IMPLEMENTATION OF MODEL BASED TESTING FOR TESTING KAWN SUBSCRIPTIONS MANAGER APPLICATION

Arnaldo Marulitua Sinaga[1], Mikhael Hutapea[2]*

[1,2]Program Studi Sarjana Terapan Teknologi Rekayasa Perangkat Lunak, Institut Teknologi Del, Jl. Sisingamangaraja Laguboti
[1]Email: aldo@del.ac.id
[2]Email*: mhutapea751@gmail.com

ABSTRAK

Penelitian ini berfokus pada aplikasi Kawn Subscriptions Manager, yang dikembangkan untuk membantu pelaku bisnis *Food and Beverage* (F&B) mengelola langganan pelanggan. Saat ini, aplikasi Kawn Subscriptions Manager belum memiliki rangkaian pengujian standar yang telah divalidasi. Untuk itu pada penelitian ini, *Model-Based Testing* (MBT) diterapkan dalam pengujian aplikasi Kawn Subscriptions Manager sebagai metode pengujian untuk menghasilkan rangkaian pengujian yang spesifik dan teknis. MBT adalah metode pengujian berdasarkan *behavioural model* dari sistem yang sedang diuji. Kemudian untuk mengukur *adequacy* kasus uji yang dihasilkan oleh MBT digunakan *mutation testing*. Model dibuat menggunakan *tool* GraphWalker dan pengujian dilakukan secara otomatis dengan menggunakan Selenium dan testNG pada *text editor* Eclipse IDE. Dari hasil penelitian diketahui bahwa pengujian MBT menghasilkan 73 kasus uji dari model perilaku fungsi yang diuji. Pengujian dengan MBT menunjukkan bahwa tidak ditemukan kegagalan pada fungsi yang diuji. Kemudian diperoleh rata-rata *mutation score* sebesar 72,18% yang menunjukkan bahwa *test case* yang dihasilkan oleh MBT cukup *adequate* atau berada pada kategori medium. Namun, penting untuk dicatat bahwa meskipun MBT berhasil dalam menghasilkan kasus uji yang memeriksa perilaku aplikasi, ditemukan bahwa metode ini ternyata tidak mampu mendeteksi kegagalan yang disebabkan oleh mutan yang tidak mempengaruhi perilaku aplikasi secara signifikan. Penemuan ini menunjukkan bahwa MBT dapat menjadi alat yang efektif untuk menguji fungsi-fungsi utama, tetapi tetap dibutuhkan metode tambahan untuk mengidentifikasi kegagalan yang mungkin terjadi akibat perubahan kecil atau tidak signifikan pada kode aplikasi.
Kata kunci: Model Based Testing, Mutation Testing, GraphWalker, Behavioural Model, Adequacy.

ABSTRACT

This research focuses on the Kawn Subscriptions Manager app, which was developed to help Food and Beverage (F&B) businesses manage customer subscriptions. Currently, the Kawn Subscriptions Manager application does not have a standardized test suite that has been validated. Therefore, in this research, Model-Based Testing (MBT) is applied in testing the Kawn Subscriptions Manager application as a testing method to produce a specific and technical test suite. MBT is a testing method based on the behavioral model of the system being tested. Then, to measure the adequacy of test cases generated by MBT, mutation testing is used. The model is created using the GraphWalker tool and testing is done automatically using Selenium and testNG in the Eclipse IDE text editor. From the results of the study, it is known that MBT testing generates test cases from the behavior model of the function being tested. Testing with MBT shows that no failures were found in the tested functions. Then the average mutation score is 72.18% which indicates that the test cases generated by MBT are adequate or in the medium category. However, it is important to note that while MBT was successful in generating test cases that examined the application's behavior, it was found that the method was unable to detect failures caused by mutants that did not significantly affect the application's behavior. This finding suggests that while MBT can be an effective tool for testing key functions, additional methods are needed to identify failures that may result from minor or insignificant changes to the application code.
Keywords: Model Based Testing, Mutation Testing, GraphWalker, Behavioural Model, Adequacy.

1. INTRODUCTION

Model based testing (MBT) is an automation of black box test design, where test cases will be generated automatically from the behavioural model of the system under test (SUT) so that it will speed up the testing process [1]. The main advantage of using MBT is that the time required to model system behaviour is faster than writing and executing test cases manually. MBT also has a wider range of test cases

***Corresponding Author**
Submitted : 23 Agustus 2023
Accepted : 14 September 2023
Online Published : 31 Oktopber 2023
ISSN: 2337-7631 (Printed)
ISSN: 2654-4091 (Online)

that are not possible to create manually [2]. MBT is also more time-efficient, so it costs less when compared to test case-based testing [3].

