International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878 (Online), Volume-9 Issue-1, May 2020

Smart Test Case Quantifier Based on Mc/Dc Coverage Criterion with Uml Sequence Diagram

S. Shanmuga Priya

Abstract*: Software testing is an inevitable phase in the Software Development Life Cycle (SDLC). It plays a vital role in ensuring the quality of the product. Testing can be performed on requirement, design and code. Traditional test case generation approaches majorly focuses on designing and implementation models, as a huge percentage of errors in the software are caused due to the lack of understanding in the early phases. So, it's advisory to follow testing in the initial phase in SDLC as most of the errors/faults can be eliminated and can be prevented without disseminating to the next phase. So, testing must not be secluded to a single phase alone in SDLC. In software testing, test case generation is the fundamental and challenging part. The major purpose to design test case is to create set of tests that are valuable in validation. Due to the increase in software size, it's not quite advisable to have the choice as manual testing to test the code as its error prone, complex and time consuming. Automation of testing would help the tester to test effectively and timely. Applications of new techniques improve the test process and cut down the tester's effort. The proposed work presents a sandwich of black box and white box testing. Design-based testing, a black box approach is implemented, that aids in fixing the errors at initial phase. Unified Modeling Language (UML) Use case diagram, Activity diagram and Sequence Diagram are considered for designing. The code-based generation of test cases, is a white box testing approach, test coverage criteria Modified Condition/Decision Condition (MC/DC) has been used to ensure the maximum coverage of the code during testing phase for generating the test cases. The objective of this paper is to automate the generation of minimal number of test cases required to test a system with maximum coverage by removing the redundant test cases using MC/DC criterion. This present an idea to the beginners of the testing about minimal test cases they necessitate for testing their system.

Keywords : Test case generation, Unified Modeling Language (UML), Use Case Diagram, Activity Diagram, Sequence Diagram, MC/DC Coverage criterion.

I. INTRODUCTION

Due to the advent of technologies, software development has become complex. Software development starts with customer requirement and has a need that the built software product must meet it. The first step in developing a project is the requirement specification. The requirement definition document is defined in the natural language which could be easily understood by the customer, and it specifies the requirements in technical language that is used by the developer. The requirement specification must be unambiguous and consistent. Any formal language is used to map the requirement definition document into requirement specification. Usually, the formal language is not understood by the client, it must be ensured that all the requirements are mapped into specification. Once the requirement specification is completed, it is followed by the software design. For large software projects, high level design called the architectural design, which decides how the product must be structured into modules, that is to be implemented as per the requirements is carried out initially. A more detailed design gives how the product works, defines the internal structures and the interaction that occurs between the modules. The software design is then followed by the implementation or coding. According to the specified software architecture, the modules are implemented. Once the software requirements are defined, mapped into specification, architectural design, detailed design, coding and integration of modules are done, then the product is validated in order to ensure that it meets the needs of the customer and the software product works as intended. During code testing, test cases are to be written in order to verify whether the software performs the basic operations as desired. Generating test cases based on code, helps in capturing the errors at implementation level, this helps in eliminating the misunderstanding and removes the errors in the code [1]. Unit testing is carried out to ensure that each and every modules works properly. Integration testing is performed to ensure that the modules interact with each other properly. System testing is done and then acceptance testing is carried out to check whether the product is accepted by the customer. Mostly in a typical application development, the software testing is isolated to the end of the Software Development Life Cycle. Even though there are vast testing techniques to derive test cases, still researchers are working in this area. The phases of SDLC are: i) Requirement Analysis and Gathering, ii) Designing, iii) Coding, iv) Testing and v) Operation and Maintenance. Testing should be accompanied in each of these phases. If testing is isolated as a single phase late in the cycle, errors in the requirement specification or design may incur exorbitant costs. Not only must the original error be corrected but the entire structure built upon it must also be changed. Therefore, testing should not be isolated as an inspection activity. Software is usually subjected to several types several types of verification and test.

Manuscript received on April 02, 2020.
Revised Manuscript received on April 15, 2020.
Manuscript published on May 30, 2020.
* Correspondence Author
S. Shanmuga Priya*, Department of Computer Science and Engineering, New Horizon College of Engineering, Bangalore, India. Email: priya_soundarajan@yahoo.co.in

Various types of software testing techniques have been developed till date, but which type of testing technique will be suitable and sufficient for checking a particular product, in which phases of Software Development Life Cycle (SDLC) is not yet clear.

The problem here is to identify the testing techniques which can be applied throughout the phases of software development life cycles.

One such testing technique has been identified and presented in this paper which is a combinational approach of black box and white box techniques for testing the Medical Consultation System application. In black box testing, the internal structure of the item being tested is unknown to the tester and in white box testing, the internal structure is known. In a software application counting millions line of codes, the white box testing techniques alone require lot of time to cover all the selected test criteria. A solution to this problem is the automation of testing. In general, automating test is easier to implement on white box testing techniques than black box techniques. In white box techniques, the test cases to be generated rely on the code itself, and thus it is possible to use code analysis techniques to extract the necessary information needed to generate the test cases and the test data.

II. RELATED WORKS

Many works are available in the literature review that discuss about requirement analysis, software design, and medical diagnosis [2] [3] [4] [5]. The various approaches used for generating test cases such as scenario based test case generation approach [6], model based test case generation approach, and genetic based test case generation approach [7] [8] [9].

As the software developer and software tester cannot test everything in software (exhaustive testing), a combinatorial design of test can be done that helps in identifying the minimum number of test that can be done with maximum coverage. In survey, it is found that there are different techniques available for generating test cases using UML models like use case diagram, sequence diagram, activity diagram, etc. Shanmuga Priya et. al., [10], has given a survey on the usage of various UML diagrams in generating the test cases. The process of generating test cases from design helps in discovering the problems at the early phase of the software development thus saves the time and the cost of resources involved during the SDLC.

Ranjita Kumari Swain et al. [11] proposed a method for generating test cases using activity diagram by considering a case study of Soft drink Vending Machine (SVM). The authors first constructed an activity diagram, from which they derived the required information for forming an activity flow graph. From the graph, different control flow sequences were identified by traversing it using depth first traversal technique. The various activity paths were generated from which the test cases were generated using coverage criteria of activity path.

Sophine Clachar and Emanuel S. Grant [12] proposed an approach to design UML Class Diagram of a safety critical system. They remodelled it using formal methods. Sanjai Rayadurgam and Mats P. E. Heimdahl [13] proposed a method to automatically generate test cases to the MC/DC structural coverage criterion. They have demonstrated how model checker could be used to generate test sequences. Vikash Panthi and Durga [14] proposed automatic test case generation using sequence diagram. The proposed approach is a graph based one in which using sequence diagram. Constructed graph was traversed using Depth First Search technique. Conditional coverage criterion was used and converted it into code from which the test cases were derived.

Ranjita Swain et al. [15] proposed a methodology to generate test case from UML State Chart diagrams. The procedure followed to generate test case was carried out by constructing state chart diagram for an object, traversed state chart diagram. They used conditional predicate coverage and the selected predicates from the state chart were transformed into source code from which the test cases were generated. They used transition path coverage to minimize the number of test cases.

