

Software Engineers' Mental Processes for Problem Solving

Khairul Azman Aziz

A thesis submitted for the degree of Doctor of Philosophy at the University of Otago,
Dunedin, New Zealand.

21st June 2013

ABSTRACT

Problem solving by software engineers is an integral part of the information technology profession and is acknowledged as such both in academia, where these skills are taught, and amongst employers. In executing a task such as software development, software engineers use documentation, write and test codes and often follow established development methodologies. These tasks require substantial mental processes. Whereas formal development methodologies and the software solution can be observed visually, engineers' mental processes are abstract and implicit. The engineers themselves often find it difficult to articulate these mental processes. This study aims to uncover how software engineers solve problems by posing the question: What are the mental processes software engineers go through to solve problems? When analysis of the data revealed mental processes were not undertaken in isolation, two further questions were posed: What other important components in problem solving are there among software engineers? and How do the mental processes of problem solving interact with the other components?

This study used a qualitative, interpretive approach to produce a theory grounded in the data. Mental processes and other components were identified through interpretation of interview transcripts from 11 software engineers relating their experience on a particular project. Projects included software migration, customisation of commercial software, modification of small modules, and new development of a multi-user information system.

A model featuring four main component groups in software engineers' problem solving was developed from the interview data. These intertwining groups are: the problem solver, the environment, problem solving aids and the solution. The study's findings offer suggestions for how the more efficient use of mental processes for software engineers could be taught in professional training schools and academic curriculums. The findings also highlighted the importance of elements such as tools and technology and the formation of project teams to facilitate the more effective use of mental process and development activities.

ACKNOWLEDGEMENTS

All praises be to the One God, without the will of Whom, this PhD endeavour would never be a success. He provided solutions to problems, means to overcome difficulties, and the spiritual and emotional strength to not give up. I am ever grateful to Him.

Pursuing a PhD study is never a lonely journey. Honourable mention must be given to the following people who gave great impact on my study. No amount of thanks would ever be enough.

- My wonderful supervisors, George Benwell and Paul Theivananthampillai, for their professional guidance and undivided dedication.
- My thesis convenor, Michael Winikoff for his continuous and relentless assistance from the start of the thesis examination to ensure I successfully reach the final stage of the PhD process: preparing the hard-bound copies. The three examiners for their critical yet detailed, constructive and important comments.
- Fourteen interviewees (including three of whom their interview transcripts were not included in the analyses) who happily gave very detailed accounts of their software development experience. These provided valuable information and insights for the analysis. Their management for their ever-willingness to grant access to their valuable and irreplaceable staff.
- My thesis proof-reader, Jill Leichter, who vigorously read through and improved the language and readability aspects of two drafts of my thesis. My research assistant, Nur Syahidah Sufi, for performing the cognitively demanding work of coding some of the interview transcripts for the inter-rater reliability tests.
- Staff of the Otago's School of Business, particularly from Information Science and Accountancy departments and the Dean's Office for their cheerful help.
- University of Otago and Universiti Kebangsaan Malaysia for their ever-needed administrative and financial support, including through the Otago's doctoral scholarship and UKM's staff scholarship.
- Dunedin's community, particularly from *Sarjana* Otago and Al-Huda Mosque, for making Dunedin to feel like home away from home.

- My wife, Nor Hamiza, and children; Asma, Aisyah, Afifah, Alyaa, Anas, Aina and Arif (the last three born during the PhD tenure); for their emotional support, forbearance and patience. Never failed to lighten up my spirit during the lows. My parents for visiting us twice in New Zealand and touring around the country for that needed breaks, and especially their never-ending prayers.

There are too many people to mention, who had a hand or two on the completion of this thesis, directly or otherwise. Again, I could not say enough thanks. But please know in your hearts that you are all passionately appreciated.

CONTENTS

Abstract	ii
Acknowledgements	iii
Table of Contents	v
List of Tables	ix
List of Figures	x
List of Abbreviations	xi
1. Introduction	1
1.1 Background	1
1.2 Research Objective	3
1.3 Research Question	3
1.4 Importance	4
1.5 Overview of the Thesis	5
2. Literature Review	6
2.1 Background	6
2.2 Information Processing Theory	9
2.3 Distributed Cognition	13
2.4 Mental Processes	14
2.5 Summary	18
3. Research Methodology	19
3.1 A Qualitative Approach to Study the Problem Solving Process	20
3.2 Grounded Theory	21
3.3 Applicability to the Study	25
4. Method	26
4.1 Initial Study	26
4.2 Data Collection	27

4.2.1	Getting Contacts and Commitment	27
4.2.2	Interviews	29
4.3	Data Analysis	32
4.3.1	Transcribing the Interviews	32
4.3.2	Coding	32
4.3.3	Categorising	33
4.3.4	Modelling Category Relationships	33
4.4	Trustworthiness of Data	34
4.4.1	Interviews	39
4.4.2	Inter-rater Reliability	40
4.4.3	Findings	41
4.5	Ethical Consideration	42
4.6	Summary	42
5.	Results	43
5.1	Overview of Participants' Task Context	43
5.1.1	Jasmine	43
5.1.2	Rachel	44
5.1.3	Jessica	46
5.1.4	John	47
5.1.5	Bill	49
5.1.6	Sam	50
5.1.7	Rick	52
5.1.8	Jack	53
5.1.9	Barry	54
5.1.10	Kyle	55
5.1.11	Patrick	57
5.2	Mental Processes	58
5.2.1	Externalising	59
5.2.2	Understanding	61
5.2.3	Decomposing	65
5.2.4	Brainstorming	67
5.2.5	Evaluating	68
5.2.6	Selecting or deciding	70
5.2.7	Focusing	72
5.2.8	Extrapolating	74

5.2.9	Designing	75
5.2.10	Verifying	77
5.2.11	Viewing differently	78
5.2.12	Simulating	80
5.2.13	Learning	81
5.2.14	Compromising	83
5.3	Other Problem Solving Components	84
5.3.1	Development activities	85
5.3.2	Solution object	91
5.3.3	Solution success	95
5.3.4	Problem information	95
5.3.5	Knowledge and experience	100
5.3.6	Structure	102
5.3.7	Tools and technology	103
5.3.8	Iteration	104
5.3.9	User understanding and communication	105
5.3.10	Team and collaboration	106
5.4	Summary	107
6.	Conclusion	108
6.1	Summary of Findings	108
6.2	General Model of Software Engineers' Problem Solving	123
6.3	Suggestions for Practice	127
6.4	Limitations of the Study	129
6.5	Future Research	130
	Bibliography	132
	A. List of Data Collection Events and Protocol Questions	140
	B. Transcript of Second Interview with Bill	143
	C. Models of Subjects' Problem Solving Components	151
C.1	Barry	152
C.2	Jack	154
C.3	Jasmine	155
C.4	Jessica	156
C.5	John	157

C.6 Kyle	159
C.7 Patrick	161
C.8 Rachel	162
C.9 Rick	164
C.10 Sam	165
D. List of Initial Codes	166
E. Inter-rater Reliability Summary	173
F. Information Sheet and Consent Form	174

LIST OF TABLES

1	Interview protocol	28
2	Interview summary	30
3	Reflections on rigour	35
4	Mental processes among participants	59

LIST OF FIGURES

- 1 General model of software engineer's mental processes for problem solving 124

LIST OF ABBREVIATIONS

AI	artificial intelligence
APEX	Application Express
BA	business analyst
CMS	content management system
DRY	don't-repeat-yourself
G1...G7	General Principle 1...General Principle 7
I1...I7	Interview Guideline 1...Interview Guideline 7
IPS	information processing system
IS	information system
IT	information technology
LTM	long term memory
MPMT	modus ponens, modus tollens
NDM	naturalistic decision making
OO	object-oriented
PBRF	Performance-Based Research Fund
RAD	rapid application development
RDM	rational decision making
RQ	research question
RTM	representational theory of mind
SQL	structured query language
STM	short term memory

T1...T3	Theorising Issue 1...Theorising Issue 3
TFD	traditional functional decomposition

Chapter 1

INTRODUCTION

Chapter 1 presents the theme of the research: the mental process that are part of a software engineer's problem solving. The research questions and their importance are highlighted. Finally an overview of the thesis is provided.

1.1 Background

How do engineers solve problems? Problem solving is one of the most essential skills demanded of a software engineer. Knowing how engineers solve problems is important to software engineering students and practitioners so they can be more aware of problem solving that work. This study aims to uncover the processes that software engineers go through to solve real-world problems via a grounded theory approach.

Problem solving skills are undoubtedly important for software engineers. Problem solving is an integral part of the information technology (IT) profession and is acknowledged both in academia where these skills are taught and amongst employers. Huang et al. (2009) examined the skills needed for field of IT and found problem solving to be one of the skills more frequently stated as required for the job. Using a grounded approach for identifying skills mentioned in job advertisements, academic and professional literature, Huang et al. categorised them into technical, business and humanistic skills. Within academic literature, problem solving skills appear in technical and business skills, and are ranked the fourth most discussed business skill out of 15.

In job advertisements, problem solving skill is one of five essential humanistic skills desired by employers. Ahmed et al. (2012) surveyed 500 job advertisements across North America, Europe, Asia and Australia and found problem solving skills to be a mix of highly and moderately demanded skill (out of three categories: highly, moderately and low demanded skill) among nine soft skills demanded for software engineering related jobs: system analysts, software designers, computer programmers, and software testers.

An educationist and researcher, Jazayeri (2004) endeavoured to restructure the curriculum of a degree in informatics to include software engineering skill sets with problem solving as one of the software engineering techniques or principles that should be taught.

To be able to teach problem solving skills, the manner in which software engineers use them must be understood.

The next questions that comes to mind is how do software engineers solve problems? The typical problem solving processes involve going through the process of identifying the problem, developing alternatives, selecting the best alternatives, implementing a solution, and evaluating results. This problem solving pathway follows quite neatly along the traditional structured system development methodology, also called the systems development life cycle with activities systems analysis, conceptual design, physical design, implementation and conversion, and operations and maintenance (Romney and Steinbart, 2012). There are other development methodologies as well. Dennis et al. (2006) grouped the methodologies into structured methodologies (such as waterfall and parallel); rapid application development (or RAD which includes phased and prototyping); and agile methodologies (such as extreme programming).

These methodologies and related activities can be physically observed. However what actually goes on in the mind of the engineer? What do they actually think? What processes they run through in their minds? These are more abstract and need a sustained research effort.

Theoretically, problem solving involves the transformation from a problem state to a desired state. For instance, software design is a problem solving task faced by software engineers that involves unlimited transformations from the initial state of the problem to a non-problem state of typically unclear goals (Malhotra, Thomas, Carrol & Miller, 1980). Human information processing theory by Newell and Simon(1972) has been commonly used to understand how human solve problems. This theory used computer processing as metaphor to understand how humans process information.

Previous studies that looked into the mental processes for problem solving used a verbal protocol analysis on software problem experiments. The aim of this study is to uncover the processes that software engineers actually go through in real life situations. Previous experiments have controlled a number of external factors and therefore the processes uncovered may not reflect what engineers actually go through in real life. This study accumulates a narration of software engineers' experiences solving their problems, and analyses those narratives using the grounded theory approach to present a model of software engineers' mental processing for problem solving.

1.2 Research Objective

The research objective is to uncover how software engineers solve problems. Specifically, this study is looking at the mental processes that software engineers go through because it is one of the most difficult components to grasp in problem solving and it warrants close examination. Other components, such as artefacts and tools used in the problem solving task can be seen visually. Formal development methodologies can be observed and are usually chosen explicitly by the engineers or management based on the type of software they intend to use, but mental processes are abstract and implicit. They cannot be seen, and sometimes the problem solver cannot explain how they think.

Ohlsson (2012) states that the main goal of any research on problem solving is to produce a general theory that contains principles that capture properties of cognitive processes humans go through to solve problems. A model for software engineers' problem solving processes will be proposed after the results are presented in Chapter 6.

1.3 Research Question

Using the grounded theory approach, the study starts off with one general research question (RQ).

RQ1 What are the mental processes software engineers go through to solve problems?

During the analysis of the interview transcripts, it was found that mental processes were not undertaken in isolation but with other components as well. Hence, two other research questions needed to be answered in order to better understand the mental processes in problem solving.

RQ2 What other important components in problem solving are there among software engineers?

RQ3 How do the mental processes of problem solving interact with the other components?

The answers to the three research questions should help in the understanding of how software engineers solve problems.

1.4 Importance

Besides contributing to the development of theories on human problem solving, uncovering how software engineers solve problems through mental processes and other components would be important in many ways, including:

1. Professionals and students of software engineering will benefit from knowledge of how software engineers actually solve problems. Jonassen (2000) asserted that learning the process of problem solving in an educational setting is very limited because the processes are not fully understood. If those processes can be articulated, the learning process of problem solving could be enhanced. The benefit for professionals and students would be through utilising appropriate mental processes and being more aware of appropriate and efficient thought processes, i.e., using the process of metacognition, which is being aware of one's own mental and thinking processes (Flavell, 1979). Metacognition allows engineers to monitor the progress of their problem solving task and apply knowledge to new issues (Mercado & Romo, 2011). Gourgey (1998) suggested that metacognition allows students to define the nature of their problem, select relevant mental models, use the most relevant strategy to implement the mental models, and to be aware of their progress towards the solution. Metacognition will allow engineers to be aware of how they are approaching a problem and to compare that with the model proposed in this study which could redirect their thinking towards a better way of solving the problem.
2. Tools could be developed or available tools modified and enhanced to facilitate the successful mental processes. Understanding the mental processes that engineers go through in programming one of the tasks they're engaged in, "will permit the design of tools that mesh with the programming process." (Singer et al., 2010, p. 174).
3. Understanding how software engineers think may be used as a framework for artificial intelligence (AI) developers to build tools that could automate substantial parts of software engineering tasks. The use of AI in software engineering aspires to the possibility of quicker software development of higher quality (Ammar et al., 2012). Traditionally, research on AI and software engineering have been two separate disciplines going in their own directions, although the idea of using AI on software engineering is an attractive idea (Jain, 2011). In the last 20 years, the use of AI on software engineering has started to receive substantial attention in the

research community, although the application has been restricted to the research community and had very little impact in practice (Ammar et al., 2012). This study on the mental processes of software engineers' tasks may help bridge the gap between reality and aspirations by studying software engineers in real-world problem solving settings.

1.5 Overview of the Thesis

This chapter introduced the reader to the research question and the importance of the study. The next chapter discusses the literature surrounding the concept of problems, human problem solving and mental processes. Chapter 3 discusses why grounded theory was chosen as the appropriate methodology for the study. Chapter 4 delves into the specific steps taken for gathering and analysing the empirical data and results of the analysis are explained in Chapter 5, with direct quotes from the interviews. A discussion of the results is presented in Chapter 6 with a model presented for software engineers' mental processes for problem solving, based on the results found utilising the grounded approach.

Chapter 2

LITERATURE REVIEW

This chapter begins with some background of software engineers, the nature of problems and problem solving concepts. The chapter then discusses selected theories that delve into human problem solving. These theories are, however, not used as a framework for this study, rather a fresh model is proposed in Chapter 6 after a detailed analysis of the results using a grounded theory approach. This chapter then discusses the mental processes undertaken by software engineers found empirically in previous studies.

2.1 Background

To understanding how software engineers solve problems, it is important to first have a grasp of who is a software engineer, what is a problem and what is problem solving.

Software engineer

There is no formal definition of a software engineer. The closest definition can be found in the field of software engineering. which is defined as:

the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (Abran, Moore, Bourque, Dupuis & Tripp, 2004, p. 1)

Therefore a software engineer could be defined as a practitioner of engineering concerned with the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software. Professional software engineers are:

...persons with professional cognitive models and knowledge on software engineering. They are trained with: (a) fundamental knowledge that governs software and software engineering practices; (b) basic principles and laws of software engineering; (c) proven algorithms; (d) problem domain knowledge; (e) problem solving experience; (f) knowledge about program developing tools/ environments; (g) solid programming skills in multiple programming languages; and (h) a global and insightful view on system development, including its required functionalities as well as exception handling and fault-tolerance strategies. (Wang & Chiew, 2010, p. 83)

There is debate and controversy as to whether software engineers can be regarded as professional engineers (Parnas, 1994, 1998), but this study is not concerned with that and uses the term loosely to refer to people who work on software.

Singer et al. (2010) studied the work practices of software engineers involved in maintaining a large telecommunications system of their company. The authors collected data in four ways, using a web survey to ask the engineers what they do, following closely an engineer performing his work for 14 weeks, observing nine engineers working for one hour, and interviewing several engineers. Some of the activities the engineers engaged in were documentation, looking at the source, writing code, researching/identifying alternatives, fixing bug, designing, testing, reviewing other's work and configuration management.

Problems and problem solving

According to Jonassen (2000), something is considered a problem if it has two critical attributes. The first attribute to a problem is that it is "an unknown entity in some situation (the difference between a goal state and a current state)" (p. 65), i.e., a matter is a problem when its current state is not the desired state. The second attribute of a problem is that solving the problem is perceived to be valuable by someone. A matter is not considered a problem if no one perceives that it is important to solve it.

Jonassen (2000) also described attributes that can differentiate problems, including structure, complexity and domain specificity. Problems can also vary between being ill- and well-structured. A well structured problem has all the components of the problem presented to the solver, requires the use of finite number of rules or principles that are prescribed and predictive, and has solutions that can be identified and understood. An ill-structured problem has components that are unknown (Wood, 1983); has either a number of possible solutions or ways of arriving at the solution, or none at all (Kitchener, 1983) and it may be evaluated using multiple criteria without certainty as to which criteria should be used often involving the solver's judgment, opinion or belief.

The complexity of a problem is related to the amount of issues, functions or variables the problem contains, the more there are in the problem, the more complex the problem (Jonassen, 2000). The level of connectivity and stability among those properties are also aspects of the complexity of the problem and although they are not exactly synonymous, the difficulty of a problem is gauged by problem solvers based on the problem's complexity. The more issues, functions and variables in the problem, the more cognitive operations necessary to solve it (Kluwe, 1995). Although there is a tendency

for ill-structured problems to be more complex, such is not necessarily the case. Jonassen (2000) gave the example of deciding which clothes to wear for different occasions as a simple, yet ill-structured problem.

Domain specificity refers to what extent the problem requires problem solving that is confined to the field of the problem (Jonassen, 2000). Problems vary with regard to whether the necessary problem solving skills are domain-specific or domain-general. Some problems require problem solving skills that are specific to them and other problems require general problem solving skills.

Jonassen (2000) conducted a cognitive task analysis on hundreds of problems collected to categorise different types of problems. By looking at the problems' characteristics, Jonassen categorised problems into 11 different types arranged according to a continuum of well- to ill-structured: logical, algorithmic, story, rule-using, decision making, troubleshooting, diagnosis-solution, strategic performance, case analysis, design, and dilemma problems. Jonassen discussed in detail each of the problem types according to its characteristics including learning activities, inputs, success criteria, context, structuredness and abstractness.

Since the software engineers in this study would be expected to be involved in problem solving tasks closest to the design problem categorised by Jonassen (2000), Jonassen's categorisation of design problem is summarised here. Examples of problems categorised by Jonassen as design problems included writing a story, designing a bridge, developing a school curriculum, and developing a strategy for investment in a money market. Design problems are one of the most complex and ill-structured problems, although software design itself is well-structured among design problems (Jonassen, 2000). Design problems are ill-structured for many reasons; there is a high number of degrees of freedom in the problem statement, late feedback, there are only better or worse answers and no correct or wrong answers, and the output of the design is very important even though the basis for its evaluation is unclear (Goel & Pirolli, 1989). Solving design problem involves an extensive structuring of the problem (Goel & Pirolli, 1989; Zannier et al., 2007).

Problem solving is the sequence of mental operations undertaken with the direction of achieving some goals (Anderson, 2004), or moving from a current state to a desired state. Newell and Simon (1972) distinguished the problem solving task from a "programmed activity" (p. 822) which is more stereotyped and mechanised. An example of programmed activity is when a subject uses simplex algorithm to solve a linear programming problem. The task involves a highly specified algorithm via a highly mechanical activity with a low error rate. Central to the problem solving is the

formation of a problem space (Newell & Simon, 1972), or mental representation of the problem situation (Jonassen, 2000).

Reality and the representation of reality are two distinct entities. When confronting a problem, decision makers may well be facing a problem in reality, but mentally conceive the problem differently. The same real problem can also be represented in multiple ways, and unfortunately even assuming the representations show accurate pictures of the reality of the problem, the manner they are shown can affect the problem solving approach and outcome. For example, although Roman numerals and Arabic numerals are used to represent the same reality, the use of the latter for multi-digit multiplication is much more efficient (Zhang & Norman, 1994).

Mental representation, generally regarded as “a mental object with semantic properties”, is a concept that receives substantive attention in philosophy and cognitive science disciplines (Pitt, para 2). The representational theory of mind (RTM) views mental representations as having relationships to intentional mental states that include “thoughts, beliefs, desires, perceptions and imagings” (para. 4).

Mental processes are sequences of intentional mental states. And since mental representations may not reflect the actual problem in reality, and hence hinder the use of correct problem solving efforts, processes must act on the mental representation of the problem.

At the very core, problem solving involves the processing of information. In information processing theory, humans can be viewed as processors of information (Newell & Simon, 1972). The human mind is seen as a computer with a limited ability to process information using both long term and short term memory. Through receptors, external information is gathered into the problem space and processing occurs in the problem space using rules extracted from the long term memory. The task environment determines the structure of the space.

2.2 Information Processing Theory

At the very core, problem solving involves the processing of information and humans can be viewed as processors of information (Newell & Simon, 1972). The unit of analysis in information processing theory is information processing system (IPS) and a human is regarded as an IPS.

An IPS, as with computers, has several components: internal memory, processing, sensors, external memory, and production systems. The internal memory consists of long term memory (LTM) and short term memory (STM). There is no evidence that LTM

has a limit over the life of a human being, therefore, LTM has the potential to be able to hold an infinite vocabulary of symbols, the smallest units of information. Processes on the LTM include 'read' and 'write', i.e., read from LTM and write on to LTM. Write takes more time than read.

Unlike LTM, STM, the second internal memory component, has a very limited capacity. In fact, Newell and Simon (1972) quoted Miller (1956) saying that the capacity of STM is five to seven symbols based on simple digit-span experiments. Processes "...take their inputs from STM and leave their outputs in STM..." (p. 796).

Humans as an IPS also perform serial processing although it does not mean accessing memory via a serial scan. Serial processing in a human means only one piece of elementary information processing can be executed at a time. Newell and Simon (1972) gave the example of processing whether a number is divisible by another number, and performing the same task on several numbers would take the same number of times as there are numbers to be processed, thus illustrating a serial process. The human would not be able to process all the numbers concurrently. Some form of concurrency may occur with motor activities, but even then there are limitations. Elementary processes typically take around fifty milliseconds, but are more influenced by the speed of reading in STM and LTM.

Besides these components, humans have sensory systems that obtain information from the external environment, such as through visual receptors (eyes) that trigger the production of associated symbols in our STM.

External memory (EM) is important for problem solving. There are huge differences in problem solving between having and not having EM. For example, a person would take up to 100 fold more time when performing multi-digit multiplications without pencil and paper than when he has those external memory tools. EM has unlimited capacity, and can be read and written on. The time required to write onto EM is shorter than onto LTM, but longer than onto STM.

The last component of the IPS is a production system. "A production is a process with two components: a condition component, and an action component" (Simon, 1978, p. 282). The production system to an IPS is the same as programming is to a computer.

The theory proposed "...that all problem solving occurs in some problem space" (Newell & Simon, 1972, p. 789). Problem solving occurs "by search" (p. 809) in the problem space. This is asserted across all types of tasks and problem solvers based on a range of studies by Newell and Simon (1972). The task environment is the task as represented to the problem solver, while problem space is how the problem solver views the task environment (Simon, 1978).

A problem space is created, or recalled from LTM if it is present there, after the problem is presented, recognised and understood. The problem space contains the possible solutions to the problem and can be modified throughout the task. Creating a problem space involves building “(1) a representation for the knowledge state, (2) operators to generate new knowledge, and (3) processes to test whether operators are applicable and whether positions are desirable” (Newell & Simon, 1972, p. 858). The problem space and methods may change throughout a single problem solving attempt.

As the place where problem solving takes place, the problem space consists of: a set of elements, a set of operators, the initial state of knowledge, the problem, and the total knowledge available. A knowledge state is the current knowledge of the problem solver, where knowledge refers to information available that can be retrieved in a fraction of a second by the problem solver at that point of time (Simon, 1978). A node in problem space is a possible state of knowledge.

Knowledge state in LTM is cumulative in nature. But because STM has a very limited capacity, it does not share the same nature. Therefore searches in STM often go to earlier knowledge states. Accumulating knowledge onto EM is much faster than onto LTM due to the different write rates between them. However, accessing the huge repository of knowledge from EM may become practically impossible. Accessing this much knowledge from LTM is more feasible, although more time is needed to store that knowledge prior to the recall.

Problem solving involves moving from one knowledge state (node) to another until the desired state is found. Moving from one knowledge state to another may require the IPS to backup the earlier knowledge state onto either EM or LTM, due to the limited capacity of STM. However, backing up the earlier knowledge state takes time away from the IPS’s actual discovery of new information to form the new knowledge state, because the IPS performs processes serially.

Well-structured problems have a finite number of nodes, which have been exhaustively identified by researchers, whereas in less structured problems, the exhaustive identification of nodes is virtually impossible. For example, structured problems such as “missionaries and cannibals” have only 16 legal positions, hence nodes, and DONALD+GERALD has $9!$, or 362,880 nodes (Simon, 1978). However, the difficulty of problem solving does not depend on the amount of possible nodes in the problem space. “Effective problem solving involves extracting information about the structure of the task environment and using that information for highly selective heuristic searches for solutions” (p. 277).

Ill-structured problems differ from well-structured problems in that, 1) the criteria to

determine whether the task goal has been achieved is more complex and less definite; 2) the information needed to solve the problem is not completely available in the problem's instructions; and 3) there is no simple tool to generate all the possible nodes or legal moves (Simon, 1978).

Structure is the information based on nodes visited that allow for the prediction of the properties of parts of the problem space not yet visited.

There are four points in a problem space where an IPS needs to make decision: "1) at a knowledge state (a node in the problem space) to select an operator to be applied, 2) at a new knowledge state, to determine whether problem solving shall continue from this state or not, 3) at a knowledge state, to determine whether the knowledge state shall be remembered, so that the return can be made to it at some later time, 4) at the decision to abandon a knowledge state, instead of continuing to search from it, to select another knowledge state as the backup state." (Newell & Simon, 1972, p. 826). The two crucial decisions are at points 1 and 2. The decision of which operator is to be applied is a foresight decision, "made on the basis of information that allows a prediction of the relative promise of different actions". On the other hand, evaluating whether to continue with a knowledge state is hindsight, and is made on the basis of information "that judges the efficacy of past actions".

Decision points relate to states when goals and operators are determined, and new states are evaluated. Intelligence plays a part here but less of a role in the process of saving states, "selecting old states for a retry", and "rejecting goals".

Method is the "device for generating a coordinated pattern of such decisions" (Newell & Simon, 1972, p. 837). Methods are "organisations for behaviour that bear a rational relation to solving a problem", or a possible way to solve a problem provided that the method is correct. When a method is provided, the behaviour will be determined therefore method is one level of organisation. An example of a method is in a chess game, a method states that a defender should be added when a certain type of attack is faced. The method does not specify which defender from the available defenders. Hence, method specifies a behaviour only partially. A problem solver has a set of methods that could control his behaviour. Examples of methods include working-forward, working-backward, and a means-ends analysis.

There are other aids that may be present in a problem space including the possibility of working backwards, the possibility of planning, and the possibility of using external memory. These may be in separate problem spaces, although human problem solvers may embed them in a main problem space.

The theory proposes that the structure of a problem space is determined by structure

of the task environment. “The only aspects of the task environment that are relevant to solving a problem in a particular problem space are those that are reflected in the structure of that space”, and “the effectiveness of a problem solving scheme depends wholly on its reflecting aspects of the structure of the task environment” (Newell & Simon, 1972, p. 824).

The foundation of the information processing theory was based on empirical works that studied short half-hour tasks of moderate difficulty that were symbolic in nature. However, Newell and Simon (1972) said that this should not be a reason for restricting the boundary for applicability of the theory. They believed the theory could be used in broader sense if it was subjected to satisfying certain conditions. Simon (1978) proposed that processes undertaken to solve ill-structured problems are generally similar to those undertaken to solve well-structured problems.

In general, to understand human problem solving, is to describe the task environment, the problem space used by the problem solver, and the program assembled to approach the task. The problem moves from one node to another until the knowledge state that includes the problem solution is found. However, the information processing theory does not describe the behavioural differences between human problem solvers based on differences in culture, education and socioeconomic class, which may affect LTM. The theory is oriented towards the mind of the IPS only. The theory of distributed cognition introduced in the next section looks at problem solving more broadly.

2.3 Distributed Cognition

Distributed cognition theory (Hollan, Hutchins & Kirsh, 2000; Hutchins, 1995), recognises that representation is not only in the mind, but can also be external. The theory views an information processing task as an interwoven link between all the decision players, external representations and the environment.

Researchers attributed the foundation of distributed cognition theory to Hutchins from the 1980s (Rogers, 1997; Zhang & Norman, 1994). The theory is regarded as

... a scientific discipline that is concerned with how cognitive activity is distributed across internal human minds, external cognitive artifacts, and groups of people, and how it is distributed across space and time (Zhang & Patel, 2008, p. 137)

Distributed cognition looks at cognitive tasks as encompassing the interactions between people involved in the task as well as the resources and materials that surround them in the environment (Hollan et al. 2000). The unit of analysis for this theory is not restricted

to the thinking undertaken in the mind of a single person, but includes the processing of several “systems” such as in an airline cockpit (Hutchins, 1995).

In his 1995 study on an airline cockpit environment, Hutchins considered the cockpit to be a unit of analysis (this is larger than a typical representation of a person as the unit) to explore how a cockpit system performs cognitive tasks related to airspeed and wing configuration. By using this level of analysis, he was able to examine the internal representation and cognitive processes without referring to the minds of the pilots. He concluded that the memory processes required during the aircraft’s descent could be distributed among the pilots or devices and the use of devices to show the representation of certain flight information allowed relief for the pilot’s memory space. It also allowed a change in mental processes from a more-demanding judgment process to a less-demanding perceptual process. For example, speed card booklets determine the reference speed (through a perceptual process), and speed bugs represent the reference speed in front of the pilot, eliminating the need for pilots (the flying and non-flying) to perform analytical calculations and to store the reference speed in their minds. A reference speed must be set and achieved in order to safely manoeuvre the aircraft to land, or to resume flight if a landing cannot be made. Such a range of speeds depends on numerous factors, fixed and variable according to the circumstances of the flight at the particular time.

2.4 Mental Processes

Several studies suggest the mental processes used by software engineers. The authors mentioned the mental processes of problem solving in different aspects of systems development, such as in software design and programming and findings from this study will be compared with them. One of the processes mentioned in the literature is mental simulation (Giordano, 2002; Kim & Lerch, 1992), the execution of the problem solution internally, and it is prevalent in software design (Kim & Lerch, 1992).

Kim and Lerch’s (1992) study compared cognitive processes (as they termed it), between using object-oriented (OO) design methodology and traditional functional decomposition (TFD) methodology in logical design processes. They defined logical design processes as the process of understanding a problem and transforming what is understood into a high-level software design. Kim and Lerch looked at two mental processes in the logical design, rule induction and mental simulation. Rule induction is finding a set of rules that can be used to solve a set of problems. An instance of the problem is used to propose a rule, and that rule is tested using another instance of the

problem. The confirmed rule will be used for the solution. Mental simulation executes a solution in the mind using either symbols or test-cases.

Their literature review suggested that logical design involves the use of two problem spaces. The first space, the task domain problem space, is an internal representation of the problem expressed in the language of the problem domain, using terms such as income and people, for a tax domain. The second space, the software domain problem space, is an internal representation of the problem solution expressed in the language of the software methodology, using terms for data structure and control logic.

The traditional functional decomposition methodology requires designers to focus on the software functions. OO design methodology requires the designer to identify objects and attributes of the task-domain problem space, and concentrate on the objects in the problem instead of the functions in the software.

Kim and Lerch (1992) performed experiments with professional programmers assigned a logical design task similar to the Tower of Hanoi problem to asked to produce a software solution. Data on mental processes for the performance of the task was collected using verbal protocol analysis. They found that the mental simulation process was less engaged when OO design methodology was used in software design than when TFD methodology were used. Subjects who use TFD methodology performed a large number of mental simulations on test cases before coming up with a set of rules for the solution.

Giordano (2002) studied the evolution of the use of graphical representation in information systems (IS) development projects among subjects where previous designs were available through a shared repository. The subjects were undergraduate computer engineering students in teams of two from various levels of study. They were asked to analyse and design information systems to the point of producing a prototype of the systems and were given access to two previous generations of systems. Their solution would be the third generation system. The study involved a mixed method that included content analysis on the design documentation, interviews (in the form of oral examinations), and quantitative analysis on the relationships of variables. The computer engineering students explained how system representations of earlier generations were adapted through generations of analysts to improve the IS designs. Giordano found that the analysts strove to achieve a cognitive fit among system representations to the system in small design steps, by simulating the relevant scenarios and conceptual operations within those steps.

Advanced students tended to introduce innovations, but made a less-detailed design with slightly flawed representations. Novices with a high grade point average tended

to concentrate on getting the details correct with a successful effort in data modelling, while filtering out idiosyncratic representations. Giordano's explanation on the students' adaptations uncovered a substantial insight into the mental processes required for a task solution.

Brooks (1977) asserted there were three processes in programming tasks: understanding, method finding and coding. Understanding involves acquiring knowledge of the basic elements of the problem, which include objects and their attributes. Understanding is achieved, by reading and asking questions. Method is an outline of the program, similar to a blueprint, which specifies the data structure and the operations to achieve the objective.

Based on a qualitative analysis on one observational study and two experimental studies, Malhotra et al. (1980) proposed three processes in software design: goal elaboration, design generation, and design evaluation. In the observational study, dialogs between a designer and his client regarding a problem to solve was videotaped and transcribed. Malhotra found that the dialogues consisted of a series of cycles, with each cycle containing a series of states; a goal statement, goal elaboration, (sub) solution outline, (sub) solution elaboration, (sub) solution explication, and agreement on (sub) solution.

Initially the client thinks of a goal and seeks help from a designer to expand the goals into functional requirements. Goal elaboration involves a decomposition of the goal into goal-components by the client. The designer assists the client with decomposition of the goal component into functional requirements because clients do not have this knowledge.

The generation of a design starts off with the functional requirement, and the elements and organisation of the design are developed to meet the functional requirements. The selection of the best design is made when more than one design are generated. While a design is partially generated, it goes through an evaluation via discussions between the client and designer on how well the design fits the requirements.

