MODELING THE OPEN SOURCE SOFTWARE DEVELOPMENT PROCESSES USING IDEF3 STANDARD

By

MOHD HAMDI IRWAN BIN HAMZAH

Thesis Submitted in Fulfillment of the Requirements for the Degree of
Master of Computer Science

JANUARY 2013
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>ii</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>vi</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>viii</td>
</tr>
<tr>
<td>LIST OF ABBREVIATIONS</td>
<td>ix</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>x</td>
</tr>
<tr>
<td>ABSTRAK</td>
<td>xii</td>
</tr>
</tbody>
</table>

## CHAPTER 1 INTRODUCTION .................................................. 1
1.1 Overview .......................................................... 1
1.2 Problem Statement .................................................. 2
1.3 Objectives of the Study ............................................ 3
1.4 Thesis Organization ................................................ 4

## CHAPTER 2 LITERATURE REVIEW ........................................... 6
2.1 Introduction ....................................................... 6
2.2 Open Source Software Development ................. 7
2.3 Software Process Model ...................................... 9
2.4 Related Works ..................................................... 12
  2.4.1 Modeling the OSSD Process Using DEMO Standard .......... 12
  2.4.2 Software Process Model Using IDEF3-SPMA ............... 13
LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLES</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Netbean Requirements and Release Process: Comparison for Simplicity</td>
<td>46</td>
</tr>
<tr>
<td>4.2</td>
<td>Netbean Requirements and Release Process: Comparison for Understandability</td>
<td>46</td>
</tr>
<tr>
<td>4.3</td>
<td>Netbean Requirements and Release Process: Comparison of Model Representation</td>
<td>47</td>
</tr>
<tr>
<td>4.4</td>
<td>Comparison for Case Study #1</td>
<td>50</td>
</tr>
<tr>
<td>4.5</td>
<td>Comparison for Case Study #2</td>
<td>53</td>
</tr>
<tr>
<td>4.6</td>
<td>Comparison for Case Study #3</td>
<td>56</td>
</tr>
<tr>
<td>FIGURES</td>
<td>TITLE</td>
<td>PAGE</td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>2.1</td>
<td>The DEMO transaction pattern.</td>
<td>18</td>
</tr>
<tr>
<td>2.2</td>
<td>A Process-Structure-Diagram (PSD)</td>
<td>19</td>
</tr>
<tr>
<td>2.3</td>
<td>An Actor-Transaction-Diagram (ATD)</td>
<td>19</td>
</tr>
<tr>
<td>2.4</td>
<td>Example of VRPML graph</td>
<td>20</td>
</tr>
<tr>
<td>3.1</td>
<td>Example of IDFE3 process description diagram</td>
<td>29</td>
</tr>
<tr>
<td>3.2</td>
<td>Example of object state transition diagram</td>
<td>30</td>
</tr>
<tr>
<td>3.3</td>
<td>Research activities for modeling the OSSD using IDEF3</td>
<td>31</td>
</tr>
<tr>
<td>4.1</td>
<td>DEMO Process-Structure-Diagram for the Netbean Requirements and Release process</td>
<td>45</td>
</tr>
<tr>
<td>4.2</td>
<td>IDEF3 Process-Centered-Diagram for Netbean Requirements and Release process</td>
<td>45</td>
</tr>
<tr>
<td>4.3</td>
<td>Process-Structure-Diagrams for the Purchasing Product Process</td>
<td>49</td>
</tr>
<tr>
<td>4.4</td>
<td>IDEF3 Process-Centered-Diagrams for Purchasing Product Process</td>
<td>49</td>
</tr>
<tr>
<td>4.5</td>
<td>Process-Structure-Diagrams for the Hospital Treatment Process</td>
<td>52</td>
</tr>
<tr>
<td>4.6</td>
<td>IDEF3 Process-Centered-Diagrams for Hospital Treatment Process</td>
<td>52</td>
</tr>
<tr>
<td>4.7</td>
<td>Process-Structure-Diagrams for the Pizza Ordering Process</td>
<td>55</td>
</tr>
<tr>
<td>4.8</td>
<td>IDEF3 Process-Centered-Diagrams for Pizza Ordering Process</td>
<td>55</td>
</tr>
<tr>
<td>4.9</td>
<td>The Comparison by Case versus Attributes</td>
<td>57</td>
</tr>
<tr>
<td>4.10</td>
<td>Model Comparisons between IDEF3 and DEMO for Notation Representation</td>
<td>58</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>CVS</td>
<td>Concurrent Versions System</td>
<td></td>
</tr>
<tr>
<td>DEMO</td>
<td>Design and Engineering Methodology for Organizations</td>
<td></td>
</tr>
<tr>
<td>IDEF</td>
<td>Integration Definition</td>
<td></td>
</tr>
<tr>
<td>IDEF3</td>
<td>Integration Definition Method for Process Description Capture</td>
<td></td>
</tr>
<tr>
<td>ITU-T</td>
<td>ITU Telecommunication Standardization Sector</td>
<td></td>
</tr>
<tr>
<td>OSS</td>
<td>Open Source Software</td>
<td></td>
</tr>
<tr>
<td>OSSD</td>
<td>Open Source Software Development</td>
<td></td>
</tr>
<tr>
<td>QA</td>
<td>Quality Assurance</td>
<td></td>
</tr>
<tr>
<td>SDL</td>
<td>Specification and Description Language</td>
<td></td>
</tr>
<tr>
<td>UOB</td>
<td>Unit of Behavior</td>
<td></td>
</tr>
<tr>
<td>VRPML</td>
<td>Virtual Reality Process Modeling Language</td>
<td></td>
</tr>
<tr>
<td>Z.100</td>
<td>International directory of online TV-channels and associated video media</td>
<td></td>
</tr>
</tbody>
</table>
ABSTRACT

