CHAPTER 7

AGENT-BASED MODEL FOR TEST CASE PRIORITIZATION

7.1 INTRODUCTION

The general definition of an intelligent agent is: “a computer system that is situated in some environment and which is capable of flexible autonomous action in that environment in order to meet its design objectives” (Luck et al 2005). The amount of flexibility in the agent’s autonomy is determined by three factors. The first is the agent’s reactivity to its environment. This depends on the agent’s ability to perceive and respond to its surroundings. The second is the level of pro-activeness, or initiative that the intelligent agent takes in meeting its design goals and the final factor is the level of social interaction among agents. Agents which are highly autonomous, have the ability to communicate with other agents to meet their requirements.

A multi-agent system communicates with the application, the user, as well as with the other agents in the system to achieve its objectives. But, in case of the individual agent, communication channels are only open between the agent and the user only. The key characteristics of multi-agent environments are:

- They provide the infrastructure for inter-agent communication
- They are usually designed to be open concept without any centralized designer
• The agents within a multi-agent system are autonomous and may be cooperative or competitive in nature.

Agents play a major role in the automation of any process, thereby reducing most of the human intervention in the process. The main objective of this chapter is to design an agent-based model for regression testing and perform, in particular, test case prioritization, thereby reducing the time and cost incurred during regression testing.

7.2 PROPOSED AGENT-BASED MODEL FOR TEST CASE PRIORITIZATION

The study proposes an agent-based model to ease the regression test process thereby reducing the overall time and cost of the complete regression process. As the process of prioritization for each operation is different, the model performs prioritization one by one by invoking the test cases present in the repository. The study has used JADE (Java Agent DEvelopment Framework, www.jade.tilab.com) which is a software framework. The details of the model are shown in Figure 7.1.

Figure 7.1 Agent-based model for test case prioritization
This model includes four agents namely the Test agent, Interface agent, Select agent and the Report agent. The test agent is considered to be the overall controlling agent that controls the other agents in the process. The tester interacts with the test agent only by informing his requirements out of the agent-based process. The test agent is responsible for performing the overall regression process by coordinating with the other agents and after completion, reports the results to the tester. The functions of the various agents involved are detailed in Table 7.1.

**Table 7.1  Function of individual agents in the model**

<table>
<thead>
<tr>
<th>Agent Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| Test Agent      | **Name:** Test Agent (Test case Agent)  
**Description:** Main Agent that initializes other agents in the complete regression process  
**Lifetime:** Until it completes the regression test process.  
**Initialization:** Initialized by the tester.  
**Incoming Messages:** To start the regression test process from the tester.  
**Outgoing Messages:** To initiate the Interface Agent first. |
| Interface Agent | **Name:** Interface Agent  
**Description:** Interfaces with application to be tested.  
**Lifetime:** Till it checks the application and the required information’s availability.  
**Initialization:** Initialized by the Test Agent.  
**Incoming Messages:** To start the process from the Test Agent.  
**Outgoing Messages:** To inform the Test Agent after Completion. |
Table 7.1 (Continued)

<table>
<thead>
<tr>
<th>Agent Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| Select Agent | **Name**: Select Agent  
**Description**: To select any one operation’s TCPR process based on the Test Agent request and to start the selected test process. The cycle continues until TCPR of all operations are performed or till the Test Agent request is satisfied.  
**Lifetime**: Till it completes the TCPR process for all operations or as requested by the Test Agent.  
**Initialization**: Initialized by the Test Agent after it receives the completion message from the Interface Agent.  
**Incoming Messages**: To start the TCPR process from the Test Agent.  
**Outgoing Messages**: To inform the Test Agent after Completion. |
| Report Agent | **Name**: Report Agent  
**Description**: To generate various reports as per the test agent’s request based on the TCPR process performed.  
**Lifetime**: Till it computes the reports of the prioritization techniques performed.  
**Initialization**: Initialized by the Test Agent after it receives the prioritization completion message from the Select Agent.  
**Incoming Messages**: To start the report generation process from the Test Agent.  
**Outgoing Messages**: To inform the Test Agent after Completion. |

The sequence of how the various agents interact and perform the regression test process is detailed in the sequence diagram shown in Figure 7.2. The tester responsible for performing the regression test, gives
details of what type of testing should be performed, the order in which it is to be performed, the reports required and so on, to the test agent. The test agent on receiving the details from the tester initiates the interface agent. The interface agent checks for the readiness of the application. Readiness of the application implies that the complete application that is to be tested is ready; the presence of the test cases that are to be prioritized and almost all information about the application is available. The interface agent on completing its functions interacts with the test agent informing its completion.

