# A TEST AUTOMATION FRAMEWORK IN POCT SYSTEM USING TDD TECHNIQUES

**RATCHANOK CHAIPRASERTH**

**A THESIS SUBMITTED IN PARTIAL FULFILLMENT**
**OF THE REQUIREMENTS FOR**
**THE DEGREE OF MASTER OF SCIENCE**
**(TECHNOLOGY OF INFORMATION SYSTEM MANAGEMENT)**
**FACULTY OF GRADUATE STUDIES**
**MAHIDOL UNIVERSITY**
**2013**

Thesis
entitled

# A TEST AUTOMATION FRAMEWORK IN POCT SYSTEM USING TDD TECHNIQUES

….……..……..
Miss Ratchanok Chaipraserth
Candidate

……..…..…..……
Asst. Prof. Supaporn Kiattisin,
Ph.D. (Electrical and Computer Engineering)
Major advisor

……..…..…..……
Asst. Prof. Adisorn Leelasantitham,
Ph.D. (Electrical Engineering)
Co-advisor

……..…..…..……
Lect. Waranyu Wongseree,
Ph.D. (Electrical Engineering)
Co-advisor

……..…..…..……
Prof. Banchong Mahaisavariya,
M.D., Dip. (Thai Board of Orthopedics)
Dean
Faculty of Graduate Studies,
KMUTT University

……..…..…..……
Asst. Prof. Supaporn Kiattisin,
Ph.D.
(Electrical and Computer Engineering)
Program Director
Master of Science Program in Technology of Information System Management
Faculty of Engineering,
Mahidol University

Thesis
entitled

# A TEST AUTOMATION FRAMEWORK IN POCT SYSTEM USING TDD TECHNIQUES

was submitted to the Faculty of Graduate Studies, Mahidol University
for the degree of Master of Science
(Technology of Information System Management)
on
December 26, 2013

……
Miss Ratchanok Chaipraserth
Candidate

……
Asst.Prof.Bunlur Emaruechi,
Ph.D.
(Environment Systems Engineering)
Chair

……..
Lect. Supaporn Kiattisin,
Ph.D. (Electrical and Computer Engineering)
Member

……..…
Asst. Prof. Adisorn Leelasantitham,
Ph.D. (Electrical Engineering)
Member

……
Lect.Waranyu Wongseree,
Ph.D. (Electrical Engineering)
Member

……
Asst.Prof.Werapon Chiracharit,
Ph.D. (Electrical and Computer Engineering)
Member

……
Prof. Banchong Mahaisavariya,
M.D., Dip (Thai Board of Orthopedics)
Dean
Faculty of Graduate Studies
Mahidol University

……
Lect. Worawit Israngkul,
M.S. (Technical Management)
Dean
Faculty of Engineering
Mahidol University

# ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to my major advisor, Asst. Prof. Supaporn Kiattisin for guidance me to the right path, my co-advisor, Asst. Prof. Adisorn Leelasantitham, and Lect. Wanranyu Wongseree for fulfill my knowledge and the best suggestion throughout this research.

I am obliged to all the lecturers and all staff of the Technology of Information System Management Program, Faculty of Engineering, Mahidol University for their service and support.

Finally, I also take this opportunity to express a deep sense of gratitude to my parents, my friends for their encouragement and fulfill a power when i was tired.

Ratchanok Chaipraserth

A TEST AUTOMATION FRAMEWORK IN POCT SYSTEM USING TDD TECHNIQUES

RATCHANOK CHAIPRASERTH 5436431 EGTI / M

M.Sc. (TECHNOLOGY OF INFORMATION SYSTEM MANAGEMENT)

THESIS ADVISORY COMMITTEE: SUPAPORN KIATTISIN, Ph.D., ADISORN LEELASANTITHAM, Ph.D., WARANYU WONGSEREE, Ph.D.

ABSTRACT

The growth of IT business is highly competitive, both in the development process and the functionality of software requirements of customer. To increase parallel of the software quality and reduced cost and time. The it business can then begin to focus on customer satisfaction; the greater the speed of the business process to maximize business opportunities.

The problems and constraint of software development under pressured environments are likely to increase the release of software that do not meet requirement specifications or are susceptible to programming errors. This paper presents an implementation of technique model which is an agile development methodology using Test Driven Development (TDD) in the automation testing framework for the software development process. It helps optimization software testing as it can locate program failures defects early, and is flexible for requirement changes in the future; either to increase or change requirement without any effect on coding behaviors. It also helps to reduce the cost of software development, as human resource time is reduced and provides rapid feedback of test result.

This researcher has applied this approach to the software development process prototype from the Point of care testing (POCT) system as an example for creating test cases using the black box and white box testing techniques that were based on the requirements specification to improve the quality of product.

KEY WORDS: TESTING PROCESS / SOFTWARE TEST PROCESS / BLACK-BOX TESTING / WHITE-BOX TESTING / TEST DRIVEN DEVELOPMENT/ TEST CASES GENERSTION

96 pages

กรอบการทดสอบระบบอัตโนมัติในระบบ POCT โดยใช้เทคนิค TDD

A TEST AUTOMATION FRAMEWORK IN POCT SYSTEM USING TDD TECHNIQUES

รัชนก ไชยประเสริฐ 5436431 EGTI/M

วท.ม. (เทคโนโลยีการจัดการระบบสารสนเทศ)

คณะกรรมการที่ปรึกษาวิทยานิพนธ์: สุภาภรณ์ เกียรติสิน, Ph.D., อดิศร ลีลาสันติธรรม, Ph.D., วรัญญู วงษ์เสรี, Ph.D.

บทคัดย่อ

การเติบโตของธุรกิจด้านไอทีในปัจจุบันมีการแข่งขันสูงทั้งในด้านกระบวนการพัฒนาซอฟท์แวร์และฟังค์ชันการใช้งานเพื่อให้สามารถตอบสนองความต้องการของผู้ใช้งานมากที่สุด ในการเพิ่มคุณภาพของซอฟต์แวร์จะควบคู่ไปกับการลดค่าใช้จ่ายและเวลาในการผลิตซอฟต์แวร์ควบคู่กันไป โดยธุรกิจด้านไอทีเริ่มให้ความสำคัญกับความพึงพอใจของลูกค้าเพื่อเพิ่มโอกาสในการเติบโตทางด้านธุรกิจมากยิ่งขึ้น

ปัญหาเดิมคือการทำงานภายใต้สภาพแวดล้อมที่มีความกดดันส่งผลให้พบบัคมากขึ้นในช่วงการทดสอบระบบและการส่งมอบโปรแกรมพบว่าฟังค์ชันการทำงานของโปรแกรมไม่ครบตามความต้องการของลูกค้า งานวิจัยนี้เสนอการใช้การพัฒนาระบบแบบอไจล์ ซึ่งเทคนิคที่เราเลือกใช้คือ TEST DRIVEN DEVELOPMENT (TDD) ภายใต้กรอบการทดสอบระบบแบบอัตโนมัติในกระบวนการพัฒนาซอฟท์แวร์ซึ่งสามารถช่วยเพิ่มประสิทธิภาพในการทดสอบของซอฟท์แวร์ เช่น สามารถพบบัคได้เร็วในขั้นตอนการพัฒนาระบบ, โปรแกรมมีความยืดหยุ่นในการปรับเปลี่ยนตามความต้องการของลูกค้าที่อาจเกิดขึ้นหรือเปลี่ยนแปลงได้ในอนาคตโดยไม่กระทบกับโครงสร้างของโปรแกรมหรือโค้ดในส่วนอื่นๆ นอกจากนี้ยังช่วยลดต้นทุนในในการพัฒนาซอฟท์แวร์ด้านทรัพยากรคน, เวลาและสามารถรายงานผลการทดสอบได้อย่างรวดเร็ว

งานวิจัยนี้นำวิธีข้างต้นมาใช้ในกระบวนการพัฒนาซอฟท์แวร์โดยใช้ในระบบ POINT OF CARE TESTING (POCT) เป็นกรณีศึกษา การสร้างกรณีทดสอบด้วยเทคนิคการทดสอบแบบกล่องดำและกล่องขาวโดยอิงตามความต้องการของลูกค้าเพื่อให้ซอฟท์แวร์มีคุณภาพมากยิ่งขึ้น

96 หน้า

# CONTENTS

# CONTENTS (cont.)

# LIST OF TABLES

# LIST OF TABLES (cont.)

# LIST OF TABLES (cont.)

# LIST OF FIGURES

# LIST OF FIGURES (cont.)

# CHAPTER I
# INTRODUCTION

## 1.1 Background and Problems

Software testing have a role driven in software development industry for evaluating software to be effective and high quality standard and aim to minimum production costs. "Software testing is an important component of software quality assurance, and many software organizations are spending up to 40% of their resources on testing. For life-critical software testing can be highly expensive." [4] Especially for the larger of software, costs and maintenance of software testing is on average 50-70%. [7] Software testing process can be applied to the development software process. First, we show how does the original development system do? An original system development we focus on design-code-test process. Testing phase was started after code was written by developer already. The problems of original software develop process included:

1.1.1 Testing phase will start near the end of development under pressure environment of work (The time of delivery, insertion task, and overload task, etc.). [2] [9] that cause why we found many bugs in testing phase.

1.1.2 Under the pressure environment that we have a less time to review requirement between system analysis and developer that are cause to misunderstood and unclearness requirement for developer. And developer always speeds up coding for delivery in time. After finish development, on testing phase we found system features not cover all the requirement specifications. That very busy to solve this problem due to program customization will make an affective with structure of program. [9]

1.1.3 Retest after fixed bugs are takes to long time. Due to the pressure in various development programs, on testing phase we found many bugs more than expected. The testing process after fixed bugs must to repeat same process testing that will takes too long time.

This paper presents an implementation of technique model which is an agile development using Test Driven Development (TDD) in automation testing framework for software development process. And apply this approach with prototype in software development process using the Point of care testing (POCT) system as an example to create test case with black box and white box testing techniques based on the requirements specification

## 1.2 Objectives

The objective of this work is

1.2.1 To prove the theory of TDD help us increase code quality for reduce cost and time in maintenance phase.

1.2.2 To prove the theory of TDD can reduce cost and time in maintenance phase.

1.2.3 To prove the theory of TDD can rapid feedback of test result

## 1.3 Scope of Work

The scope of this work included the following:

1.3.1 Framework for automation testing

1.3.2 Test driven development for implement in development process

1.3.3 Black-box and White-box testing techniques for creates test cases

1.3.4 Point of care testing system for implement approach as case study

## 1.4 Expected Result

1.4.1 Increase code quality of software for reduces cost and time in maintenance phase.

1.4.2 Reduce cost and time in maintenance phase.

1.4.3 In development process can rapid feedback of test result

# CHAPTER II
# LITERATURE REVIEW

## 2.1 Overview

Software testing have a role driven in software development industry for evaluating software to be effective and high quality standard and aim to minimum production costs. "Software testing is an important component of software quality assurance, and many software organizations are spending up to 40% of their resources on testing. For life-critical software testing can be highly expensive." [4] Especially for the larger of software, costs and maintenance of software testing is on average 50-70%. [7] Software testing process can be applied to the development software process. First, we show how does the original development system do?

An original system development we focus on design-code-test process. (See Fig.1) Testing phase was started after code was written by developer already. The problems of original software develop process included:

(a) Testing phase will start near the end of development under pressure environment of work (The time of delivery, insertion task, and overload task, etc.). [2] [9] that cause why we found many bugs in testing phase. It difficultly to estimate time plan to fix bug that makes time plan will delay and cost to fix bug is very high.

(b) Under the pressure environment that we have a less time to review requirement between system analysis and developer that are cause to misunderstood and unclearness requirement for developer. And developers always speed up coding for delivery in time. After finish development, on testing phase we found system features not cover all the requirement specifications. That very busy to solve this problem due to program customization will make an affective with structure of program. [9] (Program customize is limited from the structure design)

(c) Retest after fixed bugs are takes to long time. Due to the pressure in various development programs, on testing phase we found many bugs more than

expected. The testing process after fixed bugs must to repeat same process testing that will takes too long time.

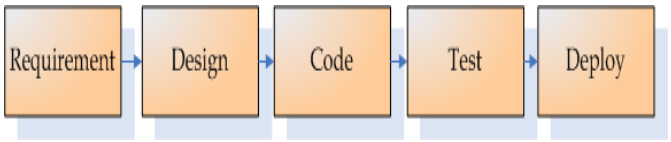


Figure 2.1 An original Software Development Life Cycle (SDLC)

In Figure 2.1 is an original software development that are including.

- **Requirement:** First analyses, then determining and prioritizing customer requirements.
- **Design:** Step two is after gathering requirements to make system design framework with visual basic, java or etc.
- **Code:** Step tree is development process and begin implementing of code.
- **Test:** Step four is testing process and begins to create test case for implementation of test.
- **Deploy:** After finished development, the last step of SDLC is implementation on customer site. Make sure that system runs smoothly.

In Figure 2.1 shows, processes of a waterfall model. The benefit to the waterfall model is focused on gathering the requirements. Requirements are very important so that when new programmer to join the development process. The requirements will be used to accustom themselves with the development process how waterfall is a joint that needs to be changed constantly. Changes in demand may mean the design and/ or progression has to be repeated to adjust the need of customer. As changes the required in the future and therefore the cost of the development process are increased. The problem of an original software development that are following as:

i) Customers cannot imagine and conclude their needs until they see the program.
ii) Developer cannot know the limitations or problems until them coding program.
iii) Customers can try to use program at the end of which there is almost no time to change anything else.

## 2.2 Related Work

Among the existing researches, agile software methodology is an ideal framework for software development process that provides development repetitions all through the SDLC process. Provides to be implemented of TDD process Muller and Hagner [13] implemented the TDD compared to programming the experiments conducted with 19 students, measuring the productively of TDD concern in (1) time process if development (2) quality of code (3) comprehensible. They divided the students two group included TDD group and the control group. The students do with the same process. They completed a given program has been provided with the necessary design and publishing. Researchers build the coding in this concept to simplify for acceptance testing in the automation system.

TDD group students implemented the test case during the code is finished as report. And the control group implemented automatically test cases after the finish of code was implemented in the second step (IP) following as the development process. Acceptance testing (AP) The IP students are aware of the acceptance test they do not pass, they will be given the chance to correct. Their code, the researchers found no difference between groups in the development process time, no dependable TDD group reduction after IP and reliability. Higher after the process of AP, but the TDD errors are less significant statistically when code is reused based on the results, the researchers summarized that coding in the first test does not led to the development of faster or with higher quality. But the acceptance of a measured in condition of the proper use of the connection existing results of these trials need to be considered in the context of the limitations of the sample size is small. Students have limited experience with TDD and the results have been blurred due to the large variability of the data, the external validity of the results they can. Be improved by using additional education. With professional programmers

Boby George and Laurie Williams [14] started a group of designing the experiments with 24 pair developers. One group developed a program using TDD while the control group used an original waterfall model. Experimental results, focus on external validity concerns, indicate that TDD developers create higher quality code because they passed 18% more functional black-box test cases. However, TDD group took 16% more time. Results of statistical analysis can showed that a moderate

statistical existed between time spent and the resulting quality. Finally, the developers in the control group often did not write the required automated test cases after finishing their code. Thus it could be known that waterfall model do not support suitable testing. This instinctive watching supports the awareness that TDD has the important for increasing the quality of testing.

## 2.3 Agile Development

Agile software methodology is a ideal framework for software development process that provides the development of SDLC is a simple set of steps with developers, managers and customers. In order to have enough information in the system. A results, software quality and software framework to generate in the future. Agile methodology focus on empirical rather than defined methods (like a waterfall model) is all about the flexible to changing the requirement in the future development. By the way, define what we mean which one is planned before enforcing these plans. However, the plan relates to things like agile and adjusts these plans to the results. Extreme Programming (XP) is a good example of how agile that will be following as:

i) Discussion and communication between the customers and the development team.

ii) Clearly and simple design

iii) The rapid feedback received on the 1st of software testing.

iv) The delivery and implementation of the changes early.

Agile methodology is to cut down the system size of the big into a fitting size together when the time is right as well such as design, programming and testing. So while the reason to support both waterfall and agile methods. But look closer statement following reasons for selecting the agile methodology than the waterfall method. i.e.:

(a) After the each process is completed in the waterfall model cannot be back because most software is designed and implemented difficult to change over time and the needs of the user. So this problem can be solved by backward and designing new systems all the way to increase more costly and inefficient parts. Agile

methodology can be adapted to changes in each stage of the development process. By help to make changes easier without restructuring of the whole program. This method not only reduces costs but also improves customer satisfaction too.

(b) Another advantage of agile model is one has a release product at the end of each tested step. This can make sure the defects are found and get rid of in the development process, and the product can be retested again after the first bug found. This is impossible in the waterfall model, so the product is tested only once at the end after finished development, which means any bugs can found in all whole of development process.

(c) Agile modular nature is object-oriented design and suitable for applications whose working patterns are released in a timely manner, although not always completely meet the needs of customers. While there is only one opening in the waterfall method and any problems or delays, customers are not highly satisfied.

(d) Agile methodology provide for define changes following the customer requirements and customer satisfaction. As this solution is impossible when the waterfall model is implemented, so any changes to be occur that make the each steps of process has to be repeated all steps again.

(e) But both models do allow for a class the waterfall group is done at each step. As for agile model, each coding function can be represented to divide groups. This allows for many parts of the process to be complete at the same time, although group is more effectively used in agile methodology.

## 2.4 Test Driven Development

Difference from the original waterfall model, the agile development methodology (see Figure 2.2) is focus on repetitious and incremental of development. Software development as agile model is developed incrementally and is not build the entire system at once. Spend less time on documentation requirements development trend the customer requirements have occurs simultaneously while the development, which is different from the waterfall model with test system performance after development are finished.

Test Driven Development (TDD) was developed the technique by Kent Beck, 2003[1]. TDD is a modern alternative to system as one of agile development that return design, code, test, deploy to the deploy, test, code, design but called run, test, code, refactoring by creating a test code from test case that are generated from requirement specification [9] before writing functional code [2]. Testing phase will be moved from the end to the front of coding and design in software life cycle process [9]. TDD is related to the test-first concepts of extreme programming [1] [2]. Advantages of this concept, developer can analysis input and output clearly before writing functional code so they was forced to think about what we want to do and how to test code therefore we can found early bug , faster feedback of test result and better collaboration between customer and developer [9].

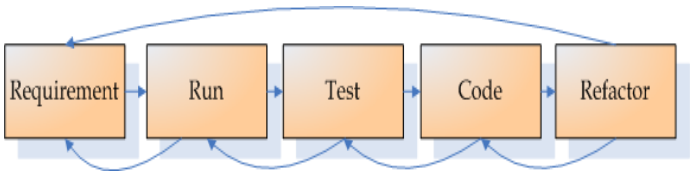


Figure 2.2 A Simple of TDD Process

TDD start after we keep requirement from customer to created requirement and after then we will generate test case for prepare step of testing to make test script in TDD. And TDD process will complete when all test script cover all the requirements specifications [2] [1] [9]. The steps of TDD including: (see Figure 2.3)

i) Add Test: After we analyzed requirement specification to created test case of each sub-section with black box and white box testing techniques for optimization input data. Then we write a small test code (no more than five).

ii) Watch Test Fail: After test code was added, we try to compile test code, due to not have functional code was created and it should be fail [9]. The test bar should turn red.

iii) Fix Compile Error: We write just enough functional code for purpose to quickly to pass test.

iv) Run Tests: We compile test again after write a small of functional code to fix error. If test fail, developer can look at error code and go to fix error in sub test script easily [2]. We try to write code to fix bug until test pass. The test bar will turn green.

v) Refactoring: After all tests was passed, we adjusting the code as verify and correcting without changing external behavior and hidden bugs [9]. [8] Aim to remove unnecessary code, improve structure, and make code easier to understand and overview [9]. After refactoring, run the tests and make sure they still pass.

vi) Repeat: Do the step above again until you can't find any more tests that drive writing new code.

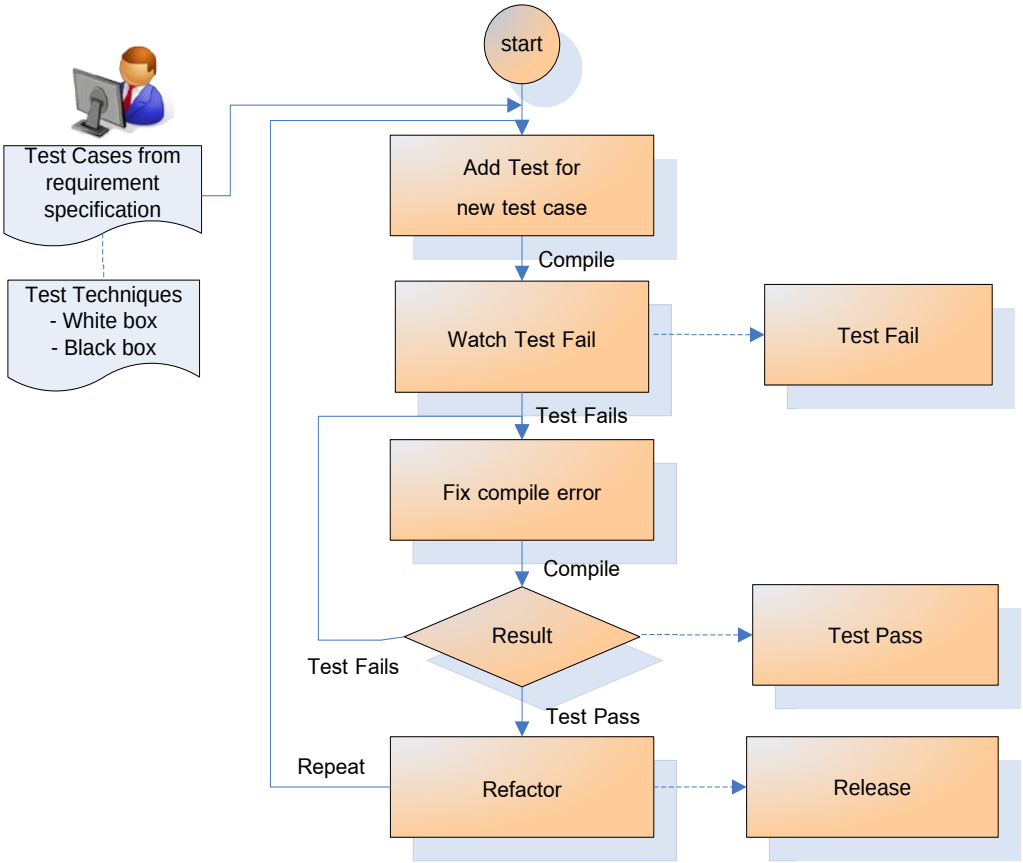


Figure 2.3 TDD Process Flow Overview

## 2.5 Testing Process

System testing to find an error. Improves system integrity by checking errors in the design and system. The tests are intended to help prevent and detect error in the system, which will increase value to the product in accordance with customer requirements. Software testing is required to measure the standards or requirements so it can provide accurate results from wrong. The following [11] to use for understanding these concepts: [10] [12]

- **Mistake:** A human error that make the process incorrect result.
- **Defect:** Process or the data identification is incorrect.
- **Failure:** The incompetence of a program to execute its expected function within the requirement specifications.
- **Error:** The disagreement between test condition and the true.
- **Specification:** The document has identified the need perfectly. Such as design, requirements system need, or other characteristics of a system and the step in determining whether these provisions.

