be spent to execute test cases that with minimal risk of failing, but also create variation in the test runs. The prioritization of test cases is also proven to be efficient (as shown in table 5) when utilizing the historical test data, making a test run more efficient and straightforward regarding what test cases are of more importance. However, there are possible improvements to be made for the algorithms to work in the long run, such as the need to be able to modify the priorities assigned by the different filtering functions.

5.1. Discussion

It is worth noting that some unexecuted test cases were created multiple years ago or lack execution instructions, making them irrelevant and not representative of new test cases as expected. The ideal solution would be for test case creators to archive unused cases, but currently, it appears that only a small portion of this data is included in the test runs generated using the tags employed by the thesis collaboration team.

In addition to maintaining the test management tool by achieving outdated test cases, implementing a definitive standard for various values would significantly enhance the handling of historical data. Currently, certain values are given as strings, allowing users unlimited freedom to enter their input. This flexibility poses challenges when attempting to develop algorithms to analyze these values in the historical data, as it is difficult to account for every possible input scenario. During the development of the algorithms this problem became apparent while handling the firmware release. Their structures would vary significantly, making comparisons extremely challenging. Examples of these varied structures and values include 11.12.5, CVP-20.3.4 and M2026LEmKII_OE_444.

For the timeframe of this thesis two weeks were allocated for analysis. However, to get data built on a good basis requires multiple measurements and therefore having more time would have been beneficial in conducting a more thorough analysis. Specifically, running multiple iterations with the updateTestRun algorithm (see table 7), over several weeks, rather than just a single instance, allowing the collection of more comprehensive data. Additionally, the extended timeframe would have enabled the test leaders to elaborate their responses and provide more detailed feedback, ultimately enriching the overall quality of the results.

Finally, the interviews that were conducted throughout the thesis also provided insights into potential improvements, such as adding functionality to evaluate the history of a certain product or chip in combination with the test case and specific test runs. Although, this is out of this thesis scope and will not be discussed further.

5.2. Research Questions

The following section answers the research questions stated in section 1.4.

- How are the test runs created now?

  Today each test run is created by a test leader selecting test cases manually based on experience and prior knowledge. Each test case is associated with one or more tags, which allows the test leader to filter out test cases with these tags when test creating test runs. Once the test leader has chosen the desired to filter their test cases on, they then select multiple or single cases that should be included in their test run. The test leader thereby manually minimize test runs and prioritize test cases.

- How much time does it take to create a test run?

  The test run only takes a few minutes to create. However, it is very difficult to prioritize or filter test cases and is currently done simultaneously as the test cases are executed.

- Which existing APIs are most suitable for collecting relevant data?

  Four existing APIs were utilized to collect relevant data. Two APIs fetched test cases by tag name or test run ID, while the other two retrieved results by test case ID or test run ID.

- What data are essential for achieving the objective of this thesis?

  All the APIs were HTTP GET method was used contained data in JSON format. Since test leaders use tags to filter test cases when creating a test run, tag names were used to collect associated test cases. Each test case has a unique ID, which could be used to retrieve the result (status) of an execution of a test case. The statuses of every test case were of importance to be able to automatically prioritize the test cases in a test run. Furthermore, every status for a test case contained data about what release and date the status was created on, which could be used for prioritization and minimization as well.

- How can the analyzed data be presented to provide actionable insights for the QA-department?

  The two algorithms created will communicate with the Test Management tool to create and update actual test runs. The results will thus be visible in this tool, and any test cases that are excluded or removed when running the algorithms will be logged in separate text files, enabling test leaders to track this information and gain a better overview.

- Are test runs updated when another team in the QA-department pass a test for the same release?

  There is no way to automatically update a test run depending on other team's results. However, the results from other teams are in the summary for each test case and can therefore be checked manually.

5.3. Ethical reflection

This chapter will include ethical reflection which is a powerful tool to develop respect for the concerns of others.

5.3.1. Confidentiality

When working within a corporation there is confidential information that the company wants to keep internal and not spread to external sources. Not all information is outspoken to be confidential but instead each employee needs to use their common sense to keep information from spreading. When working with the thesis it is of most importance that any code snippet and figures used in the report or presentation do not contain any confidential information.

5.3.2. Code of conduct

During the thesis selection process, finding a company whose code of conduct the thesis authors stands behind is substantial. When developing for a corporation their work will be used for achieving the company's business goals and therefore it is important that they feel that they stand behind the company's view of their role and influence in society. Taking this into account will ensure that the work has a positive impact on society and reflects their personal beliefs.

5.4. Future development opportunities

Even though the development of the createTestRun- and updateTestRun-algorithms mark a great starting point for benefitting from historical data, there are still numerous ways for further development and enhancement. The following subsections outline two propositions of directions for future work.