With several advantages that MBT has, this method will be applied to the testing of Kawn Subscriptions Manager application. Kawn Subscriptions Manager is a web-based system developed to manage the subscription packages of F&B businesses at Kawn. Previously, the Kawn Subscriptions Manager application did not have a validated standard test suite. For this reason, a method that produces a good standard test suite and is validated based on the adequacy of the test items is needed. The model generated by MBT will help understand the system under test (SUT) more easily and can be reused so that when the system has new features, the tester can gradually add these features to the model [4]. This will help test the Kawn Subscriptions Manager application when there are additional changes or features occur in the application.

Then the fault-based testing method with a mutation testing approach is used to measure the adequacy level of the test suite produced by MBT. The adequacy criterion is a measure that assesses the ability of the test suite to detect errors in the SUT [5]. The use of this mutation testing approach aims to evaluate the ability of the test suite generated by MBT to detect faults contained in the Kawn Subscriptions Manager. The purpose of this research is to create a series of automatic tests made from models with MBT techniques in the Kawn Subscription Manager application and calculate the adequacy level of the test suite generated by MBT using the mutation testing method.

The second part of this article contains a review of various literature that has been collected and the methods used by researcher to conduct the research. The third section describes the implementation and experiments conducted and also a discussion of the research results. The fourth section contains conclusions about the research that has been done and suggestions needed for improvement in future research.

2. MATERIAL AND METHODS

In this section, a summary of previous research that has been done and related to MBT, GraphWalker, mutation testing, and Kawn Subscriptions Manager is explained. Also in this section, we will explain the methodology that will be used in this research along with the research design that will be carried out.

Model Based Testing

Model-based testing is a testing method that relies on a model of the system being tested and/or its environment to obtain test cases where the model describes the behaviour of the system and also the possible behaviour of the system's environment [6]. The model of a system is obtained from the requirements and behaviour of the system under test (SUT) [2]. Models of a SUT can be in the form of UML diagrams and finite state machines [7].

GraphWalker

GraphWalker is an open source MBT tool for generating models in the form of graphs [8]. The graph in GraphWalker is a finite state machine and from the resulting graph, test cases will be generated automatically [9]. The resulting graph is a model that represents the behaviour of the SUT.

Mutation Testing

Mutation testing is a testing technique that modifies the program by inserting faults into the program to create new versions of the program called mutants [10]. The original program modification process is done by changing the syntax in the program with the mutation operator. Mutation operators used in mutation testing can be customized according to the programming language of the system to be tested [11]. Kawn Subscriptions Manager is an application built using the Python programming language, so the types of mutation operators used are mutation operators for Python programs. Some mutation operators that can be implemented in Python programs include [12]:

1. AOD - Arithmetic Operator Deletion
2. AOR - Arithmetic Operator Replacement
3. ASR - Assignment Operator Replacement
4. COD - Conditional Operator Deletion
5. COI - Conditional Operator Insertion
6. CRP - Constant Replacement
7. ROR - Relational Operator Replacement
8. IOD - Overriding Method Deletion
9. IOP - Overridden Method Calling Position Change
10. SCD - Super Calling Deletion

11. SCI - Super Calling Insertion
12. DDL - Decorator Deletion

To measure the level of adequacy in mutation testing, we can use mutation score. Mutation score is a comparison between kill mutant and non-equivalent mutant [13]. Mutation score can be calculated using the Equation 1.

$$Mutation\ Score = 100 * \frac{D}{(N-E)} \tag{1}$$

With D stands for death mutant, N stands for total mutant, and E stands for equivalent mutant. Also, to calculate the average mutation score from all tested functions can be done using Equation 2.

$$Average\ Mutation\ Score = \frac{Total\ Mutation\ Score}{Number\ of\ Mutants} \tag{2}$$

Kawn Subscriptions Manager

Kawn Subscriptions Manager is a system developed to manage the subscription status of F&B businesses in Kawn. Kawn itself is an online cashier application designed to be a companion for F&B businesses in inventory, pricing, transactions, and business bookkeeping activities. Kawn Subscriptions Manager helps manage the subscription process of F&B businesses that subscribe to the Kawn application. Figure 1 shows the login page of the Kawn Subscriptions Manager application.

