UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Karl Parelo

Testing a query-based system using metamorphic Testing - a case Study of Optime OÜ

Bachelor's Thesis (9 ECTS)

Supervisor: Alejandra Duque Torres

Tartu 2023

Implementing metamorphic Testing on a query-based system - a case Study of Optime

Abstract:

Automated testing has been the industry standard in software development for a long time. However, some companies still strugle to implement testing for their application or are not satisfied with the results of their current testing framework. This problem is magnified for query-based systems (QBS) with vast amounts of data, where conventional testing methods fall short. Motivated by this challenge, this thesis explores the MT approach as a potential solution to address the testing challenges associated with QBSs. This exploration also offers valuable guidance for companies facing similar testing issues with their respective QBS.

Keywords:

Query systems, Testing, Development

CERCS: P170 Computer science, numerical analysis, systems, control

Metamorfoosse Testimise implementatsioon päringsüsteemil Optime näitel

Lühikokkuvõte:

Automatiseeritud koodi testimine on tarkvaraarenduses juba pikalt olnud standardiseeritud. Siiski leidub firmasid, kes ei ole rahul oma praeguse testimise raamistikuga või ei ole leidnud oma projektile sobivat lahendust. Kui tegemist on päringusüsteemidega, mis tegelevad suurte andmehulkadega, siis on tavapäraste testimismeetoditega lahenduse leidmine veel keerulisem. Sellest probleemist motiveerituna uurib see lõputöö metamorfoosset lähenemist potensiaalse lahendusena päringusüsteemide testimisele. See uuring pakub ka väärtusliku kogemust firmadele, kes seisavad silmitsi sarnaste probleemidega enda päringusüsteemide testimisel.

Võtmesõnad:

Päringu süsteemid, Testimine, Juhend, Tarkvara Arendus

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

Table of Contents

1 Introduction

Software testing is an essential part of the software development process. Testing ensures the code functions as the developer expects it to. The simplest way to accomplish this is by manually running it and observing that it works as desired. Manual testing is, however, repetitive, and time-consuming. It becomes increasingly expensive when the system grows in complexity and size, as modern software systems and projects tend to [1]. The usefulness of testing depends on coverage. The value of tests is small to start off with, as having partial coverage of the codebase requires all the setup of testing infrastructure without any of the benefits. A developer must continuously and extensively test the code to validate its correct functioning and lack of bugs. To tackle this issue, test automation has emerged as a viable solution. Test automation usually consists of a separate program that runs any tests written beforehand. The main benefits of test automation are eliminating repetitive tasks for developers, formalizing the testing pipeline, and creating more elaborate tests than can be accomplished manually.

Despite these benefits, automated testing necessitates that developers precisely know the expected outcomes of their methods for any given input. This involves methodically writing out potential expected outputs for functions when provided with their inputs. This challenge is formally recognized as the Test Oracle problem: the difficulty of determining the correct output of a software system for a specific input [2], [3]. For instance, when testing software applications like compilers, search engines, machine learning algorithms, simulators, and QBSs, it is difficult to verify if the output is accurate for a particular input [2].

Metamorphic Testing (MT) is a software testing approach proposed by Chen et al. [4] to alleviate the test oracle problem. MT examines the relations between input-output pairs of consecutive system under test (SUT) executions rather than the validity of a single input and output. Such relations are known as Metamorphic Relations (MRs), which check how two consecutive outputs should vary in response to specific input changes.

Many companies use QBS, but unfortunately, the maintenance of their functionality is often neglected during the production cycle. This thesis explores the use of the MT approach for a QBS used by a company named Optime. The company employs a QBS to store and manage the data of its customers. This QBS is also used by an application built for the maintenance of the data, as well as for the customers themselves to have control and overview of the data.

This introductory chapter explains the significance of this thesis for the company of Optime, test developers and researchers. It begins by discussing the motivation behind investigating this topic, followed by the value provided by the thesis. Finally, the chapter gives an overview of the subsequent chapters, which detail the implementation process of the MT approach for QBS.

1.1 Motivation

The motivation behind this thesis topic stems from the real-world issues faced by the author while working as a developer at Optime. The QBS queries data that is then served by a front-end application to the client. Neither the QBS or the front-end application test the data being shown autonomously. Because of this, the only way to know the validity of the data shown to the clients is by manually testing the results in the front-end. The QBS is a large part of the value that the application provides for the user, so finding any faults in its operation is vital. A manual testing solution, as outlined in the introduction, is not efficient in the long run and has problems with code coverage. Because of this, the company struggles to maintain the functionality of its product throughout the production cycle.

1.2 Contribution

The thesis explores the MT approach as a potential solution for tackling the testing challenges associated with QBSs. This exploration also provides valuable guidance for other companies encountering similar issues with testing their respective QBS.

The contribution can be broken down as:

- A guide for a developer or company to understand the possibilities and limitations of MT. The background and methodology sections of this thesis introduce current and future best practices of MT. The practical implementation of a testing solution in Optime further shows what benefits MT could bring.
- Offering a practical example of how to apply MT for a QBS application of a company. The approach taken in this thesis and the recommendations for alternative choices can be compared to their own development needs and limitations to find the best fit for their company.

