International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 10, No.1 (Jan-2021)
http://dx.doi.org/10.12785/ijcds/100112

Analyzing UML Use Cases to Generate Test Sequences

Zahra Abdulkarim Hamza[1] and Mustafa Hammad[2]

[1,2] Department of computer science – College of IT, University of Bahrain, Sakheer, Kingdom of Bahrain

Received 4 Apr. 2020, Revised 15 Jul. 2020, Accepted 5 Aug. 2020, Published 1 Jan. 2021

Abstract: Software testing is a main phase in the software development life-cycle. Testing tasks are always heavy and time-consuming due to their critical role and importance. Furthermore, testing requires several preparation steps, such as the test sequences. There are many ways to generate the test sequences to perform software testing. In this paper, UML use case diagrams are used to generate test sequences for software testing. The approach is proposed to make use of the UML use case diagrams in more than translating the software requirements to software specifications. The approach consists of several phases. Starting from converting the UML use case diagram into activity diagrams, going through the simplification step, and ending with extracting the needed information to generate the test sequences. The approach is evaluated using nine case studies from a business and systematic perspective. Moreover, the results are compared with the prior work.

Keywords: Software Engineering, Testing, Test Sequences, UML, Use Case

1. Introduction

Software testing is the process of ensuring the software conformance with its requirements. Such a process is a mandatory phase in each process model of the Software Development Life-Cycle (SDCL). Testing tasks require several pre-steps to be done as testing preparation. Testing preparation involves generating test strategies, test plans, test sequences, test suites, and test cases. All the testing preparation steps are important for successful testing.

Testing preparation is done with consideration of multiple points. Testing type is of the main points that require special test preparation. Testing types could be a black box, grey box, white box, or any other type. Moreover, the testing goal, which is related also to the testing type, should be considered in the testing preparation. For example, when the goal is to test the units' integration, the testing preparation will be different than testing a single unit of software.

In this research, the focus is on the test sequences that are used in software testing. Test sequences are certain actions that are ordered to be followed sequentially to test software [1]. Software testers use test sequences to prepare testing tasks and testing requirements. Generating test sequences is a costly process in terms of time, effort, and money.

UML diagrams are used in software testing for the test sequences' generation process. As the UML diagrams are generated in the analysis phase to identify the software specifications, developers can benefit from them to easily generate the test sequences as well. Some of the UML diagrams need to be pre-processed before using them in the generation of the tests. For example, when a UML diagram is used and its information is not enough, it can be combined with other UML diagrams to extract more information.

Test sequences generation can also support the process of generating the test suites. The test suite is a collection of test cases [2]. With consideration of test sequences, the test cases can be generated easier and better. Test sequences in the test cases' generation helps in understanding the flow of data and how the software parts are related to each other.

Many points are considered as difficulties in generating test sequences using UML use case diagrams. In addition to the time and effort of the required analysis, it is an expensive task. Moreover, such a process has a high probability of conceptual and human errors.

In this paper, an approach is proposed to generate the test sequences using UML use case diagrams. The approach consists of several processes to be done to generate the test sequences. As input, UML use case diagrams will be used to be processed and converted into

E-mail: 20112417@stu.uob.edu.bh, mhammad@uob.edu.bh

activity diagrams. Each of the activity diagrams will be simplified and divided into small activity graphs, in such a way that each activity graph is representing exactly one test sequence. After that, the activity graphs are used as a source to extract the information and generate the tests, which is the last process in the approach.

The paper is organized in several sections. The next section covers the reviewed literature and the main differences between the prior work and the proposed approach. After that, the section of the proposed approach comes to explain in detail each step and sub-process to generate the test sequences. Then, the results are presented along with the related discussion. Finally, the paper is concluded by stating what has been done, limitations, and future work.

2. Literature review

UML activity diagram is one of the used models in the testing phase. The approaches in [3 and 4] are suggested to automate the test cases' generation from UML activity diagrams. The approaches were proposed to automate the test case generation. Other approaches in [5] and [6] are also proposed to generate test cases from UML activity diagrams. However, these approaches have been developed with consideration of the Activity Path Coverage (APC) criterion. APC criterion is involved to help in detecting the faults of loops in software programs. Linzhang et al. [7] proposed an approach of reusing the UML activity diagrams models with a gray-box testing method. Moreover, a prototype has been implemented to support the approach and prove the concept. Bhukya [8] suggested an approach to generate test cases from activity diagrams. Such diagrams are used to describe the behavior of the systems, which makes them useful in generating test cases process. Also, there is another approach developed in [9] to generate the test cases from the UML activity diagrams. However, the approach is based on graph transformation. Two graph grammars have been proposed for the generation process. The first graph grammar is to interpret the activity diagram into an intermediate form, and the second one is to generate the test cases. Kim et al. [10] proposed an approach for test case generating using the information of the activity diagrams. The activity diagrams are converted to directed graphs with explicit input and output. Then, the test cases are generated from the directed graphs. Another work [11] proposed an approach to examine the errors that may occur in deriving test cases from the activity diagram. There is another approach that has been presented in [12] that uses a genetic algorithm to generate test cases from UML activity diagram. Using extension theory, Liping et al. [13] proposed an approach also to generate test cases using UML activity diagrams.