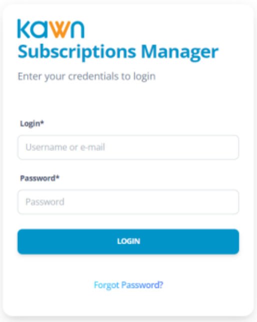

Figure 1. Kawn Subscriptions Manager

The modules in Kawn Subscriptions Manager include:

1. Client: Client module consists of clients and outlets. In the client mode, there is a client update feature and in outlet mode, there are add, update, and delete outlet features.
2. User: In user module there are add, update, delete users, email confirmation, profile management, and password reset features.
3. Subscription: The subscription module consists of two parts namely subscription plan and subscription. In subscription plan there are add, edit, delete, activate, and deactivate features, while in subscription, there are add, deactivate, and activate features for extending the subscription period.

In this research, the functions that will be tested are the authentication and the functions of adding data. The data addition function itself consists of the function of adding user, adding outlet, adding subscription plan, and adding subscription. The reason for choosing these functions is that all five functions have fields that can be filled in by the user and each field has a number of input requirements. In addition, these functions are easier to modify because the program code contains operands, constants, methods, and statements.

The business process of the authentication function of the Kawn Subscriptions Manager application that will be tested can be seen in Figure 2.

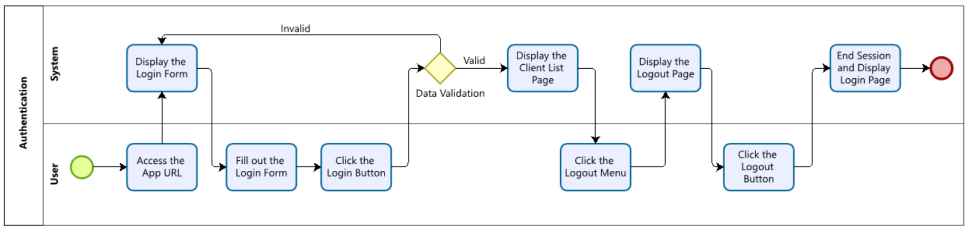

Figure 2. Authentication Business Process

The authentication business process starts with the user entering the application Uniform Resource Locator (URL) into the browser. Then the system will display a login page containing a form that can be filled in by the user. If the user fills in the form with the correct input and is registered in the system, then the user will be directed to the client page. However, if the user fills out the form with the wrong input or is not in the system or does not fill out the form, the user will be directed back to the login page. Then users who want to exit the application can select the logout menu. Then the system will display the logout page and the user can press the logout button. The system will stop all user access to the application and the user will be directed back to the login page.

As for the business process of the create data function which consists of creating users, outlets, subscription plans and subscriptions can be seen in Figure 3.

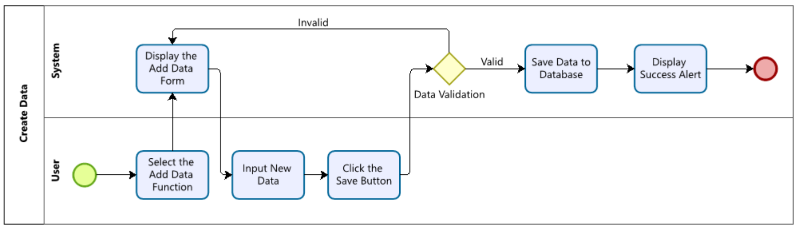

Figure 3. Create Data Business Process

The create user, outlet, subscription plan and subscription functions have the same business process flow, so they are described in one create data business process. The create data business process starts when the user selects the add data menu. Then the system will display the add data form page that can be filled in by the user. After the user fills in the form and presses the save button, the system will check the input data. If the data filled in is in accordance with the specified format, the data will be stored in the database. Then the system will display a success alert on the data list page. However, if the data filled in by the user does not match the specified format, the user will be directed back to the add data form.

Research Methodology

The methodology used in conducting this research is experimental methodology. Experimental methodology is a method that involves a deep understanding of how to carefully design and run experiments, take into account various variables that might affect the results, and ensure that the data obtained is reliable [14]. In this study, the experimental methodology was carried out in several stages. The research design can be seen in Figure 4.