5.4.1. Integration

While the thesis goal was a proof-of-concept an apparent next step in development would be integrating the functionality of createTestRun- and updateTestRun-algorithm into the test management tool used by the QA department. The integration would enable the QA team to leverage the benefits of automated test case selection and optimization. Furthermore, it would be a great opportunity to continue the improvement and refinement of the algorithms through real-world usage and feedback.

5.4.2. Artificiell intelligence

Another way to further develop the algorithms could be to incorporate AI in different ways. Machine learning could be a good tool to be able to predict the likelihood of test case failures or successes, by training machine learning models on historical test data. By using the machine learning algorithm neural networks when creating a test run it would give a “simulated cognitive” ability to select test cases based on the historical data and patterns.

6. Terminology

In this chapter terms and shortenings are listed following their explanations.

6.1. Terms

- Test leader – Creates and manages the test runs.
- Project – A folder for different test runs.
- Test run – A collection of selected test cases.
- Quality Assurance (QA) – Focuses on verifying that products meet specified requirements through extensive testing and evaluation.
- Passed/Failed – Outcome of a test case.
- Regression testing – Re-testing after changes or fixes.
- Firmware – A software program embedded into a hardware device. It provides the necessary instructions for the device to operate, controlling its functionality and behavior.
- Staging environment – A pre-production environment mimicking the production environment. Often used by developers to test code and gather feedback before launching.
- AI – Artificial intelligence.
- Garbage data – Refers to data in the memory/database of a computer system which will not be of any use in the future.

References

[1] S. Bhunia and M. Tehranipoor, *Hardware Security: A Hands-On Learning Approach*, 1at ed. Morgan Kaufmann Publishers, 2019, 23-45. [Online] Available: ScienceDirect

[2] F. Farooq and A. Nadeem, "A Fault Based Approach to Test Case Prioritization," *2017 International Conference on Frontiers of Information Technology (FIT)*, Islamabad, Pakistan, 2017 [Online]. Available: IEEE Xplore, http://www.ieee.org. [Accessed: Mar. 10, 2025].

[3] R. Kumari, "What is Scripted Testing | How it Differs from Exploratory Testing?”, *testsigma.com*, Apr. 25, 2024. [Online]. Available: https://testsigma.com/blog/scripted-testing/. [Accessed: Feb. 25, 2025].

[4] K. Devaraj, "What is Scripted Testing | How it Differs from Exploratory Testing?”, *testsigma.com*, Apr. 8, 2024. [Online]. Available: https://testsigma.com/blog/exploratory-testing/. [Accessed: Feb. 25, 2025].

[5] M. Goodwin, “What is an API (application programming interface)?”, *ibm.com*, Apr. 9, 2024. [Online]. Available: https://www.ibm.com/think/topics/api. [Accessed: Feb. 25, 2025].

[6] The Postman Team, “What is a REST API? Examples, Uses. And Challenges, *postman.com*, June. 28, 2023. [Online]. Available: https://blog.postman.com/rest-api-examples/. [Accessed: Feb. 25, 2025].

[7] C. Lindholm and K. Nilsson, “databas”, *ne.se*, [Online]. Available: https://www-ne-se.ludwig.lub.lu.se/uppslagsverk/encyklopedi/l%C3%A5ng/databas. [Accessed: Feb. 26, 2025].

[8] “What is MySQL?”, *geeksforgeeks.org*, Apr. 28, 2025 [Online]. Available: https://www.geeksforgeeks.org/what-is-mysql/ [Accessed: Feb. 24, 2025].

[9] The Postman team, “About Postman”, *postman.com*, [Online]. Available: https://www.postman.com/company/about-postman/. [Accessed: Feb. 23, 2025].

[10] The GitLab team, “About GitLab”, *about.gitlab.com*, [Online]. Available: https://about.gitlab.com/company/. [Accessed: Feb. 24, 2025].

[11] “What is JavaScript?”, *geeksforgeeks.org*, Nov. 29, 2024. [Online]. Available: https://www.geeksforgeeks.org/what-is-javascript/ . [Accessed: Feb. 23, 2025].

[12] “About Node.js”, *nodejs.org*, [Online]. Available: https://nodejs.org/en/about [Accessed: Feb. 24, 2025].

[13] Enyichi Agu, “What Exactly is Node.js? A Guide for Beginners”, *freecodecamp.org*, May. 25, 2023. [Online]. Available: https://www.freecodecamp.org/news/what-exactly-is-node-guide-for-beginners/. [Accessed: Feb. 29, 2025].

[14] Martin Heller, “What is Visual Studio Code? Microsoft’s extensible code editor”, *infoworld.com*, Jul. 8, 2022. [Online]. Available: https://www.infoworld.com/article/2335960/what-is-visual-studio-code-microsofts-extensible-code-editor.html. [Accessed: Feb. 29, 2025].