The sequence diagram is another UML model that is used in the process of generating test cases. For example, Sarma et al. [14] proposed an approach to automate the test case generation from UML sequence diagrams. Such an approach does not require any modification or manual changes to be re-used. The First step is generating a Sequence Diagram Graph (SDG) from the UML sequence model. Then, the test cases are generated from the SDG, which is the same as the work in [15]. A similar approach is suggested by Nayak and Samanta [16] to automatically generate test cases from UML sequence diagrams. However, it requires some data to be retrieved from other UML diagrams i.e. class diagrams, to completely generate the test cases. Another work in [17] developed an algorithm to derive tests from a sequence diagram using NEG and ASSERT operators. The derived tests using such an approach are also in the form of sequence diagrams. Furthermore, another generation approach is proposed in [18], which also uses the UML sequences diagram. The approach is automating the test path generation from the UML sequence diagram and has been evaluated using a case study.

UML state diagram is also used in the test case generation process. Kim et al. [19] proposed an approach that uses UML state diagrams to generate test cases. The approach is based on coverage criteria, which is developed using the control and flow information of the UML state diagram. Furthermore, the approach proposed by Samuel et al. [20] is also using the UML state diagram to generate test cases. The approach is fully automated and no manual steps to be done. The generation process uses the logic of the data flow as basic information. Moreover, control information is also used in the approach. Another work in [21] and [22] proposed approaches that use the UML state diagram in test cases' generation. These approaches are depending on genetic algorithms in the generation processes.

Generating test cases can be done also from UML use case diagrams. An approach presented in [23] shows a way for generating test cases from the UML use case diagrams. The approach has been implemented and tested using a real project. The testing results show that the majority of the generated test cases using the proposed approach are the same as the actual ones. Another work in [24] presented a new approach that uses the UML use case specifications. The first step in the approach is to generate activity diagrams for each actor from the use case specifications. After that, the test generation process started to generate the test sequences.

There is another way of generating the test cases from UML models, which is combining the UML diagrams information. As an example, the approach in [25] is proposed in which it combines class, object, and state diagrams' information to generate test cases. The extracted information from the diagrams is compiled using Intermediate Format (IF). Then, the test cases are generated from the IF output. Moreover, the suggested approach in [26] shows that the collaboration of UML diagrams could assist in generating the test cases. The approach has been tested and the results proved that there is a possibility to generate the test cases from the software design and not from code. Over and above, another work in [27] represented a technique, which is based on combining UML sequence and state diagrams to generate the test cases. The technique is used to generate test cases to perform class and integration testing for object-oriented programs. Besides, there is another approach [28], which is combining a UML use case and sequence diagrams to generate the test cases. First of all, two graphs are constructed from use cases and sequence diagrams. Use case Dependency Graph (UDG) is created from the use case diagrams and Concurrent Control Flow Graph (CCFG) from the sequence diagram. Those graphs help in generating test cases when combining their information. Furthermore, Ghose et al. [29] proposed an approach that uses UML class and sequence diagrams for test case generation. The information from both class and sequence diagrams is integrated into the Variable Assignment Graph (VAG). Then, the test cases are generated from the VAG.

The UML behavioral models combination is a way of generating test cases. For example, the work in [30] represented an approach to generate test cases by combining UML sequence and activity diagrams. Both activity and sequence diagrams are of the behavioral UML models. The main feature of the approach is that the number of test cases is reduced and achieves at the same time the test coverage criteria.

Generating a UML use case diagram from the requirements is already resolved by many proposed approaches. For example, the work in [31] proposed an approach to generate the UML use case from requirements using natural language processing.

In this paper, the UML use case diagram is used to generate the test sequences but with different considerations. The first point that the proposed approach overcomes the complicated process of generating the test sequences of the prior work. Moreover, the test sequences that the proposed approach is generating are straight forward without going deep into the sub-tests, because they are already covered by the main one. Furthermore, path coverage is considered in the proposed approach.

3. Proposed Approach

The proposed approach consists of several processes, as shown in Figure 1. The input is the UML use case diagram, which is a basis in the approach. Each of the UML use case diagrams of software is analyzed in the first process to extract the needed information to create customized activity diagrams. The activity diagrams will be generated based on a set of rules. In the next process, an activity graph will contain a set of nodes that are represented sequentially according to the extracted information from the previously generated activity diagram. Then, the tests will be generated from the activity graphs. The next sections explain in detail each process of the approach with examples.

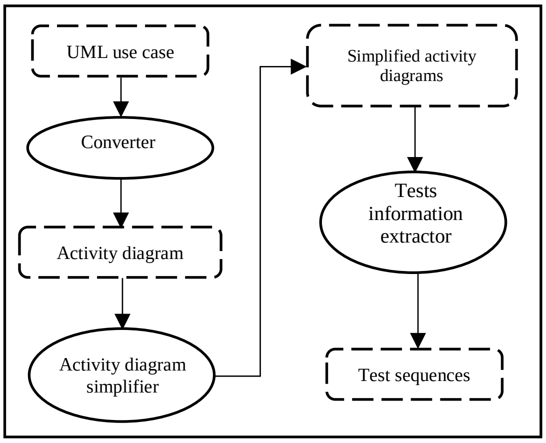
UML use case
Converter
Activity diagram
Activity diagram simplifier
Simplified activity diagrams
Tests information extractor
Test sequences

Figure 1. The proposed approach for generating test sequences from UML use case diagrams