Puneet Patel and Nitin N. Patil [16] proposed automated test case generation from Activity Diagram. Test suite was generated by following activity path coverage criterion by covering loop faults. The approach followed in their work was by constructing an activity diagram for ATM. Santhosh Kumar Swain and Durga Prasad Mohapatra [17] proposed a method to automatically generate test cases from behavioral UML models. UML activity diagram and sequence diagram were considered in their approach. They considered case studies of Automatic Teller Machine (ATM) for withdraw operation for exploring the work. Their approach was three fold process in which at first a Model Flow Graph (MFG) was generated from sequence diagram and activity diagram separately. Secondly, test sequences were generated by taking pre-conditions and post-conditions into account. Third, generated test cases from the generated test sequence by satisfying message-activity path test adequacy criteria with no redundant test cases obtained.

Monalisa Sarma et. al. [18] presented an approach to generate test cases from UML sequence diagram. PIN authentication sequence diagram of an ATM system was considered in their proposed work. The UML sequence diagram was converted into Sequence Diagram Graph (SDG). Information for specifying input, output, pre-condition and post-condition of a test cases were extracted using data dictionary in Object Constraint Language (OCL) 2.0, use case and were stored in SDG. By following their proposed algorithm, TestSetGeneration, using message path coverage criteria which take SDG as input, produced test suite as output. Namita Khurana [25] et. al., has presented an approach that uses state chart diagram, converted to a state chart graph, the sequence diagram being converted to sequence graph. Later, both the graphs were converted into System Testing Graph (SYTG). Genetic algorithm was applied to generate test cases and optimize it. But the work does not mention the results clearly.

Syed Asad Ali Shah et al. [26] presented an automated test case generation technique by using UML sequence and class diagram without any intermediate form of conversion.

The UML sequence and class diagrams were converted into XML format by using Visual Paradigm. C# code was used for reading the XML file. Meiliana et. al. [27] presented a modified DFS algorithm that generated automatic test cases using UML sequence diagram and activity diagram. They also showed experimental results, by deriving test cases based on activity diagram, sequence diagram and System Testing Graph (SYTG). Priya Kamath et. al. [28] has considered UML Activity diagram for generating the test cases and proposed a method for generating optimized test cases.

III. PROPOSED SYSTEM

The proposed system includes the various phases of the SDLC and it concentrates on the i) Requirement Gathering and Analysis Phase, ii) Designing Phase, iii) Implementation Phase and iv) Testing Phase. The proposed work considers medical consultation system as a case study (application) for which the test cases are to be derived using UML Sequence Diagram designed from the requirements which is a black-box testing strategy in the design phase. Later in the implementation phase, the source code for the considered work (Medical Consultation System) is developed. During testing phase, the white-box testing is performed on the available code using Modified Condition/Decision Condition as the testing criterion to generate minimum number of test cases (quantity) required for testing the code. The modules involved in each phase are depicted in the figure 1:

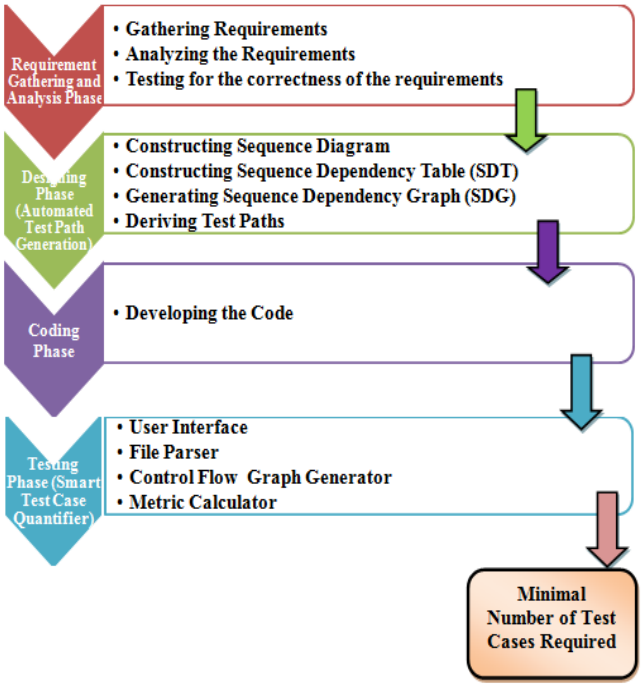

Fig 1. SDLC activities of the proposed system

Figure 2 shows the steps involved in designing phase to testing phase.

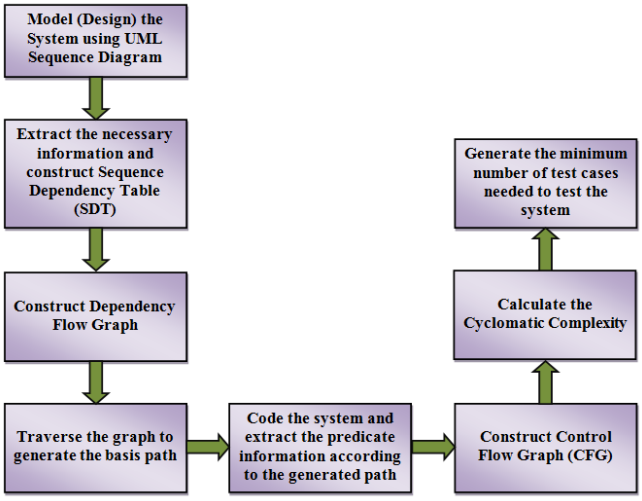

Fig 2. Steps involved in designing phase to testing phase

A. Implementation of the proposed system

Requirement Gathering and Analysis Phase

As medical expert systems are domain dependent system, the requirements analysis must be carried out before software design. The requirement gathering and analysis phase defines the requirements of the application to be developed, independent of how these requirements will be accomplished by defining the problem that the customer wants to solve. The problem statement for the considered case study is described below.

Problem Statement for Medical Consultation System

A computerized medical consultation system has been designed which can be considered as an information guidance/medical assistance system. The proposed system gives a convenient way for handling the patient, in case, of non-urgent consultation, not severe or chronicle disease is with the patient. This system is helpful for the patient those who needs to take medical consultation as a routine service, if required, and suggested by the doctor that can be assisted via Skype or email or through telephones (telemedicine), without the patient stepping to the doctor's door. The patients can e-communicate with the doctor with a prior e-appointment. With this system, the care taker of the patient (on behalf of the patient) or the patient himself/herself can access the service from any computer, at any location with Internet access. The patient can login to the system with the PatientID provided by the system. Through the confidential account, the patient can get access to the system by providing valid username and password. On successful login, the patient can produce the details such as the last consulted date (if required), doctor to whom they wish to consult (mandatory field) based on the criteria, the symptoms they have (mandatory field) and can scheduled an e-appointment with the doctor. The patient can also view the profile, update the profile and search for the required details. The requested doctor based on his/her availability; the doctor can refer to the patient's previous history from the database and get access to the details required.

On finding the adequate details with respect to the case history, and if the symptoms are mild, the doctor can get connected with the patient at the appointed time through any of the connecting assistance stated above, can have a face to face communication online and can recommend treatment (e-prescription) to the patient for cure. In case, if the doctor finds the history is inadequate for diagnosing, the doctor can recommend the patient to take additional prescribed tests and submit the reports in person or update through the consultation system for further proceeding, that may depend upon the severity of the disease. If the requested doctor is not available, a notification can be sent to the patient stating the non-availability of the doctor, and can be e-recommended with another set of doctor list that are available. The patient can be given a choice to pick another doctor from the list, if desired and the same e-consultation procedure describe above will be followed. The considered application helps the patients to communicate with the doctors' online and saves their time. Doctors can have immediate access to the patient's history and can make an accurate diagnosis for providing correct treatments on-time.