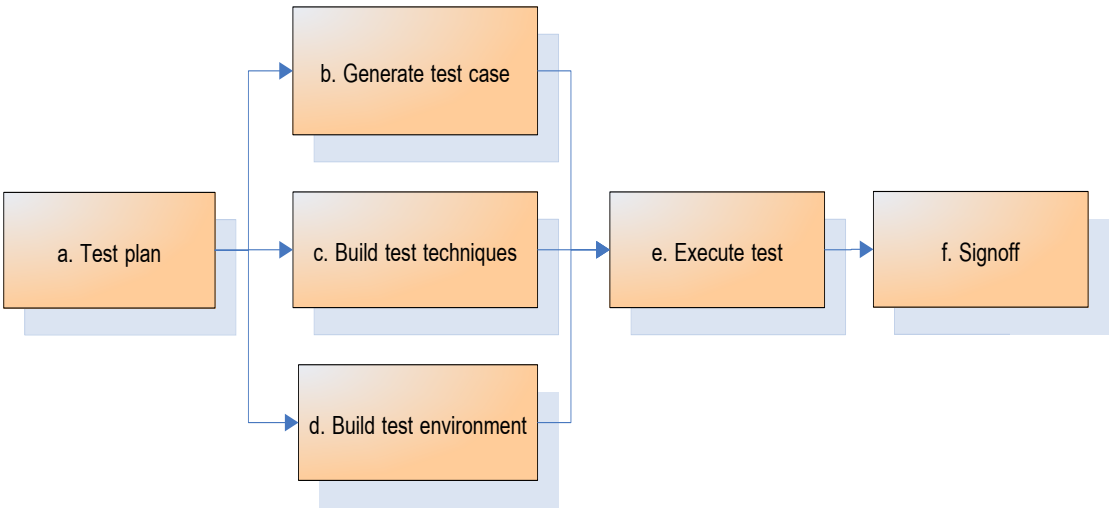


Figure 2.4 Test Process Flow

The diagram in Figure 2.4 is the test process concept are included.

**(a) Test plan:** Concern with the requirement, test plan, human resource, time period and define responsibilities. The reason that we need to create test plan include: [12]

- Preparation: To make ensure that every test is considered and process with the carefully.
- Communication and training: A training session for help in testing phase.
- Effectiveness: To provide a plan to test the scheme. Limitations and the need for the purpose of determining the expectation. Resource requirements to find the cause of incorrect results and efficiency.

The objective of documenting test plans is providing tools for supporting the test team. Summarized as follows: [12]

- Reduces the risk of work that may be omitted from the error.
- The prioritization of tasks to perform to complete.
- Used for the organization of the test.
- Used as a tool to check the progress of the test system.
- Assess the risks involved in the organization.

**(b) Generate test case**: Involves identifying test cases, input-output data, expected results and test condition. In general, the tester will define the conditions of the test, such as test case generation, input and out put data. The desired and expected results of the test system were derived from requirement specifications in association with the developer.

**(c) Create test configure**: Include set up process like a black-box and white-box testing.

**(d) Create test environment**: Include preparing the hardware, software and other related installations.

**(e) Execute program testing:** All test cases are defined before the development is completed. In action, the test results are recorded and reported errors to occur through the development concerned.

**(f) Signoff**: Signoff occurred after UAT testing and achieved criteria through out.

## 2.6 Testing Techniques

Software testing becomes more complexity especially for a large of software system. Thus we also apply various testing techniques into our approach, which we are more confident that achieve more effective testing and help us to provides a roadmap decision for create test case. So the test case is not from the assumption absolutely. The both of test techniques include black box and white box testing will allow design criteria for generating test cases and test condition in automation system to get better results than assumption or use only one testing techniques. [4].

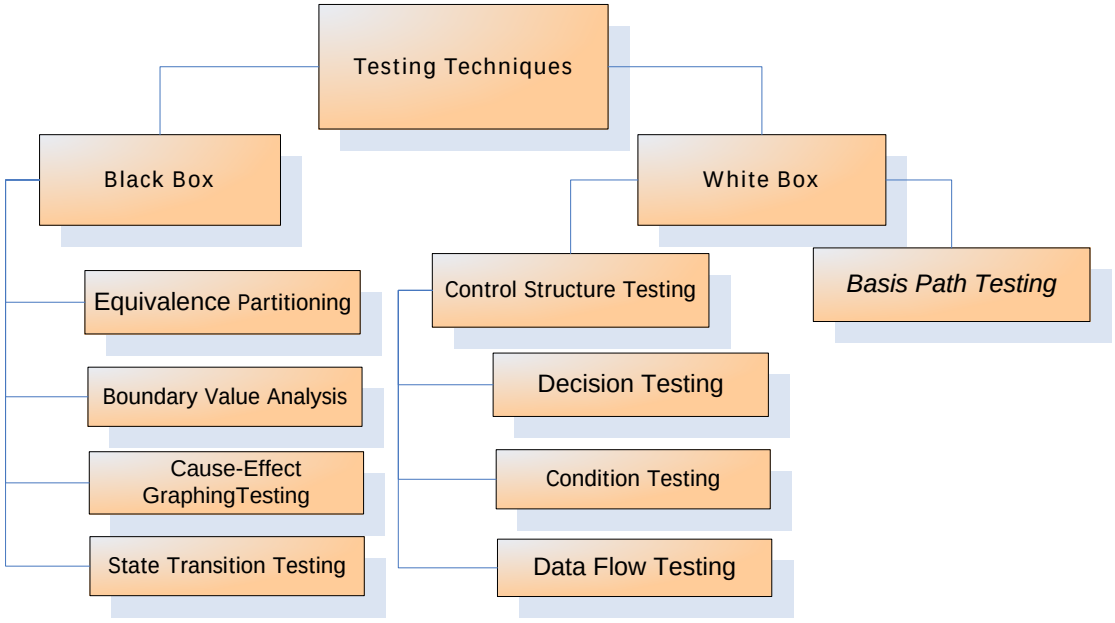


Figure 2.5 Classification of Test Techniques

The both of testing techniques are used in our approach and classification see in Figure 2.5 [4] [6]

2.6.1 Black box testing

2.6.2 White box testing

**2.6.1 Black box testing** is functional testing by ignore the source code and focuses on the output that comes out after the response of the system by choosing input and execution condition [4]. Black box testing can called opaque testing,

behavioral tests or closed box testing. Concept of black box testing is choosing suitable test data for system functional [12] see in Figure 2.6. The other includes:

- The expected result of the program is used to generating the test cases i.e. test cases are defined from requirement specification.
- The inside structural of code is not used for generating test cases.

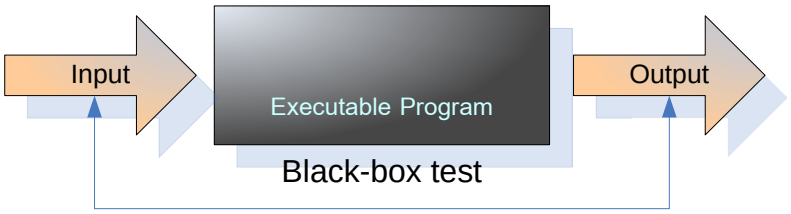


Figure 2.6 Black-box Testing

We chosen testing techniques include:

**(a) Equivalence Partitioning**: This technique is divided input and output domain in classes of data . This value is used as a criterion to define information as data valid and invalid data and help to reduce the amount of data used for the test. Test was chosen to represent class of data and data state in the same way both valid and invalid values (range, specific value, set, boolean) are used in Table 2.1 [4]. [6] The other concept includes: [12]

- Divide the input space into equivalent classes.
- Can reduce the equivalent test cases if it can be identified.
- Approximate by specifies the class which is assigned to different behavior.
- Assignation behavior requires for the class members.
- Software which is likely to be created for all cases that it fail or none such as if the program is not designed for negative numbers, it always fails for negative numbers too.
- Every condition specified as input is an equivalent class.
- Should consider outputs also and then give test cases for different classes.
- Selecting the parity equivalence for each of inputs value, test cases have to be selected. Choose the test cases covering as many valid equivalence value

as possible or have a test case that covers at most one valid class for each input and plus a separate test case for each invalid class.

**Table 2.1 Example of Equivalence Partitioning value [12]**

| **Data Type** | **Test condition** | |
|---|---|---|
| | ***Equivalence Partitioning*** | ***Input*** |
| Valid | Result 1: Contains numbers | String |
| | Result 2: Lower case | String |
| | Result 3: Upper case | String |
| | Result 4: Special character | String |
| | Result 5: string lenghgt 0-N(max) | String |
| | Result 6: Int in valid range | Numeric |
| Invalid | Result 1: non-ascii char | String |
| | Result 2: str len > N | String |
| | Result 3: Int out of range | Numeric |

**(b) Boundary Value Analysis:** These are called extreme cases. This technique fulfill the equivalence partitioning technique by choose value beside input and output domain include maximum, minimum, inside ,outside, below and above boundaries for generating test cases [4][6]. The test occurs at the boundaries of the input data. Design verification test on upper and lower limits of equivalence class and focus on out put of program in Table 2.2. The other concept includes:

- Systems always fail with the special values.
- The values are always on the boundary of class equivalence.
- Test cases at the boundaries with high returns value.

**Table 2.2 Example of Boundary Value Analysis value**

| Data Type | Test condition | |
|---|---|---|
| | ***Equivalence Partitioning*** | ***Value*** |
| Valid | Result 1: $1 \leq x \leq 6.0$ | 1.0,6.0 |
| Invalid | Result 1: $1 \leq x \leq 6.0$ | 0.9, 6.1 |

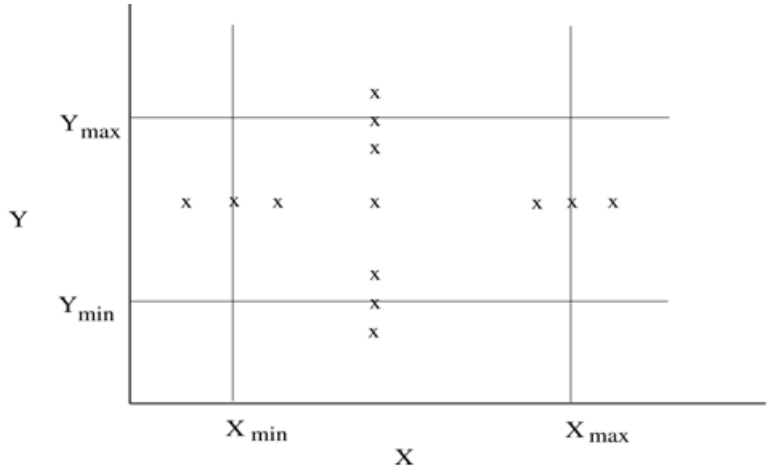


Figure 2.7 Example BVA of test cases for two vars – x and y

**Table 2.3 Example of test case by BVA method**

| Test Case | Meaning | Expected Output |
|---|---|---|
| 1 | Min | Minimal |
| 2 | Min+1 | Just above Minimal |
| 3 | Norm | Average |
| 4 | Max-1 | Just below Maximum |
| 5 | Max | Maximum |

**(c) Cause-Effect Graphing Testing:** This technique considers only the external behavior of a system that aim in selecting test that logically relate causes (inputs) to effects (outputs) for generating test cases [4][6]. A test design can see the logic conditions and related actions was presented by node and link between node for show relationship. Then it was used to create decision table and test data. Afterwards, the graph must be converted to a decision table see example in Table 2.4 includes:

- Helps in selecting combinations as input conditions to handle multiple inputs, different combinations of equivalent classes of inputs can be tried.
- Number of combinations can be large, if n diff input conditions such that each condition is valid/invalid, total: 2n
- Identify causes and effects in the system
    - Cause: distinct input condition which can be true or false
    - Effect: distinct output condition (T/F)
- Identify which causes can produce which effects; can combine causes
- Causes-effects are nodes in the graph and arcs are drawn to capture dependency; and/or are allowed
- From the Causes-effects graph, can make a decision table
    - Lists combination of conditions that set different effects
    - Together they check for various effects
- Decision table can be used for forming the test cases

**Table 2.4 Example of test case by cause- effective graphing technique**

| Test Case | Input (Causes) | | | | | Expected Output |
|---|---|---|---|---|---|---|
| | NL | NH | CL | CH | DC | |
| 1 | 20 | - | - | - | normal | nothing |
| 2 | 15 | - | - | - | normal | alert green flag low |
| 3 | - | 99 | - | - | normal | alert green flag high |
| 4 | - | - | 1 | - | normal | alert red flag low |
| 5 | - | - | - | 120 | normal | alert red flag high |
| 6 | - | - | - | 120 | abnormal | alert red flag high+ flag delta check |
| 7 | - | - | - | - | abnormal | alert flag delta check |

* Define: Normal Range= 20-99, NL: Normal low, NH: Normal high, CL: Critical low, CH: Critical high, DC: Delta check

**(d) State Transition Testing:** This technique based on transitions state testing from one state of an object to another state which state transitions are possible and impossible value. Incorporate the state transition testing preparation into the test case generating process [6] in Table 2.5

**Table 2.5 Example of state transition testing**

| **From state** | **To state** | | | | | |
|---|---|---|---|---|---|---|
| | Add | Edit | Update | Delete | Warning | Close |
| Add | N | N | Y | N | N | Y |
| Edit | N | Y | Y | N | N | Y |
| Update | Y | Y | Y | Y | N | Y |
| Delete | N | N | Y | N | Y | Y |
| Warning | N | N | N | Y | Y | Y |
| Close | Y | Y | Y | Y | Y | Y |

**2.6.1 White box testing** White-box testing is structural test which gets into account the internal mechanism of a system or component. It's can call as a structural testing; glass box testing and clear box testing by interested in source code and focus on conditions and paths for generate test cases. The main concept includes:

- A test to see the structure or passage in the program
- To create test scripts for the experiment in various conditions
- The test scripts must consist of a set that can be processed as normal and abnormal part
- Make every effort can be executed statement at least one

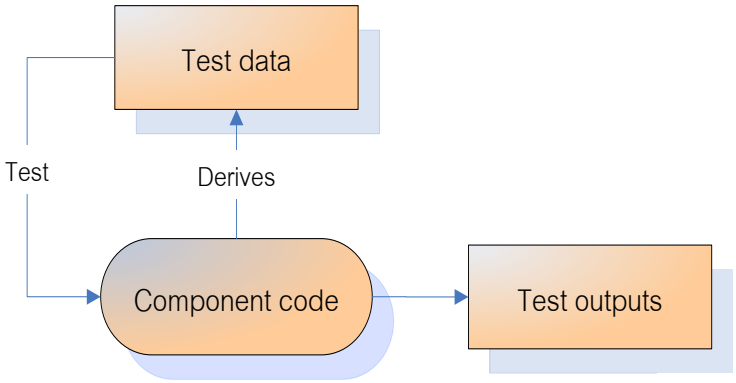


Figure 2.8 White-box Testing

We chosen test techniques include:

**(a) Basis Path Testing:** This technique is procedure to find possible executable path in order to maximize the coverage of each test case can representation structure program as simple flow graph see Fig. 8.

- Flowchart Change to the Graph that contains vertices and edge.
- Make sure all the paths are covered
- Determine the paths
- Construct a logic flow chart

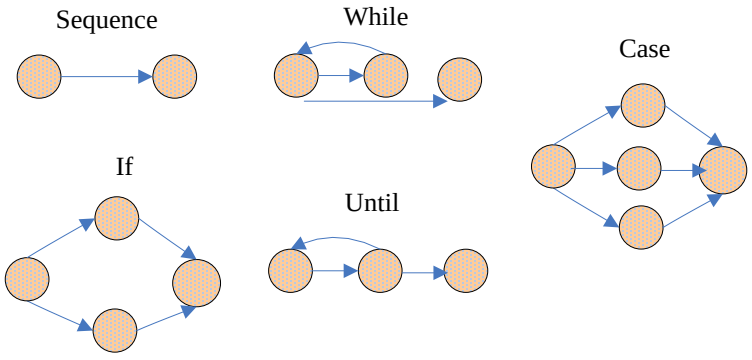


Figure 2.9 Example of simple flow graph

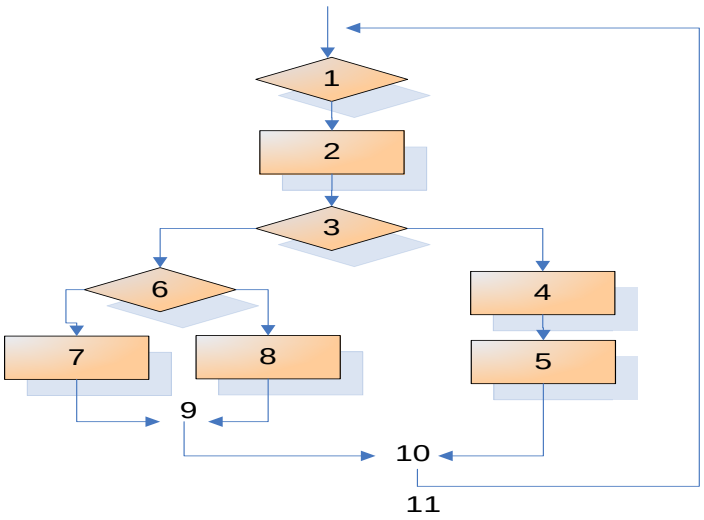


Figure 2.10 Example of flowchart

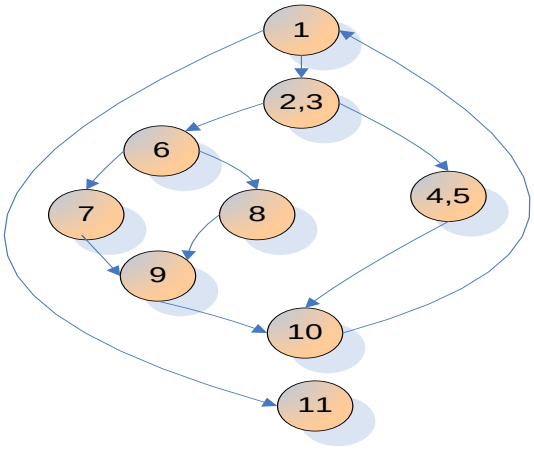


Figure 2.11 Example of flowchart change to graph

**The Independent Path include: [11]**

Path 1: 1, 11

Path 2: 1, 2, 3, 4, 5, 10, 1, 11.

Path 3: 1, 2, 3, 6, 8, 9, 10, 1, 11.

Path 4: 1, 2, 3, 6, 7, 9, 10, 1, 11.

**Cychomatic Complexity: V(G)**

A way that allows for Basis Path.

V (G) = E - V + 2. ;V: Vertices , E: Edge

**(b) Control Structure Testing:** set of internal structure testing include:

- **Decision Testing:** Test all possible decision cases in program that make sure all the paths are covered. We can use multiple conditions are if, for, while and switch involve decision test. [4]
    - Not require a decision to evaluate to false if there is no else.
    - Loop to be executed exactly once and more than once.
- **Condition Testing:** Test focus to considering each and every condition in program that possible outcome to execute a branch at least once. [4]
    - Also known as branch coverage.
    - If the state of program has multiple conditions, all conditions in deciding not need to be evaluated to T and F.
- **Data Flow Testing:** Focus to looks at how data moves in a program and fill the gaps between basis path testing and decision testing. [4]

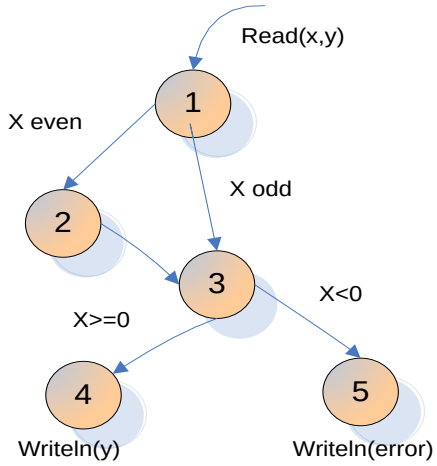


Figure 2.12 Example of data flow graph

Data flow testing used for monitoring the movement of data within the program which will occur when variables are declared , access and change the value in the work of the program . The concept of this method is the failure majority within the program are due to incorrect processing of configuration variables that occur. As in the case of incorrect input data or determine the conditions that lead to inaccuracies of the system program. The testing are point to examining the data flow within the application

is mainly to show the sequence of events associated with the presence of a primary user to ensure that:

- Every variable must be configured prior to use.
- Every variable must be configured to use at least one time.

Experiment with this method was tested techniques that depend on the values associated with the variables that affect processing directly. Experiment with this approach not only demonstrates a mechanism of application programs , but also pay attention to the way the variable is set (define) and (user) values at different internal control mechanisms . It will be necessary to determine the sequence of events associated with the presence of data and unwanted things that can happen to data. We also need to examine the impact of the introduction to each calculation every time. Comprising the following steps:

Step 1: Create control flow graph of the program being tested.

Step 2: Create a relationship between the definition and uses of variables to test coverage criteria set.

Step 3: Test case created by using the number of paths from the step 2

## 2.7 Test Case

Test case is a group of variables or conditions that were defined the system requirements. The element specifies the form input data, action or sequence of events, including the expected results to be used for testing the accuracy of the system being developed. There is also the definition of a test case for several formats include.