![Figure 7.2 Sequence diagram – overall agent process](image)

The test agent after receiving the ready signal from the interface agent initializes the select agent giving details of what type of prioritization is to be performed. The selection agent, on initialization, selects one operation at a time and performs the prioritization for that operation. When one operation is selected, the model concerned of that operation is invoked as follows:
1. If the operation selected is Basic (option 1), the basic model is invoked. The functioning of the basic model is shown in the sequence diagram in Figure 7.3.

The model includes five different agents namely basic test agent, basic interface agent, test coverage agent, basic prioritization agent and performance agent.

Figure 7.3 Sequence diagram – Prioritization of Basic operations

As shown in Figure 7.3, the basic test agent plays the major role of coordinating with the other agents of the basic model. This basic test agent is initiated by the select agent of the agent model. On initialization, the agent sends the ready signal to the basic interface agent who identifies the basic test cases from the repository, identifies any new basic operations in the application, generates test cases, if required and places them in the repository. Once the whole process is completed, it responds back to the basic test agent.
The basic test agent on receiving the completed signal initializes the test coverage agent. This agent calculates the coverage (statement, branch, loop, condition) of all the test cases and stores back the test cases along with the coverage information in the repository. Once completed, it responds back to the basic test agent. On receiving the completed signal, the basic test agent initializes the basic prioritization agent, which prioritizes the test cases. This agent checks for what type (coverage-based, cost-based, search-based or time-based) of prioritization is to be performed and based on the tester’s request via basic test agent, prioritizes with the best technique, as per the analysis in Chapter 3, for that type.

When the whole prioritization process is complete, the agent responds back to the basic test agent.

The basic test agent then initiates the performance agent that reports back the performance of the prioritization process performed. Finally the basic test agent submits all the reports to the select agent which stores the information in the repository which can be accessed by the tester. The functions of the individual agents are shown in Table 7.2.

**Table 7.2 Functions of individual agents in the basic model**

<table>
<thead>
<tr>
<th>Agent Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Test Agent</td>
<td>Name: Basic Test Agent</td>
</tr>
<tr>
<td></td>
<td><strong>Description</strong>: Main Agent that initializes other agents</td>
</tr>
<tr>
<td></td>
<td><strong>Lifetime</strong>: Untill it completes the test process.</td>
</tr>
<tr>
<td></td>
<td><strong>Initialization</strong>: Initialized by the Select agent from the agent main model.</td>
</tr>
<tr>
<td></td>
<td><strong>Incoming Messages</strong>: To start the test process from the select agent from the main model.</td>
</tr>
<tr>
<td></td>
<td><strong>Outgoing Messages</strong>: To initiate the basic interface agent first.</td>
</tr>
<tr>
<td>Agent Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Basic Interface Agent       | **Name**: Basic Interface Agent  
**Description**: Interface with application to be tested.  
**Lifetime**: Till it initiates the test case generation process.  
**Initialization**: Initialized by the basic test agent.  
**Incoming Messages**: To start the test process from the test agent.  
**Outgoing Messages**: To inform the test agent after completion. |
| Test Coverage Agent         | **Name**: Test Coverage Agent  
**Description**: To initialize the computation of Coverage.  
**Lifetime**: Till it computes the coverage of the required test cases.  
**Initialization**: Initialized by the basic test agent after it receives the completion message from the interface agent.  
**Incoming Messages**: To start the coverage computation from the basic test agent.  
**Outgoing Messages**: To inform the basic test agent after completion. |
| Basic Prioritization Agent  | **Name**: Basic Prioritization Agent  
**Description**: To decide as to what type of prioritization is to be done and start the prioritization process..  
**Lifetime**: Till it computes the prioritization of the test cases.  
**Initialization**: Initialized by the basic test agent after it receives the coverage completion message from the coverage agent.  
**Incoming Messages**: To start the prioritization computation from the test agent.  
**Outgoing Messages**: To inform the basic test agent after completion. |
Table 7.2  (Continued)

<table>
<thead>
<tr>
<th>Agent Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| Performance Agent    | **Name:** Performance Agent  
                          **Description:** To compute the performance of the various prioritizing techniques to improve the fault detection effectiveness thereby reducing the execution time.  
                          **Lifetime:** Till it computes the performance of the prioritization techniques.  
                          **Initialization:** Initialized by the basic test agent after it receives the prioritization completion message from the prioritization agent.  
                          **Incoming Messages:** To start the performance computation from the test agent.  
                          **Outgoing Messages:** To inform the basic test agent after completion. |

2. If the operation selected is WEB (option 2), the WEB Model is invoked. The functioning of the Web Model is as shown in Figure 7.4.