The functions that *patient* can perform in the system are:

- ✓ Search for doctor,
- ✓ Make online appointment,
- ✓ View the health care details such as their history, reports, expenses spared, doctor's diagnosis, medicines prescribed, lab reports

The functions that *doctor* can perform in the system are:

- ✓ Give appointments to patients,
- ✓ View patient's history,
- ✓ Give e-prescription,
- ✓ Can update the patient details pertaining to diagnosis.

The functions that *administrator* can perform in the system are:

- ✓ Add/delete users (patient details, doctor details)
- ✓ Grant permission to doctors,
- ✓ Generate and view reports,
- ✓ Views the complaints of doctors and patients and takes necessary actions.

B. Design Phase - Automated Test Path Generation

The design phase starts with the requirement document as input. The requirements are mapped into architecture which defines the components, their interfaces and behaviors. The pictorial representation of models helps in visualizing, specifying and constructing the complex system by depicting the structure as well as the relationships among them. In this context, Unified Modeling Language (UML) facilitates a greater and better understanding of the processes and activities involved in developing the system [19]. The UML diagrams always have a unique place in representing the analysis, design and implementation phases in SDLC [20]. Model-based method for generating the test cases mainly utilizes use case diagram, activity diagram or sequence diagram [21]. The proposed system considered these entire three diagrams for generating the test cases. However, a major focus is on using sequence diagram, whereas the other two diagrams helps in getting a clarification on how to proceed.

Use Case Diagram

Among the different UML diagrams, the use case diagrams play a key role in software development as the other diagrams like UML activity diagrams, class diagram, sequence diagram, collaboration diagram, are basically generated from the base of use case flows in the use case diagrams [22]. In the analysis phase of a SDLC, use case diagrams are used for analyzing the requirements, thus resulting in representing the relationships between the different processes involved in the project under development. The UML use case model gives the black-box view of the system, by hiding the implementation decisions. The use case model [23] represents systems functions and the environment. UML use case model aids in deciding the number of modules that has to be developed, identifying the actors and their responsibilities, producing the relationship between the actors and the use cases (activities performed by the actor on the system), and used for designing the backend, database.

Steps involved in creating use case diagram are as follows:

1. Read the requirement specification and tokenization of sentences.
2. Prepare the list of nouns that represents the actors and the list of verbs, usually denotes the action performed by the nouns on the system, which can act as the use case.

Step 2 is repeated until all the nouns and verbs in the specification are identified.

Figure 3 shows the steps involved:

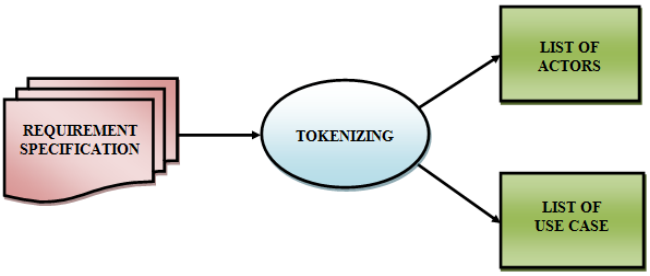

Fig 3. Steps involved in producing actors and use cases

As a result of the above activity, the table shows the list of nouns (actors) and list of verbs (use cases) identified are:

List of actors identified (Noun)	List of Use cases identified (Activities)
Patient	Login
Doctor	Provide personal details
Administrator	Update personal details
	Search details
	View details
	Pick a doctor
	View patient's history
	Schedule an e-appointment
	Make diagnosis and update details
	Recommend e-prescription / treatment

Add/Delete user

Grant access to the user and generate report

View complaints and take action

Brief Use case Description

Use case Name	Description
Login	The patient/doctor/administrator has to sign-in in order to perform the desired activities
Provide personal details	Every registered patient can provide the details to create his/her own profile containing personal details
Update personal details	The patient can update his/her own profile also can update the report health record, x-rays, test reports, etc based on the tests recommended by the doctor for further diagnosis.
Search details	The patient/doctor can search for the required details by entering the keyword
View details	The patient can view his/her own profile, lab reports, doctor's prescription, medical expenses etc.
Pick a doctor	The patient can select doctor based on his/her requirements
View patient's history	The doctor can have access to the patient's history
Schedule an e-appointment	The patient can request for an appointment to a particular doctor, based on his/her availability
Make diagnosis and update details	The doctor can make the diagnosis as per the symptoms given by the patients during e-communication via skype and can update the diagnosis details in the database, that can be viewed by the patient when required. Also, the doctor can update the health summary report of the patient after consultation
Recommend e-prescription / treatment	The doctor can give e-prescription and also can recommend for further test/treatments based on the patient's health condition
Add/Delete user	The administrator can add or delete user (doctor/patient)
Grant access to the user and generate report	The administrator is having the authority to give the access privilege to the user and can generate various reports based on the details required
View complaints and take action	The administrator can view the complaints that have been made by the patient's against the doctor/other issues. Also the administrator can bring it to the notice of the respective party involved

Figure 4 shows the use case diagram:

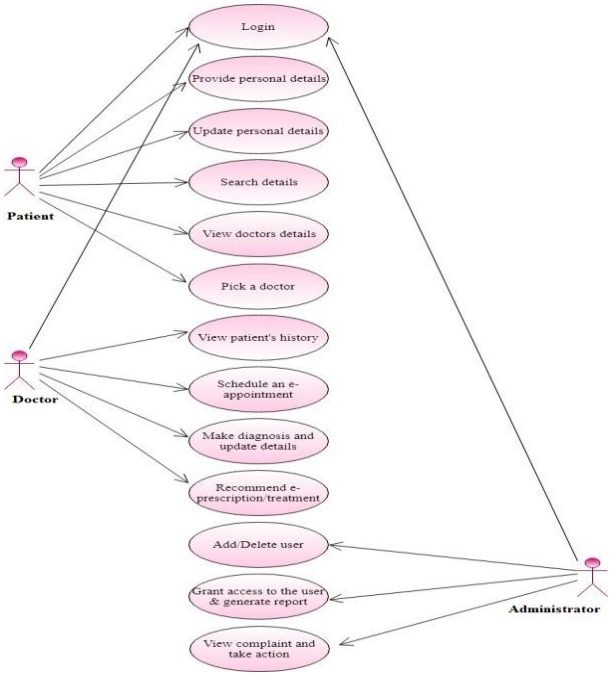

Fig 4. Use Case Diagram for Medical Consultation System

Designing using Sequence Diagram

Sequence diagrams bring out the interactions among classes in terms of an exchange of messages over time. Figure 5 shows the flow diagram of the test case generation from UML Sequence Diagram.

