CHAPTER 13 – PROJECT EVALUATION

1 Introduction

The purpose of this project evaluation is to assess the various software methodologies that were used throughout the development of the framework, the accuracy of the estimations, and the usefulness of the reviews. The product will be reviewed and evaluated for whether it accomplishes the ideas presented in the initial overview and for the quality of the product.

Overall, the Spiral model development life cycle was used in the development of the framework. This model divides development into five main stages: requirements, design, coding, testing, and maintenance. The MSE project milestones correspond to a similar cycle: requirements, design, and implementation. All the required documentation for each phase was completed and reviewed by the committee members.

2. Estimation

2.1 Numbers of lines of code

During the first phase, the estimation of the number of lines of code for this project was 1500, based on the pre-design of class diagram. At the beginning of second phase the estimation was redefined, the number of source lines of code was 1610, based on the Function Point Analysis (FPA). The final product has come to the total of 4050 lines of code with comments and 2200 without comment. However, this project was created based on agentMom1.2 to extend the capabilities that has 300 lines of code without comment. Thus, the actual estimation was
1500 – 300 = 1200 (first phase)
1610 – 300 = 1310 (second phase)
and the total code that was created during this project was
2200 – 300 = 1900.

Therefore, the difference between estimation and actual result is approximately 600 lines of code. So, the number of lines of code is underestimated. One of the reasons for this might be the complexity of secured communication. The complexity of secured communication was not taken into account in the estimation. In the source code, the number of lines of code for secured communication is greater than the other type of communication. Moreover, another reason might be that the Java language factor used to convert function point to source lines of code during the estimation is 46 while several other sources use language factor of 50 for Java. If the number 50 were used, the estimation would come closer to the final product.

2.2 Time Duration

During the first phase, the estimation of time to complete this project was 715 hours, based on the COCOMO I model. The number of 715 hrs is based on Boehm that there are 152 working hours in a month, 152hrs/month*4.7month = 715hrs. The actual time spent on this project based on the time log is 719 hrs. The total time is very close to the estimation from the first phase.

During the second phase, WBS document was developed to estimate the time for the third phase. The estimation was 328, but the actual time is 386. The underestimation on this come from the testing and debugging process that took longer time than the estimation. The estimation for the testing and debugging was 88 hrs, but the actual time was 136 hrs. Other than the testing and debugging, the processes are completed as expected time.

However, the original date that this project was planned to complete during the first phase was the beginning of August of 2003, and then the date was changed to the middle of December of 2003 during the second phase. During the summer of 2003, I decided to take a summer job, so the project was paused, and the date was changed to December. During the fall of 2003, I had been sick for a few times, so the processes were slow, and the project could not be completed on December as planned. When the project was first planned and put into motion, none of these was expected or planned.

Please refer to section 10 for the time log break down on this project, including break down of each phase and total project break down.

The net effect is the project is finished, all other goals were met, and the time spent based on time log is fairly accurate despite the fact that the actual date is not met.

3. The Object Constraint Language

The Unified Modeling Language (UML) has become a de facto industry standard for Object-Oriented Analysis and Design. It is very powerful and expressive and can represent all necessary modeling needs for a project of this size and scope.

The Object Constraint Language (OCL) is more straightforward in approach and fits the UML modeling well. It has reasonable tool support with several parser available from downloads. USE is a very productive tool to simulate scenarios and test the models developed in OCL via UML. The only drawback for OCL and USE is the lack of documentation available if someone is learning these technologies without a strong background in formal methods and tools. More experience with UML and specifically OCL before the start of the project, would have been of great help. A great amount of time in this project was used to produce the OCL document. OCL can be confusing with little or no background using the OCL language.

4. Source Code

One of the goal in this project is to provide fully commented the source code. As agentMom 1.2 that this project based on provide very little comment in the source code. It is very difficult to figure out the intention of the original developer of what each member or method's responsibilities. I have learned a lot from this project on how to write good comments in Java and how to produce the JavaDoc. It is a very good lesson to learn that the comments are very valuable for the source code and for the future developer.