- A group of test data is input, processing limitations and the expected of results, which were developed for a specific purpose, like a monitoring the work of the program or checking the system requirements.
- An Input is defined to be used, and the steps that must be followed when testing software.

Test case is designed to find out what the error is not discovered earlier as soon as possible. Creating a test case can be done by checking the system requirements to create a group of input data and expected results in terms of valid and invalid type.

The goal of testing is to find faults of the program as possible by using work time to a minimum. Test case design has been developed to allow developers to use the test in a systematic way. This ensures that the testing is complete and efficient in finding errors in software correctly and quickly.

**2.7.1 Structure of test case:** Test case with official forms to be composed of three main parts as follows:

- Information: includes basic information about the test case included number, tester name, test case version, test case name, objective and test detail.
- Activity of test case: classicist an environment associated with testing activities to do before the test case and the activities to be done after the end of the test case , an action step to take while there include test data input is prepared for testing.
- Result: The resulting information includes the expected results, and actual results of operation of the system.

**2.7.2 Component of test case:** A set of actions and expected results depended on the system requirements for the test case must include the following.

- The aim of the test or the details of the system being tested.
- A description of the methods used in the test.
- Installation of other components in the tests, such as hardware, software and operating system used. As well as other installation information to meet the system requirements.
- Data as input, output, action and expected result, as well as the conditions of the test.

These components need to be referenced in conjunction with the test case at all levels of testing , such as testing each module , test by add module or user acceptance testing.

**2.7.3 Creating a test case:** Test case creation to be used for testing usually includes the following steps.

- Define the steps required to test to a minimum by using a compact and clear detail.
- The first step of creating a test case, consider the process in the each step from start to finish process.
- Generating a test case for testing the features system. Need to study thoroughly to understand the system requirements before creating test case.
- If you have installed other components are special. Make sure that those procedures are incorporated into the test case successfully.
- Define the expected results of the program.

The test case IEEE829 compliant with the following details included.

- Test case ID: Number of the test case with a unique number to identify the test sequence.
- Test case description: a brief description of the test case.
- Test prerequisite: the processing conditions tested.
- Test steps: the steps in the test system. Details are included in every step of processing the test.
- Expect result: Prediction of outcome or what users can expect from system requirements.
- Actual result: the actual results of the actions defined by input, which can be summarized as Pass / Fail, if the expected results match the actual results are not considered if pass or fail.

The example of table template which used for generating test case scan see in Table 2.6

**Table 2.6 Example of test case template**

| Test case ID | Step type | Test condition | Step name | Test data | Data type | Expect result | Actual result | Test result |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |

Hence, we used all reviewed techniques and focused on using TDD for implementation in POCT system, the black-box and white-box testing techniques were used to generate condition, input data, output data that are implemented in Chapter III.

# CHAPTER III
# MATERIALS AND METHODS

In this chapter we will explain; materials used, how we collect test data and generated the test case for two cases- with testing techniques and without testing techniques, general testing process and our methods step by step.

## 3.1 Materials

### 3.1.1 Hardware

Personal Computer

- CPU : Intel Core2 Duo
- RAM : 2 GB
- Hard Disk : 300 GB

### 3.1.2 Software

- Operation System : Windows XP
- Programming Language : Visual studio 2008 by C#
- Design : MS Visio
- Documentation : MS Word, MS Excel

## 3.2 Test Automation Framework

For our experiment, we used test framework [5] for definition scope of our work, including of four important parts: (a) test suite (set of test cases), (b) test runner (management test case execute and black box testing techniques) (c) software under test and white box testing techniques, and (d) report of test that consists of test result reply [3] see in Figure 3.1.

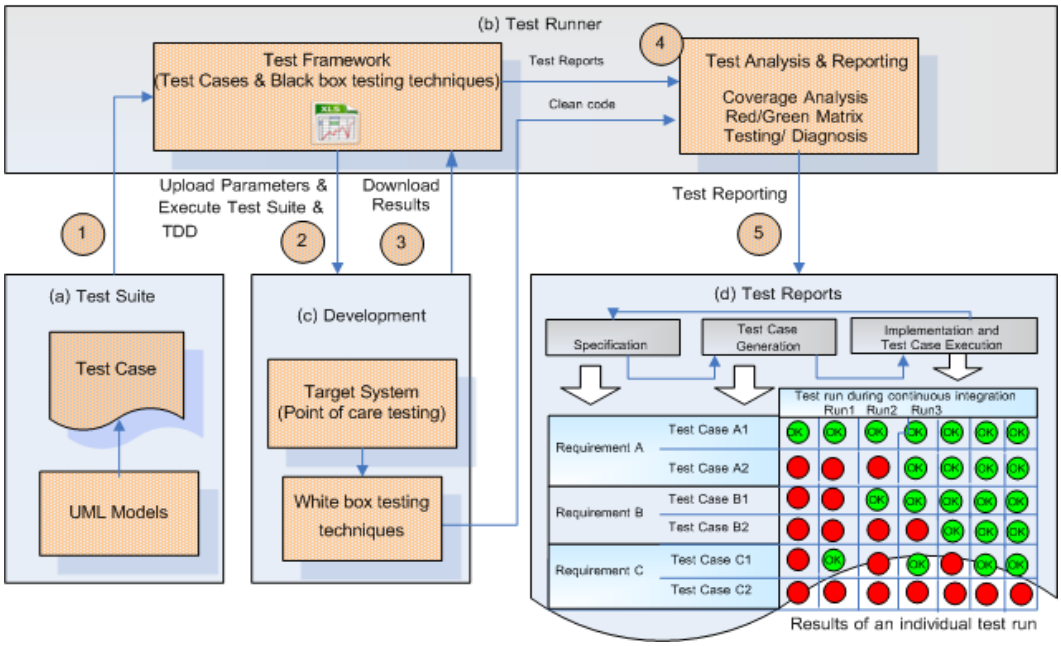


Figure 3.1 An automation test framework [3]

(a) Test suite: Based on collection of test cases that generated from requirement specifications. First we need to split requirement to smaller test cases [7] that are used for define input data , output data and test steps to test functional code and report test coverage matrix on finally.

(b) Test runner: This part we prepare all test cases and control test report management. Test cases we used create the test script for implemented functional code by TDD techniques. Test cases was built and adaption with black box testing techniques, then we will make test scripts for import parameter (input data) of each features that are enable to be automation testing [3] on a POCT system.

(c) Development: After finish created test script from test cases before implement a functional code under target system (POCT system) and adaption with white box testing techniques. Test coverage will complete when the program was executed cover all test cases based on condition and flow control.

(d) Test report: After finish development, we can summary test report that included: coverage test case matrix for check requirement cover all test cases, red-green matrix(result of test-pass/fail) and testing diagnosis data in case error that receive data from software under test cannot available in software behavior system[3].

## 3.3 Case Study

Our method implemented in the point of care testing (POCT) system as an example to create scenario test and write test cases with black box and white box testing techniques. POCT is medical diagnostic system outside the laboratory conducted close to the site of patient care by Non-laboratory clinical professionals. To help the physician and the patient to be an effective treatment, i.e., analysis serves as a distribution point, both in hospitals, clinics, and patient follow-up. This system is different from blood samples that send to a clinical laboratory. Normally the doctors, nurses or other medical personnel who perform a blood test from POCT instrument and the results are available more quickly than if the sample had been sent to the laboratory which is very important to the decision of physicians to safe critically ill patients promptly and reduce patient waiting times.

This application consists of many modules, i.e., (a) Access program (b) request management (c) request monitoring, (d) result monitoring, (e) instrument monitoring and (d) summary reports to summarize test and staff workload. We focus on 4 modules to generated test cases see in figure 3.1, the features details are as follows:

- Authentication: Management username and password.
- Request management: Management request and patient.
- Request monitoring: Monitoring to follow request completed or incomplete. System manage request by sample status; order, process, complete and approve status.
- Result monitoring: Entry result of sample and show flag alert when result value over reference range, critical range and approve result for send back to hospital information system (HIS).

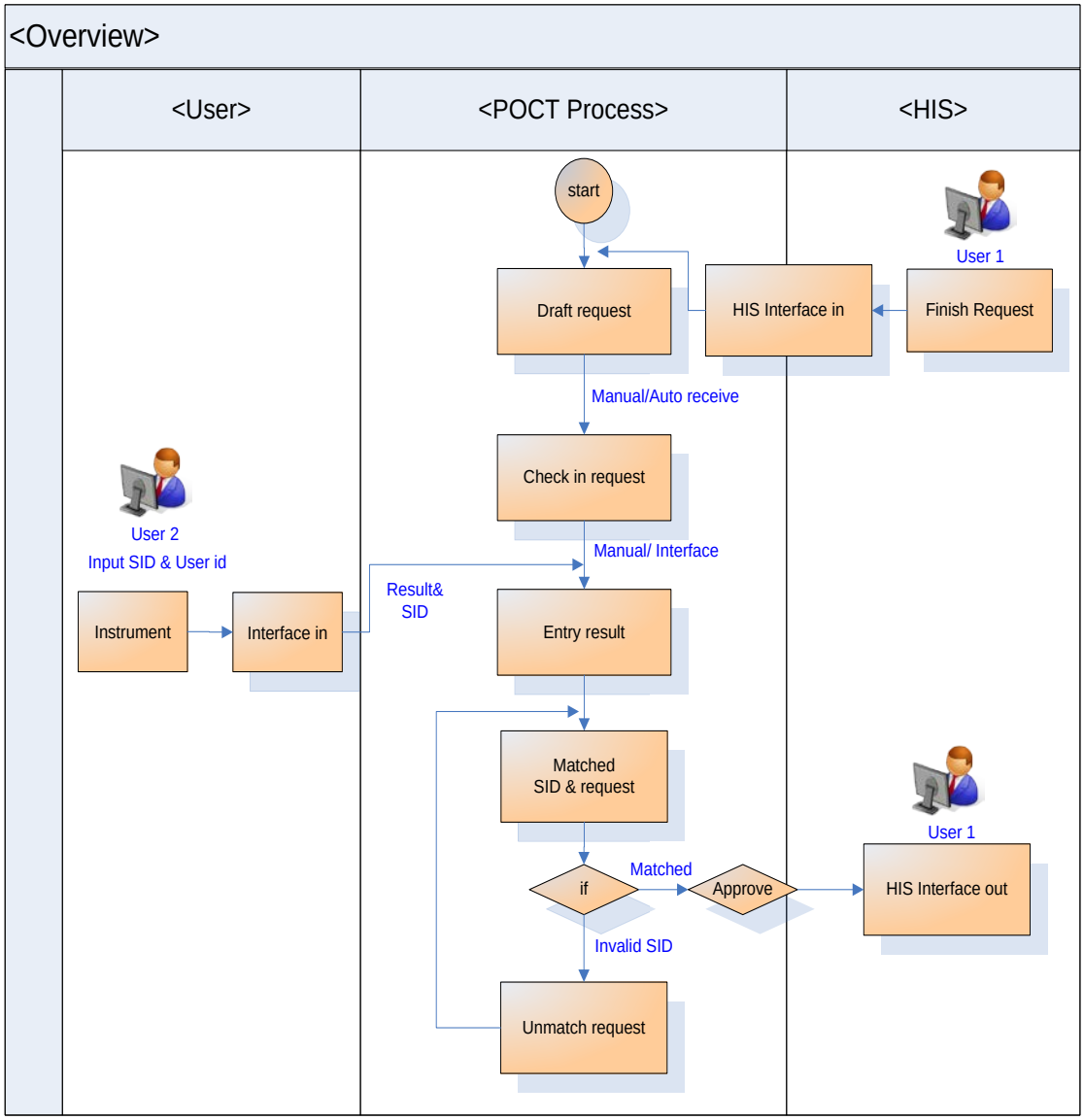


Figure 3.2 General flow of POCT system

## 3.4 Test Case Generation

Our method starting with test case generation and features that use to create test case have four main features of POCT system include as Table 3.1.

**Table 3.1 Four main features of POCT system**

| Test Case | Test Suites | TC No. | Test Case Name |
|---|---|---|---|
| **TC001** | **Authentication** | | |
| | | TC001-001 | Log in |
| | | TC001-002 | Log out |
| | | TC001-003 | Forget password (Reset password) |
| | | TC001-004 | Change password |
| **TC002** | **Request management** | TC002-001 | Add new request |
| | | TC002-002 | Edit request properties |
| | | TC002-003 | Delete request |
| | | TC002-004 | Search existing patient |
| | | TC002-005 | Add new patient |
| | | TC002-006 | Edit patient properties |
| | | TC002-007 | Delete patient |
| **TC003** | **Request Monitoring** | TC003-001 | View request (Status: Incomplete, Complete) |
| | | TC003-002 | Search |
| | | TC003-003 | Print result |
| **TC004** | **Result Monitoring** | TC004-001 | Receive sample |
| | | TC004-002 | Entry Result & Flag alert |
| | | TC004-003 | Approve result |
| | | TC004-004 | Unapprove result |

Creating a test case we split into two sections to create test case with testing techniques (black-box and white-box testing techniques) and test case generation without testing techniques. The first step we define the data dictionary and functional program (see Table 3.2, 3.3) of each module. The next step is divided test input data into two parts includes valid and in valid data for use to building the test case (see Table 3.4) and activity flow of log in module see in Figure 3.2.

### 3.4.1 TC001 Authentication module

Authentication module including functions following as login, log out, forget password (Reset password) and change password.

**3.4.1.1 Log in function** Test the login function of application. For any input other than a valid username/password pair of a non-disabled user, access will be denied.

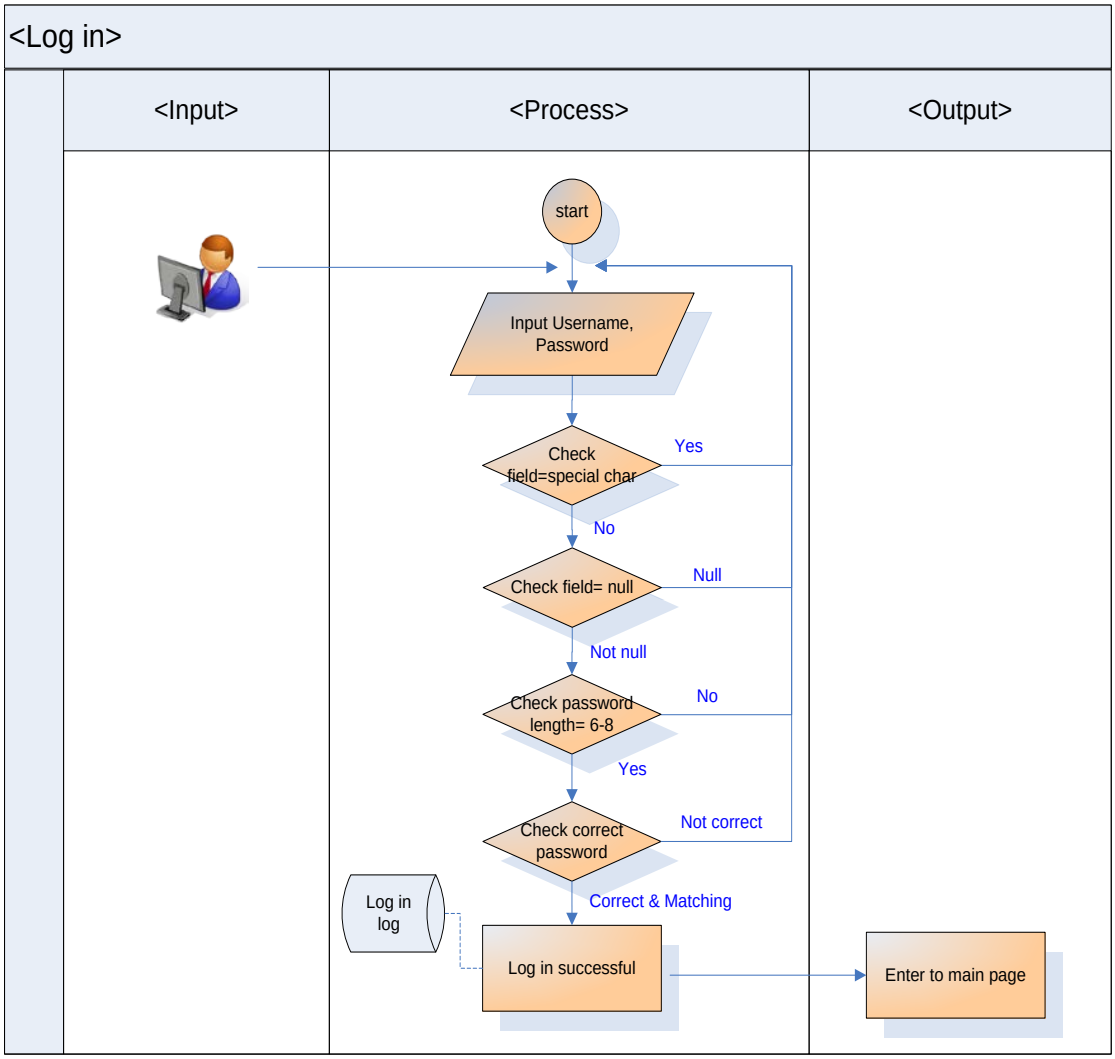


Figure 3.3 Activity flow of log in module

**Table 3.2 Data dictionary of authentication module**

| Authentication | | | | | |
|---|---|---|---|---|---|
| **No.** | **Field** | **Type** | **Length** | **Require** | **Remark** |
| 1 | pk | int | 11 | * | |
| 2 | code | varchar | 128 | * | |
| 3 | title | varchar | 128 | | |
| 4 | name | varchar | 128 | * | |
| 5 | mname | varchar | 128 | | |
| 6 | lname | varchar | 128 | | |
| 7 | id_card | varchar | 13 | * | |
| 8 | user_name | varchar | 16 | * | |
| 9 | password | varchar | 8 | * | |
| 10 | email | varchar | 128 | | |
| 11 | status | int | 11 | * | |

**Table 3.3 Functional of authentication module**

| **NO.** | **Function** |
|---|---|
| 1 | CodeStatus ValidateRequireUserAndPass(Authentication) |
| 2 | CodeStatus ValidateRequirePass(authentication) |
| 3 | Authentication CheckLogin(Authentication) |
| 4 | Authentication CheckLog out |
| 5 | CodeStatus ForgotPassword(string) |
| 6 | CodeStatus Authentication Changepassword(Authentication) |

**Table 3.4 Classify input data as valid and in valid data: Log in**

| Class No. | Description | Data Type |
|---|---|---|
| 1 | Username contain only a-z, A-Z and 0-9 | Valid |
| 2 | Username is special character | Invalid |
| 3 | Username length = 1-16 char | Valid |
| 4 | Username length > 16 char | Invalid |
| 5 | Username length > 1 char | Invalid |
| 6 | Username = blank | Invalid |
| 7 | Password = blank | Invalid |
| 8 | Password length = 6-8 char | Valid |
| 9 | Password length < 6 char | Invalid |
| 10 | Password length > 8 char | Invalid |

**Table 3.5 Test cases log in module without testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC001-001.1 | 6. Username = blank<br>8. Password length = 6-8 char | 1. Input username is empty<br>2. Input password | Username: blank<br>Password: 123456 | Test to fail | 3 |
| TC001-001.2 | 1. Username contain only a-z, A-Z and 0-9<br>4. Username length > 16 char<br>8. Password length = 6-8 char | 1. Input username length > 16 char<br>2. Input password | Username: administrator1234567890<br>Password: 123456 | Test to fail | 3 |
| TC001-001.3 | 2. Username is special char<br>8. Password length = 6-8 char | 1. Input username is special char<br>2. Input password | Username: +_)(*&^%$#@!~<br>Password: 123456 | Test to fail | 3 |
| TC001-001.4 | 1. Username contain only a-z, A-Z and 0-9<br>4. Username length =1- 16 char<br>7. Password= blank | 1. Input username<br>2. Input password is empty<br>3. Submit | Username: administrator<br>Password: blank | Test to fail | 5 |

**Table 3.5 Test cases log in module without testing techniques (cont.)**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC001-001.5 | 1. Username contain only a-z, A-Z and 0-9<br>4. Username length =1- 16 char<br>9. Password length < 6 char | 1. Input username<br>2. Input password length <6<br>3. Submit | Username: administrator<br>Password: 123 | Test to fail | 5 |
| TC001-001.6 | 1. Username contain only a-z, A-Z and 0-9<br>4. Username length =1- 16 char<br>9. Password length >8 char | 1. Input username<br>2. Input password length >8<br>3. Submit | Username: administrator<br>Password: 1234567890 | Test to fail | 3 |
| TC001-001.7 | 1. Username contain only a-z, A-Z and 0-9<br>4. Username length =1- 16 char<br>8. Password length = 6-8 char | 1. Input username<br>2. Input password<br>3. Submit | Username: administrator<br>Password: 12345678 | Test to pass | 4 |

**Table 3.6 Test cases log in module with testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC001-001.1 | 6. Username = blank<br>8. Password length = 6-8 | 1. Input username is empty<br>2. Input password= 6 | Username: blank<br>Password: 123456 | Test to fail | 3 |
| TC001-001.2 | 1. Username contain only a-z, A-Z and 0-9<br>4. Username length > 16<br>8. Password length = 6-8 | 1. Input username length =23 char<br>2. Input password = 6<br>3. Submit | Username: administrator1234567890<br>Password: 123456 | Test to fail | 3 |
| TC001-001.3 | 1. Username contain only a-z, A-Z and 0-9<br>4. Username length > 16<br>8. Password length = 6-8 | 1. Input username length =17<br>2. Input password = 6<br>3. Submit | Username: administrator1234<br>Password: 123456 | Test to fail | 3 |

**Table 3.6 Test cases log in module with testing techniques (cont.)**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC001-001.4 | 2. Username is special char<br>8. Password length = 6-8 | 1. Input username is special char<br>2. Input password = 6 | Username: +_)(*&^%$#@!~<br>Password: 123456 | Test to fail | 3 |
| TC001-001.5 | 1. Username contain only a-z, A-Z and 0-9<br>4. Username length =1- 16<br>7. Password= blank | 1. Input username<br>2. Input password is empty | Username: administrator<br>Password: blank | Test to fail | 5 |
| TC001-001.6 | 1. Username contain only a-z, A-Z and 0-9<br>4. Username length =1- 16<br>9. Password length < 6 | 1. Input username<br>2. Input password length =3 char<br>3. Submit | Username: administrator<br>Password: 123 | Test to fail | 5 |
| TC001-001.7 | 1. Username contain only a-z, A-Z and 0-9<br>4. Username length =1- 16<br>9. Password length >8 | 1. Input username<br>2. Input password length = 10 char<br>3. Submit | Username: administrator<br>Password: 1234567890 | Test to fail | 3 |
| TC001-001.8 | 6. Username = blank<br>7. Password= blank | 1. Input username<br>2. Input password<br>3. Submit | Username: blank<br>Password: blank | Test to fail | 4 |
| TC001-001.9 | 1. Username contain only a-z, A-Z and 0-9<br>4. Username length =1- 16<br>8. Password length = 6-8 | 1. Input username length = 13 char<br>2. Input password= 8<br>3. Submit | Username: administrator<br>Password: 12345678 | Test to pass | 3 |
| TC001-001.10 | 1. Username contain only a-z, A-Z and 0-9<br>3. Username length= 1-16<br>8. Password length = 6-8 | 1. Input username length = 16 char<br>2. Input password= 6<br>3. Submit | Username: administrator123<br>Password: 123456 | Test to pass | 3 |
| TC001-001.11 | 1. Username contain only a-z, A-Z and 0-9<br>3. Username length= 1-16<br>8. Password length = 6-8 | 1. Input username length = 1 char<br>2. Input password= 6<br>3. Submit | Username: a<br>Password: 123456 | Test to pass | 3 |

**3.4.1.2 Log out function** Test the logout functions of the application. On logout, access to the application is terminated and No further actions can be taken, except logging in.