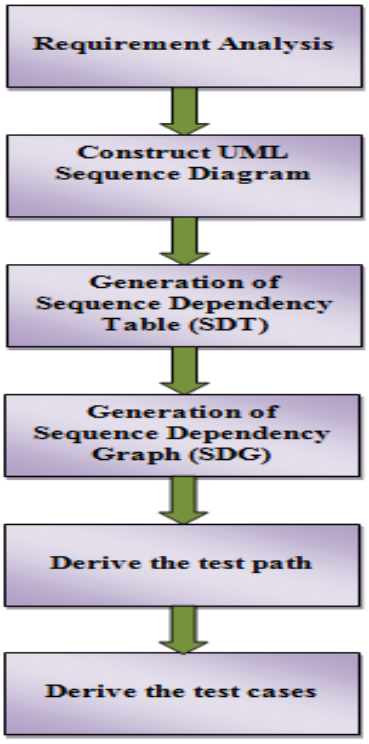

Fig 5. Flow Diagram for automated test path generation

The proposed method for automated test path generation for the medical consultancy system is done by performing the following steps:

i) Construct sequence diagram for medial consultation system (separately for each the use cases identified) as shown in the figure 6.

ii) From the sequence diagram, extract the necessary information and construct Sequence Dependency Table (SDT) as shown in the figure.
iii) From the Sequence Dependency Table (SDT), construct Sequence Dependency Graph (SDG).
iv) Traverse SDG using Depth First Search (DFS) algorithm and derive the test paths.

i) Creating the Sequence Diagram

Figure 6 shows the sequence diagram of the proposed system. The objects considered here are patient, interacting with the doctor/system through a well built user interface, the doctor in turn can interact with the system (database) for fetching the required information for making the diagnosis. There are two types of messages used in this diagram, namely input message and output message. The messages shared are depicted by using two different types of arrows, a solid arrow representing the input message and a dotted arrow representing the output message. The format used for input message is, messagename(parameters) whereas, the output message carries the message alone. Numbers placed in front of the messages (horizontal line) denotes the order in which the messages are exchanged between the objects during the interaction. The sequence of events that occurs in the interaction is represented as a vertical dashed line in the sequence diagram.

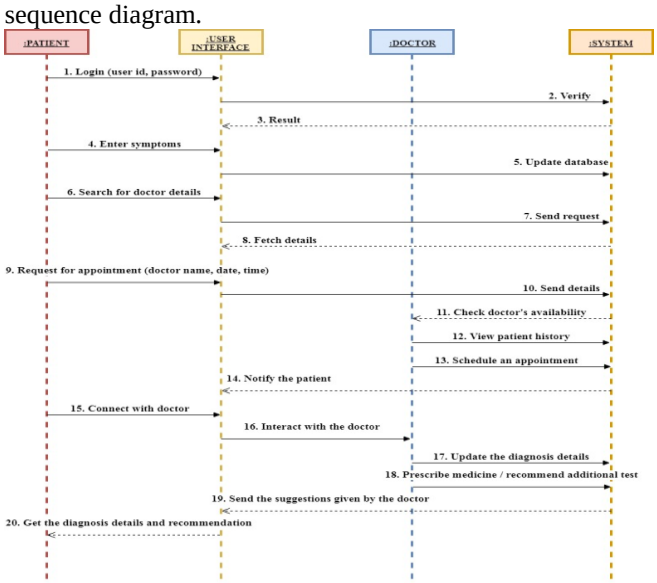

Fig 6. Sequence Diagram for Medical Consultation System

A brief description of the sequence diagram for the medical consultation system is presented below.
1. Login: The patient, if a registered user of the system can logon to the system by providing valid user id and password.
2. Verify: Once the user provides the userid and password, verification is made against the system database to check the authentication of the user. If the provided user id and password matches, the user is considered to be a valid user, and can access the system. Else, the access will be denied.
3. Result*:* Produces the result to the user through the user interface. The result could be either allowing the user to process further, or denying the access.
4. Enter symptoms*:* The patient after successful login is expected to edit the personal details if there is any change after the registration time and they are suppose to select the following details: previous consultation date (optional), and the symptoms they have now (mandatory field).
5. Update database*:* The patient after entering the details through the user interface, clicks on the submit button in the application, which in turn, will commit those information in the database, for further references.
6. Search for doctor details*:* Once the information about the patient is updated in the database, the system will actually provide some recommended list of expert doctors in the respective field and will also suggest available dates of the doctors, based on which the patient can fix an appointment.
7. Send request: The request for searching doctor details will be sent to the system.
8. Fetch details*:* The requested information will be fetched from the database and in turn will be displayed to the user (patient).
9. Request for appointment (doctor name, date, time*):* Based on the recommendation list of doctors provided by the system, the patient can request for an appointment with the doctor, by selecting the name of the doctor, time and date on which they want to meet the doctor. The system on getting the details of the patient connects to the doctor. If the doctor is available, he can respond to the patient, else, the patient will be notified with the unavailability of the doctor.
10. Send details*:* After the patient requesting for the appointment with a particular doctor, the details are updated in the database.
11. Check doctor's availability*:* The system, after receiving an appointment request from the patient, a notification message will be sent to the corresponding doctor, to check his availability. The doctor will also be notified with the patient name and ID, so that the doctor can have further follow-ups, if required.
12. View patient history: If the doctor is available on the particular date, as a next level proceeding, the doctor can view the patient previous history and can check with the given symptoms (as updated by the patient). This will be helpful for the doctor to make the diagnosis.
13. Schedule an appointment*:* The doctor can fix the appointment and the same is updated in the backend.
14. Notify the patient*:* After the doctor confirming his/her availability, a notification message will be sent to the patient, by conforming the data and time of appointment.
15. Connect with doctor*:* The patient can connect to the doctor at the appointed time and date and can communicate in person with the doctor via skype.
16. Interact with the doctor*:* Once the connection is established, thereafter, the doctor and patient can communicate with one another, where the patient can share his/her present health condition and the doctor's before diagnosing can get to know more about the patient's condition. This will help the doctor in making a clear diagnosis.

17. Update the diagnosis details*:* After interacting with the patient, the doctor can make the diagnosis and update the details.
18. Prescribe medicine/recommend additional test*:* Based on the patient's condition, the doctor can prescribe the medicines for cure or if the doctor feels some additional tests are needed for further diagnosis.

19. Send the suggestions given by the doctor: As soon as the doctor updated the diagnosis details, notification will be sent to the patient regarding the updates.
20. Get the diagnosis details and recommendation: The patient can access the information and do as directed.

ii) Sequence Dependency Table Construction

Table 1 shows the Sequence Dependency Table (SDT) which contains six columns holding the information. The SDT is constructed by extracting the information from Sequence diagram. First four fields from the table are considered for generating the test paths as well the Sequence Dependency Graph (SDG). The descriptions of information contained in the sequence dependency table are as follows:

i) Symbol: An alphabetic letter, given for each and every activity involved as a part of the interaction between the objects.
ii) Activity Name: Describes the activity carried out.
iii) Sequence Number: An integer number that gives the trace of flow of data occurs when the messages were exchanged between the involving objects.
iv) Dependency: Symbol(s) of each sequence that the current activity depends upon. For example, the activity 'C' depends on Activity 'A' which means the result that is to be returned depends on the valid User ID and password as entered by the Patient.
v) Input: Gives the data that must be supplied by the user in order to get the expected output.
vi) Expected Output: Gives the expected output as a result of the occurrence of the current event.