1.2 Outline

The thesis is structured as follows:

Section 2 is background. It serves to introduce and define the key concepts utilized in this thesis. The section covers essential topics such as software testing automation, QBS, and MT.

Section 3 covers related works.

Section 4 is methodology and provides a description of the company, the system under test, and the set of metamorphic relations used.

Section 5 is implementation. It explains everything related to the application developed for the MT of Optime’s QBS. It then covers the results from the MT and future improvements to the application.

Section 6 is the Conclusion to the thesis.

2 Background

This section presents the key concepts utilised in this thesis.

2.1 Software testing automation

Testing is a vital part of the development process but is often seen as a repetitive and time-consuming task. It has been shown that tests should assume 40% to 50% of the development resources to achieve the best results [5]. This shows the usefulness and efficiency gained from having extensive automated testing. The benefits of automated tests are as follows:

- Continuous execution to ensure timely discovery of bugs.
- Reducing repetitive tasks for developers to free up time for other tasks.
- Ensuring consistent output with no human input leads to differences or inconsistent results.

Automated tests are superior to manual tests in all these cases, but manual testing still has its benefits. Manual testing can often find bugs that were not thought of~~,~~ when designing automated tests. Manual testing also does not require the program to function correctly during testing. The human testers can evaluate the behaviour and result of the program during runtime, comparing it with the expected behaviour in their mind. If the application fails in a way not foreseen by the developer, then a badly designed automated test might accept a false result which would obviously be deemed a failure by a manual test.

An automated test needs to be provided with the expected result beforehand. This is named Assertion-based testing and is a common way to write automated tests[6]. Assertion-based tests require the developer to give unambiguous expected results to each test. The shortcomings arising from this approach are collectively called the oracle problem[2]. The oracle problem has been known since the 1980s[7]. E. Weyuker argued in the 1982 paper that there are testable and untestable programmes. For complex functions, the possible range of results might not be comprehendible for the developer, and therefore considered untestable. Today this is exemplified with systems involving artificial intelligence[3].

2.2 Metamorphic Testing

MT is a software testing approach introduced by Chen et al. [4] to address the test oracle problem. MT uses the internal properties of the SUT to validate expected outputs. It achieves this by examining the relations between inputs and outputs from multiple SUT executions, known as MRs. These MRs define how outputs should change when specific variations occur in the input. Figure 1, shows the basic workflow of MT. The basic workflow involves at least five steps:

1. Identify a list of MRs that the SUT should satisfy.
2. Create a set of initial test data (TD).
3. Generate follow-up test data (FTD) by applying selected MR-specified transformations to the inputs.
4. Execute the corresponding SUT with TD and FTD.
5. Verify that the changes in the output observed in the td and FTD match the changes defined by the MR.

The final step requires thorough analysis, as the absence of violations does

not guarantee the correctness of the SUT. If an MR is violated, it indicates a potential fault in the SUT, assuming the MR is accurately defined.

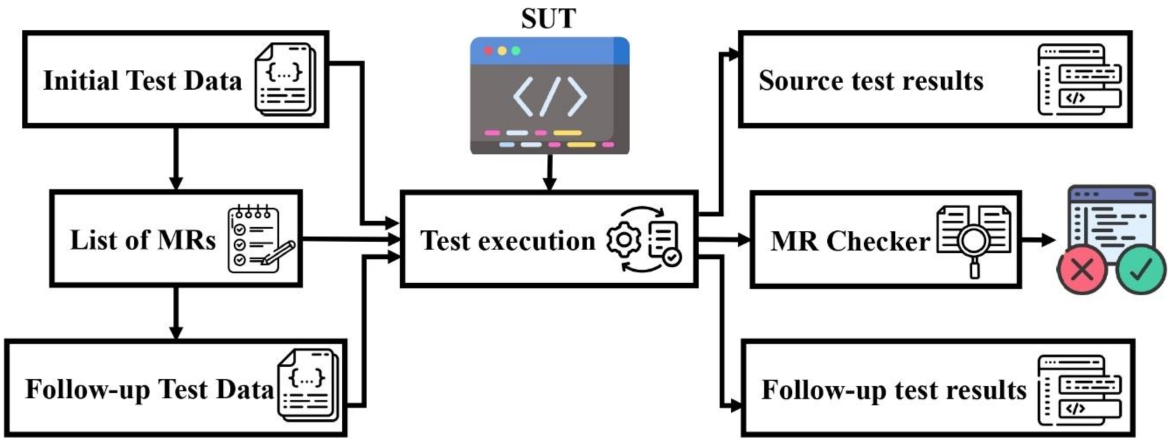


Figure 1. MT basic workflow.

The difficult part in creating the MT approach is deciding on the MRs to use. This is usually done manually, but innovative approaches have been developed to automate this process as well [2], [8]. When an MR is violated by at least one test input-output pair, then there is a strong likelihood that the SUT has a fault. However, the absence of violation does not ensure that the SUT is fault-free. As a result, the suitability of the MRs employed significantly impacts the effectiveness of MT [8].

2.3 Query-based systems

QBS refer to systems that manage searching and retrieving data based on user queries. This is a common requirement for modern applications like information systems, websites, and data analytic platforms [2]. YouTube uses QBS for videos, Facebook for users, and University of Tartu uses it for theses. These platforms use similar operations for manipulating the output of the queries, consisting of filtering, ordering, and paging.

