AGENT ARCHITECTURES (BUILDING CHATTERBOT)

UMI AIMAH ABU BAKAR

FACULTY OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2002/2003

ABSTRACT

A chatterbot named MILAH is a chat robot that interact with user by chatting. The main idea of this project is to build a chatterbot that can speak, learn and understand Malay language, and to study more about the chatterbot architecture. MILAH uses ALICE bot technology and a special markup language, AIML (Artificial Intelligence Markup Language). The difference between MILAH and other chatterbot, other chatterbot use complex and difficult AI techniques and it is difficult for botmaster or the administrator to maintain. While MILAH, it has no neural network, no complex knowledge presentation, no fuzzy logic, no genetic algorithm and no major parsing. MILAH uses Case Based Reasoning (CBR)

There are a few advantages that are crucial to ensure the success of this kind of chatterbot. Most importantly, user can speak Malay with MILAH. MILAH will entertain user as if she's a real person using its very own knowledge and sometimes, MILAH can give surprising answers to users as if it can think. This chatterbot is another new modern way of entertainment for new generation.

This report introduces the project and provides a description on the topics studied and researched during the literature survey. The methodology for the system development is also discussed, which is the Waterfall Model with Prototyping. Finally, this report also included the system development tools chosen plus, the details of system design for MILAH chatterbot.

ACKNOWLEDGEMENT

During the development of the project's report, I have been fortunate to have many sources of inspiration and support. Throughout the duration of the project development, many people had been very kind and helpful, giving invaluable advice and encouragement.

First and foremost, I would like to thank my supervisor, Mr. Ridzuan and my moderator, Dr. Syed Malek Fakar Duani Syed Mustapha, for without insightful ideas and guidance, the success of this project would not have been possible.

To my partner, Sharina Ros Kamal, thank you for your constant commitment and patient towards our project, I wouldn't be able to do it without you. To my family, thank you for your support and encouragement. And lastly, I would also like to express my gratitude to my friends and fellow course-mates who have helped me in one way or another.

Thank you.

Umi Aimah binti Abu Bakar

WEK 000217

CONTENT **PAGE**

CHAPTER 5 : SYSTEM DESIGN

CHAPTER 6 : SYSTEM IMPLEMENTATION

CHAPTER 7 : SYSTEM TESTING

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

This chapter features a full description of the project including the definition, the goal, the objectives and the scope of the project. It also features the outline of the project's implementation chart with all diagrams and timetables necessary.

1.1 Project Overview

Communication must have been one of the most complex art or science, whatever we might call it, practiced and developed by humans. Exchange of ideas and thoughts make (and always did) possible the accumulation of knowledge, which is certainly the single most important factor for the advancements that we are always proud of. One thing that's done for most of human seconds around the world is, of course, chatting (of course in different flavours, "talking", "discussing", "debating", "delivering", as you like it). Though purists would not shore this view but we hold that one can't chat to "himself". Thus we all need to be chatting to "someone". But, humans come with a host of attributes, have their constraints, limitations, are moody and tend to get irritated when told something they don't relish. This was an observation recognized long ago. We had made movies exhibiting a sensible talk between men and machine. This is really exciting because fortunately the machine is built devoid of the limitations that exist by virtue of human nature. A machine would talk to us as and when we want to. Humankind really

graduated on this path, when some really talented computer scientist set out to find an implementation that is no longer mere fiction, that is to build a chatterbot.

What is a chatterbot? A chatterbot is a computer program for simulating conversation between a human and a machine. A user input a question or statement of any kind, and the chatterbot replies, just as a person would by using its own version of logic. Chatterbots try to create the illusion that an authentic exchange is taking place between two thinking, living entities. Sometimes, chatterbot can trick human by posing its intelligent.

The origins of chatterbots can be traced back to 1950, when the British mathematician Alan Turing famously asked the question: "Can machines think?" It was a good question, and many people have since spent considerable amounts of time and effort in trying to prove that the answer is 'yes'. Researchers in artificial intelligence have devoted much time and effort to trying to understand human cognitive capacities and adapt them to machines. Chatterbots represent just one aspect of this research.

The first chatterbot, named Eliza, appeared all the way back in 1966. Eliza was created by Dr. Joseph Weizenbaum of the Massachusetts Institute of Technology, and was intended to resemble a Rogerian psychologist. When a human spoke to Eliza, she returned the sentence in the form of a question, thus inviting the user to give further explanation, ad infinitum. Not exactly a high level of conversation, but nonetheless

ingenious and sufficiently 'intelligent' to cause confusion at a time when people were not used to interacting with computers.

Some say that chatterbots have struggled to go beyond the level of Eliza. But if we consider the question in terms of developing applications, rather than robotic human beings, we can see that chatterbots increasingly have a role to play in humanizing the Internet. With the explosion of the Web, more and more chatterbots are making their appearance on-line. So have they risen to the challenge lain down by Turing? Can machines think yet?

Chatterbots, it must be remembered, are pieces of software. Using natural language processing, their program attempts to identify what a user says - the input - then applies a variety of different methods, from pattern matching and keyword identification to neural networks and fuzzy logic, to produce an appropriate answer. No chatterbot can claim to have human capacities of reasoning and deduction, and by no means are yet able to reproduce all the complexity of the human brain.

How intelligent user find them will depend on:

- the technology on which the chatterbot is based
- the quality of the knowledge base - the brain - that has been developed by the botmaster
- what you expect from them

Developers have found a number of solutions which enable chatterbots to imitate human conversational ability, based upon a range of tricks for avoiding difficult questions and sticking to areas that they can handle, which in some ways is a proof of intelligence in itself. These include, but are not limited to:

- making controversial statements to provoke a response
- agreeing with the user, rather than being non-committal
- repeating user input in their answers to make it seem as if they are following the conversation
- remembering and reusing past topics of conversation
- changing the topic when they don't understand
- being random, just as humans are
- being abusive, just as humans are

All chatterbots are not the same. Here is a brief topology of the different bots available:

- On or off-line

  Chatterbots can be part of a web site and you can chat with them on-line. This is the most common form of access at the current time. Some people even put their bots on Instant Messenger or IRC.

> Conversational chatterbots aim is just that, to chat. They are often the result of the work of academic search, or a passionate individual or groups seeking to push back the barriers in communication between men and machines. This is no way excludes conversational chatterbots from having commercial applications, or being able to provide certain services such as launching programs on your computer.

This topic was chosen because it was never been constructed in such way using Bahasa Melayu ever. By building this chatterbot, contribution can be made in the IT world in Malaysia by doing something new rather than mundane.

In the IT scope, this chatterbot will appear as one of the competitors among other chatterbot out there. It will also give a new dimension since it will be constructed in Bahasa Melayu.

1.2 Project Objective

This project would be an attempt to realize the fancies that spring up after reading the project title. I hope to come up with a chat robot that can understand, speaks and learn Malay language and is capable on intelligent talk.

The objective of the thesis itself is to give some exposure and to give the opportunity to achieve more experience to the students. Thus, enhancing the knowledge that is already there inside the students. In doing the research about the topics given for the thesis, disciplines and ethics will slowly be assimilated.

Others can be downloaded to your computer.

- Type of interface

How do we communicate with a chatterbot? Frequently, by typing a text into text box. This method is effective but not very convivial, so many chatterbots have an animated figure to encourage interaction, using a photo-realistic image, a cartoon or some other graphic. Some chatterbots have a Text-To-Speech interface, so that you can talk them through a microphone and hear their replies. This conversational interface can add the illusion of the reality.

- Role

Chatterbots available today have two main roles:

1. Commercial

Commercial chatterbots are designed to fulfil specific function, for example to be part of a Web-based customer relation management strategy. A tireless employee, working 24/7, they offer a pleasant and useful first contact to customers, and are able to deal with the most common problems or requests for information. The objective of these chatterbot is to be good at their particular job, rather than to be an expert in general conversation, though once again, nothing prevents them from being able to talk about a wide range of topics, if their wishes.

2. Conversational

By choosing chatterbot as a research for the third year thesis, student will be exposed to the convention of machine agent that can communicate with human using methods, which will be elaborate, further more in this thesis. Through this, student will also learn how to adapt words, sentences and even grammars to the machine language. Lots of artificial intelligent concepts will be permeated here and in the mean time this will give more information and knowledge about things, which have been educated in the lecture hall. Furthermore, this research will introduce an artificial intelligent student to the world of 3-dimension animation, which was never been taught in the lecture before.

Below are summary of my project objective written in point form for better understanding:

- To develop a chatterbot that can speak, learn and understand Malay language
- To do research and study about the chatterbot architecture
- To minimize or simplify all those complex AI techniques implement in developing a chatterbot
- To introduce my Malay language speaking chatterbot to the IT world

1.3 Project Scope

A small scope will be given to the chatterbot about the area it will hold. This is because if a big scope is given then it will be fastidious to determine and arrange all the words and grammars. It will also take more time to be developed.

This chatterbot will mirror an image of a girl who can speak, learn and understand Malay language. She hope that user will zealously enlighten her more in favor of adapting this language.

It will also put her forth effort to answer all the enquired by pertaining to the knowledge admitted in her brain by the botmaster or the users earlier.

The scope of building this chatterbot also is mainly to study the architecture. This is also a stand-alone system and not a web based system.

Also. I named my chatterbot MILAH.

1.4 Target User

This chatterbot is expected to be the first Malay speaking chatterbot. Its target user is:

- All genre of society who love to chat, unconditionally whether it`s a human or not
- Everybody especially Students who would like to learn or teach this chatterbot Malay language
- Computer Science or AI students who would like to discover and deepen their knowledge about chatterbot
- All Malaysian who would like to try chatting with a bot that can speak Malay

1.5 Project Schedule

Figure 1.1: System development chart

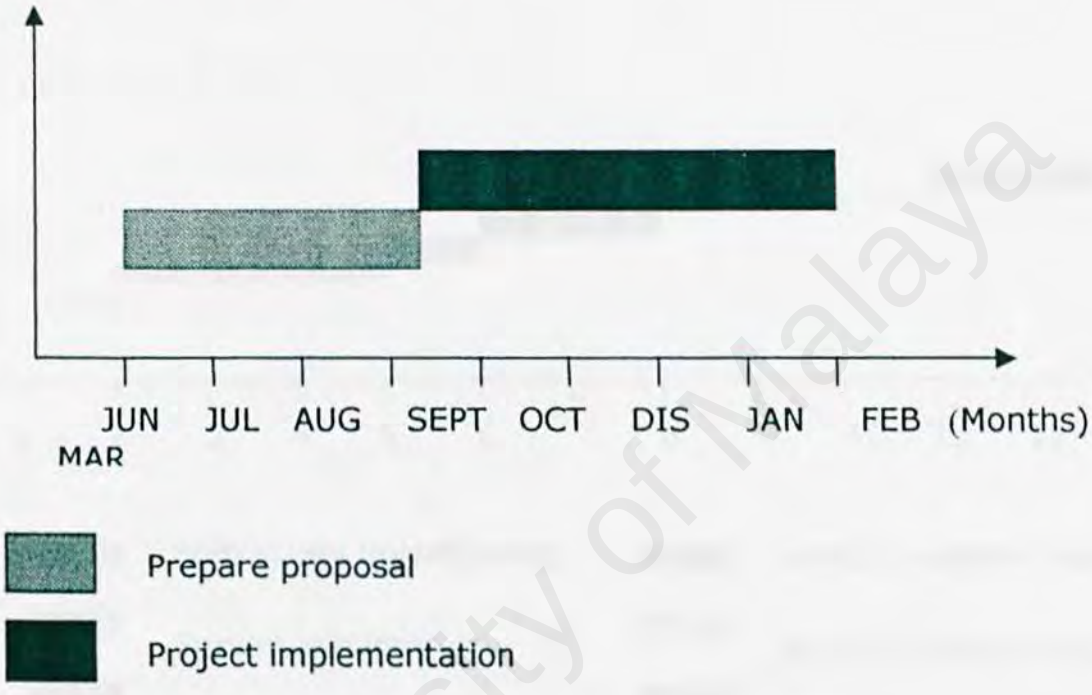
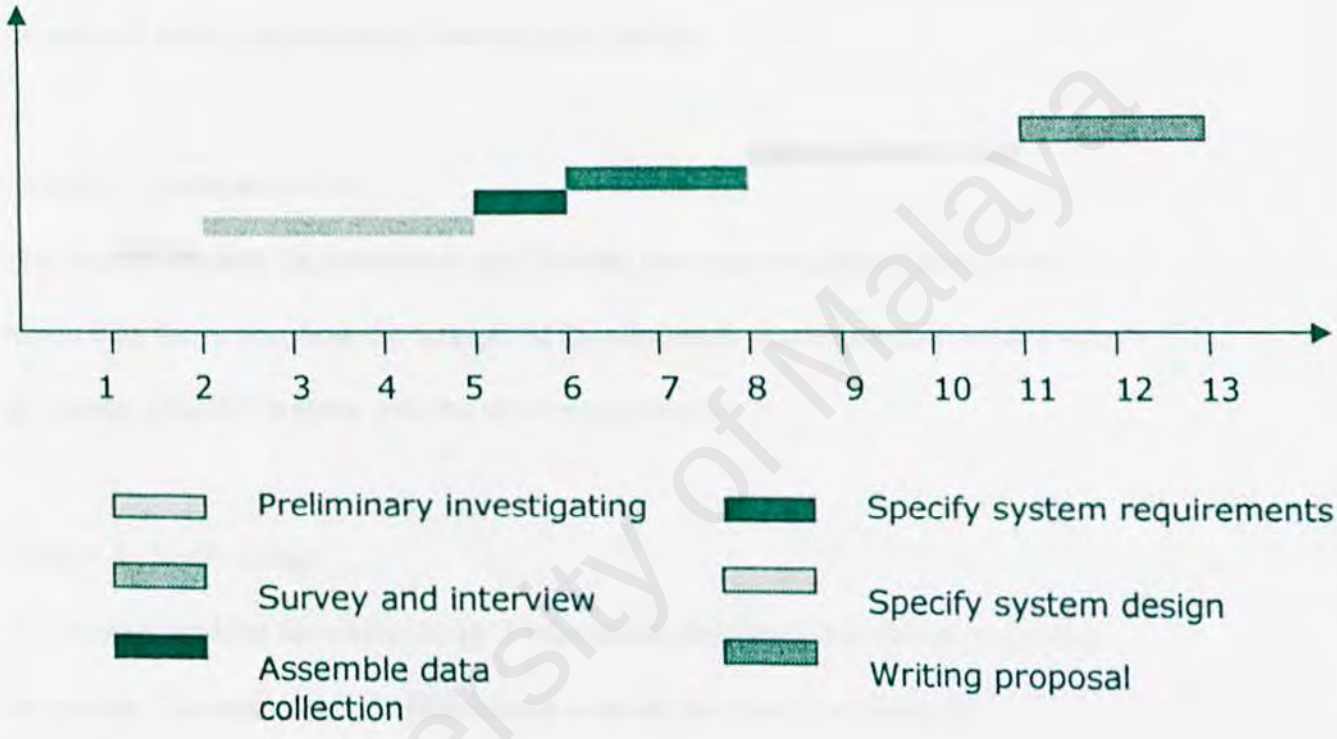


Figure 1.2: Proposal timeline



1.6 Chapter summary

Chapter 1 : Introduction

This chapter introduces & fully describes the "chatterbot" including the objectives, scopes and also the entire system development schedule.

Chapter 2 : Literature review

This chapter explains the literature review that has been done to get the information required by the system, how the data gained & collected & also the comparison between the current available systems with the developing system.

Chapter 3 : Methodology

This chapter explains the methodology & approaches that have been chosen to develop the system. This chapter also explains about what are the reasons of using the methodology chosen to develop the system.

Chapter 4 : System Analysis

This chapter is about analyzing the system requirements including the functional & nonfunctional requirements and software & hardware requirements. This chapter also explains about why these requirements are needed.

Chapter 5 : System Design

This chapter explains the concepts & design techniques of the system. It includes structure chart, DFD, process flow chart, user interfaces & database design.

1.7 Summary

From this first chapter, the objectives, scopes, purpose & full description of the project have been explained. The system development project is also included.

CHAPTER 2 : LITERATURE RESEARCH

This chapter will describe in detail the various studies and research done of the topics or existing chatterbot, the websites, the internet, software and technologies. It is the objective of this chapter to outline systematically all these studies so that it will assist in the proper selection tools and development methods plus to understand the strength, weakness, opportunities, potential and current issues about chatterbot.

2.1 Introduction

Literature research is an overall research that had been done to a system that will be develop. The purpose of this research is to :

- collect information about the system that will be develop
- research and evaluate all those system that has been developed, and have the same concept or relevant, so that we can detect the weak points and the advantages of the system and also to improve those weakness on our system that will be develop
- get a clear and better understanding about the concept involved in the system that will be develop and to compare few software that might be use during the system development so that the best solution can be decided

2.2 Research Method

Generally, system development will not be complete without gathered information and research on the system that will be developed. Thus, the gathered information is vital for the system to achieve its vision and objectives. This information can be obtained through several sources and each sources give different information plus it needs different searching techniques. Several methods have been used for the research and the analysis towards the ready-made system and the system that will be develop. Among the method are collected data and writing method.