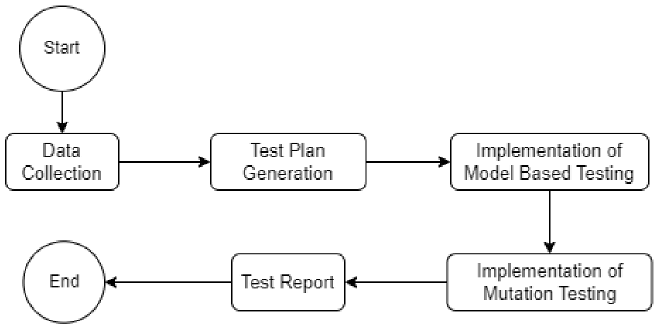
Start
Data Collection
Test Plan Generation
Implementation of Model Based Testing
Implementation of Mutation Testing
Test Report
End

Figure 4. Research Design

The first stage of this research is data collection, which is a crucial step in understanding the Kawn Subscriptions Manager app thoroughly. Data collection aims to obtain relevant and in-depth information related to the application. The data collection process is done through various methods that involve extracting literature sources and documentation related to the application being tested. This includes searching, understanding, and analyzing various documents, guides, technical specifications, and other resources related to Kawn Subscriptions Manager. The results obtained from data collection will be used for the behavioural model design stage of the Kawn Subscriptions Manager.

The second stage is to develop a test plan. The test plan will clearly reveal the test objectives, strategies and approaches, test procedures, test environment, test completion criteria, components to be tested and personnel. The third stage is done by applying model-based testing (MBT) to test the application. The stages carried out in MBT include understanding the system under test (SUT), test selection criteria, test case generation and test execution. The fourth stage is the application of mutation testing to measure the adequacy level of the test cases generated by the MBT method. The stages in mutation testing are generating mutant programs and mutation testing. The last stage is writing a test report. The test report contains testing activities that have been carried out, test results, and conclusions.

An explanation of the implementation of MBT and MT in this study will be explained in this section. The details of the testing stages with model-based testing can be seen in Figure 5.

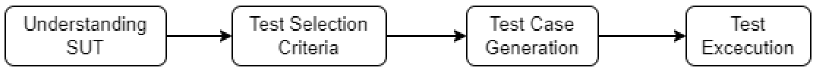

Figure 5. Stages of Model-Based Testing

The first stage in MBT is understanding the system under test (SUT). This stage is done by analyzing the Kawn Subscriptions Manager application. The analysis is done by referring to the Kawn Subscriptions Manager technical document. The expected results at the analysis stage are an overview of the application, an overview of business processes, specifications and identification of functions. Then the behavioural model design is based on the understanding and analysis of the SUT.

The second stage is testing selection criteria. At this stage, a behavioural model is made with the GraphWalker MBT tool. Behavioural models are made in finite state machine format. Behavioural models are made based on the flow of business processes of each function. The third stage is done by generating test cases. Test cases are generated automatically according to the behavioural model that has been made in the previous stage. This process is automated by creating an interface model and applying it to a test script in the Eclipse IDE. The fourth stage is the stage carried out by executing the test case on the test object. The test execution process is done automatically with Eclipse IDE. As for the details of the testing stages with mutation testing can be seen in Figure 6.

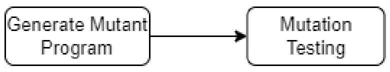

Figure 6. Stages of Mutation Testing

The first stage in mutation testing is generating mutant programs. At this stage, the process of creating a mutant program is carried out. Mutant programs are generated by modifying the function program code that has been tested in the previous stage. The program code is modified using the mutation operator. Mutant programs generated by each function will differ according to the number of mutation operators that can be applied to the program code. Mutation operators that will be applied to the original program modification process are mutation operators for python language programs. The program code modification will be done manually.

The second stage is mutation testing. This stage is done by testing test cases against the mutant program that has been created. The test cases tested are the same test cases used to test the original program. At this stage, the mutation score calculation process is carried out to determine the level of test suite adequacy produced by model-based testing.

3. RESULTS AND DISCUSSION

This section contains an explanation of the implementation process of applying MBT to Kawn Subscriptions Manager application testing and experiments with mutation testing.

Test Object Behavioural Model

Before the behavioural model is created, the analysis and design of the function is first carried out. For example, in the authentication function, from the analysis, it is found that the authentication function consists of login and logout functions. In the login function, there are two inputs, namely username/email and password. Users who want to access the system must go through the authentication process. If the user provides correct input on the login form, the user will be directed to the client page, but if the user does not provide input or is wrong, the user will remain on the login page. For the logout process, the user will exit the system to the login page. Based on this analysis, the design of the behavioural model of the authentication function can be seen in Table 1.