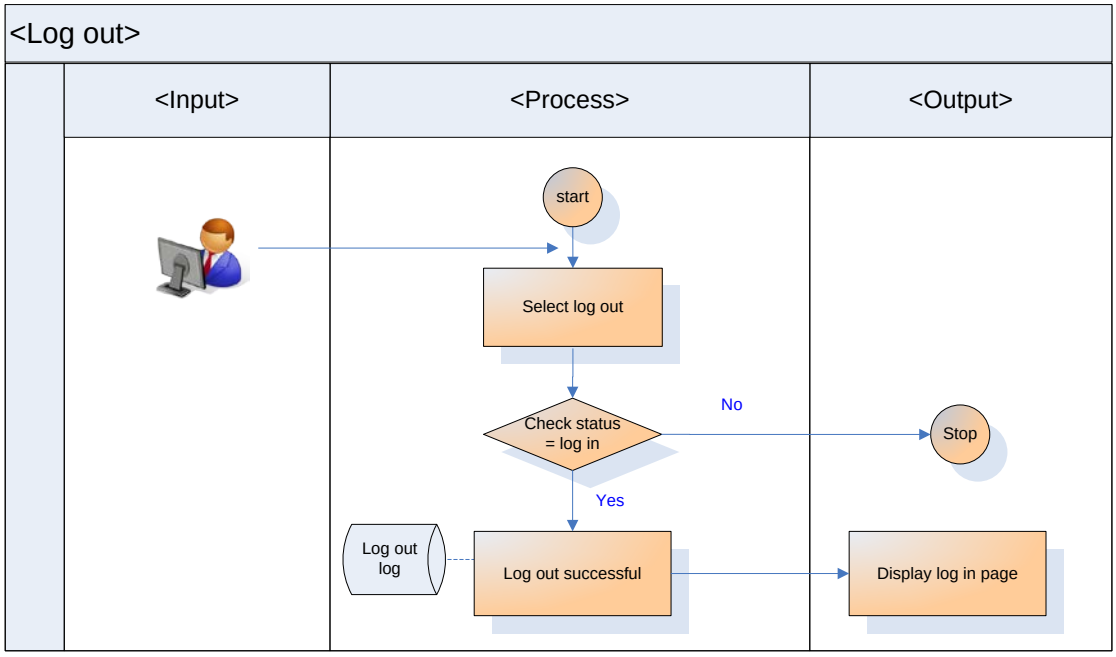


Figure 3.4 Activity flow of log out

**Table 3.7 Classify input data: Log out**

| Class No. | Description | Type |
|---|---|---|
| 1 | Store log file into the database. | Valid |
| 2 | Does Not store Log file into the database | Invalid |

**Table 3.8 Test cases: Log out without testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC001-002.1 | 2. Does Not store Log file into the database | 1. select log out<br>2. Submit | - | Test to fail | 3 |
| TC001-002.2 | 1. Store log file into the database | 1. select log out<br>2. Submit | - | Test to pass | 3 |

**Table 3.9 Test cases: Log out with testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC001-002.1 | 2. Does not store log file into the database | 1. select log out<br>2. Submit | - | Test to fail | 3 |
| TC001-002.2 | 1. Store log file into the database | 1. select log out<br>2. Submit | - | Test to pass | 3 |

**3.4.1.3 Forget password function** Tests the functions for password reset. Users are allowed to reset their own password, but for other accounts only privileged users can perform resets.

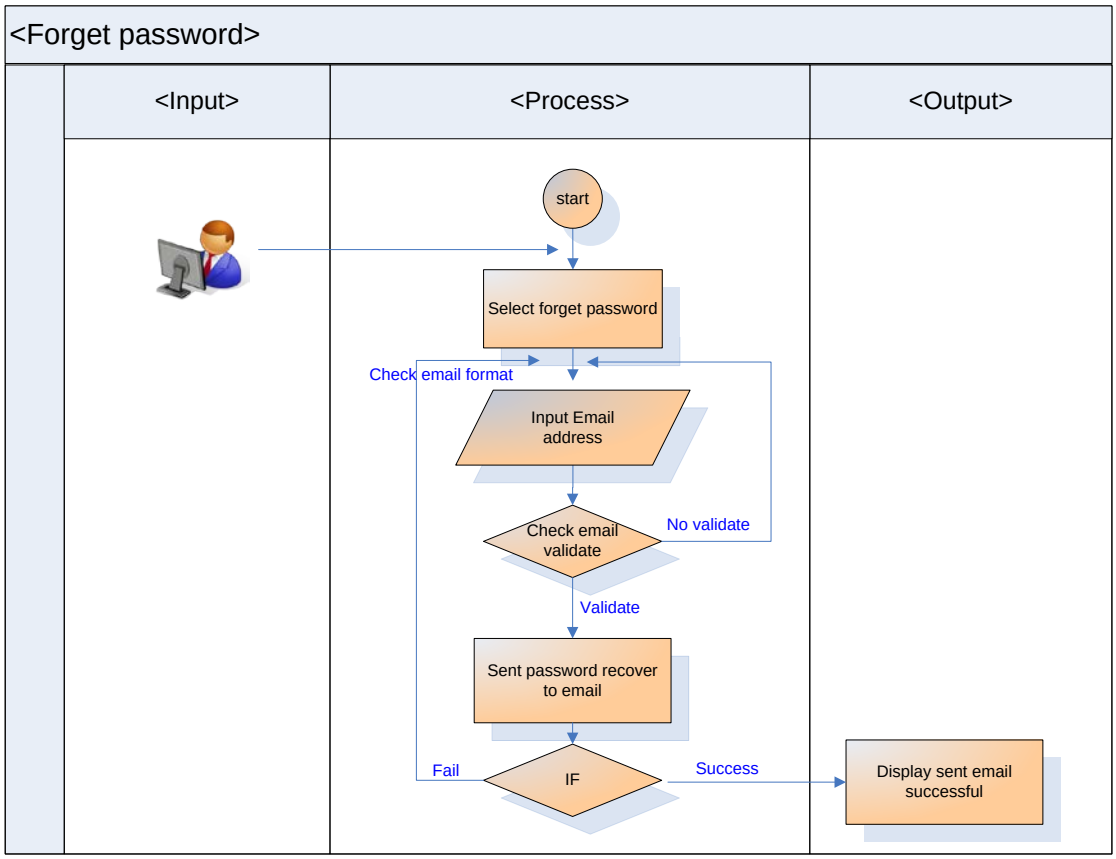


Figure 3.5 Activity flow of Forget password (Reset password)

**Table 3.10 Classify input data: Forget password (Reset password)**

| Class No. | Description | Type |
|---|---|---|
| 1 | Email contain the local part and domain name and have “@” between local part and domain name. | Valid |
| 2 | Email only allow character and numbers ( a-z , A-Z, 0-9 ) | Valid |
| 3 | Email should accept only 10 - 30 characters | Valid |
| 4 | Minimum 1 dot should be there. | Valid |
| 5 | Maximum 2 dots | Invalid |
| 6 | Email is blank | Invalid |
| 7 | Email use special symbols | Invalid |
| 8 | Email length > 10 - 30 characters | Invalid |
| 9 | Local part should Not contain any special chars. | Invalid |
| 10 | Local part is Not allowing double quote symbol. | Invalid |
| 11 | Local part is blank | Invalid |
| 12 | Domain name Not valid. | Invalid |
| 13 | Domain name is blank | Invalid |
| 14 | Domain Name contains special characters. | Invalid |
| 15 | Missing @ sign and domain name | Invalid |
| 16 | Two @ sign | Invalid |
| 17 | "@" Not between local part and domain name. | Invalid |

**Table 3.11 Test cases: Forget password without testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC001-003.1 | 6. Email is blank | 1. Select forgot password<br>2. Input email address<br>3. Submit | Email: blank | Test to fail | 3 |
| TC001-003.2 | 7. Email use special symbols | 1. Select forgot password<br>2. Input email address<br>3. Submit | Email: ^&&)) | Test to fail | 3 |
| TC001-003.3 | 8. Email length > 10 - 30 characters | 1. Select forgot password<br>2. Input email address<br>3. Submit | Email: ratchanokchait1234567 @hotmail.com | Test to fail | 3 |
| TC001-003.4 | 2. Email only allow character and numbers ( a-z , A-Z, 0-9 ) | 1. Select forgot password<br>2. Input email address<br>3. Submit | Email: ratchanok @yahoo.com | Test to pass | 3 |

**Table 3.12 Test cases: Forget password with testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC001-003.1 | 5. Maximum 2 dots | 1. Select forgot password<br>2. Input email address | Email:ratchanok @yahoo.co.in | Test to fail | 3 |
| TC001-003.2 | 6. Email is blank | 1. Select forgot password<br>2. Input email address | Email: blank | Test to fail | 3 |

**Table 3.12 Test cases: Forget password with testing techniques (cont.)**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC001-003.3 | 7. Email use special symbols | 1. Select forgot password<br>2. Input email address | Email: ^&&)) | Test to fail | 3 |
| TC001-003.4 | 8. Email length > 10 - 30 characters | 1. Select forgot password<br>2. Input email address<br>3. Submit<br>4. Alert sent fail | Email: ratchanokchait1234567 @hotmail.com | Test to fail | 3 |
| TC001-003.5 | 9. Local part should Not contain any special chars. | 1. Select forgot password<br>2. Input email address<br>3. Submit | Email: $$**%%@yahoo.com | Test to fail | 3 |
| TC001-003.6 | 10. Local part is Not allow double quote symbol. | 1. Select forgot password<br>2. Input email address<br>3. Submit | Email: “ratchanok” @yahoo.com | Test to fail | 3 |
| TC001-003.7 | 11. Local part is blank | 1. Select forgot password<br>2. Input email address<br>3. Submit | Email : @gmail.com | Test to fail | 3 |
| TC001-003.8 | 12. Domain name Not valid. | 1. Select forgot password<br>2. Input email address | Email : ratchanok @gfdgfgfsdg.gfddsg | Test to fail | 3 |
| TC001-003.9 | 13. Domain name is blank | 1. Select forgot password<br>2. Input email address | Email : ratchanok@ | Test to fail | 3 |
| TC001-003.10 | 14. Domain Name contain special characters. | 1. Select forgot password<br>2. Input email address | Email : rohan@***$$$.com | Test to fail | 3 |

**Table 3.12 Test cases: Forget password with testing techniques (cont.)**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC001-003.11 | 15. Missing @ sign and domain name | 1. Select forgot password<br>2. Input email address | Email : ratchanok | Test to fail | 3 |
| TC001-003.12 | 16. Two @ sign | 1. Select forgot password<br>2. Input email address | Email : ratchanok @gmail@gmail.com | Test to fail | 3 |
| TC001-003.13 | 17. "@" Not between local part and domain name. | 1. Select forgot password<br>2. Input email address | Email : rohan&yahoo.com | Test to fail | 3 |
| TC001-003.14 | 1. Email contain the local part and domain name and have @symbol between local part and domain name.<br>2. Email only allow character and numbers ( a-z , A-Z, 0-9 )<br>3. Email should accept only 10 - 30 characters<br>4. Minimum 1 dot should be there. | 1. Select forgot password<br>2. Input email address<br>3. Submit | Email : ratchanok @yahoo.com | Test to pass | 3 |

**3.4.1.4 Change password function** Test the functions for password change. Users are allowed to change their own password, but for other accounts only privileged users can perform changes.

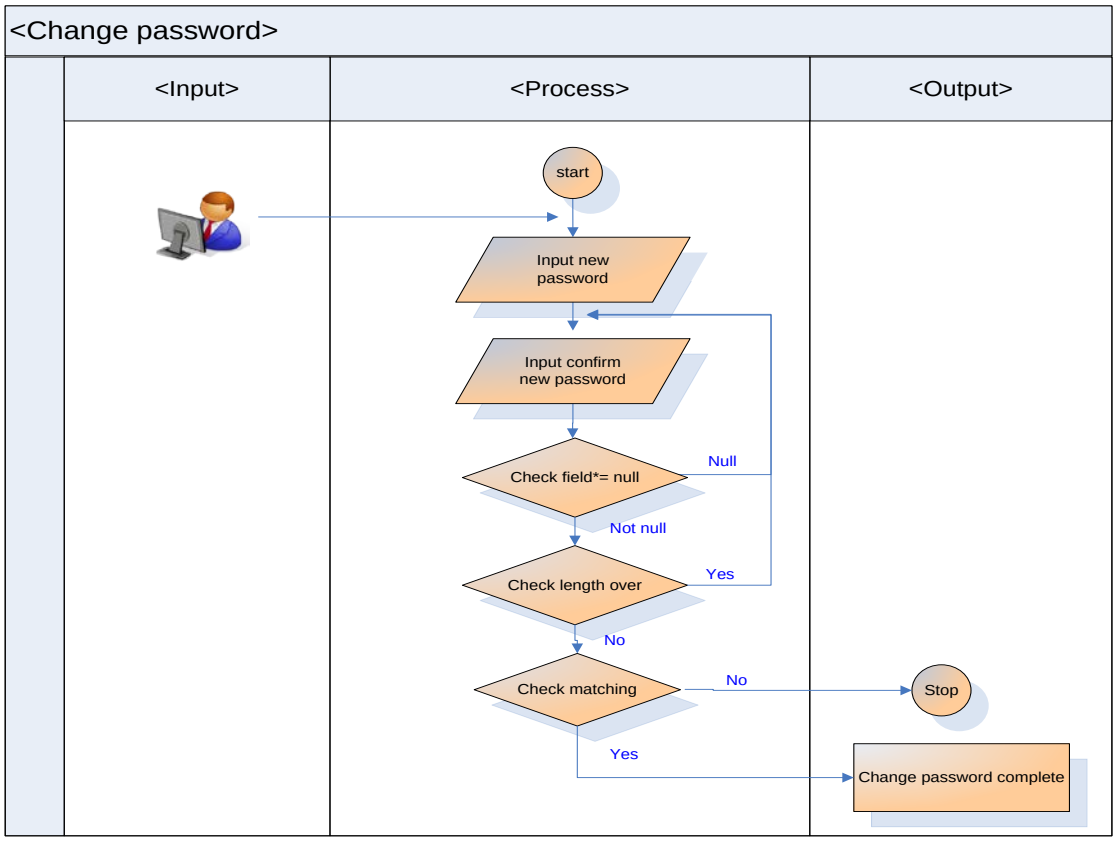


Figure 3.6 Activity flow of Change password

**Table 3.13 Classify input data: Change password**

| Class No. | Description | Data Type |
|---|---|---|
| 1 | New Password = blank | Invalid |
| 2 | New Password length < 6 char | Invalid |
| 3 | New Password length > 8 char | Invalid |
| 4 | Confirm Password = blank | Invalid |
| 5 | Confirm Password length < 6 char | Invalid |
| 6 | Confirm Password length > 8 char | Invalid |
| 7 | New password duplicate old password | Invalid |
| 8 | New password not match the confirm password. | Invalid |
| 9 | New Password length = 6-8 char | Valid |
| 10 | Confirm Password length = 6-8 char | Valid |
| 11 | New password not duplicate old password | Valid |

**Table 3.14 Test cases: Change password without testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC001-004.1 | 1. New Password = blank<br>4. Confirm Password = blank | 1. Input new password<br>2. Input confirm password<br>3. Submit | New pass= blank<br>Confirm pass= blank | Test to fail | 3 |
| TC001-004.2 | 2. New Password length < 6 char<br>10. New Password length = 6-8 char | 1. Input new password<br>2. Input confirm password<br>3. Submit | New pass= 12345<br>Confirm pass= 123456 | Test to fail | 3 |
| TC001-004.3 | 3. New Password length > 8 char<br>10. New Password length = 6-8 char | 1. Input new password<br>2. Input confirm password<br>3. Submit | New pass= 123456789<br>Confirm pass= 123456 | Test to fail | 3 |
| TC001-004.4 | 4. Confirm Password = blank<br>9. New Password length = 6-8 char | 1. Input new password<br>2. Input confirm password<br>3. Submit | New pass= 123456<br>Confirm pass= blank | Test to fail | 5 |
| TC001-004.5 | 5. Confirm Password length < 6 char<br>9. New Password length = 6-8 char | 1. Input new password<br>2. Input confirm password<br>3. Submit | New pass= 123456<br>Confirm pass= 12345 | Test to fail | 5 |
| TC001-004.6 | 6. Confirm Password length > 8 char<br>9. New Password length = 6-8 char | 1. Input new password<br>2. Input confirm password<br>3. Submit | New pass= 123456<br>Confirm pass= 123456789 | Test to fail | 3 |
| TC001-004.7 | 8. New password Not match the confirm password. | 1. Input new password<br>2. Input confirm password<br>3. Submit | New pass= 123456<br>Confirm pass= abcd | Test to fail | 4 |

**Table 3.14 Test cases: Change password without testing techniques (cont.)**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC001-004.8 | 9. New Password length = 6-8 char<br>10. Confirm Password length = 6-8 char<br>11. New password Not duplicate old password | 1. Input new password<br>2. Input confirm password<br>3. Submit | New pass= 123456<br>Confirm pass= 123456 | Test to pass | 5 |

**Table 3.15 Test cases: Change password with testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC001-004.1 | 1. New Password = blank<br>4. Confirm Password = blank | 1. Input new password<br>2. Input confirm password | New pass= blank<br>Confirm pass= blank | Test to fail | 3 |
| TC001-004.2 | 2. New Password length < 6 char<br>10. New Password length = 6-8 char | 1. Input new password<br>2. Input confirm password | New pass= 12345<br>Confirm pass= 123456 | Test to fail | 3 |
| TC001-004.3 | 3. New Password length > 8 char<br>10. New Password length = 6-8 char | 1. Input new password<br>2. Input confirm password | New pass= 123456789<br>Confirm pass= 123456 | Test to fail | 3 |
| TC001-004.4 | 4. Confirm Password = blank<br>9. New Password length = 6-8 char | 1. Input new password<br>2. Input confirm password<br>3. Submit | New pass= 123456<br>Confirm pass= blank | Test to fail | 5 |
| TC001-004.5 | 5. Confirm Password length < 6 char<br>9. New Pw length = 6-8 | 1. Input new pw<br>2. Input confirm pw | New pass= 123456<br>Confirm pass= 12345 | Test to fail | 5 |

**Table 3.15 Test cases: Change password with testing techniques (cont.)**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC001-004.6 | 6. Confirm Password length > 8 char<br>9. New Password length = 6-8 char | 1. Input new password<br>2. Input confirm password<br>3. Submit | New pass= 123456<br>Confirm pass= 123456789 | Test to fail | 3 |
| TC001-004.7 | 7. New password duplicate old password | 1. Input new password<br>2. Input confirm password | Old password= 123456<br>New pass= 123456<br>Confirm pass= 123456 | Test to fail | 3 |
| TC001-004.8 | 8. New password Not match the confirm password. | 1. Input new password<br>2. Input confirm password | New pass= 123456<br>Confirm pass= abcd | Test to fail | 4 |
| TC001-004.9 | 9. New Password length = 6-8 char<br>10. Confirm Password length = 6-8 char<br>11. New password Not duplicate old password | 1. Input new password<br>2. Input confirm password<br>3. Submit | New pass= 123456<br>Confirm pass= 123456 | Test to pass | 5 |
| TC001-004.10 | 9. New Password length = 6-8 char<br>10. Confirm Password length = 6-8 char<br>11. New password Not duplicate old password | 1. Input new password<br>2. Input confirm password<br>3. Submit | New pass= 1234567<br>Confirm pass= 1234567 | Test to pass | 3 |
| TC001-004.11 | 9. New Password length = 6-8 char<br>10. Confirm Password length = 6-8 char<br>11. New password Not duplicate old password | 1. Input new password<br>2. Input confirm password<br>3. Submit | New pass= 1234568<br>Confirm pass= 1234568 | Test to pass | 3 |

### 3.4.2 TC002 Request management module

Request management modules include request, patient, test item and test in request data for fulfill request to complete process.

**3.4.2.1 Add new request function** Testing add new request function of application. For any input as invalid data or Not input require field, progression cannot be complete.