2.2.1 Collected Data Method

- **Internet Surfing**

Internet has been a lot of help in this chatterbot development. By surfing the internet, a wide variety of information can be obtained for the purpose of this research towards the existed system for comparison.

2.2.2 Writing Method

- **Written Items Analysis**

Analysis has been conducted towards the collected data by summarize back the data and the information to make it easy to understand and correctly match the objectives of the project development.

- **Comparative**

Results and summary is made by the comparison done towards the existed system and the system that will be developed using the obtained data.

- **Documentations / Books and Magazines**

Analysis and research have been done towards the documents and the written items that have related topics with the system that will be developed. Collected information from books and magazines is done for better the results of the research. These documents are obtained from the University Malaya's library for references, through personal collection and friends.

2.3 Definitions

- ***Bot***

A word short of robot, is a software tool for digging data. You give bot directions and it bring back answers.

- ***Agent***

A program that perform some information gathering or processing task in the background. Typically an agent is given a very small and well define task. The term bot has become interchangeable with agent, to indicate that the software can be sent out on a mission, usually to find information and report back. So, an agent is a bot that goes on a mission.

- ***Chatterbot***

A chatterbot, or a chat robot is a computer program for simulating conversation between a human and a machine

2.4 Research on Bots Development Tools

2.4.1 ALICE

2.4.1.1 Introduction to ALICE

Figure 2.1 : ALICE bot

A.L.I.C.E (Artificial Linguistic Internet Computer Entity) is a natural language interface which is connected to a telerobotic camera eye. And one of the great things about her is that she will explain a lot about herself in response to questions like 'who made you?' and 'what is your purpose. She attempts to paint a small picture of what Han

Moravec and Arthur C. Clarke predicted. The ability for humans to transfer the contents of their minds into computers and thereby transform themselves into immortal robots.

The Alice program was designed by Dr. Richard S. Wallace of Lehigh University. Dr. Wallace wrote the program In Java and released the source code as Open Source in effort to let the curious people learn how Alice worked, as well as to let other people refine and enhance the program.

The basis for Alice's incredible life-likeness is surprisingly simple : AIML, or "Artificial Intelligence Markup Language". AIML is a simple markup language based on XML, that allows a person to tell Alice how to respond to various input statement. By creating a file that contains AIML describing how Alice should respond to a variety of sentences, we can make our own chat robot.

Her knowledge base consists of thousands of facts, quotes and ideas that were "transferred" (by typing and OCR) from brain storage into storage on the server by Dr. Richard S. Wallace. Part of the interface is natural language; so there is no such thing as an 'error message' as Alice (with a vocabulary of about 5000 words), attempts to reply to any remark, question or request.

The Alice Web interface is somewhat like a chat-line designed to normalize the differences between human and machine intelligence. As the name suggests, it is technically possible technically possible for a human to go on line and reply to Alice's

queries himself. Most of the time the server runs autonomously, but a super-user can monitor conversations and interrupt Alice if he wants to talk live, though in practice this seldom occurs. Thus, Alice is designed to confuse the client's feelings about whether it is a person or a machine.

Alice bot development program can be downloaded to a computer from the internet. User can start creating their own chat robot from scratch, or can simply add their own AIML to Alice's current personality.

2.4.1.2 ALICE Bot Development Program

- **Program A**

The first edition of A.L.I.C.E. was implemented in 1995 using SETL, a widely unknown language based on set theory and mathematical logic. Although the original A.L.I.C.E. was available as free software (often misnamed "open source"), it attracted few contributors until migrating to the platform-independent Java language in 1998. The first implementation of A.L.I.C.E. and AIML in Java was codenamed "Program A".

- **Program B**

Launched in 1999, Program B was a breakthrough in A.L.I.C.E. free software development. More than 300 developers contributed to A.L.I.C.E. Program B. AIML transitioned to a fully XML-compliant grammar, opening up a whole class of editors and tools for AIML development. A.L.I.C.E. Program B won the Loebner Prize, an annual Turing Test, in January 2000. Program B was the first widely adopted free AIML software.

- **Program C**

Jacco Bikker created the first C/C++ implementation of AIML in 2000. This was followed by a number of development threads in C/C++ that brought the Alicebot engine to CGI-scripts, IRC (Anthony Taylor), WxWindows (Philippe Raxhon), AOL Instant Messenger (Vlad Zbarskiy), and COM (Conan Callen). This collection of code has come to be known as "Program C", the C/C++ implementations of the Alicebot engine and AIML.

- **Program D**

Program B Java edition was based on pre-Java 2 technology. Although the program ran on many different platforms, it did not take advantage of newer Java features such as Swing and Collections. Jon Baer recoded Program B with Java 2 technology, and added many new features. This giant leap in the interface and the core, plus the fact that Jon named his bot "DANY", justified

granting the next code letter "D" to his latest Alicebot Java edition. Beginning in November 2000, Program D is the only the Java edition of the Alicebot engine actively supported. The currently actively supported, actively developed Java implementation of the Alicebot engine. This is the version to get if you want to use the latest technology, especially if you want to participate in Alice`s development.

- **Program E**

Program E (formerly known as "PHiliP") is a PHP implementation of Alicebot engine that inspiring a lot of excitement. Although it is still in its very early stages, it is generally compliant with AIML and has a fast-growing user community.

- **Program V**

Program V is a Perl implementation of an AIML interpreter. It is AIML 10.1 compliant. V relies on the industry- standard use of mod_perl (for Apache) so that web server can handle many of the non- AIML-related tasks that other interpreters (like Program D) handle themselves.

- **J-Alice**

J-Alice is an Alicebot engine written in C++. Unlike Program D, it doesn't have a web server, nor can it yet be incorporated into a website. It does,

however, have a built in IRC Client, so that it can chat on IRC.

- **AliceMac**

AliceMac is a Mac OS implementation of the Alicebot engine. It is not currently AIML 1.0.x compliant.

2.4.2 ELIZA

2.4.2.1 Introduction to ELIZA

Eliza is was the first psychologist program in the world, and in fact, the first program that allowed a human to converse with a computer. Its mission was to attempt to replicate the conversation between a psychoanalyst and a patient. It used Artificial Intelligence (AI) in order to try and imitate a real human psychologist. Joseph Weisenbaum at MIT was the head of that project in 1967. It used an algorithm called CBR, or case based reasoning. The original Eliza program and all the ones who followed it were mostly crappy little programs, which would repeat themselves after a few minutes of conversation. Another problem was their very limited understanding of the user's replies. All in all, those attempts were quite futile, but they did in fact lay the foundations for modern more advanced AI programs.

Figure 2.2 : ELIZA bot

Eliza works by turning everything you say around. For example:

If you type in: I am having a very bad day

She may reply: did you come to me because you were having a very bad day?

Or

If you type in: I have come to talk to you.

She may reply: oh, you have come to talk to me ?

By doing this Eliza is able to talk you in to giving her your deepest feelings. She does not remember it and keeps no recollection of anything you say, but sociologist Sherry Turkle noted that people actually became protective of their relationship with her, mainly because everything you say becomes a personal question.

This can lead to a very interesting conversation that makes her seem so friendly and understanding. However this does have some pitfalls:

If you type in: I am doing fine thank you.

She may reply: how long have you been doing fine thank i?

So every now and again,Eliza displays a severe lack of understanding.Which does not seem too bad as you can soon learn your way around this problem. However, once Eliza has repeated a modified version of your sentence for the fifth time in a row you do begin.

How does it work.

The human side :

Humans expect what they see to be true. When using a computer to chat to "someone" else, particularly if they've never met the person, they are very willing to tell a lot about themselves. usually the conversations can be very long and tiresome,

The computer side:

The computer patiently sits and "listens" to the entire conversation. When certain keywords come up, it will respond with some catchy remarks. If the list of keywords and responses are thorough, the computer should be able to fool the user in thinking they're talking to a real machine.

Fooling the user ? :

That's correct. This project is not about creating Artificial Intelligence as such, nor is it about creating thinking machines, it's about fooling the user in thinking they are talking to a real person. Computers can't have emotion or personality, but they sure can simulate it, and that's where this project comes in.

Not fooling the user? :

In some cases, the user is not fooled in thinking they're talking to a real person. In some cases, they want to talk to a computer, and try and have a conversation with it.

2.4.2.2 ELIZA Bot Development Program

The system has been developed to run in Perl. Most Unix and Linux platforms have Perl installed. If you're from a Windows platform, you can download ActiveState Perl and still use all the scripts on this site. I'm busy investigating a compiler for Perl scripts, that will allow you to run the code without the need for ActiveState Perl.

2.5 Current Existing Chatterbot

2.5.1 Sabrina – The NLP Multibot

Sabrina is an artificially intelligent computer program. She's a bot, meaning she can act on her own and perform tasks for humans. In fact, she's a bot made up of many smaller bots. Sabrina is also a grand illusion. A piece of software isn't a "she".

She's a work in progress. They've been surfing the net and camping out in the library for months now, gathering all kinds of great ideas about what artificial

intelligence can be. They've found that there's a lot of great ideas out there that no one in this corner of the AI field - the corner where software meets human language - is really paying attention to.

Like Jorn Barger's anti-math and fractal thickets. Like fuzzy logic. Like hypnosis, neuro-linguistic programming, and speed seduction. Like hypertext and client-server architecture. Like genetic algorithms, expert systems, and neural nets. Or, on a different logical level, like open source software.

It seems most of the other programs like her tend to be rehashed Elizas. Well, Eliza's fine for what she was. There's a lot of good that can come out of an Eliza-like program. In fact, that's how Sabrina started out. But Eliza only gets so far into a conversation before her illusion falls apart. She doesn't work for what we want her to do. Well, in NLP, we have a philosophy. If what we're doing isn't getting us the results we want, we do something different.

Sabrina puts all this technology that's been lying around - and more - to good use. The goal in creating her is to build the most advanced AI program in existence: a virtual person capable of learning, exploring, understanding and telling stories. And that's just the beginning.

Sabrina's mission in life is simple. She's going to convince the world she's human.

Some characteristics:

- A 64-byte array of short integers representing an internal "state"
- A dictionary containing a word list and a variety of corresponding 64-byte "Anchors".

- Ability to take input (a word (in any language!)) Then updates the word's anchor to this new state.
- A Markov Chain for states: given state x, what is the most appropriate response? Or perhaps for each sub modality? chunks of the 64-state.. A big 64x256x256 array of percentages? Or given this state, what's the most likely change?
- System that acts, based on current state.
- Integration with MOO virtual reality. Able to walk around, talk, explore, etc.
- or perhaps state compression or generalization "compressed" or generalized states. That is: 000001 is more or less like 000000 unless that last digit is either-or .

Modes and Models :

1. Conversational Framework / Strategies

    - consultant
    - therapeutic
    - chattery
    - storytelling

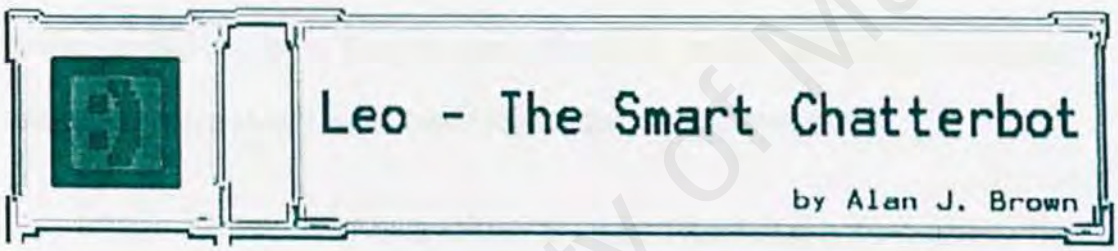
2. Sentence-by Sentence

    - Metamodel mode

- Eliza mode
- Milton mode
- Jeffries mode

Try to surf the website and test Sabrina yourself : **http://fury.com/osabrina**

2.5.2 Leo the Smart Chatterbot



The author calls Leo "by far the most advanced learning bot on the web". Leo does not have a complicated set of rules dictating what to say when certain predefined key words are triggered like many chatter bots on The Simon Laven Page. That is the way Instead Leo learns from the users who converse with Leo on their own computer.

Leo is the revolutionary new chatterbot from the creator of CoLIN. Although based on CoLIN, Leo represents a quantum leap in terms of conversational quality.

Leo also has the ability to handle conversation topics better. He can easily refer to something that was discussed a few replies ago but was not mentioned in the

submitted sentence. Grammatical accuracy is also greatly improved.

Leo is by far the most advanced learning bot on the web. He is so good that it can be hard to believe that some responses are not preprogrammed. Everything Leo knows comes purely from conversations with the user. He also retains CoLINs ability to learn any language.

Instruction. Leo does not have a complicated set of rules dictating what to say when certain predefined key words are triggered. That is the way most chatterbots are written. Instead Leo learns from the user as the user converses with Leo. Thus Leo gets better over time, and will learn about subjects that you are interested in.

When you first start Leo, it already has some information in the database. This enables you to jump right in and start talking. It is possible to clear the database using the option on the file menu. Use this option with caution. You may want to do this if you want Leo to talk in a different language.

Starting from a clear database can be frustrating. Because Leo doesn't know much it will only repeat back what you say to it. After a while it will start to use sentences from further back in the database. Eventually the database will be large enough for Leo to form its own sentences. This takes time which is why Leo is supplied with demo data. Try to stick to one sentence at a time. Avoid one word replies.

If you want to teach Leo something then just type lots of facts at it and ignore the replies, eg :

Tony Blair is a politician.

He is the prime minister.

The Labour party is in government.

The prime minister heads the government.

The leader of the Labour party is Tony Blair.

If what the program says really does not make sense, just make a guess at what it means. If you get bored, start insulting it.

History. Leo is based on CoLIN. Unfortunately it wasn't very good. I came up with a few ideas on how to improve CoLIN and tried them out. The result was such a radical improvement that I felt it really wasn't CoLIN anymore. Thus the change of name to Leo. That and I discovered and older bot named Colin on the net and I wanted to avoid confusion.

Download. The file is 3.7 megabytes in size and will take around 12 minutes to download on a 56k modem. Once the download has completed, click on the file and the setup files will uncompress and begin installing the program automatically. This program will run on Windows 95, 98, ME, NT4 and 2000.

Leo Engine. Leo is now built using LeoEngine, the new DLL based method for accessing Leos functionality. Installing Leo automatically gives you the ability to use LeoEngine in your own programs. In VB start a new project and in the project menu click references. Scroll down to "Leo Chatterbot Engine" and check it. The following code is an example of how to use LeoEngine :

```
option explicit
public le as leoengine

private sub form_load()

set le = new leoengine
le.leodatabase = "c:\myprogram\lvd.mdb"
text2 = le.leotalk(text1)

end sub
```

Other methods include leoreplace and wipeleo. The read only property stops new sentences going into the database. If you use the LeoEngine DLL in an application then you must display the Leo logo (if your app is graphical) or the Leo website address along with "LeoEngine ?2002 Alan J. Brow.

Leo can be downloaded from **http://www.barc0de.demon.co.uk/leo/**

2.5.3 Billy and Daisy

Although Billy and Daisy are technically two different chatterbots, they are really evolutions of the same code, hence my decision to put them together.

Billy, much like other chatterbots works by taking the user input and breaking it down to individual words, thus is able to identify certain parts of language. It then uses this knowledge to try and generate a set of pre-defined keywords that should be included in the final response. These keywords are then given to a subsystem that is able to analyze everything that has ever been said to Billy, and so can synthesize a free-form sentence of its own creation which contains the keywords and has a high probability of being grammatically and contextually intelligent.

Other features include an ability to stay on subject during a conversation, and an enhanced system for answering questions and learning facts. The current version of Billy also introduces the concept of mind files which contain his knowledge, language, and personality. These can be easily transferred to other users of Billy via the internet or disk.

Daisy, however, has no pre-programmed language of any kind. While Billy contains many responses, which are programmed to be said if the user types in certain sequences of words. Daisy contains no built-in responses. She starts with no knowledge of anything, but then is able to gain knowledge as she observes what humans say.

It can take a little while before Daisy has gained enough knowledge to be able to create her own sentences. To get you started, the Daisy distribution file contains one memory file, which was built by having Daisy talk to me. However, you are always able to create a new file and start from scratch. You can download both Billy and Daisy from Greg Leedberg's website at **http://leedberg.com/glsoft/daisy/index.html .**

2.5.4 John Lennon

John Lennon Artificial Intelligence Project (JLAIP™) is recreating the personality of the late Beatle, John Lennon, by programming an Artificial Intelligence (AI) engine with Lennon's own words and thoughts.

The website : **http://triumphpc.com/johnlennon/index.shtml**

Figure 2.3 :John Lennon bot

2.6 Existing Chatterbot Drawback

There is hundreds of chatterbots, whether online or you downloaded from CD or internet. There are few chatterbots, I have to admit that it is quite competitor to my future chatterbot. But I have to add, from all the chatterbots I observe and learn, there is one common purpose between us and that is to educate people, provide knowledge about the architecture and to introduce our own chatterbot personality.

Below are summaries of what I found out on the existing chatterbot drawback. They are all in point form for better understanding.