5. Time

The time spent in this project is shown in section 10 at the end of this chapter. From the total time used in this project, the majority part was documentation and coding (over 50%). The documentation took up the biggest part in this project. The problem could be that I do not have enough experience in developing the documents. Hence, the time used for producing document could be reduced with the experienced developers. Furthermore, the testing and debugging took up the biggest part during the third phase. Please refer to the "Testing and Debugging" part in the "Problem encountered" for more details on this.

6. Product Quality

With the amount of time spent in documenting, coding and testing, I believe the final product is fairly good within the frame of project requirements. I also believe that all the documents produced during this project will be very helpful to the developers who use agentMom.

However, this product could be improved in several ways. First of all, the new java.io package (java.nio package) could be used to improve the unicast and multicast communication, but the compatibility with the existing software will be lost. Secondly, the multicast and broadcast communications in this project are unreliable communication. The product could improve by building reliable multicast and broadcast communication. However, the agentMom package's size would significantly increase, and it would decrease the advantage of multicast and broadcast communication since it would introduce a lot of overhead in communication. Also, the reliable unicast communication is already available as another choice.

7. Name convention and backward compatibility

Because one of the requirements in this project is to be able to compatible with the previous version of agentMom, the name of classes, variables and methods are not consistent. For example, the Conversation class is responsible for unicast conversation, and the MulticastConversation class is responsible for multicast conversation. We can see that the class Conversation should be named UnicastConversation instead of Conversation. It is because the previous version of agentMom support only unicast communication, and it was not designed to extend the other types of communication at the first place. Thus, the names related to unicast conversation are not consistent with the newly developed conversations.

8. Security feature in multicast

Security feature in the multicast uses symmetric key algorithm. That is only one key is used to encrypt and decrypt message for all agents in the group. It is because we provide the SSL communication in secured unicast conversation, so agent can securely distribute the multicast key. Using public-private key algorithm in multicast would be really complicate, and it requires a lot of overhead and not effective because each agent has to keep track of the public keys for each agent in the group. For example, if an agent wants to send a message to other five agents in the group, this message is needed to encrypt five times using five different public keys. Then, the receivers have to ignore message that is not encrypted by the receivers' public key. Four messages are ignored.

9. Problems encountered

9.1 Undocumented Source Code

As state earlier in this document, the original source code contains very little comment. Although, there is a user's manual for agentMom 1.0, the actual source code had been changed to version 1.2. It took quite a long time to fully understand the source code, and be able to use it. The lesson learns from this greatly changes the style that I used to code.

9.2 Lost of Data

I had experienced the effect of the lost of data two times without a recent backup. Both of them were during second phase. The first time was caused by the Eclipse development tool. I have used Eclipse to develop both the diagrams and source codes, and the synchronization between diagrams and source codes is very useful. However, I did not realize that the source codes would be deleted when I deleted the diagram. So, a great amount of time was spent to rework on the source codes. The second time was caused by hard drive failure. Although, I had a recent backup, but the environment was lost. It took many weeks to install and regain the environment. The lesson learns from this problem is that environment greatly affecting the productivity, and environment configurations are also needed to backup.

9.3 Testing and Debugging

As the agentMom package grew in size, it became incredibly hard to trace bugs. Also, the product is actually a framework that allows other developers to easily create multiagent systems using MASE methodology. Thus, to fully test the framework, the multiagent systems are needed to develop. This increase the time in testing and debugging because the bugs might come from the implementation of the multiagent systems, not from the actual framework, as I experienced many times during the testing. As we can see from the third phase's time log break down in section 10, the biggest part is the testing and debugging process.
Moreover, testing and debugging are incredibly hard because there is no closed network available for testing. I have to use the open network in the CIS department that I have no control over the network configuration. When an agent does not receive the message, it could be many reasons, bugs in the software or the network environment (packet was dropped). The lesson learn from this problem is that closed network is very useful for multiagent system in testing before test it in an open network because many factors can affect the result, and we cannot properly narrow down the source of problem. This is the most important reason why the testing took longer than the estimation.