Table 1. Authentication Behavioural Model Design

Transitions	***From State***	***To State***
OpenLoginPage	-	LoginPage
InputCorrectCredentials	LoginPage	ClientPage

Transitions	*From State*	*To State*
InputIncorrectCredentials	LoginPage	LoginPage
InputEmptyCredentials	LoginPage	LoginPage
OpenLogoutPage	ClientPage	LogoutPage
Logout	LogoutPage	LoginPage

Based on the results of the analysis of the behaviour of the test object through the business process, a behavioural model of the test object is generated. The process of creating a behavioural model of the object to be tested is done using GraphWalker studio. The model is stored in json form and used to generate test cases. In this research, five behavioural models were produced according to the number of functions tested. Figure 7 is a behavioural model for the authentication function.

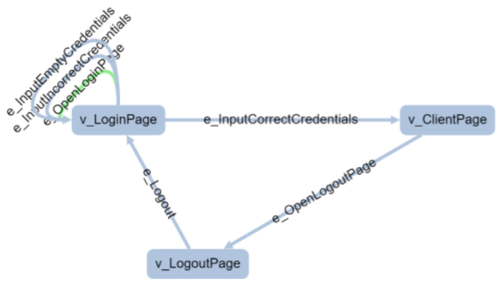

Figure 7. Authentication Behavioural Model

From the behavioural model of the authentication function above, there are three vertices and six edges. Details of the vertices and edges of the behavioural model of the authentication function can be seen in Table 2.

Table 2. Authentication Behavioral Model

Edge/Vertex	*Description*
v_LoginPage	A state indicating that the login page has been successfully opened and the SUT displays the login page.
v_ClientPage	A state indicating that the user has successfully logged into the application and the application displays the client page.
v_LogoutPage	A state indicating that the user is on the logout page.
e_OpenLoginPage	Action to start the browser and display the login page. This element is the start element.
e_InputCorrectCredentials	Action to enter a value into the login form where the value entered is a valid or correct value.
e_OpenLogoutPage	Action to press the logout button.
e_Logout	An action that indicates the process of exiting the application.
e_InputIncorrectCredentials	Action to enter a value into the login form where the value entered is an invalid or wrong value.
e_InputEmptyCredentials	Action to enter a blank value into the login form.
v_LoginPage	A state indicating that the login page has been successfully opened and the SUT displays the login page.

In testing the Kawn Subscriptions Manager application, five behavioural models were generated according to the number of functions tested. In the behavioural model for the create user, outlet, subscription plan and subscription functions, there are seven vertices and nine edges each.

Model Interface Generation

In this stage, the model interface generation is done from the behavioural model that has been created. The generated interface provides a method that can be called by the class where the test script is created to retrieve information about the model and how the model should run. The interface model generated is five interfaces according to the number of models created. Each edge and vertex in the behavioural model will become a method in the interface and can be called in the class that implements the interface.

Testing Implementation

Before the test is executed, a test script is first created by implementing the method on the interface. Testing is done by executing all the methods in the test script. Testing is said to be successful if all methods in the test script are executed at least once. This means that all edges and vertices in the model must be passed. In other words, all functions in the software have been tested and there are no errors or weaknesses in the developed system. After testing the original program, it will be continued with the implementation of mutation testing.

Mutant Program Creation

Mutant programs are generated using the mutation operator. The mutation operator for each function can be based on the program code complexity of each function. This will make the type of mutant program and the number of mutant programs for each function different. The type of mutation operator used in each function can be seen in Table 3.

Table 3. Authentication Mutant Program

Function	***Description***	***Mutation Operator***
Authentication	In the authentication function code, there are conditional operator, and relational operator and a constant.	COD, ROR, CRP.
Create User	The create user code contains a decorator, assignment operator, overriding method, and constants.	ASR, DDL, IOD, CRP.
Create Outlet	In the code of the create outlet function, there are three operators, namely arithmetic, conditional and relational and one constant. Then there are method calls (overriding method) and super method.	ROR, IOD, SCD, IOP, CRP, COD, AOD.
Create Subscription Plan	The create subscription plan code contains a decorator, arithmetic operator, relational operator, overriding and super method.	COD, ROR, DDL, AOR, IOD, SCD.
Create Subscription	In the create subscription code, there are relational operators, arithmetic operators, conditional operators, decorators, overriding methods, and super methods.	IOD, COD, ROR, DDL, SCD, IOP, AOR, COD.

Based on the type of mutation operator in Table 3, the mutant program for each function tested can be obtained. For example, the mutant program for the authentication function can be seen in Table 4.

