USING RULE-BASED TECHNIQUE IN DEVELOPING THE TOOL FOR FINDING SUITABLE SOFTWARE METHODOLOGY

Mastura Hanafiah[1], Zarinah Mohd Kasirun[2]
Department of Software Engineering,
Faculty of Computer Science and Information Technology, University Malaya
50603, Kuala Lumpur, Malaysia
Email:[1]*masturahanafiah@yahoo.com*, [2]*zarinahmk@um.edu.my*

ABSTRACT

Software development methodology involves many activities and processes that are carried out when building a software system. There are a lot of available methodologies; one should be suitable for a software project. However, deciding which methodology to be applied requires some assessments on the project nature and characteristics at the early project cycle. This research focuses on finding the relationship between software project factors and methodologies, and further developing a tool that can help software practitioners in choosing the most suitable methodology. Before the tool is developed, a review on several lifecycle models has been carried out in order to examine the relevant software factors. Factors like project size, complexity, requirement stability, duration, performance requirement, modularization, project team, and criticality have the main impact on some methodologies. The relationship between methodologies and software factors is formulated using rule-based approach, and managed properly by a sequence of steps; identifying the initial selection of input variables, counterexamples of bad sub rules, pruning the variables, merging categories, and identifying hypothetical examples. RETE algorithm has been chosen as the problem solving technique in managing the rules. The tool is able to help software practitioners in early decision making process to use appropriate methodology in their software project.

Keywords: Software development process, software engineering, software development methodology, software project factors, RETE algorithm, rule-based implementation.

1.0 INTRODUCTION

The growing numbers of software applications today has resulted in many ways of techniques and approaches of software engineering practices. Many software process methodologies have been criticized, revised and re-modeled in order to fulfill the way a software development work. Software methodology is a formalized development process that is used when building software which starts when the software is conceived until the software is in use. The basic activities involved in building software are analyzing the requirements, designing, implementing, validating and maintaining the system. Even though the process of developing software is similar from one methodology to another, the approaches are rather different. This many approaches may lead to an unsuitable choice of software development methodology. If inappropriate methodology is used, it may create many difficulties during the development and maintenance period, and in addition, it can lead to software project failure.

However, there is no general software development technique powerful enough to handle all the variations of today system [12]. Selecting a software development methodology is highly driven by many factors. Human factors, technology factors, and application nature contributes to the choice of a software development process in a project. Hughes emphasizes that methodologies and technologies to be used requires decision-making process, or project analysis [6]. Thus, an assessment of the project is needed before making such decision. The characteristics of the projects depend on the nature of individual project; therefore, each project requires its own analysis. This research focuses on these main areas:

1. Identifying various methodologies in software development.
2. Identifying the critical factors that affect the choice of methodology.
3. Developing a tool (SUIT-Method) for selecting the most suitable software development process using suitable problem solving methods in artificial intelligence field.
4. Evaluating the tool's usability, functionality and quality

This paper is organized as follows. Section 2 presents the background of existing popular methodologies and the identification of related software factors. Section 3 discusses about the possible approach to formulate the relationship between software methodologies and software factors. Section 4 is the development of the tool using rule-based approach. Section 5 discusses about the tool's evaluation and results. Section 6 summarizes how the research objectives being achieved, its contribution and future works.

2.0 SOFTWARE METHODOLOGIES AND SOFTWARE FACTORS

Until today, there are a lot of methodologies available in the area of software development. This research addresses those that are the most common among software practitioners. The more common process models presented in the survey of system development process models lies from these three approaches [16]: Adhoc Development, Waterfall Model and Iterative Process. Process capabilities in Adhoc Development are unpredictable because the software process is constantly changed or modified as the work progresses. Schedules, budgets, functionality, product quality and performance are generally inconsistent. Waterfall Model, on the other hand, is more rigid and unrealistic; however, it is still widely used. This is because Waterfall Model becomes the basis of the other process models and it is suitable for certain project conditions. Iterative models are the solution to the linear waterfall models because of its flexibility and faster results. Each iteration is a mini-Waterfall process where feedback from one phase is used for the design in the next phase. Iterative models include Prototyping (Rapid Application Development and Throwaway or Exploratory), Iterative and Incremental, Spiral Model and Rational Unified Process. Agile has becoming popular these days because it offers lighter weight methodology. Abrahamsson et al. describes agile process as people matter, less documentation is possible, communication is a critical issue, modeling tools are not as useful and big up-front design is not required [17]. There are four agile methods have been analyzed and identified to have clear distinction in four areas: team size, iteration length, iteration support and criticality; they are Extreme Programming, Scrum, Crystal and Feature Driven Development [17].

Each methodology has its own characteristics; thus, each should be dependable on many different factors. Table 2-1 summarizes the main characteristics of the non-agile methodologies while Table 2-2 characterizes agile methodologies in terms of team size, team distribution, iteration length and criticality.

Table 2-1: Software Lifecycle Analysis

Lifecycles	Characteristics
Waterfall (WA)	- each phase completes before next phase can begin. - much formality, ceremony and detailed documentation. - good when requirements are stable, well defined, and small.
Throwaway prototyping (TP)	- explore new technologies and determining the applicability and effectiveness before adopting such technology. - good for proof-of-concept or research and development work
Evolutionary prototyping (EP)	- suitable for larger system where the details are difficult to be established. - need a multi-competence skill of people - modular and iterative - suitable for highly interactive application with stylish user interface.
Rapid Application Development (RAD)	- requires sufficient human resources to create the right number of teams. - need committed developers and customers - not for system with high performance issue. - suitable only for low technical risks - modularized for component reuse
Iterative and Incremental (II)	- requirements are poorly defined or unstable - when the technology is risky - to build a reusable application framework - staffing is unavailable for a complete implementation - rate of requirement change in each iteration should be zero - high risk in user interface but low risk in budget and schedule
Spiral (SP)	- suitable for large-scale systems and software.