Modeling the Open Source Software Development Processes using IDEF3 Standard

By

Mohd Hamdi Irwan Hamzah

January 2013

Open Source Software Development (OSSD) process model closely resembles the modeling process of conventional software development process model where the most common element in the development process of the project will be identified. Lately, significant demands for process modeling languages have also been raised because of the complexities inherit by the previous OSSD process model methods. It is also been noted that the available languages are not satisfactory and this prompted a search for language that can meet a higher level of abstraction.

In this research, the propose technique is the Integration Definition (IDEF) method for Process Description Capture (IDEF3) which might offer remarkable alternative to develop OSSD process models. IDEF3 exhibits two unique features within the context of OSSD process modeling. IDEF3 supports both process-centered and object-centered knowledge acquisition strategies enabling users to capture assertions about real-world processes and events in ways paralleling common forms of human expression.
The research constructed IDEF3 model to describe the Net Beans Requirements and Release process. This research also investigated the simplicity and the understandability of using IDEF3 to construct OSSD process models by means of a case study (Net Beans Requirements and Release process). The model constructed using IDEF3 standard is compared to the model produced by using DEMO standard to show the result in term of simplicity and understandability. This quality attributes are extracted from the numbers of constructs produced and the notations that each model used. Comparison and verification result shows that modeling the OSSD using IDEF3 standard always produced less numbers of constructs and easy to understand notation compared to the OSSD model produced by DEMO standard. This indicated that the IDEF3 standard conforms to the simplicity and understandability quality attributes.

It is concluded that this research have successfully applied the IDEF3 standard in modeling the OSSD processes. This research also contributed in proving that the IDEF3 standard does support simplicity and understandability.
ABSTRAK

Permodelan Proses Pembangunan Perisian Sumber Terbuka menggunakan IDEF3 Standard

Oleh

Mohd Hamdi irwan Hamzah

Januari 2013

Proses Permodelan Pembangunan Perisian Sumber Terbuka (OSSD) hampir menyerupai proses pemodelan pembangunan perisian konvensional di mana elemen yang paling biasa di gunakan dalam proses pembangunan projek akan dikenalpasti. Kebelakangan ini permintaan untuk bahasa proses pemodelan yang baharu telah meningkat agak tinggi kerana kerumitan yang diwarisi oleh kaedah proses permodelan OSSD yang terdahulu. Selain itu bahasa yang sedia ada juga tidak begitu memuaskan dan ini mendorong kepada pencarian bahasa yang boleh memenuhi tahap abstrak yang lebih tinggi.

