**GROUP ASSIGNMENT 2**

Expert Systems with Uncertainty

**ARTIFICIAL INTELLIGENCE**

CSC 3206

**TUESDAY P3 1**

Muhammad Awad Luckhoo (15042278)

Choong Kai Wern (15053648)

Teh Cuok Syen (15061567)

Ong Li Shen (15071863)

Mu Chun Khang (13079272)

Mah Qi Hao (16080111)

# TABLE OF CONTENTS

[TABLE OF CONTENTS 2](#_Toc485898682)

[QUESTION 1 3](#_Toc485898683)

[Literature Review 3](#_Toc485898684)

[Comparative Analysis 3](#_Toc485898685)

[Justification 3](#_Toc485898686)

[ES Design 3](#_Toc485898687)

[QUESTION 2 4](#_Toc485898688)

[Literature Review 4](#_Toc485898689)

[Quality of Recommendations 4](#_Toc485898690)

[QUESTION 3 5](#_Toc485898691)

[Contributions of Members 5](#_Toc485898692)

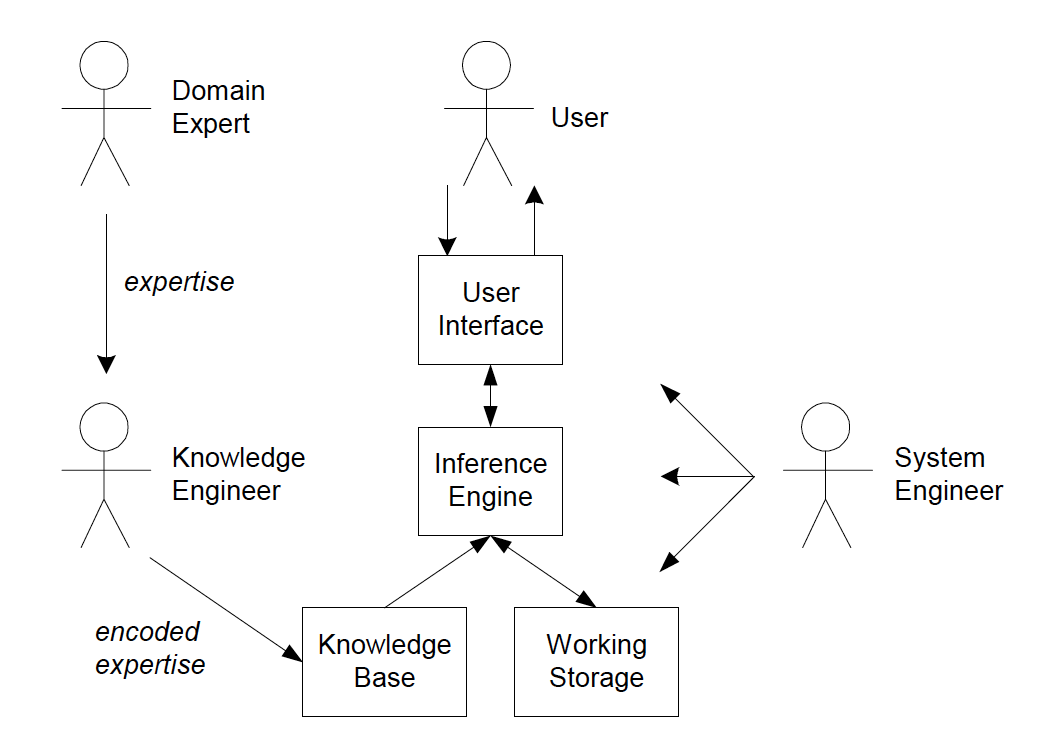
[Gantt Chart 5](#_Toc485898693)

[REFERENCES 6](#_Toc485898694)

# QUESTION 1

## Literature Review

First and foremost, the paper conducts a literature review on expert systems (ESs) before attempting to dive into other sections. This way, the readers can comprehend ESs in terms of components, applications and drawbacks. In essence, ESs are computer applications that employ non-algorithmic expertise to solve specific problems (Merritt, 2012). They consist of several major system components and interface with individuals playing different roles. Figure 1 depicts the components of an ES.