	- use prototyping as a risk reduction mechanism - need considerable risk assessment expertise - high risk in any technical aspects (budget/schedule high, quality high, performance high, new technology high, etc) at all stages
Rational Unified Process (RUP)	- support iterative and incremental development - extensive documentation needed - multiple roles needed, thus need enough resources

Table 2-2: Prescriptive Characteristics of Agile methods [1]

	Extreme Programming (XP)	**Scrum (SC)**	**Crystal (CR)**	**Feature Driven Dev. (FDD)**
Team Size	2-10	1-7	Variable	Variable
Iteration Length	2 weeks	4 weeks	< 4 months	< 2 weeks
Distributed Support	No	Adaptable	Yes	Adaptable
System Criticality	Adaptable	Adaptable	All types	Adaptable

From these characteristics, several factors that have impact on methodology selection are identified. They are size, complexity, requirement stability, duration, user interface requirement, performance requirement, criticality, iterative and incremental behavior, modularity, proof of concept system, process and documentation requirement, risk assessment, project team, iteration length, and sufficiency of resources.

3.0 THE RULE-BASED APPROACH

The process of determining the most suitable methodology for a software project cannot be avoided without a decision making process in place. Considering the fact that the pattern of producing a methodology is based on software factors, some suitable decision making techniques are analyzed. From the previous similar researches [8][13][2], it can be concluded that the method that is mostly used in selecting a methodology is rule-based approach. Other methods that are also analyzed are case-based reasoning and database system. Case-based reasoning is excellent if there exists sufficient past similar cases. However, during the early assessment of this research, similar cases could not be established because of lack of reliable data regarding software factors and methodologies from real software industry [19]. Database system, on the other hand, is excellent at finding exact matches, but they are poor at near or fuzzy matches. Moreover, the factors and methodologies that are evaluated in this research cannot be stated with absolute confidence [19].

Table 3-1 lists the characteristics of rule-based approach which make rule based implementation applicable for SUIT-Method.

Table 3-1: Characteristics of Rule-Based Approach

Characteristics of Rule Based System	**Applicability to SUIT-Method**
The algorithm involves significant conditional branching or decision-making.	A lot of factors depends on the previous factors, thus involves conditional branching and decision making.
There are three or more conditions in the rules exist (for example, a block with 3 or more nested if-statements in pseudo-code).	More than three conditions are definitely needed to justify suitable methodology.
The rules are likely to change over time due to the nature of the application.	With feedback and evaluation, the rules are likely to change in the future.
The performance requirements will accommodate a rule engine solution.	Even though performance does not really matter, it however increases the efficiency of the system.

Because the rules involve a lot of factors and the resulting actions might depend on the previous conditions, the rules can become more complicated to manage and maintain. Furthermore, some factors only applicable to some methodologies, and some factors can only be combined with some other factors. Because of too many factors and variables, some techniques suggested by Whalen and Schott are used when constructing a rule-based application [14]. These techniques involve several steps:

1. Initial selection of input variables.
   In order to deal with too many variables, a matrix between factors and methodologies has been constructed. Those factors commonly appear in the methodologies are taken into account.
2. Counterexamples to bad sub rules.
   Some factors cannot be combined with other factors, which can lead to bad sub rules. Thus, two categories of rules are identified: rules for small size system with low complexity and high requirement stability, and rules for system with at least medium size, complexity and low requirement stability.
3. Pruning variables
   The factors that are identified earlier but do not make any difference are eliminated. These includes project type and application domain.
4. Merging categories
   Many categories can be formed to identify software methodology. It can be based on project size, or it can be based on project type, etc. For this research purpose, only two categories are identified that will further simplified the rules construction.
5. Hypothetical examples
   The best starting point for a set of hypothetical examples is to look at the most extreme case. Thus, the rule starts with identifying factors for system with small size, less complex and high requirement stability.

3.1 The Initial Rule Algorithm

A. Category 1: Rules for system with small size, low complexity and high requirement stability

Fig. 3-1 and 3-2 show the conditional branching for system with small size, low complexity and high requirement stability. Fig. 3-1 concerns with system whose duration is less or equal than three months, while the other figure concerns with system whose duration is more than three months. If the system is a proof of concept, the system will suggest adapting Throwaway Prototyping at the beginning of the development cycle. If the duration is less than three months, the potential candidate for this type of system is using Rapid Application Development; however, the system must be properly modularized and has sufficient resources in order to complete it within the short time. If these conditions are not met, then, the system checks whether it needs high user interface interaction or stylish user interface. The suitable methodology that is suitable for this kind of system is Evolutionary Prototyping if the performance is a main concern, otherwise the system can opt to Waterfall model where the short duration cycle is not applicable anymore to the system. If the system does not need be completed within three months, the other candidate for the system is Evolutionary Prototyping and Waterfall model. These two methodologies are chosen because of the size and complexity of the system is fairly small and low, respectively. Other methodologies are more suitable with larger and higher complexity systems. The questions are similar to those that presented for short duration ones and it is clearly illustrated in Fig. 3-2. Evolutionary Prototyping is appropriate if the system is properly modularized, having sufficient resources as well as needs high user interface interaction and high performance requirement. If any of these conditions are not met, then it is more appropriate to use Waterfall model. This model can work well for this category because of the stability of the requirement.