Malhotra et al. (1980)'s study indicated that previous studies on mental processes were conducted on one problem that contained multiple sub-problems as a unit of analysis. Malhotra et al.'s results show that mental processes are non-linear and do not work on one goal level. Rather, processes work on sub-goals in a serial manner, and then the cycle repeats with the next sub-goal. However, Malhotra et al.'s study is limited to the design process. Software engineers not only participate in design, they may participate in various stages in software development that includes activities such as analysis, design, building (coding), and implementation. This study aims to look at

software engineers' problem solving with a more holistic view with the expectation that there will be other processes besides the ones uncovered by Malhotra et al. (1980) that are not part of the design.

Zannier et al. (2007)'s study also uncovered systems development processes aimed at explaining how software designers make decisions. They interviewed 25 software designers using the critical decision method (Klein & Calderwood, 1991), performed a content analysis on the interview transcripts and compared results across the 25 case studies. In general, they found that a design decision was primarily oriented towards naturalistic decision making, but contended that this finding might be biased due to most of the subject's "agile" software methods. They came to several conclusions. They found that software design is primarily about problem structuring and the more structured the design problem, the more developers used a rational decision making approach to solve the problem, while using naturalistic implementation. On the other hand, the less structured the design problem, the more developers used a naturalistic decision making approach to solve the problem, while using rational implementation in the sense that they made a consequential choice in order to compare alternatives¹.

The design decision model involves three components, namely structuring flow, structuring mechanisms and structuring perspective. When software developers faced a design change requirement that was ill-structured, they went through a process whereby they restructured the problem and the product itself also became the object of subsequent re-structuring. This cycle is called "structuring flow". When structuring the problem or making a design decision, software developers use information as a "structuring mechanism". Such mechanisms include personal experience, ideas, opinions, knowledge, group interaction, external influences, existing model of the problem, existing work processes, and preferred evaluation criteria. Along the structuring flow, software developers make design decisions based on a "structuring perspective". Their perspective may be based heavily on rational decision making (RDM-based), or naturalistic decision making (NDM-based).

¹ Zannier et al. (2007) operationalised the distinction between RDM and NDM in terms of four aspects. For the first aspect, the decision goal, a decision maker aiming to optimise design based on right/wrong cues follows an RDM approach, whereas he follows an NDM approach when he aims for a satisficing design based on better/worse cues. For the second aspect, the decision method, a decision maker follows the RDM approach if he considers several options, and follows the NDM approach if he makes a single evaluation. For the third aspect, the decision environment, a decision maker follows the RDM approach if he is not concerned with time pressure, and follows the NDM approach if he is. For the last aspect, decision knowledge, a decision maker follows the RDM approach if he is aware of all possible courses of action using explicit searches for decision making, and follows the NDM approach if he is unaware of all possible courses of action and relies on unspecific experience or knowledge.

2.5 Summary

This chapter introduced a definition of a software engineer and concepts regarding problem solving and mental processes. Theories that looked into human problem solving have been presented. Intuitively, the highly cited theory of information processing by Newell and Simon (1972) that became the foundation of numerous studies on problem solving could be used as a framework for this study. However, the theory was primarily used to look at a very detailed, low level problem solving with a short and contrived problem solving task environment. Zhang and Norman (1994) and Zhang (1997), however, considered that the problem solving unit of analysis should be expanded to account for more factors. This cast doubt on the appropriateness of the information processing theory for this study on real-life problem solving among software engineers. Some of software engineers' major tasks, such as software development, typically lasted for months, therefore using verbal protocol analysis may not be appropriate for this study.

In addition, the following quote further strengthened the decision not to use information processing theory as a framework and to use a grounded theory instead to develop a new model for understanding software engineers' problem solving processes and to look again at what components are important for those processes:

However, people are not computers, and whilst they certainly process information, it is an activity that takes place within a complex web of other human actions by the individual and as part of environments, communities, cultures and societies in which the individual participates (Star, 1989, 1992, as quoted by Prior, 2011, p. 30)

Chapter 3

RESEARCH METHODOLOGY

This chapter discusses the research methodology adopted in this study. Two contrasting methodologies often debated in the literature are quantitative and qualitative methodologies. Quantitative is described as “positivist” and “empiricist” and qualitative research is “naturalistic” and “interpretive” (Bryman, 1984), although there have been instances where a qualitative method (such as case study) was used in a positivist study (Myers, 1997), and a non-naturalistic method (experiment) was used in an interpretive study (Klein & Myers, 2001).

The distinction between positivist and interpretive research is the philosophical viewpoint on valid research. Myers (1997) described a set of differences between the two, and this is restated in brief here to elucidate the philosophical position that was chosen for this research. The two viewpoints occupy the extreme ends of the continuum beginning with their basic definition of reality. Positivists assume that there is an absolute objective reality and that it can be described and measured independently of the reality by an observer. Interpretive researchers, on the other hand, assume that reality is relative to the observer and that it is described and measured through social constructions such as language, consciousness and shared meaning.

To achieve an increased understanding of phenomena, positivists attempt to test their theory. Studies are usually classified as having a positivist view when they are conducted with a systematic application of these components: 1) a formal proposition, 2) quantifiable measures of variables, 3) hypothesis testing and 4) drawing inference from a sample to the population (Orlikowski & Baroudi, 1991). On the other hand, interpretive research attempts to understand phenomena through meanings that people assign to them. They do not predefine a dependent and independent variable but focus on the full complexity of human sense making as the situation emerges (Kaplan & Maxwell, 1994).

Reality in this study involves mental processes and other components related to them such as the problem solving task, solution objects, problem information and development activities. This study assumes that there is a possible absolute reality to these constructs, albeit some are more difficult to identify or measure than others, and hence identification is through the researcher’s interpretation of language. More profoundly, as the core issue of the study, mental processes are highly interpretive.

Mental processes cannot be seen by the naked eye, although the results are visible. Even the respondents had difficulty relating their mental processes and therefore at best, their mental processes are the interpretation of the researcher. Based on these characteristics, this research is interpretive because the very abstract nature of the reality being pursued can be only be realistically explored through an interpretation of language.

3.1 A Qualitative Approach to Study the Problem Solving Process

This research is focused on uncovering the mental processes that occur during problem solving and their relationships with other important components, rather than confirming existing theories on problem solving. Hence, the suitability of qualitative methods for the analysis is considered.

Kaplan and Maxwell (1994) gave a practical example of a study evaluating a clinical laboratory information system which identified the type of objectives in IS research where a qualitative method could be helpful. These objectives included understanding how people think about things and why, examining causal processes rather than simply testing causal relationships, and studying processes as they develop rather than just the outcomes. These issues are considered to be the “black boxes” in quantitative research and have been left unexplored.

Discussions regarding the use of qualitative methods for understanding the problem solving process in other domains are very active. In the field of entrepreneurial cognition, although the use of qualitative method is strongly advocated, it is still not very prevalent (Hindle, 2004). In the field of nursing, researchers have started using qualitative methodologies to better understand how nurses make decisions (Campbell, 2008). Through the application of grounded theory in her own study, Campbell (2008) uncovered the important concepts that student nurses went through in their clinical decision making such as “connectedness” between the nurses and patients, the nurses “knowing” certain aspects of patients’ cases, and a feeling of “heroism” by the nurses. These concepts were uncovered through content analysis of interviews, and may not have been anticipated in a quantitative study.

Yang’s (1997) study explained the information-seeking element of the problem solving processes of novice learners. Data was collected using verbal protocol and observation, and analysed using grounded theory. The study offered insights into the stages, patterns and themes of the information seeking process. He put forward an information

seeking model that contains five activities: selecting sources of information, executing the search, interpreting the problem, judging the relevance of each choice, and retaining or rejecting information. Patterns of information seeking were categorised/described as prescriptive, exploratory, purposive, associative, intuitive, curious, tangential and accidental. Among the learners, four themes of the information seeking process were uncovered. Firstly, he found that learners structured their learning environments to optimize their own learning. Secondly, learners were proactively self-reflective in their information seeking. Thirdly, there was a specific progression in the information seeking when learners interpreted meanings; from chaos to order, fuzzy to clear, and unstructured to structured. Fourthly, learners experienced varying degrees of disorientation.

Fidel (1993), while addressing information retrieval research, described qualitative research as being non-controlling, holistic and case oriented. It was about processes, and was open and flexible, diverse in methodology, humanistic, inductive and scientific. Despite the fact that it is less used, these qualities make the qualitative method ideal for exploring human behaviour in depth. As a non-controlling methodology, qualitative research describes how people behave in real life situations, as they happen. By being holistic and case oriented, qualitative research “provides for broad understanding of a particular phenomenon by focusing on unique cases, but at the same time taking into account all the themes that are involved” (p. 224). By being process oriented, qualitative research explores the “dynamics of a process... rather than the static attributes of a process” (p. 225). By being open and flexible, qualitative research allows for the discovery of patterns beyond the scope of existing preconceived ideas or theories. With a diverse methodology, qualitative research can examine a phenomenon by using methods that complement one another while overcoming inadequacies.

Grounded theory was considered to be suitable for this research. This method aims to produce a theory grounded in the data through constant interplay between data and analysis. The theory formed is not based on experience or speculation, but rather on the conceptual understanding of incidents. This study aimed to uncover the mental processes, relevant components and their relationships, in order to develop a model of concept relationships derived from an analysis of data on actual problem solving tasks undertaken by software engineers.

3.2 Grounded Theory

According to Strauss & Corbin (1994) grounded theory was first presented by Glaser and Strauss (1967). Its main feature is the continuous interplay between data and analysis

in the formation of theory. Unlike a positivist approach, where existing theories are tested with a set of data, grounded theory research begins with data collection and through analysis of that data and further collection of new data and analysis, a theory is developed. Because it is grounded in the data, the theories developed using this method represent reality more closely than theories that are formed based on mere experience and perception (Strauss & Corbin, 1998).

Grounded theory does not dictate the data collection method and whether the data is qualitative or quantitative. In fact, a well known description of grounded theory is “all is data”. Data collection techniques such as interviews, field studies, documents, and video recordings may be similarly used in different methodologies. and although it is regarded as a qualitative method of research, grounded theory may use quantitative data.

Strauss and Corbin (1994) highlighted other requirements which studies claiming to use grounded theory should fulfill:

- There must be constant comparisons made
- Questions asked should be theoretically oriented
- There should be theoretical coding of data

Although Strauss and Corbin (1994) preferred that grounded theory studies adhere to the above boundaries, they acknowledged that diversions may occur.

One of the main differentiating factors between grounded theory and other methodologies is the need for constant comparison and several types of comparisons are made (Glaser & Holton, 2004). Firstly, incidents are compared between each other to find similarities and conditions that make them different, and concepts and hypotheses are generated from this information. The concepts and hypotheses are then compared to new incidents, and are elaborated on or added until saturation is achieved.

The next factor is asking theoretical questions such as, “How do these two concepts relate?” (Strauss & Corbin, 1998, p. 76). These types of questions help the development of the theory and are different from questions such as “How do I get access to this organization?” (1998, p. 76) which is a question relating to practical aspects of the research. The right questions will lead researchers to look for patterns between incidents that would suggest codes at a conceptual level above the descriptive level of the incidents (Glaser & Holton, 2004).

Another element of grounded theory is theoretical coding. This starts off with open coding, which involves line-by-line coding of the data in any way possible (Glaser &

Holton, 2004). The researcher needs to be skilled at generating codes and deciding on their relevance. By starting off with open coding, it allows the researcher to put grounding categories into the data rather than follow preconceived ideas. The open codes form the substantive codes that conceptualise the empirical substance of the incidents. Selective coding is then conducted when the central themes of the research are identified and the focus of the research has been decided. Substantive codes are then categorised to form theoretical codes that conceptualise how the substantive codes are related to each other (Glaser & Holton, 2004).

The two founders of grounded theory later diverged on how they thought grounded theory should be developed and their diverging approaches are now known respectively as Straussian and Glaserian (Heath & Cowley, 2004). The diversion became apparent with the publication by Strauss (1987) which gave detailed guidance on the process of grounded theory, and was initially intended for use by novice researchers. Glaser and some other researchers regarded Strauss' writings as less of a grounded theory because the detailed methodological emphasis was never intended for a grounded theory (Heath & Cowley, 2004). Heath and Cowley (2004) asserted that such divergences are only methodological in nature rather than ontological and epistemological and explained the philosophical divergence between the two schools in terms of the theory generation process and theory form.

Although not listed as part of the boundary that defines grounded theory, the Glaserian approach insists that coming into the research without, or with a minimal literature review is essential to allow sensitizing the researcher towards the data. The original philosophy was that the data analysis should not be influenced by previous literature, rather the researcher should look at the data with an open mind. When a researcher has already read previous literature, the analysis and theory that is developed may be restricted or funneled by previous studies, and this could dilute the groundedness of the new theory being developed from the data. Thornberg (2012) argued against the delay for literature review until after data collection and analysis of grounded theory studies, despite the original philosophy of the methodology. The Straussian view of grounded theory also held to the more relaxed view of the use of literature review prior to data collection.

The use of grounded theory in IS research is an emerging trend and relatively new (Hughes & Jones, 2003). Hughes and Jones (2003) reported their use of the method in their study regarding the use of IS evaluation method in the public sector. They had decided on the use of interpretive and qualitative case studies, and chose grounded theory as the method for data collection and analysis because they understood that the

method supports interpretive studies and the inductive discoveries of theories. Hughes and Jones (2003) started their analysis with open coding and when more data was collected and coded, categories emerged. Later, categories were merged to form more abstract categories, giving rise to a hierarchy of categories.

From using grounded theory in their study, Hughes and Jones (2003) learned that the method can help researchers understand the problem under study and externalise the findings. However, they were unsure if using “seed” categories already in mind was the correct way of executing the method. They used seed categories that were developed based on knowledge of prior literature and justified this from the recommendations of Miles and Huberman (1984), Walsham (1993) and Fitzgerald (1997). They also acknowledged that carrying out the method was time consuming with regard to the task of transcribing, coding and comparing. Furthermore, they felt that their understanding of the method was insufficient, and that made the early stages of coding difficult.

Hansen and Kautz (2005) reported using grounded theory in their study on how system developers use specialist methodologies. They performed a single case study on a large software company, with data collected via semi-structured interviews and the company’s documents. Their motivation for using grounded theory method was because they wanted to uncover how developers use the development methodologies rather than verifying whether developers use methodologies in certain ways. In applying the grounded theory method, Hansen and Kautz (2005) utilised three techniques in sequence that were suggested by the method, open coding, axial coding and selective coding. Axial coding was referred to as aiming to classify codes into categories. Similar to Hughes and Jones (2003), Hansen and Kautz (2005) also came into the study with knowledge of the literature and a broad theoretical foundation that was used for the interviews. They also mentioned the dilemma of whether such prior knowledge would render their utilisation of grounded theory method incorrect, and justified their actions by quoting Walsham (1993).

In terms of their study, among others, they found that the utilisation of development methodology is influenced by the universality of the methodology, the process of methodology introduction, experience, creating confidence and participation. In terms of the utilisation of grounded theory, they found that it allowed for very detailed explanations of the subject of the study. They also found challenges in terms of the length of time required for the analysis which took longer than anticipated and found difficulty using available tools during axial coding.

3.3 Applicability to the Study

Based on the discussion in the previous sections, this study is positioned as an interpretive, qualitative research using the grounded theory approach. In developing a grounded theory for this research, however, a strict adherence to the principles of grounded theory method was not undertaken. The data collection and analysis proceeded after some literature review. Nevertheless, this knowledge was used with caution, so as not to desensitize the researcher from theory development. Furthermore, the founders of the theory and subsequent researchers that applied grounded theory diverged on how they viewed the need to limit prior knowledge of the subject under study. The next chapter discusses in detail how the grounded theory method is executed.

Chapter 4

METHOD

This chapter discusses data preparation, analysis processes and their trustworthiness. Generally, data was first collected and then analysed, but it was not a mutually exclusive one-way process. Some data analyses were done to facilitate subsequent data collection.

4.1 Initial Study

An initial small scale study was conducted prior to the main study to ascertain mental processes and other related components that software engineers potentially went through, and to develop a set of more effective interview protocols that could be used in the main study for getting relevant responses, and for being more efficient during the time allocated for the interviews. A detailed description of the initial study can be found in Aziz, Benwell and Theivananthampillai's (2008) paper.

The context of the study was on student software development projects. The third-year IS students of a New Zealand university were divided into groups to develop a system for a real-life organisation as part of a project-based paper. The project took two semesters to complete. Two groups of students participated in the preliminary study.

The interview protocol was constructed without particular references to any established instruments, hence the need for the initial study. The software development task was a very complex task for the subjects and could be broken down into project selection, analysis of requirements, project planning, database implementation, application development and quality assurance. The students' previous knowledge relevant to the task included knowledge of the problem domain, (accounting and horse-racing²), and technical IS that included concepts learned from courses that taught the entity-relationship diagram and structured query language (SQL).

To obtain information regarding the problem, the students reported performing interviews, requesting sample spreadsheets, and seeking advice from online help or forums. Some very limited accounts of mental processes were obtained and included

² Knowledge of horse racing was important for the project that needed the students to develop an application system for an organisation that promotes and organises thoroughbred horse racing in a region within New Zealand.

the visualisation of oneself as a user, an attempt to understand the requirements, and the grouping and breaking down of elements of the problem or system.

Some indication of the component relationship's effect on performance was observed. One group's system was particularly more successful than the other. It was found that the more successful group had a team leader with a stronger application domain knowledge in addition to some work experience, and conducted many more interviews to obtain more comprehensive problem information.

Several lessons were learnt from the initial study. Firstly it was difficult to obtain simultaneous cooperation from both student software engineers and users. Therefore, an alternative way of examining performance included perceived performance questions for the engineers in the protocol. The original interview protocol was found to be poorly structured, because the questions were ordered based on importance, given the time limit of each interview. The resulting interview protocol used in the main study is listed in Table 1, (see Appendix A for the original set of protocols).

The preliminary study also showed that the performance of software development differed greatly between a group that has a leader with work experience, and a group that does not, and this was a fundamental difference between the two groups and a further motivation to conduct a study on practicing software engineers rather than student surrogates in order to obtain a better understanding of software engineers' problem solving.

4.2 Data Collection

Data collection for the main study involved getting a commitment from potential participants and conducting the actual interview.

4.2.1 Getting Contacts and Commitment

The data collection processes started with obtaining contacts and securing participation commitments. The potential pool of interviewees were software engineers in general, although they may be appointed to more specific positions such as analyst, developer or programmer. Different organisations give different titles to their software engineers and this study was not concerned with different titles, but focused on staff that worked directly on software.

The approaches were made through personal contacts who are heads of information technology (IT) organisations or IT departments such as the IT department of a local

Table 1

Interview protocol

Knowledge of the Problem Domain 1. How much do you know about the domain of the system? 2. What do you think would be required in the system? 3. Does the perceived application requirement change? 4. What changed the perceived requirement? 5. How does your knowledge guide you in making mental models of the system? Information about the Problem 1. What do you ask during user-developer meetings? 2. What was the information regarding the problem that you obtained from material/document or other sources? 3. Do you collect any material/document from the user stating their requirements? 4. To what extent do your team members' thoughts influence your mental model? 5. Does the software development tools guide aid your software design task? Problem solving Task 1. Explain the types of tasks given to you in systems design. 2. What are the goals for your tasks? 3. How complex do you think the tasks are? 4. Are the tasks interconnected? 5. How do tasks assigned to you relate to those of your team members? Mental Processes to Form Solution 1. What do you do to form a mental model of the system? 2. What tools do you use to aid in the formation of a mental model? 3. How do you rely on existing internal and external representations of the problem to form a mental model of the system? 4. Explain your strategy for selecting the most representable mental model. 5. How do you utilise previous iterations of mental models to produce a better mental model? Success of Solution 1. Are the system's functionalities clear and easily understood? 2. How do the functionalities address the tasks of the user? 3. Is the system completed in time? 4. Are users satisfied with the system? 5. What concerns are raised regarding the system?
--

university, ABC Technologies³, DEF Ltd and a local city council. Meetings were held with the heads to explain the purpose of the research, to elicit cooperation on their part and to encourage their staff to participate. The department and organisation heads chose staff that they considered to be suitable for the project and ready to volunteer. After the first round of interviews with the volunteers, all of them were found to be suitable for the study, with the exception of those from the city council, because after the first interview with city council staff, it was found that they did not develop the software themselves, and their involvement was limited to the analysis and administration of the software projects. Hence, they were excluded from the study.

Personal contacts who are software engineers were also directly approached. These engineers worked autonomously within their organisations. Positive commitments were received from two engineers in the same local university: one engineer worked under an academic division, one engineer worked under a department of a different division.

More contacts were sought through the Yellow Pages to add to the number of participants. As a result of a search in the “information technology” heading, contact was made via email to various organisations and a positive response was obtained from GHI Payroll Ltd.

4.2.2 Interviews

A total of eleven software engineers were interviewed from six organisations. Twenty one interviews were conducted with them; every subject was interviewed twice except for one. In his case, after coding the interview transcript, it was found that the initial interview was sufficient, hence a second interview was not needed. Second interviews were held for all the other subjects as the subsequent codings of the first interview transcripts indicated more information was needed. A summary of the interviews is shown in Table 2.

Each interview was audio recorded. Two of the interviews suffered device error during the interviews. To remedy the problem, field notes were written for those interviews and were used to supplement the transcripts of the interviews. One of the field notes was prepared immediately after the interview from the interviewer’s memory to avoid losing the facts of the interview. This was written in the form of third person narrative. The other field notes were written during the interview in the form of a dialogue script, similar to transcripts of interview audio recordings. One interview recording was particularly inaudible when the interviewee was presenting a PowerPoint

³ Pseudonyms are used to protect the anonymity of the subjects and their organisations.

slide explaining the mechanics of his system. No field notes were prepared for this because it was impossible for the highly technical discussion to be recalled from the interviewer's memory.

The interviewees were asked questions as drafted in the protocol. Often the exact wordings of the protocol were not used as they were too formal, or could not be understood by the interviewees. The interviewer felt it was more natural to ask the subjects in more general terms and allow them to elaborate without restriction. Probing questions were used to dig further into their description. In order to assist the use of probing questions, the interviewees were requested to draw time lines of their task. They were also requested to provide copies of design documents that could be helpful to support analysis. However, the documents received were very few in number.

Some questions lead to very lengthy answers, thus it was found that the allocated interview time was insufficient. The subsequent interviews for each participant were used to ask the questions further questions on issues that were found to be interestingly salient. During the analysis of the interview transcripts, emails were sent to a few of the participants to clarify ambiguous coding.

Table 2
Interview summary

Participant's name Degree, Present position, Previous experience	Description of organisation	Interview number, length, pages, words (notes)
Jasmine Business Computing, Analyst, >15 yr in IT including 10 yr with present employer	A medium sized university IT department that supports the IT needs of a large university	1) 37 min, 7 pp, 3500 words 2) 17 min, 4 pp, 1400 words (excluding error in audio recording of second interview supplemented by field notes)
Rachel Computer Science, part time analyst, 11 yr in current position and 4 yr in own IT business in between	Same as Jasmine	1) 36 min, 7 pp, 2400 words 2) 29 min, 11 pp, 5000 words
Jessica Accounting and Information Science, unknown, 5 yr in current organisation	ABC Technologies is a small company that develops tailor made business systems for clients	1) 41 min, 12 pp, 3400 words 2) 29 min, 8 pp, 2400 words

Participant's name Degree, Present position, Previous experience	Description of organisation	Interview number, length, pages, words (notes)
John IT, Developer, 5 yr in IT companies with 3 yr in current organisation	Same as Jessica	1) 36 min, 5 pp, 2400 words 2) 51 min, 12 pp, 4100 words
Bill No degree, Senior Developer, 8 yr in current organisation and prior to that, 2 yr in accounting firm	Same as Jessica	1) 36 min, 9 pp, 4500 words 2) 44 min, 10 pp, 5000 words
Sam Computer Science, Lead Developer, 4 yr in current organisation and 2 yr in consulting company prior	DEF Ltd is a small company that develops tailor made business systems for clients	1) 30 min, 7 pp, 3700 words 2) 39 min, 9 pp, 5200 words
Rick Computer Science, Developer, 3.5 yr in current organisation	Same as Sam	1) 31 min, 9 pp, 2900 words 2) 19 min, 9 pp, 1900 words
Jack Philosophy, Web Analyst, 2 yr in current position and <5 yr in own web business prior	A medium sized academic division in a large university	1) 66 min, 8 pp, 3300 words (including a portion that was inaudible when he was explaining the technicality of his system) 2) 39 min, 11 pp, 4800 words
Barry Information Science, Technical support, 12 yr in current organisation	A small academic department in a large university	1) 73 min, 18 pp, 10100 words (no second interview)
Kyle No degree, Software Developer, 7 yr in IT and was a carpenter prior	GHI Payroll is a medium sized software development company that develops their own brand of payroll software	1) 51 min, 9 pp, 4600 words 2) 7 min, 3 pp, 700 words (excluding error in audio recording in the second interview supplemented by field notes)
Patrick Chemistry with one year of Computer Science, Senior Software Developer, 15 yr in IT, including 3 yr at current organisation, 10 yr as a scientist prior	Same as Kyle	1) 39 min, 11 pp, 4600 words 2) 19 min, 6 pp, 2100 words

Note: Participant and organisation names are pseudonyms.

4.3 Data Analysis

The analysis process proceeded from data collection. However, it was not a one way, mutually exclusive transition, which is more prevalent in quantitative data analysis. After some interviews were conducted, an initial analysis of the transcriptions was coded to identify interesting issues that would require further probing and to assist in the second round of interviews.

4.3.1 Transcribing the Interviews

Each interview recording was transcribed by listening to the audio recording, and then typing each sentence or partial sentence into text. After transcribing an answer to a question, the audio was relistened to in order to check for accuracy or to recognise some words which could not be made out in the first listening. An hour interview typically took at least six hours to transcribe, a rate which varied based on speech clarity. A recheck of the whole interview for accuracy was performed before coding.

4.3.2 Coding

The coding used in this study assigned meanings to sentences or a group of sentences. Initially texts were coded onto open codes. Coding was done using NVivo software and all interview transcripts were imported into NVivo and coding was done while reading a transcript. A meaningful sentence or sentences that conveyed a particular meaning were highlighted and assigned to a new code or a code created earlier. The new code would be given a descriptive name and these often became very long because the coded text had to convey complex meanings. After the first round of coding on all subjects, 435 codes were created, with the majority of them containing only one extract of text. This is the result of using open coding and is common occurrence among researchers using grounded theory. Appendix D shows a complete list of the initial codes and the corresponding number of sources (interview transcripts where the code is assigned) and number of references (extracts of text assigned to the code).

Coding involved the interpretation of the text by the researcher and this opened up the question of whether the assigned codings were accurate and consistent. To ensure consistency, coding was done iteratively with the initial coding of transcripts done after

the first round of interviews. While extracting relationships, further thought was given as to whether the code given to the resulting text was appropriate. If it was not, the text in the transcript was reassigned to a more appropriate code. Coding accuracy was ensured by engaging an independent coder to perform coding on a sample of the text, as discussed in Section 4.4.2. Codes used are open codes based on the data and not from the literature, hence restricting the possibility of preconceived ideas to contaminate the findings (addressing G4).

4.3.3 Categorising

Working with 435 codes would not allow theorising based on a conceptual view of the data. The raw codes were categorised according to similarities between the codes into 20 broad categories. At this point, some codes are grouped into various subcategories under the category of “Mental process”, while some were lumped under the main mental process category. These subcategories roughly resemble the final list of processes. Some codes were lumped into the main category of “Activity” that was later renamed “Development activity”.

After the second categorization exercise, all codes under the Mental Process category found the appropriate subcategory. And after the third round of categorising codes under Activities are grouped into subcategories. Besides Mental Process and Development Activity, other categories that have subcategories are; Knowledge and Experience, Solution Object, Information (later named Problem Information). Other categories do not contain subcategories.

The final list of categories, later in the thesis referred to as problem solving components, contain 12 categories. This was after continuous iteration done on the categories that include pulling some main categories to become a subcategory of another. For example the category “External Aid” has been moved to a position as a subcategory under Problem Information.

The continuous refinement of the categories based on the data from multiple respondents enhances the abstraction level of the categories, while being grounded on the specific experiences of the respondents (addressing G3).

4.3.4 Modelling Category Relationships

After codes are categorised, relationship codes between two categories are created. The method for finding the relationships is as follows:

- Drill down from each category into the individual codes and see if the codes pertain to actually two or more categories or subcategories. The open codes were originally created with the attitude that anything meaningful to the study should be coded, therefore some of the original codes already contained relationships to categories formed later.
- Drill down from each code to the extracts to see if the extracts actually relate to two or more categories or subcategories as part of the iterative coding.

When a relationship between two categories was found, it was put into a relationship code. The relationship type created included phrases such as, “about”, “as a basis for”, “a factor to consider in”, “is defined by”, “is a prerequisite to”, “output”, and “then”. These relationships indicated when two categories were related to each other merely due to the chronology of their occurrence, as a reliance, or through cause and effect. The categories and the relationships between them became the basis for the presentation of the findings in sections 5.2 and 5.3.

4.4 Trustworthiness of Data

Any research, quantitative or qualitative, must have a scientifically rigorous implementation of data collection and analysis in order to have meaningful results. However, the same verification techniques cannot be blindly applied across quantitative and qualitative research. For the purpose of this study, three main sources for ensuring rigour were referred to for three levels of method implementation: general implementation (Klein & Myers, 1999); interviewing (Myers & Newman, 2007) and theorising (Lee & Hubona, 2009). The itemised guidelines, principles and issues suggested by these authors are summarised and addressed in Table 3.

Table 3
Reflections on rigour

Code	Level, Source Principle/guideline/issue	Action
General Implementation, Klein and Myers (1999)		
G1	Contextualization	Findings are presented after each participant's context was laid down.
G2	Interaction between researchers and subjects	The researcher is aware that his interpretation may be influenced by his interaction with the respondents during the interview.
G3	Abstraction and generalisation	The continuous refinement of the categories based on the data from multiple respondents enhances the abstraction level of the categories, while being grounded on the specific experiences of the respondents
G4	Dialogical reasoning	A grounded theory approach undertaken without "seed" categories was used, limiting any forms of preconceived ideas that may contaminate the findings.
G5	Multiple interpretation	Interpretation was undertaken by the researcher and an independent coder. Different interpretations that lead to poor inter-rater reliability measures were reconciled.
G6	Suspicion	This principle was not addressed as there was no apparent reason to be suspicious of the respondents regarding professional related issues such as software engineering.
Interviewing, Myers and Newman (2007)		
I1	Situating the researcher as actor	Identifying the background of the researcher may not be appropriate in this thesis.
I2	Minimise social dissonance	The researcher introduced himself, his background and purpose of study to the interviewees to make them comfortable.
I3	Represent various 'voices'	Not implemented. The study specifically looked at the engineers point of view.
I4	Everyone is an interpreter	Interpretation was undertaken by the researcher and an independent coder. Different interpretations that lead to poor inter-rater reliability measures were reconciled.

Code	Level, Source Principle/guideline/issue	Action
I5	Use mirroring in questions and answers	Detailed questions were asked following protocol questions.
I6	Flexibility	When interviewees showed passion retelling their story, interviews were allowed to go over the original stipulated time.
I7	Confidentiality of disclosures	The participants signed the ethics and informed consent forms. Access to confidential materials was limited to the researcher and his supervisors. Pseudonyms were used in this thesis and in all other related publications.
Theorising, Lee and Hubona (2008)		
T1	Common scientific basis	Consistent and accurate interpretation was sought by using open codes, and conceptual categories from the codes went through inter-rater reliability testing.
T2	Fallacy of affirming the consequent	The study noted that the findings do not prove or disapprove established theory, rather the findings suggest new theory.
T3	Summative validity	Addressed to a limited extent by engaging an independent coder that holds a postgraduate qualification in IT, therefore getting professional feedback on the analysis performed

Klein and Myers (1999) suggested the hermeneutic circle as the core principle for conducting interpretive research⁴. The principle suggests that in order to understand human behaviour, an iterative understanding between individual parts and the whole part is developed. A full understanding of the social construct is understood when the big picture is used to understand the individual elements, whereas the elements are interpreted to form a bigger picture. Klein and Myers broke down this fundamental principle into six other principles: contextualisation, interaction between the researchers and the subjects, abstraction and generalisation, dialogical reasoning, multiple interpretations, and suspicion.

Principle one (G1) is that research should reflect on the social and historical context of the issue being studied which allows the reader to understand how the issue came about. Principle two (G2) states that the researcher should also be aware of and take into

⁴ As has been mentioned in Chapter 3 that this research is methodologically interpretive in nature, the use of Klein and Myers' (1991) principles for interpretive research is hence justified.

consideration that the interpretation of the social construct is affected by his interaction with the subjects. Principle three (G3) holds that concepts should be developed to explain the social issue in general and as interpreted from the detailed understanding of individual cases. Principle four (G4) requires that the researcher acknowledge and express any preconceptions he had prior to the research that may have prejudiced the way he approaches his research. The fifth principle (G5) asks for an interpretation of the issue from multiple stakeholders. Finally, the sixth principle (G6) suggests that the researcher be critical of the data by being sensitive to the existence of any biases from the subjects that are not immediately apparent (Klein & Myers, 1999). Klein and Myers noted, however, that this principle has detractors and applying should be at the researcher's discretion.

Myers and Newman (2007) presented seven general guidelines for using interviews as the main data collection method. The first (I1) is to identify the researcher's own position in the context of the study. Then the researcher should take care to minimise the reasons that might make the interviewee feel uncomfortable (I2). These two guidelines support the general principle of interaction between the researcher and the subject (Klein & Myers, 1999). The third guideline (I3) suggests that the researcher interview different types of stakeholders on the issue. The fourth guideline (I4) suggests the researcher should recognise that every stakeholder is an interpreter (which follows the principle of multiple interpretations, Klein & Myers 1999). The fifth guideline (I5) suggests that an answer from the interviewee should guide the next question, and may include the phrases used. Therefore, the interviewer should be flexible when asking questions and should consider the interviewee's attitude (I6). Finally, the researcher should take care with the confidentiality of the subjects (I7).

In addressing three main issues for the scientific rigour and relevance of IS studies, particularly when using qualitative methods, Lee and Hubona (2009) suggested a formal logic framework called MPMT (*modus ponens, modus tollens*). The three issues are: the common scientific basis between positivist and interpretive research, the fallacy of affirming the consequent, and summative validity.

Modus ponens is the logic rule containing a major premise of, "if p is true, then q is true"; a minor premise of, " p is true"; and a conclusion of, "therefore, q is true" (Lee & Hubona, 2009, p. 239); where q is a subset of p . *Modus tollens* is the logic rule containing a major premise of, "if p is true, then q is true"; a minor premise of, " q is not true"; and a conclusion of, "therefore, p is not true" (Lee and Hubona, p. 239).