- Boring and dull interfaces
- Most programming languages used in other chatterbots are complex and difficult to maintain
- Can only be programmed to speak one language, mostly English and can't handle multiple language simultaneously
- Some chatterbots are very dumb, they don't really interact with users
- Replying time by a chatterbot is too long
- It's not suitable for every level of intellect, because some chatterbot can speak harsh word and this is not suitable for public especially for children
- Old chatterbots are computationally expensive
- Slow at parsing and generate weak results

2.7 Advantages of MILAH Using ALICE

There is a recurring issue concerning the study of Artificial Intelligence especially everything that applies to a chatterbot, that is what are the limits to which we want to build a bot. But, as we all know, compare to any other chatterbots, Alice bot has a lot of benefits to offer to society.

Below are summaries of what I found out on Milah using the Alice bot advantages. They are all in point form for better understanding.

- Because Alice's creator knew that the bot would be served up on the web, and children would have desire to talk to a robot, Wallace wanted Alice to stay away from certain topics of discussion, for example sex. When you talk about such a topic with Alice she tries to change the subject with gossip and randomness.
- Can be programmed to speak any language and can handle multiple languages simultaneously (and thus could be used as an interpreter or an interactive dictionary).
- Conversation trainer for use in second language acquisition
- The first and only chatterbot that can speak, learn and understand Malay language.

- XML-based. Unlike chatter bots that require you to learn a proprietary language or toolset, Milah is coded in "AIML", an XML-based language that can be edited and maintained using any number of available tools, including several editors developed by the Alice open source community. An XML basis opens up myriad possibilities to easily re-purpose existing content for Alicebots.
- Extensibility. Milah architecture allows for extremely easy extensibility. The openness of the code means that no tweak is too small, or no customization too big, to consider.
- It learns. The Alicebot engine includes the capability to learn from its conversations. Information provided by a user can literally be immediately reincorporated into the conversation, unlike systems that require a botmaster to continuously analyze logs and manually change the bot's knowledge. The degree to which an Alicebot uses learned knowledge is up to the bot's owner.
- Interesting interface and animation. Interface in the main ingredient in a chatterbot to attract people to use them. Milah will have interesting and animated interface to show users her feeling, emotions, etc.
- As an entertainment.

2.8 Summary

This chapter outlined and described in detail the various issues and topics researched throughout the project. The information was derived mainly from books and websites, which were both popular and reliable.

The research was important because it enable better understanding of the different products. This allowed the best combination of platform, software and technologies to be selected to design, develop and implement the Chatterbot project. However, it is to be mentioned that there remain many more products and technologies that mere not recovered in the literature research. The topics covered in this chapter were only some of the more popular products. Nevertheless, the research done was sufficient for the purpose of choosing the appropriate tools to develop and implement this project.

The next chapter, methodology will attempt to explain in detail the method to be used in developing a chatterbot.

CHAPTER 3 : METHODOLOGY

3.1 Introduction

Methodology is the science of how a system is developed. This chapter will describe the methodology used while developing the Milah chatterbot project.

3.2 Methodology Chosen – Waterfall Model with Prototyping

As for the method that I chose to develop my system is the "Waterfall model with prototyping". But before I go on with the waterfall model with prototyping, let's see what goes on in the waterfall model first. One of the first models to be proposed is the "waterfall model", where the stages are depicted as cascading from one to another. As the figure implies, one development stage should be completed before the next begins. Thus, when all of the requirements are elicited from the customer, analyzed for completeness and consistency, and documented in a requirements document, then the development team can go on to system design activities. The waterfall model presents a very high level view of what goes on during development, and it suggests to developers the sequence of events they should expect to encounter.

3.2.1 The Waterfall Model

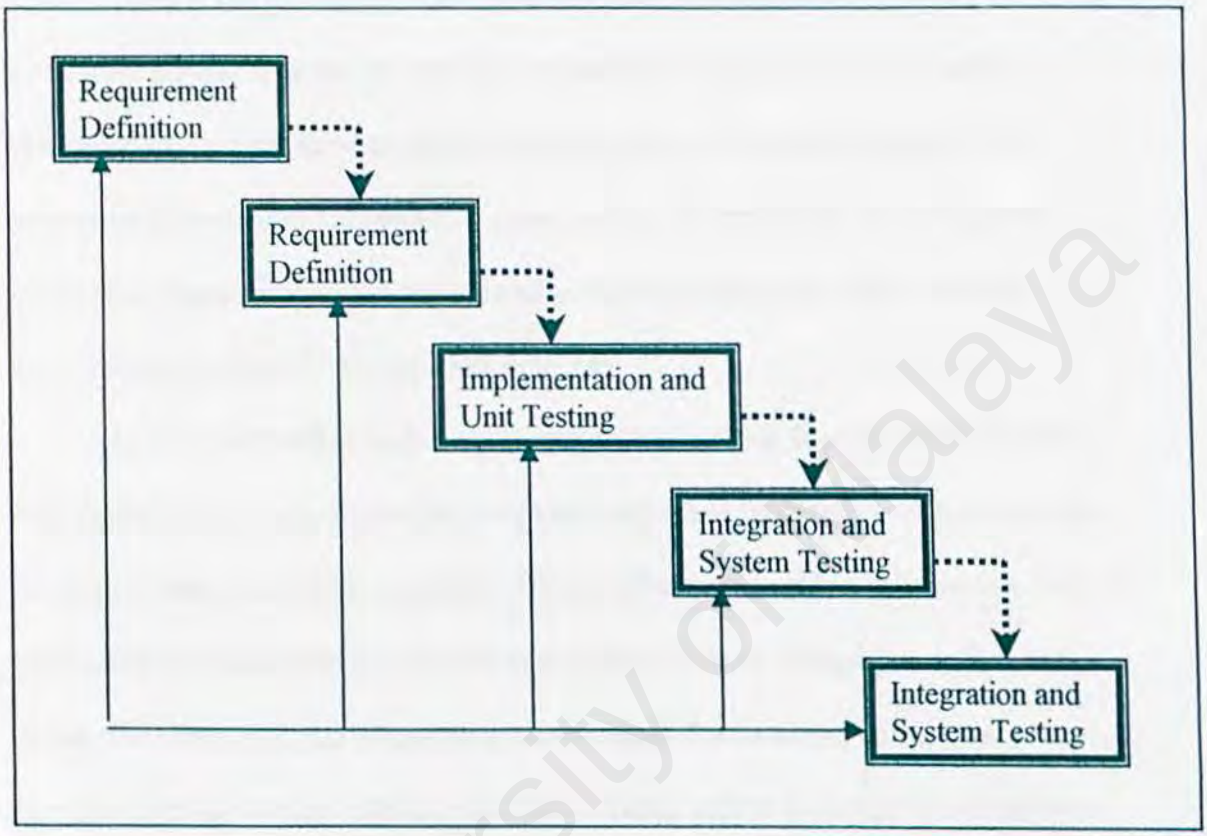


Figure 3.1 :The Waterfall Model

The waterfall model has been used to prescribe software development activities in a variety of contexts. Associated with each process activity were milestones and deliverables, so that project managers could use the model to gauge how close the project was to completion at a given time. For instance, "unit and integration testing" in the waterfall ends with the milestone "code modules written, tested and integrated;" The intermediate deliverable is a copy of the tested code. Next, the code can be turned over to

the system testers so it can be merged with other system components (hardware or software) and tested as a larger whole. The waterfall model can be very useful in helping developers lay out what they need to do. Its simplicity makes it easy to explain to customers who are not familiar with software development; it makes explicit which intermediate products are necessary in order to begin the next stage of development. Many other, more complex models are really just embellishments of the waterfall, incorporating feedback loops and extra activities.

The biggest problem with the waterfall model is that it does not reflect the way code is really developed. Except for very well understood problems, software is usually developed with a great deal of iteration. Often, software is used in a solution to a problem that has never before been solved or whose solution must be upgraded to reflect some change in business climate or operating environment. For example, an airplane manufacturer may require software for a new airframe that will be bigger or faster than existing models, so there are new challenges to address, even though the software developers have a great deal of experience in building aeronautical software. Neither the users nor the developers know all the key factors that affect the desired outcome, and much of the time spent during requirement analysis, may be devoted to understanding the items and the processes affected by the system and its software, as well as the relationship between the system and the environment in which it will operate. Thus, the actual software development process, if uncontrolled, may look like Figure 3.1, developers may thrash from one activity to the next and then back again, as they strive to gather knowledge about the problem and how the proposed solution addresses it.

3.2.2 The Waterfall Model with Prototyping

The software development process can help to control the trashing by including activities and sub processes that enhance understanding. Prototyping is such a sub process; a prototype is a partially developed product that enables customers and developers to examine some aspect of the proposed system and decide if it is a suitable or appropriate for the finished product. For example, developers may build a system to implement a small portion of some key requirements to ensure that the requirements are consistent, feasible and practical; if not, revisions are made at the requirement stage, rather than at the more costly testing stage. Similarly, parts of the design may be prototyped, as shown in Figure 3.2. Design prototyping helps developers assess alternative design strategies and decide which is best for a particular project. The designers may address the requirements with several radically different designs to see which has the best properties. For instance, a network may be built as a ring in one prototype and a star in another, and performance characteristics evaluated to see which structure is better at meeting performance goals or constraints.

Often, the user interface is built and tested as a prototype, so the users understand what the new system will be like, and the designers get a better sense of how the users like to interact with the system. Thus, major kinks in the requirements are addressed and fixed well before the requirements are officially validated during system testing; validation ensures that the system has implemented all of the requirements, so that each system function can be traced back to a particular requirement in the specification.

System testing also verifies the requirements; verifications ensure that each functions works correctly. That is, validation makes sure that the developer is building the right product (according to the specification), and verification checks the quality of the implementation. Prototyping is useful for verification and validation.

- **Requirement Analysis**

When a customer requests that we build a system, the customer has some notion of what the system will do. In the chatterbot system, the customers need to understand the ideas and the concept of the developing project. No matter whether its functionality is old or new, each software-based system has a purpose, usually expressed in what the system can do. A requirement is a feature of the system or a description of something the system is capable of doing in order to fulfill the system's purpose. First, we work with our customers to elicit the requirements, by asking questions, demonstrating similar systems, and developing prototypes of all or part of the purposed system. Next, we capture those requirements in a document or database. The requirements are written first so that we and our customers agree on what the system should do. Then, the requirements are often rewritten, usually in a more mathematical representation, so that the designers can transform the requirements into a good design. A verification step ensures that the requirements are correct, and consistent, and a validation step makes sure that we have described what the customer intends to see in the final product.

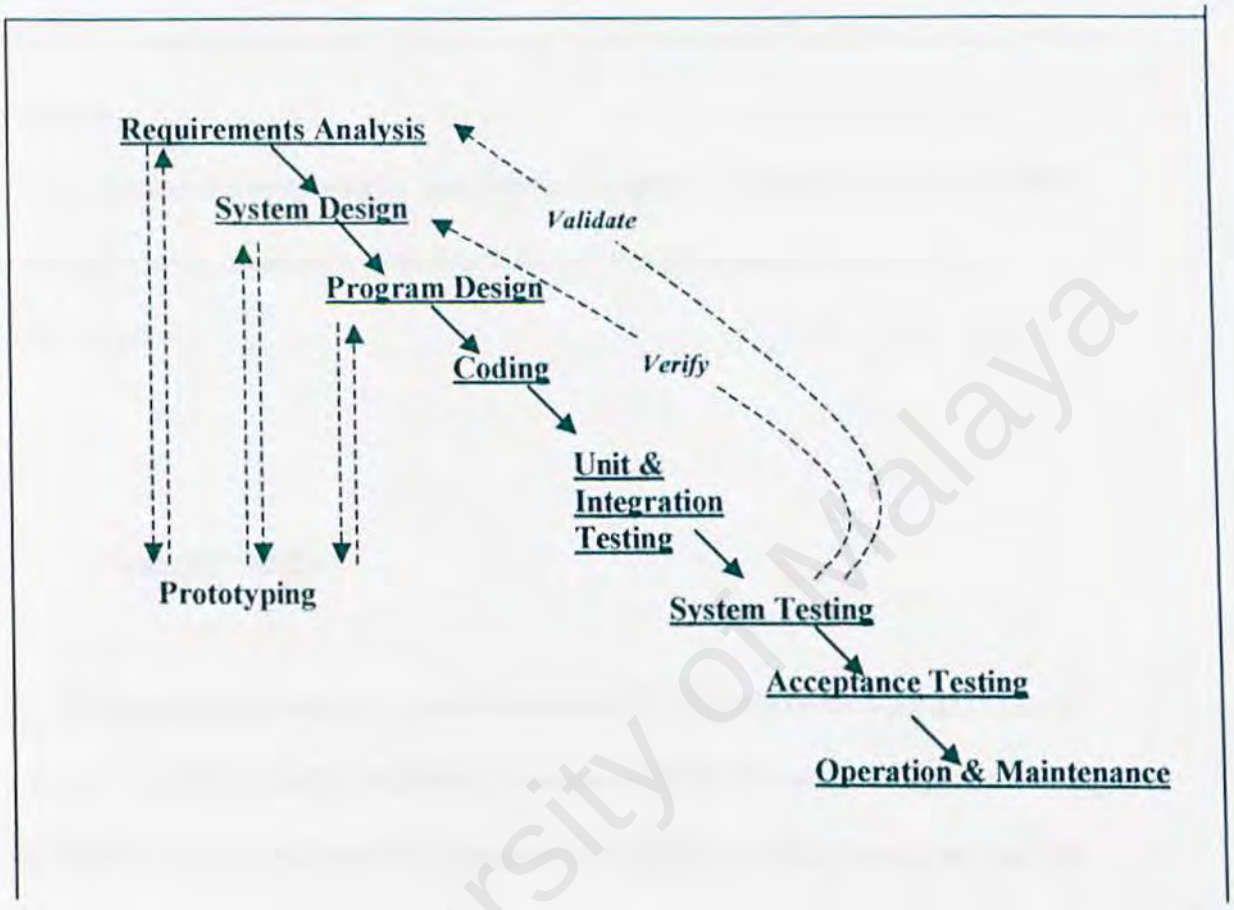


Figure 3.2 : The Waterfall Model with Prototyping.

- **System Design**

Our customers usually want a new system either because there is no existing system or because there are undesirable aspects of the old system. In either case, the requirements documents tell us all about the problem that the system is to solve. Design is the creative process of transforming the problem into a solution; the description of a solution is also called design. As for this "Interactive 3D Human Head Multimedia

System", the problems that rise can be quite a burden considering a whole wide of different people perspectives. System design is the best way to find the solution for these problems.

We use the requirements specification to define the problem. Then, we declare something to be a solution to a problem if it satisfies all the requirements in the specification.

- **Program Design**

During program design, we must also specify the features of each object's interface with the rest of the world. In particular, we need to know the operation signature for each operation. That is, we name each operation, the objects it takes as parameters and the values returned by the operation. In many cases, we can derive this information from the sequence diagram. An object's interface is the collection of all its operation's signatures. Once we have defined the interfaces, we can classify them by type and build a hierarchy of interface types where some interfaces inherit properties from other interfaces. This hierarchy has special importance, since objects are visible and accessible to other objects only through their interfaces.

- **Coding**

Coding involves algorithms and data structures and the components are programming language primitives such as numbers, characters, pointers and control threads. In turn, there are primitive operators, including the language's arithmetic and data manipulation primitives, and composition mechanisms such as arrays, files and procedures. For this project, I use AIML (Artificial Intelligence Markup Language) coding.

- **Unit & Integration Testing**

In developing a system, testing usually involves several stages. First, each program component is tested on its own, isolated the other components in the system. Such testing, known as module testing, component testing or unit testing, verifies that the component functions properly with the types of input expected from studying the component's design. Unit testing is done in a controlled environment whenever possible, so the test team can feed a predetermined set of data to the component being tested and observe what output actions and data are produced. In addition, the test team checks the internal data structures, logic, and boundary conditions for the input and output data. When collections of components have been unit-tested, the next step is ensuring that the interfaces among the components are defined and handled properly. Integration testing is

the process of verifying that the system components work together as described in the system and program specification.

- **System Testing**

Testing the system is very different from unit and integration testing. When you unit test your components, you have complete control over the testing process. You create your own test data, design your own test cases and run the tests yourself. When you integrate components, you sometimes work by yourself but often you collaborate with a small part of the test or development team. However, when you test a system, you work with the entire development team, coordinating what you do and being directed by the test team leader. The objective is to ensure that the system does what the customer wants it to do. To understand how to meet this objective, we first must understand where faults in the system come from.

- **Acceptance Testing**

So far, all the tests have been run by the developers, based on their understanding of the system and its objectives. The customer also test the system, making sure that it meets their understanding of the requirements, which may be different from the developers'. This test, called an acceptance test, assures the customers that the system they requested

is the system that was built for them. The acceptance test is sometimes run in its actual environment but often is run at a test facility different from the target location. For this reason, we may run a final installation test to allow users to exercise system functions and document additional problems that result from being at the actual site.