QBS suffer from the test oracle problem to a great extent. Giving the user the freedom to manipulate the querying of a large and changing dataset with multiple operations leads to a large set of valid outputs. Traditional assertion-based automated testing struggles to find the limit between correct responses and nearly correct failures. MT can disregard the size of the data or how the data changes over time, relying on defining MRs that remain constant over time. With the MT approach, having a large dataset only increases the time to run a test, not the complexity of designing it.

3 Related Work

This section provides a brief overview of previous work relevant to this thesis.

Since its introduction in 1998 by Chen et al.[4], MT has been applied to a wide range of applications. There have been workshops held about MT since 2016, which is mostly a forum to share papers on theoretical applications of MT [9]. MT has also been proven to have application in areas previously thought of as untestable by conventional automated testing[10]–[12]. However, there have been few studies on the effectiveness of MT in practical long-term application, where it is required to perform over time and at scale [12].

This thesis is concerned with a QBS, so results focusing on that are particularly relevant. MT has successfully been implemented in several types of QBSs. Search engines are a type of QBS and rely on useful query result for their functioning. Google and Bing both have benefited from MT [10], [13], [14]. RESTful API based applications like Spotify and Youtube eploy similar search functionality as search engines, also benefit from MT [15]. The same concept of search appliesto e-commerce sites such as Amazon and Walmart [16], and data repositories like the NASA's Data Access Toolkit [17]. The solutions proposed for these QBS MT implementations follow a similar workflow. This involves identifying some queries that are particularly vital to the working of the platform and creating MRs that adequetly test their functionality.

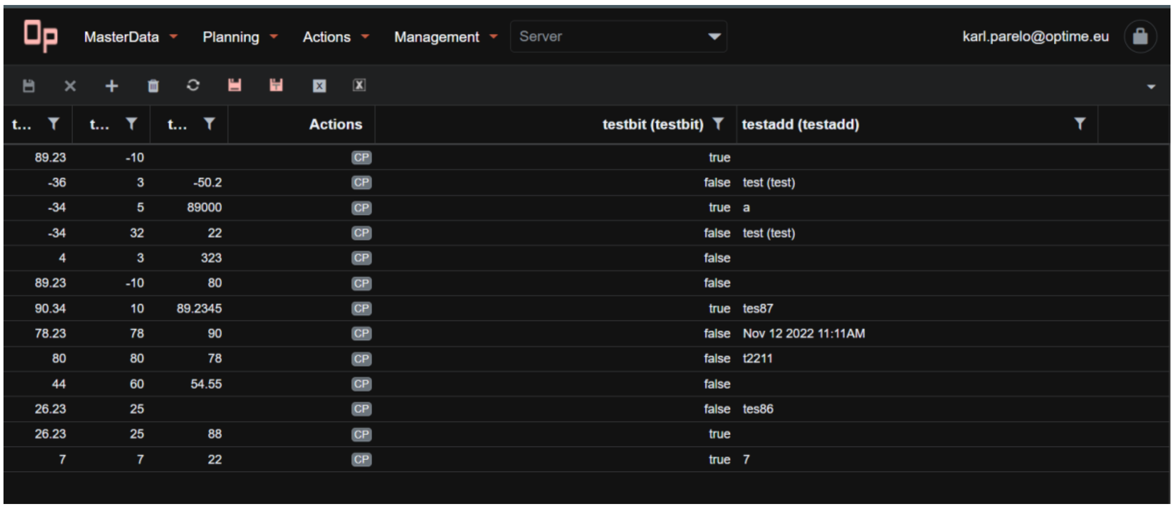
Most of these QBS MT implementations find the suitable MRs manually. There are proposed solutions for automising this process aswell [2]. This thesis creates MRs manually following examples from the cited works on QBS systems.

4 Methodology

4.1 Optime

Optime OÜ is a company that creates custom solutions for business reporting, employing data management and consolidation tools. Optime uses the Microsoft Business intelligence platform to create data warehouses and custom products that offer customers solutions for their business intelligence and accounting problems[18]. Data warehouses are created and managed according to each client's needs dynamically. A data warehouse combines, cleans and arranges data from all the sources that a client is using for storage. A Microsoft Online Analytical Processing (OLAP) data cube is built on top of the data warehouse to speed up querying and serving user requests. Custom Dax solutions and data orchestration transforms the data from the warehouse into a model to be used in Microsoft Power BI reports, Microsoft Excel or the Optime application.

Optime clients have access to the Optime application, which serves data from the OLAP cube or from the data warehouse in an easily understandable form. The application also allows users to create custom MDX scripts or simple operations like editing or deletion to manipulate their data in the web browser. The following picure is an example of viewing a table from the Optime application.



t...	t...	t...	Actions	testbit (testbit)	testadd (testadd)
89.23	-10		CP	true	
-36	3	-50.2	CP	false	test (test)
-34	5	89000	CP	true	a
-34	32	22	CP	false	test (test)
4	3	323	CP	false	
89.23	-10	80	CP	false	
90.34	10	89.2345	CP	true	tes87
78.23	78	90	CP	false	Nov 12 2022 11:11AM
80	80	78	CP	false	t2211
44	60	54.55	CP	false	
26.23	25		CP	false	tes86
26.23	25	88	CP	true	
7	7	22	CP	true	7

Figure 2 [19]. An example of what a client sees when using the Optime application.