A. *Convert UML use case into activity diagrams*

Each UML use case is converted to a customized activity diagram. Such an activity diagram contains sequential activities that should be done to perform a task, which is represented by the use cases in the UML use case model.

```
For each $A_i$
{
    create activity diagram
    make $A_i$ a starting point
    For each $RA_{i,j}$
    {
        convert $UC_j$ into an activity
        connect $A_i$ to the activity
    }
    For each R
    {IF R is equal to "includes"
        {convert the included UC into an activity
        connect the previous activity to the included activity }
    ELSEIF R is equal to "extends"
        {convert the extended UC into an activity
        connect the previous activity to the extended activity
        connect the previous activity to the endpoint}
    }
}
Connect all the activities to the endpoint.
```

Figure 2. Pseudocode of the proposed approach for converting a UML use case into an activity diagram

Figure 2 shows how the task of UML use case conversion into an activity diagram is performed using pseudocode and the definitions:

Ai = {Ai: A is actor and i is actor number}

UCj = {UCj: UC is use-case and j is use case number}

R = {r: r is a relationship between UC and another UC, and r ∈ {"extends", "includes"}}

RAi, j = {RA is a relationship between Ai and UCj}

As shown in Figure 2, for each actor, there is a customized activity diagram. The actor is represented by a starting point in the activity diagram. For each RAi, j, which is a relationship between the actor (i) and a use case (j), there is a line that connects the starting point (Ai) and the use case (UCj). However, for each R, which is either an “extends” or “includes” relationship, there is a different process. If R equals to “includes”, the previous activity will be connected to the included one. If R equals to “extends”, there will be a fork and join. In other words, the activity will be duplicated. One of the duplicated activities will be connected to the other activity that represents the extended use case.

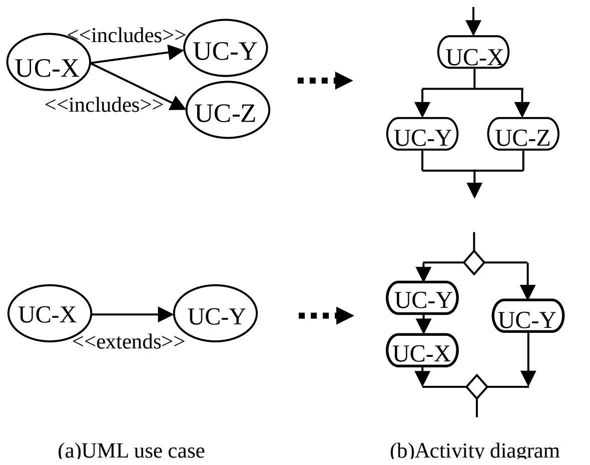

Figure 3. “extends” and “includes” relationships of UML use case diagrams

Determining the UML use case elements is the first step in the process. For each actor of the UML use case diagram, there is a customized activity diagram and the actors are converted to starting points. All the use cases, which are connected to the actor directly are converted to activities and connected to the starting point. For the relationships between the use cases, the process is different, as shown in Figure 3. If there is an “includes” relationship between two cases, the included use case is converted to activity and connected to the activity that includes it. For the “extends” relationship, the extended use case that is already depicted as activity in a previous step will be connected to the endpoint, as well as, to the use case that extends it after converting it into activity. Finally, all the activities should be connected to the endpoint.

Figure 4 and Figure 5 show an example of the conversion process from the UML use case diagram into an activity diagram. The use cases in the UML use case and their corresponding nodes in the activity graph are represented by (UC-#). All the possible situations of the use cases are depicted in Figure 4. The Actor has relationships with three use cases as shown in Figure 4. The use cases are in a normal use, an “includes” relationship, and the other one is in an “extends” relationship. The various situations are used to show how each of them is represented in the activity diagram, which is Figure 5. In the activity diagram, the actor is considered as a starting point and represents the first level. The starting point is connected to three activities (UC-1, UC-2, UC-5) through three edges. Those activities are representing the three use cases that are connected to the actor in Figure 4. As the use case (UC-1) in Figure 4 does not have any relationship with other use cases, activity (UC-1) in the activity diagram (Figure 5) is directly connected to the endpoint.

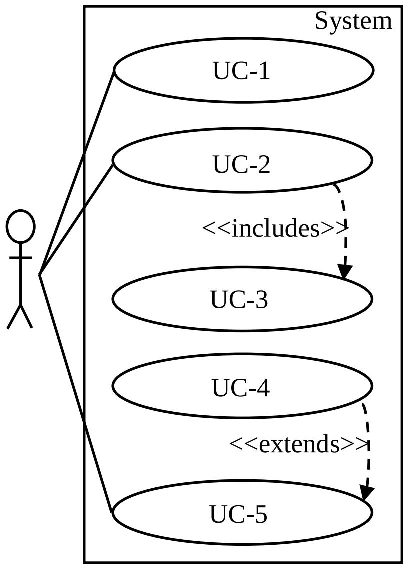

Figure 4. Example of UML use case diagram

The use cases that are in relationships with others are represented differently in the customized activity diagram as shown in Figure 5. The use cases (UC-2 and UC-5), as shown in Figure 4 are connected to other use cases through “includes” and “extends” relationships, which makes the representation of their corresponding activities connected to other activities as shown in Figure 5. For the “includes” relationship, the use case that includes the other use case is depicted first as an activity in the activity diagram, and connected directly through an edge to the included use case activity. For the “extends” relationship, the use case that extends the other one is depicted before the extended one in the activity diagram.