TABLE I: SEQUENCE DEPENDENCY TABLE

SYMBOL	ACTIVITY NAME	SEQUENCE NUMBER	DEPENDENCY	INPUT	EXPECTED OUTPUT
A	Login	1	-	User ID and Password	Valid User ID and Password
B	Verify	2	A	-	Validate User ID and Password / Invalid Patient ID or Password
C	Result	3	B	-	Take to the next screen on entering valid User ID and Password
					End on entering invalid user ID or Password
D	Enter Symptoms	4	C	Patient updates the change of details, if any. Also updates the symptoms and last consulted date (Optional)	Checks for the entered details (Valid)
					Invalid or missed details, can prompt the user for entering it again (End)
E	Update database	5	D	-	Update the information in the database
F	Search for doctor details	6	E	Doctor's name	Checks for the availability of expert in the recommendation list
G	Send request	7	F	-	A request is sent to the system for checking the availability of the selected doctor
H	Fetch details	8	G	-	Found the requested information
					Requested information not available in the database
I	Request for appointment (doctor name, date, time)	9	H	Doctor name, date, time	User selecting the doctor name, date and time from the recommendation list
					User not selecting any doctor
J	Send details	10	I	-	Notify the patient with the selections made
K	Check doctor's availability	11	J	-	Doctor available
					Doctor not available
L	View patient history	12	K		Patient details available
					Patient details not available
M	Schedule an appointment	13	L	-	Appointment fixed
					Appointment cancelled, due to the non-availability of the doctor on the requested date and time
N	Notify the patient	14	M	-	Send notification to the patient
					Notification not sent

O	Connect with doctor	15	N	Skype userid, password	Connection is successful
					Connection is unsuccessful
P	Interact with the doctor	16	O	Voice messages	Patient and doctor interact with each other like the patient answering doctor's queries, if any
Q	Update the diagnosis details	17	P	Disease details	Update the details in the database as entered by the doctor against the particular patient successfully
					Updation is not successful
R	Prescribe medicine / recommend additional test	18	Q	Medicine name, time for taking it, number of days to be taken / name of the test	Doctor make diagnosis and prescribe medicine
					Suggest to take some other tests for further diagnosis
S	Send the suggestions given by the doctor	19	R	-	Send notification to the user indicating that doctor's recommendation can be viewed
					Notification not sent
T	Get the diagnosis details and recommendations	20	S	-	Display the diagnosis details and recommendations made by the doctor
					Not displayed
U	End	-	C D H I K L M O Q R S T	-	Close the application

iii) Generating Sequence Dependency Graph

From the created sequence diagram, the messages are collected. The SDG as shown in the figure 7 is generated by using the following Sequence Dependency Graph Generator algorithm:

Algorithm 1: Sequence Dependency Graph Generator

Input : Sequence Dependency Table
Output : Sequence Dependency Graph
Algorithm

1. Start
2. For each and every message present in the SDT, one node with the symbol as label and one edge is created.
3. Check the type of message shared between the objects.
   i) If it's a normal message, repeat step 1.
   ii) If it's a conditional based message, there arise one node and two edges, in which one edge depicts the basis flow i.e., the true path and other flow describes the alternate flow i.e., the false condition.
4. Repeat step 1 or 2, based on the type on message until all the symbols in the SDT are converted into a node with appropriate edge.
5. Connect the nodes and edges in the sequence as they occur in the SDT.

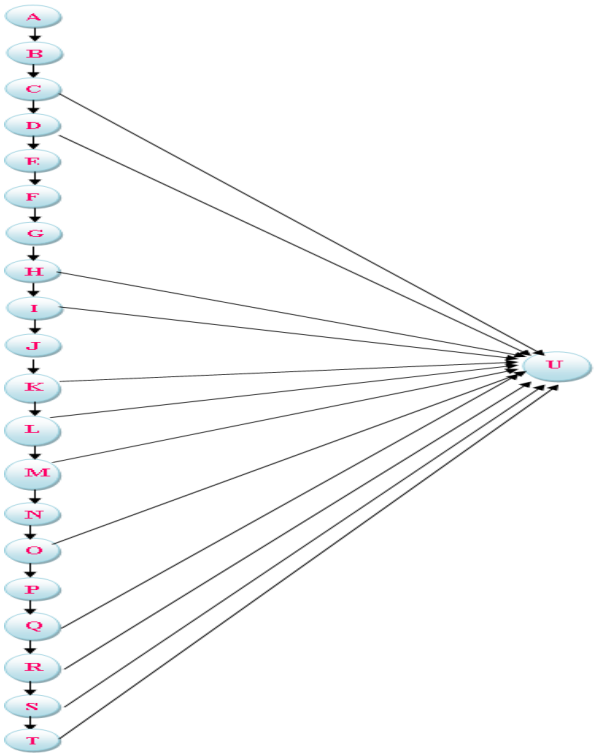

Fig 7. Sequence dependency graph

iv) Test Path Generation

The SDG is traversed by using Depth First Search technique for obtaining the test path. The test path thus generated is shown below that gives the input to the software tester for preparing the test cases, for ensuring a complete coverage of the code during testing.

Algorithm 2: Test Path Generator

Input : Sequence Dependency Graph
Output : Test Path
Algorithm

1. Start
2. Consider the dependency graph root node n as an input node.
3. *For* each node, set visited = 0.
4. *If* visited (n) == 0, then
5. Set visited (n) = 1
6. *End if*
7. *For* each vertex 'm', that is adjacent to 'n' do
8. *If* !visited(m) then
9. Call Test Path Generator
10. *End if*
11. *End for*
12. *End*

Test Case #	Test Path
1	Login → Verify → Result → End
2	Login → Verify → Result → Enter Symptoms → End
3	Login → Verify → Result → Enter Symptoms → Update database → Search for doctor details → Send request → Fetch details → End
4	Login → Verify → Result → Enter Symptoms → Update database → Search for doctor details → Send request → Fetch details → Request for appointment (doctor name, date, time) → End
5	Login → Verify → Result → Enter Symptoms → Update database → Search for doctor details → Send request → Fetch details → Request for appointment (doctor name, date, time) → Send details → Check doctor's availability → End
6	Login → Verify → Result → Enter Symptoms → Update database → Search for doctor details → Send request → Fetch details → Request for appointment (doctor name, date, time) → Send details → Check doctor's availability → View patient history → End
7	Login → Verify → Result → Enter Symptoms → Update database → Search for doctor details → Send request → Fetch details → Request for appointment (doctor name, date, time) → Send details → Check doctor's availability → View patient history → Schedule an appointment → End
8	Login → Verify → Result → Enter Symptoms → Update database → Search for doctor details → Send request → Fetch details → Request for appointment (doctor name, date, time) → Send details → Check doctor's availability → View patient history → Schedule an appointment → Notify the patient → Connect with the doctor → End
9	Login → Verify → Result → Enter Symptoms → Update database → Search for doctor details → Send request → Fetch details → Request for appointment (doctor name, date, time) → Send details → Check doctor's availability → View patient history → Schedule an appointment → Notify the patient → Connect with the doctor → Interact with the doctor → Update the diagnosis details → End
10	Login → Verify → Result → Enter Symptoms → Update database → Search for doctor details → Send request → Fetch details → Request for appointment (doctor name, date, time) → Send details → Check doctor's availability → View patient history → Schedule an appointment → Notify the patient → Connect with the doctor → Interact with the doctor → Update the diagnosis details → Prescribe medicine/recommend additional test → End
11	Login → Verify → Result → Enter Symptoms → Update database → Search for doctor details → Send request → Fetch details → Request for appointment (doctor name, date, time) → Send details → Check doctor's availability → View patient history → Schedule an appointment → Notify the patient → Connect with the doctor → Interact with the doctor → Update the diagnosis details → Prescribe medicine/recommend additional test → Send the suggestion given by the doctor → End
12	Login → Verify → Result → Enter Symptoms → Update database → Search for doctor details → Send request → Fetch details → Request for appointment (doctor name, date, time) → Send details → Check doctor's availability → View patient history → Schedule an appointment → Notify the patient → Connect with the doctor → Interact with the doctor → Update the diagnosis details → Prescribe medicine/recommend additional test → Send the suggestion given by the doctor → Get the diagnosis details and recommendations → End

C. Implementation Phase

In the implementation phase, the modules are developed. There are five modules as follows:

i) Login for different users
ii) System Administrator
iii) Patient/Doctor Registration
iv) Scheduling an appointment
v) Consultation

i) Login for different users

There are several users of the system like Doctors, Patients/Care Taker, Hospital management authorities, Administrator who are considered as the only authorized persons for using the system. They can access the services offered by the system by providing the correct user name and password. In case, if a wrong credentials are entered, the system rejects the access to the services provided.

ii) System Administration

The system administrator is given with the access right of adding, editing, updating, searching the information in the system, providing access rights to the user and can access/control the services offered by the different uses of the system.

