A Review: Methods of Acceptance Testing

Ryan Then Ye Tong, Yeow Kai Yuan, Ng Wen Dong, and R. Kanesaraj Ramasamy(✉)

Faculty of Computing and Informatics, Multimedia University, Cyberjaya, Malaysia
r.kanesaraj@mmu.edu.my

Abstract. This study article seeks to establish promising research areas by identifying the acceptance testing method with the best performance in the literature. Accuracy and correctness, which are the most important metrics in acceptance testing, are used to evaluate performance. Acceptance testing is a method of evaluating a system to see if it meets the requirements. In other words, it is a process for determining whether or not software fits the client's requirements.

There are various types of acceptance testing, including User Acceptance Testing, Alpha Testing, Beta Testing, and Business Acceptance Testing. The acceptance testing tasks are carried out in stages so that if the present conclusion is satisfactory, it can be used for the more difficult testing activities. One sort of acceptance testing, namely User Acceptance Testing, will be covered in this study. As a result, it is critical to employ these testing procedures in order to assist prevent software failures and needless losses. In the literature review section, a variety of reviews on the topics of acceptance testing using various approaches have been studied and addressed. Recommendations would be made based on the research study of the many testing techniques, with the goal of proposing one specific suitable testing method among the many that have been examined.

Keywords: User Acceptance Testing · Diabetes Mobile Health system · Acceptance Test Generation · Natural Language Processing · Object Constraint Language · Test case prioritization

1 Introduction

1.1 Problem Statement

Software acceptance testing plays a pivotal role in the fate of a software product. The outcome of an acceptance test is critical for the customer in deciding whether to accept or reject the software product [11], and by extension, this determines the success or failure of the product. To ensure that the software is judged with the highest possible standard of objectivity against the acceptance criteria, which is the criteria that the product must satisfy in order to be accepted by the customer [11], it is necessary for the acceptance testing method to be capable of performing the comparison with the highest possible degree of accuracy and dependability [10]. As such, a great deal of focus has to be paid on the rigorousness of the test case scenarios and inputs generated by an acceptance testing model.

1.2 Objectives

- To study different coverage metrics with respect to acceptance testability.
- To recommend methods or algorithms that facilitate better acceptance testing performance.
- To provide empirical and theoretical information on acceptance testing.

1.3 Expected Findings

In this study, we have reviewed several research papers, proposals and exploratory articles on acceptability testing. Several approaches that may be utilised while undergoing testing have also been highlighted. The primary goal of this approach is to check and confirm that the software tested has satisfied the needs of the client while also being dependable. Furthermore, descriptions of journals that were discovered are given in the report to assist the reader in comprehending all of the approaches that were discussed. That is to say, each method from the review would be analysed according to the ways on how the method works, the ways on how the algorithm is designed, and the ways on how the algorithm is being implemented. Based on the research findings, we would then select one preferable testing technique which is considered the best among all the other methods gathered from the literature reviews, as the recommendation of this report.

2 Literature Review

One of the more straightforward methods of implementing a User Acceptance Testing (UAT) method is described by Otaduy & Diaz [3]. Verifying software in a real-world environment by the intended audience is known as user testing. The goal of this testing is to guarantee that the software fits the established criteria as well as the client needs. Some extra approaches must be introduced in order to lower the cost of developer- customer communication when this testing is being conducted. When doing this User Acceptance Testing (UAT), developers will create the UAT scaffolding that will guide consumers through the testing process. To ensure that the consumer has a better comprehension of the testing, a mind map editor named FreeMind is often used. Normally, the software testing includes different levels of testing which are unit, functional, integration and system. The above-mentioned tests are not performed until the client needs are completed in a real- world situation, and they cannot determine if the customer criteria are met or not. The programme will only be tested in a real-world situation by the target audience during the UAT. First, acceptance tests are developed to outline the program's intended functionality. UAT is conducted after the product has been developed and launched in order to verify and accept the software. The authors of this study report offered a way for supporting self-paced UAT. UAT sessions are described in terms of mind maps, which are usually referred to as test maps. The study's main contribution is a method that combines mind mapping and wikis to allow customers to do UAT independently. Wikis were recognised to be advantageous in the workplace for organisations in need of a collaborative medium. This is followed by a DSL for an UAT with concrete syntax realised using mind maps, which is then implemented. The third and final stage will be