**Figure 1:** Components of an ES

Knowledge base is a declarative representative of the expert knowledge, and mostly expressed in if-then rules. Working storage is the data pertinent to the problem that is being solved. Inference engine is the code found the at system core that procure recommendations from the knowledge base and problem-specific data stored in the working storage. User interface is what controls the dialog and interaction between the system and the user. Herein, the roles of the individuals also imperative in the ES. Domain expert is the one who is an expert in solving the problems the system aim to solve. Knowledge engineer is the one who encodes the expert’s knowledge into a form usable by the ES. User is the one who will be consulting the ES.

One of the earliest ESs is Dendral that analyses mass spectral patterns to deduce chemical structure of unknown compounds (Duda & Shortliffe, 1983). On the other hand, Mycin is another ES which was developed in Stanford University to help physicians in selecting antibiotics for patients suffering from severe infections (1983). Meanwhile, PROSPECTOR is an ES designed to evaluate resource, to identify ore deposits and to select drill sites (1983).

While ESs seem like a viable way to replace experts given the number of successful applications, they are not without their limitations. The greatest bottleneck in building an ES lies in the process of knowledge engineering. Interviewing domain experts to convert their expert knowledge into declarative rules can be arduous task.

## Comparative Analysis

For the purpose of building an ES, the paper discusses two main development tools: ES shell and ES programming language. To illustrate the differences between both ES development tools, this section of the paper shall provide an exhaustive comparative analysis. In this context, an example is given for each respective ES development tool. The tools are first discussed in terms of their common purpose. Subsequently, their distinct features are deliberated individually. This should provide readers with a good overall comparison of both ES development tools.

It is imperative to note that ESs use reason of knowledge to solve complicated problems. The systems are represented predominantly with if-then rules, and not the conventional procedural code. At this juncture, the development tools for ESs must be specialized. An ES development tool is simply a software development environment that contains the basic components of an ES: knowledge base and reasoning engine (Wtec, 1993). Typically, ES development tools come with prescribe methods of building applications through configuration and instantiation of these components (1993). Besides that, developers are also offered numerous choices when designing an ES. The methodology, mode of knowledge representation, software development package and hardware to be implemented on are among the factors for consideration (1993). Herein, all ES development tools, be it ES shell or ES programming language, are crucial in aiding developers construct ESs.

ES Shell

ES shells consist mainly of a user interface, inference engine and an editor to assist developers in building their knowledge base for ESs (Wtec, 1993). A great example of an ES shell is Jess which stands for The Java Expert System Shell (JessRules, 1997). Jess is an ES shell written in Java entirely, driven by a Lisp-styled scripting language (University of New Mexico, 2017). Java provides the external mechanisms that generate and control the rules (1993). To use Jess, the data must first be converted into text before it is handled by the interpreter (2017).

Jess can be utilized in two ways, namely as a rule engine or as a general-purpose programming language (Wtec, 1993). A rule engine is a special program that applies rules to large sets of data in an efficient way. A rule-based program can have up to thousands of rules, but Jess will have no problem applying them as data in the form of a knowledge base. In particular domains, the rules are represented as the heuristic knowledge of human experts, while evolving situations are represented by the knowledge base. On the other hand, as a general-purpose programming language, Jess can be extended easily as it directly accesses all Java classes and libraries (University of New Mexico, 2017). New commands can be written in Java or Jess to be integrated into Jess. Thus, Jess is very customizable when it comes to building applications.

The advantages of Jess include the ease of working with the code builder because it is an independent scripting language. Jess releases the burden from developers because they are not required to declare each rule as a set of nested class instances (Retchford, 2017). On top of that, the programming effort for building the user interface and inference engine is also greatly reduced. Projects can be completed faster and cheaper in an efficient manner. However, Jess has the disadvantage of Java being disconnected from the rule engine. Normal Java syntax cannot be used to debug the syntax after external files and strings have been used to specify the rules (DeveloperIQ, 2014). Besides that, Jess implements the Rete algorithm to match rules against the knowledge base. Since the algorithm trades space for speed, Jess requires relatively-large memory usage for moderately-sized programs.