4.2 System Under Test

The SUT is the back-end code of the Optime application. The Project is written in C# as an ASP.Net Core application targeting the .NET 6.0 framework. Since the application is query-based, it deals with getting user queries from the front-end application and sending an appropriate request to the data warehouse to fulfil the query. Currently, the data warehouses are hosted in either MS SQL Server locally, in Azure cloud SQL Server or MS Fabric SQL servers. The actual location of the data is not important for MT testing of the back end, as the server can be set as a separate API call beforehand. The steps performed by the SUT upon receiving an API call are as follows:

- Check for Authorization of the credentials used for making the call.
- Authenticate the credentials with Microsoft authentication services.

- Translate the incoming API call from JSON to T-SQL query.
- Send the query to the appropriate data warehouse endpoint.
- Parse the response from T-SQL back to JSON and send a response message to the API call.

The MT in this thesis does not target the first two points in the list. The SUT currently only allow authorized user login and authentication on behalf of an application is not possible. Adding user credentials to the code for silent login is not good practice according to Microsoft identity platform documentation [20]. The application authenticates the user interactively a Microsoft account. This simplifies creating test data for the MT to query, as it can be authorized to the author's account and viewed in the Optime application.

The tests are performed against a post call to the SUT. The call has multiple operations applied to the query that can be used to create MRs. The call also only gets responses from the data warehouse, not from the OLAP cube. This means that the tests only target the backend data management functions and not any data management done in the cube. This ensures that any faults or successes can be contributed to the backend system and ensures encapsulation for testing.

4.3 Metamorphic Relations Formulation

Finding the best set of MRs, as described earlier, require consideration and effort to lead to useful MT. Manual trial and error can be used by changing some possible variables in the input and observing the changes in output. Continuing this process leads to recognizing patterns and formally stating these patterns develops into MRs. This thesis uses the described method in formulating MRs. This was done by observing the calls from the Optime front-end application and constructing MRs to verify the results of the most often sent queries.

The MRs are described using mathematical notation here. The expectation for all the queries is that only the variable in question is being changed, everything else in the query stays the same on subsequent query calls.

MR1 - "Page size"

Increasing the page size should increase the number of results of the query.

Let page size = x, number of results = y

$0 \leq x,y <$ total number of rows in database

MR1:

$$\forall x \; |Query(pagesize = x)| = y \Rightarrow |Query(pagesize = x+1)| = y+1$$

MR2 - “Page“

changing the page should change the values in some range depending on how many values are in the table (eventually last page and increasing that will not change the values)

Let t = total number of rows in the database, s = pagesize
Then the biggest value for page is max = t/s
The page starts at 0 and can only be positive.
The set of all possible values for page is defined as:

$$A = \{x \mid 0 \leq x \leq max\}$$

The set of results for a single query for any page value is defined as:

$$B_x = Query(page=x), x \in A$$

MR2:

$$\forall i \ \forall j \ i, j \in A, i \neq j, x_i \in B_i \Rightarrow x_j \notin B_i$$

MR3 - “Sort”

Changing sort order from ascending to descending or vice versa flips the order of the rows.

Let A = {“Ascending”, ”Descending”}

Let B = set of results for some valid query with order from set A

B = $\{x_i \mid 0 \leq i \leq \text{page size}\}$

MR3:

$$\forall x \ \forall y \ x,y \in A, x \neq y, Query(order = x) = B_m \Rightarrow Query(order = y) = B_n \ \forall m, m \in B_m, n \in B_n \ m_i = n_{pagesize - 1}$$

MR4 – “Filter”

adding a filter to any column will return equal or less values than the original query.

Let x be some filter applied to a column. Can be many data types, not just string

MR4:

$$\forall x \ Query(filter = x) \subseteq Query(no\ filter)$$

5 Implementation

5.1 Application setup

The application to carry out the testing is a C# console application. It is targeting the net 6.0 framework. It includes third-party libraries to accomplish this. A Microsoft SQL client library is used to connect to databases where data is queried and stored, a Microsoft Identity and Active Directory library to authenticate the user and the Newtonsoft library to parse API responses into class objects. The whole application is 274 lines of code.

The application consists of a single class split over 3 different files for clarity and seperation of concerns. There is another file for classes for possible API responses. The main file is Called Program.cs and contains the methods related to creating the queries for each different MR. The two other files have the API and database communication code respectively.

The application's execution logic can be layed out in steps:

- Establish a connection to the SUT running locally on the system.
- Ask for the user to authenticate themselves interactively using the Microsoft authentication platform.
- Change the server to be queried from.
- Get the names tables and respective columns that are being tested.
- Create a list of TCs and FTCs for all tables for each MR.
- Make all the TCs and FTCs in parallel and wait until all have responded.
- Check MR validity for each table or column on each MR.
- Save the responses to a database.