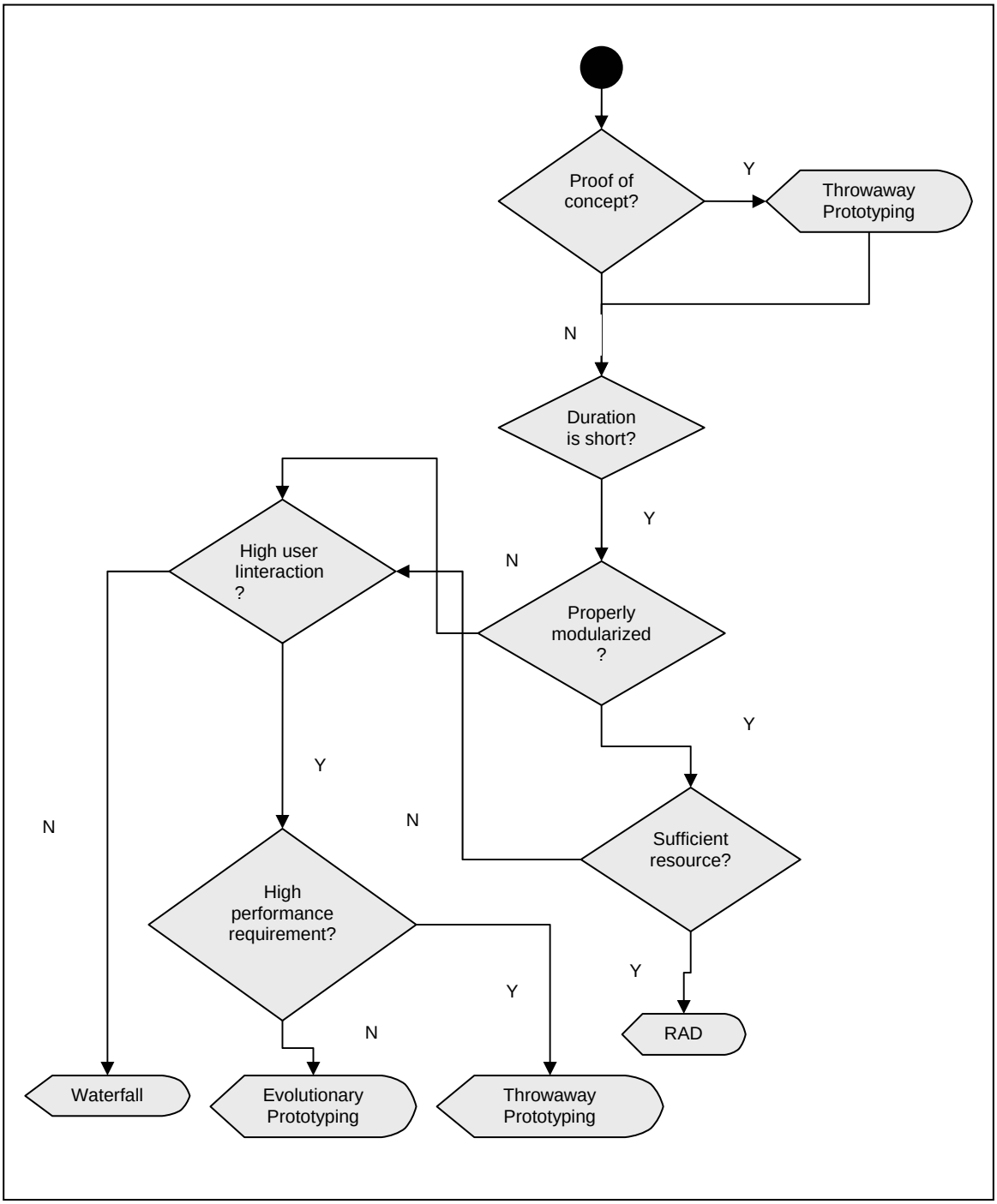

Fig. 3-1: Category 1 with Short Duration

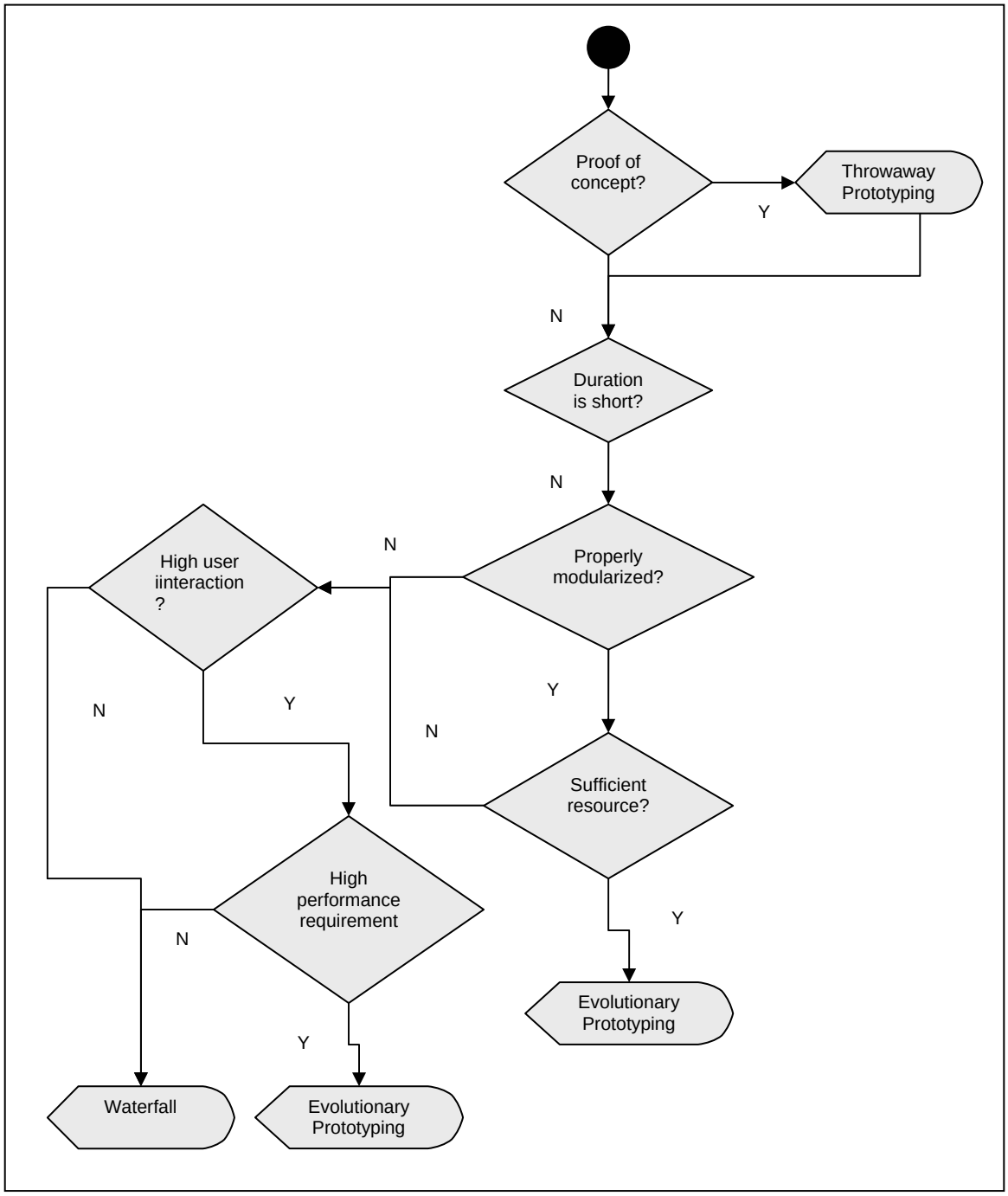

Fig. 3-2: Category 1 without Short Duration

B. Category 2: Rules for system with at least medium size, at least medium complexity and low requirement stability

Fig. 3-3 illustrates the flow chart for Category 2 system, whose size is at least medium, complexity is at least medium and requirement stability is fairly low. The system has priority to assess on the system's agility to determine whether the system has the potential to be implemented using Agile methodologies. The very first question deals with iterative and incremental behavior. This behavior can be predicted in many ways. Some customers even request that the system needs to be built iteratively and incrementally. This could be the case if the system is modular enough, can be delivered independently, and whole system requirements cannot be captured during the early project cycle. If it is not iterative and incremental, then definitely it is not suitable for agile methods. If it is, then the next question asks about the agile attributes: self-organization team, team commitment,

adaptive and emergent, and collaborative and communicative working style. If the system has all of this attributes at least at the medium level, then it can be said that the system is suitable for agile methods. To determine which agile methods to be used, the project needs to answer another set of questions: team size, iteration length, team distribution and criticality. This decision is based on Table 2-2. If the system is not suitable for agile methods, the system displays another set of questions to determine for Rational Unified Process, Spiral, Iterative and Incremental and Evolutionary Prototyping as shown in Fig. 3-3. If the system can be popularly modularized, the customer needs a more defined process and documentation, and the project has sufficient resources, then the project can go for Rational Unified Process. If, however, the project cannot be modularized properly, but performance is a main concern then the project can use Evolutionary Prototyping, otherwise, the system checks whether the project might have any kinds of risk. If the system has risk in user interface but not in other risks like budget, schedule, technology, and quality, then the system may use Iterative and Incremental model. Spiral can be employed if the project has risk in all areas.