Dalam kajian ini, teknik yang dicadangkan adalah kaedah Integration Definition (IDEF) method for Process Description Capture (IDEF3) yang menawarkan alternatif yang lebih baik untuk pembangunan proses permodelan OSSD. IDEF3 juga menawarkan dua ciri-ciri unik dalam konteks proses pemodelan OSSD dimana IDEF3 menyokong strategi berorientasikan proses dan strategi berorientasikna objek yang membolehkan
pengguna untuk merekodkan proses-proses dalam kehidupan dunia sebenar dan peristiwa yang berlaku secara selari dengan ungkapan seharian.


Sebagai kesimpulan kajian ini telah berjaya menggunakan standard IDEF3 dalam memodelkan proses OSSD. Kajian ini juga telah menyumbang dengan membuktikan bahawa standard IDEF3 menyokong atribut ringkas dan kefahaman.
CHAPTER 1

INTRODUCTION

1.1 Overview

Open Source Software (OSS) has been popular and in a very high demand lately (Lonchamp, 2005). Open source software (OSS) can be defined as a free software that you can modify and used its source code and claims as your own freely. The ease of using the OSS has really brought it to the center of attention in research. The development processes of the OSS have also been identified to be one of the crucial things that can affect the effectiveness of the software. So it is vital to capture the development process of the OSS in order to predict the outcome of the development processes. Meanwhile the software process model can be defined as a model describing the overall flow and sequence of software project life-cycle activities, including project planning, tracking, requirements management, software construction and release (Huysmans et al., 2010). In order to capture and model the OSSD process, there are numbers of standard exist that can be used for this purposes and integration Definition (IDEF) method for Process Description Capture (IDEF3) might offer remarkable alternative to develop OSSD process models. IDEF3 exhibits two unique features within the context of OSSD process modeling. IDEF3 supports both process-centered and object-centered knowledge acquisition strategies enabling users to capture assertions about real-world processes and events in ways paralleling common forms of human expression. Moreover, the models created using the IDEF3 standard
are capable of providing a sufficiently high-quality model. The importance of using IDEF3 in modeling the OSSD process is that IDEF support the simplicity and understandability which can help the model user easily used the modeling standard with little training.

1.2 Problem Statement

The Open Source Software Development (OSSD) process model closely resembles the modeling process of conventional software development process model where the most common element in the development process of the project will be identified by the OSSD process model (Huysmans et al., 2010). Lately there have been significant demands for process modeling standard which have been raised because of the complexities characterized by this OSSD process model from the previous methods. The most significant issue is the available OSSD process model produced a lot of constructs, hence increasing the model complexities (Huysman et al., 2010). These means that the previous model are not supporting the quality attribute for simplicity which can cause for not having well documented process and no proper advertised development model (Jensen and Scacchi, 2006) (Lonchamp, 2005).

According to previous studies, OSSD process is documented by using semi-structured hyperlinked models, formal computational process models, process flow graphs, and use cases (Huysmans et al., 2010). There is also no single OSSD process that is valid for all OSS project because of the OSS processes can be eccentric to the specific
project being developed (Lonchamp, 2005). This is because dealing with open
environment can be very difficult to understand. The existing modeling language such
as DEMO has been identified to have a drawback concerning the lack of support on
understandability due to specific notation standard used (Huysman et al., 2010). The
difficulty to use the modeling standard can prevent the model user to successfully
model the OSSD process.

1.3 Objectives of the Study

The study is to propose IDEF3 standard in modeling the Open Source Software
Development process. This study expands the use of IDEF3 standard in modeling the
OSSD process by addressing the following objectives:

1. To propose and apply IDEF3 in an OSSD higher level language abstraction.

2. To compare and prove that IDEF3 standard support the simplicity and
understandability attributes for OSSD process modeling.