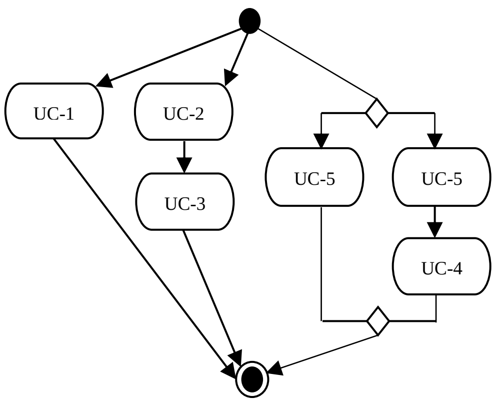

Figure 5. Activity diagram resulted from the conversion process of the UML use case diagram shown in Figure 4

B. Simplifying the activity diagrams

The second process in the approach is to simplify the activity diagram. The generated activity diagrams from the UML use case might be complicated. The complication appears because of certain reasons, such as the relationships between the actors and the use cases, and the relationships between the use cases themselves through "includes" and "extends". The simplification is intended to have sub-activity diagrams, which are represented by directed activity graphs. Each of the activity graphs is representing exactly one path of the previously generated activity diagram. As each of the generated activity graphs is equivalent to one path, the number of the activity graphs is the same as the number of paths of the original activity diagrams.

```
For each Activity_Diagram
    For each Edge in First_level
        Create Activity_Graph // Starting_point with an edge ONLY
            For each Activity in First_level{
                Connect the Activity to the Starting_point
            IF (Activity has EXACTLY ONE Edge){
                IF (Next_level == End_point)
                        Connect Activity to End_Point
                ELSE{
                        Connect to Activity of Next_Level
                        First_level <- Next_level}
            }ELSE IF (Activity has MORE THAN ONE Edge){
                WHILE(Activity has Edges AND Edges>1){
                    Duplicate the previously generated path
                    // Based on no. of edges
                    For each Duplicated_path
                        Connect (distinctly) Activity from Next_level
                            First_level <- Next_level}
                        GO TO CHECKING EDGES POINT}
                        }
```

Figure 6. Pseudocode of simplifying the activity diagrams by converting them into directed activity graphs

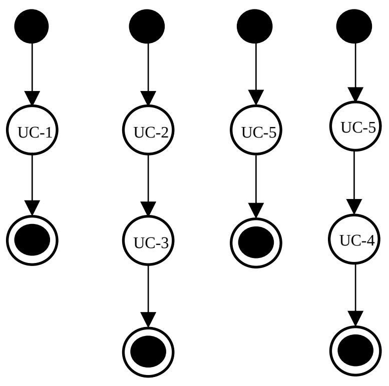

Figure 7. Activity graphs generated from simplifying the activity diagram shown in Figure 5

As shown in Figure 6, the path extraction is done by going through each edge of the activity diagram. For each edge, each activity is captured and depicted as nodes in the activity graph. Starting from the activities of the first level, a checking process is done to count the outgoing edges to decide the next step. If there is only one edge, a checking process is done to find if the next level is an endpoint or another activity. If there is more than one edge, the previously captured path is duplicated, and the activities of the next level (which are indicated by the number of the edges) are connected distinctly to each one of the duplicated paths. This process is repeated until the next level becomes an endpoint.

Figure 7 shows an example of the simplification process results. The activity diagram, which is represented in Figure 5, has four paths. So, the simplification process generates four directed activity graphs. In Figure 7, the first and last sub-activity graphs contain three nodes including the starting and endpoints. The second and third sub-activity graphs contain more than three nodes as there were "includes" and "extends" relationships between the use cases in the UML use case diagram.

C. Generating test sequences

The last process in the proposed approach is to generate the test sequences. The test sequences are generated through information extraction from the directed activity graphs. Each of the directed activity graphs is considered as one test sequence, and the software testers should go through all of the test sequences even if they are duplicated in each activity graph. Such information should be considered in the testing because there might be other users that are sharing a use case but taking different paths to the end. Moreover, the use cases' nodes (UC-#) should be extracted as functions to be tested in the testing process. The order of the functions is important and should be considered as well because it is representing how the functions are related to each other and some of them cannot be tested before the other ones.

Figure 8 shows the tests that are generated from the sub-activity graphs shown in Figure 6. All of the tests are for the user type (A). The first test has only one function to be tested, which is represented by (UC-1). The second test contains two related functions that should be tested, which are represented by (UC-2 and UC-3). The last test has also two related functions (UC-5 and UC-4) to be tested.

Test Sequences (TS#):
Actor A:
TS1: UC-1 → END
TS2: UC-2 → UC-3 → END
TS3: UC-5 → END
TS4: UC-5 → UC-4 → END

Figure 8. Generated test sequences from the UML use case diagram shown in Figure 4

4. Experimental evaluation

This section explains how the proposed approach is evaluated. The used case studies and the evaluation procedure is described in the coming subsections.

A. *Case studies*

The proposed approach has been applied and evaluated through nine projects as case studies. Two case studies are business models, and the rest are system models. The projects' requirements of the case studies are already converted into UML use case diagrams. They are published in public as UML use case examples. Choosing business models and system models in the evaluation are proof that the proposed approach is useful in the IT projects as well as the business projects, although the focus of the research is on the IT projects and system models.