They maintain the patient diagnosis details, advice given by the doctor, diet advice details, billing details of the in-patient and out-patient, various report details like X-ray, Urine test, Blood test, of the patients. The administrator can also generate related reports of patient, prescriptions provided by the doctor, bill reports etc as per the requirements.

iii) Patient/Doctor Registration

This module is implemented for the patients and doctors to register their details. The patient can enter his/her details like name, age, contact details, previous medical history (optional), symptoms, last consultation date (optional), consultation needed for details, etc. The doctors can also register their information like their name, registration number, qualification, specialization, available time, etc. Once the patient/doctor updates their information, will be issued with a Patient ID/Doctor ID respectively, thereafter, which will act as a unique key for accessing the information further.

iv) Scheduling an appointment

The registered patients can access the information of the doctor at any time, and can fix an appointment with him/her based on the requirements (like selecting the date, time, place etc). The patient is also provided with an option of uploading their treatment related documents such as X-ray, blood report, urine report or other supporting document for the doctor to make a diagnosis about the present health condition. This information can be accessed by the doctor before fixing an appointment with the patient.

v) Consultation

The doctor can check with the request made by the patient and also can view the patient's previous medical history for diagnosing the disease. The doctor can confirm the appointment based on his/her availability and a notification will be sent to the patient stating the same. One the particular date and time, the doctor and the patient can get connected through skype, can have an interaction where the doctor can answer the queries posted by the patient, also doctor can raise queries to the patient for getting further clarification, if any, for diagnosis. After the consultation is over, the doctor can suggest the treatment and/or advises that can be recorded and notified to the patient.

D. Testing Phase: Smart Test Case Quantifier

The smart test case quantifier is an automated tool developed for generating the minimum number of test cases needed for testing the application. The modules involved are [24]:

i) User Interface
ii) Parsing the input file
iii) Control Flow Graph Generator (CFGG) and
iv) Metric Calculator

Figure 8 shows the use case diagram of the Smart Test Case Quantifier.

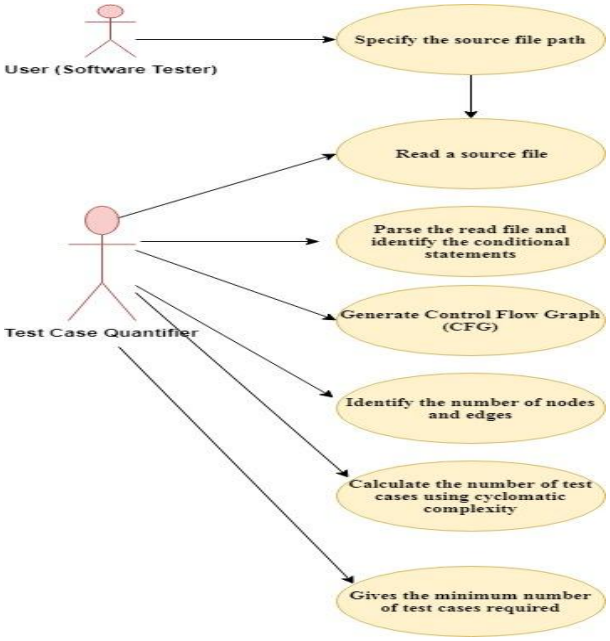

Fig 8. Use Case Diagram of Smart Test Case Quantifier

The detailed descriptions of the modules in the smart test case quantifier are as follows: The user interface module gets the path of the files that are to be tested which contains the source code. The source files are passed on to the parser module that parses the content, by using recursive descent parser algorithm. The parser produces the output that contains the conditional statements in the source code, based on which the test case can be derived. The derived conditional statements are given as an input to the Control Flow Graph Generator, which builds a graph,

which contains the statement as node and edges between them. The output of CFGG is given as input to the metric calculator. It generates the minimum number of test cases needed to test the given code. The steps involved in finding the same are as below:

Algorithm 3: Minimum Test Case

Input : Control Flow Graph (CFG)
Output : Minimum number of test cases
Algorithm

1. Start
2. Consider the starting node in the CFG.
3. Calculate the number of nodes and edges.
4. *If nodes < edges*, then
5. Calculate Cyclomatic complexity as M = E – N + 2
6. *End if*
7. Display the number of test cases.
8. End

IV. RESULTS AND DISCUSSIONS

The following screen shots are the results of the algorithms applied during design phase for generating the SDT, path generation.

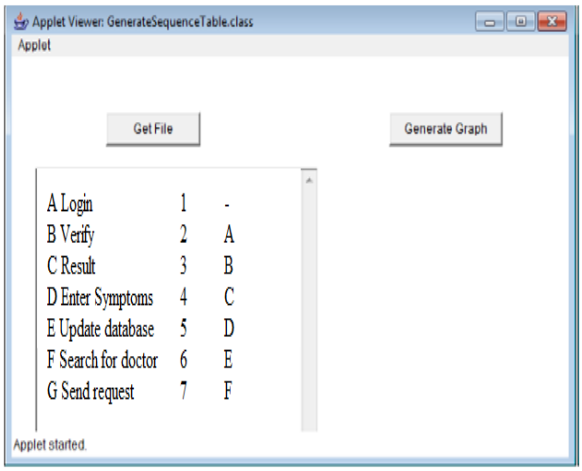

Fig 9. Sequence Dependency Table

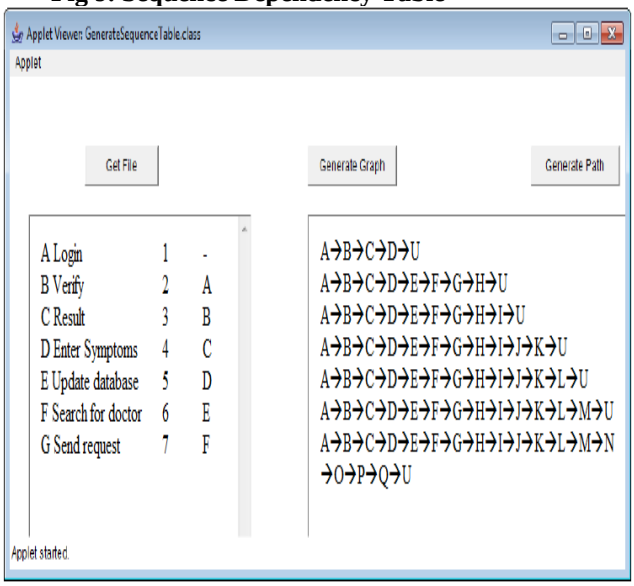

Fig. 10 Test Path Generation

The following screen shots shows the implementation details of the medical consultation system.