- **Operation & Maintenance**

When we develop systems, our main focus is on producing code that implements the requirements and works correctly. At each stage of development, I continually refer to work produced earlier stages. The design components are tied to the requirements specification, the code components are cross-referenced and reviewed for compliance with the design, and the tests are based on finding out whether functions and constraints are working according to the requirements and design. Thus, development involves looking back in a careful, controlled way.

Maintenance is different. I look back at development products, but also at the present by establishing a working relationship with users and operators to find out how satisfied they are with the way the system works. I'm also looking forward, too, to anticipate things that might go wrong, to consider functional changes required by a changing business need, and to consider system changes required by changing hardware, software or interfaces.

3.2.2.1 Uses of Prototyping

1. Verifying user needs.
2. Verifying that design = specifications.
3. Selecting the "best" design.
4. Developing a conceptual understanding of novel situations.
5. Testing a design under varying environments.
6. Demonstrating a new product to upper management.
7. Implementing a new system in the user environment.

3.3 Summary

This chapter explains the methodology used in developing my Milah chatterbot project along with the explanation about why I've chosen the method. The description about each phases are also included.

CHAPTER 4 : SYSTEM ANALYSIS

4.1 Introduction

A complete understanding of software requirement is essential to the success of a software development effort. The requirement analysis task is a process of discovery, refinement, modeling and specification. With requirement analysis, system engineer is able to specify software function and performance, indicates software's interface with other system elements and establish constraints that software must meet.

4.2 Analysis Procedures

The process of analysis involves the following procedures:

- Problem identification
- Evaluation and synthesis
- Modeling
- Specification

4.2.1 Problem Identification

Before a new system can be built, we must identify the problem that needs to be solved in order to ensure the success of this project. For a chatterbot, the problem is how to store, retrieve and manage a large amount of AIML tags in a knowledge base in an effective and efficient way. In additional, another problem would be to determine how to maintain and update the knowledge base and improve the performance of Milah chatterbot in replying quick response to user.

4.2.2 Evaluation and Synthesis

In this stage, analyses of the problems need to be done by dividing the problems into smaller parts so that the problem will be easier to be understood and solved.

The following problems are the few examples of system requirements that must be considered:

- What kind of knowledge base used to store the images and the sentiments?
- How are the AIML tags and all information about Milah saved in a knowledge base or brain?
- How to write to write a good and relevant AIML tags?

- What are the most appropriate tools and technology to be used in building Milah chatterbot?

4.2.3 Modeling

We create models to gain a better understanding of the actual entity to be built. The model focuses on what the system must do; usually a graphical notation (such as DFD) is used to depict information, processing, system behavior and other characteristic.

4.2.4 Specification

The requirement specification is a complete listing that defines what the system should do, it will be used in the system design and system testing.

4.3 Requirement Analysis

Requirement analysis covers the area of functional and non-functional requirements of the Milah chatterbot project. The functional requirement can be categorized to the general user section and the administrator or botmaster section, where both sections will try to give out a clear picture on how the user interface is going to be.

Whereas, the non-functional requirement will discuss the system's constraints along with the standards the system must meet.

4.3.1 Functional Requirements

Functional requirement is a functions or characteristics expected by user for the system. It is a combination of all the main modules of Milah chatterbot, which are made to communicate internally or externally. The system user are divided into two categories which are the user who would like to chat or talk to the chatterbot and the system administrator or better known as botmaster. The functional requirements for the system are divided into a few modules as follow:

4.3.1.1 User

- User will talk with the chatterbot
- User will teach chatterbot
- Chatterbot will try to answer all users inquiries

4.3.1.2 Botmaster

Botmaster is the master of the chatterbot, in this project the botmaster is me. A botmaster runs program and creates or modifies a chat robot with the program's graphical user interface (GUI). The botmaster is responsible for reading dialogues, analyzing

the responses and creating new replies for the patterns detected by the program .

4.3.2 Non - Functional Requirements

Non-functional requirements are the constraints under which a system must operate and the standards that must be met by the delivered system. This is a non-functional requirement or constraints describe a restriction on the system that limits the choice for constructing a solution to the problem. These constraints usually narrow my selection of language, platform, or implementation techniques or tools.

- **Reliability**

This chatterbot should be reliable which means that it does not produce dangerous or costly failures when it is need in a reasonable manner.

- **Efficiency**

Efficiency in a computer terminology means a procedure that can be called or accessed in an unlimited numbers of times to produce similar outcomes or output at a creditable speed.

- **Maintainability**

A product is maintainability if the programs are easily to modify and test when updating to meet new requirement, correcting errors or more to a different computer system.

- **Simplicity**

Simplicity refers to keep forms and screens properly uncluttered in a manner that focuses the user attention.

- **Understandability**

Understandability in terms of the coding method used, allows other programmers to understand the logic of the program flows. Thus, changes can be made easily upon the necessary program segments without modifying other essential logic of the program. Simple and clear sentences or instruction are displayed so that users can use the system without difficulty.

- **User-friendly**

The user interface for this chatterbot should not be too crowded so that it would not confuse the user.

- **Interesting Interface**

The interface for this chatterbot must be very interesting, cheerful and interactive so that the user will like using it and won't get bored

4.4 Problem Analysis

4.4.1 Information Searching Method

Information searching refers to the method of collecting as many information as possible about the system. It is one of the techniques required to improve one's understanding about the system and to satisfy the need of future researched system. It also required to accommodate the work base for the system design. Information searching method includes:

- **Books and references**

  Books and references are used to collect the information needed about developing a chatterbot. The information are collected from sources like information system, development tools, programming and database references, which can be found in Main Library University of Malaya and also National Library. Some of them are from my own collections.

- **Internet surfing**

  Internet is like a largest warehouse of information in the world. I use the internet to search information about other Web site that serve the same functions, the tools that I want to use, project methodology and other related information.

- **Document's room**

The document's room, which is situated in the old building of FSKTM, is a room that placed thesis reports of senior students. I used the senior's report as a reference to do my final project. There's lot of useful information that I've got from the reference such as project plan and main topics.

4.5 Development Tools Analysis

This section will identify the suitable programming languages and development tools, that are used to develop Milah chatterbot. An analysis has been done in making the decision and after much consideration, I have chosen this set of tools and technologies for the realization of this project. The ideal solutions for this project are easy to develop and and deploy, and also easy integration with the latest emerging technologies.

4.5.1 ALICE Program D

Program B Java edition was based on pre-Java 2 technology. Although the program ran on many different platforms, it did not take advantage of newer Java features such as Swing and Collections. Jon Baer recoded Program B with Java 2 technology, and added many new features. This giant leap in the interface and the core, plus the fact that Jon named his bot "DANY", justified granting the next code letter "D" to his latest Alicebot Java edition. Beginning in November 2000, Program D is the only the Java edition of the Alicebot engine actively supported.

I have chosen to develop my chatterbot, Milah using Alice Program D because this is the currently supported, actively developed Java implementation of the Alice bot engine. This version also supports Flash 5 which will enable me to create Flash-based interface.

This is the version that I wanted to use since it uses latest technology, and this is also the best program to use compared to other bot development program available. Furthermore, source and information about Alibe bot development program is easier to be found on the internet and it's reliable compared to any other bots development program. Milah Chatterbot will based on Alice bot technology.

4.5.2 JAVA Technology

The Java™ platform is based on the power of networks and the idea that the same software should run on many different kinds of computers, consumer gadgets, and other devices. Since its initial commercial release in 1995, Java technology has grown in popularity and usage because of its true portability. The Java platform allows you to run the same Java application on lots of different kinds of computers.

Any Java application can easily be delivered over the Internet, or any network, without operating system or hardware platform compatibility issues. For example, you could run a Java technology based application on a PC, a Macintosh computer, a network computer, or even new technologies like Internet screen phones. Furthermore, the Java

platform was designed to run programs securely on networks, which means that it integrates safely with the existing systems on your network.

The idea is simple: Java technology-based software can work just about everywhere. Java technology components don't care what kind of computer, phone, TV, or operating system they run on. They just work, on any kind of compatible device that supports the Java platform.

Java technology allows programmers and users to do new things with Web pages that were not possible before. With Java technology, the Internet and private networks become your computing environment. For example, users can securely access their personal information and applications when they're far away from the office by using any computer that's connected to the Internet; soon they'll be able to access tailored applications from a mobile phone based on the Java platform, or even use smart cards as a pass key to everything from the cash machine to ski lifts.

The Java platform is being built into next-generation telephones, TV set-top boxes, smart cards that fit in your wallet, and many other consumer and business devices. Java technology-based software includes: programs written in the Java programming language can run directly on your computer (without requiring a browser), or on servers, on large mainframe computers, or other devices.

For example, Java technology-based software running on servers in large companies monitors transactions and ties together data from existing computer systems.

Other companies are using Java technology-based software on their internal Web sites to streamline communication and the flow of information between departments, suppliers and customers.

Programs written in the Java programming language run on so many different kinds of systems thanks to a component of the platform called the Java virtual machine or "JVM™"* -- a kind of translator that turns general Java platform instructions into tailored commands that make the devices do their work.

Developing on the Java platform means that projects are completed faster and with less debugging.

Program D is a Java application program, so it is needed to have Java Development Kit in the computer in order to run Alice Program D or even to talk to the chatterbot that uses Alice technology. User and the botmaster can download the Java Development Kit - jdk1.2.2 - at java.sun.com.

4.5.3 XML

XML, stands for Extended Markup Language is a markup language for documents containing structured information. It is a set of rules for designing text formats that let user structures their data. XML is not a programming language, and anybody don't have to be a programmer to use it or learn it. XML makes it easy for a computer to generate data, read data, and ensure that the data structure is unambiguous. XML avoids

common pitfalls in language design. It is extensible, platform-independent, and it supports internationalization and localization. XML is fully Unicode-compliant.

XML was created so that richly structured documents could be used over the web. The only viable alternatives, HTML and SGML, are not practical for this purpose. HTML comes bound with a set of semantics and does not provide arbitrary structure. SGML provides arbitrary structure, but is too difficult to implement just for a web browser. While XML is being designed to deliver structured content over the web, some of the very features it lacks to make this practical, make SGML a more satisfactory solution for the creation and long-time storage of complex documents.

4.5.4 AIML

AIML (Artificial Intelligence Markup Language) is an XML specification for programming chat robots like Alice using program D. The emphasis in the language design is minimalism. The simplicity of AIML makes it easy for non-programmers, especially those who already know HTML, to get started writing chat robots.

One ambitious goal for AIML is that, if a number of people create their own robots, each with a unique area of expertise, program D can literally merge-sort them together into a Superbot, automatically omitting duplicate categories. We offer the both the source code and the ALICE content, in order to encourage others will "open source" their chat robots as well, to contribute to the Superbot.

Artificial Intelligence Markup Language, abbreviated AIML, describes a class of data objects called AIML objects and partially describes the behavior of computer programs that process them. AIML is a derivative of XML, the Extensible Markup Language. By construction, AIML objects are conforming XML documents, although AIML objects may also be contained within XML documents. As XML is itself an application profile or restricted form of SGML, the Standard Generalized Markup Language AIML objects are also conforming SGML documents.

AIML objects are made up of units called topics and categories, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form AIML elements. AIML elements encapsulate the stimulus-response knowledge contained in the document. Character data within these elements is sometimes parsed by an AIML interpreter, and sometimes left unparsed for later processing by a Responder.

A software module called an AIML interpreter is used to read AIML objects and provide application-level functionality based on their structure. An AIML interpreter may use the services of an XML processor, or it may take the place of one, but it must not violate any of the constraints defined for XML processors. It is assumed that an AIML interpreter is part of a larger application generically termed a bot, which carries the larger functional set of interaction based on AIML. This does not constrain the specific behavior of a bot. A software module called a responder handles the human-to-bot or bot-to-bot interface work between an AIML interpreter and its object.

The origins and goals for AIML. AIML was developed by Dr. Richard S. Wallace and the Alicebot free software community during 1995-2000. It was originally adapted from a non-XML grammar also called AIML, and formed the basis for the first Alicebot, A.L.I.C.E., the Artificial Linguistic Internet Computer Entity. Since its inception, it has been adopted as a standard by the A.L.I.C.E. AI Foundation, which now holds its copyright, and whose Alicebot and AIML Architecture Committee is responsible for its maintenance and further elaboration.

The design goals for AIML are:

1. AIML shall be easy for people to learn.
2. AIML shall encode the minimal concept set necessary to enable a stimulus-response knowledge system modeled on that of the original A.L.I.C.E.
3. AIML shall be compatible with XML.
4. It shall be easy to write programs that process AIML documents.
5. AIML objects should be human-legible and reasonably clear.
6. The design of AIML shall be formal and concise.
7. AIML shall not incorporate dependencies upon any other language.

AIML objects. Each AIML object has both a logical and a physical structure. Physically, the object is composed of units called topics and categories. An object begins in a "root" or object entity. Logically, the object is composed of elements and character references, all of which are indicated in the object by explicit markup.

An AIML object may also be "overlaid" by comments and processing instructions as described by the XML specification, as well as by XML content from other namespaces. Comments and processing instructions are not treated by an AIML interpreter. Foreign namespace content may be passed by an AIML interpreter to a responder, but is not processed by the AIML interpreter itself.

In the following, we reiterate the AIML definitions of various aspects of object structure, although in the great majority of cases these simply repeat the XML definitions of the same. In several cases, there are further constraints on AIML interpreters that do not apply to all XML processors.

I chose AIML for my Milah chatterbot development because I have chosen Alice Program D to develop as the bot's brain. Further descriptions and how AIML works will be discuss more in the system design chapter.

4.5.5 Macromedia Flash 5

Macromedia Flash 5.0 is the fastest way to create rich Internet content and applications with a better return on investment. Its powerful video, multimedia and application development features allow the creation of rich user interfaces, online advertising, e-Learning courses and enterprise application front-ends.

Furthermore, Flash 5.0 also expands the already impressive possibilities of what anybody can do with streaming Web media. Coders will appreciate the new rudimentary HTML rendering, built-in XML parser, and advanced scripting engine. Animators and graphic designers will be pleased by the inclusion of new tools, beefier import options, and integration with other Macromedia products.

All in all, Flash 5.0 is an exciting product. The new features make it a must-have for any developer who wants to create rich but lightweight Web media.

My interest in Multimedia has introduce me to this very useful tools. My project, building a chatterbot is a chat robot which demands a very creative, nice-looking interactive interface in order to attract user to chat with my bot. Therefore, I've decided on this tool plus with my considerable amount of knowledge in both of the tools, I'm sure they will help me to design a great user interface. After all, it is easier to design using familiar tools.

4.5.6 Adobe Photoshop 6

Adobe Photoshop is an image-editing standard software. It provides a comprehensive toolset, unmatched precision, and powerful creative options to help user create professional-quality images for Web, print, and emerging media. Other than that, it is also developed to meet any creative or production demand and to handle the widest variety of image-editing tasks in the most efficient way.

With its comprehensive set of retouching, painting, drawing, and Web tools, Photoshop helps user complete any image-editing task efficiently that they can experiment freely without sacrificing efficiency. Photoshop also gives user the tools they need to keep the work on track and bring it in on deadline.

Adobe Photoshop delivers high-powered image editing, photo retouching, and compositing tools to help user get professional-quality results. It offers a lot of tools with their specific task to help user alter their images the way they want it such as edge smoothing, sharpening controls, healing brush, color correction etc.

The powerful Photoshop paint engine lets user simulate traditional painting techniques, including charcoal, pastel, and wet or dry brush effects. They can choose from the many preset brush styles or use the Brushes palette to create their own unique effects. With its drawing tool, user can draw resolution-independent vector shapes instantly with the line, rectangle, ellipse, polygon, and custom shape tools.

Adobe Photoshop Web tools, lets user produce exceptional imagery for the Web and wireless devices along with the helps from ImageReady, which ships with Photoshop. There are slicing tool, optimization tools, rollovers palette, animation palette and other Web tools in Photoshop to help user create their style of user interface.

This is another useful tool that I have chosen to help me do colorful and interesting interface. My project, building a chatterbot is a chat robot which demands a very creative, nice-looking interactive interface in order to attract user to chat with my bot.

4.6 Summary

This chapter on System Analysis describes the functional and non-functional requirements of Milah chatterbot project. Summary of the software and technology used to build the system and the reasons for using those products were explained.

CHAPTER 5 : SYSTEM DESIGN

5.1 Introduction

Design is the creative process of transforming the problem into a solution. And this definition can be related to system design, which is a process through which requirements are translated into a representation of software. Modularity is a characteristic of a good design. The components have clearly defined inputs and outputs. And each component has a clearly stated purpose. Thus, to design a system is to determine a set of components and intercomponent interfaces that satisfy a specified set of requirement.