TABLE I. DETAILS OF THE CASE STUDIES

Project	**Use-cases**	**Actors**	**Actor – Use-case relationships**	**Use-case – use-case relationship**
ATM	11	3	12	9
POS	9	3	3	7
FRIEND	7	2	4	6
E-library	12	2	7	5
Online shopping	20	5	9	11
Credit-card processing	6	3	14	2
Hospital m anagement	9	1	6	6
Restaurant	7	5	5	2
Check-in (Airport)	7	2	3	6

There certain information that has been considered from UML use case diagrams in the generation process. As shown in Table 1, for each project, the number of actors, use cases, relationships between actors and use cases, and relationships between the use cases are considered and counted. Such statistics are important to study if there are relationships between these numbers, which are shown in Table 1, and the number of the test sequences. Moreover, these statistics are also an indicator of the complexity level of the projects.

The first set of case studies is system UML use case diagrams. The Automated Teller Machine (ATM) system [32] is the first case study that is chosen to test the approach. The ATM system is designed and developed to assist the customer-machine interaction. The second case study is the Point-Of-Sale (POS) system [33]. Such a system is required to manage and record the sales transactions and handle the payment processes. Another used case study is First Responder Interactive Emergency Navigation Database (FRIEND) system [34]. The FRIEND system is one of the accident management systems. Furthermore, E-library [35] is also a case study that is used in the evaluation. E-library is a system that enables the readers to borrow books and return them at a specific time. Whenever there is a delay, the reader should pay a specific amount of money. Another case study is online shopping [35]. Through the online shopping system, customers can view items, add to cart, checkout, and pay. Credit card processing [35] is a subsystem of the online shopping system. Such a project is describing the process of credit card payments. Another system that is chosen as a case study is hospital management [35]. Such a system is used by the hospital receptionist to manage the patients' appointments and admissions. The other two case studies, which are restaurant and Check-in (Airport) [35] are business models. The Restaurant business model is created to manage the processes of buying meals and the related payments. However, the Check-in business model is to describe the check-in process at the airports.

As shown in Table 1, different numbers are varying between the nine projects for the actors, use cases, and relationships. The online shopping system has the highest number of use cases, actors, and relationships. However, the check-in has the least number of use cases and actors among the case studies. The case studies have been selected with consideration to the different complexity levels and sizes. Those differences are important to evaluate the proposed approach and how it will work in different situations.

B. *Experimental procedure*

To evaluate the proposed approach, the selected case studies are used to apply the approach. All the projects (case studies) have been processed by each process of the approach. In the beginning, for each project, the UML use case diagrams are converted into customized activity diagrams as a first step. Then the simplification process is done to generate directed activity graphs. Finally, the needed information is being extracted to generate the test sequences.

As an evaluation of the proposed approach, the results after applying the approach to the case studies are compared with the other approaches' results. The other approaches were considering also UML diagrams in generating the test sequences, which makes the comparison applicable to our approach.

C. *Results*

The approach has been evaluated by applying the proposed approach to the case studies. First of all, the functional requirements have been processed to generate the UML use case. Then, the UML use case diagrams have been converted to activity graphs using a set of rules. The activity diagrams are simplified before using them by generating directed activity graphs. In the end, the information has been extracted to generate the test sequences.

TABLE II. GENERATED TEST SEQUENCES

Project	Business/ System	No. of generated test sequences
ATM	System	24
POS	System	8
FRIEND	System	15
E-library	System	11
Online shopping	System	25
Credit-card processing	System	15
Hospital management	System	10
Restaurant	Business	7
Check-in (Airport)	Business	9

TABLE III. COMPARING THE PROPOSED APPROACH WITH THE PRIOR WORK

Prior work	**Braind et al. [36]**	**Fr¨ohlich et al. [37]**	**Ryser et al. [38]**	**Hartmann et al. [39]**	**Tiwari et al. [24]**	**Proposed approach**
Used model	Use case, class, and seq.	State-chart	State-chart and dependency-chart	Activity diagram	Use case textual description and diagram	use case diagram
Case Study	**No. of generated test sequences**					
ATM	43	30	32	30	32	24
POS	39	26	30	26	30	8
FRIE-ND	39	30	32	30	32	15

The results after applying the approach on the nine projects are shown in Table 2. The highest number of test sequences was for the online shopping system. Such a high number was expected because online shopping has high numbers for actors, use cases, and relationships between actors and use cases, and between the use cases themselves. The opposite is also right and expected for the system that has the least number of test sequences, which is the restaurant. Therefore, we can notice that there is a relationship between the numbers of actors, use cases, relationships, and the number of test sequences. Since we have a high number for the UML use case elements, the high number, we will get a high number of test sequences.

Certain case studies have been used in the prior work for the same problem. Table 3 shows the results of the proposed approach in comparison with other previously proposed approaches. The generated test sequences of the proposed approach are 24, 8, and 15 for ATM, POS, and FRIEND system, respectively

D. Discussion

By observation, there are important points that resulted from the experimental evaluation. First of all, although the number of actors and use cases affecting the number of the test sequences, the number of the between the use cases themselves are playing the main role in increasing the test sequences, especially the “extends” relationship is always increasing the generated test sequences. In each occurrence, the “extends” relationship is always adding one more test sequence, due to the optional feature of performing the extended use case. In general, the number of actors, use cases, and the relationships are related to the number of test sequences positively.