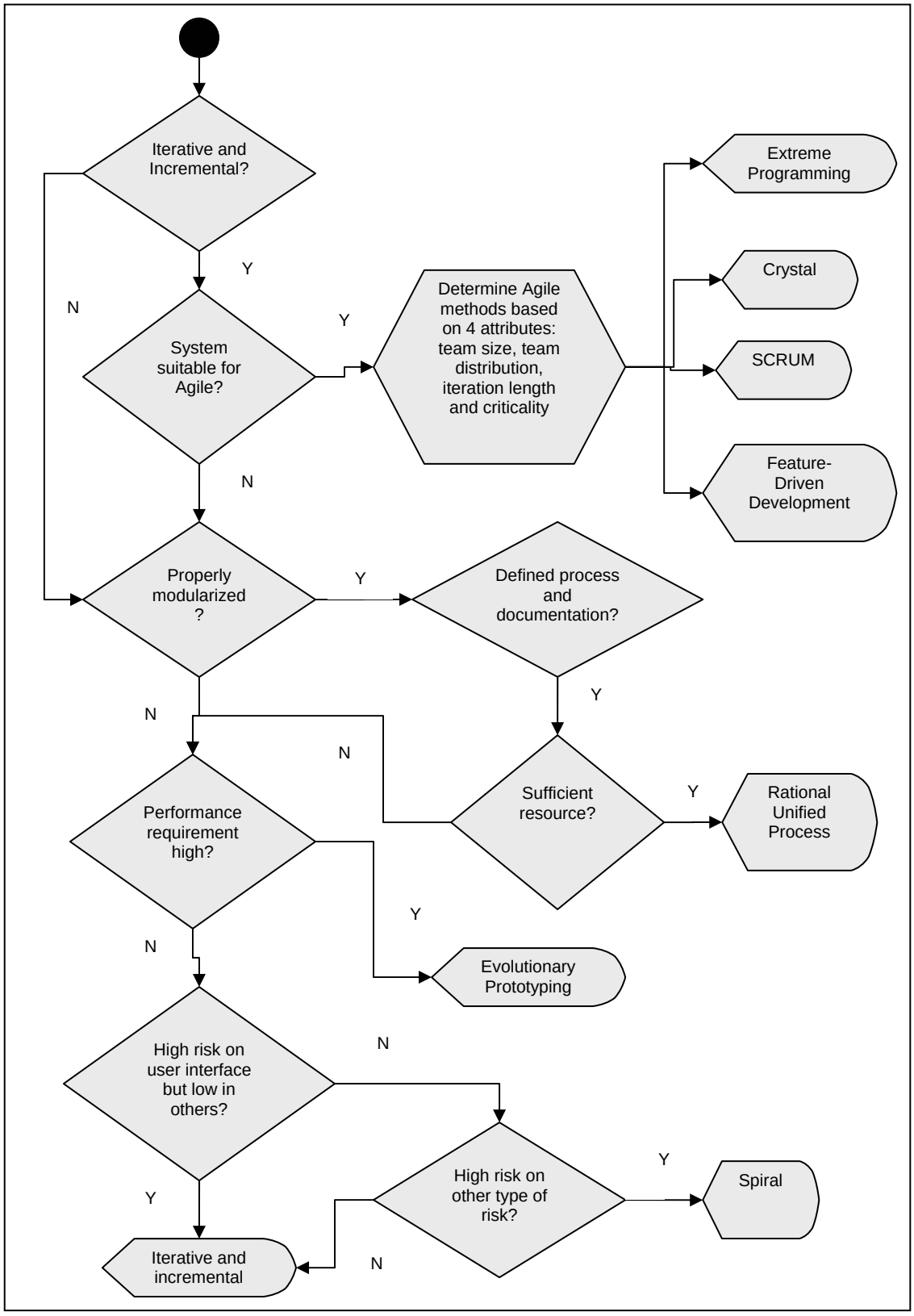

Fig. 3-3: Category 2 Flow Chart

3.2 Rule-Based Implementation

The implementation of the rule engine is adapted from a third part rule engine, Drools 3.0.1 (http://labs.jboss.com/portal/jbossrules). Its framework is shown in Fig. 3-4.

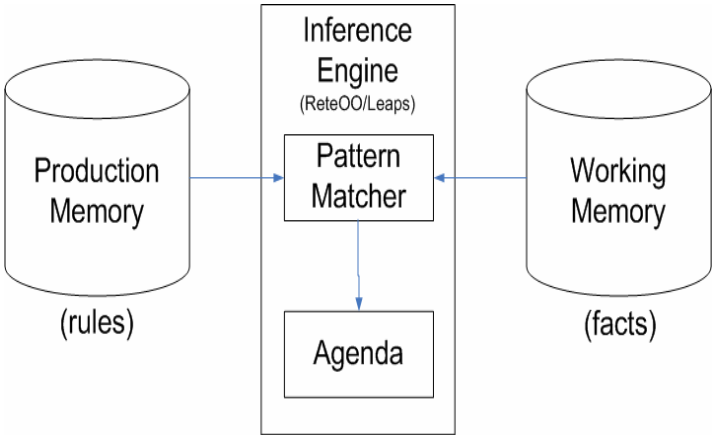

Fig. 3-4: Basic Rete Network [9]

Drools is a rule engine that uses the rule-based approach to implement an expert system. The conflict resolution strategies employed in Drools are Salience and LIFO (last in, first out). Using Salience strategy, a rule can be given priority values by giving a higher number than the other rules. Rules with higher salience will always be preferred. In LIFO, a counter is given by the working memory, however, execution of the rules are arbitrary for those that have the same value. Drools is chosen as the rule engine for SUIT-Method because it is open source, written in Java and well documented. It also uses Rete algorithm and implements forward chaining mechanism. Drools uses Production Rule System as the basis of its framework. It works by storing the rules in Production Memory and asserting facts that the Inference Engine matches against the Working Memory. Facts are asserted into the Working Memory where they may then be modified or retracted. If there are a large number of rules, a conflict may occur, thus, the Agenda manages the execution order of these conflicting rules using Conflict Resolution strategy [9].

3.3 RETE Algorithm

The problem with most rule-based engine is because of the inefficiency of rule engine implementation [4]. Friedman-Hill describes that the percentage of facts that change per unit time in the working memory is fairly small. This efficiency is caused by keeping a list of the rules and continuously cycling through the list, checking each one's left-hand-side against the working memory and executing the right-hand-side of any rules that apply. This may lead to get the same results as the previous iteration.

To overcome this limitation, RETE Algorithm is created [3]. The key ideas of Rete algorithm are [1]:

1. If-then forward chaining rules can be reorganized for efficient pattern matching.
2. A decision tree is created that combines the patterns in all the rules of the knowledge base.
3. Comparisons of variable bindings across patterns must be checked in a relational database table that 'remembers' what partial matches have already been tested when the patterns have been determined that matched with the facts.
4. Variable bindings are saved and reused, rather than recomputed.

RETE is data-driven where it compares the data in the working memory against the conditions of the rules and determines which rules to fire. When matches are found, rules are triggered and fired, producing no assertions or removing assertions. When new assertions are added, the loop through a rule set repeats [1]. The RETE algorithm is implemented by building a network of nodes with each node represents the set of variable bindings that match an assertion or a collection of assertions. According to Berwick, RETE algorithm works by moving assertions through the graph, saving incremental match information as it goes, thus no rules reevaluation is necessary. This will guarantee an optimal approach, as the engine knows which conditions might possibly change for each fact, and only those that must be reevaluated.

4.0 SYSTEM DEVELOPMENT

SUIT-Method proposes to users a suitable software development methodology based on the user inputs, which are the values of identified software project factors that match closest to their project. The system uses evolutionary prototyping to build the system as it involves integration with another third party engine, Drools, which requires an initial experimentation.

The functionality of the tool is described as use cases. Some relevant use cases for this system are Create Project Information, Create Project Size, Create Project Complexity, Create Requirement Stability, Create Category 1 Project Details, Create Category 2 Project Details, View Suitable Methodology, Create Project Execution (its successfulness of using such methodology).

The basic scenario of using this system is described as follows. The user will enter the project details, followed by project size, complexity and requirement stability. Project size will be measured either by analogy or function points, while complexity will be measured either by software complexity model, or using complexity level. Requirements stability measurement can be achieved either by prediction, answering a set of questions or using requirement stability index. From these three factors, the system determines whether the project falls under the first category or the second category. The system then presents the relevant screen based on matched category. A set of questions will be asked depending on the category before determining which methodology is appropriate for the project. The system will display a review of the factors that have been entered. As the user satisfies with the entry, the user proceeds to find the most suitable methodology. The system, using rule-based approach, finds the matching rules and shows the results on screen. At any point of time, if a project has been created and a methodology has been determined, the user can record the successfulness of using such methodology in the project. However, the data collection for this project execution is not guaranteed because of the time gap during the project development and project completion, and the user might not even remember to record the project execution after the project is completed.