**Table 3.16 Data dictionary of request management module: Request**

| Request | | | | | |
|---|---|---|---|---|---|
| **No.** | **Field** | **Type** | **Length** | **Require** | **Remark** |
| 1 | pk | int | 11 | * | auto number |
| 2 | sid | varchar | 32 | * | |
| 3 | hn | varchar | 16 | * | |
| 4 | title | varchar | 32 | | |
| 5 | name | varchar | 128 | * | |
| 6 | mname | varchar | 128 | * | |
| 7 | lname | varchar | 128 | * | |
| 8 | age | int | 11 | * | |
| 9 | gender | varchar | 32 | * | |
| 10 | fileType | varchar | 255 | * | |
| 11 | right | varchar | 255 | * | |
| 12 | ward_operator | varchar | 255 | * | |
| 13 | ward_requester | varchar | 255 | * | |
| 14 | doctor | varchar | 255 | * | |
| 15 | request_date | date time | | * | |
| 16 | receive_date | date time | | * | |
| 17 | approve_date | date time | | * | |
| 18 | comment_request | varchar | 255 | | |
| 19 | status | int | 11 | * | 1:Process<br>2:Receive<br>3:Approve<br>4:Reject |

**Table 3.17 Functional of request management module: Request**

| No. | Function | Remark |
|---|---|---|
| 1 | class CodeStatus ValidateRequireField(class Request) | Check Require Field |
| 2 | string CreateSID() | Create SID |
| 3 | int CalcAge(class Patient) | Calculate Age rounded to integer |
| 4 | class RequestStatus CheckReuqstStatus(class Request) | Check current status of order |
| 5 | class Request CreateRequest(class Request) | |
| 6 | class CodeStatus ReceiveRequest(class Request) | |
| 7 | class CodeStatus ApproveRequest(class Request) | |
| 8 | class CodeStatus RejectRequest(class Request) | |
| 9 | List<class Request> GetRequest(class Request) | check from Pk |
| 10 | List<class Request> FindRequest(class Request) | check from hn, name, mname, lname, SID, operator, requester, requestDate, status |
| 11 | List<class Request> FindAllRequest() | Select All Request |

**Table 3.18 Functional of request management module: Patient**

| Patient | | | | | |
|---|---|---|---|---|---|
| **No.** | **Field** | **Type** | **Length** | **Require** | **Remark** |
| 1 | pk | int | 11 | * | auto NO. |
| 2 | sid | varchar | 11 | * | |
| 3 | hn | varchar | 16 | * | |
| 4 | title | varchar | 32 | | |
| 5 | name | varchar | 128 | * | |
| 6 | mname | varchar | 128 | | |
| 7 | lname | varchar | 128 | * | |

**Table 3.18 Functional of request management module: Patient (cont.)**

| Patient | | | | | |
|---|---|---|---|---|---|
| **No.** | **Field** | **Type** | **Length** | **Require** | **Remark** |
| 8 | id_card | varchar | 13 | | |
| 9 | telephone | varchar | 128 | | |
| 10 | address | varchar | 255 | | |
| 11 | contact | varchar | 255 | | |
| 12 | birth date | datetime | | | |
| 13 | gender | varchar | 32 | | |
| 14 | lastUpdate | Datetime | | | |
| 15 | Status | Int | 11 | * | 1:Process<br>2:Receive<br>3:Approve<br>4:Reject |

**Table 3.19 Functional of request management module: Patient**

| **No.** | **Function** |
|---|---|
| 1 | class CodeStatus ValidateRequireField(class Request) |
| 2 | string CreateSID() |
| 3 | int CalcAge(class Patient) |
| 4 | class RequestStatus CheckReuqstStatus(class Request) |
| 5 | class Request CreateRequest(class Request) |
| 6 | class CodeStatus ReceiveRequest(class Request) |
| 7 | class CodeStatus ApproveRequest(class Request) |

**Table 3.20 Functional of request management module: TestItem**

| TestItem | | | | | |
|---|---|---|---|---|---|
| **No.** | **Field** | **Type** | **Length** | **Require** | **Remark** |
| 1 | pk | int | 11 | * | auto number |
| 2 | code | varchar | 16 | * | |
| 3 | name | varchar | 128 | * | |
| 4 | display_range | varchar | 128 | * | |
| 5 | unit | nvarchar | 128 | * | |
| 6 | price | double | 11 | * | |
| 7 | Normal_low | double | 11 | * | |
| 8 | Normal_high | double | 11 | * | |
| 9 | critical_low | double | 11 | * | |
| 10 | critical_high | double | 11 | * | |

**Table 3.21 Functional of request management module: TestItem**

| **No.** | **Function** | **Remark** |
|---|---|---|
| 1 | bool CheckDuplicateCode(class TestItem) | Check Test Code for Not duplicated |
| 2 | class CodeStatus ValidateRequireField(class TestItem) | Check Require Field |
| 3 | class CodeStatus CreateTestItem(class TestItem) | |
| 4 | class CodeStatus EditTestItem(class TestItem) | |
| 5 | class CodeStatus DeleteTestItem(class TestItem) | |
| 6 | class TestItem GetTestItem(class TestItem) | Check from pk |
| 7 | List<class TestItem> FineTestItem(class TestItem) | Check from code, name |
| 8 | List<class TestItem> FineAllTestItem() | Select All Test Item |

**Table 3.22 Functional of request management module: Test in request**

| Test in request | | | | | |
|---|---|---|---|---|---|
| **No.** | **Field** | **Type** | **Length** | **Require** | **Remark** |
| 1 | pk | int | 11 | * | auto number |
| 2 | request_pk | int | 11 | * | |
| 3 | code | varchar | 16 | * | |
| 4 | name | varchar | 128 | * | |
| 5 | display_rang | varchar | 128 | * | |
| 6 | unit | nvarchar | 128 | * | |
| 7 | price | double | 11 | * | |
| 8 | Normal_low | double | 11 | * | |
| 9 | Normal_high | double | 11 | * | |
| 10 | critical_low | double | 11 | * | |
| 11 | critical_high | double | 11 | * | |
| 12 | delta check | double | 11 | * | |
| 13 | resutl_value | double | 11 | | |
| 14 | result_flag | varchar | 32 | | |

**Table 3.23 Functional of request management module: Test in request**

| **No.** | **Function** |
|---|---|
| 1 | class CodeStatus ValidateRequireField(class TestInRequest) |
| 2 | string CalcFlag(class TestInRequest) |
| 3 | class CodeStatus CreateTestInRequest(class TestInRequest) |
| 4 | class CodeStatus SaveResultAndFlag(class TestInRequest) |
| 5 | List<class TestInRequest> FindTestInRequest(class Request) |

**3.4.2.1 Add new request function** Testing add new request function of application. For any input as invalid data or Not input require field, progression cannot be complete see Fig 3.7.

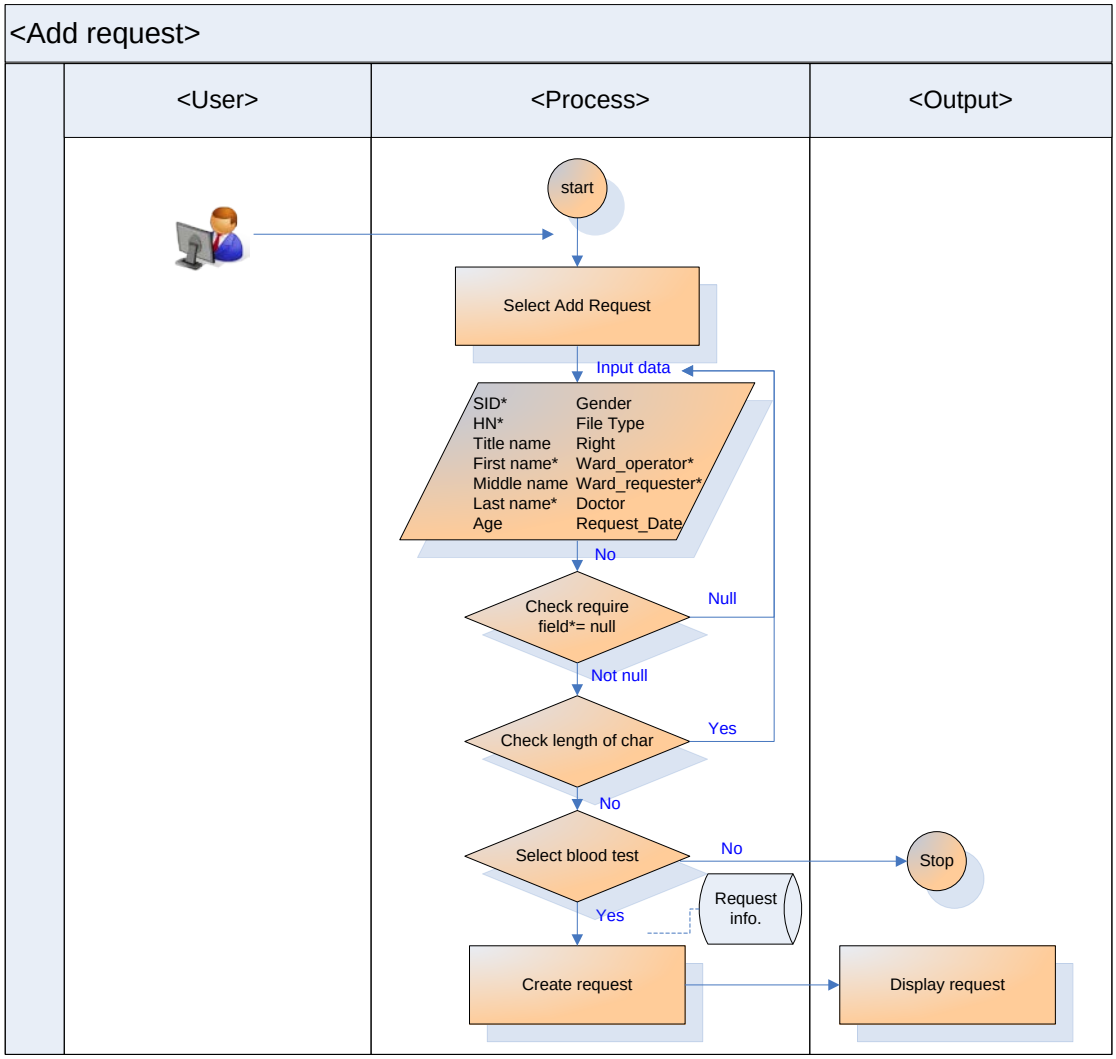


Figure 3.7 Activity flow of Add request

**Table 3.24 Classify input data: Add new request**

| Class No. | Description | Data Type |
|---|---|---|
| 1 | Create SID Not complete | Invalid |
| 2 | HN = blank | Invalid |
| 3 | First name = blank | Invalid |
| 4 | Last name = blank | Invalid |
| 5 | Ward_operator = blank | Invalid |
| 6 | Ward_requester = blank | Invalid |
| 7 | SID length > 11 | Invalid |

**Table 3.24 Classify input data: Add new request (cont.)**

| Class No. | Description | Data Type |
|---|---|---|
| 8 | SID length < 1 | Invalid |
| 9 | HN length > 16 | Invalid |
| 10 | HN length < 1 | Invalid |
| 11 | First name length > 128 | Invalid |
| 12 | First name length <1 | Invalid |
| 13 | Last name length > 128 | Invalid |
| 14 | Last name length <1 | Invalid |
| 15 | Title name length > 32 | Invalid |
| 16 | Title name length <1 | Invalid |
| 17 | Middle name length > 128 | Invalid |
| 18 | Middle name length <1 | Invalid |
| 19 | Age <0 | Invalid |
| 20 | Request status <=0 | Invalid |
| 21 | Test in request <=0 | Invalid |
| 22 | SID length = 1-11 | Valid |
| 23 | HN length = 1-16 | Valid |
| 24 | First name length = 1-128 | Valid |
| 25 | Last name length = 1-128 | Valid |
| 25 | Title name length = 1-32 | Valid |
| 27 | Middle name length = 1-128 | Valid |
| 28 | Test in request >=1 | Valid |
| 29 | Create SID complete | Valid |
| 30 | Request status <=0 | Valid |

**Table 3.25 Test cases: Add new request without testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC002-001.1 | 21. Create SID complete | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | SID= 570000001 | Test to pass | 3 |
| TC002-001.2 | 7. SID length > 11 | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | SID= 5700000000001 | Test to fail | 3 |
| TC002-001.3 | 2. HN = blank | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | HN= blank | Test to fail | 3 |
| TC002-001.4 | 9. HN length > 16 | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | HN= 12345678901234567890 | Test to fail | 3 |
| TC002-001.5 | 3. First name = blank | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | First name = blank | Test to fail | 3 |
| TC002-001.6 | 11. First name length > 128 | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | First name = 1234567890…..> 128 chars | Test to fail | 3 |
| TC002-001.7 | 4. Last name = blank | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | Last name = blank | Test to fail | 3 |

**Table 3.25 Test cases: Add new request without testing techniques (cont.)**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC002-001.8 | 13. Last name length > 128 | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | Last name= Chaipraserth …..> 128 chars | Test to fail | 3 |
| TC002-001.9 | 17. Middle name length > 128 | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | Middle name= Chaiii…..> 128 chars | Test to fail | 3 |
| TC002-001.10 | 19. Age <0 | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | Age <0 | Test to fail | 3 |
| TC002-001.11 | 20. Request status <=0 | 1. Select add request<br>2. Input data<br>3. select test | Request status <=0 | Test to fail | 3 |
| TC002-001.12 | 21. Test in request <=0 | 1. Select add request<br>2. Input data<br>3. select test | Test in request <=0 | Test to fail | 3 |
| TC002-001.13 | 22. SID length = 1-32<br>23. HN length = 1-16<br>24. First name length = 1-128<br>25. Last name length = 1-128<br>26. Title name length = 1-32<br>27. Middle name length = 1-128<br>28. Test in request >=1<br>30. Request status <=0 | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | SID length = 570000001<br>HN length = 1234567<br>First name length = Ratchanok<br>Last name length =Chaipraserth<br>Title name length = Ms.<br>Middle name length = Sue<br>Test in request >=1<br>Request status <=0 | Test to pass | 3 |

**Table 3.26 Test cases: Add new request with testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC002-001.1 | 21. Create SID complete | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | SID= 570000001 | Test to pass | 3 |
| TC002-001.2 | 1. Create SID Not complete | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | SID= XXXXXXX | Test to fail | 3 |
| TC002-001.3 | 7. SID length > 11 | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | SID= 5700000000001 | Test to fail | 3 |
| TC002-001.4 | 10. SID length < 1 | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | SID= invalid data | Test to fail | 3 |
| TC002-001.5 | 2. HN = blank | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | HN= blank | Test to fail | 3 |
| TC002-001.6 | 9. HN length > 16 | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | HN= 12345678901234567890 | Test to fail | 3 |
| TC002-001.7 | 10. HN length <1 | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | HN= invalid data | Test to fail | 3 |

**Table 3.26 Test cases: Add new request with testing techniques (cont.)**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC002-001.8 | 3. First name = blank | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | First name = blank | Test to fail | 3 |
| TC002-001.9 | 11. First name length > 128 | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | First name = 1234567890…..> 128 chars | Test to fail | 3 |
| TC002-001.10 | 12. First name length <1 | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | First name = invalid data | Test to fail | 3 |
| TC002-001.11 | 4. Last name = blank | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | Last name = blank | Test to fail | 3 |
| TC002-001.12 | 13. Last name length > 128 | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | Last name = Chaipraserth…..> 128 chars | Test to fail | 3 |
| TC002-001.13 | 14. Last name length <1 | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | Last name = invalid data | Test to fail | 3 |
| TC002-001.14 | 17. Middle name length > 128 | 1. Select add request<br>2. Input data<br>3. select test | Middle name= Chaipraserth…..> 128 chars | Test to fail | 3 |

**Table 3.26 Test cases: Add new request with testing techniques (cont.)**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC002-001.15 | 18. Middle name length <1 | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | Middle name= invalid data | Test to fail | 3 |
| TC002-001.16 | 19. Age <0 | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | Age <0 | Test to fail | 3 |
| TC002-001.17 | 20. Request status <=0 | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | Request status <=0 | Test to fail | 3 |
| TC002-001.18 | 21. Test in request <=0 | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | Test in request <=0 | Test to fail | 3 |
| TC002-001.19 | 22. SID length = 1-32<br>23. HN length = 1-16<br>24. First name length = 1-128<br>25. Last name length = 1-128<br>26. Title name length = 1-32<br>27. Middle name length = 1-128<br>28. Test in request >=1<br>30. Request status <=0 | 1. Select add request<br>2. Input data<br>3. select test<br>4. Submit | SID length = 570000001<br>HN length = 1234567<br>First name length = Ratchanok<br>Last name length =Chaipraserth<br>Title name length = Ms.<br>Middle name length = Sue<br>Test in request >=1<br>Request status <=0 | Test to pass | 3 |

**3.4.2.2 Edit request properties function** Testing add new request function of application. For any input as invalid data or not input require field, progression cannot be complete. Classify input data same the function of add new request.

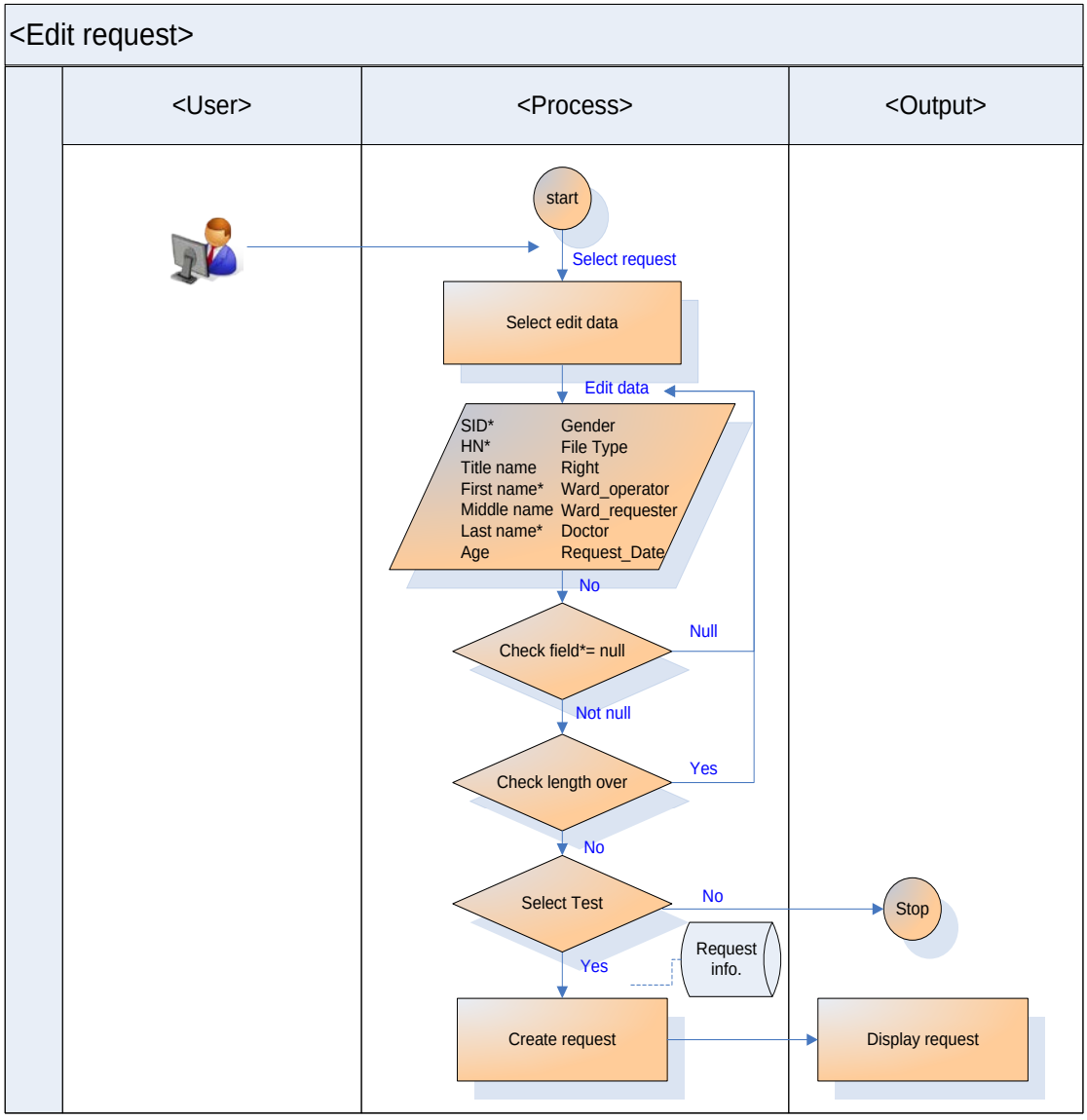


Figure 3.8 Activity flow of Edit request

**Table 3.27 Test cases: Edit request properties without testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC002-002.1 | 21. Create SID complete | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | SID= 570000001 | Test to pass | 3 |
| TC002-002.2 | 7. SID length > 11 | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | SID= 5700000000001 | Test to fail | 3 |
| TC002-002..3 | 2. HN = blank | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | HN= blank | Test to fail | 3 |
| TC002-002..4 | 9. HN length > 16 | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | HN= 12345678901234567890 | Test to fail | 3 |
| TC002-002.5 | 3. First name = blank | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | First name = blank | Test to fail | 3 |
| TC002-002.6 | 11. First name length > 128 | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | First name = 1234567890…..> 128 chars | Test to fail | 3 |
| TC002-002.7 | 4. Last name = blank | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | Last name = blank | Test to fail | 3 |
| TC002-002.8 | 13. Last name length > 128 | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | | Test to fail | 3 |

**Table 3.27 Test cases: Edit request properties without testing techniques (cont.)**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC002-002.9 | 17. Middle name length > 128 | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | Middle name= 1234567890…..> 128 chars | Test to fail | 3 |
| TC002-002.10 | 19. Age <0 | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | Age <0 | Test to fail | 3 |
| TC002-002.11 | 20. Request status <=0 | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | Request status <=0 | Test to fail | 3 |
| TC002-002.12 | 21. Test in request <=0 | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | Test in request <=0 | Test to fail | 3 |
| TC002-002.13 | 22. SID length = 1-32<br>23. HN length = 1-16<br>24. First name length = 1-128<br>25. Last name length = 1-128<br>26. Title name length = 1-32<br>27. Middle name length = 1-128<br>28. Test in request >=1<br>30. Request status <=0 | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | SID length = 570000001<br>HN length = 1234567<br>First name length = RatchaNok<br>Last name length =Chaipraserth<br>Title name length = Ms.<br>Middle name length = Sue<br>Test in request >=1<br>Request status <=0 | Test to pass | 3 |