Lee and Hubona (2009) suggested the application of *modus tollens* as a common scientific basis for positivist and interpretive research (T1). In statistically based positivist

research, the application is typically applied by testing a sample from a population, the q . In interpretive study, p represents, the "...reader's interpretation...of a text is a valid hermeneutic interpretation" (Lee & Hubona, p. 243); q represents, the "... reader's interpretation of any passage... or set of passages... will not give rise to a contradiction... with regard to the reader's interpretation of any other passage or set of passages in the text" (Lee and Hubona, p. 243). Therefore, when applying *modus tollens*, the major premise is that if the readers' interpretation of a text is valid, then his interpretation of any passage or passages will not give rise to a contradiction of his interpretation of any other passages in the text. If his interpretation of a new passage gives rise to a contradiction, his interpretation of the text is not a valid interpretation.

The second issue is regarding the fallacy of affirming the consequent (T2). This fallacy occurs when the researcher finds q is true and concludes that therefore p is true. Since q is an instance of p , a true q does not necessarily mean p is true in all cases. The best that could be concluded is that the researcher has not rejected the possibility of the theory being true (Lee & Hubona, 2009). However, scientific reasoning can be applied to reject a theory when q is found to be false.

The third issue is the summative validity (T3). Lee and Hubona (2009) distinguished summative validity from formative validity by giving the analogy of baking a cake. When a baker follows the recipe correctly, he achieves formative validity. This, however, is not enough. After the cake is baked, he should taste it to make sure the cake was successfully baked, and this is summative validity. In a positivist research, formative validity is achieved by correct sampling of examples. Lee and Hubona suggested that summative validity could be tested by plugging in an out-of-sample data into the mathematical model that was posed from the original sample. Summative validity is achieved when the observed value of the new data is within a prediction interval suggested by the model.

In an interpretive study, formative validity is achieved by various techniques normally used in qualitative research and summative validity is achieved when the theory formed is understood and agreed upon by the subjects themselves (Lee & Hubona, 2009).

Several steps were taken to ensure the rigour of the study. Although most of the steps have already been described in previous sections, they are reiterated here for clarity and summarised in Table 3 where they are grouped into the phase of the study where they occurred. The three phases of the study are interviews, coding and results.

4.4.1 Interviews

The main concern with rigour was about conducting the interviews so that the objective of the study was addressed. In order to address this concern, semi-structured interviews were conducted. Protocols were developed by applying them to the small scale study. Interviewees were asked to recall a challenging decision making incident and to relate a general account of it, draw a time line and detail how each decision was made at each point on the time line.

Before the interview started, the interviewer introduced himself, shared his educational background and explained the objective of the research (addressing I2). Then the interviewee was asked to select a systems task that he/she considered to be important, with the instruction that it should be recent enough to be fully recalled. System tasks were typically longer and not critical, therefore it was important to ask the interviewees to select a system task that was important and easily recalled.

Next the interviewee was asked for a general overview of the system task and context. Then he/she was asked to draw a time line to show the flow of processes he/she went through. Later he/she was asked questions similar to the protocol questions design. This was to ensure that the discussion avoided “site effect on researcher” (Miles & Huberman, 1984) by sticking to the objectives of the study and to use limited interview time efficiently. However, some interviews extended over the stipulated time when interviewees appeared to become passionate as the interview progressed (addressing I6, see Section 4.4 for an explanation of the guideline). Additionally, the use of a retrospective interview avoided “researcher effect on the site” (Miles & Huberman, 1984). Another step taken to reduce researcher effect on the site was achieved by informing the interviewee of the exact objective of the study. Based on the answers to the protocol questions, further questions were asked in order to explore details (addressing I5).

The next concern was the accuracy of the interview transcription and this was addressed during the interviews by using audio recording to facilitate a verbatim transcription. Field notes were prepared to supplement the audio recording to cover for circumstances where errors and deficiencies in the recording occurred.

The transcription of the recordings was done iteratively to ensure the transcripts mirrored the audio recordings. An audio recording was relistened to both bit by bit and as a whole. After a part of the sentence was listened to and transcribed, it was relistened to in order to check the accuracy of the transcription and to clarify some words that were not recognised previously. Sometimes bits were re-listened to more than once. Similarly, after the whole recording was transcribed, it was relistened to to verify or amend the

transcription.

4.4.2 Inter-rater Reliability

An inter-rater reliability test was conducted to ensure the validity of the coding and categorisation ascribed by the researcher. In general terms, an inter-rater reliability test measures how two coders agree on the codes assigned to text. The higher the agreement, the higher the reliability coefficient.

In NVivo, a function to calculate inter-reliability exists in the form of Cohen's Kappa. Whereas simple agreement-disagreement percentages show the extent of coders' agreements, they do not take into account the probability of agreement. Cohen's Kappa takes this probability into account, hence it more accurately represents an agreement between the coders.

For practical reasons, an inter-reliability test on the complete data set was not conducted because of the sheer volume of data. However, a sample of the data set was used for the test. In particular, an independent coder (IC) was engaged to apply coding on to the interview transcripts of four randomly selected interviewees. The IC just recently graduated with a Masters in IT and therefore is expected to be well-informed of the nature of software engineering work (partially addressing T3). Coding was done using the final categories that were identified during the process undertaken as described in Section 4.3.3.

Cohen's Kappa was obtained for each of the subcategories and categories (when there was no subcategory) for each interview transcript. The result was exported from NVivo into an electronic spreadsheet to calculate the aggregate Kappa for each subcategory for all four engineers. The threshold for the measures were equal or above 0.62 indicate substantial agreement, and below 0.45 is poor (El Emam, 1999).

The result was a mixture of perfect, substantial, moderate, poor, and no agreements. There are instances when Kappa is a perfect 1. However upon close examination, neither the researcher nor the independent coder coded any text from the transcripts of the four subjects into that particular category. By default, Kappa would be 1 when this occurs.

To proceed with a valid presentation of the results, those categories that received a poor Kappa measure were looked at again. The researcher and the IC discussed each of the identified categories to reaffirm the definition of the category, and looked at whether the text coded fit the intended definition of the category. In most of the categories, a discussion between the researcher and the IC resulted in either or both of them reassigning the text to the appropriate categories. This resulted in the Kappa improving

to at least the “good” level (hence addressing G5, I4, T1). In the rare cases where no significant improvement was achieved, the category was left out of the presentation of the results. Appendix E shows the Kappa coefficients of each category/sub-category before and after reconciliation among the coders.

4.4.3 Findings

The findings of the study are presented with the participants’ contexts laid out in order to facilitate an understanding of the specific details based on a general understanding of the participants’ environment (addressing G2). This also allowed the researcher to interpret the details not merely in isolation, but with a full comprehension of each underlying issue (addressing G1).

There were several issues that could have hampered the validity of the findings. One question arose as to whether the problem solving practices of all system developers can be represented by results from the eleven engineers interviewed. To address this, it should be noted that this study did not intend to generalise the findings for other engineers, rather the intention was to investigate what was happening within the chosen sample and to find some patterns there. The patterns that emerged from the data are discussed in more general terms to allow them to be tested in future studies (addressing G4). The engineers involved in the study were from diverse backgrounds and worked in diverse contexts. Although the study began with a convenience sampling, i.e. personal contacts, the study continued to recruit developers through the Yellow Pages. The eleven developers came from small to large organisations and although most of them worked in teams and under direct instruction from a team leader or departmental head, some developers worked autonomously on the system task or at least worked without direct instruction or supervision. The task contexts of the developers also included successful and less successful systems.

It should be noted that the study’s findings do not reject or confirm the propositions’ of existing theories. Either of these conclusions would lead the thesis into the issue of fallacy by affirming the consequent (addressing T2). Nonetheless, this suggests a development of a new theory (see Chapter 6). Another issue is auditability. Verification of this study’s findings was facilitated by the incorporation of the interview texts whenever an assertion was made (see Chapter 5). Furthermore, an analysis of the steps taken to arrive at the findings was explained in Section 4.3 together with sample working tables. Other working tables are included in the appendices.

4.5 Ethical Consideration

Strict ethical consideration was adhered to in the study. Before getting commitments from participants, they were supplied with information on the study and the ethical issues addressed. The university's ethical consent form was reviewed and signed by the interviewees prior to the commencement of the interviews. Particular care was taken with the confidentiality of the participants such that only the researcher and his supervisors had access to audio recordings and transcripts. A sample of the information and consent form is given in Appendix E. Names of individuals and organisations in the thesis are replaced with pseudonyms or referred to in general terms (addressing I7).

4.6 Summary

This chapter included a discussion on how the data was collected and analysed. Trustworthiness and ethical issues of the data were also addressed, and are summarised in Table 3. The next chapter will discuss the results.

Chapter 5

RESULTS

The results are organised into three parts. The first part explains the context in which the engineers worked. The second part explains the mental processes they went through to solve their problems and how those processes interact with other components of problem solving. The third part discusses the other components in detail.

As an introduction to the results, this study found 14 mental processes that engineers go through in problem solving. Besides the mental processes, other important components of problem solving were found: development activities, solution object, solution success, problem information, knowledge and experience, organisation and structure, tools and technology, iteration, user understanding and communication, and team collaboration.

5.1 Overview of Participants' Task Context

A software engineer's task was selected based on its importance and how easily it was recalled. One task was discussed in detail by each interviewee and here they are presented to familiarise the reader with the task context before a more detailed discussion on the problem solving components in Sections 5.2 to 5.3. Each software engineer's use of the problem solving components are also described in brief, and modeled in a pictorial form in Appendix C.

5.1.1 Jasmine

Jasmine's task was to develop an online graduation application form and to bring both the form and the graduation ceremony database into the Oracle database. The project started in May 2008 and at the time of interview, Jasmine expected to hear the decision on funding approval in May 2009. Jasmine was the main person involved in the analyses, and had meetings with stakeholders. She indicated that she might be getting programmers involved at a later stage. At the last interview, she had just produced a project proposal document and was waiting for signoff. The database created required eighteen new database tables. The task required an understanding of

the application process for participation in the graduation ceremony by different student groups (undergraduate to PhD and certificate). While analysing current practice, she found some weaknesses in the process and discussed them with people involved in those processes. In terms of familiarity with the task, she had been previously involved in the migration of a related application, and therefore was familiar with the business processes involved in that application. The graduation ceremony application, however, involved aspects of business processes that she had not previously encountered.

Jasmine prepared a specification or requirements document, which identified problem areas to be addressed or improved. Before starting to build the system, she needed to understand the existing system on which the new system would be built and what would require modification within the old system.

Jasmine had previous experience in the migration of the graduation system, which aided her process of understanding but she needed to understand the current business process first. This understanding added to her application domain knowledge, and was used as the basis for design particularly the design of the database structure.

In thinking about the solution proposal, Jasmine referred to problem representation. She was told that some redundancies occur in the user tasks, and she envisioned a solution that eliminates this redundancy. Documents prepared by the user department represent what the current system does and what the users require the new system to achieve.

Designing was aided by the process of externalising components of the interface and this allowed her to focus her attention on the design. The resulting proposal was a database structure very different from the structure of the existing system.

5.1.2 Rachel

Rachel's task was to modify a module on program and paper maintenance. Rachel was the only person involved in the project, which was passed to her from a different analyst who had worked on it. She received the small project in June 08 and implementation was in October 2008. The request was to simplify an online form by moving a section on restricted papers to a column in another section, to remove the fields from the corresponding database table and to modify another table. While analysing the relationships, Rachel found that the users' request regarding the database table was inappropriate as the database would have lost a lot of information if the one-many relationship, as modelled in the original system, was implemented as a one-one relationship. Her solution was to maintain the table relationship but to modify the interface to look more

like a one–one relationship. To Rachel, the complexity of the task was, “On a scale of 1 to 10, it would be about 6.” Part of the complexity was attributed to the fact she took over another person’s task, so the task was not hers from the start. Thus she had to understand the request as well as the on-working solution which lacked documentation. There was no negative feedback for her solution and it has already been implemented.

To start the task, Rachel needed to understand the interim solution attempted by the previous engineer as well as the client’s request. To assist with her understanding of the existing system, Rachel needed to go through the process of tracing and externalising. Based on the database tables, she drew the tables in diagrams similar to entity–relationship diagrams to understand the relationship between the tables. Externalising was used to attain understanding of the big picture of the whole system, a prerequisite to starting with the system build, specifically programming.

Rachel evaluated two main solution options; following the client request by modifying the one–many to one–one database table relationship, or maintaining the one–many relationship. Rachel needed to evaluate the user’s request because based on her understanding, the request seemed to be incongruent with the goal of the system, and the client did not understand their own needs and the technical aspects of the database structure of the system. Only after evaluating the request could she proceed with building the necessary changes to the system.

To understand the specific workings of detailed components of the system, she used the tracing process. The tracing process involves using the system to see what it does, and changing the codes to see the effects from the changes.

From the start, while preparing the specifications document (informally), Rachel broke down the requirements into small pieces and worked on them in turn, particularly the easier-to-implement requirements first.

In order to fulfil the user’s needs while keeping the original database structure intact to reserve the power of the data, Rachel had to compromise between the user’s needs with the fundamentals of the database technology. She modified the interface such that it shows as if the data is only in one table, through the use of an on/off button.

Rachel explained to the user why their request could not be followed and why they reached a compromised solution.

Rachel used tools or technology in two ways to assist her mental processing. She used Application Express (APEX) to design the user interface of the system and to trace how the system works.

5.1.3 Jessica

Jessica's task was to migrate the financial system of a client, JKL Poultry, to a Microsoft Dynamics AX enterprise resource planning (ERP) system. Several staff members were involved, and Jessica worked on the design and implementation of the project. Other staff, such as the project manager and functional consultants, were involved in the analysis. Jessica had to customize some elements of the system, such as reports and forms in accounts payable, accounts receivable and inventory management and the general ledger, to fit with the practices of the company. She also had to develop other elements, such as new fields, pop-up screens and warning messages that were not available in the previous system. The project went through a lot of iterations; some after the system has already been implemented. Some small changes were lumped into one iteration, although each change may have gone through its own iteration. Other changes were due to the user's request not being fully understood. Jessica's task was a mixture of simple and complex subtasks. The project took six months to complete.

Jessica's first mental process was to understand the problem by externalising its representation in a readable form based on the user requirements gathered and communicated by consultants.

Once some level of understanding of the problem was reached, Jessica brainstormed the idea for possible solutions. She evaluated each specific client request and the solution options to determine whether requests could be built into the system and which solutions would fulfil the requirements.

Evaluating client requests and solution options was done prior to building the system. The evaluation criteria included the functionality of option, cost and delivery time. The evaluated and selected solution options were used as the basis for design. Before incorporating the design onto the system, she drafted the design on a paper in order to visualise it first. In the process of designing the interface, Jessica used information about the business process.

Her development work involved customising and adding new functionality to the existing technology that Jessica and her team chose to work with: the Dynamic AX ERP system. Based on requirements she customised the AX system. The customization was verified from other members of Jessica's organisation in order to ascertain feasibility of the work.

Jessica needed to learn about Dynamic AX in order to use it as the technology behind the system. External sources of information that she sought included manuals, online forums and training. An effective way of learn and understanding the technology is to use it and see what it does or achieve.

When a solution proposal was not feasible, probably due to technical or technological limitations, Jessica had to view the problem differently and suggest a work-around. The work-around solution was produced after making compromises between the user's request and the limitations of the technology. The solution was then tested, the units and systems testing, before implementation on the client's site.

Use of certain technology both helped and restricted Jessica's task. As mentioned earlier, the restrictions of Dynamics AX forced Jessica to view the problem differently and conduct a work-around. Specifically, two limitations were mentioned; a less flexible query capability and the lack of a graph generator. In the latter case, Jessica built reports with tables instead of graphs.

On the other hand, the technology enabled her task in other ways. Generally, the availability of Dynamics AX as a customizable and modifiable ERP system released her and her team from the task of building an ERP system from scratch.

Iteration occurred on several objects in Jessica's development task. Firstly, the client requests went through iterations. Due to changing user requests, the end solution went through iterations as well. In fact, each iteration increases the complexity and refines the end solution.

5.1.4 John

John's task was to modify a restocking application of Accpac system for a client, MNO Hospital. The modification was done to reduce the processing of items transferred by eliminating steps to improve performance. This included combining three modules into one. John was solely responsible for the development, and other staff in the organisation were involved in securing the sale and designing more generally how the current software needed to be modified. John was involved in suggesting modifications on a more detailed level. John's task involve the activity of customization of Accpac system to suit the requirement of the client. The initial analysis and design was conducted in the middle of 2008 and John was involved in the project from August until its completion in October 2008. Design was done collaboratively with another developer who actually talked to the clients regarding their needs. However, there were still requests for changes coming through after the completion. The work was on an interface, and no changes to the underlying Accpac database were needed. To John, the complexity of his task was, "from 1 to 10, I'd say 7" due to limitations in Accpac and the need to use third party modules. Understanding what the users wanted was not clear, because he admitted that he did not fully understand this until he actually watched the user operating the system.

He started off with a more general solution, and each iteration was made to narrow down the system to suit the users' requirements. In general, the users were satisfied with the system although there was one issue that could not be addressed due to time and budget constraints.

Firstly, John needed to understand the requirements and business process in terms of how the user uses the system: how often the systems are used and how many users at any one time and the load on the system. John evaluated how difficult the client request was before start building it. There were instances when John had to extrapolate on the unit used for processing shipping in the system. His assumption was that each box handled is considered one unit. This assumption was based on his experience in similar projects. However, his assumption resulted in an inaccurate end solution. He found out that sometimes less than a whole box was shipped.

The end solution went through several iterations which increase its complexity and refines it. There were also instances where the end solutions triggered changes in the client's request. One example was when the user tried out the solution, it became apparent that the user wanted to just key-in certain primary keys and then press tab for the system to show the details of the particular record. Such client requests and others became the basis for building the system. And because the requirements were not detailed enough at the beginning, John's application domain knowledge did not reach that level of detail, and more iterations were required to reach a solution.

Sometimes the client request was not practical enough to be implemented, so John had to view the problem differently and suggest a work-around to the problem. This necessitated compromises because the Accpac technology had limitations regarding the modification of the core database tables. One limitation that occurs very occasionally just could not be properly categorised in the system and the corresponding document could not be generated. To do the modification to cater for this would have required significant additional time and hence a significant extra charge to the client. John proposed a way for the client to document that type of transfer and prepare the document manually.

In the process of design of the conceptual level, John relied on his experience with similar systems. Conceptually, such systems go through the process of capturing, validating and processing the data. Design was done collaboratively with another developer who actually talked to the clients regarding their needs.

John used another tool, a third party module, on top of Accpac. John had difficulty using the tool because he had no knowledge and training with it. When John could not solve a particular problem, he would externalise the problem verbally and this helped him find the solution to the problem.

5.1.5 Bill

Bill's task was to install a Dynamics AX system for a client, PQR, and develop in the solution an application for time-sheet approval which was lacking in the standard AX system. The solution required an environment where the database could be used by many users in the organisation while reducing the impact of user licensing. To achieve this, a web-based access to the AX system was developed. His team used Microsoft Outlook for the time-sheet application and this was integrated into the AX system. The project started in April 2007 and the production system went online at end of that year. Bill had been involved with internal projects regarding time-sheets, although it was not a web-based implementation. In the time-work allocation aspect of the time sheet approval, he had to prioritise the client's request over the accounting requirements. The client request was to simplify the time-cost allocation against accounting standards.

Bill and several members worked together on the development of the database design, but he was in charge of the web development. Because of time demands, the system went live in an incomplete state. Some of the problems required just tweaks but others were not worth fixing as they would have required a substantial time investment to solve and a quick manual fix worked well. Some problems, however, necessitated a new request after a year.

Other members of his team sold the Dynamic AX technology to the client. Once the client agreed to the Dynamic AX as a concept, Bill and other developers were given the task to refine and develop the system to address the specific needs of the client.

To design the system, Bill relied upon his previous experience in time-sheet systems in his own company, as well as the requirements set early in the project in the request for proposal. Design of the concept and the underlying database was done collaboratively with other team members, before segregation of parts of the system, where Bill was assigned with the design and development of the website where users would enter time-sheet information.

The earlier phase of the website development involved designing the mock-up screenshot and interactive visual demo of the website, without working data in the background. The client's requirements were not detailed enough to design the interface, therefore, Bill had to extrapolate from his experience as to how the website should work. The website was designed based on assumptions and the available requirements later verified with the user, resulting in some throwaways.

When the design was laid out, it helped Bill to understand it better. In addition, after they started to build the system, Bill and his teammates were given specifics that gave them a better understanding of what they each had to do. To understand the effect of

specific programming codes, Bill traced the changes to codes and saw the output of the change. Bill needed to understand the business process of the user; when a time-sheet was submitted, when the approval takes place, which approval would generate a journal entry, and at what stage journaled invoices get produced.

When asked about whether Bill brainstormed his ideas for a solution proposal, he mentioned that the constraint of delivery time restricted the extent of brainstorming. Only one particular solution option was identified and Bill then focused on that option. In particular, he selected Outlook for managing the timesheet data as a default and no other alternative was considered. In focusing this way, Bill simulated in his mind the processes that the software option goes through.

Bill made certain compromises when the user's request was incongruent with the accounting aspects of the system. When time charged to the user's client was unequal to the total value of time worked by each worker, the difference would need to be allocated across each worker. The user did not want to have to deal with how that allocation was made. Bill's compromise was that the system would make the allocation automatically based on a certain rate of cost and this compromise was built into the end solution.

One of the processes Bill went through was breaking down the system. Working with his teammates, he broke the system down into three main components; the interface in the form of a website, the database and the web service where the interface was tied to the database. Once a certain aspect of design was built and was solid enough, Bill cleaned and standardised the codes.

The end solution went through several iterations and was tested. However, the end solution implemented at the user's site revealed missing functionality that had not been specified for and built regarding undoing time-sheet posts and generated invoices. This new request from the client could not be worked into the system as it required a substantial amount of time and hence extra cost to the user. Rather a work-around was proposed. The work-around required the user to contact the developer when a need arose for an undo, and the developer would remove the related records through direct access to the database and not via the web interface. The new request would become a requirement for a future potential upgrade of the system.

5.1.6 Sam

Sam's task was to develop a securitisation system in place of a legacy system for a client involved in hire purchase and lease. The system was about managing an aggregation of accounts receivables as tranches and securing them to financiers to obtain cash for

working capital. Apart from migrating to a new system, the task also necessitated changes to the back-end processing of the receivables, such as adding one-off rollups and a monthly aggregation. The project was initiated in February 2008 and Sam became involved in March. There was no proper gathering of requirements and a major problem was the client's lack of knowledge of their own back-end process. The client was also unable to provide documentation of the existing system early in the project. As the lead developer, Sam was involved with other developers, while other staff was involved in selling the concept. There were many delays and the system was not delivered within the time frame. Sam developed a system that was flexible and functioned well but because the basic algorithms of the processes were not supplied by the clients, it was inaccurate. Sam left the company in February 2009 when the project had not been completed⁵. Although the original request for the system was to concentrate on the back-end processing, after a working system was presented to the client they then requested a good interface.

Sam was tasked to break down the systems development project and later he prepared the requirements document. Sam went through a substantial amount of effort to understand the client's business process. In trying to understand the business process, he viewed it in terms of the nouns, verbs and the dependencies between them. In fact in order to better understand their business practices, (he later found that the users themselves found this difficult), he did research on them using external sources such as Wikipedia.

Because the users did not themselves understand integral calculation methods for running their business, their requirements were insufficient to the task, and this adversely affected the accuracy of the end solution. However, the users were surprisingly satisfied with how the system worked. To the extent that previously they had not worked much with the interface, but then requested that it be more aesthetically pleasing so the users' requests and requirements went through several iterations. Each iteration refined the end solution, while the development team obtained the correct calculation formulae. In the time leading up to the end solution however the users were generally dissatisfied because their targeted delivery time was not met.

The possible reason for mixed user satisfaction was that Sam broke the system down into very small components and focussed on each component's programming code or scripts in turn to produce the best component, despite the serious deficiency in the users' requirements. While focusing on and building the core architecture of the system, he also allowed the architecture to cater for changes that he expected would eventually be

⁵ Rick, who was working with Sam, saw through the completion of the project.

uncovered. He conducted unit tests on each component to ensure each they worked well, regardless of the other components. He also drew diagrams of dependencies and paths on paper as a way of externalising the problems.

When an instance of his solution proposal cannot be solved, Sam tried to look at the problem differently. He acknowledges the need for having a broad perspective in looking at the problem. And that he needed to view the proposed system from two different perspectives; as a user and as a developer.

Another factor for good quality on some components of the system was collaboration. When Sam or other developer worked on a code, they got another developer who worked on similar code before to comment. The output also become more understandable than when only one developer worked on a component without others verifying it. This also facilitated learning on the part of the developer who prepared the code. He acknowledges that his team members had close working relationship together.

5.1.7 Rick

Rick was involved in the development of the same project as Sam but he concentrated on the detailed programming, while Sam was also involved in higher level design. Rick worked on the development for four to five months and had to learn about amortisation. Ninety percent of the help he needed from other staff concerned business logic and ten percent was technical help. Rick did not consider his tasks on the project to be complex. The system was implemented several months late, and as yet, there has been no feedback. The delay in completion was partly due to one of the important modules needing additional documentation from the supplier of the original system. After the documents were obtained, the system worked correctly.

Rick talked to the users to fully understand their core business practices and how they worked. Talking to Sam for between 10 to 20 minutes helped Rick understand what he was supposed to do. Because there was insufficient problem representation, particularly regarding the calculation method, Rick and Sam needed to figure out the amortisation process and the required calculations. When the solution did not function, Rick went back to investigate the causes of the problem by breaking down the task into activities and estimating the time needed for each activity. To do this, he relied on his experience and simulated in his mind those processes and the time they would require. These process steps are similar across systems, particularly at low level coding. He was confident that his programming knowledge helped substantially with this.

When asked whether he brainstormed ideas and evaluated them in turn, he said he

would use this strategy if performance was the main factor. When functionality is the factor, he focuses on only one option and works on that. When an idea just does not work and after trying to diagnose the problem he does not succeed, he would throw away the idea and view the problem differently.

Rick always kept the delivery date in mind and his concern was justified when the solution was delayed due to the insufficient requirements (as mentioned in Sam's case).

5.1.8 Jack

Jack's task was to create a portal through which each staff member of his division could build and manage their own profile webpages with content from an existing university database and from manual input. The task originated from a need to develop a content management system (CMS), but expanded over time. Jack started as a co-developer of the system, but later on, stakeholders and external technical experts were brought in to work on the project for a limited time. The project started in June 2007 when Jack began learning the Zend framework that he would use for the portal. He worked on the CMS portion in the earlier period until February 2008 when he started on the MyProfile module. There was no formal gathering of requirements from the client and requests from users were added during the development period. As Jack had been involved with users for tasks external to the system, in an attempt to solve their problems, he incorporated those tasks in this project. The whole system was named Bellacms. Some time after October 2008, with the addition of a user and two technical people to the team (one from the framework company, one a university PhD student in Information Science), the focus of the system shifted to a module called MyResearch. This module was chosen to serve the purpose of supplying required information for the 2012 Performance-based Research Fund (PBRF)⁶ round, and to replace the Research-Master system used by the publications office. The system has been successful in terms of doing what it was supposed to do as required by the university, but Jack felt it was a failure in terms of what he was originally aiming for, which was developing a whole portal. He felt that the task was very complex.

In going into a solution approach, Jack evaluated several CMSs in terms of their functionality as a basis. Because none offered the functionality Jack required, Jack decided to build a CMS in-house.

The process of understanding was undertaken by Jack on numerous occasions. While

⁶ PBRF is a research fund provided by the New Zealand government to research institutions in the country. The amount allocated to a particular institution in any particular round is determined by the research performance of the institution.

he was working alone, he became familiar with the various modules or problems. When working with his team, he worked to gain a unified understanding of concepts among his team members.

Each problem was defined and then ranked by the team in relation to each other. Ranking was externalised on a whiteboard using sticky notes. Each sticky was focused on in turn based on the order of priority, and a sticky that was down on the priority list would only be worked on after the sticky above it was addressed.

Every member of the team would go back and come up with a solution to the sticky, and later rediscuss it with the team. After team members came together to discuss their proposed solutions, Jack needed to understand each of their proposals.

One factors that made Jack to decide on a particular technology was the availability of external help as well as collaborative work with the builders of that technology. Specifically, besides fitting the need for a dynamic website as the end solution, he chose PHP partly because the sheer amount of people using it meant that there was a vast amount of information and guidance available on the language.. He chose Zend Framework for the portal development partly because of the direct support from Zend Technologies themselves who attached one of their own developers to the project, and partly because Jack had previous experience with the framework.

Jack went through the process of breaking down the solution proposal into modules and then into smaller units such as functionality. To accomplish this task, Jack used a decoupling process where segments are determined by how segment is able to execute a task independently as well as with other segments.

In executing his task, Jack went through iterations on the client's request. Each request changed the goal and requirements of the system, such that Jack felt they were experiencing scope creep. However, through collaborative work with an end-user, Jack managed to obtain a clear source for representation of the requirements and business processes.

5.1.9 Barry

Barry's task was to create a web-based academic survey for Ali, a PhD student in his academic department. The data collected from the survey would be stored in the university database. The student supplied a list of questions, and Barry designed the web pages in consultation with him. The student also needed to be able to use Java. Barry proceeded with the request even though he disagreed with the use of Java as it would limit the amount of people who could participate in the survey. The web survey

was completed on time and worked well.

Barry started off with the process of understanding the problem representation, specifically a list of survey questions showing the type of answers expected for each question. He then broke down the questions into groups because some questions were related and dependent on other questions.

Next, Barry decided on the option of building the web survey from the ground up because available customizable web survey tools could not be modified to address the dynamic nature of the survey, and to ensure that the responses data would be kept within the university premises. And by using basic html, Barry's solution proposal was not influenced by the limitation of any specific technology or tools. For Ali's survey, Barry did not develop the web interface and the database in tandem rather he developed the web interface first until it was agreed on by Ali and his supervisors, following the specific request of the latter.

True to form, the web interface went through numerous iterations before getting agreement from all parties. Even the questions went through iterations, including the identification of issues highlighted by Barry regarding the reasonableness of certain questions. The not-yet refined problem representation triggered communication to the client to verify and rectify what was specified. Due to Barry's experience in developing other academic web-surveys, he put forth his concerns regarding the defective problem representation to Ali. In general, if the problem representation was solid enough, Barry could design the database part of the survey without much difficulty. In cases where Barry faced a survey question that could be treated differently in terms of ways answers could be given, Barry would not make an assumption on the correct type of response, but would rather verify with Ali which type of response Ali was expecting.

Barry designed the front-end of the website following Ali's request, even though he did not want to use Java because it had the potential to limit the number of respondents to the survey. Barry was not familiar with the specific Java script to achieve the desired result and sought online guidance. When one version of solution interface was shown to Ali, it triggered a new request from Ali for modification to the survey, and each iteration refined the end solution.

5.1.10 Kyle

Kyle's task was to create an application for system implementers to import payroll and award rules from one GHI Payroll database to another. For example, when GHI Payroll software needs to be installed on a client's premises, the consultants will import

payroll rules in GHI Payroll software from another client and then adjust them to suit the new client. There had already been an attempt at an application to do this but it was unsuccessful. Kyle reviewed it and took five months to create a new application with better functionality. He was the only developer, although he talked to consultants and software architects to ascertain their requirements. The application was completed to the satisfaction of the stakeholders.

Kyle started off with the process of understanding the requirements of the application and the interim solution that had already been elsewhere. The idea of the previous interim solution was to import payroll rules from one database directly into another database. It did not work.

Collaborating with the consultants and the software architect, the requirements were refined to represent the needs of the application. Although Kyle's knowledge of the software was very limited, by talking to other developers in his organisation, he learned about payroll software in general to understand the purpose of the application he was working on in relation to the main software. His acquired knowledge was only on a small portion of the software and at the database table level. To come up with a solution proposal, Kyle viewed the problem differently than the previous developer and proposed a solution with an interim step: he imported payroll rules from a database into a file and then imported the rules from the file into another database.

Before coming up with the solution concept, Kyle went through the process of breaking down the interim solution to understand it in more detail. Then he focused on the details of the interim solution and the proposed solution. When confronted with problems in building the application he would brainstorm ideas on how to solve it, and focus on one idea which he thought might work. He would simulate in his mind how the idea would work and write up corresponding codes to test and verify the idea. If the idea did not work, he would turn to the next idea he had brainstormed earlier.

When designing the user interface for the application, due to the lack of detail with the user's requirements, he would extrapolate on what could be a good interface, and start designing based on that assumption. When the design was shown to the architect, the architect disagreed with the design and gave suggestions on how he wanted it to work, and Kyle went back to refine and modify the design.

Through collaborative work with the stakeholders and iterations of the requirements which improved the satisfaction of the stakeholders, Kyle designed a successful application. However because there were numerous iterations and frequent changes were requested by the stakeholders, the delivery time was delayed.

When conducting unit tests on the system, Kyle used NMock framework to mock

components dependent on the application. Kyle found restrictions in the framework in what it can and cannot mock, but he also found that such restrictions favourably affected the quality of his tests.

5.1.11 Patrick

Patrick's task was to port the pay processing module in the GHI Payroll software from Delphi to a C# platform to improve processing speed and increase flexibility for future changes. There was no additional functionality required from the new module and the interface was to be kept exactly the same. The project took 18 months. To be able to execute the port, changes had to be made on the architectural framework, so much of the task involved understanding the current module. Three to four people worked on the analysis and design, and on average six people worked on porting, although sometimes up to twelve people were working on the porting during one particular period. Patrick was involved in managing the distribution of the task and the reviews and he also worked on the porting. Although it was a big and lengthy job that required a large team, Patrick did not consider the task to be too complex as it was a direct port and the aim was very clear. Most of the complexity centred on the modification of the required platform. The porting of the process pay module was completed on time and is currently running well internally and clients are able to upgrade to it. At the time it was completed, however, due to insufficient testing, it still had issues.

Coming into the project, Patrick had some domain knowledge of the payroll software from previous experience debugging and software enhancement, as well as from working with people who had worked on the software for a longer time. Patrick was given responsibility of managing all staff on the project but before assigning activities to staff, he had to break down the whole project.

In order to start building the module, Patrick needed to understand the mechanics of the existing pay processing module, particularly the codes written in Delphi. Patrick externalised what he understood from the existing codes into model diagrams that showed the 50 steps necessary to execute the pay processing. There were also documents available that contained the representation of the pay processing module in Delphi.

Before the porting could be conducted, the framework used within C# had to be modified and extended. In certain issues with the framework, the team needed to view the framework differently by working around it, writing codes differently, or modifying the software to work around the framework.

While coding in C# to cover Delphi codes that do the same thing, Patrick went

through the process of refactoring the Delphi codes and this factoring process went through iterations. After one refactoring round was made, reading the codes revealed more segments that could be refactored, and hence there was another round of refactoring. During each iteration, he relied on his knowledge of the Delphi code as a basis for the refactoring.