In order to achieve the objective of the study, this study consists of single OSSD
process within specific OSS project for the case study. The study for the OSS will be
focusing on the use of Net Beans Requirement and Release process. The IDEF3 model is
constructed to describe the Net Beans Requirement and Release process. After that the
model produced using IDEF standard is compared to DEMO based on simplicity and
understandability attributes.
The outcome of this study is that it should be able to apply IDEF3 standard in modeling the OSSD process. Apart from that, the study also should prove that IDEF3 standard support the simplicity and understandability for OSSD process modeling.

1.4 Thesis Organization

This thesis is divided into five chapters. The first chapter described the abstract, introduction of the research, problem statement and objective of the study.

The second chapter will contain the literature review of this study. In this literature the study will focus on the software process modeling and concentrate about the open source software development process. The literature will cover several topics that are related to modeling the software process in order to give an initial idea and knowledge to readers for this research area. This chapter also elaborates on the technique in modeling the software process. This chapter will try to identify several techniques that were used in modeling the software process including the proposed technique for this research which is IDEF3. Those identified technique were quite important because it reflected the techniques that is available and can be used in software process modeling.

Chapter three will discuss on the methodology used in this study and chapter four will show the results that have been produced throughout the study on this research. This result will be presented and analyze so that it can be discuss later.
The last chapter is the conclusion and future research of this study. The research done will be concluded and some of the contributions which are derived from the study will be highlighted. Future researches to enhance this study also are pointed out in this chapter.
CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Open Source Software (OSS) lately has become one of the most preferably software used among the developer for constructing their software system. Open Source Software (OSS) is software whose source code may be freely modified and redistributed with few restrictions (Lonchamp, 2005). Lonchamp (2005) also stated that OSS is produced by loosely organized, ad-hoc communities consisting of contributors from all over the world who seldom if ever meet face-to-face, and who share a strong sense of commitment. The emergences of OSS are also because of the certain value it has to offer which really have led to the study on the development of OSS. The open source software development (OSSD) processes model promotes the parallel and repeated development techniques where user can participate freely, joining vast development communities and effective user testing (Khanjani and Sulaiman, 2011). However to model the OSSD process is not an easy task because OSS project is not well documented and their development model is not properly advertised (Jensen and Scacchi, 2006) (Lonchamp, 2005).

There is also no single OSSD process that occurs valid for all OSS project because of the OSS processes can be eccentric to the specific project being developed (Lonchamp, 2005). According to the previous studies, OSSD process will be documented by using semi-structured hyperlinked models, formal computational
process models, process flow graphs, and use cases (Huysmans et al., 2010). Besides that, significant demands for process modeling languages have also been raised because of the complexities characterized by the OSSD process model from the previous methods. It is also been noted that the available languages are not satisfying modeling approach of a higher level language abstraction.

2.2 Open Source Software Development

Open Source Software Development (OSSD) models is a collaborative bug driven development where high diversity capabilities and qualification of globally distributed participants volunteers themselves to take part in this development (Dietz et al., 2005). These communities will deliver frequently new software release where the development activities are executed in parallel and the interaction occurs exclusively through web-based technologies.

The OSSD model plans also are not the same as the conventional development plan. The OSSD follows an iterative and parallel development approach with a user driven development direction, no central management, free participation, large development communities and effective user testing where else conventional methods have defined teams and requirements. However the OSSD development methodology usually are not well documented, applied informal testing and non-formal Quality Assurance methods (Aberdour, 2007). Besides that, under the OSSD model quality
management the projects also do not collate empirical evidence regarding quality and only few measurable quality goals are defined (Aberdour, 2007).

Nonetheless Linux and the Apache Web Server are the successful products from the OSSD model deliverable that seem to be in high quality. This means that OSSD model can also produce good quality product which have led developer to explore the open source software environment. Open source software as most people would agree, has as its underpinning certain legal and pragmatic arrangements that ensure the source code for an open source software development will be generally available. There is a central person or body that selects some subset of the developed code for the official release in open sources software development. This person in charge will release the new software and distributed the release so that it is widely available and accessible.