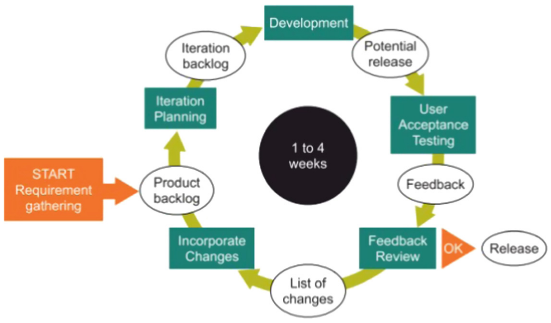


Fig. 1. Outline of the Process.

the implementation of self-paced testing using Wiki, Fitnesse and the Freemind mind map editor. Testmap is used to connect both systems. This study's primary emphasis is on UAT inside Agile techniques. In these methods, small sets of needs are developed in a short amount of time, known as 'iterations', and these small groups of requirements are iterated upon (Fig. 1).

The author also indicated in the article that UAT sessions would gather three sorts of data: scaffolding, UAT cases, and UAT scripts. Mind maps may be used to collect all of the above described data. Scaffolding refers to both the Sprint features that are going to be tested and the ready to go kickoffs that compose the scenario of UAT. In general, the feature to test becomes a node on the map from which its many kickoffs hang up. A set of test variables, circumstances of the execution, and expected outcomes in UAT scenarios are all documented as part of the UAT test set. UAT Actions are test cases that describe a series of actions and will be arranged into Pages to help customers comprehend. UAT Scripts are executable artefacts produced by UATCases. Each UATScript runs a UATCase with a distinct set of input data to test the application's behaviour in various circumstances. To conclude, Agile approaches place high demands on user acceptance testing, if only because of the regularity with which it must be performed. During user acceptance testing, in-person meetings with customers and engineers may need to be supplemented by asynchronous means for them to communicate.

Acceptance testing is one of the black box testing according to the situations such as system action sequences done by the user. Testing the systems of IoT, such as DiaMH, comes with remarkable challenges due to the involvement of various interconnected components, and the possibility of independent failure [5]. For instance, DiaMH (Diabetes Mobile Health) is an IoT system that keeps track of the level of glucose of the patient, alerts both the patient and doctor when the level of glucose is out of the specified range, and adjusts the insulin dosing. Generally, an entire plan of testing includes the unit testing on the components at the early phase, the integration testing on the components altogether, and most importantly, the acceptance testing carried out at two distinct stages. For the first stage of the acceptance testing, the testing on the virtualized IoT systems would be conducted using the virtual device to examine the algorithms of the system such as the communication among each module. During this stage, physical issues such as hardware problems would not be perceived. Besides that, the second stage