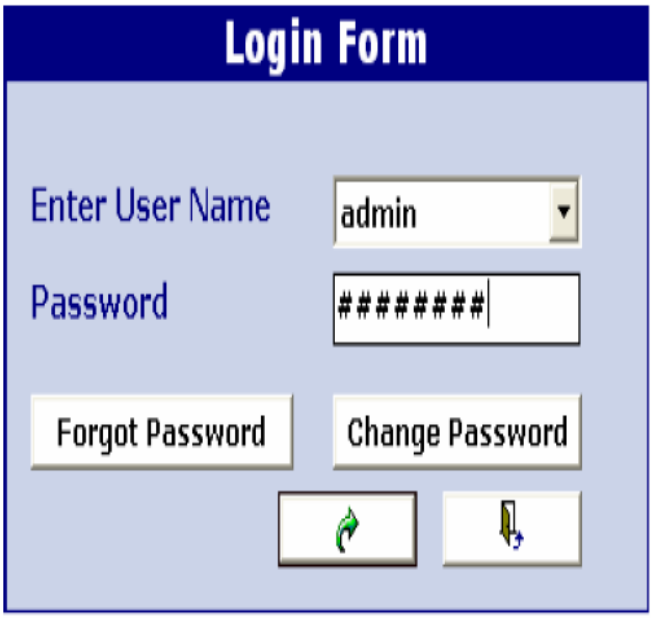

Fig. 11 Login Form

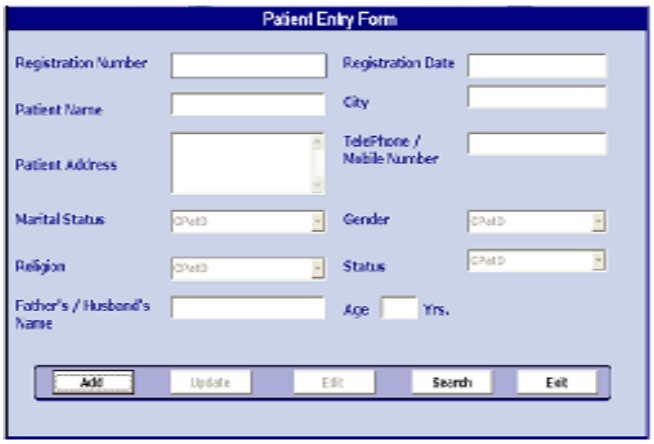

Fig. 12 Patient Entry Form

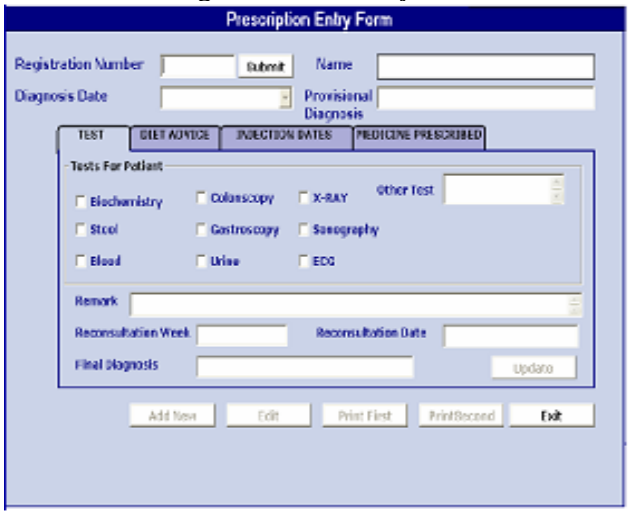

Fig. 13 Prescription Entry Form

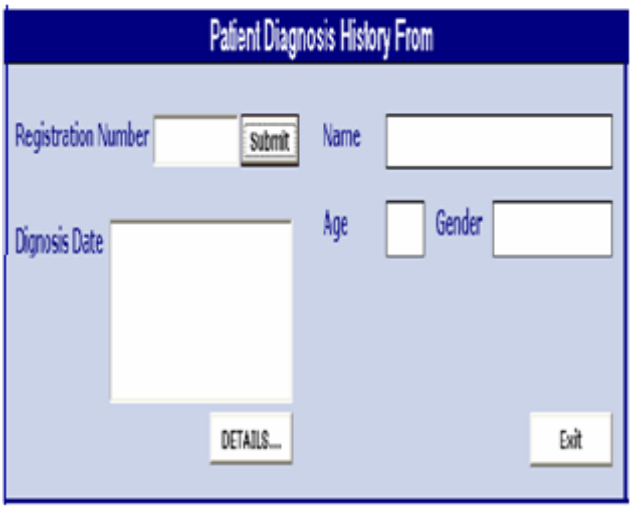

Fig. 14 Patient Diagnosis History Form

The following screen shots show the implementation of smart test case quantifier in testing phase.

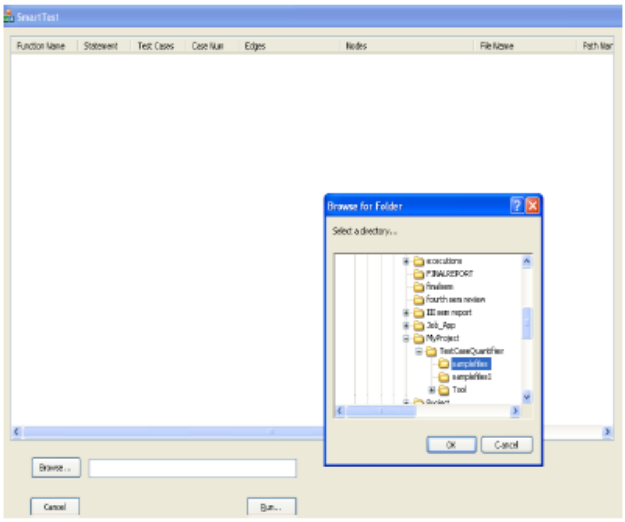

Fig. 15 Selecting the source file for generating

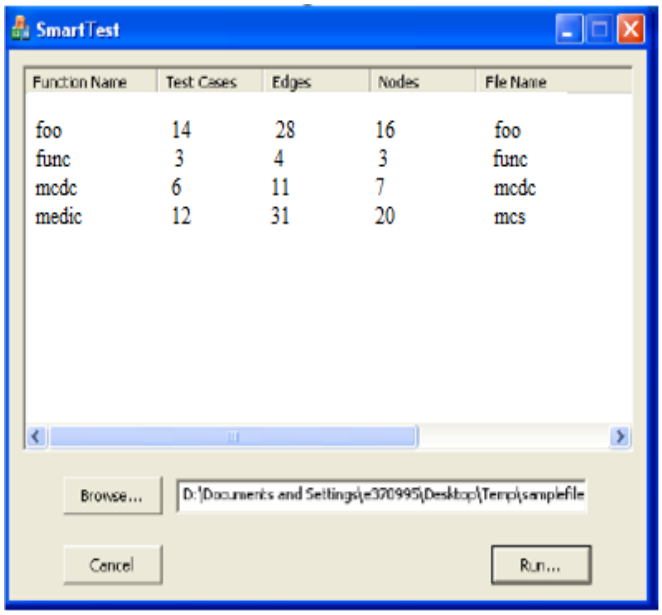

Fig. 16 Displaying minimum test cases required

V. CONCLUSION

During test case development there might be enormous test paths available, and the software tester could get a confusion which path must be considered and which one to leave out. So it is essential to get those basic paths to be considered in order to ensure complete coverage. This proposed approach helps in generating the test path for which the test cases could be produced by the software tester during testing to check whether the system works as intended. Moreover, the proposed approach assists the developers to significantly improve their design quality. They could find the faults in the implementation early and prevent those getting disseminated to the successive phases. As exhaustive testing is not possible, the test cases chosen to test a system can be minimal but should ensure maximum coverage required to test an application. This might help the software tester to reduce the testing time. The proposed system for medical consultancy system automatically generates the test cases from the UML Sequence diagram in the design phase by considering the requirements gathered in the requirement gathering and analysis phase which is necessary for functional testing. From the constructed Sequence Diagram, the Sequence Dependency Table is built, from which the Sequence Dependency Graph is generated and then the test cases are derived from the Sequence Dependency Graph. In the implementation phase, the source code for the medical consultancy system is done. In the testing phase, the smart test case quantifier is produced which gives the minimum number of test cases needed to test the system, which is the basic need for performing structural testing. The selection of coverage metric MC/DC criterion plays an important role in generating the number of test cases which eliminates the redundant test cases hence, favors in reducing the test cases. The logic followed for test case quantifier, clearly tells that the need for identifying all the possible paths is very important.