These basic arrangements to ensure freely available source code have led to a development process that is radically different, according to OSS proponents, from the usual, industrial style of development. The main differences usually mentioned are:

- OSS systems are built by potentially large numbers where hundreds or even thousands of volunteers.
- Work is not assigned; people undertake the work they choose to undertake.
- There is no explicit system-level design, or even detailed design (Vixie, 1999).
- There is no project plan, schedule, or list of deliverables.
The differences in the development of OSS have made it clear that this development occurs in a distributed geographically manner. The developer who hardly meet face to face and work in random locations communicate through email and bulletin boards are the one responsible for this development of OSS. This distributed remotely location of the developer who participates in OSS development also have led to certain problem compared to the traditional software development. The coordination of the OSS development is lacks of system-level design, plan, defined process and schedules.

2.3 Software Process Model

Software process model is difference from software life cycle in a way that software process models frequently embody a network sequence of activities, objects, transformation and event that represent strategies for accomplishing software evolution (Kling and Scacchi, 1982). This software process model can be used to develop a software life cycle that is more precise and formalized descriptions. This can be realized through the use of the notation, syntax, or semantic that is provided in software process model which is appropriate for computational processing.

Apart from that, software process model are also viewed as multiple interconnected task chains (Kling and Scacchi, 1982) (Garg and Scacchi, 1989). The task chain is representation of non-linear sequence of actions that transform and structure the available computational objects or resources into intermediate or finished products. This non-linear sequence means that the actions in sequence are iterative,
non-deterministic, accommodate multiple or parallel alternatives and as well as partially ordered to account for incremental progress. Task actions in turn can be viewed a non-linear sequences of primitive actions which denote atomic units of computing work, such as a user's selection of a command or menu entry using a mouse or keyboard. Winograd and Flores (1986) have labeled the “structured discourses of work” to represent the units of supportive work between human computers while Bolcer and Taylor (1998) stated the “workflow” to embody the task chain.

The descriptive or prescriptive action sequences are characterized by the employment of the task chains. The plan structure for what action should be accomplish and in which order are idealized through the prescriptive task chains. The examples of a task chain for the object-oriented software design activities are showed such as below:

- Develop an informal narrative specification of the system.
- Identify the objects and their attributes.
- Identify the operations on the objects.
- Identify the interfaces between objects, attributes, or operations.
- Implement the operations.

The sequences of actions above sometime require multiple iterations and non-procedural primitive action invocations in the course of incrementally progressing toward an object-oriented software design. Apart from that, the overall production of network and web are the result of task chain joining and splitting process (Kling and Scacchi, 1982). This production of web and network converts raw computational,
cognitive, and other organizational resources into assembled, integrated and usable software systems which also represent the organizational production system. The production lattice therefore structures how a software system is developed, used, and maintained. However, prescriptive task chains and actions cannot be formally guaranteed to anticipate all possible circumstances or idiosyncratic foul-ups that can emerge in the real world of software development (Bendifallah and Scacchi, 1989) (Mi and Scacchi, 1990).

Apart from that, only partially or fairly accurate description of software development will be in some way realized by any of the software production web. Moreover when a planned task chain is insufficient or breaks down, the unanticipated task called the articulation work will be performed. It is work that represents an open-ended non-deterministic sequence of actions taken to restore progress on the disarticulated task chain, or else to shift the flow of productive work onto some other task chain (Bendifallah and Scacchi, 1987) (Grinter, 1996) (Mi and Scacchi, 1990) (Mi and Scacchi, 1996) (Scacchi and Mi, 1997). Thus, descriptive task chains are employed to characterize the observed course of events and situations that emerge when people try to follow a planned task sequence.

Articulation work in the context of software evolution includes actions people take that entail either their accommodation to the contingent or anomalous behavior of a software system, or negotiation with others who may be able to affect a system modification or otherwise alter current circumstances (Bendifallah and Scacchi, 1987) (Grinter, 1996) (Mi and Scacchi, 1990) (Mi and Scacchi, 1996) (Scacchi and Mi, 1997).
This notion of articulation work has also been referred to as software process
dynamism.

2.4 Related Works

This study is originated from several of past researchers that work on the area
of modeling the software process. This study focused on modeling the OSSD processes
using IDEF3 standard. The related works regarding this study are described on the
following subtopic.