**Table 3.28 Test cases: Edit request properties with testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC002-002.1 | 21. Create SID complete | 1. Select request<br>2. Select edit<br>3. Update data | SID= 570000001 | Test to pass | 3 |
| TC002-002.2 | 1. Create SID Not complete | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | SID= XXXXXXX | Test to fail | 3 |
| TC002-002.3 | 7. SID length > 11 | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | SID= 5700000000001 | Test to fail | 3 |
| TC002-002.4 | 10. SID length < 1 | 1. Select request<br>2. Select edit<br>3. Update data | SID= invalid data | Test to fail | 3 |
| TC002-002.5 | 2. HN = blank | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | HN= blank | Test to fail | 3 |
| TC002-002.6 | 9. HN length > 16 | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | HN= 12345678901234567890 | Test to fail | 3 |
| TC002-002.7 | 10. HN length <1 | 1. Select request<br>2. Select edit<br>3. Update data | HN= invalid data | Test to fail | 3 |
| TC002-002.8 | 3. First name = blank | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | First name = blank | Test to fail | 3 |
| TC002-002.9 | 11. First name length > 128 | 1. Select request<br>2. Select edit<br>3. Update data | First name = 1234567890…..> 128 chars | Test to fail | 3 |
| TC002-002.10 | 12. First name length <1 | 1. Select request<br>2. Select edit<br>3. Update data | First name = invalid data | Test to fail | 3 |

**Table 3.28 Test cases: Edit request properties with testing techniques (cont.)**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC002-002.11 | 4. Last name = blank | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | Last name = blank | Test to fail | 3 |
| TC002-002.12 | 13. Last name length > 128 | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | Last name = Chaipraserth…..> 128 chars | Test to fail | 3 |
| TC002-002.13 | 14. Last name length <1 | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | Last name = invalid data | Test to fail | 3 |
| TC002-002.14 | 17. Middle name length > 128 | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | Middle name= Chaipraserth…..> 128 chars | Test to fail | 3 |
| TC002-002.15 | 18. Middle name length <1 | 1. Select request<br>2. Select edit<br>3. Update data | Middle name= invalid data | Test to fail | 3 |
| TC002-002.16 | 19. Age <0 | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | Age <0 | Test to fail | 3 |
| TC002-002.17 | 20. Request status <=0 | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | Request status <=0 | Test to fail | 3 |
| TC002-002.18 | 21. Test in request <=0 | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | Test in request <=0 | Test to fail | 3 |

**Table 3.28 Test cases: Edit request properties with testing techniques (cont.)**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC002-002.19 | 22. SID length = 1-32<br>23. HN length = 1-16<br>24. First name length = 1-128<br>25. Last name length = 1-128<br>26. Title name length = 1-32<br>27. Middle name length = 1-128<br>28. Test in request >=1<br>30. Request status <=0 | 1. Select request<br>2. Select edit<br>3. Update data<br>4. Submit | SID length = 570000001<br>HN length = 1234567<br>First name length = Ratchanok<br>Last name length =Chaipraserth<br>Title name length = Ms.<br>Middle name length = Sue<br>Test in request >=1<br>Request status <=0 | Test to pass | 3 |

### 3.4.2.3 Delete request function

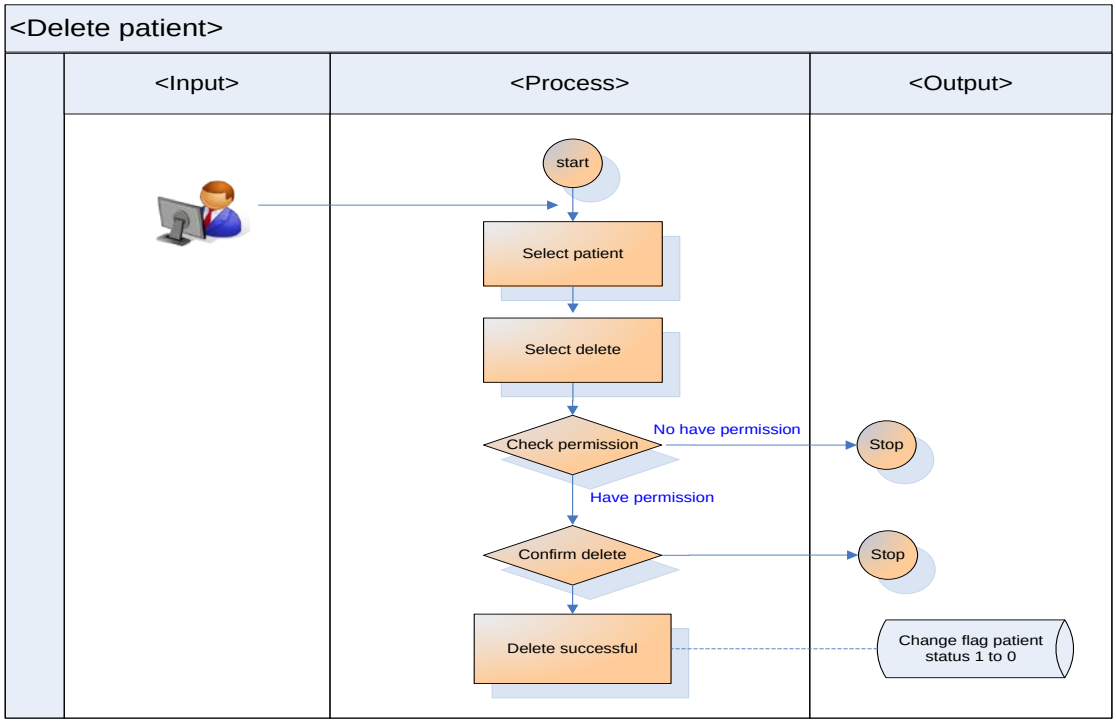


Figure 3.9 Activity flow of Delete request

**Table 3.29 Classify input data: Delete request**

| Class No. | Description | Data Type |
|---|---|---|
| 1 | Permission access= No | Invalid |
| 2 | Delete request Not complete | Invalid |
| 3 | Permission access= Yes | Valid |
| 4 | Delete request complete | Valid |

**Table 3.30 Test cases: Delete request without testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC002-003.1 | 1. Permission access= No | 1. Select request<br>2. Select delete | N/A | Test to fail | 5 |
| TC002-003.2 | 2. Delete request Not complete | 1. Select request<br>2. Select delete<br>3. Confirm delete<br>4. Submit | N/A | Test to fail | 3 |
| TC002-003.3 | 3. Permission access= No<br>4. Delete request Not complete | 1. Select request<br>2. Select delete<br>3. Confirm delete<br>4. Submit | N/A | Test to pass | 3 |

**Table 3.31 Test cases: Delete request without testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC002-003.1 | 1. Permission access= No | 1. Select request<br>2. Select delete | N/A | Test to fail | 5 |
| TC002-003.2 | 2. Delete request Not complete | 1. Select request<br>2. Select delete<br>3. Confirm delete<br>4. Submit | N/A | Test to fail | 3 |
| TC002-003.3 | 3. Permission access= No<br>4. Delete request Not complete | 1. Select request<br>2. Select delete<br>3. Confirm delete | N/A | Test to pass | 3 |

### 3.4.2.4 Search existing patient function

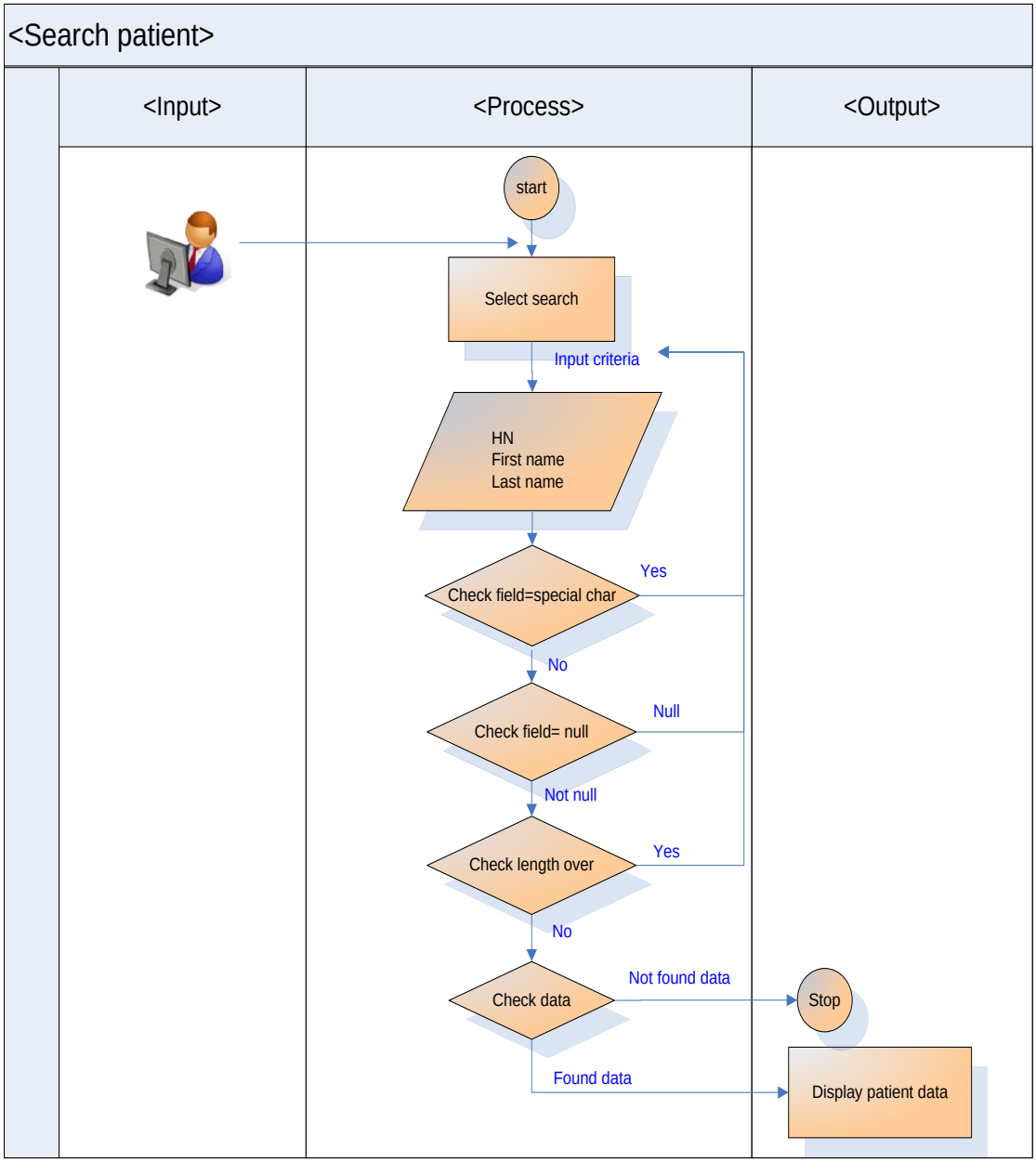


Figure 3.10 Activity flow of search existing patient

**Table 3.32 Classify input data: Search existing patient**

| Class No. | Description | Data Type |
|---|---|---|
| 1 | HN = blank | Invalid |
| 2 | First name = blank | Invalid |
| 3 | Last name = blank | Invalid |
| 4 | HN length > 16 | Invalid |
| 5 | First name length > 128 | Invalid |
| 6 | Last name length > 128 | Invalid |
| 7 | SID length = 1-11 | Valid |
| 8 | HN length = 1-16 | Valid |
| 9 | First name length = 1-128 | Valid |
| 10 | Last name length = 1-128 | Valid |

**Table 3.33 Test cases: Search existing patient without testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC002-004.1 | 1. HN = blank | 1. Input criteria<br>2. Submit | HN = blank | Test to pass | 3 |
| TC002-004.2 | 2. First name = blank | 1. Input criteria<br>2. Submit | First name = blank | Test to fail | 3 |
| TC002-004.3 | 3. Last name = blank | 1. Input criteria<br>2. Submit | Last name = blank | Test to fail | 3 |
| TC002-004.4 | 7. SID length = 1-11<br>8. HN length = 1-16<br>9. First name length = 1-128<br>10. Last name length = 1-128 | 1. Input criteria<br>2. Submit | HN = 57000001 | Test to pass | 3 |

**Table 3.34 Test cases: Search existing patient with testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC002-004.1 | 1. HN = blank | 1. Input criteria<br>2. Submit | HN = blank | Test to pass | 3 |
| TC002-004.2 | 2. First name = blank | 1. Input criteria<br>2. Submit | First name = blank | Test to fail | 3 |
| TC002-004.3 | 3. Last name = blank | 1. Input criteria<br>2. Submit | Last name = blank | Test to fail | 3 |
| TC002-004.4 | 4. HN length > 16 | 1. Input criteria<br>2. Submit | HN=57000000000000001 | Test to fail | 3 |
| TC002-004.5 | 5. First name length > 128 | 1. Input criteria<br>2. Submit | First name= Ratchanok….> 128 chars | Test to fail | 3 |
| TC002-004.6 | 6. Last name length > 128 | 1. Input criteria<br>2. Submit | Last name= Chaipraserth…> > 128 chars | Test to fail | 3 |
| TC002-004.7 | 7. SID length = 1-11<br>8. HN length = 1-16<br>9. First name length = 1-128<br>10. Last name length = 1-128 | 1. Input criteria<br>2. Submit | HN = 57000001 | Test to pass | 5 |

**3.4.2.5 Add new patient function** Testing add new patient function of application. For any input as invalid data or not input require field, progression cannot be complete.

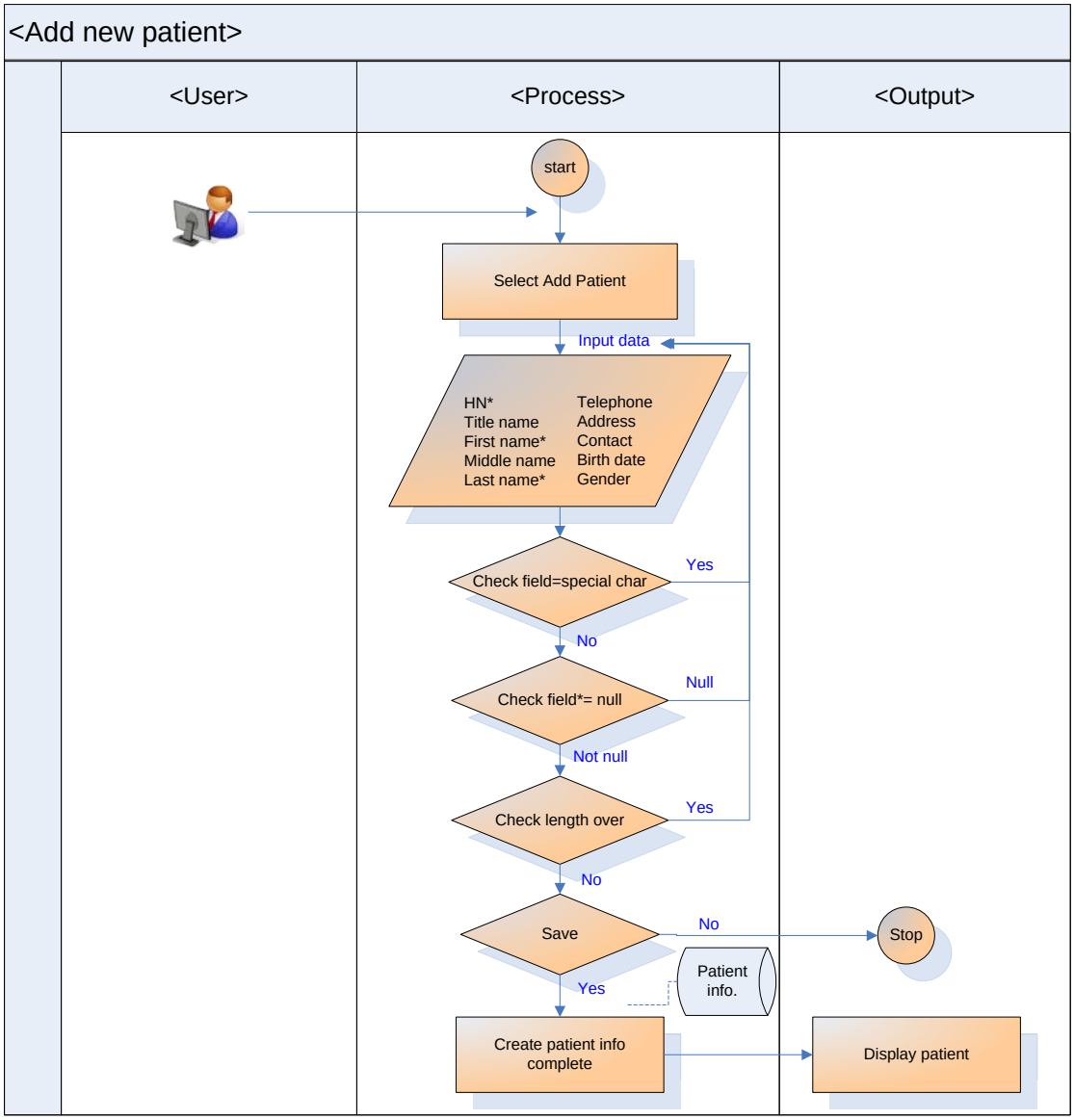


Figure 3.11 Activity flow of Add new patient

**Table 3.35 Classify input data: Add new patient**

| Class No. | Description | Data Type |
|---|---|---|
| 1 | HN = blank | Invalid |
| 2 | First name = blank | Invalid |
| 3 | Last name = blank | Invalid |
| 4 | HN length > 16 | Invalid |
| 5 | First name length > 128 | Invalid |

**Table 3.35 Classify input data: Add new patient (cont.)**

| Class No. | Description | Data Type |
|---|---|---|
| 6 | Last name length > 128 | Invalid |
| 7 | Last name length <1 | Invalid |
| 8 | HN length = 1-16 | Valid |
| 9 | First name length = 1-128 | Valid |
| 10 | Last name length = 1-128 | Valid |

**Table 3.36 Test cases: Add new patient without testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC002-005.1 | 1. HN = blank | 1. Select new patient<br>2. Input data<br>3. Submit | HN = blank | Test to pass | 3 |
| TC002-005.2 | 2. First name = blank | 1. Select new patient<br>2. Input data<br>3. Submit | First name = blank | Test to fail | 3 |
| TC002-005.3 | 3. Last name = blank | 1. Select new patient<br>2. Input data<br>3. Submit | Last name = blank | Test to fail | 3 |
| TC002-005.4 | 7. SID length = 1-11<br>8. HN length = 1-16<br>9. First name length = 1-128<br>10. Last name length = 1-128 | 1. Select new patient<br>2. Input data<br>3. Submit | HN = 57000001 | Test to pass | 5 |

**Table 3.37 Test cases: Add new patient with testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC002-005.1 | 1. HN = blank | 1. Select new patient<br>2. Input data<br>3. Submit | HN = blank | Test to pass | 3 |
| TC002-005.2 | 2. First name = blank | 1. Select new patient<br>2. Input data<br>3. Submit | First name = blank | Test to fail | 3 |
| TC002-005.3 | 3. Last name = blank | 1. Select new patient<br>2. Input data<br>3. Submit | Last name = blank | Test to fail | 3 |
| TC002-005.4 | 4. HN length > 16 | 1. Select new patient<br>2. Input data<br>3. Submit | HN=57000000000000001 | Test to fail | 3 |
| TC002-005.5 | 5. First name length > 128 | 1. Select new patient<br>2. Input data<br>3. Submit | First name= Ratchanok….> 128 chars | Test to fail | 3 |
| TC002-005.6 | 6. Last name length > 128 | 1. Select new patient<br>2. Input data<br>3. Submit | Last name= Chaipraserth…> > 128 chars | Test to fail | 3 |
| TC002-005.7 | 7. SID length = 1-11<br>8. HN length = 1-16<br>9. First name length = 1-128<br>10. Last name length = 1-128 | 1. Select new patient<br>2. Input data<br>3. Submit | HN = 57000001 | Test to pass | 5 |

**3.4.2.6 Edit patient properties function:** Testing adds new patient function of application. For any input as invalid data or not input require field, progression cannot be complete. Classify input data same the function of add new request.