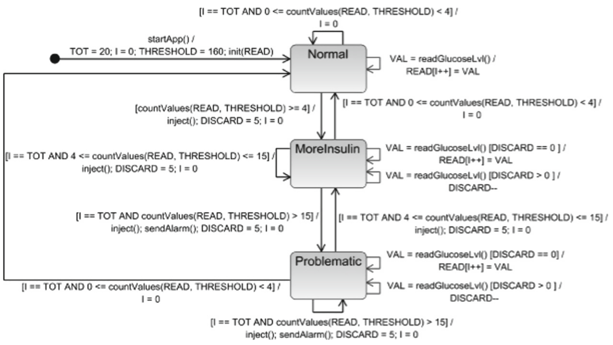

[I == TOT AND 0 <= countValues(READ, THRESHOLD) < 4] / I = 0
startApp() / TOT = 20; I = 0; THRESHOLD = 160; init(READ)
Normal
VAL = readGlucoseLvl() / READ[I++] = VAL
[countValues(READ, THRESHOLD) >= 4] / inject(); DISCARD = 5; I = 0
[I == TOT AND 0 <= countValues(READ, THRESHOLD) < 4] / I = 0
MoreInsulin
[I == TOT AND 4 <= countValues(READ, THRESHOLD) <= 15] / inject(); DISCARD = 5; I = 0
VAL = readGlucoseLvl() [DISCARD == 0] / READ[I++] = VAL
VAL = readGlucoseLvl() [DISCARD > 0] / DISCARD--
[I == TOT AND countValues(READ, THRESHOLD) > 15] / inject(); sendAlarm(); DISCARD = 5; I = 0
[I == TOT AND 4 <= countValues(READ, THRESHOLD) <= 15] / inject(); DISCARD = 5; I = 0
Problematic
VAL = readGlucoseLvl() [DISCARD == 0] / READ[I++] = VAL
VAL = readGlucoseLvl() [DISCARD > 0] / DISCARD--
[I == TOT AND 0 <= countValues(READ, THRESHOLD) < 4] / I = 0
[I == TOT AND countValues(READ, THRESHOLD) > 15] / inject(); sendAlarm(); DISCARD = 5; I = 0

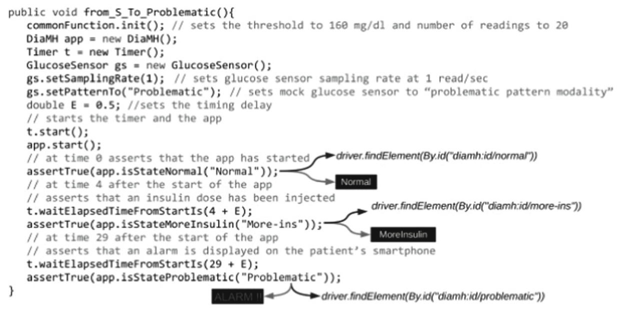
Fig. 2. The key anticipated activities of DiaMH.

of the acceptance testing would be conducted using the proper set of the real devices, hardwares, and applications. In short, the intention of the first stage is to test software created such as the applications of DiaMH running on various mobile devices, while the intention of the second stage is to test the system in real-life situations including the connection between the applications and the hardwares.

From the perspective of acceptance testing, it requires a detailed representation of the system actions. With that, the actions of the system throughout the testing tasks are formalised as a state machine based on the study of several techniques of modelling tests. As shown in Fig. 2, the condition of a patient could be represented by the states of Normal, MoreInsulin, as well as Problematic. When the application of DiaMH is started, the initial condition of a patient is set to Normal, and then the level of glucose is monitored every second with a sampling rate of 1 Hz so to adjust the condition accordingly. Once the actions and states of the application have been formalised according to the state machine, the test scripts would be performed in accordance with the available structure of testing, where it contains the test case with a list of instructions. Hence, input data is expected in order to derive the concrete test cases that are operable. Suppose the acceptance testing on the localisation of UI elements is conducted on a web application, there would be two core processes which are visual-based and structure-based, where the former is used for determining the elements by using the techniques of image recognition, and the latter is used for locating the elements by using the information found. With that, Fig. 3 shows the simple test script which implements the test case of from_S_To_Problematic.

As shown in Figure 3, each implementation of the scenario is separated to attain maintainability and reusability of codes, as well as the design principle of Separation of Concerns. Since the state is dependent on the time elapsed, the test scripts are also dependent on the timing from the moment of starting the application. Apart from that, mutation testing is also applied to access the test script quality. It is useful in discovering errors that could be produced in the phases of development and maintenance. All in all, it comes to a conclusion that the testing is able to accomplish an effectiveness of 93% after eliminating the mutants that could cause the unexpected actions on the DiaMH application [5].