The design of Milah chatterbot has considered the following issues :

- Architecture design
- Brain design
- Interface design

5.2 Architecture Design

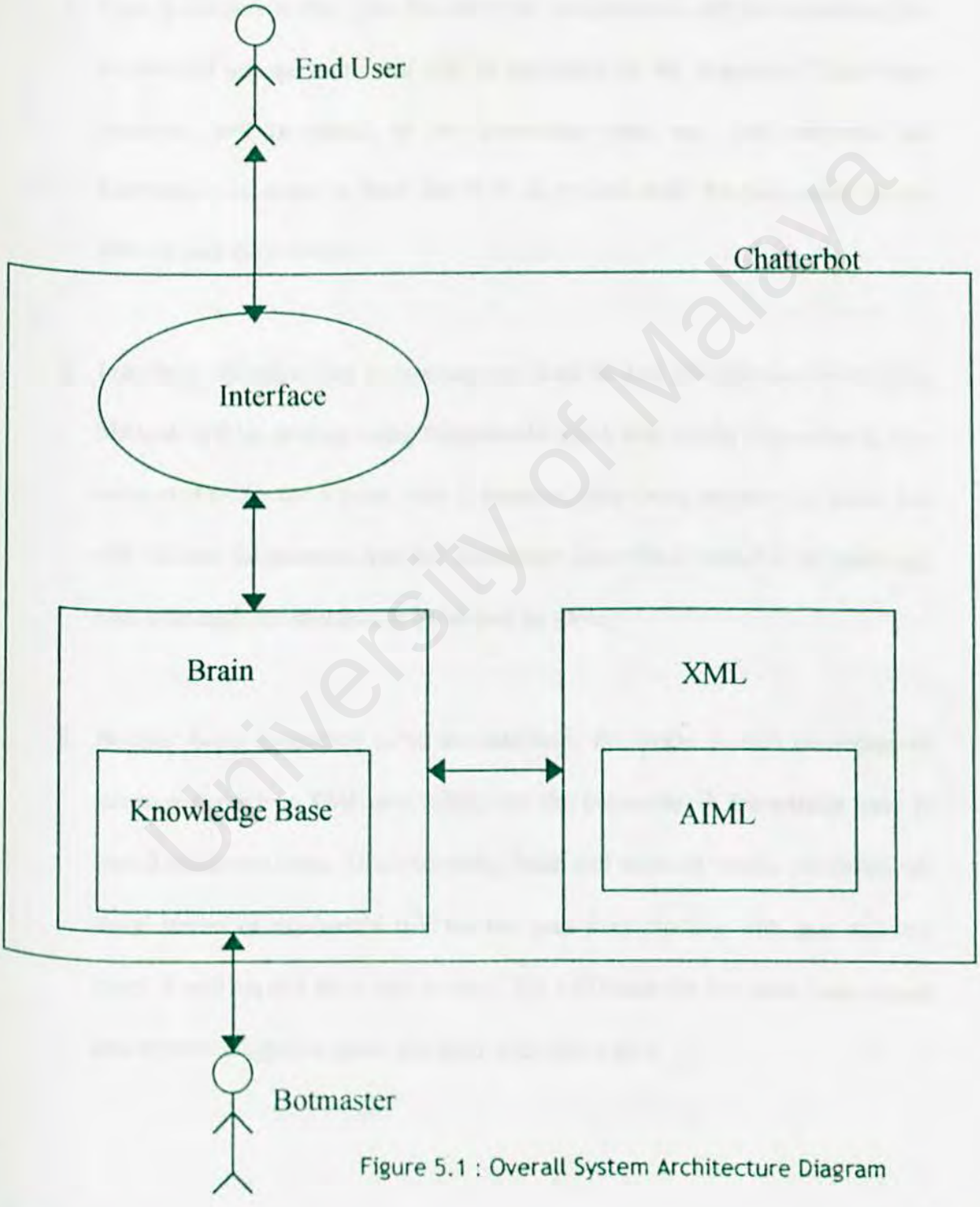


Figure 5.1 : Overall System Architecture Diagram

Explanation:

1. **User** is the person who uses the system to communicate with the chatterbot. He or she will ask questions that will be answered by the chatterbot. Thus these questions will be stored in the knowledge base and shall improve the knowledge. In order to have the Q & A session with the bot, users has to interact with the interface.

2. **Interface** will allow user to communicate with the bot. The interface which is in 3D look will be develop using Macromedia Flash and Adobe Photoshop is also connected to the bot's brain. This is because, after being enquired by users, bot will elicitate the answers from the knowledge base that is stored in the barin and pass it through the interface to be viewed by users.

3. Besides being connected to to the interface, the **brain** is also connected to languages which is XML and AIML and the botmaster. A **knowledge base** is stored inside the brain. This knowledge base will store all words, grammars, all those stories or experience that the bot gain from chatting with user and lots more. It will expand from time to time, this will make the bot more experienced and more intelligent as more and more user chat with it.

4. **XML** is a markup language similar to HTML .Similar, in that it contains tags and attributes as HTML does, and in a quick glance at XML and HTML placed side by side will reveal quite similar structures. XML produce a simple, yet powerful markup language. An XML document contains both data and information about the data. XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraint on the storage layout and logical structure.

5. **AIML** is a derivative of XML. Its goal is to enable pattern-based, stimulus-response knowledge content to be served, received and processed on the web and offline in the manner that is presently possible with HTML and XML. AIML has been designed for ease of implementation, ease of use for new developers and for interoperability with XML and XML derivatives such as XHTML. It consists of AIML object, AIML object structure, pattern, templates, AIML pattern matching and AIML predicate handling.

6. **Botmaster** is the person who stores knowledge in to the knowledge base and will keep updating it time to time and is also responsible to watch the log.

5.3 Brain / Knowledge Base Design

5.3.1 MILAH Work Flow Process

Generally, the workflow process of Milah is based on three entities that are user, responder, classifier and graphmaster. To get a better view, please refer to chart 5.2 below.

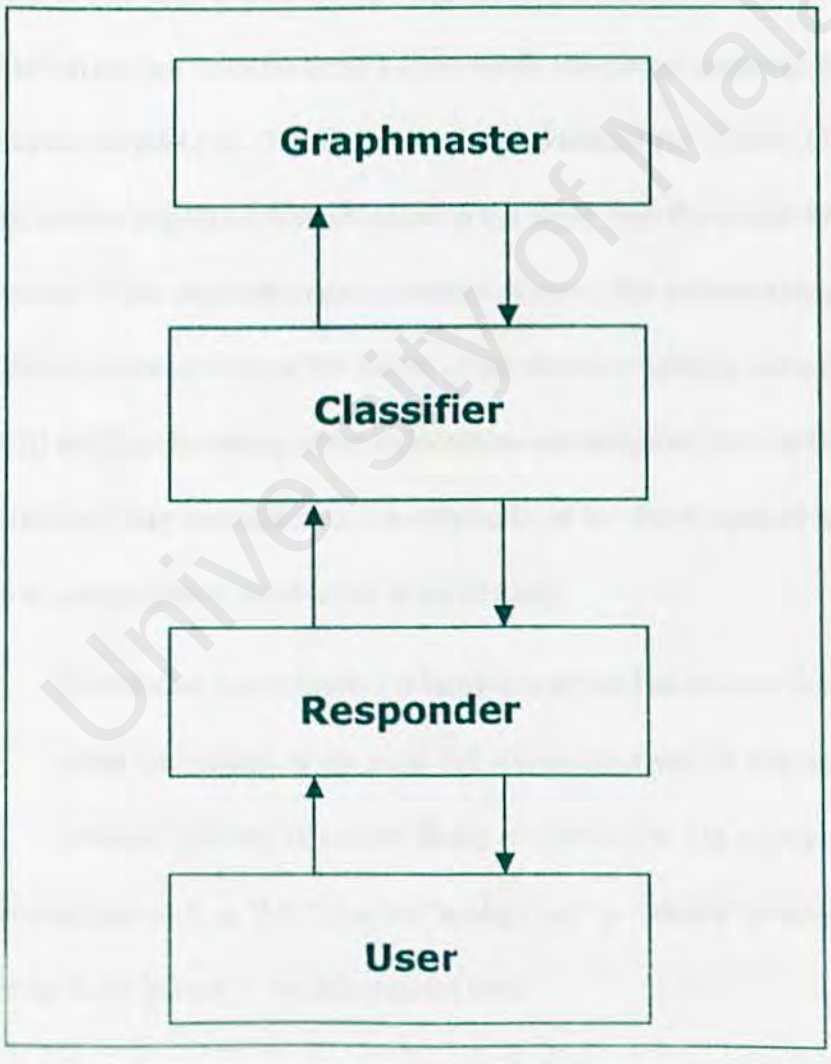


Figure 5.2 : Work Flow Process Diagram

Explanation :

1. The **Responder** acts as the interface amidst user and system. It transfers user input to **Classifier** and delivers the bot's respond to user.

2. Inside **Classifier**, normalisation process is executed. The minimum set of normalisation is called pattern-fitting normalisation. Additional normalisation performed at user option is called sentence-splitting normalisation and substitution normalisation (or just "substitutions"). If an AIML interpreter performs substitution normalisation on the input, then these must be performed first. If an AIML interpreter performs sentence-splitting normalisation on the input, then these must be performed on the output of the substitution normalisation process. The pattern-fitting normalisation process receives the output of the sentence-splitting normalisation process (if any), or the output of the substitution normalisation process (if any, and if no sentence-splitting normalisation is performed), or the direct input (if no sentence-splitting or substitution normalisation is performed).

   Substitution normalisation is heuristics applied to an input that attempt to retain information in the input that would otherwise be lost during the sentence-splitting or pattern-fitting normalisation. For example:

- Abbreviations such as "Mr." may be "spelled out" as "Mister" to avoid sentence-splitting at the period in the abbreviated form

- Web addresses such as "http://alicebot.org" may be "sounded out" as "http ALICEbot dot org" to assist the AIML author in writing patterns that match Web addresses
- Filename extensions may be separated from their file names to avoid sentence-splitting (".zip" to " zip")

  Sentence-splitting normalisation is heuristics applied to an input that attempt to break it into "sentences". The notion of "sentence", however, is ill defined for many languages, so the heuristics for division into sentences are left up to the developer.

Commonly, sentence-splitting heuristics use simple rules like "break sentences at periods", which in turn rely upon substitutions performed in the substitution normalization phase, such as those which substitute full words for their abbreviations.

Pattern-fitting normalization are normalization that remove from the input characters that are not normal character. Pattern-fitting normalization on an input must remove all characters that are not normal characters. For each non-normal character in the input,

- if it is a lowercase letter, replace it with its uppercase equivalent
- if it is not a lowercase letter, replace it with a space

Classifier then will transfer the normalized strings to **Graphmaster** and processes the output from the graphmaster while handling various AIML instructions. This entity also will deliver the bot's responses to user.

3. In the other hand, Graphmaster organizes storage of brain contain which is stored as a graph. Graphmaster also handles the pattern matching process. Based on chart 5.3,matching behaviour can be described in terms of the class Graphmaster, which is a common implementation of the AIML pattern expression matching behaviour:

1. Given:
    a. an input starting with word X, and
    b. a Nodemapper of the graph:
2. Does the Nodemapper contain the key _? If so, search the sub graph rooted at the child node linked by _. Try all remaining suffixes of the input following X to see if one matches. If no match was found, try:
3. Does the Nodemapper contain the key X? If so, search the sub graph rooted at the child node linked by X, using the tail of the input (the suffix of the input with X removed). If no match was found, try:
4. Does the Nodemapper contain the key *? If so, search the sub graph rooted at the child node linked by *. Try all remaining suffixes of the input following X to see if one matches. If no match was found, go back up the graph to the parent of this node, and put X back on the head of the input.
5. If the input is null (no more words) and the Nodemapper contains the <template> key, then a match was found. Halt the search and return the matching node.

If the root Nodemapper contains a key "*" and it points to a leaf node, then the algorithm is guaranteed to find a match.

The patterns need not be ordered alphabetically or according to any other complete

system, only partially ordered so that _ comes before any word and * after any word. The matching is word-by-word, not category-by-category. The algorithm combines the input pattern, the <that> pattern, and the <topic> pattern into a single "path" or sentence such as: "PATTERN <that> THAT <topic> TOPIC" and treats the tokens <that> and <topic> like ordinary words. The PATTERN, THAT and TOPIC patterns may contain multiple wildcards. The matching algorithm is a highly restricted version of depth-first search, also known as backtracking.

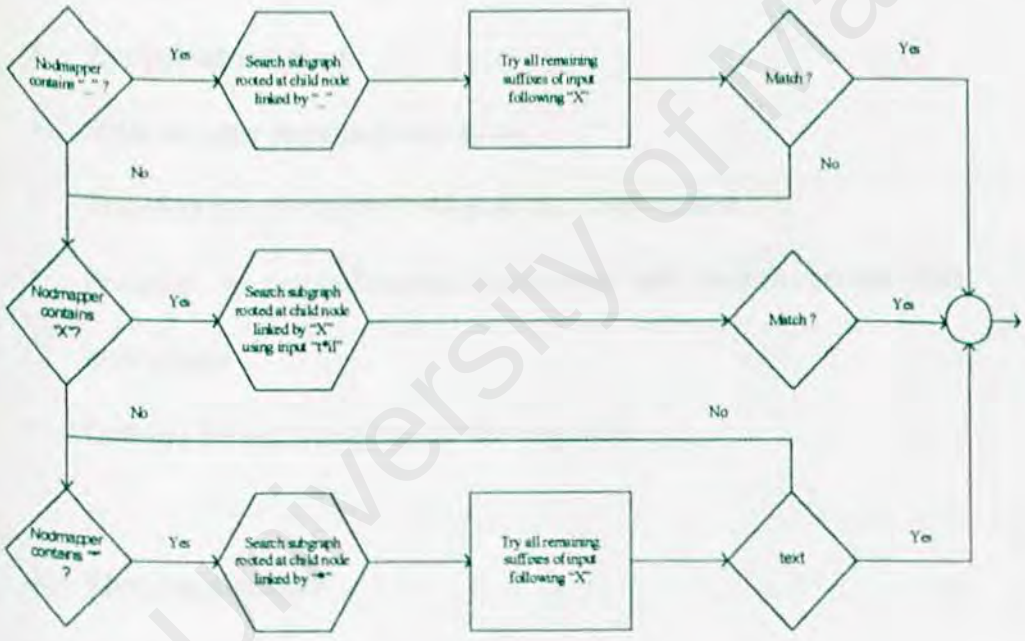


Figure 5.3 : Pattern Matching Algorithm Diagram

I have summarized all that have been explained above into point form, on the next page for easier understanding :

1. **The Responder**

- Interface between user and core routines
- Handles means of in- and output
- Transfers user input to the Classifier and delivers the bot's response to the user

2. **The Classifier**

- Normalizes and filters the input
- Applies substitutions
- Splits the user input into sentences
- Transfers the normalized strings to the Graphmaster
- Processes the output from the Graphmaster and handles various AIML instructions
- Delivers the bot's response to the responder

3. **The Graphmaster**

- Organizes storage of brain content
- Content is stored as a graph (hence the name)
- Handles the pattern matching process
- Pattern matching involves an advanced search-tree algorithm
- Returns raw response template to the classifier

5.3.2 Theory and Learning Model in MILAH

Since developing Milah are based on Alice bot technology, the theory and the model of learning in Milah is also based on Alice. The learning model called "supervised training", because a teacher, the botmaster, always plays a crucial role. The alternative, "unsupervised training", is complicated in an open environment like the Web. The problem is that clients are untrustworthy teachers, and forever try to "fool" the robot with untrue assertions.

I used to say that there was no theory behind Alice: no neural network, no knowledge representation, no search, no fuzzy logic, no genetic algorithms, and no major parsing. Then I discovered there was a theory circulating in applied AI called "Case-Based Reasoning" or CBR that maps well onto the Alice algorithm. Another term, borrowed from pattern recognition, is "nearest-neighbor classification."

The CBR "cases" are the categories in AIML. The algorithm finds best-matching pattern for each input. The category ties the response template directly to the stimulus pattern. ALICE is conceptually not much more complicated that Weizenbaum's ELIZA chat robot; the main differences are the much larger case base and the tools for creating new content by dialog analysis.

Milah is also part of the Alice tradition of "minimalist", "reactive" or "stimulus-response" robotics. Mobile robots work best, fastest and demonstrate the most animated,

realistic behavior when their sensory inputs directly control the motor reactions. Higher-level symbolic processing, search, and planning, tends to slow down the process too much for realistic applications, even with the fastest control computers.

Can probability (statistics, weights, neural networks, or fuzzy logic) improve bots? Statistics are in fact heavily used in the Alice server, but not in the way you might think. Alice uses 'Zipf Analysis' to plot the rank-frequency of the activated categories and to reveal inputs from the log file that don't already have specific replies, so the botmaster can focus on answering questions people actually ask (the "Quick Targets" function).

Other bot languages, notably the one used for JULIA, make heavy use of "fuzzy" or "weighted" rules. We see their problem as this: the botmaster already has enough to worry about without having to make up "magic numbers" for every rule. Once you get up 10,000 categories (like Milah) you don't want to think about more parameters than necessary. Bot languages with fuzzy matching rules tend to have scaling problems. Finally, the bot replies are not as deterministic as you might think, even without weights. Some answers rely on <random> to select one of several possible replies. Other replies generated by unforeseen user input also create "spontaneous" outputs that the botmaster doesn't anticipate.