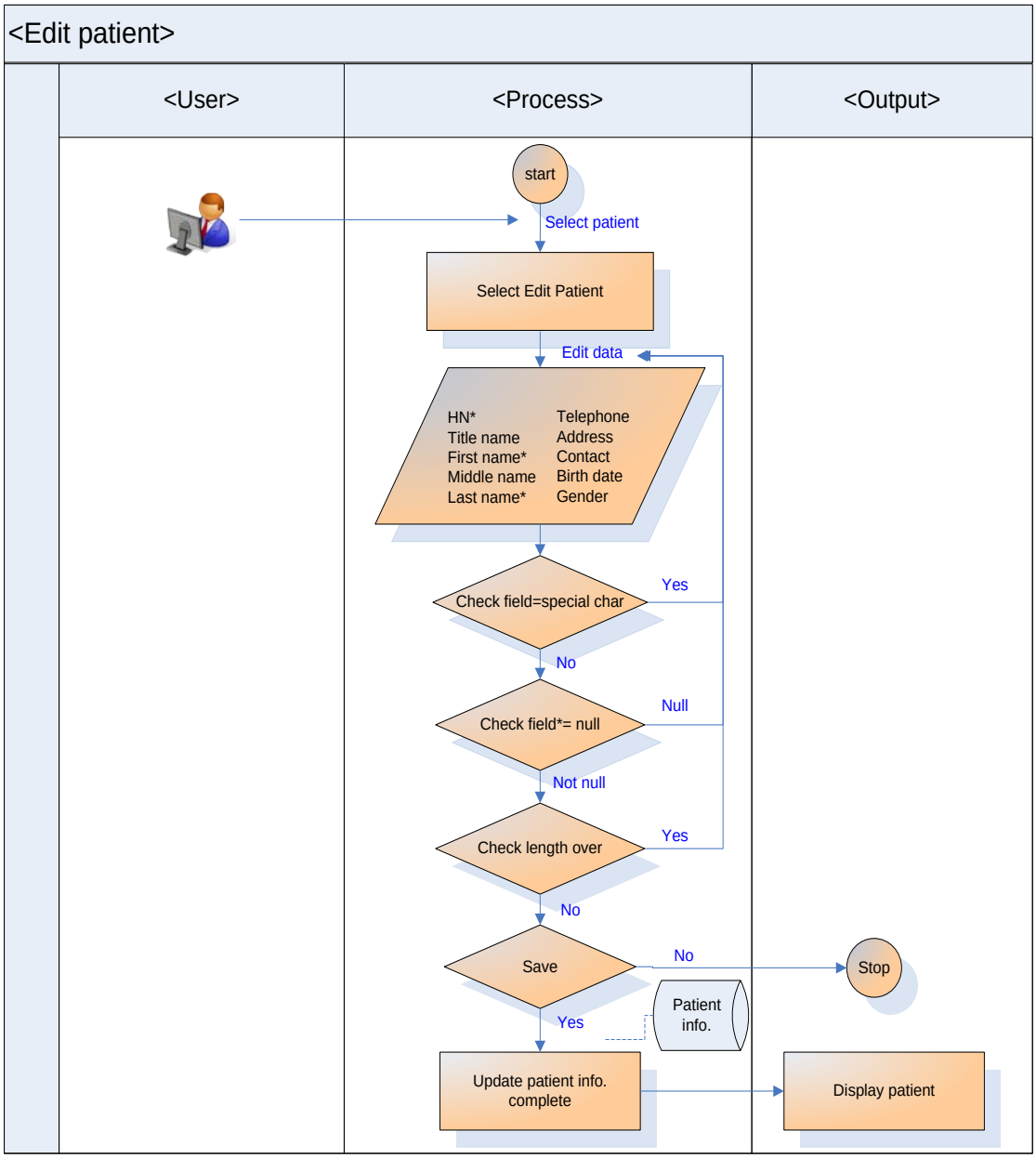


Figure 3.12 Activity flow of edit patient properties

**Table 3.38 Test cases: Edit patient properties without testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC002-006.1 | 1. HN = blank | 1. Select patient<br>2. Select edit<br>3. Update data<br>4. Submit | HN = blank | Test to pass | 3 |
| TC002-006.2 | 2. First name = blank | 1. Select patient<br>2. Select edit<br>3. Update data<br>4. Submit | First name = blank | Test to fail | 3 |
| TC002-006.3 | 3. Last name = blank | 1. Select patient<br>2. Select edit<br>3. Update data<br>4. Submit | Last name = blank | Test to fail | 3 |
| TC002-006.4 | 7. SID length = 1-11<br>8. HN length = 1-16<br>9. First name length = 1-128<br>10. Last name length = 1-128 | 1. Select patient<br>2. Select edit<br>3. Update data<br>4. Submit | HN = 57000001 | Test to pass | 7 |

**Table 3.39 Test cases: Edit patient properties with testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC002-006.1 | 1. HN = blank | 1. Select patient<br>2. Select edit<br>3. Update data<br>4. Submit | HN = blank | Test to pass | 3 |
| TC002-006.2 | 2. First name = blank | 1. Select patient<br>2. Select edit<br>3. Update data<br>4. Submit | First name = blank | Test to fail | 3 |

**Table 3.39 Test cases: Edit patient properties with testing techniques (cont.)**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC002-006.3 | 3. Last name = blank | 1. Select patient<br>2. Select edit<br>3. Update data<br>4. Submit | Last name = blank | Test to fail | 3 |
| TC002-006.4 | 4. HN length > 16 | 1. Select patient<br>2. Select edit<br>3. Update data<br>4. Submit | HN=57000000000000001 | Test to fail | 3 |
| TC002-006.5 | 5. First name length > 128 | 1. Select patient<br>2. Select edit<br>3. Update data<br>4. Submit | First name= Ratchanok….> 128 chars | Test to fail | 3 |
| TC002-006.6 | 6. Last name length > 128 | 1. Select patient<br>2. Select edit<br>3. Update data<br>4. Submit | Last name= Chaipraserth…> > 128 chars | Test to fail | 3 |
| TC002-006.7 | 7. SID length = 1-11<br>8. HN length = 1-16<br>9. First name length = 1-128<br>10. Last name length = 1-128 | 1. Select patient<br>2. Select edit<br>3. Update data<br>4. Submit | HN = 57000001 | Test to pass | 7 |

### 3.4.2.7 Delete patient function

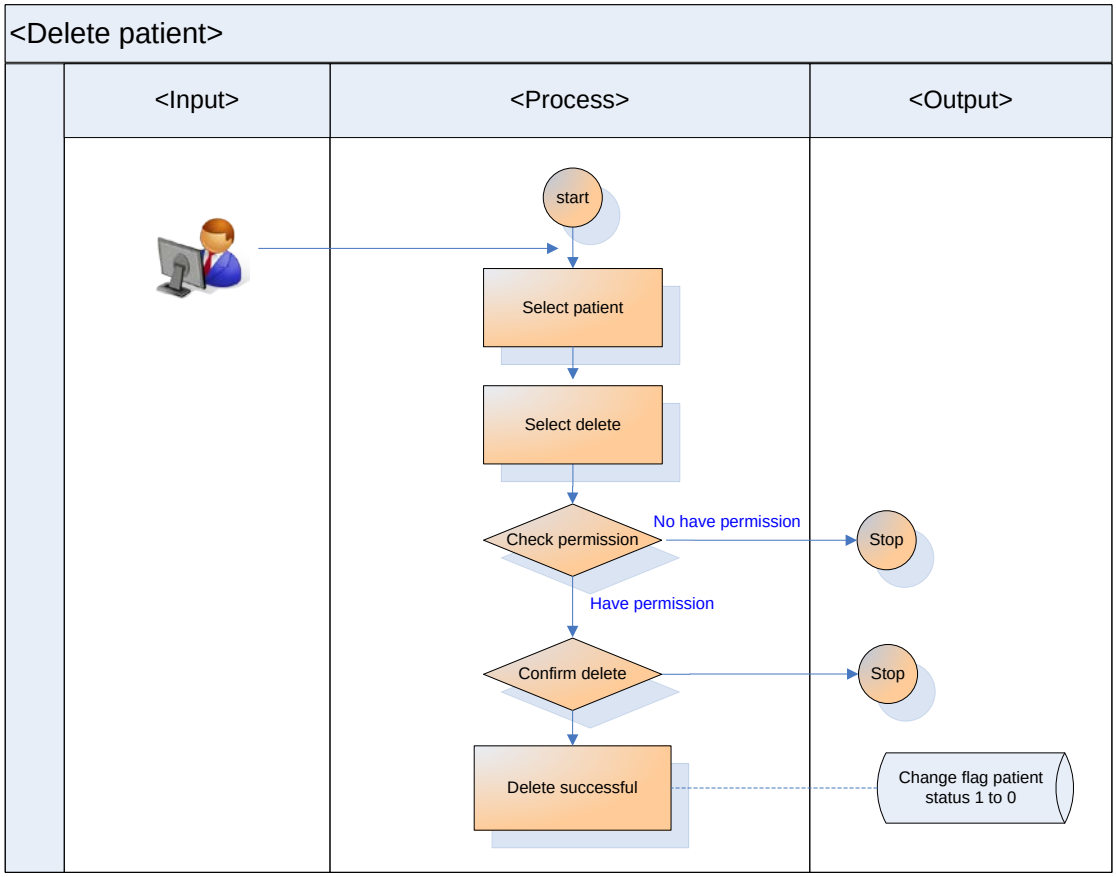


Figure 3.13 Activity flow of delete patient

**Table 3.40 Classify input data: Delete patient**

| Class No. | Description | Data Type |
|---|---|---|
| 1 | Permission access= No | Invalid |
| 2 | Delete request Not complete | Invalid |
| 3 | Permission access= Yes | Valid |
| 4 | Delete request complete | Valid |

**Table 3.41 Test cases: Delete patient without testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC002-007.1 | 1. Permission access= No | 1. Select request<br>2. Select delete | N/A | Test to fail | 5 |
| TC002-007.2 | 2. Delete request Not complete | 1. Select request<br>2. Select delete<br>3. Confirm delete<br>4. Submit | N/A | Test to fail | 3 |
| TC002-007.3 | 3. Permission access= No<br>4. Delete request Not complete | 1. Select request<br>2. Select delete<br>3. Confirm delete<br>4. Submit | N/A | Test to pass | 3 |

**Table 3.42 Test cases: Delete patient with testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC002-007.1 | 1. Permission access= No | 1. Select request<br>2. Select delete | N/A | Test to fail | 5 |
| TC002-007.2 | 2. Delete request Not complete | 1. Select request<br>2. Select delete<br>3. Confirm delete<br>4. Submit | N/A | Test to fail | 3 |
| TC002-007.3 | 3. Permission access= No<br>4. Delete request Not complete | 1. Select request<br>2. Select delete<br>3. Confirm delete<br>4. Submit | N/A | Test to pass | 3 |

### 3.4.3 TC003 Request Monitoring

#### 3.4.3.1 View request function

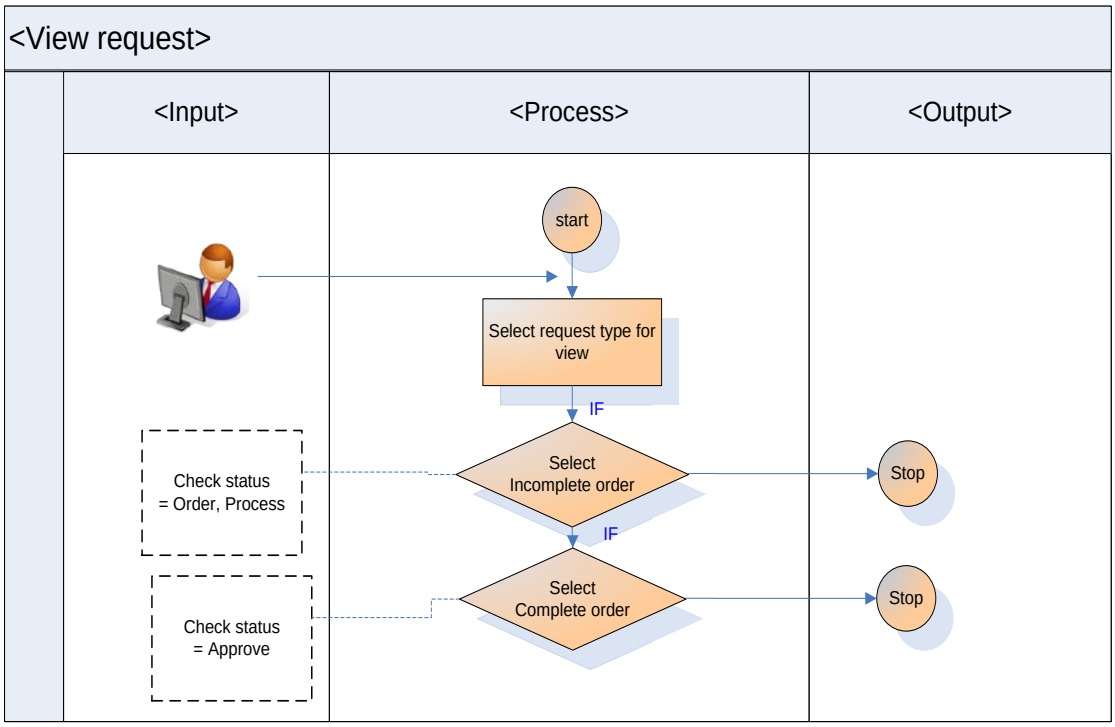


Figure 3.14 Activity flow of view request

**Table 3.43 Classify input data: View request**

| Class No. | Description | Data Type |
|---|---|---|
| 1 | Request is null | Invalid |
| 2 | PK<=0 | Invalid |
| 3 | Request is Not null | Valid |
| 4 | PK>0 | Valid |

**Table 3.44 Test cases: View request without testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC003-001.1 | 1. Request is null | 1. Select request status<br>2. Submit | N/A | Test to fail | 5 |
| TC003-001.2 | 2. PK<=0 | 1. Select request status<br>2. Submit | N/A | Test to fail | 3 |
| TC003-001.3 | 3. Request is Not null<br>4. PK>0 | 1. Select request status<br>2. Submit | N/A | Test to pass | 3 |

**Table 3.45 Test cases: View request with testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC003-001.1 | 1. Request is null | 1. Select request status<br>2. Submit | N/A | Test to fail | 5 |
| TC003-001.2 | 2. PK<=0 | 1. Select request status<br>2. Submit | N/A | Test to fail | 3 |
| TC003-001.3 | 3. Request is Not null<br>4. PK>0 | 1. Select request status<br>2. Submit | N/A | Test to pass | 3 |

### 3.4.3.2 Search request function

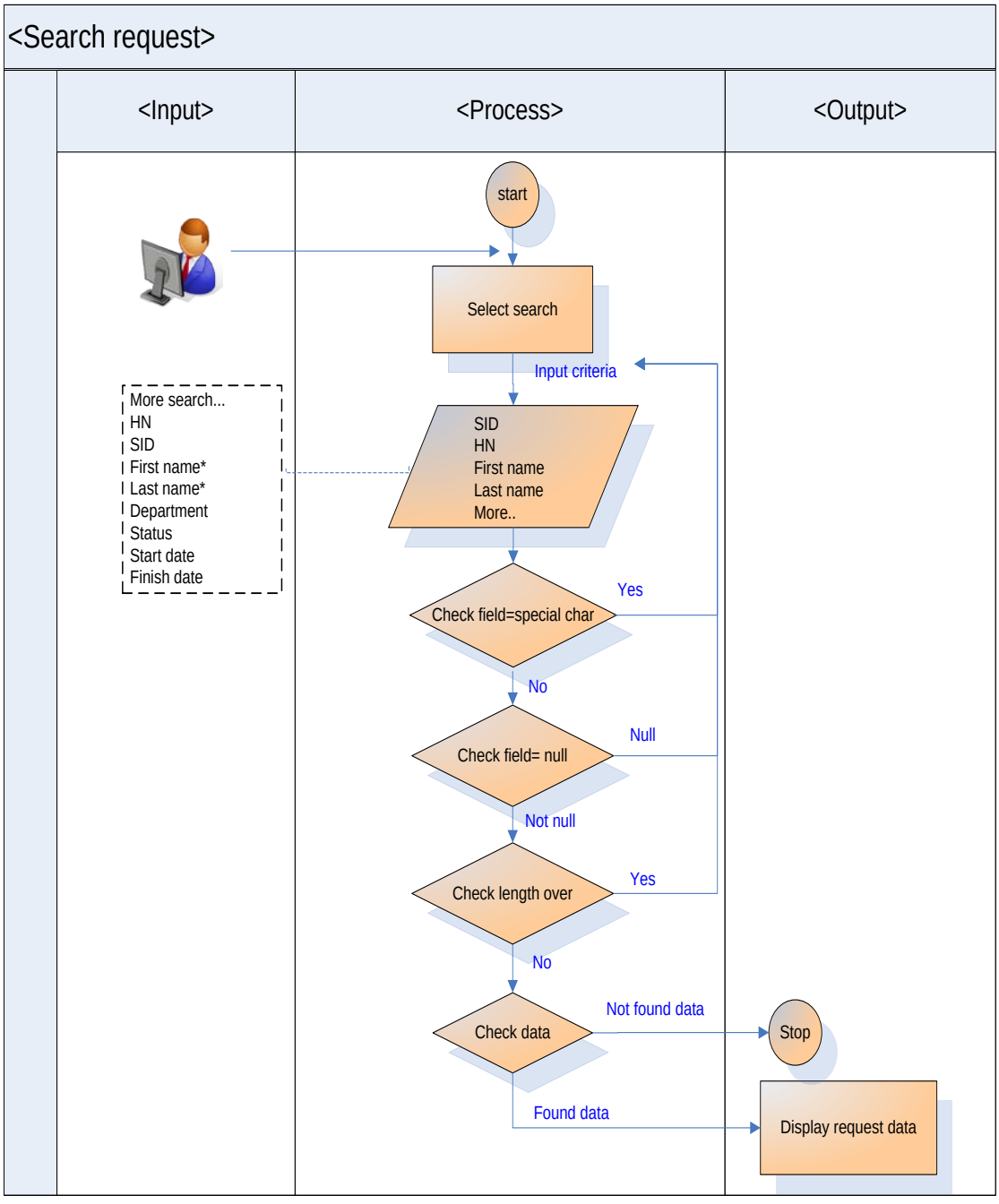


Figure 3.15 Activity flow of search request

**Table 3.46 Classify input data: Search request**

| Class No. | Description | Data Type |
|---|---|---|
| 1 | HN = blank | Invalid |
| 2 | SID= blank | Invalid |
| 3 | First name = blank | Invalid |
| 4 | Last name = blank | Invalid |
| 5 | Department = blank | Invalid |
| 6 | HN length > 16 | Invalid |
| 7 | First name length > 128 | Invalid |
| 8 | Last name length > 128 | Invalid |
| 9 | SID length = 1-11 | Valid |
| 10 | HN length = 1-16 | Valid |
| 11 | First name length = 1-128 | Valid |
| 12 | Last name length = 1-128 | Valid |
| 13 | Start date is date | Valid |
| 14 | Finish date is date | Valid |

**Table 3.47 Test cases: Search request without testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC003-002.1 | 1. HN = blank | 1. Input criteria<br>2. Submit | HN = blank | Test to pass | 3 |
| TC003-002.2 | 2. SID = blank | 1. Input criteria<br>2. Submit | SID = blank | Test to fail | 3 |
| TC003-002.3 | 3. First name = blank | 1. Input criteria<br>2. Submit | First name = blank | Test to fail | 3 |
| TC003-002.4 | 4. Last name = blank | 1. Input criteria<br>2. Submit | Last name = blank | Test to fail | 3 |
| TC003-002.5 | 5. Department = blank | 1. Input criteria<br>2. Submit | Department = blank | Test to fail | 3 |
| TC003-002.6 | 9. SID length = 1-11<br>10. HN length = 1-16<br>13. Start date is date<br>14. Start date is date | 1. Input criteria<br>2. Submit | HN = 57000001 | Test to pass | 5 |

**Table 3.48 Test cases: Search request with testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC003-002.1 | 1. HN = blank | 1. Input criteria<br>2. Submit | HN = blank | Test to pass | 3 |
| TC003-002.2 | 2. SID = blank | 1. Input criteria<br>2. Submit | SID = blank | Test to fail | 3 |
| TC003-002.3 | 3. First name = blank | 1. Input criteria<br>2. Submit | First name = blank | Test to fail | 3 |
| TC003-002.4 | 4. Last name = blank | 1. Input criteria<br>2. Submit | Last name = blank | Test to fail | 3 |
| TC003-002.5 | 5. Department = blank | 1. Input criteria<br>2. Submit | Department = blank | Test to fail | 3 |
| TC003-002.6 | 6. HN length > 16 | 1. Input criteria<br>2. Submit | Hn= 5700000000000001 | Test to fail | 3 |
| TC003-002.7 | 7. First name length > 128 | 1. Input criteria<br>2. Submit | First name = RatchaNok…> 128 | Test to fail | 3 |
| TC003-002.8 | 8. Last name length > 128 | 1. Input criteria<br>2. Submit | Last name = Chaipraserth…> 128 | Test to fail | 3 |
| TC003-002.9 | 9. SID length = 1-11<br>10. HN length = 1-16<br>11. First name length = 1-128<br>12. Last name length = 1-128<br>13. Start date is date<br>14. Start date is date | 1. Input criteria<br>2. Submit | HN = 57000001 | Test to pass | 5 |

### 3.4.3.3 Print result function

**Table 3.49 Classify input data: Print result**

| Class No. | Description | Data Type |
|---|---|---|
| 1 | Select request is null | Invalid |
| 2 | Print Not success | Invalid |
| 3 | Select request is Not null | Valid |
| 4 | Print success | Valid |

**Table 3.50 Test cases: Print result without testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC003-003.1 | 1. Select request is null | 1. Select request<br>2. Select print | N/A | Test to fail | 3 |
| TC003-003.2 | 2. Print Not success | 1. Select request<br>2. Select print | N/A | Test to fail | 3 |
| TC003-003.3 | 3. Select request is Not null<br>4. Print success | 1. Select request<br>2. Select print | N/A | Test to pass | 3 |

**Table 3.51 Test cases: Print result with testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC003-003.1 | 1. Select request is null | 1. Select request<br>2. Select print | N/A | Test to fail | 3 |
| TC003-003.2 | 2. Print Not success | 1. Select request<br>2. Select print | N/A | Test to fail | 3 |
| TC003-003.3 | 3. Select request is Not null<br>4. Print success | 1. Select request<br>2. Select print | N/A | Test to pass | 3 |

### 3.4.4 TC004 Result Monitoring

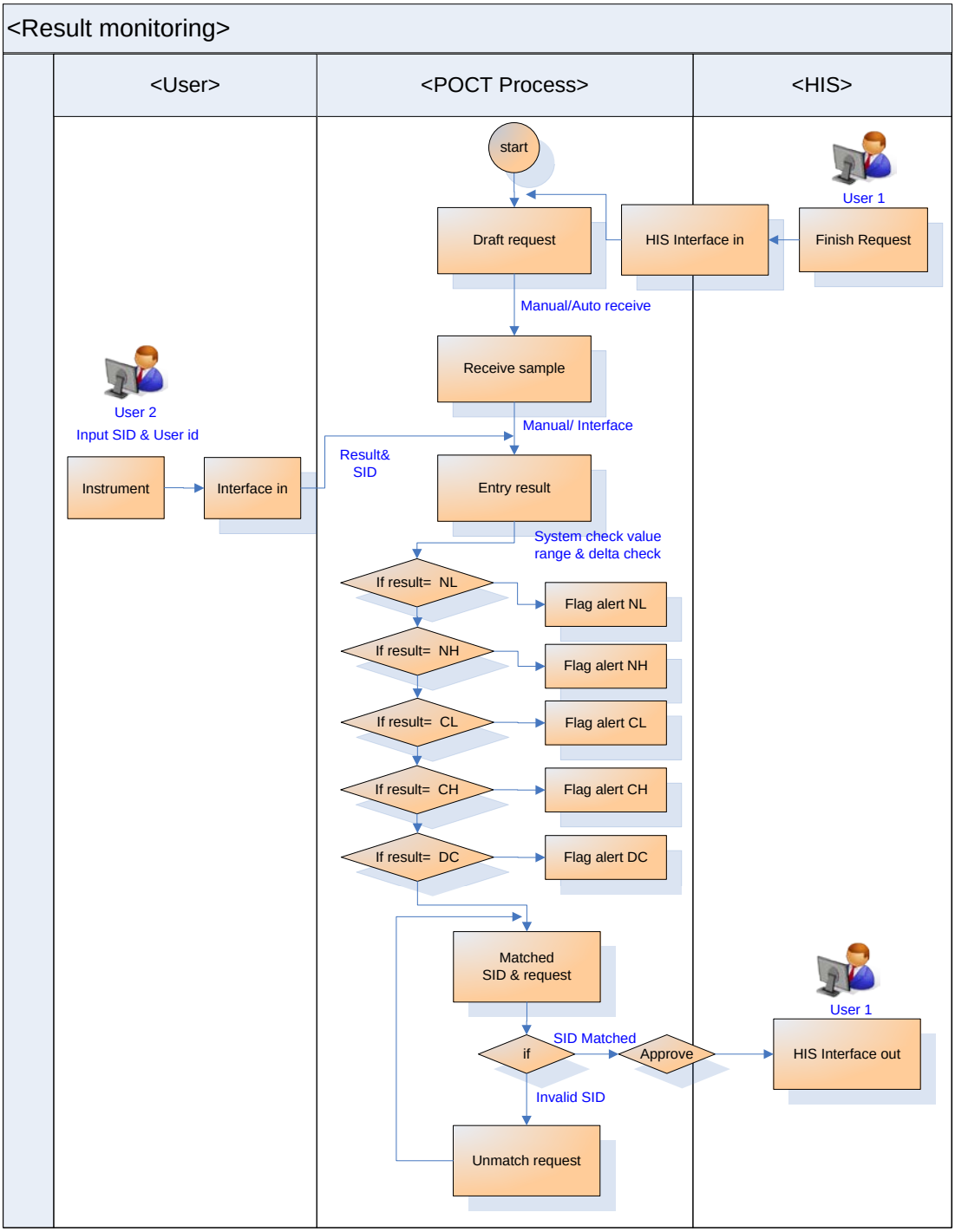


Figure 3.16 Activity flow of result monitoring

### 3.4.4.1 Receive sample function

**Table 3.52 Classify input data: Receive sample**

| Class No. | Description | Data Type |
|---|---|---|
| 1 | Request is null | Invalid |
| 2 | Request status >=1 | Invalid |
| 3 | Request is Not null | Valid |
| 4 | Request status =0 | Valid |