On the basis of research [4], there are five important points that are generalised such as context, objective, method, results and conclusion. According to the context,



```
public void from_S_To_Problematic(){
   commonFunction.init(); // sets the threshold to 160 mg/dl and number of readings to 20
   DiaMH app = new DiaMH();
   Timer t = new Timer();
   GlucoseSensor gs = new GlucoseSensor();
   gs.setSamplingRate(1); // sets glucose sensor sampling rate at 1 read/sec
   gs.setPatternTo("Problematic"); // sets mock glucose sensor to “problematic pattern modality”
   double E = 0.5; //sets the timing delay
   // starts the timer and the app
   t.start();
   app.start();
   // at time 0 asserts that the app has started
   assertTrue(app.isStateNormal("Normal"));
   // at time 4 after the start of the app
   // asserts that an insulin dose has been injected
   t.waitElapsedTimeFromStartIs(4 + E);
   assertTrue(app.isStateMoreInsulin("More-ins"));
   // at time 29 after the start of the app
   // asserts that an alarm is displayed on the patient’s smartphone
   t.waitElapsedTimeFromStartIs(29 + E);
   assertTrue(app.isStateProblematic("Problematic"));
}
```

```
driver.findElement(By.id("diamh:id/normal"))
driver.findElement(By.id("diamh:id/more-ins"))
driver.findElement(By.id("diamh:id/problematic"))
```

Fig. 3. The test script of the method of from_S_To_Problematic.

with the development of software-based systems, the requirements are changed. The change in demand will impact the relevant acceptance tests that must be adjusted appropriately. However, the acceptance tests for them are not always current or consistent, and such a lack of coherence can lead to quality issues in software, unplanned costs and project delays. Based on this purpose, a GuideGen method is developed to keep consistency with changing requirements and its acceptance tests. When requirements change, GuideGen automatically creates natural language information on how to update the tests of acceptance. Besides, the outcomes of the evaluation demonstrate that GuideGen's recommendations are greater than 80 percent accurate for the agile requirements in the real world, and approximately 67 percent accurate for conventional specifications. It shows that the provided method uses a complete guide to upkeep of acceptance tests and staying consistent with ever-changing specifications.

Based on the method study, GuideGen is implemented as a test tool and there are also two studies that are analysed. At first, utilising three industry data sets to examine the created guidance scheme for its correctness, completeness, understandability, and relevance; second, by engaging 23 practitioners from ten firms to assess the approaches and tool's applicability and utility. The current method investigated for change impact assessment can determine the tests that are truly influenced by changes in a rule as a first step toward solving this restriction. In the first study of the result, in around 67 to 89 percent of all adjustments, GuideGen provided accurate guidance. By this metric, this method outperforms traditional methods on agile requirements. In the second study of the result, GuideGen is considered valuable. However, practitioners might recommend a GuideGen plug-in for profitable solutions over an independent application. In addition, the impacted acceptance tests can be identified for 63 percent to 91 percent of the criteria changes in trial. It can be difficult to determine which acceptance test is impacted in response to new requirements if the relationship is one- to-many. Both NARCIA and ImpRec have been analysed and shown to be useful for change effect analysis and it is shown that NARCIA might have a 90 percent accuracy rate; however, further research is needed to determine if ImpRec can be used for this purpose or not. The technique makes it easier to connect acceptance tests with actual requirements and can help specifications engineers and testers communicate more effectively.

Subjectivity, which is another important aspect of acceptance testing, is covered by Acceptance Test Generation (UMTG), which is a model that generates requirements specifications from statements written in natural language. It is an automated acceptance testing approach that focuses on embedded systems, as described in the journal [2]. A domain model and use case requirements are used by the authors to automate the production of test cases in the embedded system domain. They employ constraint solving and Natural Language Processing (NLP) to automate the identification of use case scenarios and test inputs. They also utilise NLP to extract behavioural information from use case requirements using a limited use case modelling technique called Restricted Use Case Modelling (RUCM). They create Object Constraint Language (OCL), a declarative language for describing rules that apply to Unified Modelling Language (UML) restrictions and capture the pre and post conditions of use case phases, using an advanced NLP approach termed semantic role labelling. It was decided that the branch, def-use, and subtype coverage criteria would be used to determine which use cases to consider. Route conditions that incorporate OCL limitations are used to implement alternate flows in each scenario generated by this tool. The test inputs are determined by using the Alloy analyzer to solve the route conditions (Fig. 4).