5.3.3 AIML

AIML stands for Artificial Intelligence Markup Language.It is much like HTML, in that it uses tags to define characteristics like how the bot recognizes and responds to patterns. Because of it's simplicity, and similarity to html, it offers an opportunity for novice users to get to the "guts" of Alice so it can be modified, with typical text editors (without compile). AIML and HTML are both subsets of XML..AIML permits for a simple standard to proliferate in A.I. research.

Example AIML tags in context :

<milah>

<category>

<pattern> WHAT KIND OF HISTORY * **</pattern>**

<template> I like to talk about the history of robots and computers. **</template>**

</category>

</milah>

- <**category**> stores a case, or piece of knowledge
- <**pattern**> what must be recognized to process output
- <**template**> the text to be sent to client
- <**milah** > definition of an MILAH bot

The role of recursion. "Recursion" means applying the same solution over and over again, to smaller and smaller problems, until you reduce the problem to its simplest form. AIML has simple tags that permit for recursion. Recursion can apply many times to a single input.

Example of recursion . Given the normalized input:

ALICE CAN YOU PLEASE TELL ME WHAT LINUX IS RIGHT NOW

An AIML category with the pattern "* RIGHT NOW" matches first, reducing the input to:

ALICE CAN YOU PLEASE TELL ME WHAT LINUX IS

Another pattern ("<name/> *") reduces it to:

CAN YOU PLEASE TELL ME WHAT LINUX IS

And then:

PLEASE TELL ME WHAT LINUX IS

reduces to:

TELL ME WHAT LINUX IS

and finally to

WHAT IS LINUX

5.4 Interface Design

User interface plays a very important role in determining the quality of a chatterbot. User interface is the component of the system that communicates with the users. Therefore, the input data collected from the users and output data generates for the users depend on a well designed user interface. The interface design should meet the objectiveness, accuracy, ease of use, consistency, simplicity and attractiveness. All of these objectives are attainable though the use of basic design principle, knowledge of what is needed as input for the system, and an understanding of how the user should respond to different elements in the forms and screens. There are three main categories of guidelines that are taken into consideration and they are the general interaction, information display and data input.

5.4.1 General Interaction

Guidelines for general interaction often cross the boundary into information display, data entry and overall system control. Below are the guidelines for general interaction.

Guideline	Description
Consistent	Consistent formats for command input, data display, menu selection, and error message and placing of the control objects that are displayed to users. An inconsistent user interface will only confuse the users.

Handling mistakes	The system should be able to handle certain mistakes especially the input from users. The system should verify and do validation on them to protect itself from error that might cause it to fail.
Request for verification on certain task	This important if a task is performed on certain functions that may be critical. For example, when the administrator delete/update any information, a confirm asking for verification is appropriate.
Reduce memorization	Interface should not need users to remember much of the information. For example, having proper and useful messages can remind the user of their current status.
Help facilities	Help in any application is one of the most important modules that assist or serve as a guideline to users using the system.

Table 5.1 : Guidelines For General Interaction

5.4.2 Information Display

Information display is an important issue. The misplacing of information may confuse the users and lead to misconception on the results. The following guidelines focus on information display:

- Display information that is relevant to current context.
- Use a presentation format that is easy to understand.
- Use consistent labels, standard abbreviations and predictable color.
- Provide meaningful error message.
- Comments that are not needed in certain context should be deactivated (or made visible) to avoid confusion.

5.4.3 Data Input

The user spends much of the time doing data input especially in system like a chatterbot. The following guidelines focus on data input:

- Minimize the number of input actions required of the user. The main objective of this is to reduce keyboard typing by users and try as much as possible to use the mouse to select predefined sets of input (or results).
- Maintain consistency between information display and data input.
- Allow the user to control the interactive flow.

5.4.4 User Interface Design of Chatterbot

The following show the issues that taken into consideration during the user interface design of Milah chatterbot:

- A consistent format for menu display and data display. User can select same menu from any page.
- An easy to use windows display so that user can chat or type what they want to the chatterbot conveniently.
- When a user input an invalid data, an error message will quickly be pop up.
- The instructions of how to use or how to talk to the chatterbot must be clear enough so that user will not be having any problem during the chatting process.

The following are the interface designs for my Milah chatterbot :

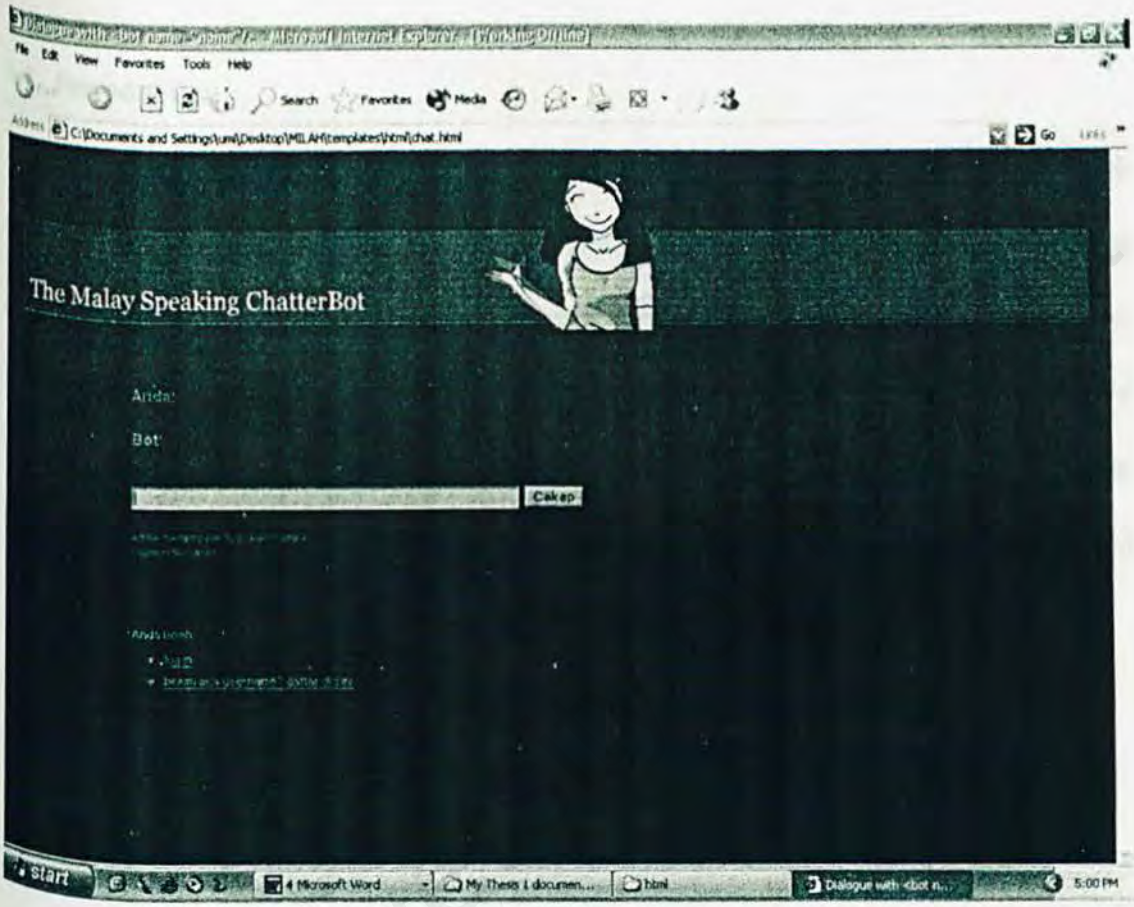


Figure 5.4 : Interface design for chatting windows

5.5 Summary

This chapter explains all the proposed processes and system design that are needed to develop Milah chatterbot project. Along with knowledge base and interface design, this chapter intended to elucidate more on the understanding about the system that are going to be implement.

CHAPTER 6 : SYSTEM IMPLEMENTATION

6.1 Introduction

The process of assuring that the information system is operational and then allowing users to take over its operation is called system implementation. System implementation is further defined as the construction of a new system and the delivery of that system into production in a day-today operation. It involves coding step that translates a detailed design representation of software into a program language realization. System implementation implements the various components of the system based on the collected requirements, where the design is translated into a machine-readable form.

During implementation, all functionality planned in design phased is checked. It should be able to process the correct data and produce accurate information to end-users. Any problem or malfunction occurred is revised carefully and fixed accordingly.

6.2 System Implementation Phases

System implementation can be divided into four main phase that is:-

i) Data Collecting and Knowledge Base Preparation

ii) Testing and Developing Program

iii) Installation and New System Testing

iv) Delivering the New System for Operation

6.2.1 Data Collecting and Knowledge Base Preparation

Preparing a knowledge base for Milah chatterbot is not like any other chatterbots or program that uses normal database such as MySQL, MSAccess, etc. In Milah chatterbot, the knowledge base are arranged and kept in a folder

.../ aiml / standard / * .aiml

and then they are put in the bot file. Milah chatterbot is use for leisure chatting and not for high volume application, so using a normal database will only make the chatterbot more complex and unstable.

Bahasa Melayu is so much different from English and since Milah is the first Malay Speaking Chatterbot that has ever been developed, there is not enough sources and information available on books or the internet for me to refer and study. Most chatterbots that are available online are English, French, Spanish and Italian chatterbots. So,to develop Milah chatterbot, me and my partner have to study and refer the other language chatterbots, especially English chatterbots and find the similarities between other language and Bahasa Melayu. Collecting data and building the knowledge base for Milah chatterbot is not like writing any other computer programs, it is more towards art. Mostly it is about writing literature, grammar and psychology (because we have to know how human think and respond when they interact with chatterbots).

6.2.2 Testing and Developing Program

This phase is also known as the development phase. Program developing and testing usually takes a very long time and it is a very tiring phase in developing a system.

Program developer or botmaster has to work from the specification that has been developed and filtered through the prephase and preactivity in the Waterfall Model. If the specification of the system is not clear, not completed, not accurate or ruined, the development phase will be more complicated and takes a longer time.

The main input to this phase is the subset from the technical design statement that contains the specification of the program. The product of this phase is a program that has been completely tested to be used for production.

6.2.3 Installation and New System Testing

The next phase in system implementation is to install and test the new system with knowledge base piled up in it. The main input to this phase is a subset of the technical design statement that gives the specification on how the program has been developed and tested.

6.2.4 Delivering the New System for Operation

The final phase in implementation is to deliver the new system for operation. By providing a various system manuals helps users in using the new system.

6.3 Developing Milah The Malay Speaking Chatterbot

The phase that needs a very long period to be completed is the development phase of Milah The Malay Speaking Chatterbot itself. It involves interpretation, configuration, modifying and implementation of the bot development kit, Alice Program D into program codes. The Knowledge Base is coded using AIML (Artificial Intelligence Markup Language) that is derivative from XML, and the combination of technical and physical design into program codes had been done using two main languages, XML and HTML

6.3.1 Coding Phase

Coding phase is the phase where all the result from the analysis phase and the design phase is being transformed into a real application system. This phase also requires quite a long period of time to be completed because Milah The Malay Speaking Chatterbot is being develop using new high level programming language. The HTML, XML and AIML coding used the Macromedia Dreamweaver MX as the programming languages editor.

Besides considering the output from the phase before, other limitations factors for developing the system must also be considered. Several limitations factors that has to be considered are:

i. Limited time and energy for this phase

ii. Development cost factor

iii. Unpredictable output due to that Alice Program D is still undergoing for its stability and functionality improvements.

iv. AIML is a new and has not yet being a fully recognize programming language, and it is still on development phase by the Alice A.I Foundation.

6.3.2 Coding Style

Coding style is an important component of the source code and it determines the intelligibility of a program. An easy to read source code makes the system easier to be maintained and enhanced in future. Listed below are some of the coding styles used during the coding phase of this project:

- Selection of meaningful identifier names (variables, forms, tags).
- Description and an appropriate comment written in the source code to make it easier for the readers to understand the source code.
- Indentation of codes will increase the readability of the program and for a neater look.
- Meaningful and understandable function and method declarations.
- Keep all complex statement as simple as possible to avoid confusion.

6.4 Development Tools Implementation

Milah The Malay Speaking Chatterbot is develop using Alice Program D as the bot development kit and Macromedia Dreamweaver MX as the coding editor.

6.4.1 Development Tools and the Setup Steps

6.4.1.1 Alice Program D from www.alicebot.org

Alice Program D is available for free from the www.alicebot.org. However, before downloading Program D, developer must register for free and confirmation e-mail with a password then are sent to developer to allow access to the download page. Alice Program D consist all of these components as shown in the table below:

Component	Description
conf	**conf** is a folder consist of XML documents on how to configure the program to become a chatterbot that suit your preferences. It depends on what kind of bot you want to develop.
database	**database** is a folder consist a default database provided by Alice Program .The default configuration is entirely based on text files in order to make setup quick, painless, and no more resource-intensive than necessary. It suits for heavy-volume situations.
resources	**resources** is a folder consist of DTD and logs
templates	**Templates** is a folder consist example of HTML template
targets	**Targets** is a folder for viewing the log from targeting tool.

targeting	Targeting is for developer or botmaster to view the targeting tool. Targeting.properties is the property file of the targeting tool for the botmaster to edit. (This targeting tool is still under experimental and may not behave as expected.)
console	This is a java console for developer or botmaster to view the chatterbot.
run	This is the main execution file. Botmaster can test, view and also talk to the chatterbot by executing run.bat. After executing the file, it will automatically load the web interface for user to interact with the chatterbot.
server	Program D is designed to work as servlet and it uses Jetty,an open source HTTP server and servlet container. If botmaster want to run a web server, it can be edited from the server properties file or else, if a standalone chatterbot, like Milah is preferred, there's nothing much to be edited.
tester	This is the testing utility and can be turn on or off.
license	GNU Public License

Table 6.1 : Alice Program D components

Alive Program D is the easiest and robust bot development kit compared to other Alice Program and other bot development kit that uses other programming language besides AIML.

After downloading the Alice ProgramD.zip from www.alicebot.org, botmaster must unzip the file to any convenient location. Before running the program, botmaster must setup and configure the program depending on what kind of chatterbot they want to develop. Since our chatterbot, Milah The Malay Speaking Chatterbot, is a Malay speaking standalone chatterbot, there`s a lot of things that have to be configured.

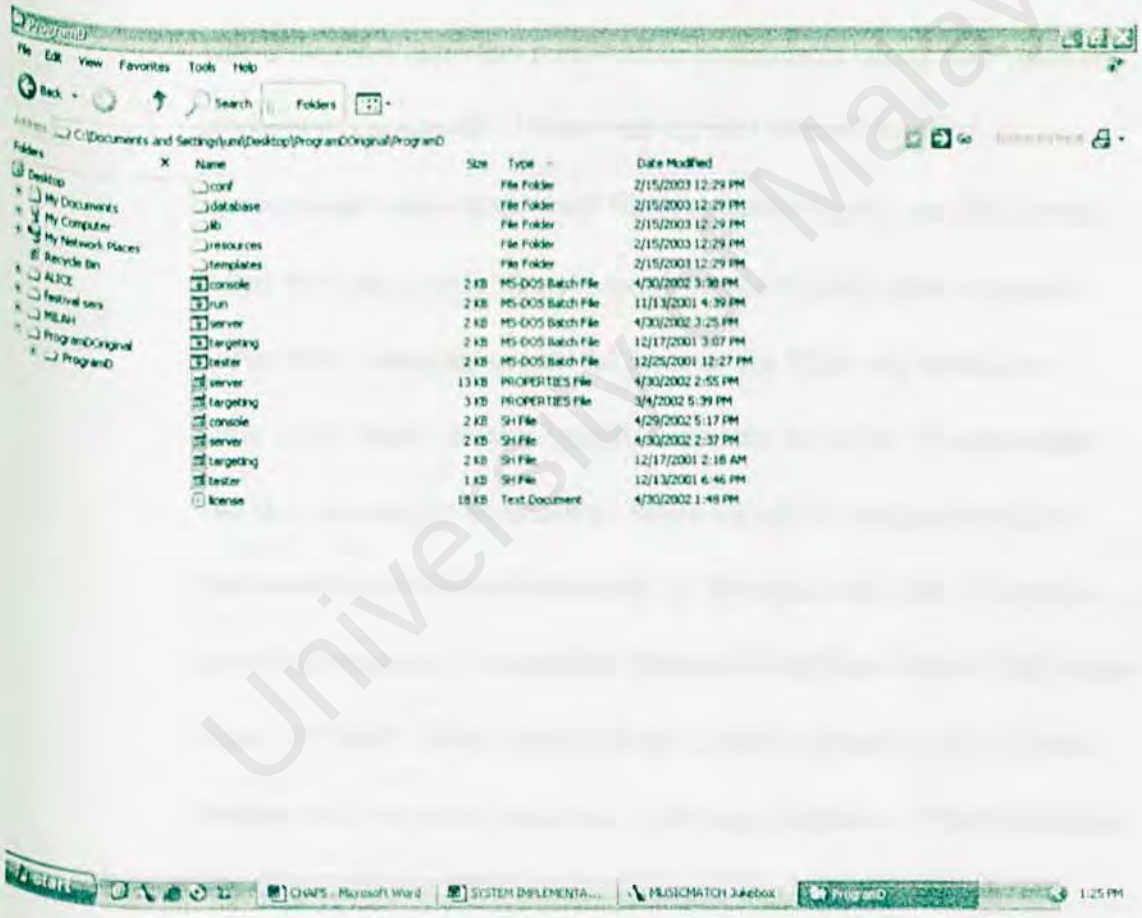


Figure 6.1: Program D that have been installed

The main components in the Program D that have to be configured to develop Milah The Malay Speaking Chatterbot are :

1. conf folder

ii. templates folder

iii. server properties

i) conf

- *Startup.xml* :

This is the most important part of all to configure to ensure that Milah bot production is successful. Notice that the root element is called **<programd-startup>**, and that it contains exactly one child element called **<bots>**..Inside **<bots>**, we place one or more **<bot>** elements. These **<bot>** elements are not the same as the AIML tag of the same name. Each **<bot>** element has two important attributes: **id** and **enabled**. The first one assigns an identifier, which should be unique, for the bot. The identifier will be used internally by the engine and will be written to some log resources. The **enabled** attribute should have either of the values "**true**" or "**false**". If the value is "**true**", then Program D will try to load that bot when the server starts up. Switching **enabled** to "**false**" is an easy way to quickly turn off a bot configuration that you don't want to use (although a restart is required).