Since the system will be a web-based application, a three-tier architecture is used, where the presentation layer and service layer run on a Tomcat 5.0 application server. MySQL database system serves as the persistence layer. The framework of the system is based on the model-view-controller design pattern. The architecture is shown in Fig. 4-1.

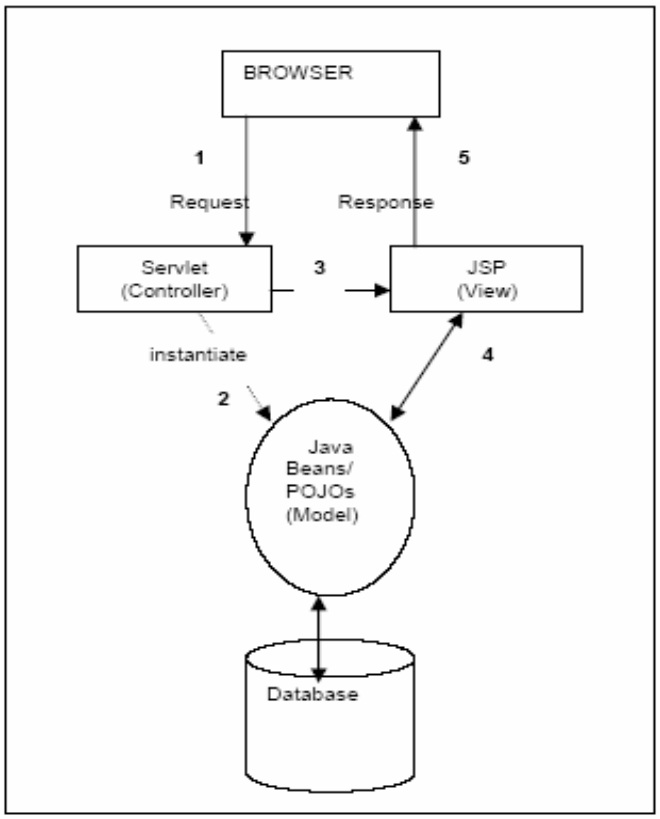

Fig. 4-1: Model View Controller [11]

This architecture uses a hybrid approach to serve dynamic content since it uses java servlets and JSP [11]. JSP is used to generate the presentation layer and servlets act as the controller to process the request by creating necessary beans or database retrieval as well as deciding which JSP page to forward the request to. There is no processing logic inside JSP; it is simply responsible for retrieving any objects or beans that may have been created by the servlet and presents the content to the screen.

This system takes advantage of object-oriented technology where UML is used as the diagram notation. The object model used in this system is shown in Fig. 4-2. Each project may have size, complexity, requirement stability and other factors. Methodology will be part of a project once it can be determined. The project can also have a project execution reference once if it is determined. Each project represents the project attributes and execution which can later be used for any statistical analysis.

Java is chosen as the main language for this system because of its simplicity, object-oriented support, platform independent, safe, multithreaded, high performance, dynamically linked and garbage collected [5]. The IDE used for this system is Eclipse 2.5 M6. Other than compiling java programs and running java classes, this IDE is able to compile the rule as well as generate the RETE node. Some samples of the rules are shown below in Fig. 4-3.