Most of the application runtime is taken waiting for the many API calls to finish. The amount of API calls sent can be calculated as follows:

N = (tables)(columns in each table)*(MRs)*(FTCs*)

This number changes depending on how many FTCs are included, but over 500 calls are sent with the test tables used in the final application version. Since this is the main bottleneck for runtime duration, all the calls are constructed beforehand and executed asynchronously in parallel. This ensures that the backend fills the requests with as little downtime as possible. The diagram in figure 3 illustrates this concept. It shows how only the names of the tables are queried in sequence. All other TCs and FTCs are sent to the SUT in parallel, and responses are collected in parallel aswell.

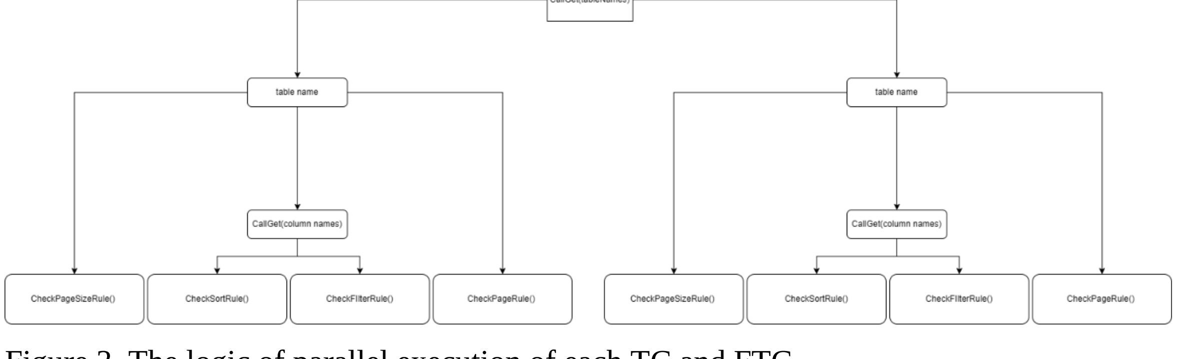
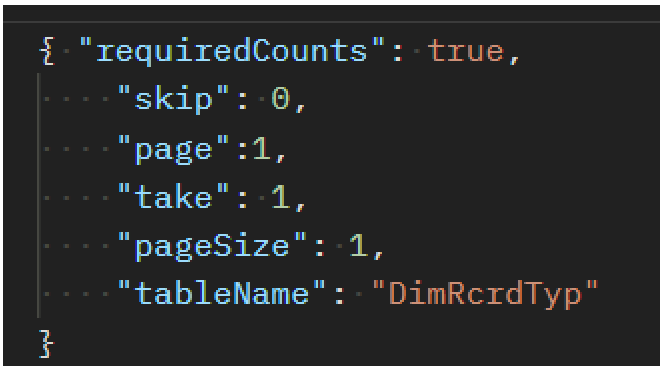


Figure 3. The logic of parallel execution of each TC and FTC.

5.2 Test data

The tests are carried out on data from the “Masterdata” source. Data requested from that endpoint gets served from an Azure hosted Windows machine that locally runs a server with that specific test database. This source has no tables connected through the OLAP cube.

The data is queried with a POST request to the SUT on a localhost https port. The Header includes the API endpoint and a bearer token that is received upon successful authentication from online Microsoft services. Each MR has a slightly different request body to accommodate for the different requirements. As seen in figure 4, the “Page size” MR takes very simple variables for its query.



```
{ "requiredCounts": true,
    "skip": 0,
    "page":1,
    "take": 1,
    "pageSize": 1,
    "tableName": "DimRcrdTyp"
}
```

Figure 4. The picture shows the body of a POST call for the TC of the “Page size” MR.

Once an MR gets responses for all the calls it requires, the MR validity check step begins. The SUT response is analysed based on each MRs criterion. This is accomplished with C# native functions. The results are saved to an Azure SQL Server hosted by Optime to store information about the Optime application statistics.

The application reads and processes data from the tables but does not save the values of any cell. This ensures that if any table were to contain sensitive data to the company, then it would not be saved to anywhere. The tables used for the testing include test tables created and populated to be different from each other and cover as many database states as possible. This includes tables with varying amounts of rows, columns covering every data type and cells populated possible value states such as null.

5.3 Results

The results are saved to a single table. Each row in the table represents an MR test. The columns store the table name, result, MR, and details of the test. Figure 5 shows a part of the results. As seen in Figure 5, no actual data is being saved other than table and column names. Even this is done just for clarity, and can be discarded, if necessary, after calculating if a MR is failed or passed for any one column or table name.