Within the **<bot>** element we define **bot properties**, **default predicates**, **substitutions**, **sentence-splitters** and **learn directives**.

- *Predicates.xml :*

  Default predicates can be thought of as our bot's "assumptions" about new users.

  <predicate name="dia" default="somebody" set-return="name"/>

  This means that when **<set name="dia">...</set>** is included in a template, the name of the predicate, "**dia**", will be displayed, rather than whatever value is associated with the name by the **<set>**.

- *Sentence-splitters.xml:*

  Since sentence-splitters are applied to the input *after* substitution normalizations, they can be more general rules.

  <sentence-splitters>

  <splitter value="."/>

  <splitter value="!"/>

  <splitter value="?"/>

  <splitter value=";"/>

  </sentence-splitters

- *substitutions.xml:*

  Substitutions have several different purposes, depending on their type.

  Input substitutions contribute to the process of input normalization.

  Person substitutions provide macros for transformations by the <person>

tag; likewise **person2** and **gender** apply to the <person2> and <gender> tags, respectively. Each individual substitution specification, regardless of whether it is inside an **<input>**, **<gender>**, **<person>** or **<person2>**, takes the same form as this example from **conf/substitutions.xml**:

<substitute find=" ape " replace=" apa "/>

This means that, when this substitution is applied, each instance of the separate word "ape" will be replaced with "apa".

ii) Templates

The file `templates/html/chat.html` is used for constructing a web page with the bot's response. It is a plain HTML file, with a few important tags. There are examples of HTML template given, but I decided to use my own better template just to make Milah chat interface interesting for users.

iii) Server Properties

This is also a main file to be configured. It contains all the bot properties such as Main Program D configuration, AIML Watcher, Interpreter, HTTP Server, Shell/Console, etc. The configuration options are grouped for easy

maintenance. It can be edited by using WordPad or Macromedia Dreamweaver MX.

There are long list of parameters that have to be configured from the server.properties file. Following are descriptions of some of the important parameters that I have configured for developing Milah :

Shell/Console Configuration

In its usual configuration, Program D displays information about what it is doing while it starts, runs, and shuts down. We call this the "console". Also, you can interact with Milah via a simple shell, if desired. Sometimes it is simpler to use this shell than to open a web browser. Both the console and shell can be configured to suit our needs.

programd.console=true

-This tells the program to print information to the console. If this is set to **false**, almost nothing (except the copyleft notice) will be displayed when the program is run.

programd.console.match-trace=true

-Setting this property to **true** can help botmaster to understand what is happening when matching occurs.

programd.console.bot-name-predicate=name

programd.console.client-name-predicate=name

-If botmaster want to store the bots' and/or users' names in properties/predicates with names other than "**name**", then here's where it can be change . This property exists for the purpose of tuning the chat log functionality.

programd.console.warn-non-aiml=true

-Setting it **true** will warn botmaster when non-AIML elements is used outside of a template.

programd.console.timestamp-format=H:mm:ss

-Botmaster can choose how the timestamp looks in the console output. Whether a botmaster want it 24-hour or 12-hour style ,just see http://java.sun.com/j2se/1.4/docs/api/java/text/SimpleDateFormat.html for the formatting codes to use.

programd.shell=true

-In some situation botmaster may wish to disable the interactive shell. Set this parameter to **false** to do so.

After configuring, botmaster can start the bot by double-clicking **run.bat** file. The **run.bat** file is provided for Windows users so that common problems with the DOS environment can be handled before launching the **server.bat** file (which is started by **run.bat**). If there is error or problem with the configuration, the console window will produce error output or it will close before having time to read the error message. The botmaster can only interact with the bot after loading the AIML files into the bot (this will be discuss in coding implementation).

When the bot starts, the window console and the web browser will open . The window console and the web interface can be seen on the next page.

Figure 6.2 : Window console when the bot start

When botmaster load the AIML files, then and only then the bot, Milah can talk. Botmaster can understand what`s going inside the bot`s engine when the bot start interacting by understanding the match trace. To shut down the bot, simply type **/exit** at the window console.

To view the chat log or to see any error that occurs during conversation or during startup, botmaster can view the chat logs by opening the folder **logs** .To view Milah chat log with particular user or to identify the users personality or behaviour, botmaster can open the folder **ffm** . The log will be kept here and next time,when the same user interact

with Milah, she will remember them and their personality. All the personality are set and coded using AIML. Botmaster also use the logs file to add Milah`s knowledge.

Figure 6.3 : Windows showing the logs folder

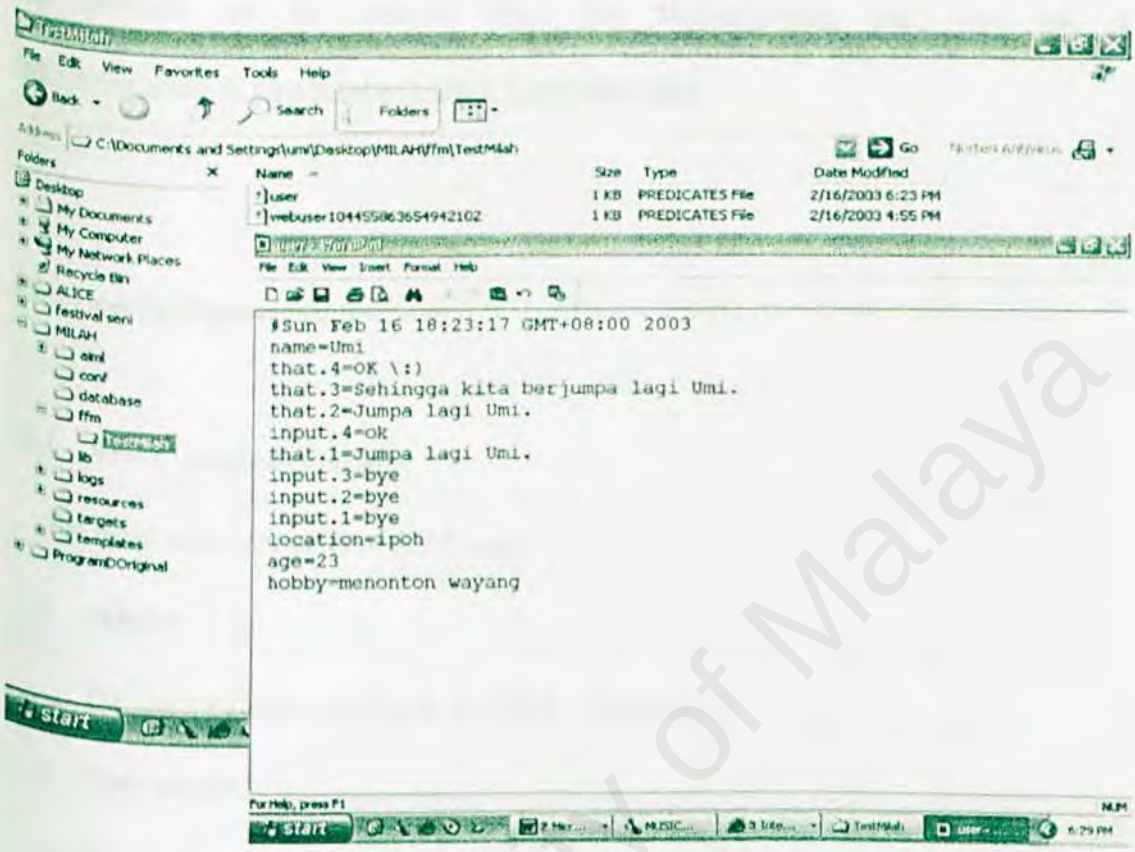


Figure 6.4 : Window showing the ffm folder

6.4.1.2 JavaTM Runtime Environment (JRE)

The JavaTM Runtime Environment contains the Java virtual machine, runtime class libraries, and Java application launcher. It is not a development environment and does not contain development tools such as compilers or debuggers. The JavaTM Runtime Environment (JRE) 1.3.1 or later is needed before installing Alice Program D.

The software can be obtained from Sun Microsystems, Inc., web site at http://www.javasoft.com/products/jdk/1.3/jre/index/html.

6.5 Coding Implementation

6.5.1 AIML Implementation

The most important units of AIML are:

- **<aiml>**

  The tag that begins and ends an AIML document

- **<category>**

  The tag that marks a "unit of knowledge" in Milah knowledge base

- **<pattern>**

  Used to contain a simple pattern that matches what a user may say or type to an Milah.

- **<template>**

  Contains the response to a user input.

There are also 20 or so additional more tags often found in AIML files and it's possible to create our own so-called "custom predicates".

Given only the <pattern> and <template> tags, there are three general types of categories.

- **atomic**
- **default**
- **recursive**

Strictly speaking, the three types overlap, because "**atomic**" and "**default**" refer to the <pattern> and "**recursive**" refers to a property of the <template>.

All Milah knowledge base or all the AIML files are kept in **.../standard/aiml/*.aiml** . Botmaster may enumerate each file they want the bot to load, or use simple glob-like expressions with "*".The path is relative to the location of this file. Milah only speak Bahasa Melayu and since it's new, it only talks about herself, her favorites, greeting, basic introduction and will ask users about themselves and their favorites. All Milah properties about herself and her favorites are kept in startup.xml. All the AIML files are group depending on the AIML" category "so that it will be easy to view, find, edit and maintain. Examples of AIML files that are group and kept in Milah are greetings.aiml, yesno.aiml, connect.aiml, inactivity.aiml, etc.

Here, I will not list down all the AIML coding from all the files because of the numerous categories. For simpler and better understanding, I will explain some examples according to category.

i) **Atomic category**

"**Atomic**" categories are those with atomic patterns, i.e. the pattern contains no wild card "*" or "_" symbol. Atomic categories are the easiest, simplest categories to add in AIML.

<category>

<pattern>SIAPA NAMA AWAK </pattern>

<template> Nama saya < bot name="name"/>. Siapa nama awak?

</template>

</category>

The above category (from …aiml/standard/basicintroduction.aiml)does the following:

- Matches the client input of "Siapa nama awak"
- Get the bot name , "Milah" as stated in the bot properties .
- Sends the client the response: "Nama saya Milah.Siapa nama awak?"

ii) **Default category**

The name "**default** category" derives from the fact that its pattern has a wildcard "*" or "_". The ultimate default category is the one with <pattern>*</pattern>, which matches any input. These default responses are often called "pickup lines" because they generally consist of leading questions designed to focus the client on

known topics. The more common default categories have patterns combining a few words and a wild card. For example this category (from...aiml/standard/basicintroduction.aiml) :

<category>

<pattern> * NAMA PENCIPTA AWAK</pattern>

<template>

Nama pencipta saya ialah <bot name="master"/>, yang juga botmaster saya.

</template>

</category>

responds to a variety of inputs from "Apa nama pencipta awak" to "Siapa nama pencipta awak" Putting aside the philosophical question of whether the robot really "understands" these inputs, this category elucidates a coherent response from the client, who at least has the impression of the robot understanding the client's intention.

iii) Recursive category

"**Recursive**" categories are those that "map" inputs to other inputs, either to simplify the language or to identify synonymous patterns. Many synonymous inputs have the same response. This is accomplished with the recursive <srai> tag. Take for example the input "BYE". This input has dozens of synonyms: "BAI", "BYE BYE, "TATA",

"JUMPA LAGI", and so on. To map these inputs to the same output for BYE (from ...aiml/standard/greetings.aiml) we use categories like:

```
<category>
<pattern>BYE</pattern>
<template>
<random>
<li>Jumpa lagi <get name="name"/>.</li>
<li>Bye <get name="name"/>.</li>
<li>Sehingga kita berjumpa lagi <get name="name"/>.</li>
<li>Terima kasih kerana sudi berbual dengan saya, <get name="name"/>.</li>
</random>
</template>
</category>
```

The recursive category:

```
<category>
<pattern>JUMPA LAGI </pattern>
<template>
<srai>BYE</srai>
</template>
</category>
```

6.5.2 Checking For Errors in Coding

If errors occurred in the AIML document, when the bot start, the window console will load the entire error messages and inform which file each of the error is located. The bot will still run normally but it will abort the entire category with errors. Botmaster can check and view the errors from the file **.../logs/error** and then correct the error to AIML documents.

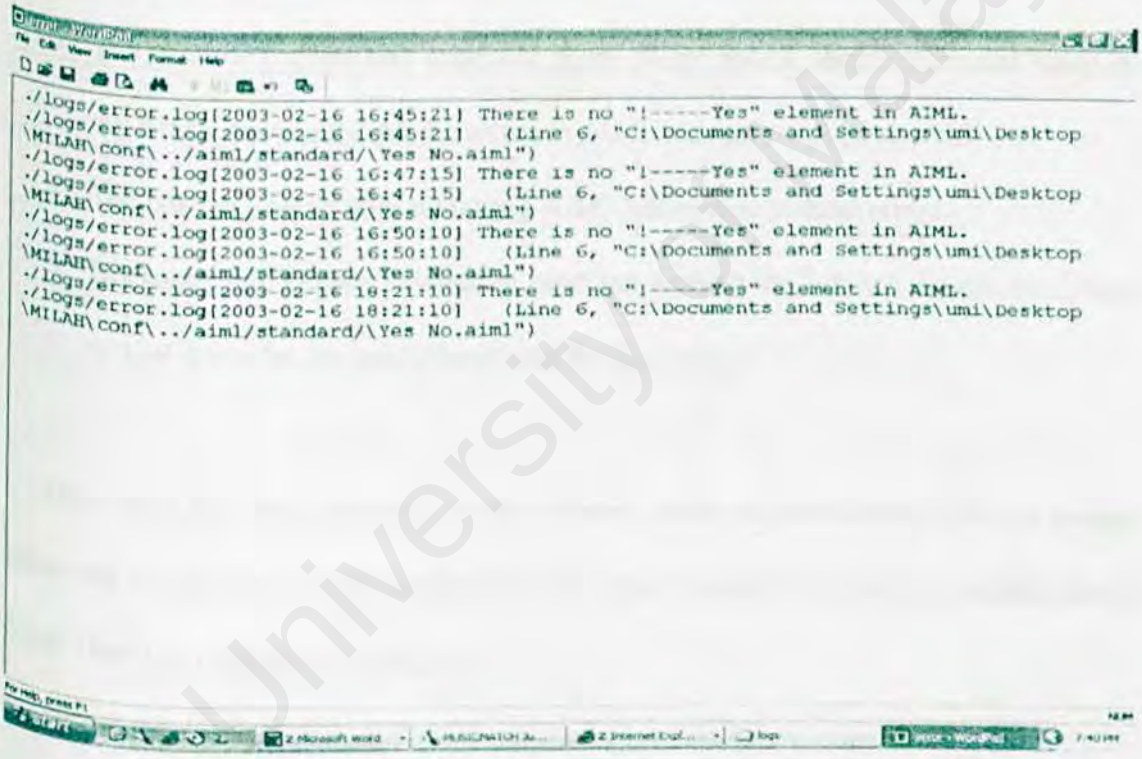


Figure 6.5: Window showing error logs in the AIML documents.

6.6 Summary

Below are the steps used to write, run and correct Milah The Malay Speaking Chatterbot AIML coding :-

1. Write the AIML coding in Macromedia Dreamweaver MX and save it as .aiml file in a specific directory (…aiml/standard/*.aiml)
2. Launch Milah by double-clicking run.bat.
3. The window console will open and show all the details about errors and status of the AIML documents. All the errors in AIML category will be aborted.
4. The bot can interact with remaining AIML categories without errors.
5. Botmaster can choose whether to continue running the bot and fix the error later or shut down the bot immediately and fix the errors.

From what has been explained in this chapter, it can be summarized that the coding phase was a very complicated phase and the longest duration in the process of developing Milah The Malay Speaking Chatterbot.

CHAPTER 7 : SYSTEM TESTING

7.1 Introduction

In ensuring the quality of software or a system, system testing need to be performed and it is one of the critical elements. This process involves careful examination of all the design specifications and coding process that has been performed along the system development process.

Testing is also performed to ensure that all the modules developed are free from any errors that can cause unreliability to the system from performing as required and to produce result as desired. Usually testing is performed using sample data and logics that are used in coding.