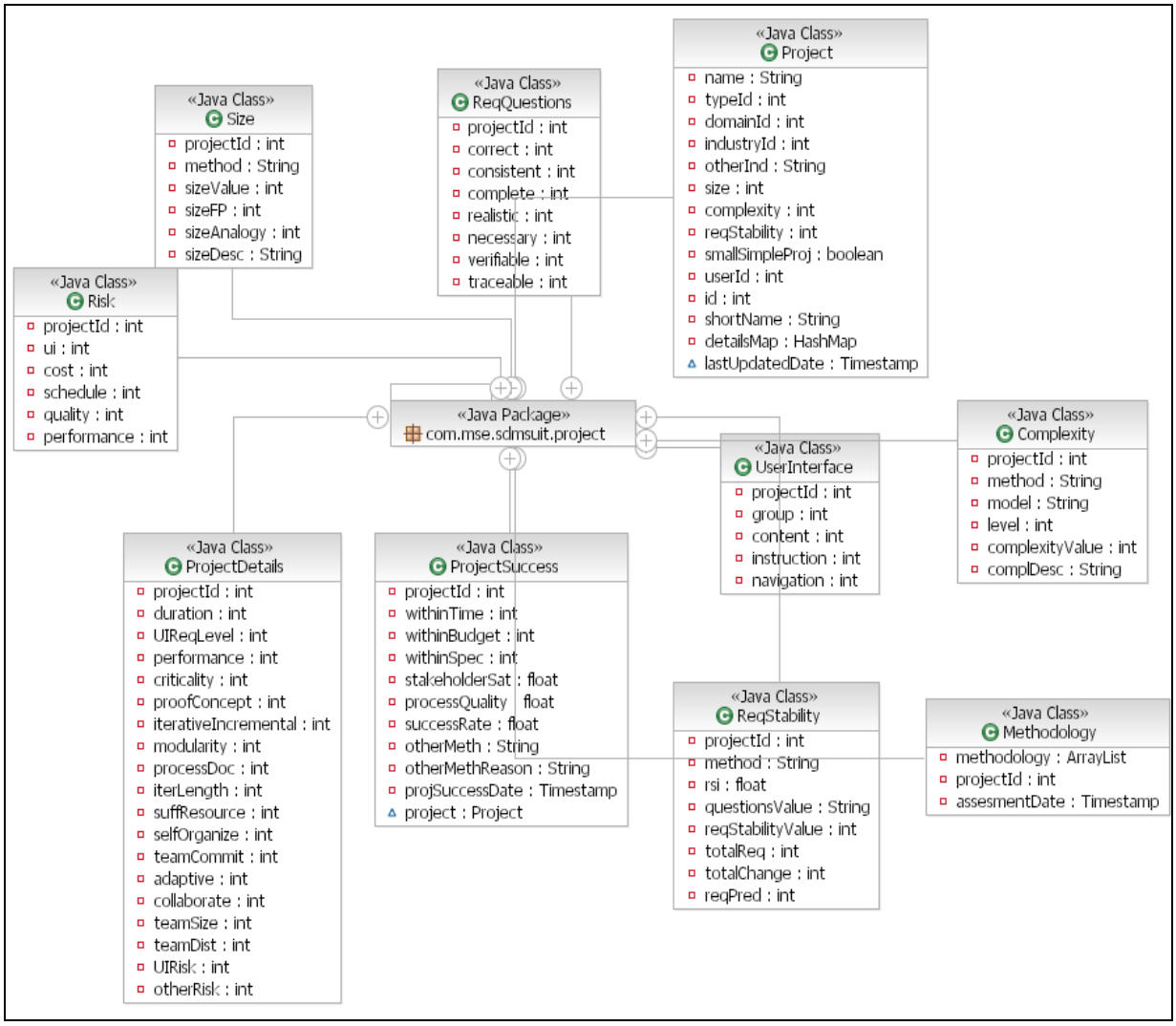

Fig. 4-2: Overview of Suit-Method business object model

```
#created on: May 17, 2007
package com.mse.sdmsuit

#list any import classes here.
import com.mse.sdmsuit.project.ProjectDetails;
import java.util.ArrayList;

#declare any global variables here

rule "#1"
//#1  1
  when
    ProjectDetails(proofConcept == 1);
    retMeth : ArrayList();
  then
    retMeth.add("TP");
end

rule "#2"
//#2    1 3 1 -1  -1  RAD
  when
    ProjectDetails(duration  ==  1,  modularity  ==  3,  suffResource  ==  1,
UIReqLevel == -1, performance == -1);
    retMeth : ArrayList();
  then
    retMeth.add("RAD");
end

rule "#3"
//#3    1 3 0 -1  -1  EP

  when
    ProjectDetails(duration  ==  1,  modularity  ==  3,  suffResource  ==  0,
UIReqLevel == -1, performance == -1);
    retMeth : ArrayList();
  then
    retMeth.add("EP");
end

rule "#4"
//#4    1 1 -1  3 3 TP
  when
    ProjectDetails(duration  ==  1,  modularity  ==  1,  suffResource  ==  -1,
UIReqLevel == 3, performance == 3);
    retMeth : ArrayList();
  then
    retMeth.add("TP");
end

    retMeth.add("TP");
end
```

Fig. 4-3: Sample of rules

5.0 RESULT AND DISCUSSION

The evaluation process involves users to use the system in real environment. The system is published for a month at a web hosting company, AssortedInternet.com. At that period of time, the users were asked to use the system themselves and fill up the feedback form at the end of their usage. Even though the system provides a feedback screen in the website, a softcopy of the questionnaire was emailed to the participants for the full system evaluation. The questionnaire form consists of several questions regarding the participant's background, system's usability, functionality, quality, comment and suggestion. System usability is a measure of an executable software unit for its ease of use, efficiency, effectiveness and satisfaction [15] [18]. System functionality questions deal with the system's main usage while the system quality is measured from the user's experience of using the system. The user

rating on these three groups of questions were measure in 5 point scale: 1-poor, 2-fairly poor, 3- satisfactory, 4 – fairly good and 5 – excellent.

Twenty five participants were selected who are working in software industry with at least two years of working experience and have played major roles in software development. This ensures that the participants have sufficient knowledge and experience to use the system and further giving dependable feedback answer. An e-mail was sent to them to logon to http://www.goldenkidz.com/SUIT-Method/jsp/login.jsp and they were free to use the system and answer the questionnaire at anytime within a week.

Twelve out of 25 people selected responded. 75% of them were holding major roles in software development either as project managers, system analysts or software developers. With regards to working experience, 75% of them already had working experience of more than two years in software industry which is sufficient enough to use SUIT-Method and able to provide reliable feedback on the questionnaire. The strong background of the participants is strengthened with 67% of them had involved in at least two software projects.

Table 5-1: System usability statistical values

Question	Mean	Mode	Median	Standard deviation
The system is easy to use	3.00	3	3	0.74
The system is user friendly	2.83	3	3	0.39
All instruction are made clear and precise	3.00	3	3	0.60
The system requires fewest steps to accomplish its goal	3.67	4	4	0.49
The system is useful	3.50	3	3	0.80
The system helps me to make decision faster	3.17	3	3	0.72
The speed of the system is satisfying	3.75	4	4	0.75
I am satisfied with the system	3.17	3	3	0.39

Table 5-2: System functionality statistical values

Question	Mean	Mode	Median	Standard deviation
Creating the project information (project details, size, complexity, requirements stability)	3.00	3.00	3.00	0.60
Creating the project Category 1 and 2 details	2.92	3.00	3.00	0.29
The proposed methodology is feasible to my organization	2.58	2.00	2.00	0.67
The proposed methodology has enough description and references	2.83	3.00	3.00	0.58

Table 5-3: System quality statistical values

Question	Mean	Mode	Median	Standard deviation
The system is free of errors	3.17	3.00	3.00	0.72
Error messages given are sensible	3.25	3.00	3.00	0.45
All widgets behave as expected	3.25	3.00	3.00	0.45
The software is well documented	2.33	2.00	2.00	0.49

Table 5-1 until 5-3 show the statistical values for system usability, functionality and quality, respectively. Standard deviations for all these attributes are fairly small, thus the data does not really spread out. This indicates that most of the respondents have similar experiences and impression when using the system. Fig. 5-1 shows the overall findings of the system. The figure shows that the system's usability has the score of 3.26, the functionality has the score 2.83, while the quality has the score of 2.98. Since this system is the first prototype showed to the user, the system of course has some flaw or imperfection in terms of its user interface, functionality, user instructions, and quality. The system, therefore, needs to be improved in all areas with some of the enhancements have been included to the delivered system. Some of the enhancements that are taken into consideration are the description on the terms used, the structure on the factors and the scale used to measure each factor.