Table 4. Authentication Mutant Program

Original Program	***Mutation Operator***	***Mutant Program***
`{% if redirect_field_value %}` `<input type="hidden" name="{{ redirect_field_name }}" value="{{ redirect_field_value }}" />` `{% endif %}`	COD	`<input type="hidden" name="{{ redirect_field_name }}" value="{{ redirect_field_value }}" />`
`if self.request.user.type == "SALES":`	ROR	`if self.request.user.type != "SALES":`
`LOGIN_REDIRECT_URL = "clients:list_client"`	CRP	`LOGIN_REDIRECT_URL = "clients:list_user"`
`if user_role == "SALES":`	ROR	`if user_role != "SALES":`

The number of mutant programs that can be generated from the authentication function is four. Mutant version one is symbolized by m_1 and so on according to the number of mutant programs generated. These mutant programs will be tested using the same model used to test the original program.

ISSN:2337-7631 (Printed)
ISSN: 2654-4091 (Online)

Experiments with Mutation Testing

The process of testing test cases on the mutant program is done in the same way as testing test cases on the original program. This test will be used to check whether the mutant that has been created is successfully detected by the test case. If the results of testing the mutant program are different from the results of testing the original program, then the test case created successfully kills the mutant program, but if the results of testing the mutant program are the same as the original program, then the test case cannot kill the mutant program. The test results can be seen in the eclipse console which shows whether the test case passes or fails.

Testing Result

The results of the MBT implementation on five functions of the Kawn Subscriptions Manager application and the adequacy calculation of the test cases generated by MTB with mutation testing were obtained. In the application of MBT, five models were generated, where one model describes the flow of one function. Each model generates methods consisting of edges and vertices. Table 5 shows all the methods generated by the models. e_EdgesName represents edges and v_VerticesName represents vertices.

Table 5. Behavioural Model Result

Function	***Methods***	
	Edge	***Vertex***
Authentication	e_OpenLoginPage e_InputCorrectCredentials e_OpenLogoutPage e_Logout e_InputEmptyCredentials e_InputIncorrectCredentials	v_LoginPage v_ClientPage v_LogoutPage
Create User	e_OpenLoginPage e_InputCorrectCredentials e_OpenUserPage e_OpenCreateUserPage e_InputCorrectData e_Close e_InputIncorrectData e_InputEmptyData e_Cancel	v_LoginPage v_ClientPage v_UserPage v_CreateUserPage v_SuccessCreateUser v_AlertIncorrectData v_AlertEmptyData
Create Outlet	e_OpenLoginPage e_InputCorrectCredentials e_OpenOutletPage e_OpenCreateOutletPage e_InputCorrectData e_Close e_InputIncorrectData e_InputEmptyData e_Cancel	v_LoginPage v_ClientPage v_OutletPage v_CreateOutletPage v_SuccessCreateOutlet v_AlertIncorrectData v_AlertEmptyData
Create Subscription Plan	e_OpenLoginPage e_InputCorrectCredentials e_OpenSubscriptionPlanPage e_OpenCreateSubscriptionPlanPage e_InputCorrectData e_Close e_InputIncorrectData e_InputEmptyData e_Cancel	v_LoginPage v_ClientPage v_SubscriptionPlanPage v_CreateSubscriptionPlanPage v_SuccessCreateSubscriptionPlan v_AlertIncorrectData v_AlertEmptyData
Create Subscrciption	e_OpenLoginPage e_InputCorrectCredentials e_OpenSubscriptionPage e_OpenCreateSubscriptionPage e_InputCorrectData e_Close e_InputIncorrectData	v_LoginPage v_ClientPage v_SubscriptionPage v_CreateSubscriptionPage v_SuccessCreateSubscription v_AlertIncorrectData v_AlertEmptyData

Function	***Methods***	
	Edge	***Vertex***
	e_InputEmptyData	
	e_Cancel	

Based on the modelling of system behaviour, a total of 73 methods were obtained, consisting of 42 edges and 31 vertices. Each of these methods will be used to generate test cases for the purpose of testing the five application functions that have been determined. The number of test cases to be written corresponds to the number of methods generated by the behaviour model. Table 6 shows the results of testing using MBT on one of the functions, namely authentication.

Table 6. Authentication Original Testing Result

Edge/Vertex	***Number Traversed***	***Result***
e_OpenLoginPage	2	PASS
v_LoginPage	7	PASS
e_InputCorrectCredentials	2	PASS
v_ClientPage	2	PASS
e_OpenLogoutPage	2	PASS
v_LogoutPage	2	PASS
e_Logout	2	PASS
e_InputEmptyCredentials	2	PASS
e_InputIncorrectCredentials	1	PASS

Based on the test results, it can be obtained that all vertices and edges are successfully executed at least once. Based on the test results on the authentication function, it is obtained that there is no failure in the authentication function. The test results on the other four functions also show that there are no failures in the function. After testing with MBT, then testing with mutation testing is carried out. The results of mutation testing on the authentication function can be seen in Table 7.