The model includes five different Agents viz. web test agent, web interface agent, test coverage agent, web prioritization agent and performance agent. The web test agent is initiated by the select agent present in the main agent model. This agent serves as an overall control agent for the web prioritization process.

To start with, it initiates the web interface agent which checks for the readiness of the application to be tested. It responds back to the Web test agent once its job is completed. On receiving this signal, the web test agent, initiates the test coverage agent which checks the parameters present in the test cases, computes the length of test cases, computes sequences from the test
cases and so on which is required for the prioritization process and stores it back in the repository with all information.

Figure 7.4 Sequence diagram – Prioritization of Web operations

The test coverage agent responds back to web test agent after completion. The web test agent then initiates the prioritization agent, where this agent prioritizes the test cases as detailed in Chapter 4. After completion, this agent responds to the web test agent which initiates the performance agent which computes the performance of the prioritization process performed.

Finally the web test agent submits all the reports generated to the select agent of the main agent model which stores all the information in the common repository and responds back to the tester.
3. If the operation selected is database (option 3), the database model is invoked. The functioning of the database model is shown in Figure 7.5.

![Figure 7.5 Sequence diagram – Prioritization of database operations](image)

The model includes five different agents namely database test agent, database interface agent, test coverage agent, database prioritization agent and performance agent.

The database test agent plays the major role of coordinating with other agents in the model. This database test agent is initiated by the select agent of the main agent model. On initialization, this agent sends the ready signal to the database interface agent who identifies the database test cases from the repository. Once the whole process is completed, it responds back to the database test agent.

The database test agent on receiving the completed signal initializes the test coverage agent. This agent calculates the coverage of all the test cases
and stores back the test cases along with the coverage information in the repository. Once completed, it responds back to the database test agent.

On receiving the completed signal, the database test agent initializes the prioritization agent, which prioritizes the test cases. This agent prioritizes the test cases based on the best technique as per the analysis in Chapter 5. When the whole prioritization process is complete, the agent responds back to the database test agent. The database test agent then initiates the performance agent that reports back the performance of the prioritization process performed. Finally the database test agent submits all the reports to the select agent which stores the information in the common repository.

This process is repeated until all or requested operations are completed. On completion of prioritization of every operation, the select agent of the main agent model stores all the reports received, in a common repository. On completion, the select agent responds to the test agent using “Prioritization Complete” signal. Finally, the test agent initiates the report agent, to generate the various reports of the regression test process. The report agent collects details from the repository and generates the overall reports of the application tested and submits the same to the test agent. The test agent finally submits all the reports to the tester.

7.3 PERFORMANCE EVALUATION

Several experiments were conducted in the study to address the following objectives for software applications.

RQ1: To find out whether an agent-based system reduce the manual work?

RQ2: Identify whether the proposed agent-based model could reduce the execution time compared to the general model.
RQ3: To identify whether the proposed agent-based model is cost effective compared to general model.

1. **Subject applications used for analysis**

For analysis, the study has considered two complete applications that involve almost all the three operations identified in the study. The subject programs considered have different characteristics.

1. **E-book (gotocode.com)** - involves all the operations (basic, web and database): E-Book allows users to register, login, browse for books, search for books by keyword, rate books, add books to a shopping cart, modify personal information, and logout. E-Book uses JSP for its front-end and a MySQL database as back-end. Test cases for basic functions, database and web operations were generated.

2. **E-Ordering**: E-ordering allows users to register, login, browse for products, search by keyword, add products to a shopping cart, modify personal information, and logout. E-Ordering uses JSP for its front-end and a MySQL database as back-end. Test cases for basic functions, database operations and web operations were generated and stored in repository.