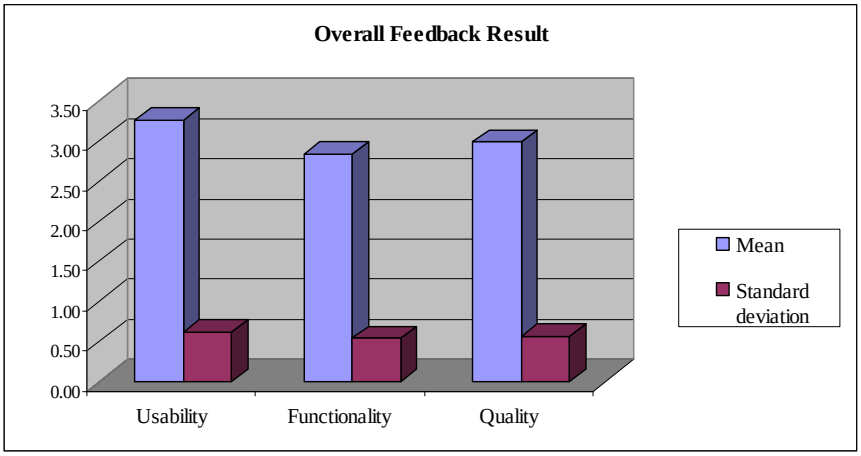

Fig. 5-1: Evaluation Feedback of SUIT-Method

6.0 CONCLUSION

SUIT-Method will be a beneficial tool to software projects at the early of project life cycle. The software methodologies and software factors have been carefully analyzed in order to fulfill the needs of developing a software project so that it is applicable to all level of software practitioners as well as software engineering students. The use of rule-based implementation together with RETE algorithm has assisted the tool development using an expert system approach. Further works may involve on enhancing the system in the area of rules management and refinement and its representation.

REFERENCES

[1] R. Berwick. *Rete, Language and Mind*. http://www.ai.mit.edu/courses/6.0.34b/recitation6.pdf. 2003.

[2] D. Cohen, M. Lindvall, & P. Costa, A New DACS State-of-the-Art Report on Agile Methods. *The DoD Software TechNews*, vol. , no. 1, pp. 13-16.

[3] A. E. Dilmann, Selecting an SDLC Methodology. *PM Solutions Technology White Paper Series*. http://www.pmsolutions.com/pressroom/White%20Papers/sdlc_methodology.pdf. 2003.

[4] L. Forgy, 1982. Rete: A Fast Algorithm for Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligent* . vol. 19, no. 1 (Sept. 1982), 17-37.

[5] E. J. Friedman-Hill, *The Rule Engine for the Java™ Platform*. http://herzberg.ca.sandia.gov/jess/.

[6] E. R. Harold, *Why Java's Hot*. http://www.cafeaulait.org/books/jdr/chapters/01.html. 2000.

[7] B. Hughes, Software *Project Management (3rd ed.).* McGraw-Hill Companies. 2002.

[8] S. Pfleeger, Software Engineering: Theory and Practice. (2nd ed.). New Jersey: Prentice Hall. 2001.

[9] R. S. Pressman, Software Engineering. A Practitioner's Approach (5th ed.). New York: McGraw-Hill. 2001.

[10] M. Proctor, et al. *Drools*. http://labs.jboss.com/file-access/default/members/jbossrules/freezone/docs/3.0.4/html/index.html. 2005.

[11] G. Rudolph,. *Some Guidelines for Deciding Whether to Use a Rules Engine*. http://www.jessrules.com/guidelines.shtml. 2003

[12] G. Seshadri, *Understanding JavaServer Pages Model 2 Architecture: Exploring the MVC Design Pattern.* 1999. http://www.javaworld.com/Understanding JavaServer Pages Model 2 architecture - Java World.htm.

[13] I. Sommerville, *Software Engineering.* England: Addison-Wesley. 1997.

[14] F. Vasquez, Selecting a Software Development Process. *ACM Communication*, pp. 209-217. 1994.

[15] T. Whalen, & B. Schott, Beginners' Strategies In Example Based Expert Systems. *Proceedings of the ACM SIGART International Symposium on Methodologies for Intelligent Systems*,pp. 65-73. 1986.

[16] IEEE Standard Glossary of Software Engineering Terminology. *IEEE Standard* 610.12-1990

[17] D. Green and A. DiCaterino, *A Survey of System Development Process Models*, Center for Technology in Government; http://www.ctg.albany.edu/publications/reports/survey_of_sysdev, 1998.

[18] P. Abrahamsson, O. Salo, J. Raikonen, & J. Warsta. *Agile Software Development Methods: Review and Analysis*. http://www.inf.vtt.fi/pdf/publications/2002/P478.pdf. 2002.

[19] E. Frokjaer, M, Hertzum, & K. Hornbaek. *Measuring Usability: Are Effectiveness, Efficiency, and Satisfaction Really Correlated?* Proceedings of the ACM CHI 2000 Conference on Human Factors in Computing Systems (The Hague, Netherlands, April 1-6 2000), pp. 345-352.

[20] I. Watson. *Applying Case-Based Reasoning: Techniques for Enterprise Systems*. San Francisco: Morgan Kaufmann Publishers. 1997.

BIOGRAPHY

Mastura Hanafiah received her Masters of Software Engineering degree in 2007 from University of Malaya and Bachelor of Science in Computer Science degree in Washington University in St. Louis, Missouri, USA. She has been working with several software companies and has involved in many software development projects. Her research focuses on software development process models.

Zarinah Mohd. Kasirun is a senior lecturer in Software Engineering at the Faculty of Computer Science and Information Technology, University of Malaya. She teaches software engineering modules at both the undergraduate and master levels. Her research areas include Requirements Engineering, Computer-Supported Collaborative Learning, Object-Oriented Analysis and Design.