ES Programming Language

Expert programming languages are high-level languages commonly used in ESs due to their high capability in handling symbolic data efficiently. One example worth mentioning is Prolog, a language that is backward-chaining where it works backwards from a goal to find supporting facts (DeveloperIQ, 2014). A Prolog program consists of a set of clauses whereby a clause is either a fact or a rule used to indicate a relationship between elements (2014). Queries are entered using command-line tools provided by Prolog.

The process of deriving reason for a Prolog program can be broken down into multiple steps (Jabbar & Khan, 2016). Initially, a goal is given. Prolog searches the database from top to bottom for a fact that matches the goal. Then, a pointer is left where the match is found before Prolog instantiates the suitable variables. When a goal matches the head of a rule instead of a fact, the atoms within the rule’s body are treated as sub-goals that must all be satisfied to prove that the head is satisfied. Herein, the order of entries is important as the arrangement influences the number of search required to satisfy the goal and find a solution. In fact, there could be situations where Prolog may not find a solution even though a human can easily identify it from the given information.

Prolog has the advantage of being a powerful language that deduces the desired supplementary facts using strong built-in deduction system. The developer is freed from implementation details as he or she only needs to define what is required rather than indicate how it should be computed. Prolog also encourages modular programming and incremental developments, making program tracing and debugging a simple process (DeveloperIQ, 2014). Nevertheless, it is very difficult to design a database that accurately represents relationships (2014). Moreover, Prolog is not a suitable language to solve complex arithmetical computations (2014).

## Justification

The researchers of this paper are now faced with a choice between two ES development tools to build a specific ES. In this context, the problem domain is education, while the knowledge domain is decision-support to decide programme of study and career options. After the researchers of this paper have done a thorough comparative analysis of both ES development tools, they decided to employ Prolog, an ES programming language. This section of the paper justifies the decision by briefly revisiting the key differences between Prolog and Jess before highlighting several winning aspects of Prolog.

Prolog and Jess are two very different tools. Prolog is console-based, while Jess is user-interface-based. This implicates that Prolog is more suited for developers, while Jess is more towards end users. Since the researchers of this paper all possess a background in programming, Prolog resembles a more intuitive tool to work with than Jess. With that, the researchers can employ Prolog focusing on answering queries rather than working on the user interface of Jess that only acts in response to input. Lastly, Prolog programs are more optimised in space in comparison with Jess (Friedman-Hill, 2002). As discussed previously, Jess uses the Rete algorithm that reuses computations to save time. Prolog is good at exploring large numbers of possibilities at once, whereas Jess explores medium-sized numbers of possibilities repeatedly (2002).

One of the important aspects of Prolog worth mentioning is collections (Cong, Shin, & Salehnia, 1992 ) As a high-level language, Prolog supports other data structures other than lists. This offers flexibility to developers in solving problems. Moreover, Prolog utilizes pattern matching (1992). As a language mostly associated with artificial intelligence and computational linguistics, Prolog uses pattern matching for associative retrieval and pattern action rules when there is explicit structure decomposition. Also, Prolog allows facts to be written as assertions (1992). These assertions will have the associative data base that allows easy and natural access. Last but not least, Prolog boasts the feature of nondeterminism that abstract points within unnecessary details (1992). In other words, nondeterminism permits abstraction from a set of computations where only one of them needs to succeed.

## ES Design

# QUESTION 2

## Literature Review

At this juncture, this paper has expounded enough on expert systems. Now, the literature review focuses on uncertainty in expert systems. Readers should be able to understand what uncertainty is, where it comes from and why it should be introduced into expert systems. Briefly, uncertainty is the lack of precise knowledge that prevents a perfect conclusion from being reached; it is where probabilistic descriptions capture the environment (Russell, Norvig, & Intelligence, 1995).

Uncertainty comes from various sources (Russell, Norvig, & Intelligence, 1995). One of the noteworthy sources is laziness. Sometimes, it becomes too much work to list out every set of antecedents or consequents required to ensure a rule that is exception-less. Besides that, theoretical ignorance also plays a role. For instance, the domain medical science cannot be encapsulated by a complete theory. Moreover, practical ignorance means that even if every rule was known, not all cases can be tested to be completely certain. The user may enter an uncertain data. There are rules where the premise or conclusion is uncertain. Combining uncertain premise with uncertain conclusion also spawns uncertainty.