Once Patrick and other programmers did the coding, Patrick arranged for the programmers, other than the ones who wrote the code, to do a code review. In performing his part of reviewing other programmer's codes, Patrick simulated in his mind how the code would work in order to confirm that the code would do what it was intended for.

After the building of the C#-based pay processing module was completed, a third party tool was used to help conduct testing of the module. Tools were also used to assist in the coding activity, including the highlighting of code errors and the use of suggestions for alternative codes. However, the tests were not sufficient such that bugs were found after the new module was implemented in the software.

In general, when pursuing a build, Patrick would work on the more obvious option. Due to the demands of the delivery time, Patrick chose this work method rather than brainstorming all the possible options first and evaluating them in turn.

This task lead to Patrick learning C# as he did not have substantial knowledge of the language.

5.2 Mental Processes

The first research question of the study is "What are the mental processes software engineers go through to solve problems?" Analysis on the interviews conducted on the subject of the study reveals the following mental processes:

1. Externalising
2. Understanding
3. Decomposing
4. Brainstorming
5. Evaluating
6. Selecting and deciding
7. Focusing

Table 4
Mental processes among participants

Mental process	Rachel	Jasmine	Jessica	John	Bill	Rick	Sam	Jack	Barry	Kyle	Patrick
Understanding	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Simulating			✓		✓	✓	✓		✓	✓	✓
Decomposing	✓				✓	✓	✓	✓	✓	✓	✓
Selecting and deciding					✓	✓		✓	✓		
Brainstorming			✓		✓					✓	✓
Compromising	✓		✓	✓	✓						
Designing	✓	✓	✓	✓	✓		✓		✓	✓	
Focusing			✓		✓	✓	✓	✓		✓	✓
Viewing differently	✓		✓	✓		✓	✓		✓	✓	✓
Verifying			✓		✓	✓			✓	✓	
Externalising	✓	✓	✓	✓	✓		✓				✓
Extrapolating				✓	✓		✓			✓	
Learning			✓				✓		✓	✓	✓
Evaluating	✓		✓	✓		✓		✓	✓		

8. Extrapolating

9. Designing

10. Verifying

11. Viewing differently

12. Simulating

13. Learning

14. Compromising

The processes and engineers that used them are summarised in Table 4.

The first research question above and the second research question, “How do the mental processes interact with the other components?”, are discussed in detail in Section 5.2.1 through Section 5.2.14.

5.2.1 Externalising

Externalising is the process of turning thoughts into something that can be perceived with the senses such as sight or hearing. It was found that seven engineers used

externalising as a mental process. Although externalising could be thought of something mechanical, it does require some cognitive load.

Externalising produces information on an external medium. Sam said, “So I just generally write diagrams of dependencies on little bits of paper and then sketch down some paths to see if I can break it.” Externalising into a visual medium is needed to assist in understanding and breaking down. Jessica said, “A good way is to actually write this down, draw a one page graph, to understand it better.” The more sensors used to perceive something, the higher the understanding achieved.

Externalising with a medium is done prior to the process of uncovering a potential solution. John said:

If I got stuck on a problem, ask for support, and a lot of the time it's just verbalising the problem and you find the solution as you say it out loud. Whereas you can just sit there half an hour and think about it, and something's not working. And once you actually put it into words, you can identify what the problem is.

Externalising is also undertaken prior to the building activity. Rachel needed to draw diagrams to preview the software before commencing on programming work. She said:

You need to get a fairly good handle on the big picture. The way... the system looks. You need to have a good picture in your mind of how it all sits together, before you start programming. Just like the examination one, I drew a big picture, and I was constantly referring to it. You see where all the data goes, you see all the fields. It's just so much easier. So I didn't start programming until I've got the bigger picture and then of course it's much easier. You know exactly what you're doing. You're naming schemes are all logical because you know what you can't do.

Some developers prefer not to externalise. Performing the development work individually and not in collaboration reduces the need to externalise. Rachel said:

And get a handle of all things. Sometimes it's just too many words to try and explain things, whereas you know, you can sort of see the pictures in your head, trying to put those into words. And sort of getting right into it, when you are right into the code and try to explain stuff, it's just too hard.

The quality of problem representation available to the developer reduces the need for them to externalise into another medium. Patrick said:

The tool was very good in terms of, like C# in that we use a lot more English-type variable names. And C# had the ability to do oral code completion. So because there were more English-type variable names, they were longer. And so with the code completion it was still quite quick to write everything out. We needed to...break things down into more sort of English-type steps so that method names too, it's probably more of an architectural thing rather than from the tool, because we ended up designing the methods with sort of sensible names and descriptive names. ...therefore, it was easier to read and there was less sort of like documentation in line that we need to do because the code basically describe itself.

In summary, externalising is related to the other components of problem solving in the following ways:

1. Externalising outputs external aid.
2. Externalising is needed for other mental process such as understanding and decomposing.
3. Externalising is undertaken prior to the activity of building.
4. The externalising process may be reduced when development is not made in collaboration with other developers.
5. The externalising process may be less needed when problem representation is more accurate.

5.2.2 Understanding

All eleven developers stated that they went through the process of understanding. Understanding is to “perceive the... meaning of... [or to] be... knowledgeably aware of the characteristic and nature of” the matter to be understood (“Understand,” n.d., para. 1). Developers went through the process of understanding the information around the problem such as the user business process and their existing system. In talking about understanding the existing payroll processing module in the existing Delphi platform, Patrick said, “Aw, it was mainly going through the code. And from the code, sort of determining how, sort of what, the code was doing, and what that piece, what that step was sort of there for.”

John said:

Because it's knowing in the back of your head, what is, what is expected, what is this product going to be used for and how they're using it. You know, um, you may be giving a specification: OK, you're going to read this data, from here and produce these documents here. But knowing how often it is going to be used, how much load is going to be in the system, how many users are going to use it, [and] is it going to be used by more than one user at a time? Things like that are important.

Information on the problem is used for the process of understanding and such sources include user requirements. Jessica said, “Well, basically, the first thing that needs to be done is to have really good understanding of what the users need. And then from there, um, you could explore your options.”

An understanding is accomplished by referring to external aids that represent the problem. Patrick said, “And there was also a lot of documentation on the existing processes.” Jasmine said:

So, there were a number of documents that were written by [the] manager [of] student administration. Last year, they were sent through to me, kind of their wish list, what they kind of, what they’re trying to achieve. And um, another one that was previous to that, these were all kind of background. And then the one with information on ...the current feral system and what its functionality is doing.

On the other hand, understanding may be adversely affected when the representation of information is not in front of the developer. Bill said his understanding was hindered when there was no representation in front of him:

If I can’t see it in front of me, personally, if I can’t see it in front of me, I find it very hard to understand. And I find that I sometimes get caught on a problem and the only reason I’m not moving anything forward with it is because it’s not being laid out in front of me. So drawing pictures and drawing diagrams, and um, yeah mainly visual.

Not only do understanding is acted on the problem, but it is also acted regarding the solution objects. Developers reported having to understand an interim solution, and front or back-end of actual solution.

Regarding a half-baked solution developed by a previous developer, Kyle said “And um basically it’s just reading through the code and figuring out what had already happened with the previous guy”.

Barry indicated how he understood the connections between the front and the back end of his web-based survey by saying:

There’s the web page, there’s the database, and there’s the script that links the two. They all ought to be kept in sync. If one is out of sync, the whole thing falls over.

Understanding is guided through their own experiences with previous similar tasks. In talking about how she understood the activities of users of the graduation system, Jasmine detailed her experience with migrating the old system and said, “So I became familiar with these activities because I’ve migrated them.”

Understanding was not only guided by personal experience, developers also gain understanding by interacting with the clients and other developers. In trying to understand the requirement of the securitisation system, Rick related:

As far as time goes, for example I talk to the lead for 20 minutes, 10 to 20 minutes, in order for me to understand what is required.

Regarding how she understood the problem, Jessica said:

Really communicate with the customer to find out what gap they need to be filled in. Sometimes we would meet up with them. Sometimes they would email us what they need, and sometimes we go to the actual site. We go on-site. And then, um, just look at how the user proposes what they're going to do. And then they'll tell us, "oh we need this, we need that, we want to find ways to make this better."

Tools or technology was another source for enhancing understanding. John said:

Oh, um, with the standard Accpac application are really good, as I said, you can script. You can profile the events that occur during creating the documents within Accpac. For the third party modules, the third party modules that we use, we had to use one of our development tools to actually identify what, um, to see the views that compose the tables. With Accpac you don't have a direct relationship. You have...everything is controlled by a system that will generate IDs, will compose, save for your transfers. You've got data from three or four different tables that need to be read in, and this is all done behind the scenes. So, yeah, Accpac is very good at showing you what it does, and how it go about things, what is required, for its own standard modules.

The process of understanding is intertwined with development activities and other mental processes. One such activity is the activity of building the system. Some level of understanding of the problem must be reached before building can start. Jessica said, "And then the next step is developing it, after I've gotten my head around it." Patrick said, "So I wouldn't say that I understand it all, but enough to do the port."

On the other hand, further understanding can be enhanced after a certain amount of building has been conducted. Rachel said:

Sometimes you sort of get down the track while doing something, you think, hmm, this is not right. It's not working. This is not going to do this any better, because quite often when you start something, you don't really have the full grip on the problem. You don't really understand what's it all doing. But once you get into it, you see the bigger picture, perhaps, and see how everything fits in. So you do, yeah, as you go, you have to constantly deal with finding what you need to do.

In the process of understanding, developers needed to go through other mental processes such as tracing and externalising. Engineers went through tracing to understand what commands in the software or the tools would accomplish. Jessica said, "And a lot of times you have to kind of really just get our hands dirty and play with the system, to get familiar with it." Rachel said:

You use the package, you sort of make the change and see what it does. The more you use the package, the more you'd have the feel of how it works. And then when you start editing in and stuff, you get into the code and see what it's doing.

Developers also externalised to assist their understanding. After getting information regarding the pay processing module in GHI Payroll, Patrick externalised that information into diagrams which then helped him to understand how the module worked. Patrick said:

And I spent, probably in this area here, probably about two or three months, um, making up BPMN diagrams of the HR log, like the pay processing takes about 50 steps. And so I made diagrams of each of these 50 steps at a high level, so one page per step of BPMN. And that helped me to understand how it, sort of, the overview of how it worked.

For developers, understanding problems is a prerequisite to the process of brainstorming ideas on how to solve problems. Jessica said:

Well basically, the first thing that needs to be done is to have [a] really good understanding of what the users need. And then from there, um, you could explore your options... You do it as soon as you analyse the users' requirement. Then you think about what options are available.

In summary, understanding is related to the other components of problem solving in the following ways:

1. Understanding is undertaken on and uses information of the problem, and is adversely affected with the lack of the problem information.
2. Understanding is a process regarding the solution.
3. Understanding is a prerequisite to other mental processes, such as brainstorming.
4. Understanding is a prerequisite to, and is assisted by, development activities such as building.
5. Understanding uses other mental processes such as externalising.
6. Understanding is guided by experience.
7. Understanding is assisted by tools and technology.
8. Understanding is done by communicating to the client and by working collaboratively with other developers.

5.2.3 Decomposing

Another important mental process is decomposing of elements into smaller elements. Eight of the developers mentioned decomposing elements as part of their mental process. Rachel said “You do little bits, and talk little bits of it. Trying to sort of divide and conquer the problem I suppose.” Jack used a decoupling approach in decomposing the solution into parts that solve distinct, smaller problems. Decoupling is an approach of breaking down problem solution components that should work independently to solve sub-problems, but could work together. He said:

I think probably one insight that I can give you is that my problem solving approach is decoupled. If I can use that. In coding, decoupling means the reduction of dependencies. So if you produce a solution to a particular problem, that solution should work independently. But that solution with another solution can work... together.

Developers went through the process of decomposing the whole development task into smaller sub-tasks. Patrick said:

And what we tended to do was [if] we have 50 steps to do, we tend to break them up... so that one person would do one step, another person would do another one...

Rick said, “So each phase we decided...what needs to be done. Broke it down. Figure out how long each part would take approximately.”

Decomposing problems allowed the development activities to proceed. In Rachel’s case, by breaking down user requirements based on ease of development, this allowed her to go through the development activity of preparing a specifications document. She said:

You pick up the easy bits that are pretty simple and you realise that the users actually do know what they want. They wanted it moved, that was easy, so I did that. But then you’d also need to talk to them to find out what they want for details of the other stuff. So once you’ve got that written down, plus it’s quite useful to show them things as well.

Patrick decomposed code segments in a way to facilitate the activity of testing. Patrick said:

And we actually found that it took like more code to actually write the new way than the old because of the fact that we’re breaking things down into smaller methods for our easier testing.

Decomposing of the problem task is required in order to facilitate the structuring and organisation of the main problem task. Patrick said:

Because you couldn't break it up into too...[many] interdependent steps, but enough for people to be able to work on their own area.

Sam needed the decomposition of tasks to allow the sub-tasks to be undertaken concurrently among the developers. He said:

Whereas if you separate things up so you can work on everything at the same time and the same sprint, then you don't get so many bottlenecks. And you have more, what I call, parallelism in your system...everybody can keep working, and not... hold up anybody else.

Besides on the task, the decomposing process is acted on, and is based on problem information. In the early stages of his task, Barry broke down the list of interview questions into several groups of related questions. Barry said:

In terms of [Ali]'s point of view, and [Ali]'s case, I guess, ... he gave me a document full of a whole bunch of not very well-arranged questions and blocks and places. I went back and looked at it. And I thought, well, OK. I immediately thought we're going to have to break this up into pieces.

Sam needed to see diagrams of component dependencies to help him break them down. He said, "So I just generally write diagrams of dependencies on little bits of paper. And then sketch down some paths to see if I can break it."

Before developing the solution that actually gets implemented, Kyle decomposed the interim solution from another developer and tried to refine that solution. Regarding the interim solution, Kyle said, "First of all, there was already something that kind of did the job. But [it] didn't, wasn't really up to scratch. So, I basically tore that to bits, and refined that." As quoted earlier, Patrick decomposed programming codes into what he calls 'methods', also known as 'procedures'.

Solution objects were decomposed based on user steps, system layers and computer tasks. Bill said:

The practical process often is actually working small bits out... Probably again starting from the user's actions. Like you know starting from, this is what it's supposed to be able to do... and then, OK, then what, I'd probably think of it in three layers. What's the database going to be to provide, or store in order to do that operation? What's the user interface going to show? And what kind of feedback [are] the users getting when something is happening? And then the middle bit is the web service, I guess. How do you tie storage and data manipulation back to the user interface? So it's, so those two edges know the storage system side and the user's edge.

Experience with similar tasks guides how developers decompose elements. For example, Rick relied on his vast experience in programming in order to break down computer processes for the solution. He said:

It's obviously experience. If you had done any programming where you have to do something to scroll for IDs, you'd probably done a loop. Of course, it's something I've done before many, many times... I guess most of the things you do, you're trying to find from experience.

In summary, decomposing is related to the other components of problem solving in the following ways:

1. Decomposing is undertaken on development activities.
2. Decomposing is a prerequisite to development activities such as preparing specifications document and testing.
3. Decomposing is performed on a solution, such as an interim solution and its programming codes.
4. Decomposing is undertaken on, and requires the use of problem information.
5. Decomposing is a prerequisite to organisation and structure.
6. Decomposing is guided by experience with similar tasks.

5.2.4 Brainstorming

Brainstorming is the process of laying down all possible problems or solutions. In this study, brainstorming is defined as not restricted to only a group environment. Working alone on a task, an engineer may lay down different ideas, before he evaluates each idea in turn. Kyle said:

I personally think that ...you'd better off to get quite a few, as many ideas as you can to start with. No matter where they come from, and no matter how stupid they sound, and to work through those ideas.

Engineers brainstorm ideas for a possible problem that causes a certain symptom of a problem. This is also referred to as making a diagnosis. Rick said, "We discovered that this wasn't working. So we had to go back and find out why. Because we had expected results, and they weren't matching it."

Brainstorming produces solution objects in the form of several solution proposals. In thinking about the database solution, Rachel thought of several options. She said, "With this one, no, there were several solutions."

The process of brainstorming would be followed by the process of evaluating the options.

In certain cases, engineers do not perform brainstorming. One of the factors is the pressure for delivering the solution on time. When there is short time, engineers do not brainstorm several ideas for solution, but rather think of one idea and quickly build and refine it. Bill said:

[There] certainly...[were]n't lots of different options considered, and the constraints that we had, time of getting it done, we just didn't look at many options. So, the solution in the end would have been a lot like what you've described in terms of making something, seeing its faults or strengths, and adding to them. So that seems to be how it works.

In summary, the process of brainstorming is related to the other components of problem solving in the following ways:

1. Brainstorming of ideas for solutions occurs after the mental process of understanding the problem.
2. Brainstorming outputs solution proposals.
3. Brainstorming is followed by the mental process of evaluating each brainstormed idea.
4. Brainstorming may not happen when there is a short delivery time pressure.

5.2.5 Evaluating

Six engineers explicitly mentioned the process of evaluating. Evaluating is a mental process of looking at attributes of a particular matter such as benefit and cost, advantages and disadvantages, importance or lack of importance, and reasonableness. Evaluated problems or solutions may then be ranked. Jack said:

So what we did was we actually got the whiteboard up and we got stickies, and basically we put the stickies on the side of the whiteboard, "and this is our priority list." So these were all the issues, kind of very broad, high level issues. And we'd only focus on the top sticky first. And we wouldn't move to the next one until the top one was sorted.

Engineers perform the mental process of evaluating client requests and problem representation. Barry evaluated the drafted questions for Ali's survey in terms of the reasonableness of the questions. He found that some questions were not reasonable. Barry said:

And the first set of iteration that [Ali] had, he was ranking specific matrix which are used to analyse functions and loop side, what part of the survey it's about. And he had like two or three that meant the same thing but were worded differently. And he had a couple that just made no sense at all. And I pointed this out. I said, These three questions here are pretty much asking the same thing. And this piece out here, I have no idea what you're talking about...Originally, it was a bit of a mess. It's a complicated way of asking what he wanted to ask. And it looks worse even.

Engineers would evaluate users' requests and determine whether it is reasonable to be implemented. Jessica said, "Basically they would start with telling us what they require. And then we go in and check to see if it's something we can do for them."

Rachel evaluated a user's request to put data into one database table and concluded that a lot of information would be lost if the request was implemented. She said:

If you start sticking them all in, you'll lose a whole set of, power in the system. Because they [tables] can't do certain things. If you put it up in one long table, then you'll lose the power in the system...But because what they're seeing is one screen then they don't understand that they are two separate tables. Yeah, they ask for all these things, and we think, yeah, they're not going to work.

When engineers brainstorm ideas, they would then evaluate each idea. Following the brainstorming process, Rachel weighs the advantages and disadvantages of each database structure option. She said:

With this one, no, there were several solutions. We sort of have to think about the pros and cons of all, like do you go with a flat file, you know a 1-to-1. Do you make it one-to-many? What happens if we do that? What's the screen going to look like? What happens if we do it this way, what happens if we do it that way? ...usually the solution that you do, you have a few ways of doing things.

Evaluating is done prior to starting the building on a selected option, which has been evaluated against other options. Before building the additional tables in Dynamics AX, Jessica evaluated the types of tables that could be used to process the intended outcome. She said:

...basically you have to know what the Dynamics AX is capable of doing, and what are the similar things that it has, and then you can from there [to] decide which one is more suitable. ...for example, in Dynamics AX you could store data in several ways. You could store it in a table; you could store it in a temporary table, or you could store it in a data type, like a set or a map, or a container. These are all very similar; they all store data. But there are some minor differences like some containers you can only sort through them consecutively, one at a time. And there are some containers where you can just insert right in the middle of the data and take data out from anywhere. So basically it depends on ...what you're using it for and you select based on that. You select the best match...Yeah, you think about what would be suitable. And then you do it.

Some of the criteria identified in the evaluation are cost, delivery time and functionality. Regarding the criteria for evaluating solution options, Jessica said:

And then from there... you could explore your options, and between the options, ...you could decide based on ease of development, based on the efficiency...[and] speed; the time it takes to develop something... Yeah cost, well, the time and the cost.

Barry evaluated the option of a cloud-based web survey service versus a web survey developed in-house in terms of which option would give the functionality Ali required. Barry said:

...those automated systems, they do exist, but they are limited in terms of what they can present, how they can present it, and where it stores all its information. I know... [Ali]'s survey we're using as example. That would have been too complex. Because it's a multi stage process where there's some conditional processing. If the answer is this, then do this, otherwise do that. Stuff like that won't be able to be hosted on some sort of externally submit-your-questions-and-we'll-make-the-page-for-you.

In summary, evaluating is related to the other components of problem solving in the following ways:

1. Evaluating is done on problem information such as problem representation and client requests.
2. Evaluating is done on solution object such as solution proposal.
3. Evaluating is done after brainstorming and before the development activity of building.
4. Evaluating uses success indicators such as cost, delivery time and functionality as criteria.

5.2.6 Selecting or deciding

Selecting or deciding is the mental process of choosing an option to proceed with, based on certain criteria. Jessica used ease and time of development as a criterion. Jessica said:

And between the options, you... could decide, based on ease of development, based on the efficiency of one type of efficiency; speed [or] the time it takes to develop something.

Others decided on a solution just as long as the solution would work. Rick said "But actual implementation is usually on what you actually think of to code, and that usually

works.” Jack said “I think [it] depends on the scale of the problem. If it’s a little bug, I’ll choose ...[the] hypothetical I think that works.”

Deciding was done regarding tools and technology to be used. Without indicating the reason, Rick decided, “You have to use database for this task.” Jack decided on PHP language because it has the largest community of adopters, promising wide support. He said:

I mean sure ...there’s languages that’s as high quality as PHP, but talking in terms of pure quantity, by far PHP has the most developers throughout the world, and thus [it] provides the largest community.

Deciding could be done in collaboration with other developers. In deciding on the appropriate database design, Bill said:

So it was quite important that the database was agreed on by everyone, because we’re going to share the same tables. We’re going to be reading and writing on the same tables.

The process of deciding occurs before the activity of building. Engineers build the solution based on the option they’ve chosen. Following the decision made on the database, Bill and his team members started building. He said:

So once we agreed on that, the different parts started. Somebody started doing changes to ...Outlook, because we already have an existing product here for that. I began doing the website and another guy began doing the web services.

In making a decision, engineers refer to several sources for guidance. Firstly, experience with a similar task is used as a basis for a decision. Bill decided on tools that he used previously on similar projects. He said:

There were some constant things that ...we kind of already decided. So we’d decided on using Outlook, because that’s something we already had used...Enterprise Practices we’ve used before. That was something we’ve picked up from the previous job, seeing someone else doing it and realised it saved a lot of time and made things easier to code.

Regarding the choice of Zend framework, Jack related:

It didn’t provide any particular advantage. I’m sure that they’ve achieved similar things as say, Django or Java or what have you. The main reason for ... going through that is that’s my background. I know it, I’ve built [with it].

To assist in collaborative work was also another basis for deciding on a tool. Again, regarding the choice of Zend framework, Jack said:

And another thing is, this is a sub-project that came out of my larger ambition, which is to launch Bellacms with the support of Zend Technologies. So getting Zend behind me in my project, that's the main advantage.

Some decisions were made after approval from the client. Jessica said:

Then you think about what options are available. And then you present it back to the client for them to decide on ...which option they would like to go for. So you give clients that choice as well.

In summary, the process of selecting and deciding is related to the other components of problem solving in the following ways:

1. Selecting and deciding may be performed in, or allow, collaboration with other developers.
2. Selecting or deciding may involve communicating with clients and getting their input.
3. Selecting and deciding is done with regards to tools and technology.
4. Selecting and deciding uses previous experience with similar tasks as a basis for decision making.

5.2.7 Focusing

One way of easing the mental representation process of a complicated system is to focus on the certain aspects of the system. Focusing refers to the process of concentrating a particular item, while thinking less about other items. Seven engineers reported focusing on certain aspects of the development.

Focusing is undertaken by engineers on a particular solution option when they look at one option while putting aside other options. They think about the details of the chosen option at least until they discover that the option will not work. Kyle said:

What happens is...as you look at each one, you ...use quite a bit of gut feeling on it, and basically you go down for as long as you think it's worthwhile. Sometimes you pick a solution, or pick an idea that you think may be useful. And you start delving into it a bit more and realise, you know, you ran into a dead end.

Focusing was also done with regards to modules. After trying to develop the Bellacms, Jack focused his attention on the MyResearch module, as instructed by the university. While focusing on MyResearch, his work on the Bellacms halted. Jack said:

Instead, what happened was a thing called, originally it was called MyHome, then it was called MyResearch. It's all about the academic and basically managing ...a piece of research and the likes of the researcher that included the CV, PBRF, profiles, all those kind of things.

Focusing was also done on the details of the programming code. Sam said:

Another thing you have to think about as a developer is you have to be very anal, if that makes sense. When, if you think about a human doing a process, if, say, I'm going to a shop to buy milk. If I were to do that, I'll just walk out the door. And if I get to an intersection I have to make a decision about where to go, 'I think it's over there, give it a go.' If that's a wrong decision I'll change my course until I get there, that kind of thing. When you're writing a computer program, and the way that we write software at the moment, obviously the computer program has to know all [of] the pathway from the start. And depending, the better developers tend to sort of realise that and understand what they're doing. One of the patterns that you get with the developers is that sometimes they get the tendency to just sit down and code it. That basically comes, that's a very human way of doing it. And what happens is that you find after doing that a few times is that you can just create. You've got to set out from the start, at least building into the system at least enough potential to change so that you can define the path in one big definition. Because that's what you have to do as a developer.

Having information shown in external aids assist engineers with focusing on things they needed to do. Jasmine said:

I started to do that by kind of just listing all the columns I have, just some information about them, and that really help me solidify where I needed to go.

Focusing could be done after the process of evaluating was undertaken to prioritise aspects of the problem or solution. With other developers, Jack evaluated and ranked issues that needed to be solved and then started focusing on those issues in turns according to the priority. He said:

So what we did was we actually got the whiteboard up and we got stickies, and basically we put the stickies on the side of the whiteboard, and this is our priority list. So these were all the issues, kind of very broad, high level issues. And we'd only focus on the top sticky first. And we wouldn't move to the next one until the top one was sorted.

In summary, focusing is related to the other components of problem solving in the following ways:

1. Focusing is undertaken on solution objects such as solution proposals, modules, and programming codes.
2. Focusing is undertaken collaboratively with other developers.

3. Focusing uses information about the problem shown on external aids.
4. Focusing is undertaken after the mental process of evaluating.

5.2.8 Extrapolating

Extrapolating is defined, in this study, as the process of extending an idea of an unknown based on information of the known. Extrapolating includes making assumptions or generalisations.

Although extrapolating may produce inaccurate solution objects. Based on his experience and information available regarding his client's business process, John extrapolated that the software to be developed would treat whole shipments in boxes. John later found his extrapolation was incorrect. John said:

Where in the system I expected everything to be in units of one. And instead there was, say boxes of 12 or boxes of six. ...the Accpac has what is called the conversion factor, and so you can just divide the unit cost by the conversion factor, and you would have the amount for each. ...they may have a box of 12...syringes, but they only want to ship six of them, they won't ship the whole box, and that's something that I took for granted. I thought they would have a box of something so they'll replenish the location with a box of something, no, they don't. They just buy, go and buy a box full and then take out as many as required.

In the mental process of designing a solution proposal, extrapolation is necessary because there is incomplete problem representation. Kyle said, "I mean you've got to come to it with some stereotype to start with, otherwise you've got no idea where to start."

After the extrapolating process the developer would go through the mental process of verifying. Bill said:

It's creating a vision of what might be. At least in the selling design, but much into the detail design. There's certainly a lot more going back to the project manager, and sometimes just directly to the customer and meeting the end user and saying, 'Is this how you'd expect to use it?' ...you get feedback in, and you actually got to throw away a lot of ...the things that you do because they just don't, they're not exactly right. Sometimes it's just a tweak.

To undertake extrapolating, previous experience with similar tasks are used as a basis. John said:

...I thought, we have another product with them where all the units measured are 'each'. From there, I took it for granted that the units measured would be 'each' for all the items. I knew that things may be delivered in boxes but they would be accounted for in systems as a unit of each price. But they weren't.

In summary, the process of extrapolating is related to the other components of problem solving in the following ways:

1. Extrapolating produces output that goes through other mental processes such as designing and verifying.
2. Extrapolating uses previous experiences with similar tasks as a source of information.
3. Extrapolating may result in an inaccurate solution object.

5.2.9 Designing

Designing is regarded as a formal and important activity in development. Due to the nature of intense cognitive demand for the design activity, it is categorised in this study as a mental process. This study found specific references to the design process amongst the eight engineers.

In earlier stages, designing a solution involved producing a model of the solution. This model may be in the form of a mock screen or interactive demo. Bill said:

You're not making a lot of effort to make sure that it's clean and tidy. It's just, it's just getting it to look right. ...It might be a screenshot, it might be a small visual interactive demo but you're actually not getting any of the application working right underneath the surface.

Among others, the design would showcase a proposal of the front-end of the solution to the user. In the design of the web survey, Barry said:

And in [Ali]'s case he didn't really care so much as long as it's neat and tidy. So I use[d] just the standard sort of method of putting things in tables and making sure rows were easy to read and easy to separate.

The design that is developed is a prerequisite to the implementation of the solution. The solution must be designed before it is implemented. Rick said:

...implementing is totally dependent on the development of the design. I'm not sure if I need to elaborate on that. I mean if the design is wrong then the implementation is wrong, and vice versa. So, the design is the most important part, because a bug in the design would take a longer time to fix than a bug in the implementation and it's harder to test the design than the implementation.

In the process of designing, developers needed information on the problem. Jessica needed information on the users' business processes. She said:

For example, the payment gateway, they would need to get first name, last name information, and then to get credit card details. And then so what I would do is I would think about what would be on the first screen that is useful. Or maybe, could I put the other details on the second screen, so that it won't be too many on one page.

Design relies partly on the specifics of client request. Barry said, "And I'll format the webpage based on pretty much how the person wants me to format the webpage." Some of the requests were laid down in requirements document which then forms the basis for design. Bill said:

In terms of their particular requirement, they were specified in the requirements document that they had. So before this, I think it was a request for proposal. And so our selling point was actually responding to something they've asked for and we went back to the design which basically is a suggestion about how it could work, in response to what they've asked for.

The ease of the design process is affected by the level of accuracy and detail of the problem representation. Barry said:

And how I have to structure it from my point of view, how I decide what it looks like on the web page. How I decide what, how it would appear inside the database. That's all related to the field of the question. And if the question is worded well, that's generally very obvious and makes sense.

Experience with similar tasks enables and provides the basis for the design process. In designing the database structure for the Dynamics AX Systems, Bill related having previous experience with time-sheet design that guided him. He said:

Well, like I said, we had seen a few timesheet systems internally... I have been involved and seen a couple of these, I've seen how they design the tables. I also knew how we've done our current one that we're using at the time and where the problems with that.

Similarly, Jasmine said:

Because I migrated both of the applications, student records and [the] graduation application, I had migrated them. I had them in my head, kind of, how they were already working. I did look at them sometime. Or maybe I had a little of background information as well already in my mind. That is why I went straight to the tables, and how I would make the table changes. Well, it worked for me.

Tools and technology were used to assist the design. Rachel said:

...we use, this is APEX, Application Express, which is Oracle. It is sort of screen designing, database interface thing. So that does most of the front-end.

Besides relying on experience, and using available tools, developers also designed through collaboration with other developers. Bill said, “And we all co-design together at this point so, at the start we had no, we started with actually database design.” John said:

He discussed with them their needs and valuated a resolution to improve their business practices; make it easier for them to get on with their work. And then came back and sat with me and discussed it. We sat down and designed it.

Despite using available information on the problem, in order to design engineers needed to go through the mental process of extrapolating to fill in the gaps of the problem or solution which were not addressed in the problem information. Otherwise, relying on available information alone would not be sufficient to come up with the design. Kyle said, “I mean you’ve got to come to it with some stereotype to start with, otherwise you’ve got no idea where to start.”

In summary, designing is related to the other components of problem solving in the following ways:

1. Designing is a prerequisite to the development activity of implementation.
2. Designing outputs problem information in the form of external aid.
3. Designing outputs a solution object in the form of front and back-end.
4. Designing is undertaken in collaboration with other developers.
5. Designing is based on and affected by the problem information such as the representation of the problem, business processes, client requests and their requirements.
6. Designing is based on solution proposal.
7. Designing is based on experience in similar tasks.
8. Designing requires as a prerequisite the mental process of extrapolating.
9. Designing is assisted by tools and technology.

5.2.10 Verifying

Verifying is the process of comparing an idea or assumption with other sources to confirm that it will do what it is supposed to do and to avoid mistakes in the development. Barry said:

When in doubt you'll have asked about it anyway. You don't make an assumption. You just go, oh I've looked at this question, if I was you I would do it like this. How would you? There's two ways of answering this question. What would you prefer?

Verifying an idea is done after an idea is formed by extrapolating. This is because when a developer makes an assumption, it needs to be confirmed via verification. When an assumption is verified, then the developer would proceed with the idea. An idea that could not be verified would be put aside. Bill said:

It's creating a vision of what might be, at least in the selling design, but much into the detail design. There's certainly a lot more going back to the project manager, and sometimes just directly to the customer, and meeting the end user and saying, 'Is this how you'd expect to use it?' ...you get feedback in, and you ...you've got to throw away a lot of the things that you do because they just don't, they're not exactly right. Sometimes it's just a tweak.

Sources of verification include users and people involved in the development. As above, Bill mentioned verifying his idea with the project manager and the user. Kyle said, "And as you go, ...it's also a good idea to bounce...bounce ideas... off other people. See what they've got to say."

Rick verified whether what he built was as required with Sam. He said:

And in implement, I'd probably ask several five to ten minutes pieces within that for verification. And then two minutes of review at the end.

In summary, the process of verifying is related to the other components of problem solving in the following ways:

1. Verifying is done after mental process of extrapolating is done.
2. Verifying is accomplished through working with other developers and with the clients.

5.2.11 Viewing differently

This study defines "viewing differently" as the process of viewing a particular issue in various aspects or from different viewpoints, and eight engineers went through this process.

Engineers reported resorting to viewing differently the problem representation. Sam mentioned the need to view concepts from the user perspective and the developer's perspective. Sam said:

So one of the tasks that's always interesting when you're a developer is, if you're a developer that has to interface with the client as well, then you have to try and separate yourself out so that you've got the language that you talk. You learn the client's language, but then you also have your own language to describing the system that can be used to reason about it, if that's right. So in this case the client would look at it in terms of say, contracts arbitrating process.

In looking at the questions listed by Ali, Barry would not only look at the questions at face-value, but try to look beyond what Ali wrote and see what he was trying to ask and expect as a response. Barry said:

When someone has a simple, a single question in a Word document or something else, what they're really trying to ask, how they're asking it, and what they expect, what to send as a result.

Viewing differently was needed when an interim solution proposed by a previous developer could not fulfill the intended result. Kyle viewed a solution proposal differently from the interim solution. He said:

So we basically did a rewrite and started from scratch, and worked on what we already had, and what it wasn't. What it wasn't and what they would like it to be doing. And just compiled that into a new set of requirements.