The approach that has been proposed in this paper, generated the least number of test sequences compared with the prior work as shown in Table 3. The other approaches, in general, have different numbers of test sequences, because of the different UML models they used and combined. The approach of Braind et al. [28] generated the highest number of test sequences. However, despite our proposed approach, Hartmann et al. [31] proposed an approach, which produced the lowest number of the test sequences. However, the number of the generated test sequences cannot be an indication of the approach's correctness, because of the different UML models. Different UML models can generate different test sequences, because of the coverage differences of the paths. Sometimes, an approach generates, for example, 10 test sequences, and another approach generates 15 test sequences. Such results do not mean that the second approach is better because of the higher number of test sequences, but it might be an indicator that the first approach is covering more functions under a single test sequence.

The proposed approach, as stated in Table 3, generated the lowest number of test sequences compared with the other approaches. Comparing the different approaches that are proposed with different UML models is important because we need to know the results of each of them to combine some UML models to get better results. The main reason for getting the small number of test sequences is depending on the information that is only shown in the UML use case diagram without the textual description. Diagrams are always kept in hands because they are used to explain the requirements specifications more than their textual description. Hence, the textual description sometimes is not being updated as much as the diagram itself. For such reason, only the UML use case diagram has been considered in the proposed approach. Another reason is that when testing the use cases, which shown in the diagram, all the possibilities should be covered, whether it is mentioned in test sequence notation or not. The relationships in the UML use case diagrams are also one of the factors that affect the results. Due to the relationships between the use cases, there might be some situations that need to be depicted well in the UML use case diagrams to avoid any errors in generating the test sequences. Moreover, the relationships are being analyzed in different ways from many points of view, which makes the UML use cases for the same system different from an organization to another.

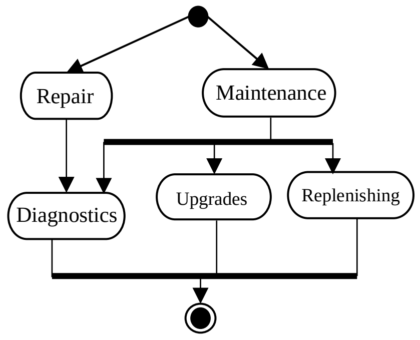

Figure 9. Activity diagram generated from the ATM UML use case diagram

The test sequences that have been generated using the proposed approach, are generated with consideration of path coverage. Path coverage is achieved from the UML use case point of view, which is covering the different situations that are caused by the relationships between the actors and the use cases and between the use cases themselves. Figure 9 shows one of the activity diagrams that has been generated from the UML use case of the ATM system. The activity diagram is for the actor “ATM Technician”. The “ATM Technician” has two activities, “Repair” and “Maintenance”, based on the UML use case diagram of the ATM system. The two activities are including the “Diagnostics” activity. However, only the “Maintenance” activity includes two activities, “Upgrades” and “Replenishing”. The activity diagram has been simplified by converting it into an activity graph, as shown in Figure 10. Each of the activity graphs is corresponding to exactly one test sequence. The nodes “A1, A2” are respectively representing “Repair, Maintenance” activities, and “A3, A4, A5” are representing “Diagnostics, Upgrades, Replenishing”, respectively. The activity graph is used to extract the information to generate the test sequences shown in Figure 11. Figure 11 shows that there are four test sequences for the actor “ATM Technician” generated from the UML use case diagram. Such an example shows that there is no relationship between the number of the use cases of the UML use case diagram and the number of the generated test sequences.

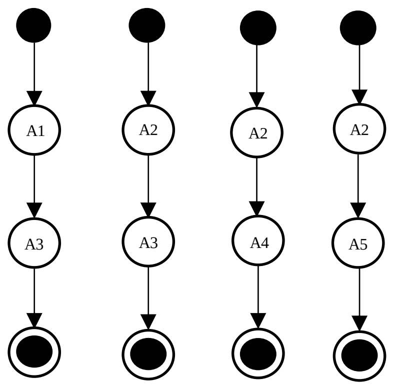

Figure 10. Generated activity graphs from the activity diagram shown in Figure 9

Test Sequences (TS#):
Actor: ATM Technician
TS1: Repair → Diagnostics →END
TS2: Maintenance → Diagnostics → END
TS3: Maintenance → Upgrades → END
TS4: Maintenance → Replenishing → END

Figure 11. Generated test sequences from the activity graphs shown in Figure 10

5. TEST SEQUENCES, TEST CASES & TEST SUITES

Generating test sequences is also assisting in creating the test suites. Test suites are a collection of test cases. To create test suites, the test cases should be created and grouped based on the testing requirements. Using the test sequences, which are sequentially ordered software functions, the test cases can be grouped. Moreover, creating the test cases with consideration of the test sequences will assist in making relationships between the test cases’ groups. Hence, the test cases’ generation will be easier and better in terms of understanding the data flow between the software parts.

Figure 12 explains our proposed approach from the angle of how the test sequences are related to the test cases and the test suites. Assuming that the use cases in the UML use case diagrams are the functions to be tested, and named with “UC#”, the test cases with “TC#” and the test sequences with “TS#”, as shown in Figure 12. To create a test suite, we have to create a collection of test cases. In the process of generating the collection of test cases, the test sequences should be considered to understand how the data flows from a software part to another. Sometimes, the test sequences are about testing only one part, and sometimes they are generated to test multiple parts sequentially.