The authors demonstrate that UMTG efficiently creates acceptance test examples for sensor systems in automobiles in two industry case studies. UMTG can produce 96 percent of the OCL restrictions for test case creation automatically and accurately. The OCL creation procedure has a high level of accuracy, with 99 percent of the created constraints being correct. The coverage criteria introduced in UMTG allow engineers to identify use case situations that might otherwise go unnoticed, demonstrating its use in the safety sector.

As a consequence of insufficient requirements, the def-use coverage age and the subtype coverage criteria are used to build test cases that cover all of the situations and input partitions tested by expert-written test cases. Automated test suites may be created to mimic manual test suites without increasing testing costs. The experience demonstrates that requirements modelling, which in UMTG is essentially limited to RUCM use case descriptions and domain modelling, is feasible in a commercial environment. Furthermore, manual test case creation is much more time consuming than utilising UMTG since use case requirements and domain models are often employed for other purposes.

In a journal paper [7], Geetha et al. examined regression testing to address its primary issue, which is a lengthy and costly testing process. With regression testing, whenever changes are made in the code, the testing process ensures that the code still functions as expected after any code updates. To do this, the testing involves the re-execution of previously executed test cases to ensure existing functionalities still work as expected. Because of this, regression testing is a very resource-intensive form of testing, as the test cases increase with the complexity of the code, making the overall test suite more cumbersome and time-consuming.

To address this issue, the authors identified acceptance-based test case reduction and prioritisation as the most efficient technique in terms of fault detection rate because it provides the earliest detection possible. New prioritisation techniques, which are based on fault prediction in acceptance testing, were developed on this basis. The authors propose a combination of both reduction and prioritisation techniques with optimization techniques in this paper. Test case reduction and prioritisation is meant to find faults as early as possible if test cases are prioritised properly. To accomplish this, the authors took

