Exploring Model Base Testing(MBT) And Graphwalker

Riddhi Mistry, Dhirubhai Ambani Institute of Information and Communication Technology (DA-IICT)
202201238@daiict.ac.in

Introduction

Testing is used for software validation and verification. Black-box and white-box testing are the two methods used for software testing. Black-box testing is used to test the system's requirements without knowing the internal part of the system. This testing does not give any information about the system or the internal structure of the code. Model Base Testing (MBT) is a type of black-box testing. MBT is used to generate abstract test cases. Abstract test cases can be transferred to scripts, lists, and tables. Using MBT, we can explore the system more. MBT generates test cases automatically sequentially, which helps to measure the requirements coverage of the system.

Flow of Model-Based Testing (MBT)

- Create the Model
- Generate the abstract test case
- Abstract test case converts to executable test case
- Execute this test case
- Analyze
- Model iteration

MBT connects system requirements with the automated generation of test cases, using behavior models to represent workflows, states, and transitions. Tools like GraphWalker and OSMO automate the generation of test cases by analyzing the model's states and paths. These abstract test cases are then converted into executable ones, integrated into CI pipelines, allowing seamless execution and visualization of test flows and outcomes. The iterative process ensures continuous improvement and alignment of the model with the behavior of the system.

Tools of MBT

Some tools of MBT are available, but one of the open-source tools is Graph Walker. Graph Walker uses the system's diagram for generating abstract test cases and shows the paths that each test case follows in the system.

MBT using Graphwalker

In Graphwalker, the process can be divided into two parts. First, the model is generated based on the system requirements. Second, this model is used to generate test cases and determine the coverage of the requirements.

GraphWalker is helps to create, visualize, and execute test cases based on system behavior. It ensures better test coverage by generating tests for all possible paths and states in the model. This reduces manual effort, improves testing efficiency.