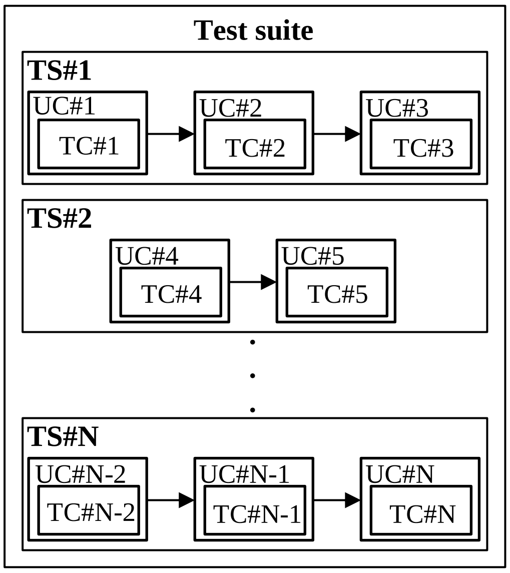

Figure 12. The relationships between the test sequences, test cases, and test suites

The proposed approach can provide the software engineers the ability to automate the SDLC processes. An approach has been proposed in [31] to study the process of generating the UML use case from the software requirements, which can be considered as a pre-step for the approach in [40] that we extended the work of it in this paper. Moreover, some surveys categorize the testing tools, such as the work in [41 and 42], which is also a helpful step to support the software testers' decisions. Such endeavors can assist the researchers in providing a complete automated process.

6. Conclusion and Future Work

In this paper, the proposed approach is aimed to generate the test sequences from the software UML use case diagrams. Such an approach helps the software engineers to make use of the UML diagrams not only for the requirements analysis but also for other SDLC phases. The proposed approach consists of several processes starting from converting the UML use case into customized activity diagrams. Then, the second process came to simplify the activity diagrams by converting them into directed activity graphs. In the end, the needed information is extracted from the directed activity graphs to generate the test sequences. The approach has been evaluated through nine case studies. The results have been compared with other approaches from the prior work that use UML models to generate the test sequences. Moreover, in this paper, the relationship between the test sequences, test cases, and test suites has been explained. Considering the test sequences while creating the test suites will ease the testing process in terms of understanding the data flow between the software artifacts.

Some issues limit the work of the proposed approach. The main issue is the order of the use cases. Such a point affects the number of the test sequences because sometimes some use cases should be tested before and after a certain event. Furthermore, some use cases should be tested before or after the occurrences of the event(s).

To improve the approach, there are several points to be considered. As for future work, the order of the use cases should be involved in the approach to cover all the situations. Moreover, more ways of generating the test sequences from the UML models are to be discovered to make use of the UML in the different phases of the SDLC.

References

[1] Matlab, "Introduction to test sequences," Mathworks, 2020. [Online]. Available: https://www.mathworks.com/help/sltest/ug/introduction-to-test-sequences.html. [Accessed 15 April 2020]

[2] IBM, "Test case and test suite overview," IBM, 2020. [Online]. Available: https://www.ibm.com/support/knowledgecenter/SSYMRC_6.0.1/com.ibm.rational.test.qm.doc/topics/c_testcase_overview.html. [Accessed 15 April 2020]

[3] Mingsong, Xiaokang and Xuandong, "Automatic test case generation for UML activity diagrams," in *Proceedings of the 2006 international workshop on Automation of software test - ACM*, 2006.

[4] D-Xu, H-Li and CP-Lam, "sing adaptive agents to automatically generate test scenarios from the UML activity diagrams," in *12th Asia-Pacific Software Engineering Conference (APSEC'05) - IEEE*, 2005.I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[5] Thanakorncharuwit, Kamonsantiroj and Pipanmaekaporn, "Generating test cases from uml activity diagram based on business flow constraints.," in *Proceedings of the Fifth International Conference on Network, Communication and Computing - ACM.*, 2016.

[6] Kundu and Samanta, "A novel approach to generate test cases from UML activity diagrams," *Journal of Object Technology*, vol. 8, no. 3, pp. 65-83, 2009.

[7] Linzhang, Jiesong, Xiaofeng, Jun, Xuandong and Guoliang, "Generating test cases from UML activity diagram based on gray-box method," in *11th Asia-Pacific software engineering conference - IEEE*, 2004.

[8] Bhukya, "Test Case Generation using UML Activity Diagram & Composite Structure Diagram," in *Doctoral dissertation*, 2015.

[9] Hettab, Chaoui and Aldahoud, "Automatic test cases generation from UML activity diagrams using graph transformation," in *6th ICIT*, 2013.

[10] Kim, Hyungchoul, S. Kang, J. Baik and Inyoung-Ko, "Test Cases Generation from UML Activity Diagrams," in *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007) - IEEE*, 2007.

[11] Felderer and Herrmann, "Manual test case derivation from UML activity diagrams and state machines: A controlled experiment," in Information and Software Technology, 2015.

[12] Jena, Swain and Mohapatra, "A novel approach for test case generation from UML activity diagram," in *International*

Conference on Issues and Challenges in Intelligent Computing Techniques - IEEE, 2014.

[13] L. L, L. X, H. T and X. J, "Extenics-based test case generation for UML activity diagram," in *Procedia Computer Science*, 2013

[14] Sarma, Kundu and Mall, "Automatic test case generation from UML sequence diagram," in *15th International Conference on Advanced Computing and Communications (ADCOM 2007) - IEEE*, 2007.

[15] Dhineshkumar, "An approach to generate test cases from sequence diagram," in *International Conference on Intelligent Computing Applications - IEEE*, 2014.

[16] Nayak and Samanta, "Automatic test data synthesis using uml sequence diagrams," *Journal of Object Technology*, vol. 9, no. 2, pp. 75-104, 2010.