For structured selection problems, the final answer often cannot be known without utter certainty (Merritt, 2012). The expert rules may be unclear, and the user might be uncertain with the answers to questions. One good instance is medical diagnostic systems where the expert system is unable to be definite about the relationship between symptoms and diseases. In fact, given how a doctor would offer multiple possible diagnoses, the expert system should be no different. Hence, expert systems must be capable of dealing with uncertainty to function in the real world. A simple scheme that can be adopted is to associate a number with each piece of information in the expert system. These numbers would represent the certainty to which the information is known.

## Quality of Recommendations

# QUESTION 3

## Contributions of Members

Muhammad Awad Luckhoo

Gathered information, researched about various expert system development tools, compare quality of recommendation of expert system, and compiled report.

Choong Kai Wern

Gathered information, justified expert system design, implemented expert system, implemented uncertainty to expert system, and evaluate the system strengths and weaknesses.

Teh Cuok Syen

Gathered information, researched about various expert system development tools, implemented the expert system, prepared Gantt chart, and compile report.

Ong Li Shen

Gathered information, justified expert system design, implemented the expert system, implemented uncertainty to expert system, and evaluate the system strengths and weaknesses.

Mu Chun Khang

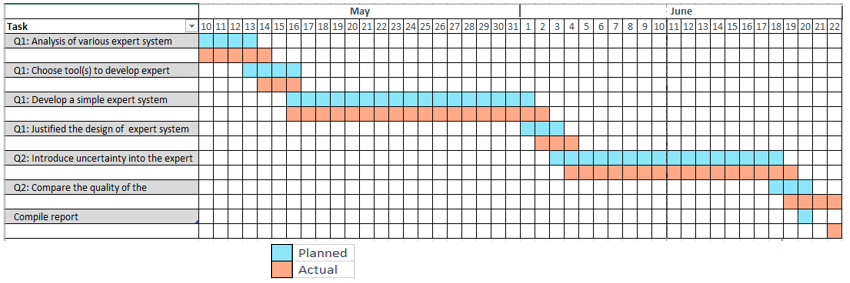
Gathered information, researched about various expert system development tools, justified expert system design, implemented the expert system, and compiled report.

Mah Qi Hao

Gathered information, justified expert system design, implemented the expert system, implemented uncertainty to expert system and evaluate the system strengths and weaknesses.

## Gantt Chart

Figure 1 shows the Gantt chart that illustrates the planned work schedule versus the actual work schedule.



**Figure 1:** Gantt chart

# REFERENCES

Cong, B., Shin, S., & Salehnia, A. (1992). Expert Systems and Expert System Languages.

DeveloperIQ (2014). Expert systems: Programming languages, tools and shells. Retrieved from <http://developeriq.in/articles/2014/jul/14/expert-systems-programming-languages-tools-and-she/>.

Duda, R. O., & Shortliffe, E. H. (1983). Expert systems research. Science, 220(4594), 261-268.

Friedman-Hill, E. (2002). Jess, the expert system shell for the java platform. *USA: Distributed Computing Systems*.

Jabbar, H. K., & Khan, R. Z. (2016). Tools of development of expert systems: A comparative study. In *Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on* (pp. 3947-3952). IEEE.

JessRules (1997). JESS, The Java Expert System Shell. Retrieved from http://www.jessrules.com/docs/52/.

Merritt, D. (2012). *Building expert systems in Prolog*. Springer Science & Business Media.

Retchford, S. (2017). A Review of Jess: The Java Expert System Shell. Retrieved from http://http-server.carleton.ca/~aramirez/4406/Reviews/SRetchford.pdf.

Russell, S., Norvig, P., & Intelligence, A. (1995). A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, *25*, 27.

University of New Mexico (2017). Case Studies: JESS and other Expert Systems Shells in Java. Retrived from https://www.cs.unm.edu/~luger/ai-final2/JAVA/CH%2026\_Case%20Studies%20-%20JESS%20and%20other%20Expert%20System%20Shells%20in%20Java.pdf.

Wtec (1993). EXPERT SYSTEMS BUILDING TOOLS: DEFINITIONS. Retrieved from http://www.wtec.org/loyola/kb/c3\_s2.htm.