I. Example of Graphwalker [Hotel Booking]

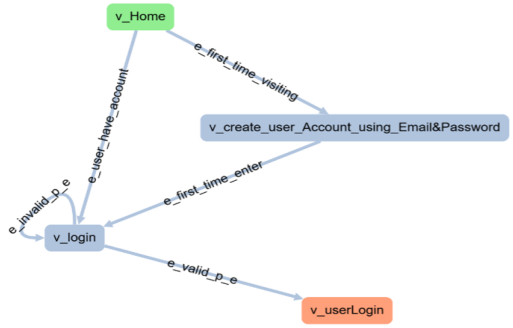

Fig. 1. Model of User Login Process

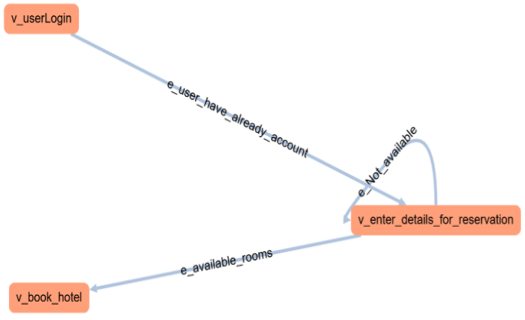

Fig. 2. Model of User Booking Process

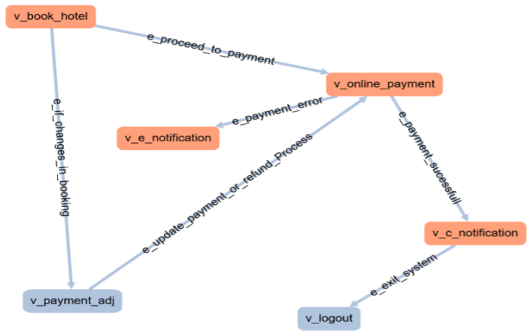

Fig. 3. Model of User Payment Process

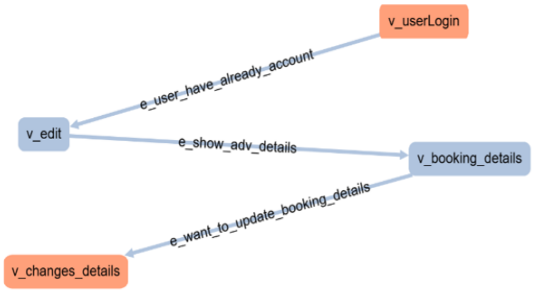

Fig. 4. Model of User Edit the Booking

Flow of the Hotel Booking Example

- **To enter the booking hotel:**
  - If the user already has an account:
    1) Log in to the system using a password and email.
    2) If anything is wrong, the user tries to log in again.
  - If the user does not have an account:
    1) Register in the system first.
    2) Then log in.
- **For booking:**
  1) After logging in, the user enters their requirements.
  2) Based on the requirements, the system suggests available rooms.
  3) If no rooms are available, the user re-enters details.
- **For payment:**
  - Online payment option is available.
    1) Confirmation message is sent.
    2) If there is an error, an error message is sent.
  - If there are any changes in booking, update the payment amount.
- **Edit Booking:**
  1) First log in to the system.
  2) Then the edit option becomes available.
  3) Update the previous booking and adjust payments.

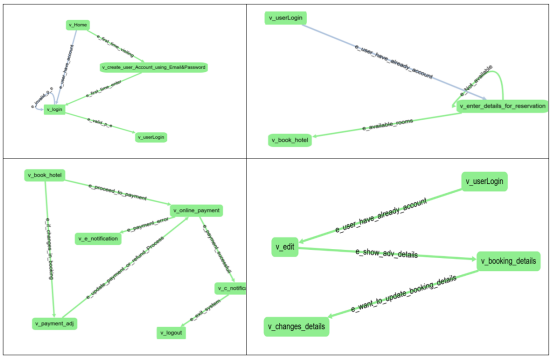

Fig. 5. Check the system's requirements coverage using run the model

Console Output when run CLI

```
Users\Riddhi Mistry\Downloads>java -jar
graphwalker-cli-4.3.2.jar offline -m
hotelbooking.json "random(edge_coverage(100))"

{"currentElementName":"have_user_have_account"}
{"currentElementName":"login"}
{"currentElementName":"login_error"}
{"currentElementName":"visit_login"}
{"currentElementName":"login"}
{"currentElementName":"have_already_account"}
{"currentElementName":"user_details_for_
reservation"}
{"currentElementName":"user_details_for_
reservation"}
{"currentElementName":"available_rooms"}
{"currentElementName":"list_available_rooms"}
{"currentElementName":"book_hotel"}
{"currentElementName":"payment_process"}
{"currentElementName":"do_online_payment"}
{"currentElementName":"engine_payment"}
{"currentElementName":"payment_success"}:
:
```

If a user can reach the payment process without logging in or completing the booking, it indicates a defect. GraphWalker detects this by simulating the correct flow, and flagging any missing steps as issues to ensure the system works as intended.

This example demonstrates how GraphWalker helps model, test, and validate a hotel booking system. By creating models for processes like login, booking, payment, and editing, it ensures complete test coverage and detects defects early. The automated test generation reduces manual work and improves efficiency.

Comparison between GraphWalker and Manual Testing

In terms of test coverage, both methods focus on the requirements. However, GraphWalker generates many test

cases by exploring all possible paths, covering more scenarios. Manual testing, on the other hand, usually only tests the basic functions and can be biased based on the tester's priorities. Due to its randomization, GraphWalker ensures more diverse and thorough testing. So, GraphWalker is better at providing broader and more automated test coverage compared to manual testing.

BENEFITS OF MBT

The main use of MBT is generating automatically test cases from the system models. MBT follows paths through the model when generating test cases and this allows us to identify which path leads to failure. The time required to generate test cases was decreased by MBT. Sometimes MBT takes more time initially due to model setup, but MBT covers all the paths of the system with all the conditions. If the system requirement changes, then in manual testing we update all the test cases, but in MBT we only change the model of the system

DISADVANTAGE OF MBT

The main disadvantage of MBT is the limited available MBT tools and the limited information about the available tools. Generating abstract test cases by MBT is less readable to the user or tester because it is presented in a high-level language that may lack the necessary details and not much clarity for practical implementations of the system. The large-scale system model has too many edges and nodes, so the system model becomes complex.

CHALLENGES IN MBT EXPLORING AND TOOLS I FACED

- Limited resources are available for MBT, and there is no guarantee that the available resources will provide the information needed.
- Running the provided GraphWalker example was challenging due to Java version compatibility issues.

ACKNOWLEDGMENT

I sincerely thank Dr. Saurabh Tiwari for giving me the opportunity to explore Model-Based Testing (MBT) and other testing tools. His guidance and support throughout the process greatly increased my interest in exploring these tools.

REFERENCES

[1] Google Drive Link for MBT Resources
[2] Google Drive Link for my Graph walker Examples