Table 7. Authentication Mutant Testing Result

Edges/Vertex	***Original***	m_1	m_2	m_3	m_4
e_OpenLoginPage	PASS	PASS	PASS	PASS	PASS
v_LoginPage	PASS	PASS	PASS	PASS	PASS
e_InputCorrectCredentials	PASS	PASS	PASS	PASS	PASS
v_ClientPage	PASS	PASS	PASS	FAIL	PASS
e_OpenLogoutPage	PASS	PASS	PASS	FAIL	PASS
v_LogoutPage	PASS	PASS	PASS	FAIL	PASS
e_Logout	PASS	FAIL	PASS	FAIL	PASS
e_InputEmptyCredentials	PASS	PASS	PASS	PASS	PASS
e_InputIncorrectCredentials	PASS	PASS	PASS	FAIL	PASS

Based on the mutation testing results table on the authentication function above, it can be obtained that there are two mutants that can be killed by the test case. These mutants are mutant versions one and three which can be killed by at least one test case/method. It can also be seen that there is one equivalent mutant, namely mutant version two. Mutant version two produces the exact same output as the original program and none of the test cases are able to detect the existence of the mutant. This is different from mutant version four, although there is no test case that detects the fault created in the program, but the output produced is different from the original program so that mutant version four is not an equivalent mutant.

The difference in output that cannot be detected by test cases or methods in the authentication function occurs because mutant version four has the same behaviour as the original program. In testing, it was found that all vertices and edges were skipped at least once. However, there is a difference in the number of client lists displayed on the client list page. This difference cannot be captured because the test case only checks where the user is directed after login.

Based on the mutation testing results, the mutation score from mutation testing of the authentication function with N = 4, D = 2, E = 1 can be calculated with Equation 1.

$$Mutation\ Score = 100 * \frac{2}{(5-1)} = 66.67$$

ISSN:2337-7631 (Printed)
ISSN: 2654-4091 (Online)

Then, the mutation score for the authentication function is 66.67%. Mutation scores for all functions can be seen in Table 8.

Table 8. Testing Result

Function	***Edges/ Vertex***	***Number of Mutant***	***Kill Mutant***	***Equivalent Mutant***	***Mutation Score***
Authentication	9	4	2	1	66.67%
Create User	16	5	2	2	66.67%
Create Outlet	16	17	11	3	78.57%
Create Subscription Plan	16	9	6	0	66.67%
Create Subscription	16	17	14	0	82.35%
Total Mutation Score					360.93%

Based on the table above, it is obtained that there is no test suite or function that reaches 100% mutation score. The highest mutation score is obtained from the create subscription function with a mutation score of 82.35% and the lowest mutation score is obtained from the authentication, create user and create subscription plan functions with a mutation score of 66.67%. Based on observations during the experiment, the difference in mutation score from each function can be influenced by several things as follows:

1. Mutant type: Mutants that are successfully killed by the test case are mutants that affect the flow or behaviour of the Kawn Subscriptions Manager application. Mutants that only affect the amount of data displayed on an application page and affect the process of adding data to the database cannot be detected by the test case.
2. Quality of test cases: Test cases generated by MBT are limited to how the system runs or how it moves from one state to another under certain conditions.
3. Program code complexity: Each function in the tested Kawn Subscriptions Manager application has a different program code complexity. Functions that have low complexity tend to have fewer mutants and a lower mutation score than functions that have high complexity. Complexity in this case can be seen from the number of operators, methods, constants and classes used in each function.

From Table 8, it is obtained that the total mutation scores of this test is 360.93%. This result is obtained from the sum of the mutation scores of the five functions that have been tested using mutation testing. Then the average mutation score can be calculated with Equation 2.

$$Average\ Mutation\ Score = \frac{360.93\%}{5} = 72.18\%$$

Based on the calculations above, it is obtained that the average mutation score of MBT implementation in testing the Kawn Subscription Manager application measured using mutation testing is 72.18%.

4. CONCLUSION AND SUGGESTION

Based on the results of the research that has been done on testing the Kawn Subscriptions Manager application with the MBT method and calculating the mutation score to calculate the adequacy level, the following conclusions can be drawn:

1. The application of the model-based testing method to the five functions of the Kawn Subscriptions Manager application tested resulted in five behavioural models with a total of 73 methods consisting of 31 vertices and 42 edges and the test results showed no failures found in the Kawn Subscriptions Manager application.
2. Test cases generated by applying model-based testing have a mutation score level of 72.18% or in the medium category or can be said to be quite adequate. Based on observations made, it was found that model-based testing cannot detect mutants that do not change the behaviour of the application.