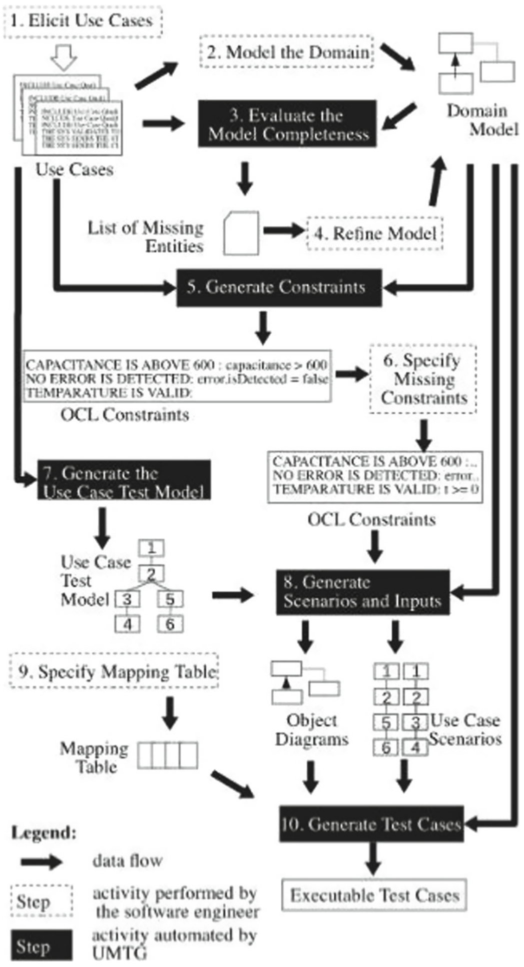

1. Elicit Use Cases
Use Cases
2. Model the Domain
3. Evaluate the Model Completeness
Domain Model
List of Missing Entities
4. Refine Model
5. Generate Constraints
CAPACITANCE IS ABOVE 600 : capacitance > 600
NO ERROR IS DETECTED: error.isDetected = false
TEMPARATURE IS VALID:
OCL Constraints
6. Specify Missing Constraints
CAPACITANCE IS ABOVE 600 :...
NO ERROR IS DETECTED: error...
TEMPARATURE IS VALID: t >= 0
OCL Constraints
7. Generate the Use Case Test Model
Use Case Test Model
8. Generate Scenarios and Inputs
9. Specify Mapping Table
Mapping Table
Object Diagrams
Use Case Scenarios
10. Generate Test Cases
Executable Test Cases
Legend:
data flow
Step
activity performed by the software engineer
Step
activity automated by UMTG

Fig. 4. Overview of UMTG Strategy.

the approach of studying genetic algorithms and multi-walk algorithms individually, and then studied the performance of an algorithm where the two techniques were combined.

3 Recommendation

After evaluating and analysing all the methods, it is concluded that the Combinatorial Logic Oriented Acceptance Test-Driven Development (CLO-ATDD) model as described in [1], has outperformed the other testing techniques. This model devises a test case formulated upon the rules of combinatorial logic. Combinatorial Testing is an approach that is specification-driven. It enables the methodical selection of test sets of programme inputs. It is a valuable approach for evaluating hardware or software systems since it detects faults depending on input or output parameter combinations. Testers can discover

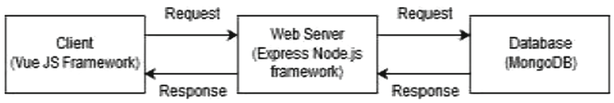


Fig. 5. The client-server architecture of CMSS.

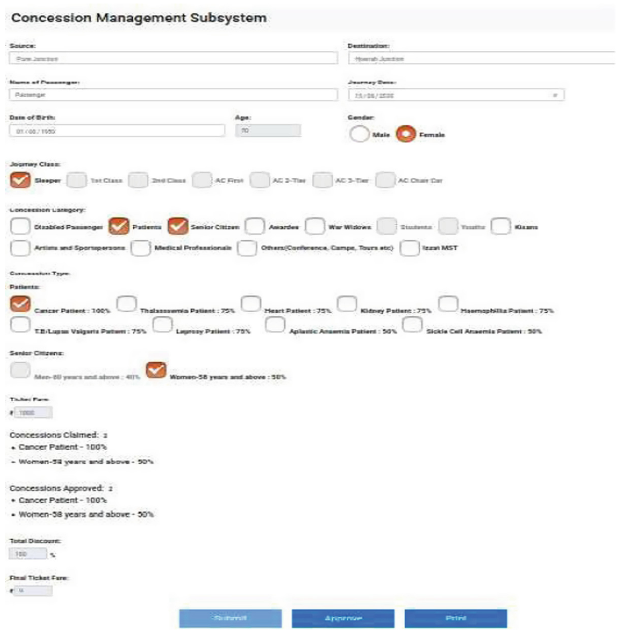
probable n-way combinatorial interactions between input variables using the combinatorial testing approach. In the CLO-ATDD paradigm, user acceptance tests that utilize combinatorial logic-oriented rules are written in Gherkin syntax which is given in the business language. Gherkin syntax offers a systematic approach for illustrating real-world application business rules.

As all phases in the Software Development Life Cycle (SDLC) can benefit from combinatorial logic, hence it is a must to concentrate on how to better include combinatorial logic into it. For the communication phase of the proposed model, the SRS document is important. To generate combinatorial logic-oriented rules, the collection of situations specified in the SRS will be applied to the combinatorial logic. These logics will be applied to the systems using combinatorial testing. Test cases are created from the user acceptance test's requirements document. For the planning phase, team members with combinatorial testing skills are favoured. As developing new techniques takes time and money, previous combinatorial testing techniques like AETG and ACTS are utilised to ensure that systems are completed on time and within budget. For the modelling phase, to illustrate the requirements specification, UML diagrams with combinatorial logic are created. Combinatorial logic is used to construct the needs specifications at this level. There will be a large number of parameter-value combinations created for a small set of parameters and values. For the construction phase, the principles of combinatorial logic are used to design user acceptability tests.