REFERENCES

1 Abdurazik A, Offutt J, “Using UML collaboration diagrams for static checking and test generation”, In International conference on the unified modeling language, pp 383–395, 2000.

2 Jaiprakash TL, Mall R., “A dynamic slicing technique for UML architectural models”, IEEE Transaction Software Engineering, pp.735- 771, 2011.

3 Lallchandani R, Mall JT., “Integrated state-based dynamic slicing technique for UML models”, IET Software, Volume 4, pp.55-7, 2010.

4 Sethukkarasi R, Ganapathy S, Yogesh P, Kannan A.,”An intelligent neuro fuzzy temporal knowledge representation model for mining temporal patterns”, Journal on Intelligent Fuzzy System, pp. 1167-117, 2014.

5 Ganapathy S, Sethukkarasi R, Yogesh P, Vijayakumar P, Kannan A., “An intelligent temporal pattern classification system using fuzzy temporal rules and particle swarm optimization”, Proceedings in Engineering Science, Volume 39, Issue 2, pp 283–302, 2014.

6 Baikuntha Narayan Biswal, Pragyan Nanda, Durga Prasad Mohapatra “A Novel Approach for Scenario- Based Test Case Generation”, International Conference on Information Technology, 2008.

7 Nirmal Kumar Gupta, Mukesh Kumar Rohil, “Improving Genetic Algorithm based Automated Test Data Generation Technique for Object Oriented Software”, 3rd IEEE International Advance Computing Conference (IACC), 2013.

8 Supaporn Kansomkeat, Phachayanee Thiket and Jeff Offutt, “Generating Test Cases from UML Activity Diagrams using the Condition-Classification Tree Method”, 2nd International Conference on Software Technology and Engineering(ICSTE), 2010.

9 V.Mary Sumalatha, G.S.V.P.Raju, “Object Oriented Test Case Generation Technique using Genetic algorithms”, International Journal of Computer Applications, Volume 61– No.20, January , 2013.

10 S. Shanmuga Priya, Dr. P. D. Sheba, “Test Case Generation from UML Models - A Survey”, International Journal of Emerging Technology and Advanced Engineering, Volume 3, Special Issue 1, pp. 449-459, January 2013.

11 Ranjita Kumari,Vikas Panthi, Prafulla Kumar Behera,“Generation of test cases using Activity Diagram”, International Journal of Computer Science and Informatics, ISSN (PRINT): 2231 –5292, Volume- 3, Issue-2, 2013.

12 Clachar, Sophine and Grant, Emanuel., “Formalizing UML Software Models of Safety Critical Systems” , Proceedings of the Annual International Conference on Software Engineering, Phuket, Thailand, 2010.

13 Sanjai Rayadurgam and Mats P. E. Heimdahl, “Coverage Based Test-Case Generation using Model Checkers”, 8th IEEE International Conference on Engineering of Computer-Based Systems (ECBS 2001), Washington, DC, USA, pp. 17-20, April 2001.

14 Vikash Panthi, Durga Prasad Mohapatra, “Automatic Test Case Generation using Sequence Diagram”, International Journal of Applied Information Systems, Vol. 2, No. 4, May 2012.

15 Ranjita Swain, Vikash Panithi, Prafulla Kumar Behera and Durga Prasad Mohapatra, “Automatic Test Case Generation from UML State Chart Diagram”, International Journal of Computer Applications, Vol. 42, No. 7, March 2012.

16 Puneet Patel and Nitin N. Patil, "Test Case Formation using UML Activity Diagram", World Journal of Science and Technology, Proceedings of National Conference of Emerging Trends in Computer Technology, pp. 57-62, April 21, 2012.
17 Santhosh Kumar Swain and Durga Prasad Mohapatra, "Test Case Generation from Behavioral UML Models", International Journal of Computer Application, Vol. 6, No. 8, September 2010.
18 Monalisa Sarma, Debasish Kundu and Rajib Mall, "Automatic Test Case Generation from UML Sequence Diagrams", 15th International Conference on Advanced Computing and Communications, 2007.
19 Booch G, Maksimchuk RA, Engle MW, Young BJ, Conallen J., "Object oriented analysis and design with applications", Pearson Edu (3rd edn.), 2010.
20 Wu CS, Khoury I., "Web service composition: From UML to optimization", Fifth International Conference on Service Science and Innovation, 2013.
21 Arezoo Yazdani SeqerlooMohammad Javad AmiriEmail authorSaeed ParsaMahnaz Koupaee, "Automatic test cases generation from business process models", Requirements Engineering, Volume 24, Issue 1, pp 119–132, March 2018,
22 Zhu LZ, Kong FS, "Automatic conversion from UML to CPN for software performance evaluation", Procedia International Workshop on Information and Electronics Engineering, pp. 2682-2686, 2012.
23 https://en.wikipedia.org/wiki/Use_Case_Diagram.
24 S. Shanmuga Priya, Sheba Kezia Malarchelvi, "Smart Test Case Quantifier Using MC/DC Coverage Criterion", International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970), Volume-4, Number-1, Issue-14 pp. 396-401, March-2014.
25 Namita Khurana,R.S Chillar, "Test Case Generation and Optimization using UML Models and Genetic Algorithm", 3rd International Conference on Recent Trends in Computing (ICRTC-2015) , pp. 996 – 1004, 2015.
26 Asad S, Shah A, Shahzad RK, Shafique S, Bukhari A, M Humayun, "Automated Test Case Generation using UML Class and Sequence Diagram", 15(3), pp. 1-12, 2016.
27 Meiliana, Irwandhi Septian, Ricky Setiawan Alianto, Daniel, Ford Lumban Gaol, "Automated Test Case Generation from UML Activity Diagram and Sequence Diagram using Depth First Search Algorithm", 2nd International Conference on Computer Science and Computational Intelligence 2017, ICCSCI 2017, , pp. 629–637, Bali, Indonesia, October 2017.
28 Priya Kamath B., Narendra V.G., "Generation of Test Cases from Behavior Model in UML", International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 17, pp. 13178-13187, 2018.

AUTHORS PROFILE

S. Shanmuga Priya is currently working as Senior Assistant Professor in the Department of Computer Science and Engineering, New Horizon College of Engineering, Bangalore, Karnataka. She received her B.E. in Computer Science and Engineering from Bharathidasan University, and M.E in Computer Science and Engineering, from Anna University, Tamilnadu, India. She has around 15 years of experience in teaching. Her research interests include Big Data, Data Mining, Software Engineering and Software Testing.