In Rachel's case, viewing differently occurred on the back-end while the front-end was the same. From the same interface, the users viewed that the information on the interface should be in one database table. Rachel, however, asserted that it should be in two tables linked by a one-many relationship. Regarding this, Rachel said, "But they [the users] see this [interface] and they think it's from part of a same table."

When facing difficulty building the software solution that utilises a certain framework, Patrick would find a different way or a work-around the framework in order to ensure that the solution works. He said, "So we had to find ways around it. Either code things differently, or try and restructure the application so that [we would] not have to do the call [on the framework]."

Sometimes an infeasible solution proposal would trigger the engineer to view a different way for a solution proposal. Jessica said, "Sometimes when there's really no solution, then we just have to say, no, sorry we can't do it, but most of the time we would find a work-around." Sam said:

In most cases it seems to me to be just that, if you can't do it this way, then you need to start to follow another direction. It's not so much about being really good at solving a problem in a certain way. It's about being able to have a broad enough perspective that you can come across the way to solve the problem or a way to solve the problem.

Allowing oneself to view differently is also a type of attitude. Working in collaboration can achieve the desired outcome only when the engineer incorporates other peoples' ideas as a way of viewing differently. Sam said, "You just sort of ask questions and discuss and allow your own idea of what should be done to be changed."

In summary, viewing differently is related to the other components of problem solving in the following ways:

1. Viewing differently was done on problem representation.
2. Viewing differently was undertaken on, and in response to, the failure of a solution.
3. Viewing differently was induced by working collaboratively with other developers.

5.2.12 Simulating

When an engineer arrives at a possible solution option, the developers would simulate in their mind whether the solution would work or not. Seven of the developers mentioned this. The object that is being simulated in the mind includes a solution proposal and programming codes. In terms of simulating the solution proposal, Rick said:

Just think about the problem that needs to be done. Our problem is that, I'll give you a simple example...one thing we had to do was generate some reports, essentially; compress them, and then email them. And that is three things to do. Generate reports, that's one thing. Compress them is one thing, then email them.

Bill said:

...I guess it also gave us, because what this product is going to do at the end was basically dump data in the table,...the AX system would pick [it] up and say, OK, now there are invoices there and there are some timesheets that I can turn into journals, and [an] invoices plan that can be turned into real invoices. So the way the system is going to interface is just having data in a certain status left on the table. And as soon as that was in a certain status, the other part of it [the system] could come and pick it up.

Some developers thought of simulation in terms of formal computer processes in pseudo-codes, or the like. Jessica said:

Or type it, or even sometimes just writing it down step by step, like writing down pseudo-codes. That's what I do. Like, first step, take data from the database. And second step, loop through data. Things like that.

Rick said:

And then I think about generating reports, and I think, well it needs to iterate through all the reports that we have ...and generate them once for each input ID that we have. So we'll do [a] loop, and then what I need to do is what exactly I've set in. Looping through each ID that's being input, and then do another loop for each report that we have. And then save it somewhere, for example.

Similarly John ran through the solution in his mind based on computer processes. He said:

So the next step from there is you create a transfer and you have a loop that you would read the item information from a grid or an array or whatever. Keep on doing that until you hit the last item, and there's validation along the way to make sure, you know, the item number's correct, the quantity is greater than zero, the quantity on hand is greater than the amount you want to ship.

Mental simulation is also undertaken during the development activity of code review. This was done by a different person than the one who wrote the programming code. Patrick said:

And then if they're ok, I'd sort of send them off to code review. And then when that was ok, then they sort of be able to get released to the...stream. So, things... seem to funnel through a lot.

In summary, simulating is related to the other components of problem solving in the following ways:

1. Simulating is done regarding solution object such as solution proposal.
2. Simulating is undertaken while performing the development activity of code review.

5.2.13 Learning

The mental process of learning is defined here as the process of finding out about something not known before through experience and available sources. Five engineers specifically mentioned learning.

Among other things, engineers learn about the tools or technology that they intend to use in their task. External medium is one of the sources for learning. Jessica did not have prior experience in Microsoft Dynamics AX before her project and she learned about the software and obtained Microsoft's certification through training and online forums. She said:

Basically, there's certification with Microsoft, and we get training, we get training even online. We can go to ...like training session[s]...And you read manuals, you look online because the online forums have a lot of information as well. And so that's how you get familiar with the product.

Engineers learn about the client's business processes. To gain back-ground knowledge of his client, Sam searched the Internet for information regarding the business nature of the industry the client operates in. He said:

Leading up to it I spent a lot of time on Wikipedia and working out formulas and stuff like that. Just doing research into how the business worked.

Learning is triggered by inadequate knowledge of the matter being learned. Patrick learned C# programming shortly before performing the actual porting project because he lacked knowledge and experience in the language. He said:

...we were starting up using C# as well so...this was the first use of C# for me. And so there was a piece of learning the language even though I taught myself a sort of C# six months previously...

In fact through the activity of building, engineers learn through experience. Patrick added, "I think...because we were new to C#, there were certain things we learned as we went and set down, sort of instinctively, what was a better way of doing it."

Another useful source of learning is through collaboration with other developers. Kyle learned the overview of GHI Payroll software from people who had worked longer on the system. He said:

It's quite a bit of talking to people who have been in the company for a lot longer than I have. ...basically it would take a sort of like four years or something to get your head around it [and] to get a fairly good grasp of what is happening with the main application.

Working collaboratively induced learning on the other developer involved. Sam said with regards to Rick:

Yeah, but on the other hand, [Rick] and I often worked together even when...it's mostly me doing the work. We worked together so that he knows, so that he learns about what I'm working on especially when I'm writing a document or a specification or something like that.

In summary, the process of learning is related to the other components of problem solving in the following ways:

1. Learning is a process regarding, and uses problem information.
2. Learning happens before (to facilitate) and after (through experience) the development activity of building.
3. Learning is a process regarding, and uses tools and technology.
4. Learning is triggered by inadequate knowledge.
5. Learning is also achieved by collaboration with other developers.

5.2.14 Compromising

Compromising is coming up with a middle-ground with an issue that has two or more conflicting requirements. Bill said, “It’s finding a way to do what they wanted, while still keeping the accounting here, so the accounting people were happy.”

The process of compromise arises from conflicts. One of those conflicts is between the requirements of the solution and client requests. In the timesheet approval system developed by Bill, the accounting aspect required differences between total time cost and time charged be allocated on a certain basis across activities whereas the users of the system did not want it to be allocated this way. Bill narrated in detail:

Particularly around the invoice planning, we had a feature which was to do with taking a group of transactions and turn it over into an invoice. And originally, the way that we were working on our system at the time was to build [on] a line by line basis. You build this line, you don’t build that line, you write this line up, you write that line down, and whatever. They wanted to do it at a summary level; ...this is how much the total transactions add up to, so let’s build that amount, or if we want to change it, just change the whole amount and spread it out across the different transactions and we don’t have to go and do it individually. And initially they didn’t even want to know anything about the individual transactions at all. They just wanted to see the summary of the amount. But as we went on it became important to them that the write off was distributed across. For example \$5000 worth of time, and you’re charging it at \$4000 so that means you’re losing \$1000. That loss of \$1000 has to be spread at an even amount on all transactions. Initially they said, let’s just not record that we’re writing it off, but don’t change the actual transaction at all. So there was a part of it [that] was [an] accounting reason, we had to change things otherwise we’d have to adjust how we want it to work.

The second conflict that forces compromise is the conflict between user request and the cost to include the request. In John’s module that processes documents regarding materials issue and shipment, the module processes the transfer of items to physical destinations, but could not handle transfers to special cases such as individual doctors. The client requested that the system be able to handle both possibilities, which would require additional substantial redevelopment and hence cost. Because the clients would not agree to the additional cost, John had to modify the system to allow the users to give instructions to the system to be able produce the transfer documents manually. John said:

They don’t want to pay any more money. And we don’t want to apply any more time for it. It was something that the system wasn’t expected to do, and it wasn’t something that could be obtained from within Accpac, because usually it would be one or two items that appear in IC shipping, whereas this product has been created ...for transferring and/or shipping over 100 items at a time. So it’s... for creating documents containing many, many items.

The third kind of conflict that forces compromise is the conflict between the tool or technology used and the client request. Sometimes the client request cannot be implemented due to limitations of the tool or technology. Jessica's clients wanted the Dynamic AX system to produce output in the form of graphs showing certain information, but Dynamics AX does not have that functionality. As a compromise, Jessica customised the system to allow it to produce table-based reports that show the intended information. She said:

We usually implement reports, like we always develop a report that ...the users require. It may not be exactly how the users wanted it to look like originally, but we'd have something for them that presents the information that they need.

The client request is used as the basis for the targeted output of the compromise. In other words, the engineer tries to develop the systems with an interface that works closely what the clients requested, but the back-end of the system does the required data processing. Rachel modified the interface to look as if the data is handled in a one-one manner, but added components to allow the system to work as a one-many. She said:

...disguising the one-to-many relationship to a certain extent as well. So we kept it but they don't see it quite as much. Yeah, so it was more like what they wanted but the underlying structure isn't, and they get the functionality of the underlying structure. So yeah, it's kind of weird, but it works.

The process of compromising results in the end solution. Bill said, "And in the end it was quite easy, it's just that the system needs to do a lot more work. Rather than just ignoring the fact that they would be charging, they had to go and pro rate it at different cost at each transactions."

In summary, the process of compromising is related to the other components of problem solving in the following ways:

1. Compromising is forced by conflicts between client request and other sources such as system requirements, cost, tools and technology.
2. Compromising uses client request as the target of output.
3. Compromising results in the end solution.

5.3 Other Problem Solving Components

The other important components of problem solving besides the mental processes are discussed in detail and in turns in Section 5.3.1 through Section 5.3.10. This answers the second research question, "What other important components are there for problem solving amongst software engineers?"

5.3.1 Development activities

This study found a number of development activities undertaken by engineers who were problem solving. Development activities may be distinguished from mental processes in the following manner:

1. Activities are specific to IS development domain, whereas processes are quite elementary and general in nature.
2. Activities involve more mechanistic functions, whereas processes are more cognitive.
3. Activities are usually stated explicitly by engineers, whereas processes are more interpreted by the researcher.

Activities found in the study were: building, coding, configuring, customising, implementing, importing data, modifying framework, porting software, preparing the specifications document, reviewing code, standardising and cleaning, and testing the software. This list is by no means exhaustive, as the nature of these activities relates to the tasks undertaken by the engineer.

Preparing the requirements document

The specifications document included the problems or matters to be addressed in the new system. Jasmine said:

My first section was all the current issue, so it was kind of correlating everything that we have identified as problem areas or areas that needed improvement.

Preparing a requirements document requires the engineer to break down or itemise the users' request. In Rachels' case, the simpler requests were developed immediately, while the more complex ones were written in the requirements document. She said:

You pick up the easy bits that are pretty simple and you realise that the users actually do know what they want. They wanted it moved, that was easy, so I did that. But then you'd also need to talk to them to find out what they want for details of the other stuff. So once you've got that written down, plus it's quite useful to show them things as well.

Preparing a requirements document is not necessarily performed as a standard development activity. Rather it was done on an as-needed basis in several cases, as with Rachel and Sam. In Sam's case, the development of the securitisation system was not organised properly and there was no formal requirements gathering activity. The requirements

document was written by Sam only when he ran into difficulty managing the project and then he found the document helpful. He said:

And ...for a while I was, or probably about a couple of months, there was this all repository of requirements. And one of the things I did in that week whereas I wasn't doing anything was I made a requirements document. And ...that's helped a lot.

Build

Build is a general term used that usually describes the activity of moving from design into the creation of the solution and usually includes program coding and database development. All eleven engineers reported building. Engineers build based on user requests or on a compromised version of their requests. John said, "So we added that to it. He wanted the reports combined from two reports to one. So we did that for him."

Building is done on solution objects such as the front-end, back-end and framework. Barry mentioned building the web pages, database and the script that connects the two. He said:

And I go, "Well I can do the whole thing now, or I can just do the webpage. And we can change the webpage and change the webpage and change again which is a lot of work to redo again and again and again. But it's a lot less work than having to keep the database and the script in sync at the same time at every iteration"...But changing the whole thing, as [Ali]'s survey would have done, would have been a lot of work to do all three components, the webpage, the database, and the script. Which is why we just did his one, we built the webpage only through all those iterations.

Engineers also build the framework underlying the solution. Sam mentioned building the framework first. He said:

When you're doing something that's extremely poorly defined and you're going to be discovering requirements as you go along and doing it very ad-hoc and without any real process, you need to [build the framework first]. Sometimes it's a good idea to focus on ...not technology for technology's sake, but focus on building a base layer that makes the technology suitable for the business, and then using that base layer to allow all the actual business stuff to change without have them to cause.

Engineers relate the general work of building, usually after undergoing some mental processes such as evaluating, learning, understanding, externalising, and selecting and deciding. In order to build, engineers needed to understand the problems, evaluate solution options, deciding on the appropriate solution, externalise their thoughts on the solution model, and perhaps learn appropriate technologies. Examples of these connections were presented throughout Section 5.2.

Tools and technology are used to assist the activity of building. It was found that restrictions and difficulty with the tools sometimes made building problematic. Jessica said:

We'd use the reporting wizard and then write the report using Dynamics AX language which was C++, and based on that we would write a query. Something like a query, but the query is not as powerful as SQL, so we cannot do any complicated queries with complicated group-bys, or complicated sum-by, group-by, things like that. Or like selecting distinct values, because in SQL you can do that in one statement you know, select distinct or select count, but in Dynamics AX might not be as powerful.

Coding

Coding refers to the activity of writing a computer program using a programming language. Coding is part of the general activity of building a software solution but not all engineers go through this activity. Jasmine, for example, has only reached the stage of documenting her solution proposal and hadn't yet started coding.

Common programming languages used by the engineers include HTML, PHP, C#. There were software solutions that have their own coding components. Jessica said:

Dynamics AX has built-in tools, like a data dictionary, which maps to the database, and then it has its own programming language. It had some, like, developer tools; simple development tools, and a report with a ...report interface. So that helps me to do my job, yeah.

Tools were used to generate code, verify written codes and suggest more efficient codes. Patrick said:

And that has been a great help in terms of speeding the code up [and] in terms of speeding the code writing, and also for checking for problems. So the ReSharper tool's basically intelligent enough to say, "Well this is sort of maybe not right because you may have a problem here in the code." Or it will suggest different ways of constructing lines of code which ...I'm personally finding it a lot more useful than just straight Visual Studio. So that's been really good.

When coding for the porting project, Patrick refactored. In this case, refactoring was finding segments of codes throughout the module that perform similar actions, creating a single segment of code, and making adjustments to other codes surrounding the segment to reflect the combination of the original segments into one segment.

Review Codes

Code review was a formal activity in Patrick's case. Code review is a collaboration with other team members that involves programmers inspecting and reading codes written by other programmers in the development team. Patrick said:

And there was a group, so not the same people reviewed the same people's code, which worked and didn't work. There were some people [who] were better reviewing than others.

This activity is similar to the approach of pair-programming (Williams et al., 2000).

Standardising and cleaning

After codes are written, during development and before implementation, engineers need to standardise and clean their codes. Bill said, "then tidy it up and get rid of all the copying and pasting and trying to standardise it."

Although the engineers understand the codes they wrote, cleaning up the codes facilitated other developers working on the software. Jack said:

And this is happening to me right now because I'm the only one who really understands the back-end. I'm having to go back and you know, clean up my codes, documents...

Testing the solution

Engineers test the solution to determine if it works as planned. Testing could also include a comparison with a benchmark output. Six of the engineers explicitly mention the use of testing, except for Jasmine who had yet to develop the system at the time of interview. It is quite possible the other developers did testing as well, but there was no explicit discussion of that from them.

Testing is done after the activity of building and coding of the software. Jessica said:

And then once we've changed, once we've done the development on our side, I will do some unit testing on our development server. And once we're happy with that, we would move it to their test system for them to test. From there, if they're happy with the system, it gets moved to live.

Testing is done by comparing output of the systems against an expected output. Rick said "We discovered that this wasn't working. So we had to go back and find out why. Because we had expected results, and they weren't matching it." To remedy the failure, Rick had to request that the clients provide the algorithm.

Testing is done prior to implementation, via unit testing and systems testing. Jessica said, “Um yeah, we do unit testing. And then we pass it on to the consultants to do further testing before they release it to the client.”

Testing is needed to identify problem or bugs in the software. Relying on feedback from the clients is sometimes inadequate as users are usually quiet when the software works. Jack said:

But in order to do that, we need more formal testing. Because you just don't get, in the initial run you don't get positive user feedback. You don't get people saying, "I really love the way that button works", or you know, "I really love how that..." So I think I completely need a formal user-test group to gain a positive feedback.

During user testing, engineers found gaps or weaknesses in the software, or business process that they overlooked and did not consider. John said:

I don't know enough about the user and how the user would actually use the module until I sat down from where I've [done] the first test deployment. We have a test company that has data captured from a live company. ...when we deploy something, we test it with current data but it won't affect the actual live data. And from there, I could see what he would do, that he would have a list of items and he would key in the numbers from there. He wouldn't need to look up any items, he wouldn't need to, yeah he knew which items and how many quantities he need to transfer.

Some specialised tools were used to assist in the testing activity. Some were developed in-house, and some were commercially available. Patrick said regarding the two:

We had, there were actually some people who were building custom testing tools so we can test the data at each step to ensure that the step in isolation was doing its job. So the data that was going into the step and the data that was coming of the step were [the] same in the Delphi version and the C# version. So that made the job a lot easier in terms of knowing that your code was right by having the tools to actually test it...I think it was, like we've now brought a third party product called ReSharper.

Configure and Customise

Configure is setting up a software application for use by a specific user. Customise is modifying a software application to suit the needs of specific user. The difference between the two is that configuring usually involves simply specifying options that are already built into the software, whereas customization involves modification to the standard way the software works. Configuration does not involve programming, whereas customization may.

Information about the problem, such as regarding the business process according to user requirements, is used as a basis for configuring and customising. Jasmine said:

And then configuration. All the right configurations for that specific company...Develop customisation, to suit the client's requirement, because that was like a standard system. And then they might have requirements specific to them. And that's where I come and develop customisation.

This activity only occurs among a few engineers that utilise applications developed by a third party as a solution to the problems of their clients, in particular how ABC Technologies work. Bill and John did not report on configuration though, because their task involved developing extra modules to the software. Bill developed the web page part of the timesheet approval module. John developed the items transfer module. Those tasks required significantly more effort than configuration and customisation, and configuration and customisation were probably overlooked by both when they narrated their problem solving processes. Other engineers developed their own software from scratch (Rick, Sam, Barry, Jack) or worked on software developed by their organisation (Kyle, Patrick, Rachel, Jasmine) and hence did not require configuration and customisation.

Modify the Framework

The activity of modifying the framework happens in specific circumstances. Two conditions are required for this activity to occur:

1. The engineer utilises a certain framework as a base for the software that he is developing.
2. There is a need to modify the framework to allow the software to accomplish certain functions originally unsupported by the framework.

Two engineers reported the need to modify the framework they used. Jack said:

...the problem we faced here was, we were designing a system to run on the [university IT department] servers, and they wanted to use Oracle. We had to...so what we faced in particular was Zend's PHP talking to Oracle database. And there's a few things in the Zend framework, the Zend-Oracle connector, they needed tweaking..

In Patrick's case, the porting activity could only be completed after the framework was modified.

Patrick said:

A lot of the framework, we would be starting a porting and finding some of the framework wasn't there. Some other people would have written the framework and we would just port it down, our new methods and things. While we had expected some of that, we didn't expect it to the degree it was happening.

Port Software

Patrick was the only software engineer to undertake port software. In his case, it involved a direct port without the development of additional functionality, changing the database structure, or modification to the interface. Porting the payroll processing module in GHI Payroll software requires the modification of the underlying framework.

Implement

Implementation deploys a software solution that has been developed onto the user's computers for use. It is an integral part of the systems development life cycle. At the time of interview, apart from Jack's and Jasmine's tasks, software solutions had been implemented for all the engineers. Sam left DEF Ltd. before the securitisation system was completed, but Rick saw through the implementation of the system.

Implementation occurs at the end of the systems development. Mental processes and other activities typically occur before implementation. Distinctively, design and testing was done prior to the implementation of the software. Jessica said, "Um yeah, we do unit testing. And then we pass it on to the consultants to do further testing before they release it to the client.." Rick said:

Well implementing is totally dependent on the development of the design. I'm not sure if I need to elaborate on that. I mean if the design is wrong then the implementation is wrong, and vice versa. So the design is the most important part, because a bug in the design would take a longer time to fix than a bug in the implementation and it's harder to test the design than the implementation.

During implementation, data from the existing system is imported into the new system. Jessica was involved in the task of importing data. She said, "We have to organise the import data, to get the right data to start with." Patrick and Kyle did not go through importing data because their task did not involve implementation of the system to specific users, but rather implementation of the general software.

5.3.2 Solution object

Based on the interviews, this study categorised parts and stages of the solution as the solution object. The solution objects are interim solutions, solution proposals, front or back-end, modules, programming codes, frameworks, and end solutions.

Interim solution refers to a solution to the same problem that was partially developed by other developers and is now at the disposal of the engineer. Rachel and Kyle worked on interim solutions for a small portion of the overall system. Other developers worked

on relatively larger and more significant problems. The problems that Rachel and Kyle tried to solve were minor and not critical and they worked on their own. The problems were not critical to the overall software and may have received low priority from the organisation because previous developers were allowed to leave this task without successful implementation, and hence their partially developed solution became interim solutions for Rachel and Kyle.

Having an interim solution does not necessarily ease the engineers into their problem solving task. Instead, Rachel found that the difficulty with her task was about understanding the interim solution. Rachel said:

On a scale of 1 to 10, it would be about 6. The complicated thing was, trying to find out where other people have got to, and where their heads were... because they don't necessarily document things... it was messy because it was somebody else's job before. If it had been mine from the start, it would have been less hassle.

A solution proposal is an idea for how to solve the problem. The idea could be at a very high conceptual level, or at a more detailed level, but the solution proposal occurs before any development work has been made regarding the proposal. A number of mental processes such as brainstorming, work toward a solution proposal. The solution proposal is then focused on, evaluated and simulated in the mind as to how it would work. In a collaborative context, a solution proposal needed to be collectively understood among the developers. When a solution proposal did not to work, engineers viewed the proposal in different ways.

The forming of the solution proposal was based on the problem representation and the goal for the solution. Jack's task involved a problem where users needed to be able to extract and present information regarding themselves on their web pages. The information to be extracted varied widely. Jack's goal was to produce one portal where users could customize their extraction and presentation based on the don't-repeat-yourself (DRY) principle. Jack then thought of a solution that would be an in-house content management system. Jack said:

So broadly speaking, if someone browses into a content management system, similar to Genular, Mapvo, Microsoft Matrix, these kind of things, and then on the other side, [there is] something similar to a portal gateway and [you] can see, for example, university information at one place. You can update your profile and enter publications to [it] and nominate your publication to 2008 PBRF. And from there you can pull that data into your [profile], and from there you can pull back your data into your web profile. All the broad problems in one hit, following the DRY principle. If you put one information in one place, you shouldn't have to do that again in another place. You can easily call them from any place.

The next stage is the end solution. The end solution is the actual solution that is developed to solve the problem. This study defines an end solution to include versions of a solution still under development right up until the version that finally gets implemented.

End solutions are related to mental processes such as compromising, extrapolating and viewing differently. Through compromising conflicting system requirements, a middle ground end solution can be reached. End solutions that are inaccurate are attributed to the inaccurate extrapolating by the engineers.

An end solution as a whole may contain within it front- and back-ends, modules, programming codes, and frameworks. Front-end refers to the interface that users see and interact with. A partially developed front-end, commonly called a prototype, helps engineers refine the design, and through client interaction with the prototype, the engineers can better understand problems and uncover what the client actually wants but had not realised beforehand. Jessica said:

But at the same time the client probably, during that time, didn't understand fully what they needed until they received... the report, and then they decided that they needed something else to be added on to that report.

Indeed the front-end was found to trigger new client requests. Sam was originally requested to put effort on the back-end of the securitisation because the system is back-end oriented, and was asked not to concentrate on the interface. However, after seeing Sam demonstrating the system, the clients then requested that Sam beautify the interface. Sam said:

One interesting aspect of happiness is that the system has almost no functional requirements involving users. Not almost no, but it has, very just sort of 9 to 10 balance. I mean only 10% of that is about what users do. And the other 90% is about essentially maintaining all these contracts and transactions and stuff like that. And all of the hard stuff doesn't really involve users. And ...it's interesting given that, and given that they know that, and given that they gave us at the start, very strict instructions to say, make the users' side of things as minimal as possible, it's just, you know, snap it on, and we did that, and it's that grey-ugly-boxy application as result. ... but it's interesting that they really want us to go and make it look nice now, like even though the system's not really.

As a form of prototype, the front-end goes through iterations in order for it to be as close as possible to the requirements. Bill said:

Like there's lots of ...versions that you're going through. I think also going back to the people who were responsible for the project at various stages and say[ing], "Is this starting to look like what you were expecting?"

Back-end is usually where the data reside and are processed. Some of the tools or technology used for the solution already provided parts of the back-end of the solution. Bill said:

In terms of which fields to be included, that kind of thing, it is influenced by the database that we were going to. Like the system that they've done at AX has a particular standard that you use, how you name things, how you do the narrations and things like that. So we'd follow that standard for the design.

Both front-end and back-end were involved in mental processes such as understanding, breaking down, designing, and viewing differently and were worked on during the building. The back-end must be developed to match the business process that the software is intended to serve. Patrick said:

I needed to know sort of how payroll works and what sort of the different attributes in the database, and in the main variables; what they represented, where they flowed on, how they finally ended up in the database, and why.

A module is a sub-system of the solution that the user views as performing a distinct function. In Jack's case, after some time working on his task, he was instructed to focus on the MyResearch module of Bellacms that supports the information needs of the PBRF round. Jack said:

Instead, what happened was a thing called, originally it was called MyHome, then it was called MyResearch. It's all about the academic and basically managing ...a piece of research and the likes of the researcher that included the CV, PBRF, profiles, all those kind of things.

Programming codes are symbols and rules used to represent instructions to a computer. Several mental processes work on codes including breaking down, simulating, focusing and understanding. Codes are broken down based on what the instructions are intended to achieve. Engineers simulate in their mind the programming codes to understand what they do.

A framework is a software platform used to develop applications and software solutions. By using frameworks, engineers can work faster because they are able to concentrate on developing the specialised part of the software rather than using their time to develop procedures already built into a framework. Jack and Patrick modified their frameworks in order to allow certain functions of the software they were developing to work.

Sometimes, the framework was not modified, rather the engineers viewed it differently to see how to allow a function to work. One way was to make the software not use the framework for the function, instead invoking specially written codes of the software just for that function. This rendered a portion of the framework redundant. Patrick said:

So we had to find ways around it. Either code things differently, or try and restructure the application so that [we would] not have to do the call.

5.3.3 Solution success

This study categorises five indicators as representing solution success based on interviews with the engineers in this study. These are quality, functionality, cost, delivery time and user satisfaction. It was not the purpose of this study to look at how other factors affect solution success. However, these indicators are important to engineers as they work.

Solution success is related to mental processes. Specifically those mental processes include compromising, evaluating, brainstorming. When engineers were concerned with cost of their project going over the budget, they went through the process of compromising. Cost and functionality were factors evaluated among solution options or tools to be used. When engineers are concerned about the delivery time of the solution, they avoided the process of brainstorming all possible options and preferred to straight away focus on the obvious option.

Solution success is also related to development activities. In particular, the inadequacy of the testing activity resulted in a late delivery time. In Patrick's case of the payroll processing module, it occurred probably because the testing was inadequate and there were more bugs that had to be resolved prior to incorporation of the ported module into GHI Payroll software. Patrick said:

Well, it got ported on time, but it hadn't been fully tested on by that point, so there were issues. If there had been a concentrated period of testing on it, then that would have helped speed things up.

Problems with the information affect solution success, as in the saying "garbage in garbage out," it was clear in the securitisation systems developed by Sam and Rick that the inadequacy of the requirements resulted in poor quality software and a late delivery. In Jack's situation, frequent changes to the goal of his task resulted in a late delivery time.

Success indicators affect each other and client satisfaction is affected by the delivery time. Sam said, "They [the clients] got annoyed that it wasn't delivered in the time frame, and they got annoyed that it kept extending out."

5.3.4 Problem information

Engineers relied on information about the users' problem in coming up with solutions to the problem. Problem representation specifically represent the business process, the existing system, the requirements on the solution and the solution goal. This study looks into the medium or source of information and the content of the problem

information. External aid is a medium that lays out information, and contains problem representation. Similarly the client request is a source of information that contains problem representation.

External aids are one of the sources for problem representation. Jasmine said:

So, there were a number of documents that were written by [the] manager [of] student administration. Last year, they were sent through to me, kind of their wish list. What they kind of, what they're trying to achieve. And another one that was previous to that, these were all kind of background. And then the one with information on the current feral system, and what its functionality is doing.

External aids are used for, and are the outputs of, numerous mental processes. The processes of breaking down, understanding, focusing, and deciding and selecting use external aids and externalising and designing produce external aids. These are explained in Section 5.2.

External aids are used to facilitate collaborative work. Sequence diagrams prepared by Sam were meant for other developers and not just for himself. Sam said:

Sequence diagrams, I mean, that's the level that I've already got in my head, so I don't, I do that for other people's benefit, rather than for myself.

The client request refers to what the client explicitly asked the solution to do. Client requests are used as basis for the mental processes of compromising and designing, and for building. There were cases when a client request was incongruent with the goal of the solution. In Rachel's case, the request from the users to change the database structure into a one-one relationship went against the goal of the Program and Paper module. Client requests go through iterations. Jessica said:

Yes, you could have like small changes that come in. So one small change, ...then next week another small change, but if you get a lot of change requests, you can group them together, and then develop them together and then release them together. To make them a bigger, bigger release. But the iteration within one change could, like if there is one change it could be just one iteration if it is simple change. Or if it is quite complicated or if there are bugs, then you could actually go through one change and then keep fixing until it works for about, maybe three times even. So within the iteration, there is an iteration.

Sam said:

So sometimes you've been told, "Oh we need to do this process, and it works in this way, things go in this way and go out that way." And then you know, a couple of months into the development, they'll come back and say, "Oh, no, we need to reassess that every time this happens."

Although intuitively the client request is expected to be the source of the solution object, sometimes solution objects actually triggered client requests. As mentioned in Section 5.3.2, the front-end of the securitisation system triggered Sam's client to request for beautification of the interface. Similarly, versions of an end solution triggered a new user request. Barry said:

He actually requested that when I had done ...the yes/no thing as requested, when I originally had this page that said, "Do you monitor your website (yes/no)," and a radio button to go 'yes'. This was on like I think the second or third iteration. He wanted this yes/no thing to appear. And then he decided on this. We just saw the table that I did underneath it, underneath this form. So that wasn't in his original design.

In John's case, the end solution triggered a change or refinement of the client request. John said:

And then [the] user wanted changes, because it was then that I sat down with him and saw him use the system and how he used it and the function that he wanted available. Like he wanted to just key in a number and then tab away and have all the stock and quantities show for that item.

A goal is what a solution is supposed to achieve. As mentioned earlier in this section, what the solution is supposed to do and what the user wants the solution to do may differ. Patrick's task had a very clear goal, which was to port the module from Delphi to C# without modifications to the interface and how the data are processed. Because the goal was clear, what needed to be done was also clear, even if it was tedious. Patrick said:

It hasn't been as complex as some of the work that I've done ...in the past, or sort of now, because this involved basically a straight port. The goal post we pretty sort of, you can actually see what you needed to do.

Measuring against the goal, Jack regarded his solution as a failure because it did not achieve the original goal of becoming the university content management system, even when the solution focused on the MyResearch module as the university requested. Jack said:

But as a project I would consider it a failure, because, well at least from the development team's point of view it's a failure because what we were aiming for was something much larger. And what we understood what the university had committed to is something much larger.

Goals that iterate frequently or keep on changing, commonly referred to as 'scope creep' result in a failure for projects to meet delivery time. In Jack's case because the short term goal kept on changing and expanding, he found that he always needed extra time. He said:

That's the bit that we're lacking, and that's why we ended up just always ask[ing] for more time, and justify why we take more time, as the project just got larger and larger and larger.

The requirement is similar to the goal, though usually it is more explicit, detailed and it is written down and agreed among the project stakeholders. Requirements are solidified through communicating with the users. Jessica said:

So for example the consultants would collect all the user requirements, like analyse what was needed, and then they would communicate with us, the developers. ...we would investigate the system to see how we could implement this change, and then estimate how long it would take. And then this would be put into a document which then the consultants would send back to the client for confirmation. And then once the client confirms, the consultant would come back to us, and we'll develop it. After we develop it, the consultants could test it. So basically there's some information coming back and forth between the client, the consultants and the developers.

Requirements are also solidified with collaboration among the developers. John said, "at the start, we had to sit down with the users and management, and me and another employee from here. And we talked about the system and their requirements."

Engineers from ABC Technologies dealt with formalised requirements. Jack, Sam and Rick's task involved rather ad-hoc requirements that iterated frequently. Part of the reason for the unstable requirements was the lack of understanding by the user of their own business. Sam said:

I tried to do some requirements gathering and did that with the client. One of the problems you sometimes encounter is that sometimes the client will often not know their own processes very well...But in reality that wasn't quite the case you know. They knew they probably didn't have even a very fine grained idea of how they make their money. They had a very high level idea of it, sort of missed some of the detail.

Due to the rather ad-hoc requirements, Sam and Rick had issues regarding the quality and delivery time of their system. The system functioned inaccurately because the formula or algorithm for the amortisation was not obtained until very late into the project. Notwithstanding the inaccurate output due to the incorrect formula, the system worked well beyond the clients' expectation.

The amorphous requirement also caused a late delivery of the system. Rick said:

Well we did spend a lot of time on it and then discovered that it didn't work. And then spent more time on it. But we did spend quite a bit of time actually to get it to work. But then there was quite a bit of time waiting to get the documentation.

Problem representation is how the problem is represented to the problem solver or engineer. It forms the basis for the solution proposal and may accurately or inaccurately represent the actual problem. John said:

Having known that these guys [were] going to ship from one location to one location, and not from one location to many locations, so I didn't need to provide the functionality and allow for more than one "to" location. Yeah, which meant only one document would be generated. It meant that there would be only one document ID, so the report was easy too, if I log on to reporting, and a lot of things.

Mental processes that use, work on, or produce problem representation include breaking down, understanding, viewing differently, evaluating, designing, externalising, and learning. How all these processes interact with problem representation is discussed in Section 5.2.