[17] L.MS and S. K, "Deriving tests from UML 2.0 sequence diagrams with neg and assert," in *Proceedings of the 2006 international workshop on Automation of software test - ACM*, 2006.

[18] P. S. S and M. P. S. K, "Test path generation using uml sequence diagram," International Journal of Advanced Research in Computer Science and Software Engineering, vol. 3, no. 4, 2013

[19] YG-Kim, HS-Hong, DH-Bae and SD-Cha, "Test cases generation from UML state diagrams," in *IEE Proceedings-Software*, 1999.

[20] S. P, M. R and B. A. K, "Automatic test case generation using unified modeling language (UML) state diagrams," IET Software, vol. 2, no. 2, pp. 79-93, 2008

[21] S. M, S. A and K. R, "Generation of improved test cases from UML state diagram using genetic algorithm," in *Proceedings of the 4th India Software Engineering Conference*, 2011

[22] D.-a. C, D. K, H. A and S. T, "Test data generation from UML state machine diagrams using gas," in *International Conference on Software Engineering Advances (ICSEA 2007) - IEEE*, 2007

[23] Hasling, Goetz and Beetz, "Model based testing of system requirements using UML use case models," in *1st International Conference on Software Testing, Verification, and Validation - IEEE*, 2008.

[24] Tiwari and Gupta, "An Approach of Generating Test Requirements for Agile Software Development," in *Proceedings of the 8th India Software Engineering Conference - ACM*, 2015.

[25] Cavarra, Crichton, Davies, Hartman and Mounier, "Using UML for automatic test generation," in *Proceedings of ISSTA*, 2012.

[26] Abdurazik and Offutt, "Using UML Collaboration Diagrams for Static Checking and Test Generation," in *International conference on the unified modeling language - Springer*, Berlin, 2000.

[27] Sokenou, "Generating Test Sequences from UML Sequence Diagrams and State Diagrams," in *GI Jahrestagung (2)*, 2006.

[28] Swain, Mohapatra and Mall, "Test Case Generation Based on Use case and sequence diagrams," *International Journal of Software Engineering*, vol. 3, no. 2, pp. 21-52, 2010.

[29] G. Sudipto, R. France, C. Braganza, N. Kawane, A. Andrews and O. Pilskalns, "Test adequacy assessment for UML design model testing," in *14th International Symposium on Software Reliability Engineering - IEEE*, 2003.

[30] Swain, S. Kumar and D. P. Mohapatra, "Test case generation from Behavioral UML Models," *International Journal of computer applications*, vol. 6, no. 8, pp. 5-11, 2010.

[31] Abdulkarim Hamza Z., Hammad M., "Generating UML Use Case Models from Software Requirements Using Natural Language Processing," in Proceedings of International Conference on Modeling Simulation and Applied Optimization (ICMSAO) – IEEE, In press.

[32] IBM. Object-Oriented Analysis and Design with UML2 and Rational Software Modeler. IBM Corporation, 2006.

[33] C. Larman. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, 3/e. Pearson Education India, 2012.

[34] B. Bruegge. Object-Oriented Software Engineering: Using Uml, Patterns and Java, Pearson, Second Edition edition, 2009.

[35] UML diagrams, "uml-diagrams.org," 2020. [Online]. Available: https://www.uml-diagrams.org/use-case-diagrams-examples.html. [Accessed 15 April 2020]

[36] L. C. Briand and Y. Labiche. A uml-based approach to system testing. In Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools, UML'01, pages 194–208, London, UK, Springer-Verlag, 2001.

[37] P. Fr¨ohlich and J. Link. Automated test case generation from dynamic models. In Proceedings of the 14th European Conference on Object-Oriented Programming, ECOOP '00, pages 472–492, London, UK, Springer-Verlag, 2000.

[38] J. Ryser and M. Glinz. A scenario-based approach to validating and testing software systems using statecharts. In In 12th International Conference on Software and Systems Engineering and their Applications (ICSSEA'99), page 7, 1999.

[39] J. Hartmann, M. Vieira, H. Foster, and A. Ruder. A uml-based approach to system testing. Innovations in Systems and Software Engineering, 1(1):12–24, 2005.

[40] Hamza, Z. A., & Hammad, M., "Generating test sequences from the UML use-case diagram". In *2019 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, pp. 1-6. IEEE, 2019

[41] Hamza, Z. A., & Hammad, M., "Testing Approaches for Web and Mobile Applications: An Overview". *International Journal of Computing and Digital Systems*, *9*(4), 657-664. 2020.

[42] Hamza, Z. A., & Hammad, M. Web and Mobile Applications' Testing using Black and White Box approaches - IET. 2019.

Zahra Abdulkarim Hamza is currently studying M.Sc. Software engineering at University of Bahrain. She is Graduated from University of Bahrain, B.Sc. Computer Science and got the best senior projects award - 3rd place, 2017, for project titled: "Automatic Diacritization for Arabic Text using Voice Recognition Technique". She Worked at Batelco for two years in Business Intelligence, Corp. Apps and Enterprise Data Warehouse.

Mustafa Hammad is an Associate Professor in the Department of Computer Science at the University of Bahrain and Mutah University. He received his Ph.D. in Computer Science from New Mexico State University, USA in 2010. He received his M.Sc. degree in Computer Science from Al-Balqa Applied University, Jordan in 2005 and his B.Sc. in Computer Science from The Hashemite University, Jordan in 2002. His research interests include machine learning, software engineering with focus on software analysis and evolution.