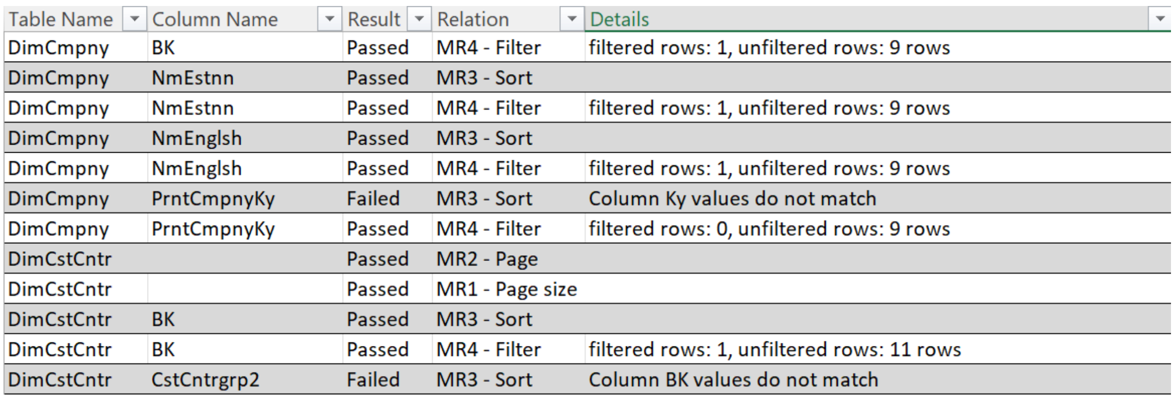


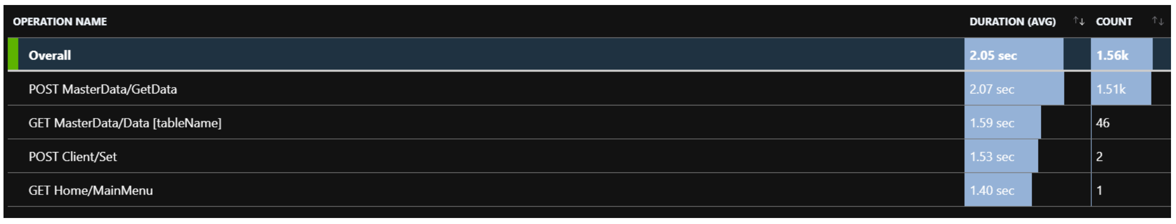
Table Name	Column Name	Result	Relation	Details
DimCmpny	BK	Passed	MR4 - Filter	filtered rows: 1, unfiltered rows: 9 rows
DimCmpny	NmEstnn	Passed	MR3 - Sort	
DimCmpny	NmEstnn	Passed	MR4 - Filter	filtered rows: 1, unfiltered rows: 9 rows
DimCmpny	NmEnglsh	Passed	MR3 - Sort	
DimCmpny	NmEnglsh	Passed	MR4 - Filter	filtered rows: 1, unfiltered rows: 9 rows
DimCmpny	PrntCmpnyKy	Failed	MR3 - Sort	Column Ky values do not match
DimCmpny	PrntCmpnyKy	Passed	MR4 - Filter	filtered rows: 0, unfiltered rows: 9 rows
DimCstCntr		Passed	MR2 - Page	
DimCstCntr		Passed	MR1 - Page size	
DimCstCntr	BK	Passed	MR3 - Sort	
DimCstCntr	BK	Passed	MR4 - Filter	filtered rows: 1, unfiltered rows: 11 rows
DimCstCntr	CstCntrgrp2	Failed	MR3 - Sort	Column BK values do not match

Figure 5. The output of the application with each row representing a MR result.

The amount of follow up tests per MR in the last version of the application was chosen based on the runtime of the application. More API calls could be made for each MR, but the results would not change unless an edge case is tested, which can be reached without unnecessary calls in between. Additional faults could be introduced with too many calls, transforming the MT into a load test on the SUT. This, however, is not the goal of this thesis.

A single run of the application outputs 192 rows of results. Each row represents the result of a MR. This takes 37 seconds, and the application makes over 500 calls against the SUT, with an average response time of two seconds, seen in figure 5. Two seconds is quite long for a single request, but only half of the requests lasted more than one second. The longest requests took 10.5 seconds, as seen in figure 6. This can be explained by some tables having more rows of data and the “Sort” and “Filter” MRs take more time to respond to those. The MT application is built to handle requests in parallel, so long response times were minimised with this implementation.

These figures were averaged over three runs for consistency, but since the application is using the same data each time, the runtime statistics are not different enough to warrant a bigger sample size of runs. This data was gathered from Azure Application Insights, which monitors the SUT running locally.



OPERATION NAME	DURATION (AVG)	COUNT
Overall	**2.05 sec**	**1.56k**
POST MasterData/GetData	2.07 sec	1.51k
GET MasterData/Data [tableName]	1.59 sec	46
POST Client/Set	1.53 sec	2
GET Home/MainMenu	1.40 sec	1

Figure 5. The average duration of requests with three runs worth of calls.