Appendix: Extended material

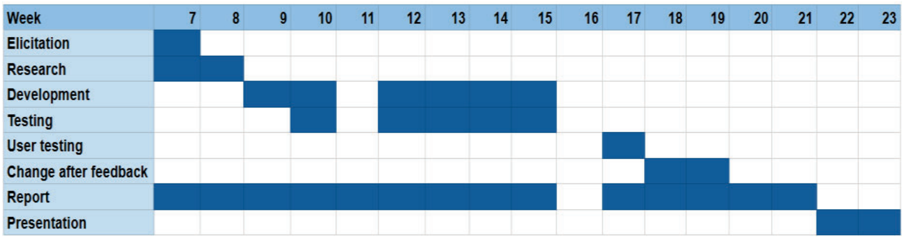

Fig. 2. The figure illustrates the final gantt chart.

```
{
    "tokens": {
        "Authorization": "apiKey YOUR_API_KEY",
        "Cookie": "JSESSIONID=YOUR_JESSION_ID_HERE"
    },
    "release": "XX.XX.XX",
    "date": "YYYY-MM-DD",
    "tags": {
        "tagname1": "YOUR_TAGNAME",
        "tagname2": null,
        "tagname3": null
    },
    "failureRate": X.X,
    "testRunID": YOUR_TESTRUN_ID_HERE
}
```

Fig. 3. The figure illustrates the configuration file before insertion of values

Interview questions and answers:

Do you believe that your team could save time and decrease redundancy by using the createTestRun algorithm?

Test Leader: *As for now, we will not be able to use this algorithm* *since some parameters that are being looked at are not as important for us and other parameters that are not evaluated are important. For example, it is of the utmost importance in our team that tests are passed on certain products/chip, not only the release.*

Do you see any other advantages with createTestRun?

Test Leader: *I can use it as a tool to evaluate which test cases that have failed a lot in a certain phase and use this information men planning the next phase.*

Do you see any disadvantages with createTestRun?

Test Leader: *In my team, everyone is not updating the status to "passed" in Test Management Tool after a problem is fixed which the algorithm relies on.*

How much time do you spend today to create test runs manually?

Test Leader: *There is no clear answer to this, it really depends on the project.*

Do you believe that your team could save time and decrease redundancy by using the updateTestRun algorithm?

Test Leader: *Same answer as for createTestRun. The algorithm will not be applicable for our team if product/chip is missing as a parameter.*

Do you see any other advantages with updateTestRun?

Test Leader: *It will help with recourse handling in the teams where the algorithm is applicable.*

Do you see any disadvantages with updateTestRun?

Test Leader: *As mentioned, we will not be able to use it.*

Do you have any feedback for improvement of the two algorithms?

Test Leader: *If the algorithms handle product/chip and make it possible to compare specific test runs, they would be of potential use for us as well.*

Fig. 4. Interview with a test leader from another QA team.

Do you believe that your team could save time and decrease redundancy by using the createTestRun algorithm?

Test leader 1: *Definitely, at least for a certain tag-combination.*
Test leader 2: *Yes, I agree with Test leader 1.*

Do you see any other advantages with createTestRun?

Test leader 1: *The possibility to evaluate between different test runs. If the same test cases get the same priority again for example.*

Do you see any disadvantages with createTestRun?

Test leader 2: *The way the priorities are chosen are not always obvious. In some cases, tests that have not been executed for an amount of time could be of more importance than their failure rate.*

Do you have any feedback for improvement of createTestRun?

Test leader 1: *It would be great if the text file that is created when running the createTestRun would be saved instead of changed when running the algorithm again. Also add more information to the text file, like the configuration settings used, date, what test run it belongs to.*
Test leader 2: *It would be good to be able to choose what statuses that one wants to add to the test run when the question about adding no priority test cases is asked. Furthermore, it would be an advantage to be able to change what priority the filters should have. It would also be nice if a comment automatically was added to every test case about what filter it passed to get a certain priority. There is*

a comment section in Test management tool.

Do you believe that your team could save time and decrease redundancy by using the updateTestRun algorithm?

Test leader 1: *Yes, this will probably be used more than createTestRun by me.*
Test leader 2: *Yes, but I will have to try it out a couple of times in a test project to make sure that nothing that I need to save in the test run gets deleted.*

Do you see any other advantages with updateTestRun?

Test leader 1: *It will help with prioritization as well.*

Do you see any disadvantages with updateTestRun?

Test leader 1: *There is a risk that you miss something and therefore we need to double check the results in the beginning.*

Test leader 2: *Can probably take some time to learn how to run the algorithm.*

Do you have any feedback for improvement of createTestRun?

Test leader 1: *It would be nice with some sort of log. In the future, an implemented button in Test Management Tool would be handy to make these updates.*
Test leader 2: *I agree with Test leader 1, maybe create text files with deleted test cases.*

Fig. 5. Interview with two test leaders from the collaboration team.

Printed by Tryckeriet i E-huset, Lund 2025

Lund University

Series of Bachelor`s theses
Department of Electrical and Information Technology
LU/LTH-EIT 2025-1065
http://www.eit.lth.se