The more accurate the problem representation, the less iterations necessary. Barry said, "But very rarely there's ever changes. And if there is it's just you know, minor textual changes on the webpage." If the problem representation is inadequate, it is difficult to properly organise and structure the project task. Sam said:

I was asked to give estimates and help with doing a plan of how it broke down and stuff like that. And my response to that was very clearly to everybody that I was involved with [was], "we can't do this because we don't know enough."

The inaccuracy and inadequacy of the problem representation obliges engineers to communicate with the users. Barry said:

And the first set of iterations that [Ali] had, he was ranking a specific matrix which are used to analyse functions and loop side, what part of the survey it's about. And he had like two or three that meant the same thing but were worded differently. And he had a couple that just made no sense at all, and I pointed this out. I said, "These three questions here are pretty much asking the same thing. And this piece out here, I have no idea what you're talking about."

Collaboration with developers is related to problem representation in two ways. Firstly, other team members may be the source of problem representation issues. Jack said:

And then as I say our focus became on replacing Research Master. So our main client was the publications office, and what we actually had was [Trudy] who was part of the publications office. She became our domain expert. So we were all working with her on a daily basis, and she was literally defining the requirements on a daily basis.

Collaboration with other developers also helped produce a more understandable problem representation. Sam said:

Because he [Rick] is very good at critiquing things and picking up on details and stuff like that, and I often worked with the other guys for those reasons because their different perspectives really helped you to produce something that's more understandable to people... if I'm just left alone on my own devices, I produce something that is very hard for other people to understand.

Business process is a set of activities or procedures within the user organisation to achieve organisational objectives. Information regarding business processes is essential for and used in the engineers' mental processes such as breaking down, designing and learning, and development activities such as configure.

Some solution proposals involved changes to the business process. In Bill's timesheet approval, the user needed to change how they process the allocation of discrepancies between the actual cost and charged cost. One way to understand business processes is through collaboration with other developers. Rick said, "You mean the help that I'm getting from him [Sam]? ninety percent was just business logic."

Engineers also reported on the existing system that their solutions are supposed to replace. Rick said:

The system ...does three things. It manages the money aspect from different bank accounts based on rolls which they define, and the other aspect of it is amortisation, which is how interest and repayments are populated against the contracts. So it's a two part system.

In order to perform his porting task, Patrick needed to understand the current Delphi-based Pay Processing Module. He said:

It was mainly going through the code [written in Delphi], and from the code, sort of determining ...what the code was doing and what ...that step was sort of there for.

5.3.5 Knowledge and experience

There are mainly two types of knowledge required for engineers to perform their problem solving task: technical computer-related knowledge and application domain knowledge. Application domain knowledge refers to knowledge of the business domain where the software solution will be applied.

Knowledge was obtained through formal education or through experience. Jessica worked on accounting and ERP systems and had an accounting degree. Bill, Jack and Kyle did not receive formal education in computer related fields, but they gathered experience through years working in IT-related work.

Experience with similar tasks provided Barry with application domain knowledge regarding academic surveys. He said:

I've done enough now so that I can identify whether for example, a question should be, you're expecting a sort of a long paragraph-type answer, whether they're expecting someone to choose from a list of options, whether the question implies some sort of from here to here range.

Jasmine said:

I had migrated an application that they have. So I have just migrated it, therefore, all of the requirements were already laid out for me. I just put it into a different technology. So while doing that, I just get to know currently what it is, currently what the application does. I would listen an awful lot about the business processes and they have got an awful lot of manual business processes that they do.

On the other hand, other engineers have a rather mixed level of application domain knowledge, some reporting having almost none. Sam and Rick have very limited knowledge regarding businesses that securitise their receivables. Kyle had very limited knowledge of how the payroll is processed. John reported to have some application domain knowledge although he acknowledged that it was inadequate. John said “Not, not enough. I didn’t know enough about the user and how the user would actually use the module until I sat down from where I did the first test deployment.”

When knowledge was insufficient the engineers did background research. Patrick said:

True except that we were starting up using C# as well so, so we were like, this was the first use of C# for me. And so there was a piece of learning the language even though I taught myself a sort of C# 6 months previously.

Insufficient technical knowledge of the tools and technology made it hard to use the technology. John said:

Because you have to allow for the user to make mistakes. You have to provide information from various areas to the users. The user expects everything that he would, before access from opening three or four different forms, to be in front of him in one form. Yeah, one of the difficult tasks was composing the view, but that was more because training than anything else.

To quickly acquire the necessary knowledge to execute the current task, collaboration with other developers was key. Patrick said:

I was working with somebody who was an expert in the field. He’s been here for 14-15 years now, so I was working very closely with him. So that’s how I gain the domain knowledge.

Sam said:

Even though superficially I probably influence things a lot, I get a lot in terms of listening to other people’s perspectives and learning about how they do things. And that’s helped me to learn all sorts of things while in pair programming.

Experience was relied upon when developers go through the mental processes of breaking down, designing, extrapolating, understanding, and deciding, as discussed in detail in Section 5.2. Experience with similar tasks was also used when engineers communicate to users regarding what changes are needed to the system. Barry said:

Do I think that my suggestion might have changed the way they do things? Well I would like to think so. It's mainly just because I've done a few of these now. I've done lots and lots and lots of them now.

5.3.6 Structure

Structure is about the formalisation of the problem solving task including the formation of a project team, the use of project management techniques such as planning and the segregation of development duties.

To perform the structuring, it was necessary to decompose the problem task first. This allowed the proper segregation of duties among developers. Patrick said:

Because you couldn't break it up too much into interdependent steps, but enough for people to be able to work on their own area...And so you could say, "OK, so these are basically these steps..." but then yeah, I ended up identifying other steps that were basically sort of just pieces of code. And you think, "Ok, this piece or sub-piece is reasonably self-contained, so we'll make it into a task."

Sam supported a connection between breaking down and structuring and the benefits accrued to this. Sam said, hypothetically:

Whereas if you separate things up so you can work on everything at the same time and the same sprint, then you don't get so many bottlenecks. And you have more, what I call, parallelism in your system if everybody can keep working, and not ...hold up anybody else.

Structure is related to collaborative work. With structuring and segregation of development duties, the collaborative work among the developers was made to become coherent. In ABC Technology, the appointment of a project team and the segregation of roles were made very clear. Jessica said, "We were both developers. So we would basically do the same thing and, say we have a list of developments, we would divide it out between us." Bill said:

So we kind of, because we were working at the same time, we reach a point where, although at different points you know, we were all actually ready to test together.

On the other hand, due to a lack of formalised appointments on the project team, the collaborative work among developers was made difficult. Jack said, "Yeah, so it opened it up from the word go, and developers would come in and co-create. The problem was we really didn't have a proper structure for collaboration."

Project planning is adversely affected when there is inadequate representation of the user problem. Sam said:

I was asked to give estimates and help with doing a plan of how it broke down and stuff like that. And my response to that was very clearly, to everybody that I was involved with, was that we can't do this because we don't know enough.

5.3.7 Tools and technology

Tools and technology are used while engineers go through certain mental processes and development activities and several types of tools and technology were explicitly mentioned. Engineers in ABC Technology uses domain specific software as part of the solution they offer to clients. This included Microsoft Dynamics AX and Accpac. Bill said:

In terms of which fields to be included, that kind of thing, it is influenced by the database that we were going to. Like the system that they've done at AX has a particular standard that you use, how you name things, how you do the narrations and things like that. So we'd follow that standard for the design.

Some tools are general programming languages, such as HTML, Java, Visual Basic, C# and other tools speed up the development of applications, such as .NET framework and Application Express. Some are database tools, such as Microsoft SQL Server and related tools and others, NMock and ReSharper, facilitate testing of the solution. Tools are also used in coding.

Tools are used to assist mental processes such as understanding and designing, but the limitations of tools trigger the need for compromising and can make building more difficult. The connections between tools and mental processes and activities are discussed in 5.2 and 5.3.1.

Engineers reported facing difficulties in building activity due to the restrictions or limitations of tools or technology. Bill said:

The solution we used has got quite a lot of grids, like tables, with standard-kind of headers on the top and then sortable columns of data, which can be selected and unselected..., and [in] ASP.NET ..., you can edit one row at a time but you can't go and edit lots of rows all at the same time. Whereas in html that is possible, but it's just the .NET framework does not allow you to do it. And so because of that, we had to find other ways of editing lines, and actually try to avoid editing lines as much as possible because it was already limited.

However some of the limitations were actually purposeful in order to limit the user of the tools to only use it the way it was supposed to be used. Kyle said:

The only restriction that I could think of I've come across before would be the mocking framework, NMock. It's got a few restrictions like, they aren't necessarily bad restrictions, but you can only mock interfaces, mock classes, rather than the objects themselves. But I mean that's not a bad thing, because that's how we should be working anyway with mock.

Some tools fit the solution proposal and some do not. In developing the database part of his solution proposal, Barry found Microsoft SQL Server nicely fit the task. Barry said:

Microsoft SQL Server is really flexible. ...I could do the database lots of different ways I guess. What guides how a survey looks and how it reacts to the database is determined by the structure of the survey, not on the tool that I'm using. The webpage that I produce has got nothing really to do with the fact that I'm using a text editor to produce it. Or the structure of the database has nothing to do with the SQL Server.

5.3.8 Iteration

Iteration is considered in this study as the repetition of the object or activity, primarily with the aim of producing the end object that is closest to the problem solution. Iteration is very prevalent across the engineers in the study and it affects the problem information and solution objects.

Problem information, such as the client request and requirements go through iterations. Jessica said:

Yes you could have like small changes ...that come in. So one small change. Then next week another small change. But if you get a lot of change requests, you can group them together and then develop them together and then release them together. To make them a bigger, bigger release. But the iteration within one change could... be just one iteration, if it is simple change. Or if it is quite complicated, or if there are bugs, then you could actually go through one change and then keep fixing until it works for about, maybe three times even. So within the iteration, there is an iteration.

Sam said:

Unless you've got really, really good BAs (business analysts) or something, which I've never experienced, your requirements and your scope are going to change.

Solution objects, typically the front-end, go through numerous iterations until they address the problem or closely match the user expectation. Bill said:

Like there's lots of ...versions that you're going through. ...I think also going back to the people who were responsible for the project at various stages and say[ing], "Is this starting to look like what you were expecting?"

It was reported that building activities go through iterations as well. In Barry's case, the way he handled Ali's web-survey development was to focus on the interface first and then the building of the interface in order to minimise iteration on the whole web survey solution. Barry said:

But changing the whole thing, as [Ali]'s survey would have done, would have been a lot of work to do all three components, the web page, the database, and the script. Which is why we just did his one, we built the web page only through all those iterations.

Iterations affect the end solutions by increasing the end solution's complexity, by refining the end solution, and/or ultimately bringing the solution closer to what the user wants or needs. Barry said, "But ...his one went through a few iterations, and it became more complex and a bit stronger than the existing one." The web survey started off being simple, but with each iteration, the web survey became more interactive through the use of Java rather than just html.

Sam said, "It [iteration reviews] was useful in that we could get answers to questions about details." Jessica said:

Yes as much as we try to get enough details on the first go, ...but sometimes you do not get the details until you're halfway through the cycle. And that's where you go back to the customer and get new information.

Kyle said:

Which is where I think the agile methodology excelled. Because you're delivering to your client, even if your client is part of the same company, delivering to them frequently. And they've got a much better idea of whether what they've asked for is what you're doing or not.

More iteration, however, exerts more time and delays the final delivery of the software. Kyle said, "But it took longer than thought earlier because of changing requirement." Hence there is a trade-off between solution refinement and solution delivery time and the iteration of the goal of the solution results in scope creep. John said, "Um yeah, which ended up, which means that we ended up in the situation of scope creep. Where ...the scope of the project just blew out completely."

Iteration may be induced by having other developers collaborating on the development task. Kyle mentioned in his case that changes in requirements were not because of a lack of information collected at the beginning, nor because the client wanted those changes. The iteration was because the architect was not happy with how the module looked.

Bill said:

Probably a quarter way through the development, we already had these tables and we actually worked through and changed them because he came on a bit later, but because of his experience, "this is, if you do it this way, it's going to be easier for upgrades and things in the future"

5.3.9 User understanding and communication

User understanding is one of the issues that affect the problem solving task of engineers. In particular, Sam and Rick indicated insufficient understanding of the users. When the

users' understanding of their own business is lacking, the requirements for the solution would be unclear. Sam's client did not know how their existing system processed the amortisation, and therefore, Sam had unclear requirements.

On the other hand, the understanding of what the users actually wanted from the system was improved when they were presented with the solution's front-end. Jessica's clients did not understand the report that they needed until Jessica showed them the report generated by the proposed system. Jessica said:

But at the same time the client probably during that time didn't understand fully what they needed until they ...they saw the report and then they decided that they needed something else to be added on to that report.

Communicating to the client directly is a source for the mental process of understanding. Jessica said:

Really communicate with the customer to find out what gap they need to be filled in. Sometimes we would meet up with them, sometimes they would email us what they need.

And in addition to requirement documents prepared by other members of her project team, Jessica obtained information regarding requirements by directly communicating with the clients. Jasmine said:

So, for example, the consultants would collect all the user requirements, like analyse what was needed, and then they would communicate with us, the developers.

When the accuracy of the problem representation is questioned by engineers, they seek confirmation by communicating directly with the user. Barry said:

If I think a question is wrong, I'll say it. I'll say, "Look, you're asking this question you've already asked up here. You're doing the same thing again", or, "You're doing it in the wrong way", or, "This question and this question kind of asking the same thing, do you really need both?"

5.3.10 Team and collaboration

Except for Barry, all the other engineers performed the problem solving task, to a varying extent, in collaboration with other developers. Jack started by working alone and had a project team only for a portion of the time. Patrick worked in a big, dynamic team, where extra members were sought for certain tasks at certain times. Kyle was the sole developer although he had architects and consultants that guided him. Sam and Rick worked together with a few other staff in their organisation for the securitisation system. Jasmine, John and Bill had proper team members throughout the life of their projects.

Rachel worked fairly alone but was involved in an interim solution developed by another developer, therefore, it was more of a one-way collaboration. Jasmine worked alone at the beginning but talked a lot with people who were involved in the development of the existing graduation system and later on, programmers were assigned to work with her.

Collaborative work played a significant role in the problem solving task, and was a part of learning, deciding, verifying, designing, focusing and understanding and reviewing codes. On the other hand, by not working collaboratively in a two-way manner, Rachel reduced the need for externalising. When working collaboratively, engineers are obliged to standardise and clean the programming codes.

5.4 Summary

This chapter discusses the mental processes software engineers go through in problem solving, and other important components related to it. The nature of the relationship between the mental processes and its components including development activity, problem information, solution objects, iterations, and collaborative work are explored. Extracts from the interviews are presented to support the findings.

In Chapter 6, a general model of software engineer's problem solving will be presented that summarises the detailed findings presented here in this chapter. The implications of the findings and suggestions for future research will also be presented.

Chapter 6

CONCLUSION

The preceding chapter discussed the results of the study in detail with supporting evidence from data collected. This chapter discusses those results in a broader sense and proposes a model to represent the mental processes of software engineers in problem solving. The reader is reminded of the limitation of the study in order to interpret the discussion more cautiously. Finally some directions of future research are suggested.

6.1 Summary of Findings

The research questions addressed are:

RQ1 What are the mental processes software engineers go through to solve problems?

RQ2 What are other important components in problem solving among software engineers?

RQ3 How do the mental processes interact with the other components?

To address these questions, the study used a qualitative method of inquiry and analysis. Eleven software engineers were interviewed regarding a particular software development project they had undertaken. These eleven developers worked in two main types of organisations; software development companies and within an educational institution. A grounded theory approach was used to uncover the mental processes, other components and their interactions in problem solving among software engineers.

Externalising

Externalising is the process of turning thoughts into something that can be visualised or heard and seven engineers externalised their thoughts. Externalising produced information on an external medium, which was then used for understanding and decomposing.

It was also found that externalising triggers the uncovering of a potential solution. During the interviews, it was unclear how talking would help someone find a solution

or an insight into the solution, however, Information processing theory proposed an answer to that question. The theory states that although an ill-structured problem has an undetermined number of possible knowledge states, the use of a heuristic search triggered by a certain problem representation, may trigger a knowledge state very close to the desired state. Therefore a thorough scan of the problem space is not required.

Externalising was needed prior to the activity of building the solution. The external medium produced by the externalising process would help the engineers see how each component of the design related to other components, and hence help with the programming.

On the other hand, externalising is not always necessary and may only be minimal. If the problem representation was already informative, engineers need not externalise their thoughts and when engineers work alone, they need not convey their thoughts to other developers through externalisation. Hence externalising is affected by whether or not a problem solving task is conducted in a team-work environment or individually, as well as the level of problem representation.

Understanding

The first most common process undertaken by software engineers is understanding. Understanding is the process of being aware of the nature and characteristics of the matter to be understood. Engineers went through the process of understanding the problem, specifically understanding the user business processes and the existing user systems. This information is represented in specifications and requirements documents and other external aids. Understanding the problem representation is also achieved through direct communication with the users and is easier when the engineers have had previous experience with similar tasks. Brooks (1977) asserted that understanding as a process in programming, involves acquiring knowledge of the basic elements of the problem via reading and questioning. Understanding is adversely affected when there's poor problem representation. It was necessary for the programmer to understand the problems of the user prior to the process of brainstorming ideas for how to solve those problems.

Engineers also needed to understand the solution objects, particularly when the project involves several developers. Engineers went through the process of understanding an interim solution, and the front- and back-end of the actual solution. Working in collaboration with other developers assisted the engineer's understanding of the solution objects. Tools and technology were helpful for enhancing the engineer's understanding of how the solution works. Tools allow the tracing of what specific codes do or

produce and then reinforce the engineer's understanding of the codes. To understand how modules and codes fit into the big picture of the solution, externalising ideas onto diagrams is helpful.

Some level of understanding needs to be achieved prior to building the solution, however, the act of building actually enhances the engineer's understanding of the solution object, particularly regarding details not realised before.

Decomposing

The study found that decomposing the problem is also a process commonly undertaken by most of the engineers. Decomposing is a process of breaking down elements into smaller elements. This finding supports the assertion that decomposition of problems facilitates a solution (Goel & Pirolli, 1989). Breaking down the whole development task into smaller tasks aids in the organisation and structuring of the task for project management purposes, such as assigning tasks to specific developers and allowing for concurrent work on the subtasks. The breaking down of elements of the problem allowed development activities for different parts of the problem to continue simultaneously. Breaking down the problem requirements also allows engineers to prepare a specifications document and test the solutions to different parts of the problem more efficiently.

Problem information and external aids are used to decompose the problems or elements of the problems. Solution objects are decomposed to refine each composition. In Kyle's case, interim solutions by previous developers were decomposed for refinement. Some of the solution objects were decomposed based on user steps, system layers and computer tasks. Decomposing was assisted by the engineers' previous experiences with similar tasks so they could reference how they decomposed the solutions for their current task.

Brainstorming

Brainstorming is the process of laying down all possible problems or solutions. This study considers brainstorming to be a process not limited to team-based problem solving. Notwithstanding this, brainstorming in a group setting is a well-adopted practice in software development (Shih et al., 2011). When an engineer thinks of all of the possible reasons for certain problem symptoms, this is called a diagnosis and is also a part of brainstorming.

Before brainstorming an idea to solve problems, engineers worked toward under-

standing the problem. Brainstorming ideas for solving a problem produces several solution proposals and these proposals are then evaluated based on their merits.

There were instances where brainstorming was not undertaken. The limitation of a restricted delivery time was often a factor found that hindered engineers from brainstorming and coming up with many possible ideas for a solution. In this case, engineers would only think of one idea of a solution, and immediately start work on that idea.

Evaluating

Evaluating is the process of looking at an idea or problem in terms of characteristics that can be weighed, such as benefit and cost, advantages and disadvantages, importance, and reasonableness. When several aspects are evaluated, sometimes they are ranked in terms of importance. Evaluating is part of the problem solving process of critical thinking. According to Weiner (2011), critical thinking is literally defined as "...[a] mental process" that includes "evaluating information" and that its outcome is problem solving (p. 82).

The process of evaluating ideas would occur after brainstorming. User requests and problem representation are also evaluated. Engineers may evaluate options using criteria such as cost, delivery time and functionality. After evaluating options, and selecting an appropriate option, building may commence.

In information processing theory, evaluating whether or not to continue with a knowledge state requires a comparison to the success of the previous knowledge state.

Selecting and deciding

Selecting and deciding is the process of choosing one option from several and proceeding to work on the chosen option. Selecting an option involved various criteria such as ease of and time required for development. Deciding on an option sometimes uses satisficing as a criteria. Satisficing is choosing an option so long as the solution could work. Once a solution is decided on and executed, it can be tweaked to eventually work the way it was intended to.

Engineers also selected or decided on which tools to use in their development task. One of the factors for deciding on a certain tool was whether there would be support for it and if it would allow for collaborative work with other developers. Looking at the number of developers worldwide using the chosen tool would provide clues as to whether or not the tool would satisfy those criteria.

There were several sources of information for making a decision. Firstly, engineers

use their previous experience with similar tasks as guidance. Secondly, they collaborate with and are influenced by other developers. Decisions are not necessarily made individually, and deciding on a tool was often in collaboration with other developers. Thirdly, engineers sometimes decided on an option by communicating to their users about the options for tools and obtaining their approval.

Focusing

Focusing is the process of concentrating on a particular item, while thinking less about other items. It is needed in problem solving for engineers because most of their problems contain many aspects that must be addressed both wholly and individually. At any particular time, engineers would focus on a sub-problem from one major problem, one solution option from various options, one module from a number of modules, and details of programming codes. Wiley and Jarosz (2012) asserted the benefit of attention focusing during problem solving allows problem solvers to control their attention, resist distraction and perform a narrowed search within the problem space.

Focusing was undertaken after evaluating and prioritising sub-problems, and after selection was made on a solution option found to be the most feasible at the time. External aid was used to enable focusing and focusing can also be done in collaboration with other developers. When done concurrently at another location with other developers, external aid use was more warranted.

Extrapolating

Extrapolating is referred to as the process of extending an idea into the unknown based on information of the known by making assumptions or generalisations.

Extrapolating is required during the design process. When information was too incomplete to proceed with the design, engineers needed to extrapolate based on their available information and through their experience in similar tasks. The output of that extrapolation would then go through the verifying process. In some circumstances, their assumptions may not hold true.

Designing

Reference to the process of designing was found among eight engineers. The earlier stages of design involved producing models of the solution. Part of the model is a proposed front end, or interface such as a mock screen or an interactive demo. Only after the design is agreed upon, can implementation proceed. To come up with the

design, engineers needed information on the problem, such as the particular business processes of the firm, to design the flow of the interface screen. This information would be included in the client request, some of which would be formally written in the requirement document that guides the design. The accuracy and detail of the problem representation makes the process of designing easier. Based on information processing theory, the more structured the task environment (in this case the more detailed the problem representation), the easier it is to arrive at the problem's solution.

An engineer's experience with similar tasks is an important source for design. Experience is used as a source for making extrapolations on design gaps that are not spelled out by the user. Design is a collaborative effort with other developers, where each developer's skills and experience are complementary, and it is aided with tools or technology.

Singer et al. (2010) found that designing was one of the activities software engineers perform. On the other hand, this study categorised design as a mental process. Singer et al. did not differentiate activities from mental processes and after initial data collection using questionnaires, They conducted a longitudinal study using a technique called "shadowing" where the observer would sit near the subject for a certain period (1/2 hour was used in their study) and write down the latter's activities. Singer et al.'s initial categories formed from the shadow notes did not include designing. Designing was found during the initial questionnaire that asked the software engineers what tasks they engaged in. Therefore, engineers self-report performing design, but observation through shadowing did not list designing as an activity. Therefore designing could well be a mental process, because the engineers are aware (in their mind) that they've performed it but this was not seen via observation, hence it was not an observable activity.

Verifying

Verifying is the process of comparing an idea or assumption with other sources to confirm that the idea does what it is intended to do. Verifying was done after engineers went through extrapolating in order to confirm their assumptions. Users and other developers were used for verification.

In a study about how an operating crew responds to accidents, Chang and Mosleh (2007) asserted that problem solvers search for the root cause of the problem by forming hypotheses (making diagnoses) and then verifying them. This study found that engineers verify an idea or assumption that they made through the process of extrapolating.

Viewing differently

References to “viewing differently” was found among eight of the engineers. Viewing differently is looking at an issue from many different aspects and viewpoints. Engineers viewed problem representation from different aspects, not just at face value. This included looking at what the users said, what they may have intended to say, and anticipating and designing for what they actually needed as opposed to what they said they needed. Being open-minded, one way of saying viewing differently, is one of the required tenets of critical thinking (Azar, 2010).

Viewing differently is triggered by previous solution objects that were found not to work. These objects include interim solutions developed by another developer, or the proposed solution by the engineers themselves that was found to be not feasible, or an underlying framework that was chosen, but then limits for achieving certain functions became apparent. While collaborating with other developers, viewing differently is also an attitude that allows solutions to achieve desired outcome.

Simulating

Seven of the engineers simulated in their mind how a solution proposal would or would not work. Besides echoing the studies by Giordano (2002) and Kim and Lerch (1992), numerous other studies related the use of a “what-if analysis” to understand user task performance, when certain conditions apply (Bostrom, 1989; Jeffrey & Putman, 1994; Zhao et al., 2008).

Some engineers simulated in their mind using formal computer processes such as pseudo-codes. Simulating is also used when reviewing code. Programmers would read codes written by other programmers and simulate in their mind the output of such codes to decide whether or not the codes would achieve their desired effect.

Ball and Christensen (2009) studied mental simulation as a process within design but this study categorised simulation and design as two different mental processes. The difference lies with Ball and Christensen considering design to be a task by itself. A possible compromise is that simulation and design could be inter-connected, although analysis on the interview transcripts did not suggest this.

Learning

Learning is the process of gaining knowledge about something through information garnered from available resources and experience. Learning was found among five engineers. Some engineers learned about the client’s business processes and nature

of their business from the internet, and others needed to learn more about the tools and technology they would be using; a learning process triggered by their inadequate knowledge .

Engineers learn about the application domain and tools in order to start with the building activity. On the other hand, they also learn while they are building. Patrick, who had not used C# in any of his prior tasks, learned the language before and during the actual coding work. Engineers also learn about specific aspects of their task by collaborating with other developers. Often they work with people who have more knowledge and experience with the task.

Compromising

Compromising is the process of coming up with a solution that best satisfies the most requirements, even if they are conflicting. The need for compromise is triggered by conflicts in the requirements. Conflicts occur between solution requirements and user request, user request and the cost to implement it, and user request and the tools or technology used for the solution.

In the first instance, some of the user requests went against what was actually required by the system in order to solve the user's problem. In other words, the user requested things they did not actually needed. In the second instance, additional requests from the user involved additional cost beyond the cost initially agreed on between the developers and the client. In the third instance, the tool or technology used in the solution has certain limitations that restricted the engineers from fulfilling some of user's request.

When faced with compromising, engineers targeted their end solution to be close as possible to the client request. Sometimes the interface was made to look like what the user wanted even though the underlying back-end worked in a different way.

While studying how software engineers strategise when facing faults in software, Lopez et al. (2012) found three strategising themes that include compromising. One of the three developers they studied related achieving a solution that worked but was still not satisfied on other aspects that is unclear. Another developer related the back-end of the application developed that worked but with the front-end handling was "not nice".

Development activities

A number of development activities were mentioned by the engineers in their problem solving task. Activities are differentiated by mental processes that have specific

meaning in the IS domain, involve more mechanistic functions, and are usually stated explicitly by engineers. These activities include preparing a specifications document, building, coding, configuring and customising, implementing, importing data, modifying a framework, porting software, reviewing code, standardising and cleaning, and testing.

Engineers reported that they prepared a requirements or specifications document containing problems or matters to be addressed in the new system. Before the document preparation, engineers decomposed or itemised user request. Preparation of the documents was usually a formal activity, although in some cases it was done on an as-needed basis. In that case, though, the development team had difficulty ensuring the accuracy of the solution.

Building refers to the general activity of moving from design into creating the solution in a more functional form the front- and back-end, framework and programming codes. Engineers build by referring to the user request and building occurs after the engineer has learned the appropriate technologies (if they are not already known), and understands the problems. An understanding is reached by externalising thoughts on a solution model, evaluating the solution options, and selecting and deciding on an option. Building also contributes to a better understanding of certain aspects of the problem or solution. Tools are used to assist in the activity of building.

Coding is writing a computer program using a programming language and is part of building. Engineers in this study used HTML, PHP and C#. Tools are used in coding for auto-generating certain codes, verifying written codes and suggesting better codes. One engineer refactored his codes.

Only one of the engineers review his code. Code review is a team activity where programmers inspect and review each other's codes, similar to the practice of pair-programming. Williams et al. (2000) reported that pair-programming produced solutions quicker and at a higher quality. Additional benefits of pair-programming found in their study include increased enjoyment and confidence among programmers, and a reduction in the likelihood of the programmers deviating from standard practices.

Some of the development tasks involved configuring and customising, particularly when projects were undertaken by ABC Technologies. They use off-the-shelf solutions to solve their clients' problems, and configure and customise the software to cater to the client's specific needs and requests. Configuring sets up the software solution by selecting the available options in the software and making it user ready. Customising modifies the software itself to suit the specific needs of the user. Business processes and user requirements were used to determine configuration and customisation of software

for the user. Singer et al. (2010) also found configuration as one of the activities engaged by software engineers they observed.

Modifying the framework was performed by two of the engineers. Modifying the framework is required when the framework being used as a base for the software solution does not support some software functions necessary for the solution to work for the user. Porting software can only proceed after modifying the framework.

Testing was conducted by engineers after building and coding of the software to determine if the software worked as planned to solve the user problems, and also to uncover bugs or problems in the software. Testing was done in segments, called unit testing, and the system as a whole was also tested (system testing), by comparing the output of the developed software with the expected output. Specialised tools, commercially available and developed in-house, were used to assist with testing. Implementation of the software on the users' site would commence after the testing was conducted and yielded satisfactory results. Inadequate testing resulted in software with quality issues.

Before implementation, engineers go through the activity of standardising and cleaning the codes. This is when, in the engineers' own words, they "tidy it up and get rid of all the copying and pasting and trying to standardise it". This activity is especially important to facilitate the possibility of other developers working on the same codes in the future.

Implementation typically occurs after all mental process and development activities have been completed. The study found specific references to implementation (putting the new software on the user's computer system), is conducted after design and testing. Implementation involves importing data from the existing systems to the new system. In the GHI Payroll software porting project of the payroll processing module, the importation of data was not needed as the project specification was to port the module in the main software only.

Problem Information

Software engineers sought information regarding user problems in order for them to come up with solutions to those problems. Aspects of problem information categorised in the study are; external aids as a medium of information, client requests as a source of information, problem representation that represents existing systems, stated goals, requirements and business processes. These aspects of problem information are used and produced by mental processes and development activities.

External aids are referred to during mental processes and development activities.

They also assist collaborative work among developers when engineers externalise their thoughts onto an external medium that can be referred to by other developers.

Client requests are explicit instructions from the user regarding what they want to be included in the software solution and to a certain extent, they are used as a basis for development of the solution. Client requests are sometimes incongruent with the actual goal of the solution, due to a lack of user understanding of their problem with their proposed solution. Client requests are sometimes triggered at different points along the solution process, or changed by the partial development of the solution, when users realise what they actually want after looking at a prototype of the solution.

Problem representation is how the user problem is represented to the engineer, and this may or may not be accurate. Problem representations are rarely static, although more accurate representations go through less iteration. Inaccurate or inadequate representation required the engineers to communicate with the users. When the problem representation was lacking, the organisation and structure of development project became problematic. Problem representation is a collaborative process among developers, in order to make the representation clearer and provide the problem representation as a source to others. Problem representations contain information regarding users' business processes and existing systems, as well as the solution goals and requirements.

Information on user's business processes and existing systems are referred to during mental process and development activities on the solution. The existing system itself would be replaced by the solution developed, while business processes may be required to be modified to a certain extent for the implementation of the solution.

Goals are what solutions supposed to achieve. Clearer goals helped engineers see what needed to be done more clearly. Scope creeps occurred when goals kept changing, and resulted in failure of the development project to achieve the original deadline. There were instances when goals were incongruent between those of stakeholders and the engineers.

Requirements are details that enable the achievement of goals, although they may refer to two different things. Firstly, a requirement refers to what is needed for the solution. Secondly, requirements are commonly referred to by engineers as documents that contain information regarding the requirements. These documents are produced as part of a standard development activity and on an ad-hoc basis. Requirements go through iterations and are solidified by interacting with the users and collaborating with other developers. Less solid requirements may result in a late delivery of the solution.

Solution Object

Solution object involves aspects of the stages of the problem solution and the parts of the solution. In this study, solution to user problem is in the form of software solution. The first stage of software solution was interim solution, where it was developed partially by a developer other than the engineer concerned in this study. The existence of interim solutions occurred in smaller problem solving tasks among the engineers. These small task probably had received less priority from the management, hence resulting in interim solution which was not implemented but passed to the next developer. It was found that interim solution was not much helpful in aiding the engineers to solve their task as they reported part of the complexity of their task was understanding the interim solution.

The second stage of the solution object is the solution proposal, which is an idea of how to solve the user problems, either at a conceptual or more detailed level. Solution proposals are arrived at through brainstorming and are understood between the collaborators. The engineers then evaluate the solution proposals, by focussing on specific parts and simulating them in their mind. Formation of the solution proposal also involved referring to the goal of the solution.

The final stage would be the end solution, although this study included versions of actual solutions developed to solve the problem right up until the version that gets implemented. The processes involved with the end solution include compromising, extrapolating and viewing differently. It was also found that the quality of the end solution is adversely affected when engineers inaccurately extrapolated.

Solution objects can be broken up into solution parts. The study found that the parts mentioned numerous times by engineers were front-end, back-end, modules, programming codes, and framework. Prototypes that show a version of the front-end of the solution assisted engineers in refining the design by inducing users to understand and uncover what they actually wanted and how the software should work for them. The front-end goes through more iterations than the back-end. One engineer concentrated on refining the front-end with the user until the user was completely satisfied, before building the back-end. Some back-ends are already partially provided for by software tools or technology. For example, Dynamics AX already provided Bill and Jessica with core parts of their back-end solution so they didn't have to build that themselves.

Similar to software tools, frameworks were also used as software platforms on which the engineers could develop the full solution. The use of frameworks allowed speedier development by allowing engineers to focus on unique aspects of the solution rather than developing the parts of the solution which are common to many programs. There were instances, however, when frameworks needed to be either modified or worked

around by the engineers when parts of the framework did not produce the desired effect.

Solution Success

This study found that engineers sought after several success indicators regarding the solution they built, specifically quality, functionality, cost, delivery time, and user satisfaction. Some of these sought after indicators affected the engineers' mental processes. Engineers reported having to compromise on the solution, in terms of how the solution accomplished certain functions, in order to not go beyond the the initial cost budget. Some engineers avoided brainstorming all possible solution options in order to focus on the more obvious solution option, when delivery time of the solution was of concern.