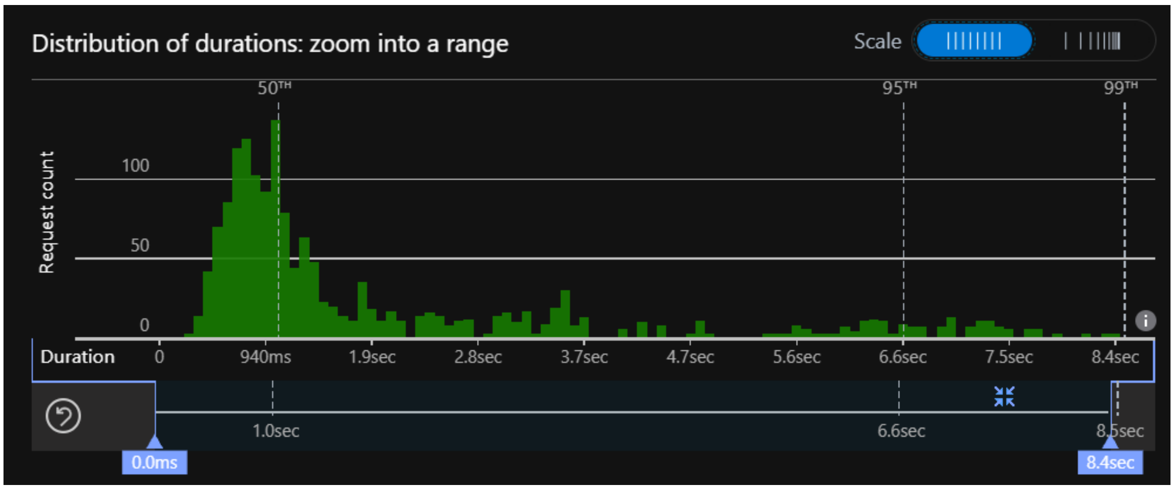


Figure 6, The distribution of request response durations for three runs worth of calls.

Out of the four MRs implemented for the SUT, only one resulted in a bug being discovered in the QBS. The “Sort” MR failed on some columns in some tables. This is shown in the statistical breakdown of the failures and passes by each MR in figure 7. The table in figure 7 also shows how the “Sort” and “Filter” MRs have many more tests than the “Page size” and “Page” MRs. This is from the former two being tested on each column of each table, rather than just the whole table. This was a previously unknown bug. After discovery by the MT application, it was also sown to exist in the front-end Optime application. The bug has been brought up in the company and has confirmed to require changes in the back-end application. The bug happens when data is sorted by some column that has null values in at least two rows in that column. These null values to not appear in the same order when sorting rows by that column in ascending vs descending order. The cause of the bug is not known yet, but probably has to do with the sorting logic not handling null cases correctly.

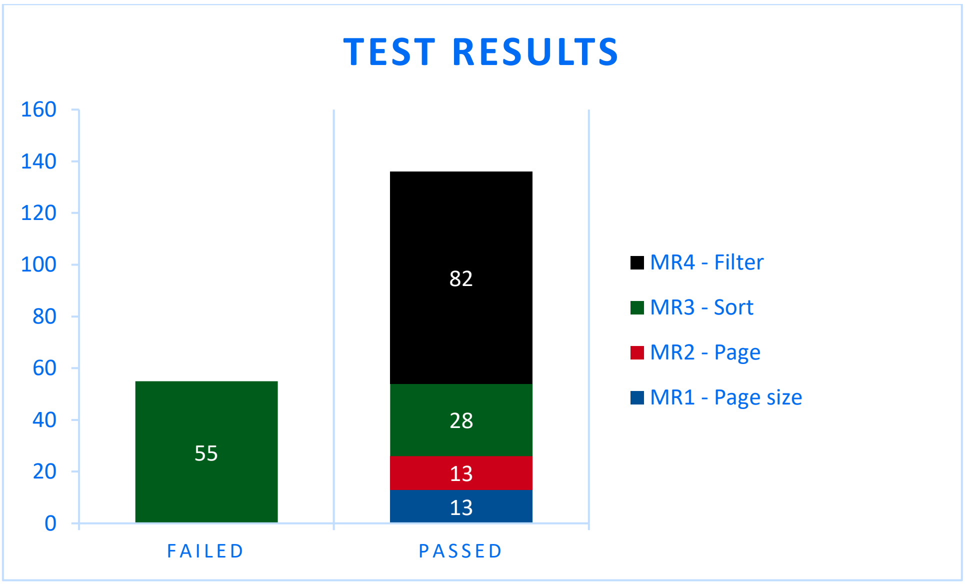


Figure 7. Breakdown of tests by MR type and MR pass or failure.

5.4 Future Improvements

The application can be improved and expanded in many ways. The query used in this thesis has many more variables that can be sent to the SUT that would change the output. The scope of only four MRs was chosen, because these are the changes that can be performed from the front-end Optime application, so they are used by clients daily and are the highest priority. The chosen MRs can be expanded upon with follow-up tests and more tests per MR. New MRs for other queries can also be added. The application serves different data through the OLAP cube, which could be tested in the same way. This would make the run time longer, but a further improvement would be automating the execution of the application. Automation would render the runtime constraint less important, as it can run in the background for as long as necessary. This requires changing the back-ends authentication systems to allow for authentication without an interacting user.

With the current implementation of MT, the strength of the MRs to identify correct functionality of the SUT is not known. Implementing mutation analysis would confirm the MRs usefulness. This would require changing the source code of the SUT and seeing if the MR results change to reflect any behaviour change.

6 Conclusion

The goal of this thesis was to explore the MT approach as a potential solution for tackling testing challenges associated with QBS applications based on an application developed for Optime. The developed application used four different MRs to find any faults in the SUT. The developed application found a fault using one of the MRs and the company has taken steps in fixing it. The development process has made it clear that implementing a MT approach is viable and can provide real benefits even for a company using a QBS. Developing further MRs and expanding the testing to other parts of the QBS can be planned for and time costs can be estimated based on the work done so far.