**Table 3.53 Test cases: Receive sample without testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC004-001.1 | 1. Request is null | 1. Select request<br>2. Select receive | N/A | Test to fail | 3 |
| TC004-001.2 | 2. Request status >=1 | 1. Select request<br>2. Select receive | N/A | Test to fail | 3 |
| TC004-001.3 | 3. Request is Not null<br>4. Request status =0 | 1. Select request<br>2. Select receive | N/A | Test to pass | 3 |

**Table 3.54 Test cases: Receive sample with testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC004-001.1 | 1. Request is null | 1. Select request<br>2. Select receive | N/A | Test to fail | 3 |
| TC004-001.2 | 2. Request status >=1 | 1. Select request<br>2. Select receive | N/A | Test to fail | 3 |
| TC004-001.3 | 3. Request is Not null<br>4. Request status =0 | 1. Select request<br>2. Select receive | N/A | Test to pass | 3 |

### 3.4.4.2 Entry Result and flag alert function

**Table 3.55 Classify input data: Entry Result and Flag alert**

| Class No. | Description | Data Type |
|---|---|---|
| 1 | Code is empty | Invalid |
| 2 | Code length > 16 | Invalid |
| 3 | Test name is empty | Invalid |
| 4 | Test name length > 16 | Invalid |
| 5 | Display_rang is empty | Invalid |
| 6 | Display_rang is empty > 128 | Invalid |
| 7 | Nomal_low > Normal_high | Invalid |
| 8 | Nomal_low < critical_low | Invalid |
| 9 | Normal_low = Normal hihg | Invalid |
| 10 | Critical_low > critical_high | Invalid |
| 11 | Normal_high > critical_high | Invalid |
| 12 | Duplicate Code | Invalid |
| 13 | Code length =1- 16 | Valid |
| 14 | Test name length = 1-16 | Valid |
| 15 | Display rang length =1- 128 | Valid |
| 16 | Result in reference range | Valid |
| 17 | Nomal_low < Normal_high | Valid |
| 18 | Nomal_low > critical_low | Valid |
| 19 | Normal_low <> Normal hihg | Valid |
| 20 | Critical_low < critical_high | Valid |
| 21 | Normal_high < critical_high | Valid |

**Table 3.56 Test cases: Entry Result and flag alert without testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC004-002.1 | 1. Code is empty | 1. Select test<br>2. Entry result | Code = null | Test to fail | 5 |
| TC004-002.2 | 3. Test name is empty | 1. Select test<br>2. Entry result | Test name= null | Test to fail | 3 |
| TC004-002.3 | 5. Display_rang is empty | 1. Select test<br>2. Entry result | Display rang= null | Test to fail | 3 |
| TC004-002.4 | 7. Nomal_low > Normal_high | 1. Select test<br>2. Entry result | Normal low= 102<br>Normal high= 101 | Test to fail | 5 |
| TC004-002.5 | 8. Nomal_low < critical_low | 1. Select test<br>2. Entry result | Normal low= -1<br>Critical low= 4 | Test to fail | 5 |
| TC004-002.6 | 9. Normal_low = Normal hihg | 1. Select test<br>2. Entry result | Normal low= 30<br>Normal high= 101 | Test to fail | 2 |
| TC004-002.7 | 10. Critical_low > critical_high | 1. Select test<br>2. Entry result | Critical low= 190<br>Critical high= 150 | Test to fail | 2 |
| TC004-002.8 | 11. Normal_high > critical_high | 1. Select test<br>2. Entry result | Normal high = 195<br>Critical high= 190 | Test to fail | 2 |
| TC004-002.9 | 12. Duplicate Code | 1. Select test<br>2. Entry result | N/A | Test to fail | 3 |
| TC004-002.10 | 16. Result in reference range | 1. Select test<br>2. Entry result | Result= 50-100 | Test to pass | 3 |

**Table 3.57 Test cases: Entry Result and flag alert with testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC004-002.1 | 1. Code is empty | 1. Select test<br>2. Entry result | Code = null | Test to fail | 5 |
| TC004-002.2 | 2. Code length > 16 | 1. Select test<br>2. Entry result | Code= 00000000000000<br>00000000001 | Test to fail | 4 |
| TC004-002.3 | 3. Test name is empty | 1. Select test<br>2. Entry result | Test name= null | Test to fail | 3 |

**Table 3.57 Test cases: Entry Result and flag alert with testing techniques (cont.)**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC004-002.4 | 4. Test name length > 16 | 1. Select test<br>2. Entry result | Test name=<br>Bun000000000000000 | Test to fail | 4 |
| TC004-002.5 | 5. Display_rang is empty | 1. Select test<br>2. Entry result | Display rang= null | Test to fail | 3 |
| TC004-002.6 | 6. Display_rang is empty > 128 | 1. Select test<br>2. Entry result | 120000000-100000….>128 chars | Test to fail | 4 |
| TC004-002.7 | 7. NL > Normal_high | 1. Select test<br>2. Entry result | Normal low= 102<br>Normal high= 101 | Test to fail | 5 |
| TC004-002.8 | 8. NL < critical_low | 1. Select test<br>2. Entry result | Normal low= -1<br>Critical low= 4 | Test to fail | 5 |
| TC004-002.9 | 9. NL = Normal hihg | 1. Select test<br>2. Entry result | Normal low= 30<br>Normal high= 101 | Test to fail | 2 |
| TC004-002.10 | 10. CL > critical_high | 1. Select test<br>2. Entry result | Critical low= 190<br>Critical high= 150 | Test to fail | 2 |
| TC004-002.11 | 11. Normal_high > critical_high | 1. Select test<br>2. Entry result | Normal high = 195<br>Critical high= 190 | Test to fail | 2 |
| TC004-002.12 | 12. Duplicate Code | 1. Select test<br>2. Entry result | N/A | Test to fail | 3 |
| TC004-002.13 | 13. Code length =1- 16<br>14. Test name length = 1-16<br>15. Display rang length =1- 128<br>16. Result in reference range<br>17. NL < Normal_high<br>18. NL > critical_low<br>19. NL <> Normal high<br>20. CL < critical_high<br>21. NH < critical_high | 1. Select test<br>2. Entry result | Result= 50<br>Test= Bun<br>Range= 50-100 | Test to pass | 5 |

### 3.4.4.3 Approve function

**Table 3.58 Classify input data: Approve**

| Class No. | Description | Data Type |
|---|---|---|
| 1 | Request status is approve | Invalid |
| 2 | Approve Not success | Invalid |
| 3 | Request status is Not approve | Valid |
| 4 | Approve success | Valid |

**Table 3.59 Test cases: Approve without testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC004-003.1 | 1. Request status is approve | 1. Select request<br>2. Select approve | N/A | Test to fail | 3 |
| TC004-003.2 | 2. Approve Not success | 1. Select request<br>2. Select approve | N/A | Test to fail | 3 |
| TC004-003.3 | 3. Request is not approve<br>4. Approve success | 1. Select request<br>2. Select approve | N/A | Test to pass | 3 |

**Table 3.60 Test cases: Approve with testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC004-003.1 | 1. Request status is approve | 1. Select request<br>2. Select approve | N/A | Test to fail | 3 |
| TC004-003.2 | 2. Approve Not success | 1. Select request<br>2. Select approve | N/A | Test to fail | 3 |
| TC004-003.3 | 3. Request status is Not approve<br>4. Approve success | 1. Select request<br>2. Select approve | N/A | Test to pass | 3 |

### 3.4.4.4 Unapprove function

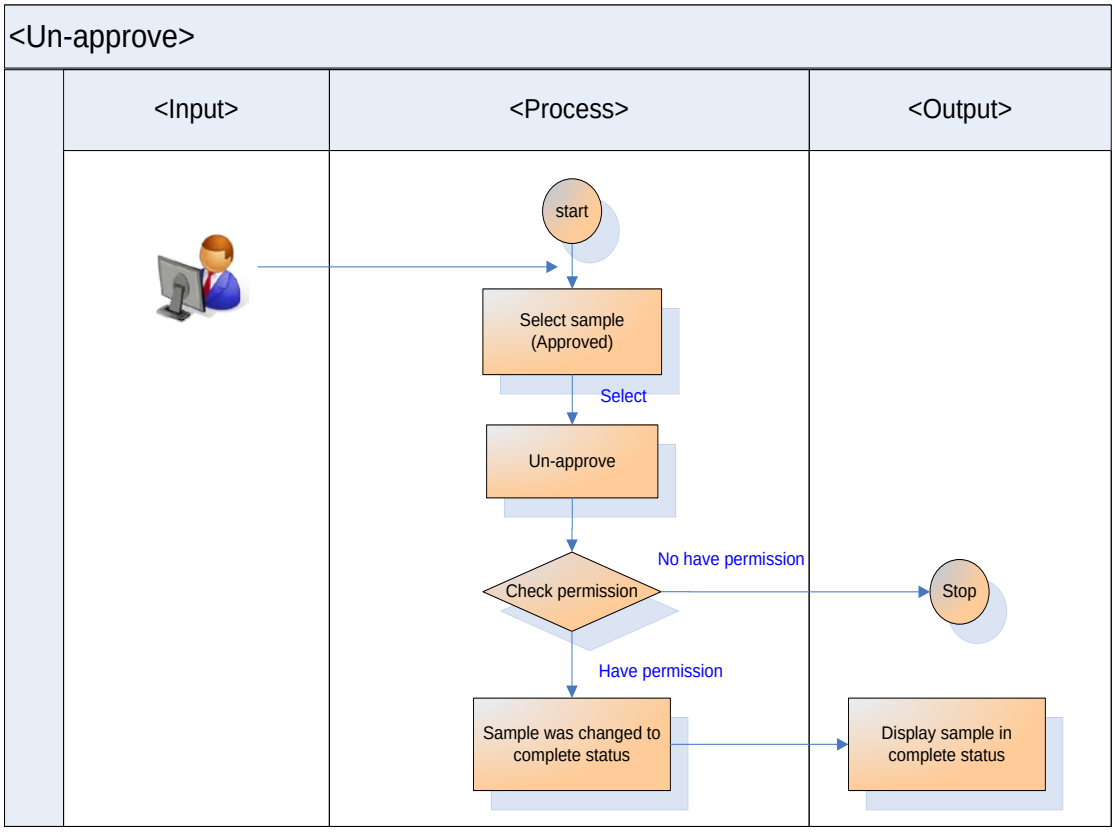


Figure 3.17 Activity flow of unapproved

**Table 3.61 Classify input data: Unapprove**

| Class No. | Description | Data Type |
|---|---|---|
| 1 | Request status is Not approve | Invalid |
| 2 | Unapprove Not success | Invalid |
| 3 | Request status is approve | Valid |
| 4 | Unapprove success | Valid |

**Table 3.62 Test cases: Unapprove without testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC004-004.1 | 1. Request status is Not approve | 1. Select request<br>2. Select Unapprove | N/A | Test to fail | 3 |
| TC004-004.2 | 2. Unapprove Not success | 1. Select request<br>2. Select Unapprove | N/A | Test to fail | 3 |
| TC004-004.3 | 3. Request status is approve<br>4. Unapprove success | 1. Select request<br>2. Select Unapprove | N/A | Test to pass | 3 |

**Table 3.63 Test cases: Unapprove with testing techniques**

| Test Case No. | Condition | Test step | Input data | Test type | Time (min) |
|---|---|---|---|---|---|
| TC004-004.1 | 1. Request status is Not approve | 1. Select request<br>2. Select Unapprove | N/A | Test to fail | 3 |
| TC004-004.2 | 2. Unapprove Not success | 1. Select request<br>2. Select Unapprove | N/A | Test to fail | 3 |
| TC004-004.3 | 3. Request status is approve<br>4. Unapprove success | 1. Select request<br>2. Select Unapprove | N/A | Test to pass | 3 |

# CHAPTER IV
# RESULTS AND DISCUSSION

A TDD of automation framework and testing techniques was studied. This chapter discussed result of our experiment, the result comparison between TDD using testing techniques for generate test case and TDD without testing techniques for generate test case.

## 4.1 Result of work

Our approach code of program was built using TDD technique. We think what to do and how to test functional code that makes us aware of production scope to be done. In development process we create functional code and test code together. Test case used to create test code following as:

## 4.2 Discussion

Our approach code of program was built using TDD technique. We think what to do and how to test functional code that makes us aware of production scope to be done. In development process although we spend much more time on a task to fix bugs to make test pass, but on maintenance phase will less debugging of program due to our approaches make more effective code, regression test much easier to make automation test when requirement was changed or bugs happening. Result of an original Software Development without testing techniques see in Table 4.1 and comparing result between TDD using testing techniques and without testing techniques see in Table 4.2

**Table 4.1 Result of an original Software Development without testing techniques**

| Result | | | |
|---|---|---|---|
| **Test case** | **An original Software Development** | | |
| | **Total steps** | **Time to fix bug (min)** | **Bug found after code** |
| **TC001** | **Authentication** | | |
| TC001-001 | 7 | 16 | 3 |
| TC001-002 | 2 | 5 | 2 |
| TC001-003 | 4 | 7 | 2 |
| TC001-004 | 8 | 20 | 3 |
| **TC002** | **Request management** | | |
| TC002-001 | 13 | 25 | 5 |
| TC002-002 | 13 | 25 | 8 |
| TC002-003 | 3 | 6 | 1 |
| TC002-004 | 4 | 8 | 2 |
| TC002-005 | 4 | 7 | 1 |
| TC002-006 | 4 | 7 | 1 |
| TC002-007 | 3 | 7 | 2 |
| **TC003** | **Request monitoring** | | |
| TC002-001 | 3 | 7 | 2 |
| TC002-002 | 6 | 15 | 2 |
| TC002-003 | 3 | 5 | 1 |
| **TC004** | **Result monitoring** | | |
| TC004-001 | 3 | 5 | 2 |
| TC004-002 | 10 | 20 | 4 |
| TC004-003 | 3 | 5 | 2 |
| TC004-004 | 3 | 5 | 2 |

The Table 4.1 is an original software development that starts process with design-code-test. We found test fails on the end of process so we spend much more time to fix bugs and re-test because we have no test script to run automation test then

we must to repeat same process testing again. That made our project time plan delay and increase cost to fix bugs in maintenance phase.

**Table 4.2 Compare TDD result between within and without testing techniques**

| Compare result | | | | |
|---|---|---|---|---|
| **Test case** | **TDD Development without testing technique** | | **TDD Development with testing technique** | |
| | **Total steps** | **Time(min)** | **Total steps** | **Time(min)** |
| **TC001** | **Authentication** | | | |
| TC001-001 | 7 | 26 | 11 | 38 |
| TC001-002 | 2 | 6 | 2 | 6 |
| TC001-003 | 4 | 12 | 14 | 42 |
| TC001-004 | 8 | 31 | 11 | 40 |
| **TC002** | **Request management** | | | |
| TC002-001 | 13 | 39 | 19 | 57 |
| TC002-002 | 13 | 39 | 19 | 57 |
| TC002-003 | 3 | 11 | 3 | 11 |
| TC002-004 | 4 | 12 | 7 | 23 |
| TC002-005 | 4 | 14 | 7 | 23 |
| TC002-006 | 4 | 16 | 7 | 25 |
| TC002-007 | 3 | 11 | 3 | 11 |
| **TC003** | **Request monitoring** | | | |
| TC003-001 | 3 | 11 | 3 | 11 |
| TC003-002 | 6 | 20 | 9 | 29 |
| TC003-003 | 3 | 9 | 3 | 9 |
| **TC004** | **Result monitoring** | | | |
| TC004-001 | 3 | 9 | 3 | 9 |
| TC004-002 | 10 | 33 | 13 | 47 |
| TC004-003 | 3 | 9 | 3 | 9 |
| TC004-004 | 3 | 9 | 3 | 9 |

The Table 4.2 is a TDD software development with testing techniques that start process with test-code-refactoring. We found a lot of test fails on early of development process. We use more time and write more code to fix bugs than original software development process. We have test code cover functional code that why finally test failed appears zero. Regression test reduced time to run automation test for check quality of code. We have successfully implement test automation framework cover all requirements in POCT system (Modules: Access program, Request management ,request monitoring, result monitoring) using TDD that helped to increase the software effective code and less debugging , software flexibility for requirement changes, reduce cost and time to fix bugs after delivery to customer.

The left side of Table 4.2 is TDD software development using test case without testing techniques we found test total 96 steps and time for development 317 minutes. In the right side of Table 4.2 is a TDD software development using test case without testing techniques we found test total 140 steps and time for development 456 minutes. The test case steps difference 44 steps and use time different 139 minutes. Compared to the ratio as follows: Test case cover more 31.43 % and time use more 30.48 %

This chapter describes the result of our experiment; we found that the experiment has given good result. We have successfully implement test automation framework cover all requirements in POCT system by using testing techniques (Modules: Access program, Request management ,request monitoring, result monitoring) and TDD that helped to increase the software effective code and less debugging in maintenance phase, software flexibility for requirement changes, reduce cost and time to fix bugs after delivery to customer.

# CHAPTER V
# CONCLUSION

Test automation framework based on TDD concept using testing techniques as black-box and white-box testing on POCT system is one of the important methods. Our studies classify test case generation which is different solution to compare between test case generation using testing techniques and test case generation without testing techniques.

## 5.1 Conclusion

We have presented to test automation framework including TDD, black-box and white-box testing techniques on POCT system that is a case study of healthcare industry. In development process, although we have spent much more time and written more codes to fix the bugs for making the passed test, but on maintenance phase will less debugs of the program due to our approaches making more effective codes. (Spending time to development the software or to fix the bugs has depended on the skills and experience of individual person.) Regression test is easy to test the automatic system in the case of changing the requirements or happening bugs. This test will help to reduce the costing and the time to fix the bugs after finishing the development processes.

## 5.2 Future work

5.2.1 Implementation all functions of POCT systems, e.g., Instrument monitor, Statistic Report, Setting.

5.2.2 Planning to use other black-box testing techniques as orthogonal array testing strategy (OATS) to minimize the amount of test cases based on

requirement specifications. The fuzzing is to manage security problems about input data (invalid, unexpected, or random data) in software focusing on exceptions such as crashes, buffer overflow, failing built-in code assertions or SQL injection.

# REFERENCES

1 Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/Test-driven_ development#Add_a_test

2 Taha Karamat, Atif Neuman Jamil, “Reducing test cost and improving documentation in TDD (Test Driven Development),” IEEE, 2006.

3 Dietmar Winkler,Reinhard Hametner,Thomas Ostreicher, Stefan Biffl, “A framework for automated testing of automation systems,” IEEE publication,2010.

4 Jovanovic, Irena, “Software testing methods and techniques,” 2008.

5 Fei Wang, Wencai Du, “A test automation framework based on web, ” IEEE publication, 2012.

6 Mohd. Ehmer Khan, “Different approaches to black box testing technique for finding errors,” IJSEA, Vol.2, No.4, October 2011.

7 Luay H. Tahat, Boris Vaysburg, Bogdan Korel , Atef J. Bader “Requirement-based automated black-box test generation,” IEEE publication, 2011.

8 Jens Uwe Pipka, “Refactoring in a test first world,” Germany, pp. 178–181.

9 Baldev S. Mattu,Ravi Shankar, “Test driven design methodology for component-based system,” IEEE publication, April 9-12, 2007.

10 Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/Fuzz_testing#Types_of_bugs

11 What is White Box Testing(Glass Box Testing), http://www.jkinfoline.com/white-box-testing.html

12 Rangsit Sirirangsi, “Software testing”, 28 March 2013.

13 M.M. Muller, O. Hagner, "Experiment about test-first programming," presented at Empirical Assessment In Software Engineering EASE, , April 2002.

14 Boby George, Laurie Williams, "A structured experiment of test-driven development," Information and Software Technology, pp.337–342, 2004.

# BIOGRAPHY

| | |
|---|---|
| **NAME** | Ms. Ratchanok Chaipraserth |
| **DATE OF BIRTH** | 16 May 1985 |
| **INSTITUTIONS ATTENDED** | Kasetsart University, 2004-2008<br>Bachelor of Science (Computer Science)<br>Mahidol University, 2011-2013<br>Master of Science (Technology Information System Management) |
| **HOME ADDRESS** | 289 Jaransanitwong Soi 67 Road,<br>Bangplud District, Bangkok 10170<br>Tel. 08-4360-8482<br>E-mail:lernukus@gmail.com |
| **RESEARCH INTEREST** | My research interest include the automation testing framework, software testing techniques, test case generation, and agile adapt in software development process. |
| **PUBLICATION / PRESENTATION** | A Test Automation Framework in POCT system using TDD Techniques, The 13th International Symposium on Communications and Information Technologies (ISCIT 2013), Samui Island, Suratthani, Thailand |