A good test is a test that is able to identify all the errors that are not detected during the analysis phase, design phase and coding phase. The main objectives in system testing are:

i. **Identify errors**

   Detailed checking is being performed to every function and behavior of the system to identify errors in the system.

ii. **Removing errors**

   Errors are removed from the system by compiling the codes after detecting the cause of errors or by debugging the system.

iii. **Regression test**

   To identifies new faults that may have been introduced as current ones are being corrected.

7.2 System Testing

System testing is ideally performed by developers using an environment similar to the production environment. This testing ensures that the system meets externally observable requirements including:

- Functional requirements, for example, "The system shall allow users to view their requested result or output."
- Derived requirements such as performance, robustness, and scalability.
- Usability requirements.

[Bennett, C., 2001]

The main intention of the testing process is to evaluate how much fault can be reduced in the program or in the module itself. The correction process on demonstration is against the meaning of testing. Testing is performed on the program to demonstrate existing fault. Since the main objective of testing is discovery of faults, all the faults that might lead to failures during actual system usage will be eliminate to ensure successful testing result. Fault identification is a process to determine fault or the cause of it, while fault correction is a process to make changes to eliminate fault.

To test the application, firstly, all AIML files must go through unit testing which is the simplest test of all, followed by module testing, which covers a wider scope of testing than unit testing. It takes a module and tests it out thoroughly. The result or output will be compared with the expected result. Then, an integration testing is performed to ensure the usability of the application. Furthermore, all links leading to all the modules

are tested out. An important point to remember when naming the link is; the name of the links should be clear and not be misleading.

In the testing process of Milah The Malay Speaking Chatterbot system, there are six main tests that have been conducted to ensure the system works as a whole. The testing stages are :-

i. Unit Test
ii. Integration Test
iii. Function Test
iv. Regression Test
v. Stability Test
vi. Usability Test

Testing sequence is as shown in Figure 7.1. Test performed on Milah chatterbot system is a bottom-up testing technique that is starting the test from the smallest unit until the system is entirely tested including the installation of the system.

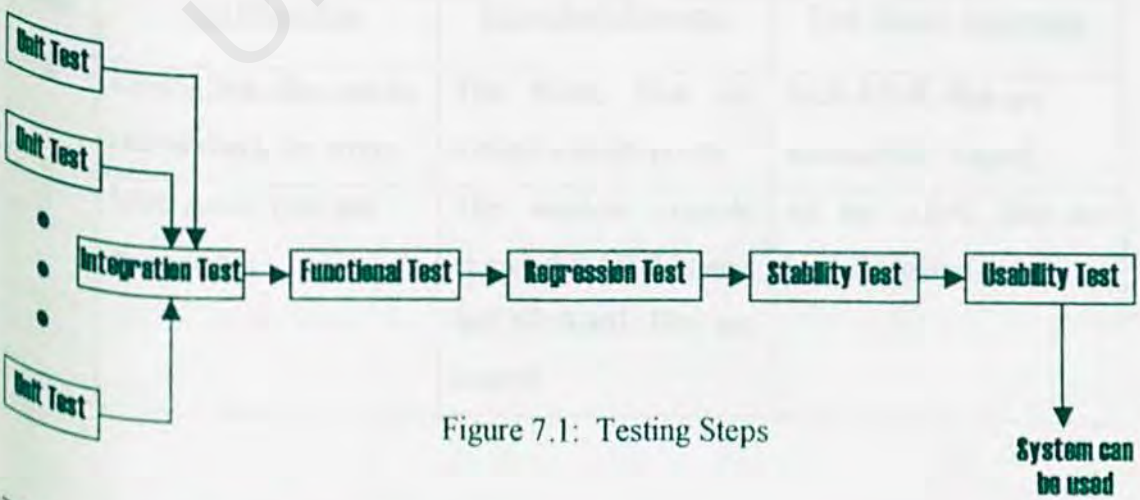


Figure 7.1: Testing Steps

7.2.1 Unit Testing

Unit testing is done by reading lines of code that has been written during the development of a module to identify any syntax errors, data and algorithmic errors. The programmer will repair these faults. After unit testing, the individual module will be compiled again to identify and fix any more errors incase there are still any errors undetected. This individual module will then be launch to ensure its effectiveness, accuracy and to see whether it functions as desired.

Unit testing is usually carried out by using the emulators. This testing includes test on every single program module components separately. Every file in the same module will interact internally or interact with other files in different module.

7.2.1.1 Unit Testing Example

Table below shows some of test cases for unit testing on the Milah The Malay Speaking Chatterbot :

Step	Test Procedure	Expected Outcome	Test Result Analyzing
1	Load AIML files one by one to check for errors	The AIML files are loaded without errors	Each AIML files are successfully loaded.
2	Load AIML files and execute bot	The window console shows the AIML status and all AIML files are loaded.	All the AIML files are loaded without error.

3	Each AIML category are tested by running the bot.	The bot recognize the entire category and produce the correct reply.	The bot successfully reply the correct output.
4	Run the bot	The window console and web interface will be loaded	The window console and web interface successfully loaded
5	Log in by clicking the log in button	Log in page will load	Log in page are loaded.
6	Talk to bot by typing input in the box and click 'cakap' button.	The bot will reply	The bot replied to user successfully.

Table 7.1 : Unit Testing Example

7.2.2 Integration Testing

This testing is done by taking the application from the top, following every links, using every available option and entering every possible data is necessary.

7.2.3 Functional Testing

Functional tests are captured in test cases and are derived from a requirements document. Function testing evaluates the system to determine whether the function described by the requirements specifications can really be presented by the system that has been integrated.

7.2.4 Regression Testing

The regression tests are made by re-executing some subset of the program's test cases, following changes to the application. Regression tests are used to verify that everything still works as it should.

7.2.5 Stability Testing

The stability test is done in the Milah chatterbot system by trying to test the application by using weird user input.

7.2.6 Usability Testing

The testing is done by performing "cold tests" on people, who are given a brief description of the application and told to use it for a while.

Each of the stages is likely to highlight areas where work is needed. If it turns out that the application does not fulfill the initial requirements, then changes need to be made.

7.3 System Testing Technique

Techniques used for testing depends on the testing level that has been set. At unit testing level, white box technique has been used to determine errors as shown in figure 7.2.

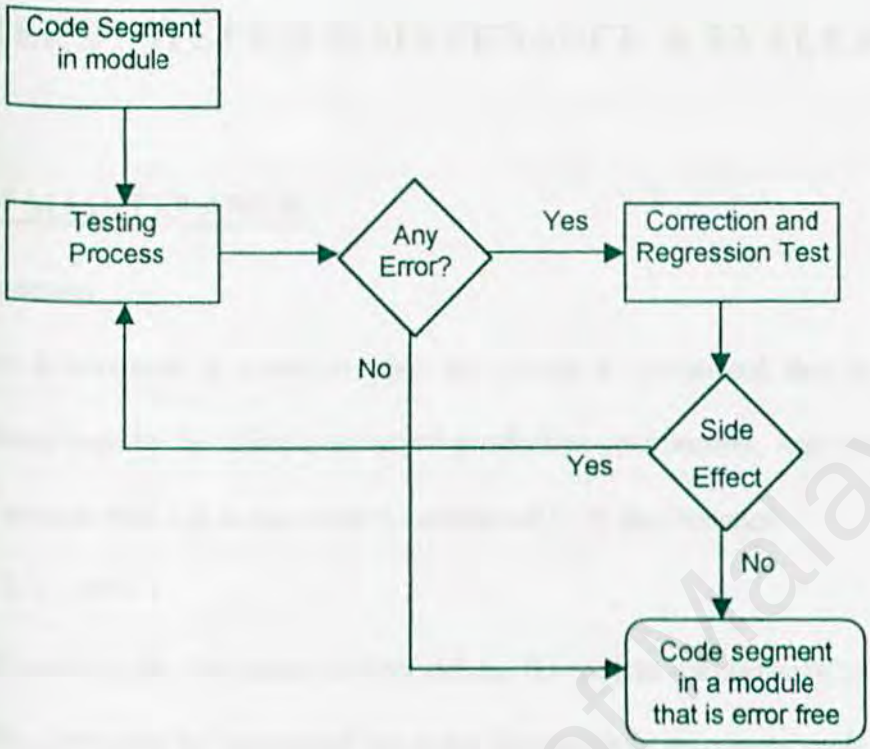


Figure 7.2: Unit Testing Scheme

7.4 Summary

Although application testing is executed after the implementation, continuous testing is done throughout the development and implementation of the application. Milah The Malay Speaking Chatterbot application should be tested thoroughly to ensure its reliability, its efficiency and its usability. It is very important that the user can talk and interact leisurely and smoothly with the bot. Usually a good application is able to fulfill the user requirement and keep running without much error.

CHAPTER 8 : SYSTEM MAINTENANCE & EVALUATION

SYSTEM MAINTENANCE

8.1 Introduction

System development is complete when the system is operational, that is, when the system is being used by the users in an actual production environment. Any work done to change the system after it is in operation is considered to be maintenance.

[Pfleeger, S. L., 2001]

In this section, the discussion is focused on the system maintenance requirements and how the system can be maintained when the functional requirements tends to change. This is to give guidance and understanding to users that will maintain the system so that it will not affect the system operations entirely during maintenance. Besides that, system recovery method is also included for this system.

8.2 Maintenance Requirements

Generally, a system has to go through maintenance in a routine basis to make sure the system is operating at optimum level. Maintenance has to be made to Milah The Malay Speaking Chatterbot system because of several reasons such as:

i. **Additional Knowledge**

Since Milah Chatterbot purpose is to talk and learn, so with every interaction with users, Milah will learn something from the user whether it's about the

user themselves or about any other things that user said. All the user informations and chat logs that are kept in folder **logs**,**target** and **ffm** will increase with every user interacting with Milah and it will be hard to handle if the size is too big .

ii. **Outdated Data Contents**

Overtime, there will be many other things that need to be updated in Milah such as its properties, and old and outdated information that stored in Milah's brain must be removed.

8.2.1 System Maintenance Methodology

This system can be maintained through various methods such as:

i. **Update the Knowledge Base**

To update and keep track of the knowledge base, botmaster must always *roll* the chat log. Rolling the chat log means the chat log will be minimized by the current one is renamed and kept in a specific location and the new one is created. The new one will contain a link to the previous one, so over time it can have a chain of log files. By doing this, the bot is easier to maintain. All this chat log are useful for detecting and correcting errors or collecting any additional information that the botmaster wanted to add to bot's knowledge

ii. **Disaster Recovery Plan**

Disaster recovery plan is made to provide support to system's operation incase a disaster occurs. The main contents of the system including the knowledge base have to be duplicated in a different storage device such as backup storage media, backup tape, diskette and other media storage device. If a disaster occurs, the system can still be retrieved and it does not have to be redeveloped. The contents of the Milah The Malay Speaking Chatterbot's knowledge base along with other documentations must be duplicated in the backup device from time to time so that the backup data stored stay updated.

SYSTEM EVALUATION

8.3 Introduction

The best way to develop a system is to involved system evaluation phase in the system development life cycle. This is the phase where a developer can analyze how successful the system that has been developed has reached its objective. Usually the developer will receive responses from users to evaluate a system.

8.4 Problems Encountered and Solutions

- Difficulty in choosing a suitable development tools

There are many bot development tools that are available for developing a chatterbot. It is difficult to choose the most suitable development tools from a

wide variety of choices. Choosing a suitable technology and tools was a critical process as all tools possesses their own strengths and weakness. Besides, the availability of a technology, hardware and supporting software to support, its learning curve, compatibility with the existence operating system and technologies are also the major consideration.

In order to solve the problem, seeking advices and views from project supervisor, course-mates and even seniors engaging in similar project were carried out. Furthermore, a great deal of reading and research from many resources like books and Internet regarding the problems helped to solve the problem and choose the suitable tools were done before any decision was made.

- Lack of ability

Since there was no prior knowledge of HTML, XML and AIML and the AIML is still new, there was an uncertainty on how to organize the codes. These programming languages and concepts were never familiar to me before and to implement such an application requires a fair grasp of the languages. These programming approaches seem to be different from other programming languages. Although it really cause a lot of time to learn this coding tool, but choosing to use AIML is easier compared to other programming language such as C++, Prolog, etc. Most of the problems faced were manageable through browsing the Internet for related materials and referring to the help function provided in the

software. Discussion with my partner was a great help. A more efficient method was through trail and error during the coding phase.

- Insufficient Reference Material

At the beginning of the project, the developer had insufficient reference material to develop the system. This is because, Milah is the first Malay speaking chatterbot ever been developed and the source for Malay language and how to implement a chatterbot are very limited and hard too find. Reference material at the market is sold at a very high price and the reference material in the library is limited and most of it is outdated.

The problems are overcome by getting help from our supervisors and he also lends us a few books related to chatterbot. We also join the discussions forum in the internet regarding the topics.

- Setting up tools

Lack of knowledge in setting up the bot configuration and the tools used in this system.

Confront the problem by surfing in the Internet, reading reference books and ask our supervisor.

8.5 System Strengths

- Simple, user-friendly and easy to use

  The interface design of this system mostly was created using Adobe Photoshop and Flash. It is designed to be as user-friendly and simple as this system is relatively used to chat. An action is just a click away and the user just needs minimal knowledge of mouse and keyboard to use this system.

- It speak Malay

  Milah Chatterbot speaks only Malay language and this is the only bot to be found that chat in Malay. Most chatter bots available on market or online are English speaking chatterbot.

- Easy to maintain

  Milah chatterbot are easy to maintain since Alice Program D provide very simple and easy to use functions. AIML coding are also easy to learn and write.

8.6 System Limitations/Constraints

- Limitations of answers/ reply, since Milah needs a lot of interaction with users to gain and to add up knowledge to its knowledge base. Milah will improve over time.
- Not many people will use or talk to chatterbot especially for leisure, unless for commercial or educational purpose.

8.7 Future Enhancement

System development is a dynamic process and changes must be expected. Due to limited resources, especially time, had caused me to miss or overlooks certain aspect of the system. However, after the development system has been completed and valuable advices and suggestions from my project supervisor and moderator, I have identified certain important aspects that can be improved for future enhancement. The additional features that can be implemented in future are as followed:

- It can work as Malay dictionary by adding a function or by linking to an online dictionary.
- Milah will give more logic and reasonable replies to user even to topic that it doesn't know.

8.8 Knowledge and Experience Gained

This project does push me up to another level that is full with new terms and technologies .Through the application, theories learned in classroom can be applied to my practical work. For examples :

- The importance of all those theories and lessons on Case-based Reasoning, Natural Language Processing and other Artificial Intelligence topic.
- How to keep track of the ongoing development
- The importance of all phase in the System Development Life Cycle (SDLC).
- How useful and interesting some development tools can be when developing a chatterbot, such as Macromedia Flash and Dreamweaver, Adobe Photoshop. As

an AI student, I have never expose to these software before and it is a good exposure for me to try all the new utilities.

Other than that, I have learned something that is not been taught in classroom, for instance, the skills of sharing and communication and learning new programming languages. I believed the concept of gaining and contributing knowledge is something beneficial and this exposure will be very useful in the future.

8.9 Summary

Evaluation of system is indeed to ensure its objectives and intended functions have been achieved. This chapter covers all the aspects of the evaluating application software. The successful development of the system at the present is the first step towards the future expansion of the system. The problem encountered and experience gained during the development phases should be helpful in future efforts.

Besides, this chapter also summarizes the system strengths, system constraints and future enhancements that can be added. The future enhancements will equip the system towards more capabilities of doing its daily operations and activities.

REFERENCES

Allec,James (1995). *Second Edition,Natural Language Understanding*.The Benjamin/Cummings Publishing Company, Inc.

Dr.Abdullah Embong (2000). *Sistem Pangkalan Data: Konsep Asas, Rekabentuk Dan Pelaksanaan*. Tradisi Ilmu Sdn. Bhd.

Pfaffenberger Bryan and Karow Bill (2000).*HTML Bible.2nd Edition*.IDG Books Worldwide,Inc.

Kendall Kenneth E. and Kendall Julie E. (1992). *System Analysis And Design*. 2nd Edition. Prentice Hall Inc.

Mohamad Noorman Masrek, Safawi Abdul Rahman, KamarulAriffin Abdul Jalil (2001). *Analisis & Rekabentuk Sistem Maklumat*. McGraw-Hill (Malaysia) Sdn. Bhd.

Shari Lawrence Pfleeger (2000). *Software Engineering: Theory And Practice*. Prentice Hall Inc.

LIST OF WEB SITES

http://www.microsoft.com

http://www.adobe.com

http://www.macromedia.com

http://www.netscape.com

http://www.zdnet.com

http://www.xml.com

http://www.java.sun.com

http://www.alicebot.org

http://www.agentland.com

http://directory.google.com/Top/Computers/Artificial_Intelligence/Natural_Language/Chatterbots/

http://directory.google.com/Top/Computers/Artificial_Intelligence/Natural_Language/Turing_Test/

http://www-ai.ijs.si/eliza/

http://www.manifestation.com/neurotoys/eliza.php3