Throughout the process of creating the MT implementation for Optime, the author had difficulty finding the proper scope for the MT to provide enough benefit for its development cost. This should be kept in mind when implementing the MT method of testing for a query system in the future. The right balance of MRs per API call and FTCs per MR must be found. Having too many FTCs leads to a long runtime with most FTCs serving no purpose, but having too few might leave out a crucial edge case. The same applies to the number of MRs per API call. This Thesis only covered one API call, but with enough depth to find a fault in the SUT. This underscores the importance of choosing the most vital calls to implement first and analysing the queries to focus on the most promising MRs.

7 References

[1] M. Shahid and S. Ibrahim, “A Study on Test Coverage in Software Testing Systematic mapping study on Intrusion Detection View project Software Traceability View project.” [Online]. Available: https://www.researchgate.net/publication/228913406

[2] S. Segura, A. Durán, J. Troya, and A. Ruiz-Cortés, “Metamorphic Relation Patterns for Query-Based Systems.”

[3] S. Segura, D. Towey, Z. Q. Zhou, and T. Y. Chen, “Metamorphic Testing: Testing the Untestable,” *IEEE Softw*, vol. 37, no. 3, pp. 46–53, May 2020, doi: 10.1109/MS.2018.2875968.

[4] T. Y. Chen, S. C. Cheung, and S. M. Yiu, “Metamorphic Testing: A New Approach for Generating Next Test Cases,” 1998.

[5] D. Kumar and K. K. Mishra, “The Impacts of Test Automation on Software’s Cost, Quality and Time to Market,” in *Procedia Computer Science*, Elsevier B.V., 2016, pp. 8–15. doi: 10.1016/j.procs.2016.03.003.

[6] B. A. Malloy and J. M. Voas, “Programming with Assertions: A Prospectus.” [Online]. Available: http://www.icsi.berkeley.

[7] E. J. Weyuker, “On Testing Non-testable Programs.” [Online]. Available: https://academic.oup.com/comjnl/article/25/4/465/366384

[8] A. Duque-Torres, D. Pfahl, C. Klammer, and S. Fischer, “Bug or not Bug? Analysing the Reasons Behind Metamorphic Relation Violations,” May 2023, doi: 10.1109/SANER56733.2023.00109.

[9] MET 23, “Metamoprhic Testing workshop Series,” *http://metwiki.net/MET23/series.html*, 2023.

[10] Z. Q. Zhou, S. Xiang, and T. Y. Chen, “Metamorphic Testing for Software Quality Assessment: A Study of Search Engines,” *IEEE Transactions on Software Engineering*, vol. 42, no. 3, pp. 260–280, Mar. 2016, doi: 10.1109/TSE.2015.2478001.

[11] T. Chen *et al.*, “Metamorphic Testing: A Review of Challenges and Opportunities,” *ACM Comput Surv*, vol. 51, pp. 4:1-4:27, Jan. 2018, doi: 10.1145/3143561.

[12] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortes, “A Survey on Metamorphic Testing,” *IEEE Transactions on Software Engineering*, vol. 42, no. 9, pp. 805–824, Sep. 2016, doi: 10.1109/TSE.2016.2532875.

[13] Z. Quan Zhou, T. H. Tse, F. Kuo, and T. Y. Chen, “Automated Functional Testing of Web Search Engines in the Absence of an Oracle *.” [Online]. Available: www.google.com

[14] Z. Q. Zhou, S. Zhang, M. Hagenbuchne, T. H. Tse, F.-C. Kuo, and T. Y. Chen, “Automated functional testing of online search services,” *Software Testing, Verification and Reliability, vol. 22, no. 4,* pp. 221–243, 2012.

[15] S. Segura, J. Parejo, J. Troya, and A. Ruiz-Cortés, “Metamorphic Testing of RESTful Web APIs,” *IEEE Transactions on Software Engineering*, vol. PP, p. 1, Oct. 2017, doi: 10.1109/TSE.2017.2764464.

[16] Z. Q. Zhou, L. Sun, T. Y. Chen, and D. Towey, “Metamorphic relations for enhancing system understanding and use,” *IEEE Transactions on Software Engineering*. 2018.

[17] C. Schulze, M. Lindvall, S. Bjorgvinsson, and R. Wiegand, “Model Generation to support Model-based Testing Applied on NASA DAT-an Experience Report,” 2015.

[18] Optime, “optime/products/data-warehouse-master-data-reporting/,” Aug. 08, 2023. https://www.optime.eu/en/products/data-warehouse-master-data-reporting/ (accessed Aug. 08, 2023).

[19] Optime, “Optime MasterData Example.” Aug. 08, 2023.

[20] Dickson Mwendia, John Flores, Pritam Ovhal, Marsh Macy, and Sahil Malik, “Desktop app that calls web APIs: Acquire a token using Username and Password,” Oct. 20, 2022. https://learn.microsoft.com/en-us/azure/active-directory/develop/scenario-desktop-acquire-token-username-password?tabs=dotnet (accessed Aug. 08, 2023).

8 License

Non-exclusive licence to reproduce the thesis and make the thesis public

I, Karl Parelo,

(*author's name*)

1. grant the University of Tartu a free permit (non-exclusive licence) to

reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, my thesis

Testing a query-based system using metamorphic Testing - a case Study of Optime OÜ,

(*title of thesis*)

supervised by Alejandra Duque Torres.

(*supervisor's name*)

2. I grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in points 1 and 2.

4. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Karl Parelo

16/08/2023