2.4.1 Modeling the OSSD Process Using DEMO standard

Huysmans et al. (2010) have conducted a case study on using the Design and
Engineering Methodology for Organizations (DEMO) for modeling open source
software development processes. The approach for this study was that first they do the
investigation on the feasibility of using DEMO to develop OSSD process models by
means of a case study. Netbeans Requirements and Release process was chosen in this
and in order to describe the Net Beans Requirements and Release process the DEMO
models were constructed. Moreover the OSSD process model produced by DEMO was
also evaluated using a quality framework for conceptual modeling. In their result they
have been able to prove that the DEMO methodology can be successfully used to
model OSSD processes to obtain abstract and high-quality OSSD process models by
means of the case study. Even though their study has produced something of value but
the DEMO have a number of possible shortcomings. The most noticeable disadvantages
of using the DEMO in modeling the OSSD processes were the understandability and implementability of the DEMO model. The DEMO model is quite hard to understand by the first timer and those who aren’t familiar with the specific notation in the DEMO. Moreover, there are no implementation-related details provide by the DEMO models thus making it also hard to learn. Hence, it can be concluded that for the purposed of communicating and re-enacting OSSD process models to other parties it may be less well suited by this implementation of DEMO.

2.4.2 Software Process Model Using IDEF3-SPMA

In another study, Atan et al. (2007) conducted a research on the integration of measurement in the modeling software process. The objective of the study was to express the significant of applying measurement in modeling software process so that it can help reduced the effort and rework in developing large software. The method used in their study focused on the IDEF3 standard notation as its approach to design software process models, IDEF3-SPMA language constructs as its medium for automatic metric calculation and measurement metric defined specifically to fit the research scope. The approach that have been chosen and used to specify the measurement metric defined in this study was the attribute grammar. In order to achieve their goals in measuring the software process model a tool was developed named Software Process Measurement Application (SPMA). The result of this study indicated that they have been able to help the designers by the automatic collection of software process
design measurement in means of preliminary evaluation of their designs based on the verification of system. Moreover from the research, they also successfully come out with a tool through extension of IDEF3 Standard, called SPMA that executes as a static analyzer to IDEF3-SPMA language, which then summarizes and lists process model designs’ measurement metric attributes.

2.4.3 Netbean Requirement and Release Process Modeling

Apart from that, Scacchi et al. (2005) also conduct a case study on the Netbean.org project to describe how a variety of modeling perspectives and techniques are used to elicit, analyze, and validate software requirements processes found in OSSD projects. The approach used by them was by applying multi-modal modeling onto the observation processes, artifacts, and other evidence composed as an ethnographic hypermedia. A set of informal and formal models of the requirements processes from their observation, codification, and documentation is needed and this can be provided by the ethnographic hypermedia. In this study, in order to proceed with their approaches they have to provide the foundational basis and this were done by undergo the process of reviewing related research. After that, they try to describe and provide examples of the modeling models they use to elicit and analyze the processes under study. At last, they examine what each modeling model is good for, and what kind of analysis and reasoning it supports. This study of theirs have resulted in indicating that there are no single model of process description sufficiently compatible to others which
means there is no best process description scheme. They also validate that in the progression of developing the multi-mode requirement process models, the incremental and progressive elicitation, analysis, and validation occurs. Moreover they have also successfully proved that multi-mode process models are well-suited for discovery and understanding of complex software processes found in OSSD projects. This goes on with the potential of multi-mode process modeling to be appropriate for the discovery and modeling of the software product requirements. Last, they observed that the software product requirements in OSSD projects are continually emerging and evolving.

In this study, the proposed modeling standard is IDEF3 where it have been identified to provide interesting alternative that is effective in modeling the process of OSSD. IDEF3 is a modeling standard that was design to aid the documentation and analyze the process of the existing or the proposed system (Mayer et al., 1995). These IDEF3 standard have provide a proven guidelines which come with the information capture languages that can aid user to capture and organize process information for multiple downstream uses (Mayer et al., 1995). The use of IDEF3 standard can be manipulated to help in various areas such as to enhance the productivity of business system analysis, facilitate design data life cycle management, support the project management process, facilitate the system requirements definition process and support coordinated activity and integration of effort (Mayer et al., 1995). More important is that IDEF3 is easy to learn and use by individuals having little or no training in structured techniques while promoting consistent, reliable practice. These criterions
that somehow differentiate the IDEF3 model with the DEMO model promotes by Huysman et al. (2010).