7.3.1 **Experimental Methodology**

The complete agent-based model was implemented using Java and the agents were implemented using the JADE framework, which is a Java compatible framework. In the study, for the agent implementation, the information required for prioritization was assumed to be present in the repository. That is the implementation of the agent was only to explore the information present in the repository. Separate repositories were maintained for each operation and the agent implementation also checked for boundary
conditions and extreme conditions. Extreme conditions occur when the information is not present in the repository. Under such situations, the agents search the repository for ten milliseconds, and returns back to the caller with a negative message. The model was not completely automated as the above assumptions were made. These assumptions could be considered as a future enhancement in the study.

The subject programs were run six times with a minimum of two hundred test cases for every operation and the average execution time calculated.

7.3.2 Evaluation of Metrics

The metric that was used to evaluate the agent-based model against the general system was the execution time. The study computed the execution time in seconds as given in equation (7.1).

\[
\text{Execution time} = T_1 + T_2 + T_3 + T_4 + \text{Delay} \tag{7.1}
\]

where

- \(T_1\) - Time taken for the agents to initialize
- \(T_2\) - Time taken to invoke the coverage tool and compute the coverage.
- \(T_3\) - Time taken to prioritize the test cases.
- \(T_4\) - Time taken to compute the metrics
- Delay - Time delay due to other factors like machine time, tool initialization time, computation time, hardware time, and the like.

The execution time computed using equation (7.1) of the agent model for the application E-Book, is shown in Figure 7.6. This is compared
with the total execution time of the general model where more of analysis
time is expended by the tester in checking and transferring the correct
information to the next stage in the model. (that is, between T1 and T2 in
equation (7.1) there will be an analysis time, between T2 and T3 an analysis
time and so on). As a result, the execution time for the general model would
be increased. Figure 7.6 gives a detailed comparison of the two models and
from this figure it is clear that the general model takes more time compared to
agent model. The general model takes an average of 42% of more execution
time compared to agent-based model in total.

![Agent Model Vs General Model - Comparison](image)

**Figure 7.6** Comparison of execution time – agent-based model vs
general model - EBook

Similarly for the e-ordering application, 220 test cases were
executed on slots of ten and the total execution time for both the models
calculated using the equation (7.1) is shown in Figure 7.7.
From Figure 7.7, it is clear that, initially, the agent model required more execution time due to the initialization of the tools than the general model but as the process continues, the execution time was reduced. In total the general model takes approximately 16% more time compared to agent-based model.

The agent-based model thus reduces the time the tester has to spend during TCPR thereby reducing the cost and human resources and increases the quality of the regression process to a greater extent. This time may vary in real time environment as more number of test cases would be considered and prioritized.

### 7.4 LIMITATIONS

For the complete implementation of the agent-based model, some limitations are considered.
1. Only partial automation of the model was done, that is some manual intervention was required when there is a tool invocation like JUnit, Code Cover and DbUnit to generate and execute test cases and compute coverage and other parameter values. The model has assumed that all the required data are present in the repository.

2. The agent selects only the best technique for prioritization for the different operations identified in the study.

3. APFD values were given in the form of a table to the agent, that is, fault analysis was not performed by the agents.

All the above limitations could be considered as a future enhancement of the study.

7.5 CONCLUSION

The study has proposed an agent-based model to perform complete regression testing, particularly test case prioritization for software applications that involve various operations like basic, web and database. Experimental results show that the proposed model is more efficient in terms of execution time compared to the general model.

The model developed satisfies the following goals:

- **Comprehensive and Customized**: The proposed agent model establishes a conceptual foundation that supports the design and implementation of prioritization techniques for software applications. The model is also customized in order to handle the challenges associated with the testing and analysis of prioritization techniques for software applications.
- **Practical and Applicable:** The model is useful for practicing software testers. It does not require knowledge of formal specification or architectural description languages and does not force testers to write descriptions about the application to be tested.

- **Platform Independent and Portable:** Wherever possible the model could be implemented as it is developed in the platform independent Java programming language and JADE framework.

- **Efficient and Scalable:** The model uses software operations testing and analyses algorithms which are simple and scalable. The experimental results show that the model provides less execution time compared to general system which involves more of manual work to both small and medium scale programs. The results suggest that agent-based systems helps in automation, thereby reducing the cost, time and human resources required during the regression process mainly test case prioritization.