10. Time log Break down

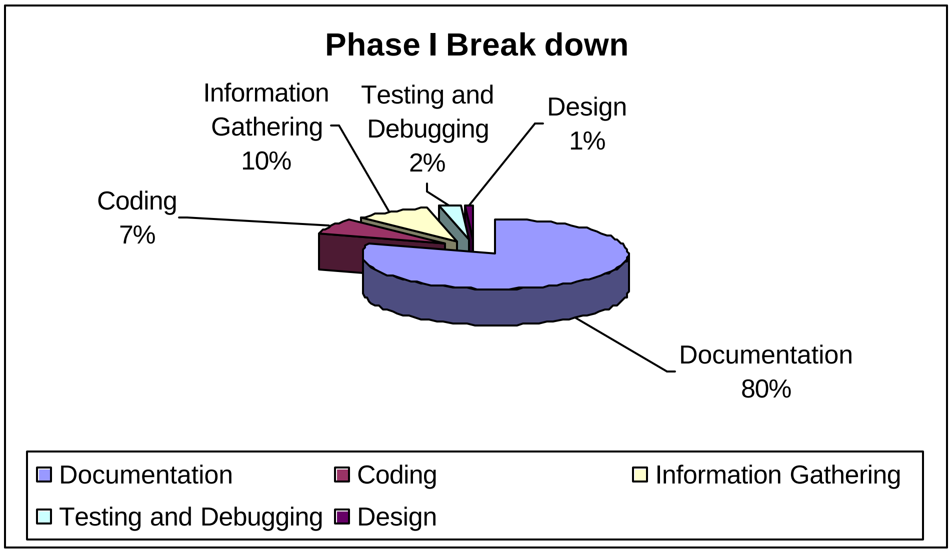

Figure 22 Phase I Break down

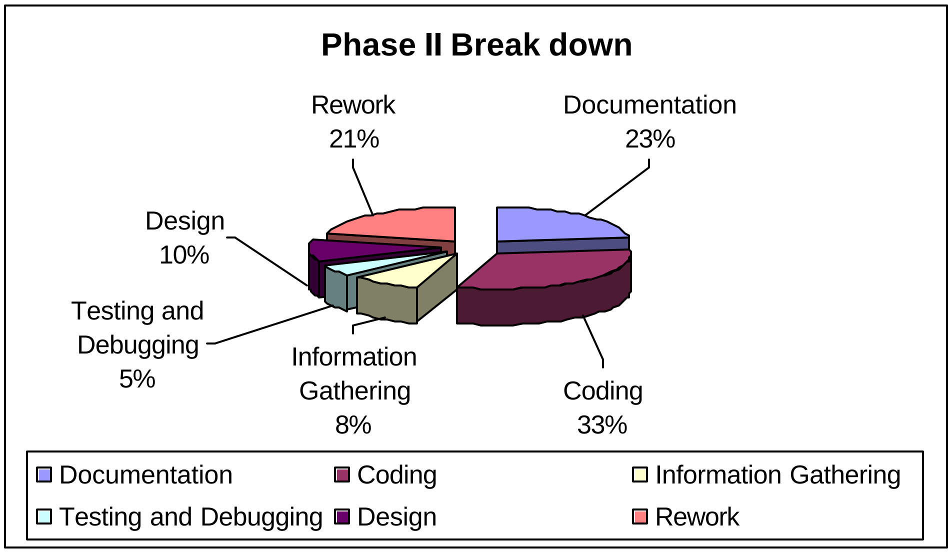

Figure 23 Phase II Break down

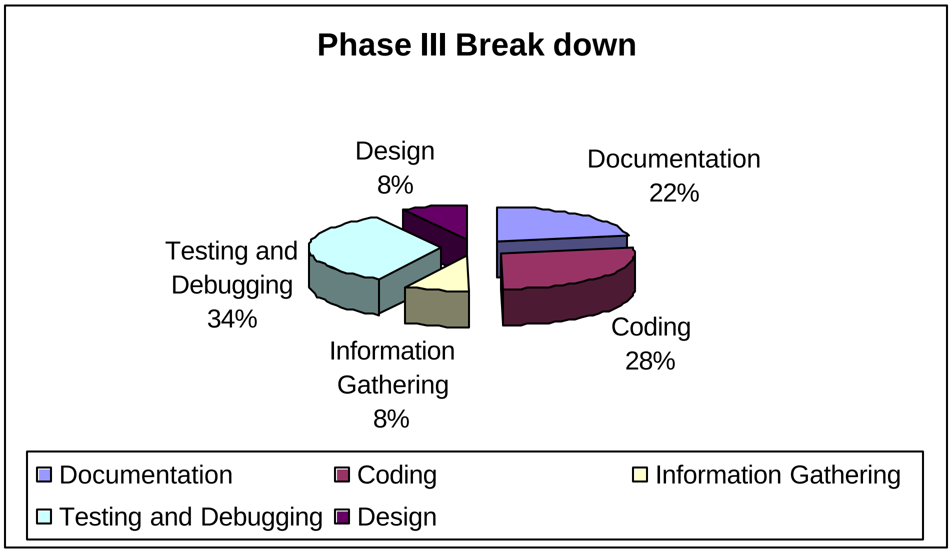

Figure 24 Phase III Break down

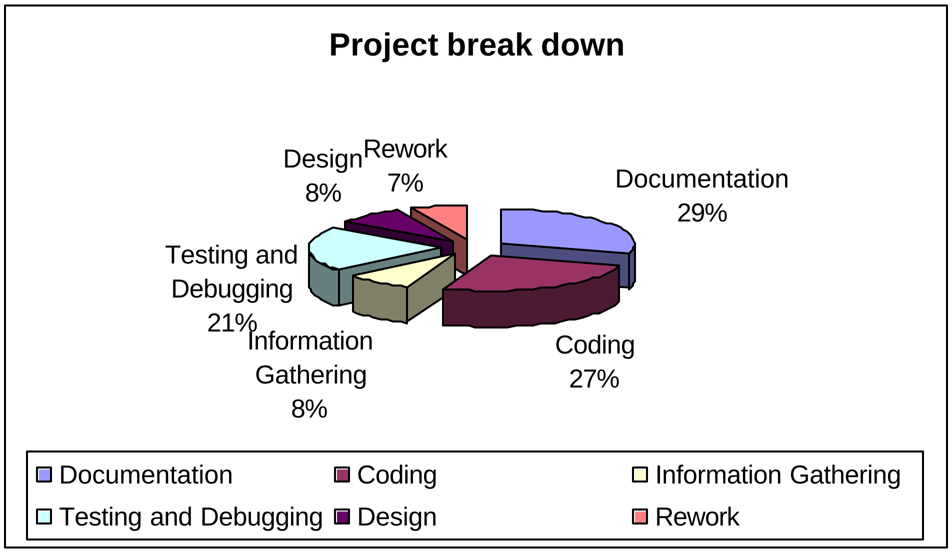

Figure 25 Project Break down

REFERENCES

Amund Tveit. “A survey of agent-oriented software engineering.” NTNU Computer Science Graduate Student Conference, Norwegian University of Science and Technology, Norway, May 2001.

IEEE Std 829-1983, "IEEE Standard for Software Test Documentation". 1983 Edition, IEEE, 1983.

IEEE Std 730.1-1995, "IEEE Guide for Software Quality Assurance Planning". 1995 Edition, IEEE, 1995.

IEEE Std 730-1998, "IEEE Standard for Software Quality Assurance Plans". 1998 Edition, IEEE, 1998.

IEEE Std 828-1998, "IEEE Standard for Software Configuration Management Plans". 1998 Edition, IEEE, 1998.

IEEE Std 830-1998, “IEEE Standard for Software Requirement Specification”. 1998 Edition, IEEE, 1998.

OMG Unified Modeling Language Specification, Version 1.5, (http://www.omg.org)

Scott A DeLoach. “agentMom User’s Manual.” July 2000.

Scott A. DeLoach, Mark F. Wood & Clint H. Sparkman, Multi-agent Systems Engineering, The International Journal of Software Engineering and Knowledge Engineering, Volume 11 no. 3, June 2

Walker Royce. “Software Project Management: a unified framework.” Fifth edition, Addison-Wesley, 1997.