2.5 Process Modeling Techniques

There are several modeling techniques exist in modeling the OSSD process. These modeling techniques provide different methods in capturing the OSSD process. The modeling techniques can be chosen based on their suitability to the user needs and user should know which techniques do provide the needed requirement. Several techniques discussed in this research are such as DEMO Modeling Technique, VRPML Modeling Technique and SDL based Approach Modeling Technique. Each of these techniques was explained in the following sub-topic below:

2.5.1 DEMO Modeling Technique

Design Engineering Methodology for Organizations (DEMO) is a method for organization engineering, an emerging discipline concerning the design and implementation of organizations (Dietz et al., 2005).

DEMO usually interested in identifying a human actor that is responsible for particular activities of certain process. Generally DEMO performs two kinds of act which were carried out by a subject or individual which were called organization. This first act would be the production act where this act would be the organization goal and
the subject would be considered has accomplishing the goal of the organization if they perform the production act. This production act (P-Act) can be in two form which the material or immaterial. The delivery of a product, services and information to the environment of an organization can resemble the p-act (Huysman et al., 2010). Beside that the other act which called the coordination acts (C-Acts) means that the subjects fulfill and agree to certain commitment. This two kind of act when perform by the subject will trigger the initiating and coordinating the execution of production acts. To differentiate and identify whose responsible for performing the acts an actor role is introduced.

A transaction is a universal pattern where it represents the successfully completed P-act performance which this involved the coordination of the transaction axiom states (Huysman et al., 2010). The basic transaction pattern is shown in Figure 2.1. This transaction contains three phases where the first phases called the order phase involve the negotiation of actors about the P-fact. This P-fact is the subject of the transaction and brought into existence by performing the P-acts. Then the P-fact would be produced in the execution phase after negotiation came into an agreement. In the result phase, the actors can negotiate and discuss about the result of the transaction (Huysman et al., 2010). These phases would then be divided into four coordination acts which label as the request, promise, state and accept. Beside that the production act also involve in this phase known as execute. Apart from that, the actors are divided into two types of actors which called the initiator (initiate the act) and the executor (the
authorized person to execute a single transaction) (Dietz et al., 2005). Figure 2.1 below would be presenting the notation used in DEMO explained earlier.

**Figure 2.1** The DEMO transaction pattern.

DEMO can be presented in various diagrams but in this study two type of diagram showed in Figure 2.2 and Figure 2.3 that reflect this research would be presented and discuss. These diagrams would be the Process-Structure-Diagram (PSD) and Actor-Transaction-Diagram (ATD). The PSD would be about the details interaction between transaction while the ATD shows the overview of the actor and transaction within the scope of the enterprise (Huysman et al., 2010). Below are the examples of PSD and ATD diagrams.
2.5.2 VRPML Modeling Technique

Virtual Reality Process Modeling Language (VRPML) is a modeling language which specifically designs to capture the software process in visual language domain specific (Zamli and Lee, 2002). VRPML model the software process and possess almost the same basic characteristic of other software process modeling language. What make it different from the rest of software process modeling language is that VRPML considers virtual environments as a major constituent, manipulatable as part of the construction of the process model (Zamli and Lee, 2002). This feature can be seen from the structures of the language itself and VRPML also supports dynamic allocation of resources through its enactment model (Zamli, 2003).
Apart from that, VRPML also can dynamically allocate and tailored the resources such as tools, artifacts, software engineer and many more to suite the explicit project from a generic model. The notation used in VRPML were presented by interconnecting nodes from top to bottom using an arcs that carry run-time control-flow signals to model the software process (Zamli and Lee, 2001). The complete description of the syntax and semantics of VRPML can be found in (Zamli, 2003) and (Zamli and Lee, 2003). The example of capturing the software process is illustrated in Figure 2.4 where the benchmark process is used to be model by using VRPML.