To put it another way, user acceptability testing refers to putting software to the test on behalf of the intended audience in order to determine whether or not it is acceptable. Finally, for the deployment phase, It is expected that the final programme will be deployed after user acceptance testing has been completed. This is the responsibility of the manager responsible for deployment.

For the implementation plan for the Concession Management Subsystem (CMSS), it contains multiple concession types and categories acting as the parameters where each category consists of several types. However, it has several constraints such as a male-type could not gain the available benefits from a widow-type due to an unworkable set of parameter values. CMSS uses the client-server architecture for the implementation where the client-side uses VueJS and the server-side uses NodeJS. As shown in Figure 5 and Figure 6, the former is presenting the architecture of CMSS while the latter is presenting the GUI of CMSS.

The HTTP request containing the parameters, the query, the headers, and the body, is sent from the client- side to the server-side. A response would be generated to store the data corresponding to the concession data from the request. The concession type is well-categorised so that the special type like widow could be applied based on the gender of females only. The ticket has different prices for the child-type and adult-type where



Concession Management Subsystem

Source: Pune Junction

Destination: Howrah Junction

Name of Passenger: Passenger

Journey Date: 15/08/2020

Date of Birth: 01/08/1950

Age: 70

Gender: Male [✓ Female]

Journey Class: [✓ Sleeper] 1st Class 2nd Class AC First AC 2-Tier AC 3-Tier AC Chair Car

Concession Category: Disabled Passenger [✓ Patients] [✓ Senior Citizen] Awardee War Widows Students Youths Kisans Artists and Sportspersons Medical Professionals Others(Conference, Camps, Tours etc) Izzat MST

Concession Type:

Patients: [✓ Cancer Patient : 100%] Thalassaemia Patient : 75% Heart Patient : 75% Kidney Patient : 75% Haemophilia Patient : 75% T.B/Lupus Vulgaris Patient : 75% Leprosy Patient : 75% Aplastic Anaemia Patient : 50% Sickle Cell Anaemia Patient : 50%

Senior Citizens: Men-60 years and above : 40% [✓ Women-58 years and above : 50%]

Ticket Fare: ₹ 1000

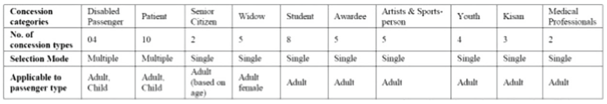
Concessions Claimed: 2
- Cancer Patient - 100%
- Women-58 years and above - 50%

Concessions Approved: 2
- Cancer Patient - 100%
- Women-58 years and above - 50%

Total Discount: 100 %

Final Ticket Fare: ₹ 0

Fig. 6. The graphical user interface (GUI) of CMSS.



Concession categories	Disabled Passenger	Patient	Senior Citizen	Widow	Student	Awardee	Artists & Sports-person	Youth	Kisan	Medical Professionals
No. of concession types	04	10	2	5	8	5	5	4	3	2
Selection Mode	Multiple	Multiple	Single	Single	Single	Single	Single	Single	Single	Single
Applicable to passenger type	Adult, Child	Adult, Child	Adult (based on age)	Adult female	Adult	Adult	Adult	Adult	Adult	Adult

Fig. 7. Categories and types of concession with selection mode.

the childrens from age between 5–12 years are priced only half the price of adult-type, and the ticket for the childrens from age below 5 years is free-of-charge (Fig. 7).

- Step 1: The types of journey class are classified into a dictionary containing the names.
- Step 2: The dictionary containing the categories with keys is declared. For example, Disability, Patient and others.
- Step 3: The empty dictionary containing the list of values of each type is defined.
- Step 4: The function calculates the values for the patient-category and the disabled-category, as two-way for pairs and three-way for triplets. The length of the list is calculated, sorted, and then multiplied by a concession value to reach the eventual concession.
- Step 5: The function which accepts n and r for calculating the combination, nCr, is declared as printCombination().
- Step 6: The function, namely combinationUtil(), accepts an array, a temporary data, starting index, and ending index. All these calculated values are stored in the dictionary of where the keys are located.