However, testing the Kawn Subscriptions Manager application is still only limited to five functions, so in the next test it is expected that all functions in the application can be tested with the MBT method. The MBT method used to test the application also produces test items that are only able to detect mutants that change application behaviour but not mutants that affect the amount of data displayed and data that is not successfully saved to the database. Therefore, another method is needed that is able to detect these types of failures.

REFERENCES

[1] J. F. S. Ouriques, E. G. Cartaxo, and P. D. L. Machado, "Test Case Prioritization Techniques for Model-Based Testing: A Replicated Study," Aug. 2017. [Online]. Available: http://arxiv.org/abs/1708.03240

[2] V. Singh and S. Ramasamy, "An exploration of model based testing," *Article in International Journal of Scientific and Engineering Research*, vol. 6, no. 2, 2015, [Online]. Available: https://www.ijser.org/paper/An-Exploration-of-Model-Based-Testing.html. [Accessed 10 October 2022].

[3] R. Reddy, P. Syed, and A. Irtaza, "Software Testing: A Comparative Study Model Based Testing VS Test Case Based Testing," 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:17864120. [Accessed 10 October 2022].

[4] L. Apfelbaum and J. Doyle, "Model Based Testing,", May 1997. [Online]. Available: https://api.semanticscholar.org/CorpusID:8815835. [Accessed 10 October 2022].

[5] H. Felbinger, F. Wotawa, and M. Nica, "Empirical study of correlation between mutation score and model inference based test suite adequacy assessment," in *Proceedings - 11th International Workshop on Automation of Software Test, AST 2016*, Association for Computing Machinery, Inc, May 2016, pp. 43–49. doi: https://doi.org/10.1145/2896921.2896923

[6] M. Utting, A. Pretschner, and B. Legeard, "A TAXONOMY OF MODEL-BASED TESTING," 2006. doi: https://doi.org/10.1002/stvr.456

[7] L. L. Muniz, U. S. C. Netto, and P. H. M. Maia, "TCG: A model-based testing tool for functional and statistical testing," in *ICEIS 2015 - 17th International Conference on Enterprise Information Systems, Proceedings*, SciTePress, 2015, pp. 404–411. doi: https://doi.org/10.5220/0005398604040411

[8] M. N. Zafar, W. Afzal, E. Enoiu, A. Stratis, A. Arrieta, and G. Sagardui, "Model-Based Testing in Practice: An Industrial Case Study using GraphWalker," in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Feb. 2021. doi: https://doi.org/10.1145/3452383.3452388

[9] J. Korhonen, W. Afzal, and E. P. Enoiu, "Automated Model Generation Using GraphWalker Based On Given-When-Then Specifications," 2020. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:1451401/FULLTEXT01.pdf. [Accessed 10 October 2022].

[10] X. Y. Djam and Y. H. Kimbi, "A Fault-Based Testing Approach in Safety Critical Medical Systems," *Journal of Software Engineering and Applications*, vol. 13, no. 06, pp. 129–142, 2020, doi: https://doi.org/10.4236/jsea.2020.136009

[11] R. Just, G. M. Kapfhammer, and F. Schweiggert, "Do redundant mutants affect the effectiveness and efficiency of mutation analysis?," in *Proceedings - IEEE 5th International Conference on Software Testing, Verification and Validation, ICST 2012*, 2012, pp. 720–725. doi: http://dx.doi.org/10.1109/ICST.2012.162

[12] A. Derezińska and K. Hałas, "Experimental evaluation of mutation testing approaches to Python programs," in *Proceedings - IEEE 7th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2014*, IEEE Computer Society, 2014, pp. 156–164. doi: https://doi.org/10.1109/ICSTW.2014.24

[13] A. Derezinska and M. Rudnik, "Evaluation of mutant sampling criteria in object-oriented mutation testing," in *Proceedings of the 2017 Federated Conference on Computer Science and Information Systems, FedCSIS 2017*, Institute of Electrical and Electronics Engineers Inc., Nov. 2017, pp. 1315–1324. doi: https://doi.org/10.15439/2017F375

[14] R. Maxion, "Experimental Methods for Computer Science Research," Institute of Electrical and Electronics Engineers (IEEE), Sep. 2009, pp. 136–136. doi: https://doi.org/10.1109/LADC.2009.29