Solution success was also related to development activities. In one instance, the implementation of the solution module took longer because there was not enough testing on the solution prior to its implementation. When the implementation was carried out, bugs were found that needed attention.

Poor problem information resulted in a poor quality solution. In one instance, because there was a lack of problem information regarding the correct formula to performing certain calculations, the software solution produced incorrect results. This problem delayed the delivery time of the software until after the poor problem information was rectified. In another instance, the delay in a software's delivery time was caused by frequent iteration of the solution goal.

The sought-after success indicators also affected each other. In one case, the users were dissatisfied when the software was not implemented within the agreed time frame.

Knowledge and experience

Technical IT and application domain-related knowledge and experience assisted the engineers with the performance of their task. The engineers in the study were diverse in terms of their knowledge and experience, particularly in their application domain knowledge where some even reported having no application domain knowledge at all regarding the tasks that were studied.

Most engineers are equipped with a substantial amount of technical knowledge, either obtained from formal education, or solely from work experience. Some engineers faced the difficulty of using software tools that they had limited knowledge about.

Not only do knowledge and experience inform the engineers' mental processes and development activities, the engineers made efforts to acquire knowledge to help with the solution process. Engineers with very limited knowledge went through the process

of learning during their problem solving task in order to gather the knowledge they required to proceed with subsequent mental processes. Knowledge was also obtained when engineers collaborated with other developers. Transfer of knowledge occurred when this happened.

Structure

The engineers' execution of the problem solving task involved, to varying extents, a structuring, or formalisation of the task through the creation of project teams and the use of project management techniques such as planning and segregation of duties. The process of decomposing the problem solving task was undertaken to allow the segregation of development duties among developers. With proper segregation, collaborative work among the developers was facilitated and enhanced. On the other hand, when the formal appointment of project teams was absent, collaborative work was found to be difficult to manage.

It was not the aim of the study to examine team dynamics in software engineering problem solving, but rather the individual mental processes engineers go through. Therefore in-depth exploration of structuring components is not discussed. However, as it was found as an important element, so it is briefly discussed here (limited by the lack of research on the topic).

Tools and technology

Several types of tools or technology were found to be used by the engineers in the study. Tools included programming languages, an application development framework, domain-specific customizable software, database management systems, and tools supporting the verification and testing of programming codes. These tools assisted the engineers in the execution of their mental processes and development activities.

On the other hand, the tools themselves required a certain amount of learning among the engineers to allow them to be able to use those tools. Furthermore, some tools have restrictions or limitations that created difficulty with the engineers' execution of their activities even though some of the restrictions are built into the tool to guiding the user on its proper use.

Iteration

Iteration is a phenomenon that occurred with several engineers in the study with actions and objects. Particularly with the problem solving task among the engineers, iteration occurred with the aim of producing an end object that would be as close as possible to satisfying the requirements of the user. Problem information, such as user requests and requirements went through iterations. Solution objects such as the front-end solutions went through iterations until they functioned according to the users' expectations.

A collaborative working environment encourage more iterations and each iteration refined and/or increased the complexity of the end solutions. However, with more iterations, usually more time was required to come up with the final solution.

User communication

The user is an important factor in software engineers' problem solving because the original problem is centred around the user. This study, however, does not look at the user from the user's perspective, but from the engineer's perspective. Two issues regarding users are important from the engineers' perspective: user understanding and user communication.

This study found that the assertion that users do not necessarily understand their problems was true in several cases. In one case, the users did not understand the formula that would be required to process their data into useable information. They also did not know how their previous system performed the calculation, nor how to perform the calculation manually.

User understanding can be improved by the engineer's actions. In one case, the users only understood what they actually wanted in a report when the engineer showed them a version of the report produced by the prototype system he built.

When an engineer doubts whether the user understands his own request, the engineer would communicate with the user to confirm what the user represented as the problem is actually true. By communicating with the users directly, the engineers own understanding of the gaps in the problem improved.

Team and collaboration

Engineers in the study reported on collaborative work in various ways, except for one engineer who attempted a development of the solution individually. Some engineers worked on a formal project team. Some worked alone for certain periods of the project

as the project champion, and were given a team to work with part of the time. Two engineers worked fairly alone on an interim solution that was partially developed by other developers. One engineer worked on a large and dynamic team.

Engineers worked collaboratively as a way of enhancing mental processes and development activities. Performing work collaboratively produced more problem information, but it also required more problem information.

6.2 General Model of Software Engineers' Problem Solving

This study contributes theoretically by offering a model of problem solving. Based on the obtained results, this study proposes a general model of software engineers' mental processes for problem solving, in Figure 1. The figure represents a model of mental processes for problem solving because the analysis used mental processes as the core focus, while uncovering surrounding important components. The components are grouped into four broad factors; the problem solver, the environment, problem solving aids, and the solution.

Within the problem solver factor, there are three main components; mental processes, development activities, and knowledge and experience. In solving a problem, software engineers went through some of the mental processes including understanding, decomposing, focusing, simulating, viewing differently, learning, deciding, brainstorming, externalising, verifying, evaluating, compromising and extrapolating. There could well be other processes beyond those stipulated in this model. Mental processes occur in a complex cyclic and iterative manner.

The mental processes interacted with development activities or actions that the engineers undertook in developing the software solution. Although mental processes work on the solution internally, it is the actual development activities that physically produced the solution. Certain activities would be executed after some mental processes, either through a simple sequential manner, or a cause-and-effect manner. For example, the activity of implementation occurs after the mental process of designing. Activities also induced some mental processes to occur. For example, the activity of building induces understanding.

To execute mental processes and development activities, engineers would refer to their knowledge and experience particularly in the application domain and the technical IT related areas. Having relevant knowledge would facilitate processes and activities by

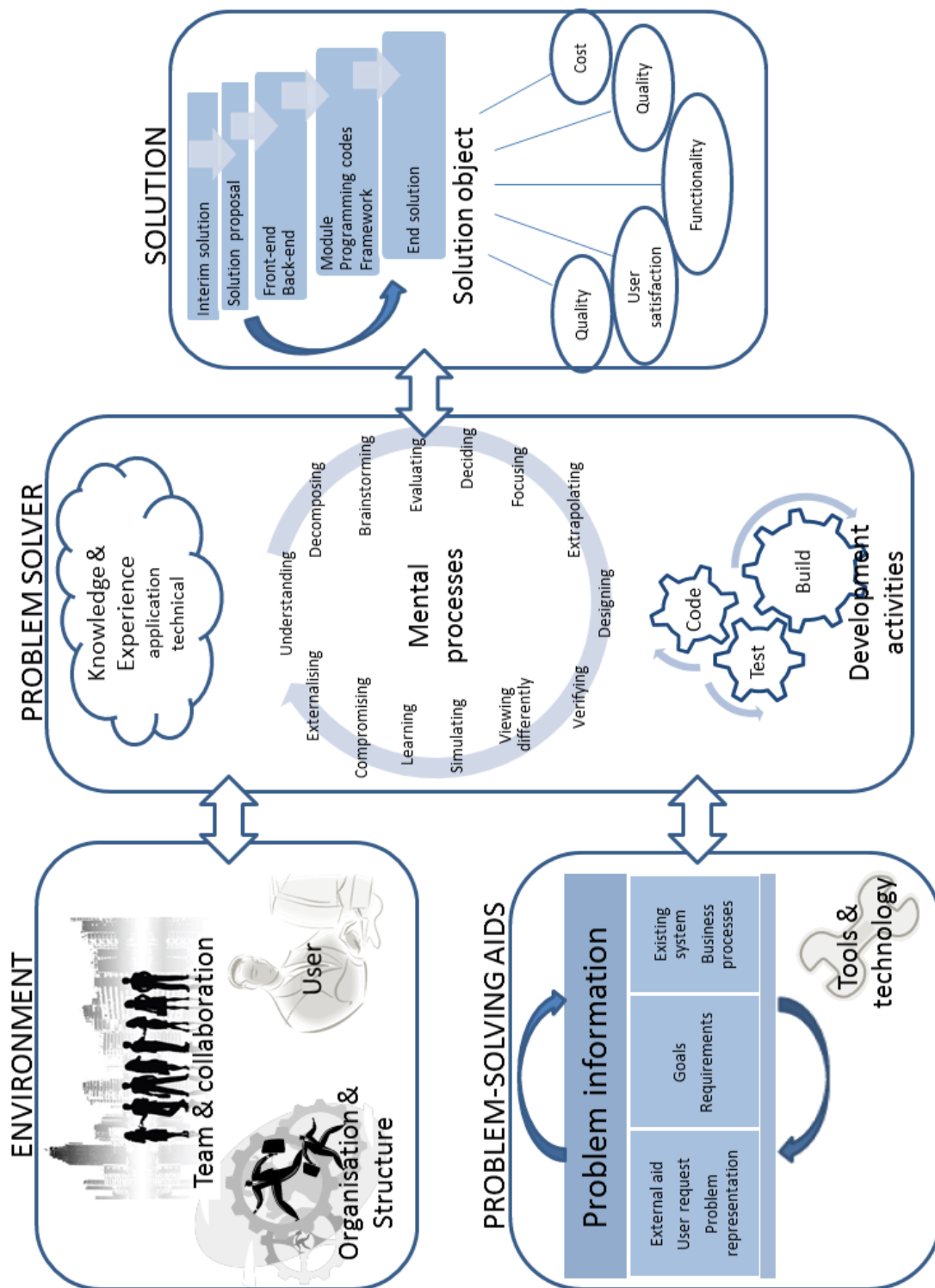


Figure 1

General model of software engineers' mental processes for problem solving

making the software development faster and easier. In addition, the mental processes and development activities would also add to the existing knowledge and experience of the engineers.

An engineer could not perform his problem solving task in isolation. He would not be able to solve the problems by executing mental processes and development activities based on his knowledge and experience alone. He would need to refer to problem solving aids, particularly specific information regarding the problem he is actually trying to solve, because the problem is encountered by users other than the engineer himself.

Problem information is obtained in the form of external aids (such as internet forums and Wikis) for general information regarding the problem, and user requests for more specific information. External aids are also used for specific information when they are accessed while engineers execute mental processes and development activities. The sources contain a problem representation which may or may not be relevant to the problem solving task. Representations are made regarding, among others, the goal and requirements that the solution is trying to achieve, and the existing system and business processes the users are engaged in that need to be addressed in the software solution. However, problem information does not only inform mental processes and development activities, it is also updated by the execution of processes and activities. Therefore, problem information also goes through iteration and with each iteration, the problem information should be more detailed and more closely support the final solution.

In addition to problem information, tools and technology are also problem solving aids that can also be used to facilitate the problem solving task. Some tools are essential to the development of the solution, such as programming languages and database management systems, and others facilitate faster and more accurate development such as software framework and developer tools. However, there may be trade-offs with tools for assisting development. While specific tools may automate development of certain parts of the solution, they may restrict innovation of the engineer. Engineers could find work-arounds that still allow for incorporating parts not addressed by the tools.

Engineers solve problems in an environmental context and not in isolation. Therefore, environment is an important factor in software engineers' problem solving. The environmental factors include elements of the user, the team and collaboration, and the organisation and structure. Since problems come from the user, engineers obtain specific information from the user. On the other hand, an engineer needs to be critical of information he obtains from the users as they might not fully understand their problem⁷.

⁷ This study uncovered issues on client understanding and communicating to client as important

An engineer that looks to improve his problem information by directly communicating with the users may gain a better understanding of both the problem and the solution.

The environment also includes teamwork and collaboration when engineers work with other people to solve the task. By having team members and collaborating with other developers, engineers could compensate for their insufficient knowledge and experience in executing mental processes and development activities. Outputs of processes and activities can be reviewed by other team members to ensure accuracy of the solution. Development activities can be arranged so that independent activities are executed in parallel to allow faster development time.

Teamwork and collaboration may happen in both formal and informal ways. Organisation and structure is a component in the environment that facilitates teamwork and collaboration. The organisation and structure includes the formation of project teams, segregating development duties, and the use of project management techniques. Without proper structuring through the formation of project teams and segregation of development duties, engineers would find it difficult to collaborate with other developers to perform their tasks.

The final factor in the problem solving model is the solution. It is the output of the whole problem solving exercise and has two components: the solution object itself, and the success attributes of the solution. Solutions go through stages of development from interim solutions, to the end solution that is then implemented. Solution objects iterate very frequently, with some parts iterating more than others. Parts include the front-end, the back-end, modules, programming codes, and framework. Each iteration should improve and add details and complexity to the end solution. Solutions objects, according to their stages, are worked on by the mental processes and development activities of the engineers.

Success attributes of solution objects include the cost of development, what functions it performs, the quality of the solution, how fast the solution is delivered, and how satisfied the users are with the solution. These attributes are kept in mind of the engineers when they are executing their mental processes and activities. So much so that not only mental processes and activities affect the success attributes, the mental processes and development activities are governed or guided by the attributes. The aim of achieving these attributes may indirectly benefit or hinder other attributes. For example, in order to achieve a certain level of quality, user satisfaction must also be

components relating to mental processes of software engineers. However, in the general model, a general component named "User" is used because users are integral to software engineers problem solving. There may be other important issues relating to users that must be considered in understanding how software engineers solve problems.

obtained but sometimes, when trying to deliver the solution on time, engineers will produce a solution that undergoes less thorough testing and the quality of the solution will suffer.

6.3 Suggestions for Practice

The mental processes uncovered in the study offer suggestions for best practice. The empirical description of behaviour and problem solving processes among systems developers may aid business practice improvements, the development of improved training, teaching and learning of methodologies (Hughes & Parkes, 2003).

Firstly, project leaders or organisation's information systems strategic planners could consider including a formal simulation process of the software development life cycle. As the study found simulation to be a common practice, it might be beneficial for simulation to be formalised in order to ensure that developers use this important step when designing systems that represent and address the problem most accurately. A suggestion for teaching simulation could be demonstrating how one developer does the program code and another programmer does a mental simulation of that code in his mind to verify accuracy of the code, a practice done during Patrick's task and in line with a recent concept of pair-programming (Williams, Kessler, Cunningham & Jeffries, 2000). Williams et al. (2000) reported that pair-programming produced solutions quicker and at a higher quality. Additional benefits of pair-programming found in their study include increased enjoyment and confidence among programmers, and reduction in the likelihood of the programmers to deviate from standard practices.

Secondly, it is suggested that developers (particularly when systems development is triggered by the developer), investigate and agree with the user on the system needs. This is to avoid a sudden drastic change in requirements that could lead to wasting time and resources. This study found this problem in Jack's case, where a lot of man-hours were spent developing a very complex system, and this time was wasted when it was decided to scale down the purpose of the system.

Thirdly, the importance of understanding the user problem can never be overstated. The study found that all engineers went through the process of understanding problem information, and although getting the requirements correct usually occurred throughout the software's development rather than just at the beginning, it still had a positive impact on the solution. Hence a close engineer-user relationship is recommended on all software projects in order to facilitate a clearer understanding of the goals and requirements. It may also be important for engineers to evaluate whether the user's

request actually represents the problem accurately or is merely a suggestion that could result in an unsuitable solution to the actual problem. This is based on Rachel's case where she had to filter some of the user's requests because they did not address the underlying problem at hand.

When a system's requirement is incomplete, as in the case where users are themselves unclear about what their needs are, three approaches were identified in this study that could be explored by developers. One approach is for the developers to design an all-inclusive solution that would cater to all needs. Once the actual need of the client was identified, the developers could scale-down the design. A second approach is for the developers to design a general concept first with as much detail as possible. Any detail left out could be uncovered later in the development. Similarly, the base layer or the software framework could be developed first that would allow specific modules to be added on later. The third approach is to concentrate on the interface in isolation first. Once the system interface is agreed upon with the users, development can proceed on the database.

Human resource departments should consider employing developers with extensive experience in technical and application domain knowledge of the systems that these organisations typically develop. This study found that all the developers had at least some technical knowledge, but some had additional application domain knowledge and that this was useful for them to develop mental representations of solutions. Although formal qualifications in IT are arguably important, consideration should also be given to developers who might not be IT graduates but have general problem solving skills, including the ability to "think outside the box", as in Jack's case.

The organisation should also provide a repository of relevant tools and technology that are clearly available for the developers' use. This would facilitate quicker development time as engineers would know which tools to use or look for in their task. A repository of knowledge, or a knowledge management system is also important, because the sharing of knowledge facilitates the mental processes and development activities of the engineer. When the engineers found that they lacked knowledge on certain aspects of the task, they could seek knowledge stored in the knowledge management system. Or at least the system could show the engineers where they need to go (besides training) to obtain the knowledge.

The common practice of formalisation of teamwork on a project is further encouraged. Engineers who had no formal teams when their development task required collaboration between developers found that their collaborative work was hampered. Whereas engineers working on a formal project team with proper segregation of development duties,

did not report difficulty in working with other developers. One of the specific ways that team structuring helped was the development of independent parts of development that could be carried out simultaneously to allow for speedier testing of the final integrated solution.

In the area of education, the mental processes identified could be incorporated in courses in software engineering university programs. This study found mental processes to involve problem information and solution objects. Students could be taught mental process skills by using examples of problem information and solution objects. For example, students could be taught about brainstorming ideas for solution objects to address the problems represented in the problem information.

6.4 Limitations of the Study

This study is purely qualitative in nature with an emphasis on depth rather than breadth. Therefore, the results of the study of eleven developers cannot be inferred onto all systems developers. The intention of the study was to explore what could be a phenomenon occurring among systems developers. Nonetheless, the study tried to address the applicability of the findings by using samples of systems developers investigating real systems and developers working in several types of organisations and on various sizes of systems projects.

Similarly, employing a qualitative method of coding and using the researcher's interpretation of the interview transcripts limits the objectivity of the study and results should be viewed from the researcher's perspective, as they are based on the researcher's interpretation. However, much care was taken to limit the subjectivity by using a pre-established method of data collection, iterative coding, and referring the coding constructs back to the participants and the literature, and obtaining feedback from the participants on the results to support summative validity. An inter-rater reliability test was conducted on the codes, and codes with weak correlation were discussed and reconciled (a detailed discussion regarding this concern was addressed in Chapter 4).

Thirdly, as was demonstrated when a mental representation of the task solution affects the problem solving performance, the true picture of this effect was not measured. The study's concern with solution performance was a one-sided view of solution success. As the study only interviewed developers, solution success was based only on the engineers' perspectives and not the users'.

However, the engineers' perspective in the interviews also included a representation of the users' views. Questions on success were conveyed by asking the engineers about

the feedback they obtained from the users, apart from whether the developers felt their project was successful. Therefore the measure of solution actually contained, indirectly and to a limited extent, included the users' views.

6.5 Future Research

One immediate topic for future research is the analysis of extra software engineers' problem solving tasks and making comparisons to the proposed model to strengthen or modify it. One of the weaknesses of this study is the lack of constant comparison that has been advocated for use in grounded theory. The proposed future study could be done to overcome this weakness before the model is applied to other areas.

This study found some themes regarding emotions being mentioned by several engineers. All of the text coded as "emotion" was coded about Sam. Based on the text, he shows some personal aversion towards the software project. When asked about how the project has been progressing after he left, he said "I haven't talked to the guys about it, but because I really don't want to know...I'm really glad not to be doing it anymore". However, he faced specific challenges with a positive attitude, in particular regarding having no domain knowledge about the user's business and therefore had to undergo the process of learning. Sam said:

You usually don't know anything about the business domain. And it's one of the interesting great things about my job was doing that. Learning about it.

Because the occurrence of the emotion theme was rare, it was decided to exclude it from the analysis. However, because Sam's case indicated the process of learning was done with a very positive attitude, it highlights another dimension to the components of problem solving regarding the factor of the problem solver himself. The study's model that contains mental processes and knowledge and experience, could also include emotion. At the very least, emotions may affect mental processes and knowledge and experience because all these occur in the engineers' conscious mind. Hence, future studies could investigate the emotional component as a factor of problem solving. For example, investigating how problem solving processes are affected by emotions.

Another area for investigation would be to use ethnography to get live data on mental processes. To understand real life problem solving requiring a longer time period, typical verbal protocol cannot be used. Numerous studies following Newell & Simon (1972) used verbal protocol analysis on subjects performing problem solving tasks experiments that took a short time (fractions of a day). This study employed retrospective interviews which did not capture live data. Future research could use ethnography. For starters,

a positivist approach to ethnography can be used to allow early focus on identified aspects of problem solving. This can be done by using the model proposed in this study as the framework with which to compare the data collected during the field work. In ethnography, the researcher is aware of the surrounding environment and observes the people in their practice. By following the model proposed in this study as a framework, the researcher would be able to keep his eyes open for the important components already suggested by the model.

Future studies could also examine the mental processing of software engineers in their problem solving tasks other than in software development. This study sampled software development (in varying forms of complexity and structuredness) as a problem solving task and did not look at other problem solving tasks such as debugging. Such studies may find support in the proposed model will enhance the generalisability of the model.

Another topic for investigation could be detailing the mental processes of problem solvers and how they relate to external and internal representations. This study identified and traced mental processes and their relationship to other components in a high level view, but due to the long term nature of the problem solving task, it was impractical to look at a lower level of analysis. Nevertheless, even with the limited detail of the result certain processes were found to be prerequisites for other processes that worked on problem information and solution objects. To verify and strengthen the relationships found in the study, a lower level, more detailed analysis should be conducted. This could be done by conducting a verbal protocol analysis where participants are observed during an experimental problem solving task and asked to talk aloud about their mental representation processes. The mental processes and activities are then linked to a content analysis of problem information and solution objects that developers use like flow diagrams or charts and entity-relationship diagrams, such as those devised by Giordano (2002).

Future related studies should use more triangulation methods. In addition to interviews, they could perform a content analysis on design documents and conduct observations of tasks engaged in by the developers. The use of the three techniques together would allow a comparison and verification between what problem solvers say they do and what they actually do. Care should be taken with regard to the intrusive nature of the observation, or any data collection method for that matter including interviews (Myers & Newman, 2007).

As mentioned in Section 6.4, the success attributes of solution objects was accounted for in the study mainly from the engineers' point of view, while the user's point of view

was collected in an indirect way. Future research could study the effect of the problem solving components on problem solving performance by measuring solution success more closely, via the user's point of view. Numerous studies have used user information satisfaction (Ives, Olson & Baroudi, 1983) as a way of quantifying a systems success. Such a triangulation of methods would enhance the value of the resulting theory.

The aspect of team dynamics could also be looked at as an important factor in problem solving. In this study, Rick's and Bill's tasks were facilitated with help from their team members. When Jack was given a team to work with on his development task, the team came up with designs that he thought none of them would have thought of individually. This suggests that collectively working on a system problem might improve the understanding of the application and technical knowledge of the developers, which in turn could result in well-thought out solutions. Lim and Klein (2006) found team-based problem solving performance was related to the similarity and accuracy among team members' mental models. Interviewing each member of a project team and probing the effects of team members on individual as well as team mental models would provide insight into this aspect of problem solving. Triangulation should be done among interview transcripts to substantiate each team member's point of view. This would allow the investigation of team dynamics (Carley, 1997) and thus has practical implications for supporting team-based systems projects, and hence increasing the relevance of IS research for practice (Avison et al., 1999; Vessey & Rosemann, 2008).

In general, this study and the suggested future studies above could also be applied to studies other than in the IS field. As a member of an accounting school, on a team with colleagues within the same school, this researcher could explore the mental processes of financial or management accountants in their problem solving tasks, assuming that the four main factors would be similar. The accountants themselves would perform activities following the mental processes they performed on problem information. Accountants would use their knowledge and experience in accounting aspects of the problem and the specific business context to rely on for the problem solving. In coming up with the solution, the accountant may involve other executives for consultations, including the user of the accounting information if not himself. The solution obtained could be measured for its success using similar attributes such as quality (dimensions of accounting information quality such as relevance reliability, completeness, timeliness are more established in research and practice).

BIBLIOGRAPHY

- Abran, A., Moore, J. W., Bourque, P., Dupuis, R., & Tripp, L. L. (2004). *Guide to the software engineering body of knowledge: 2004 version*. IEEE Computer Society, Los Alamitos, CA; Tokyo.
- Ahmed, F., Capretz, L., & Campbell, P. (2012). Evaluating the demand for soft skills in software development. *IT Professional*, 14(1), 44–49.
- Ammar, H. H., Abdelmoez, W., & Hamdi, M. S. (2012). Software engineering using artificial intelligence techniques: Current state and open problems. In *ICCIT 2012* (pp. 24–29). Hammamet, Tunisia.
- Anderson, J. R. (2004). *Cognitive psychology and its implications*. New York: Worth Pub.
- Avison, D. E., Lau, F., Myers, M. D., & Nielsen, P. A. (1999). Action research. *Communications of the ACM*, 42(1), 94–97.
- Azar, A. (2010). The effect of critical thinking dispositions on students achievement in selection and placement exam for university in turkey. *Journal of Turkish Science Education*, 7(1), 61–73.
- Aziz, K. A., Benwell, G., & Theivananthampillai, P. (2008). Uncovering the cognitive processes in software design task. In *Fourth Asia/Pacific Research Symposium on Accounting Information Systems* (pp. 343–362). Auckland, New Zealand.
- Ball, L. J. & Christensen, B. T. (2009). Analogical reasoning and mental simulation in design: two strategies linked to uncertainty resolution. *Design Studies*, 30(2), 169–186.
- Bostrom, R. P. (1989). Successful application of communication techniques to improve the systems development process. *Information & Management*, 16(5), 279–295.
- Brooks, R. (1977). Towards a theory of the cognitive processes in computer programming. *International Journal of Man-Machine Studies*, 9(6), 737–751.
- Bryman, A. (1984). The debate about quantitative and qualitative research: A question of method or epistemology? *The British Journal of Sociology*, 35(1), 75–92.

- Campbell, E. T. (2008). Gaining insight into student nurses' clinical decision-making process. *Aquichan*, 8(1), 19–32.
- Carley, K. M. (1997). Extracting team mental models through textual analysis. *Journal of Organizational Behavior*, 18, 533–558.
- Chang, Y. & Mosleh, A. (2007). Cognitive modeling and dynamic probabilistic simulation of operating crew response to complex system accidents. part 4: IDAC causal model of operator problem-solving response. *Reliability Engineering & System Safety*, 92(8), 1061–1075.
- Dennis, A., Wixom, B. H., & Roth, R. M. (2006). *Systems analysis and design*. Hoboken, NJ: John Wiley & Sons, Inc., 3rd edition.
- El Emam, K. (1999). Benchmarking kappa: Interrater agreement in software process assessments. *Empirical Software Engineering*, 4, 113–133.
- Fidel, R. (1993). Qualitative methods in information-retrieval research. *Library & Information Science Research*, 15(3), 219–247.
- Fitzgerald, B. (1997). The use of systems development methodologies in practice: a field study. *Information Systems Journal*, 7(3), 201–212.
- Flavell, J. H. (1979). Metacognition and cognitive monitoring: A new area of cognitive–developmental inquiry. *American Psychologist*, 34(10), 906–911.
- Giordano, D. (2002). Evolution of interactive graphical representations into a design language: a distributed cognition account. *International Journal of Human-Computer Studies*, 57(4), 317–345.
- Glaser, B. G. & Holton, J. (2004). Remodeling grounded theory. *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research*, 5(2).
- Glaser, B. G. & Strauss, A. L. (1967). *The discovery of grounded theory: Strategies for qualitative research*. Aldine de Gruyter.
- Goel, V. & Pirolli, P. (1989). Motivating the notion of generic design within information-processing theory: The design problem space. *AI magazine*, 10(1), 19.
- Gourgey, A. F. (1998). Metacognition in basic skills instruction. *Instructional Science*, 26(1-2), 81–96.

- Hansen, B. H. & Kautz, K. (2005). Grounded theory applied-studying information systems development methodologies in practice. In *Proceedings of the 38th Annual Hawaii International Conference on System Science - 2005* (pp. 264b–264b).
- Heath, H. & Cowley, S. (2004). Developing a grounded theory approach: a comparison of glaser and strauss. *International Journal of Nursing Studies*, 41(2), 141–150.
- Hindle, K. (2004). Choosing qualitative methods for entrepreneurial cognition research: A canonical development approach. *Entrepreneurship-Theory and Practice*, 28(6), 575–607.
- Hollan, J., Hutchins, E., & Kirsh, D. (2000). Distributed cognition: toward a new foundation for human-computer interaction research. *ACM Trans. Comput.-Hum. Interact.*, 7(2), 174–196.
- Huang, H., Kvasny, L., Joshi, K. D., Trauth, E. M., & Mahar, J. (2009). Synthesizing IT job skills identified in academic studies, practitioner publications and job ads. In *Proceedings of the special interest group on management information system's 47th annual conference on Computer personnel research, SIGMIS CPR '09* (pp. 121–128). New York, NY, USA: ACM.
- Hughes, J. & Jones, S. (2003). Reflections on the use of grounded theory in interpretive information systems research. *Electronic Journal of Information Systems Evaluation*, 6(1).
- Hughes, J. & Parkes, S. (2003). Trends in the use of verbal protocol analysis in software engineering research. *Behaviour & Information Technology*, 22(2), 127 – 140.
- Hutchins, E. (1995). How a cockpit remembers its speeds. *Cognitive Science*, 19(3), 265–288.
- Ives, B., Olson, M. H., & Baroudi, J. J. (1983). The measurement of user information satisfaction. *Communications of the ACM*, 26(10), 785–793.
- Jain, P. (2011). Interaction between software engineering and artificial intelligence-a review. *International Journal on Computer Science and Engineering*, 3, 3774–3779.
- Jazayeri, M. (2004). The education of a software engineer. In *Proceedings of the 19th IEEE international conference on Automated software engineering* (pp. 18–xxvii).
- Jeffrey, H. J. & Putman, A. O. (1994). Relationship definition and management: tools for requirements analysis. *Journal of Systems and Software*, 24(3), 277–294.

- Jonassen, D. H. (2000). Toward a design theory of problem solving. *Educational technology research and development*, 48(4), 63–85.
- Kaplan, B. & Maxwell, J. A. (1994). Qualitative research methods for evaluating computer information systems. In J. G. Anderson, C. E. Aydin, & S. J. Jay (Eds.), *Evaluating health care information systems: Methods and applications* (pp. 45–68). Thousand Oaks, Calif.: Sage Publications, Inc.
- Kim, J. & Lerch, F. J. (1992). Towards a model of cognitive process in logical design: comparing object-oriented and traditional functional decomposition software methodologies. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 489–498).
- Kitchener, K. S. (1983). Cognition, metacognition, and epistemic cognition: A three-level model of cognitive processing. *Human development*, 26, 222–232.
- Klein, H. K. & Myers, M. D. (1999). A set of principles for conducting and evaluating interpretive field studies in information systems. *MIS Quarterly*, 23(1), 67–93.
- Klein, H. K. & Myers, M. D. (2001). A classification scheme for interpretive research in information systems. In E. M. Trauth (Ed.), *Qualitative research in IS: Issues and trends* (pp. 218–239). Hershey: Idea Group Publishing.
- Kluwe, R. (1995). Single case studies and models of complex problem solving. In P. Frensch & J. Funke (Eds.), *Complex problem solving: The European perspective* (pp. 269–291). Hillsdale: Lawrence Erlbaum Associates.
- Lee, A. S. & Hubona, G. S. (2009). A scientific basis for rigor and relevance in information systems research. *MIS Quarterly*, 33(2), 237–262.
- Lim, B. C. & Klein, K. J. (2006). Team mental models and team performance: A field study of the effects of team mental model similarity and accuracy. *Journal of Organizational Behavior*, 27(4), 403.
- Lopez, T., Petre, M., & Nuseibeh, B. (2012). Thrashing, tolerating and compromising in software development. peer-reviewed.
- Malhotra, A., Thomas, J. C., Carroll, J. M., & Miller, L. A. (1980). Cognitive processes in design. *International Journal of Man-Machine Studies*, 12(2), 119–140.

- Mercado, C. A. A. & Romo, L. I. S. (2011). Mental models, strategies and metacognition in programming, and it's relation to verbal protocols as a learning mechanism³. *Analysis of State-of-the-Art Solutions for Personalised Learning Support*, (pp.32).
- Miles, M. B. & Huberman, A. M. (1984). *Qualitative data analysis: A sourcebook of new methods*. Beverly Hills: Sage.
- Myers, M. D. (1997). Qualitative research in information systems. *MIS Quarterly*, 21(2), 241–242.
- Myers, M. D. & Newman, M. (2007). The qualitative interview in IS research: Examining the craft. *Information and Organization*, 17(1), 2–26.
- Newell, A. & Simon, H. A. (1972). *Human Problem Solving*. Englewood, Cliffs, NJ: Prentice-Hall.
- Ohlsson, S. (2012). The problems with problem solving: Reflections on the rise, current status, and possible future of a cognitive research paradigm. *The Journal of Problem Solving*, 5(1), 101–128.
- Orlikowski, W. J. & Baroudi, J. J. (1991). Studying information technology in organizations: Research approaches and assumptions. *Information Systems Research*, 2(1), 1–28.
- Parnas, D. L. (1994). The professional responsibilities of software engineers. In *Proceedings of IFIP World Congress*, volume 2 (pp. 332–339).
- Parnas, D. L. (1998). Software engineering programmes are not computer science programmes. *Annals of Software Engineering*, 6(1), 19–37.
- Rogers, Y. (1997). A brief introduction to distributed cognition. <http://parvac.washington.edu/courses/inde599/dcog-brief-intro.pdf>.
- Romney, M. B. & Steinbart, P. J. (2012). *Accounting Information Systems*. Essex, UK: Prentice Hall, 12th edition.
- Shih, P. C., Venolia, G., & Olson, G. M. (2011). Brainstorming under constraints: why software developers brainstorm in groups. In *Proceedings of the 25th BCS Conference on Human-Computer Interaction*, BCS-HCI '11 (pp. 74–83). Swinton, UK, UK: British Computer Society.

- Simon, D. P. (1978). Information processing theory of human problem solving. In D. Estes (Ed.), *Handbook of Learning and Cognitive Process*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Singer, J., Lethbridge, T., Vinson, N., & Anquetil, N. (2010). An examination of software engineering work practices. In *CASCON First Decade High Impact Papers, CASCON '10* (pp. 174–188). Riverton, NJ, USA: IBM Corp.
- Star, S. L. (1989). The structure of ill-structured solutions: Boundary objects and heterogeneous distributed problem solving. *Distributed Artificial Intelligence*, 2, 37–54.
- Star, S. L. (1992). The trojan door: organizations, work, and the “open black box”. *Systemic Practice and Action Research*, 5(4), 395–410.
- Strauss, A. & Corbin, J. (1994). Grounded theory methodology. *Handbook of Qualitative Research*, (pp. 273–285).
- Strauss, A. & Corbin, J. (1998). *Basics of qualitative research: Techniques and procedures for developing grounded theory*. Thousand Oaks, CA: Sage, second edition.
- Thornberg, R. (2012). Informed grounded theory. *Scandinavian Journal of Educational Research*, 56(3), 243–259.
- Vessey, I. & Rosemann, M. (2008). Toward improving the relevance of information systems research to practice: the role of applicability checks. *MIS Quarterly*, 32(1), 1–22–1–22.
- Walsham, G. (1993). *Interpreting information systems in organizations*. John Wiley & Sons, Inc.
- Wang, Y. & Chiew, V. (2010). On the cognitive process of human problem solving. *Cognitive Systems Research*, 11(1), 81–92.
- Weiner, J. M. (2011). Is there a difference between critical thinking and information literacy? *Journal of Information Literacy*, 5(2), 81–92.
- Wiley, J. & Jarosz, A. F. (2012). Working memory capacity, attentional focus, and problem solving. *Current Directions in Psychological Science*, 21(4), 258–262.
- Williams, L., Kessler, R. R., Cunningham, W., & Jeffries, R. (2000). Strengthening the case for pair programming. *IEEE Software*, 17(4), 19–25.