- Step 7: Declare a final concession list with the same values as the final outcome. All information for the broad category, as well as a combination of wide category and concession issued, will be saved as values.
- Step 8: Iterate the loop to obtain the possible value of combination as calculated in Step 5, from the dictionary of final_result.
- Step 9: Iterate the loop to obtain the journey class from the dictionary of class_dict, to determine the customer type.
- Step 10: Declare two lists to add all sets in opposition to the corresponding travel class along with concession.
- Step 11: After going through the initial list, make a new list with the travel class, customer type, concession value, and specific combination. Use a second loop to get the whole list, which includes all of the information about a certain combination that will be saved later in another list called final.
- Step 12: Create the broad category final list and save in concession list
- Step 13: Modify the concession list using a loop based on the requirements. All test cases that have the same concession for all travel classes will be combined into a single test case.
- Step 14: Output all the outcomes to the data file, test cases to another file, and append the unsuccessful cases in the end.

This proposed technique is able to enhance the testing results with a total of 665 test cases created with the use of two-way approach, as well as a total of 1435 test cases created with the use of three-way approach. As stated by the authors in [3], this technique could yield complete efficiency and reliability, where the UAT is totally assured.

4 Conclusion

In conclusion, there are many acceptance testing models described in literature and used in the software development industry. Many models have been used for acceptance testing according to the needs of the developers and the nuances of different software products. Finding a suitable testing model for testing software is very important.

Out of the large variety of methods used for software acceptance testing that can be found in literature, we identified one as the best-performing method in terms of efficiency and reliability, which are priorities in software acceptance testing. Although other methods have merits in terms of ease of implementation and simplicity, as they are already established and well-described in literature, their performance tends to be somewhat compromised.

References

1. Tatale S, Chandraprakash V. Enhancing acceptance test driven development model with combinatorial logic. Int J Adv Comput Sci Appl. 2020;11(10):268- 78.
2. Wang C, Pastore F, Goknil A, Briand L. Automatic generation of acceptance test cases from use case specifications: an nlp-based approach. IEEE Transactions on Software Engineering. 2020 May 29.

3. Otaduy I, Díaz O. User acceptance testing for Agile- developed web-based applications: Empowering customers through wikis and mind maps. Journal of Systems and Software. 2017 Nov 1;133:212-29.
4. Hotomski S, Glinz M. GuideGen: An approach for keeping requirements and acceptance tests aligned via automatically generated guidance. Information and Software Technology. 2019 Jun 1;110:17-38.
5. Leotta M, Clerissi D, Olianas D, Ricca F, Ancona D, Delzanno G, Franceschini L, Ribaudo M. An acceptance testing approach for Internet of Things systems. IET Software. 2018 Oct 4;12(5):430-6.
6. Wang HW, Teng KN, Zhou Y. Design an optimal accelerated-stress reliability acceptance test plan based on acceleration factor. IEEE Transactions on Reliability. 2018 May 24;67(3):1008-18.
7. Geetha U, Sankar S, Sandhya M. Acceptance testing based test case prioritization. Cogent Engineering. 2021 April 18; 8(1). Available from: https://www.tandfonline.com/doi/epub/10.1080/23311916.2021.1907013?needAccess=true.doi:10.1080/23311916.2021.1907013
8. Belzunce F, Lillo RE, Ruiz JM, Shaked M. Stochastic comparisons of nonhomogeneous processes. Probability in the Engineering and Informational Sciences. 2001 Apr;15(2):199-224.
9. Hsia P, Kung D, Sell C. Software requirements and acceptance testing. Annals of Software Engineering 3. 1997; 291–317.
10. Hetzel W, The Complete Guide to Software Testing, QED Information Systems, 1984.
11. Systems and software engineering — Vocabulary, 2nd ed., IEEE, Park Avenue, New York, USA, 2017, p. 5.