![Figure 2.4 Example of VRPML graph.](image)

Figure 2.4 shows that process step abstraction were used in VRPML which this technique resemble the most atomic representation of software process such as the real activity perform by the software engineer in developing their software. The nodes
in Figure 2.4 represent the activities where the small ovals nodes with stick figures are called the activity nodes.

As shown in Figure 2.4, VRPML supports many different kinds of activity nodes. Below are the representations of the activities nodes:

- General-purpose activity nodes - shown as individual small ovals with stick figures.
- Multi-instance activity nodes shown as overlapping small ovals with stick figures.
- Meeting activity node shown as small and shaded overlapping ovals with stick figures.

The meeting activity nodes and multi-instance nodes have associated depths. These associated depths actually represent the real number of engineers that are involved in the development process. Moreover, in multi-instance activity the associated depths also can represent the number of identical activities (Zamli and Lee, 2002). Apart from that, the dotted line ovals called the macro node can assemble and grouped together a set of VRPML nodes which then can increase the graph readability.

The arrival of a necessary control-flow signal can be used to control the firing of activity nodes and this control-flow signal is generated from the transition itself. Figure 2.4 show the representation of the transition as a small white circle with a capital letter attached to an activity node or decomposable transitions shown as small black circles with a capital letter attached to an activity node. However, the initial control-flow signal must always be generated from a start node shown as a white circle enclosing a
black circle (Zamli and Lee, 2002). A stop node shown as a white circle enclosing another white circle does not generate any control-flow signal. Furthermore using the combination of a language element called merger and replicator nodes which in Figure 2.4 they are shown as trapezoidal boxes with arrows, the activity nodes can also be enacted in parallel.

2.5.3 SDL-based Approach Modeling Technique

Specification and Description Language (SDL) is a standard language for identifying and describing system (Ellsberger et al., 1997). ITU-T and Z.100 are the organizations who recommend SDL to be develops and standardize. This has been the starting point for SDL to be used as standard language throughout the telecommunication community. Since then SDL has been long used as a modeling language that capture the software process during the development and maintenance. The graphical and formal representation provide by SDL has made it successful in satisfying the requirement in capturing the software process activities (Podnar et al., 2000).

By means of the formal notation provides by SDL, it has enable model users to analyze the process and the graphical notation that can be used in SDL also provide better understanding and easily documented (Podnar et al., 2000). Moreover the model user also will find that it’s quite pleasing to used SDL standard because of the
In SDL software process entities representation, Grun and Weber (1992) have classified into three different entities which is the organizational, technical and communicational entities. The first entity which is the organization entity cover the coordination and the interaction during the software process initiate by the process participant. The organization entity represents also their organizational structure by using roles and team. The second entity called the technical entity responsible for the transformation and creation of the process artifacts. These entities cover the specification, documentation and source code (Grun and Weber, 1992). The technical entities that were included are the activities, document and tools which support the interest of software process modeling. The last entities are the communicational entities where it concern with the information changing between process participants.

The representation of SDL entities are originally based on the instance of object. The process instance will perform an action of changing state or basically producing output signal after receiving the input signal. The software representation descriptions for software process in SDL approach is describe such as below:

- A role is modeled by SDL process type which describes the flow of activities performed by particular process participant.
- A team is modeled as a block that consists of process instance.
• An activity is performed by a process participant. An input signal start the activity and output signal stop it.

• A document is modeled by SDL process types which describe the set of document states.

• A tool is also specified by SDL process type where the tool is modeled as a passive process entity with two states such as available and not available.

2.6 Software Measurement

Software measurement acquires the information related to the development of the models, theories, planning, calculating and method of using the techniques. The important of measurement cover the method for production planning, monitoring and control. This process is significant because it can avoid the risk where the software projects may become excessive and led to uncontrolled software production which can easily deviated from the industrial control. The risk occurs can also lead to certain critical damage that can harm both software producer and users. The damages done to the software producers include the schedule slippage and high cost maintenance where else the user will be producing the poor quality products, high prices and late product delivery.
REFERENCES


Vixie, P. “Software Engineering in Open Sources” Voices from the Open Source Revolution, pp. 91-100, 1999.