- Wood, P. K. (1983). Inquiring systems and problem structure: Implications for cognitive development. *Human Development*, 26, 249–265.
- Yang, S. C. (1997). Information seeking as problem-solving using a qualitative approach to uncover the novice learners' information-seeking processes in a perseus hypertext system. *Library & Information Science Research*, 19(1), 71–94.
- Zannier, C., Chiasson, M., & Maurer, F. (2007). A model of design decision making based on empirical results of interviews with software designers. *Information and Software Technology*, 49(6), 637–653.
- Zhang, J. (1997). The nature of external representations in problem solving. *Cognitive Science*, 21(2), 179–217.
- Zhang, J. & Norman, D. A. (1994). Representations in distributed cognitive tasks. *Cognitive Science*, 18(1), 87–122.
- Zhang, J. & Patel, V. (2008). Distributed cognition, representation and affordance. In *Cognition Distributed: How Cognitive Technology Extends Our Minds*. Amsterdam: John Benjamins Publishing Company.
- Zhao, L., Collopy, F., Grant, J., & Boland, R. (2008). An empirical comparison of accounting information representations. In *Proceedings of the Third Midwest United States Association for Information Systems, May 23-24, 2008 Eau Claire, WI*.

Appendix A

LIST OF DATA COLLECTION EVENTS AND PROTOCOL QUESTIONS

Event No/ Question No	Data Collection Events/ Protocol Questions
E1	Interviews with individual software developers
1	Do the students know which domain the application would be under?
2	How much do the students know about the domain of the required application?
3	What do the students think would be required in the application?
4	Do the students refer to texts in the domain of the application?
5	Do the students have a perceived model of how to design software?
6	What level of competence do the students have in software design?
7	Do the students know how the CASE tools may help them in software design?
E2	Interviews with teams of software developers
1	Does the team's perceived application requirement change?
2	What changed the perceived requirement?
3	Do the CASE tools guide the teams on how to design software?
4	What did the teams ask during their user-developer meeting?
5	Do the teams have a perceived model of how to design software?
6	Does the team's perceived model change?
7	What brought about the change in their internal model?
8	Do the teams consider time as a factor in their planning?
9	Do the teams refer to texts on software engineering?
10	Do the teams compare several alternative solutions concurrently?
E3	Interviews with users, including the use of user satisfaction questionnaire

Event No/ Question No	Data Collection Events/ Protocol Questions
1	What are the types of tasks that the user wishes the application to aid?
2	What do the tasks try to achieve?
3	Do the tasks have time constraints that must be met?
4	What are the elements in the application interface?
5	Are the elements clear and easy to understand?
6	Do the elements address the tasks of the user?
7	Does the user have the appropriate knowledge to perform his task?
8	Does the user know how to interpret the elements in the application interface?
9	How do the elements aid the user in his tasks?
10	How would the lack of the elements affect his tasks?
E4	Observations of software developer team discussion meetings
1	Do the teams have a perceived model of how to design a software?
2	Do the teams collect any material/document from the user stating their requirement?
3	Which info in the references do the teams use for understanding user requirement?
4	Does the teams' perceived model change?
5	What brought about the change in their internal model?
6	Do the teams refer to texts on software engineering?
7	Do the teams compare several alternatives solutions concurrently?
E5	Observation of discussion meetings between software developer teams and users
1	Do the elements address the tasks of the user?
2	Does the user have the appropriate knowledge to perform his task?
3	How do the elements aid the user in his tasks?
4	What did the students ask during their user-developer meeting?
5	Do the students collect any material/document from the user stating his/her requirement?
6	What brought about a change to their internal model?
7	Does the user know how to interpret the elements in the application interface?

Event No/ Question No	Data Collection Events/ Protocol Questions
8	How would the lack of the elements affect his tasks?
9	What are the elements in the application interface?
10	Are the elements clear and easy to understand?
11	How does the user model the steps to execute his task?
E6	Analysis of the applications, from prototypes, conceptual modelling diagrams and other design documents.
1	What are the elements in the application interface?
2	Are the elements clear and easy to understand?
3	Do the elements address the tasks of the user?
4	How does the user model the steps to execute his task?
5	How do the elements aid the user in his tasks?

Appendix B

TRANSCRIPT OF SECOND INTERVIEW WITH BILL

(I)nterviewer This is about the time-sheet system that you did for [PQR]. What does [PQR] stands for anyway?

(S)ubject [hidden for anonymity], but here they're called [hidden] .

I So in this system, first of all the users puts in their timesheets into the web?

S Into Outlook, Microsoft Office.

I And then the managers. You said there are two persons who will be approving. You also said that there was some invoicing. How does invoicing relate to the timesheets of employees?

S Ok. So the timesheets are submitted by all the staff. So every person in the organisation submits the timesheets for the previous week. And they have to hand them in by a certain time on Monday. The two approvals are, one is the project approval. Which is, "is this person allowed", the project manager saying, "is the staff member allowed to put time on the project?". The other manager is the employees' direct manager, the person they report too. And they are approving whether the person has met their hours for the week. And also internal things, like leave or um, other administrative expenses that are internal. Once that, once both of those approvals are complete, that time gets submitted to the financial system, Dynamics AX. And then it flows through the normal journaling process of getting turned into um, like time journals that make up a part of the financial system. And um, those roll into the WIP process and what's to be built stuff that which you charged the customer for that hasn't yet been recovered. And so that, at the end of the month they have this invoice planning process which is looking at all the journals that haven't been, um, invoiced. And put them together into bills. And so they take like, there might be 50 different people who've worked on a certain project, but you can group all those individual time records together. And get a summary amount and then write some narration about what the bills are for. Having written detail on what is done. And you know, also summarises up any expenses that goes into the same time-sheet, time and expenses come together. As well as also to um, vendor transaction like if [PQR] have paid for, maybe like a dig-a-hire or the hire of conference rooms, or bookings or anythings like that, that can also roll on as a, a chargeable expense, rechargeable expense based on the invoice. So they made the invoices outside of AX to start with, just to be able to create them, and then they again get submitted back to AX as proper invoices in the system. So our part of it is to plan the invoices, so that's what it's going to be. And they can kind of go into preview and see what it might look like on the invoice once it's printed out. And then they submit it to the general or admin staff.

- I So the employees work on certain projects, and then the amount of time that they spent is used to calculate the cost that will be charged to the clients?
- S Yes. So each person has a different charge rate, and that can be different based on the project they're on, based on, they normally have base rate. And if they are working in a different city for instance, they might charge out more because there's extra cost involved. Or for a specific job they might have given at a discounted rate. And so that gets taken into account. And yet the financial systems looks after that all. All of that get submitted into our system, it went to the part that is concerned with hours, like the number of hours, that 1 hour or 1.2 hours or something like that you know. And then the system takes the charge rate and turns it into dollar amounts.
- I And then puts them in the planned invoice?
- S Well, they don't come through, they just go into the journals in the meantime. So they sit in there as a, if you look at them as a whole, you'd see them as unbilled time, as WIP, work-in-progress. And while that's unbilled, it just remains there in a rather large bucket, and they split that work out amongst the different people who are responsible for the projects. So if 10 different people have built that project, it all funnels back to that 1 project manager for him to invoice. So it's taken all that distributed time-sheet information, consolidate together, and then gets summarised by project. And break them into invoices or bill them all together as one. Or time them all till next month in case there's something that might not be ready to be billed yet, they can hold it there, for another month. It's all potential money so that's why they have the obvious interest in getting it through.
- I In any of these 3 tasks, I remembered that you were involved in design of the concept that you guys were selling to the customer. And then afterwards you were involved in the database design, although you have assigned 1 particular person to do the database changes. And then you were assigned with building the website for the time-sheet approval. And these 2 were assigned to other people. In any of these 3, how did other team members' thoughts influence your thoughts?
- S Um, there were some constant things that we weren't going to, that we kind of already decided. So we'd decided on using Outlook. Because that's something we've already had used. So that was fairly fixed. The person doing the Outlook had to get the data from the web service side of it. And although it didn't have to be a web service, I guess it could have been other methods, like you know, a connect to a database, or other thing. But it was decided as a web service. And the web service influenced I guess the design of how the Outlook part worked. Because once those 2 bits were known, the protocol had to be established between the 2 sides. So it doesn't really affect the thing that I was doing. Can you say the question again?
- I How or to what extent did you team members' thoughts influenced your mental model, when you were solving problems here, or when you were involved in design or database? You may have a certain model of how to solve the problem, but you have other team members who have their own thoughts, somehow that influenced your original model, which made to you change that model. Was there anything like that?
- S One aspect of that was, we had quite a lot of discussion around whether a time-sheet would be, for a week, would be an entire unit, or would it be just a number of time-sheet lines. So

if you worked for the same client 5 times a week, that's 5 individual time-sheet entries. But overall for the week that might all sum up to 40 hours that you were trying to make. We had a discussion on whether or not there should be an over-arching kind of header record. Which would say that this is the record for that week. And particularly the problem was how do you approve in 2 different ways, cause if there was a header, the time-sheet as a whole would need to be approved as well as the individual ones that need to be approved. So where are those in the database, how we kind of fit the 2 approval model into, either having a header or not. And also, some of the similar problems were sometimes timesheets not submitted completely, while someone, or maybe there was a network error or something while they were submitting the time. And how do you undo, or how do you cope with the different situations where someone has already submitted some of their time, Outlook thinks that it's submitted but the other side hasn't received at all. And again that comes back to kind of a statuses on, yeah having the time-sheet status to cannot be changed when it's completed. There were more on particular fields that we needed or not. But I cannot remember any particular design decisions. We worked pretty stand-alone in a lot of ways because um, our different parts, except for the Outlook and the web service, they were integrated with each other. Web service, once the time has been posted, yes it had to go on an agreed format. But that formal had been kind of discussed up front at the database, we knew what the database is going to be. I tell you about another thing here. There was another guy at a time who, I had certain names my tables and fields in the database, and he came through and decided to rename them. Because although the ones I had were ok, they didn't follow the particular standard that the Dynamics platform uses, and he was a Dynamics programmer with lots of experience. And he said "even though your table names are fine, they'll work, there's a certain standard you should adopt in naming them". Because for instance the main tables, the kind of key table, all end with the word "table", for whatever reason. And um, so that, probably a quarter way through the development we already had these tables, and we actually worked through and change them because he came on a bit later, but because of his experience, "this is, if you do it this way, it's going to be easier for upgrades and things in the future".

I But that's not really on the design part.

S Not so much on the design, no. it's more on conventions I guess. The design, probably didn't change too much from what we'd done on the concept. It was more that we had to do some extra thing that we hadn't considered in the concept. Like, the wanted a sorting on certain of the fields. The ability to go to and select lines individually and then do operations on that. So those things hadn't been on the concept but they were suggestions that had come back from the client.

I What software tools did you use to create the website for the timesheets?

S Um the website, that was ASP.Net and Visual Basic .net too. And we use Ajax extensions for that. And yes we use the Microsoft Enterprise Practices Library, which is a standard set of, or for us we use the database access part of it. To get a common way of calling procedures and things like that. We use the Microsoft SQL Services Server. So that we can integrate reports straight into the Word application and do PDF exports and things like that.

I Were you familiar with those tools before this? Or did you need to refer to some manuals or the Internet to get used to them?

S Ajax I've used on a couple of tutorials. But that was the first site that we were using it on. I knew the Ajax concepts but not how to use the Microsoft in particular. So that was new. Enterprise Practices we've used before. That was something we've picked up from the previous job, seeing someone else doing it and realised it saved a lot of time and made things easier to code. We used the code generation module that we written previously as well to do some of the common border plates, classes, and things that we're aware that was going to match the database models and things like that. Yeah, but for the most part it was things that we were familiar with.

I Were any of those tools dictated how you attempt to solve the thing that you do. Dictated as well as limited the way you do things?

S Absolutely yeah. The, the solution we used has got quite a lot of grids, like tables, with headers on the top and then sortable columns of data, which can be selected and unselected like, like I said you can do things. And ASP.Net is very low at support for, you can edit one row at a time but you cant edit lots of rows all at the same time. Whereas in html that is possible, but the .net framework does not allow you to do it. And so because of that, we had to find other ways of editing lines, and actually try to avoid editing lines as much as possible because um , it was already limited. It actually improved the design in some ways because orginially we had it when you have an invoice planned for instance and you read through and pick a few lines and you wanted to write off the time on those, you know not charge it, you're going to have to do it on each individual line. But we find that if you actually set a total, and you say "we've got \$1500 worth of time there, but we're only going to charge 1000", and it caluculates there's 500 left. It's easier for the computer to go through and apply that 500 across the line rather than go the other way where you had to, "oh no I'm not going to charge that, I am going to charge that". It didn't really matter to them which lines are written off or not, they weren't interested in that aspect. Whereas previous versions it had been important about who's time is written off, this particular line is not charged because such and such. But for them, they wanted to know that on the whole for that project, \$500 wasn't charged for some reason. So that was 1 thing that the framework influenced. Also .net although is quite flexible in certain ways, it can be quite complicated to create your own controls. If you want to create, like this grid for instance that we created, it took probably a good 30% of the time to just making a grid that could be used across all the different pages. And that did sorting automatically that could be selected, you know, like click the lines on and off to be selected. Because we didn't do it standard, anything outside that standard is hard to achieve. It's like you'd actually require a bit of getting underneath like, taking one of the controls and extending it, to do the things that are required.

I In general which of the tasks you think most complex, most difficult?

S Um, probably actually the website development for me. The database, was a few of us working on it, so that was kind of a collaborative effort. The designs, that's fairly easy because you're just, you're just saying what it might be like. You're talking about things that you can just draw up on the screen and you can describe it. You haven't actually figured out how you're doing that. When actually doing, to say now that we've got this plan of what we're going to do, how exactly we're going to achieve that, was certainly one of the most difficult. Like I've said making controls to fit into the framework that was chosen, after realising that we've already started with the end, choosing it as a framework,

the .Net. Its limitations became apparent when we're doing things that we thought would be straight forward but they actually weren't. So trying to bend it to do the things that we wanted to do was part of the problem.

I I guess this is a very general question. What do you do to form a mental model of the system? What is your mental process of approaching the problem?

S For me, although it's mental, I've got to get it out of my head. If I can't see it personally, if I can't see it in front of me, I find it very hard to understand. And I find that I sometimes get caught on a problem and the only reason I'm not moving forward with it is because it's not being laid out in front of me. So drawing pictures and drawing diagrams, and um, yah mainly visual. Visual aid is how, is how I try to, if I can see it happening, then I think I can think of something that will achieve it. But how I actually get to achieve it I don't know. It just comes from, from the visualising I think.

I So you visualise the interface or the end solution, and then you work through how to go there?

S Yeah it's like a goal. Like once you've got a goal then the brain seems to be quite good in filling in that gap. You know like if you. It is if you don't know that goal, that you'd have most trouble. But I couldn't really describe what the process was. The practical process often is actually working small bits out. So you know, I'm a classic kind of, do a change and then refresh it to see what it looks like. If I can make a small part, and I can go back to the browser and have a look. Does it look right? Is it doing what it's describing? And then you go back and make more changes. It's actually quite back and forth between the code and the output to see that it looks right.

I So in terms of doing the website, before you start doing it you would have some of the concept design from here, and then you bring it on here.

S And then tidy it up and get rid of all the copying and pasting and trying to standardise it. Because often for this one you're rushing it. You're not making a lot of effort to make sure that it's clean and tidy. It's just, it's just getting it to look right. You know. It might be a screenshot, it might be a small visual interactive demo but you're getting any application working right underneath the surface. Although we very often borrow from the design into the actual solution, you know, maybe some of the same screens or something like that, there's a process of going through and cleaning it so that it's not quite so, ugly underneath.

I How did you break down the things into small pieces of problems when you were doing the website? Or how did you structure the way you approach?

S Probably again starting from the users actions. Like you know starting from, this is what it's supposed to be able to do. Again, you've got your goal. And then ok then what, I'd probably think of it in 3 layers. What's the database going to be to provide, or store in order to do that operation? What's the user interface going to show? And what kind of feedback the users getting when something is happening? And then the middle bit is the web service I guess. How do you tie storage and data manipulation back to the user interface? So it's, so those two edges know the storage system side and the users edge. And then in between it's just, it usually kind of do some magic tagging back and forth. And maybe changing something on the way through, or checking rules or that kind of thing.

- I Did you have some sort of strategy to select the most representable model? I've given example to some people that one approach is probably to have different sets of solutions, comparing against each other, looking into strength and weaknesses of each one, and finally picking the right one. Or some people might just go and do just one, and then tweaking it to get the best result?
- S Yeah more of like the second description there. It certainly wasn't lots of different options considered. And the constraints that we had, time of getting it done. We just didn't look at many options. So the solution in the end would have been a lot like what you've described in terms of making something, seeing its faults or strengths, and adding to them. So that seems to be how it works. I So there must have been lots of iterations since you've started doing it until it finishes.
- S Yeah. Like there's lots of um, versions that you're going through. Um and I think also going back to the people who were responsible for the project at various stages and say "is this starting to look like what you were expecting". And getting early feedback on whether or not they think it's matching the requirements spec.
- I So at the design stage until you get sign-off from the clients, do you have iterations going on here? And from which party did you get the input to make those iterations?
- S Yeah, there would be variations there. Often it's not so much about having the perfect design upfront though. At that stage, in terms of the selling anyway, it can be about creating an emotion in the customer's mind, "does that feel like the system that you're looking for? Not necessarily does it have every single feature that you're desiring". But does it, eliciting the mood or feeling that, "oh yes that looked like the system that could do the job I have at hand". And you're often not concentrating on all the big details, the you know, what format it's going to be or what screen resolution and all that. It's creating a vision of what might be. At least in the selling design, but much into the detail design. There's certainly a lot more going back to the project manager, and sometimes directly to the customer. And meeting the end user and saying, "is this how you'd expect to use it?". And you get feedback in. And you actually got to throw away a lot of, I mean you've got to throw away a lot of the things that you do because they just don't, they're not exactly right. Sometimes it's just a tweak. Sometimes it's a throw away.
- I And so that happens around here?
- S It happens here. And if you've got a more detailed design process in the middle, there as well.
- I So when you were designing with you other teammates, you were designing the database, you had to get more input?
- S Yeah. I mean we didn't kind of design on the day one and then it was alright. As we started building, we realised that something's missing, and we kind of come back together and say "well how are we going to do this extra thing that we have". Because to try to do everything up front without having anything that exists yet, is quite hard to visualise everything in your mind. And so, we'll do enough to get started. So that everyone's clear of the general gist of how it's going to work. But specifics often come out when you're here. And you realise, "oh wait, we've put a status field in but, we forgot about the status when this happens". And so we go back and, "we'd have an extra status here", and everyone

would need to understand what's that going to do and how's that going to affect their part of it.

I So it's like the database design is the underlying backbone of the thing, but you couldn't get the right backbone unless you have something that you show. Only when you show the interface you know exactly what needs to be done in the database.

S Even here you might have you might have pictures of the user interface. At the design phase you might have diagrams or examples of what it's supposed to look like. But sometimes it's not until you're building, you're actually in the midst of building it that you've realised that something's missing. And because it may not be something that is visual, maybe it's something that's underneath. And it's a "status" thing that is required, but it doesn't actually have a direct visual indicator. It's not like a field on the screen. It might be required to support some programming processes underneath that the user never interacts with, but the program needs it for itself to maintain state or something like that. So those are the kind of things that you miss at this stage. You get the obvious things, but you miss some of those, supporting attributes and whatever else.

I You made some testing at your side, and some testing at the client's side as well. So when did the production system started?

S October, November, somewhere 2007, I think. The year that we moved in here, so it's 2008 now, a year ago that we moved in here. And just as that as we were moving, it was starting to um, go into use I think, from early this year, but it was on that 2007- 2008 border somewhere.

I At the end of 2007. So, what sort of feedback do you get from the users after the system went live?

S We were under a constraint to go live on a certain date. And so regardless of whether the system was fully functional, we were going live on a certain date because somebody at the business had promised that that's when it'd happen. And so it went with some problems, and um that first week we heard back, especially with the, people who enter their timesheets right into Outlook, but some of the posting process back to the web services wasn't going in all regions. Because they had quite a dispersed national spread of where their offices was. Like from Christchurch right up to Auckland. And they even had people in Vietnam and Australia posting into the same system. So it's quite different experiences with different offices. And we started to get a lot of feedback about the posting process. How long it took, or people not seeing what they expected. When it came to entering into the timesheets, a certain expense code was missing. Or it wasn't showing as "reimbursable" where it should have been. Some of it was configuration in the AX system itself, which a lot of the data they were using to provide in the Outlook thing, like the expense codes. And some of it was just software bugs that just weren't ironed out before the time that we were working to. So we've got quite direct feedback, our helpdesk here received a lot of calls that came back through, further down to who was responsible for the area. And they'd be doing the change.

I Any of the changes were substantial?

S No, not substantial. They were usually tweaks, or turns it off an hour and we'll work on it and come back to it kind of thing. Things that weren't essential. Like they had problems with attachments to invoice plans which were supposed to be Excel only but. They attached alright but they didn't print out alright at the other end, and AX. And so that was just, they just had, for a couple of months that they'd just turn it off and then not use it. And they'll get by without it and find another way to do it. And so now they're coming back a year later or so. And we're more revisiting that again. Now they want that feature back. And we'll look at a different way of doing it because it didn't work the way that we'd described.

I And that was solved?

S Well it's not solved yet. It's just that turning it off solved it for them because they didn't actually, it was not really a requirement. It was a nice to have, which they tried to slip in. and no, we tried to slip in. by having it in but not having it working fully, it didn't work at all. You know it need to work consistently if it were to work at all.

I So for now it's been quiet?

S Since then? What turned up since then? I think another thing we found was there was certain administrative function that we need to build into the system that's too expensive to build. And so they don't get built, until they first happened. So for instance there was, from each conditions, like somebody posting their time-sheet and then needing to unpost it because of either they miss something. Or someone creating an invoice and needing to undo that action. So we didn't build that "undo"s in there, but it's possible to go into the database and make a change that does the equivalent action.

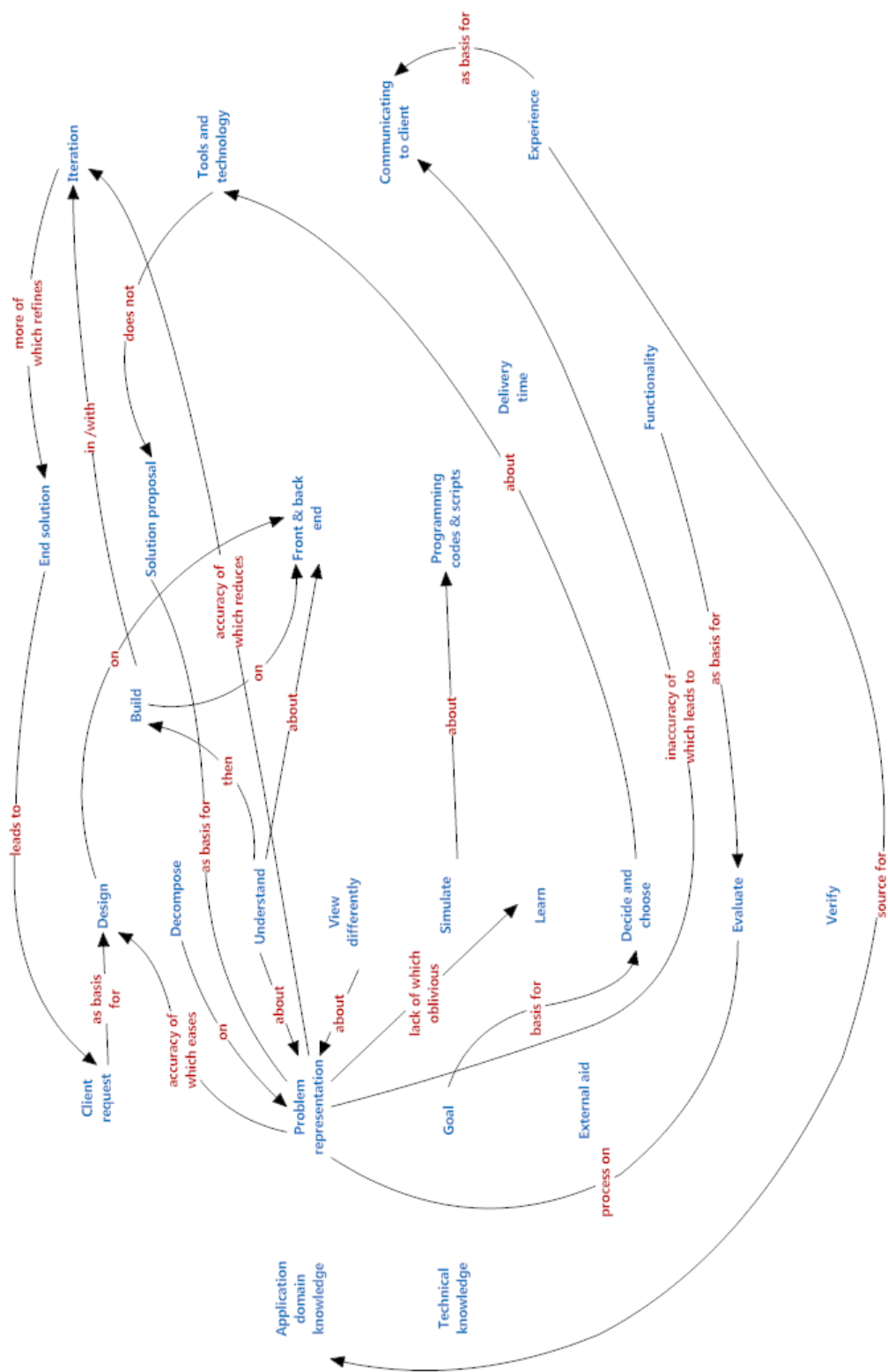
I But they couldn't do that.

S They couldn't do it. And then even though it was on their system but they didn't know where to go and do that or how. So we'd have to get directly involved. And there's a cost difference between, you know, maybe a 5-minutes, a 10-minutes support just to do that. Or maybe a day or two's work to make it a once and for all change that they can use later on. And what we found was that the ones that keep turning up, are the ones that eventually got turned into a feature. So we've put an "unpost", or "unpost time-sheet" or something function in. Since then it wasn't in the original, because it turned up enough times to make it worthwhile that someone to go and give maintenance service on them. So there was regular meetings with the client that the project managers have had, and change people overtime. And they would come and discuss the things that kind of frequently turned up. And it might not be actually important for the client to want it to be changed into a feature or not, it might only effect the one person who was getting all the, "oh can you change this" complaint. But like I said because of its frequency, it highlighted itself as something that need to be permanently fixed, not just lots of temporary fixes.

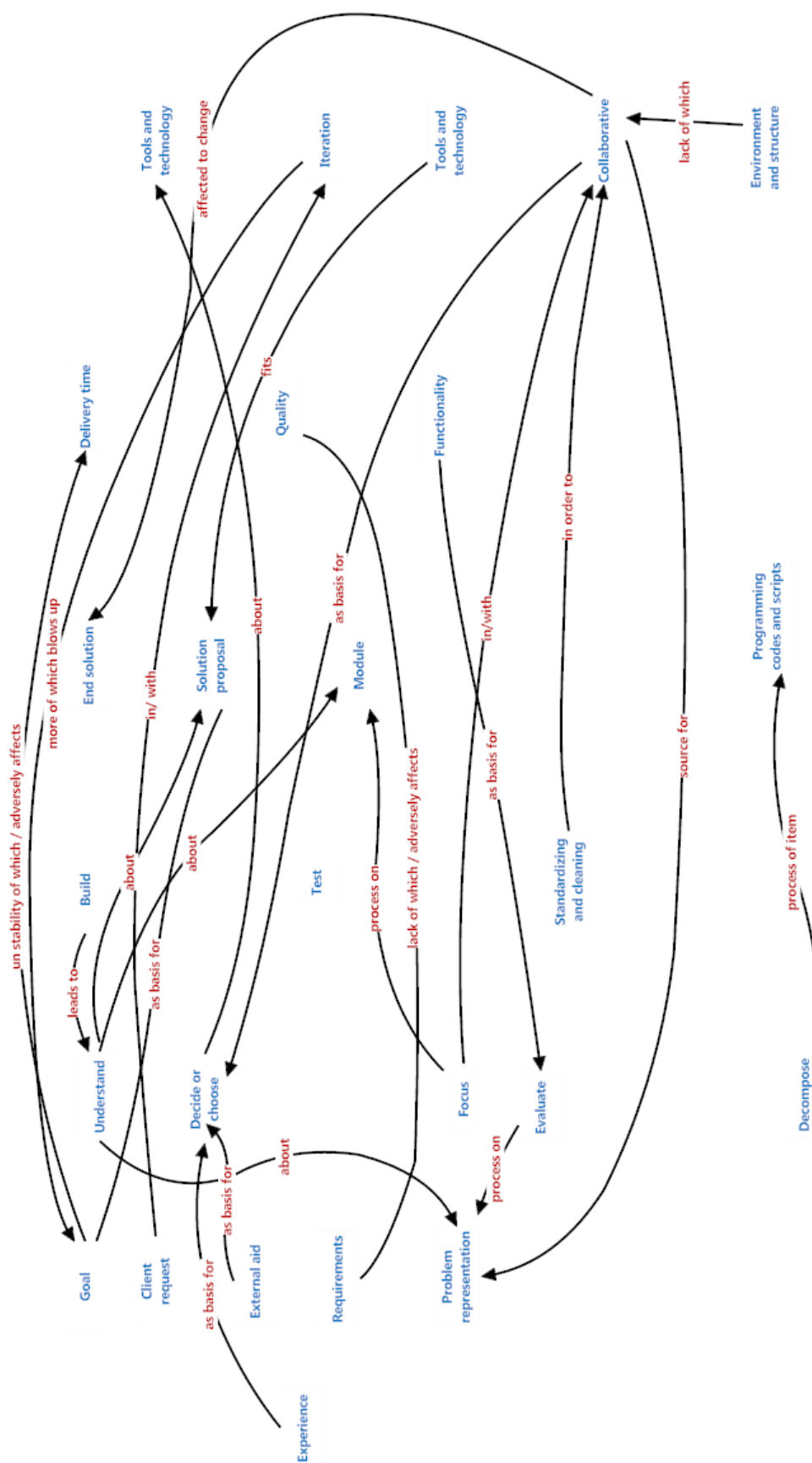
Appendix C

MODELS OF SUBJECTS' PROBLEM SOLVING

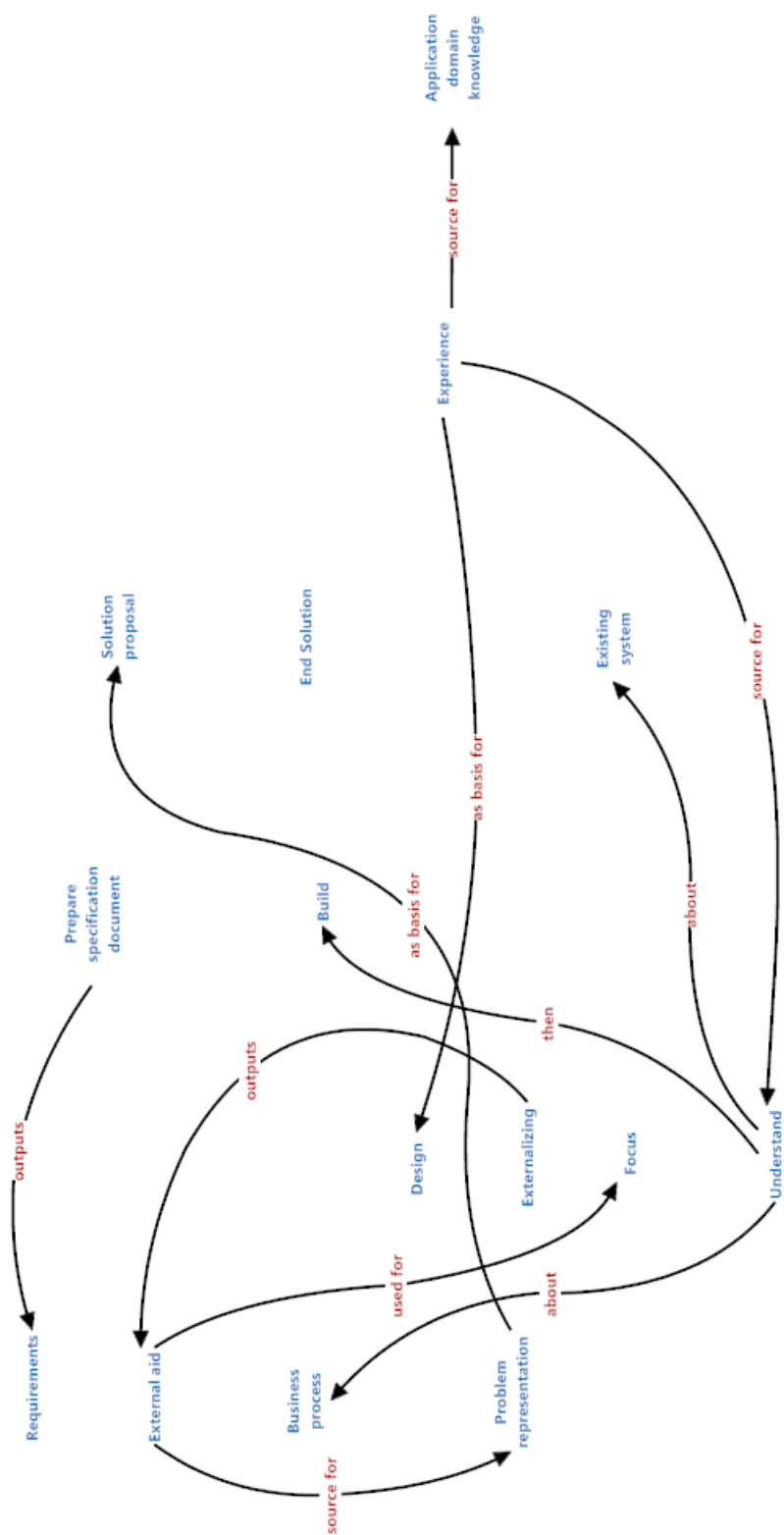
C.1 Barry



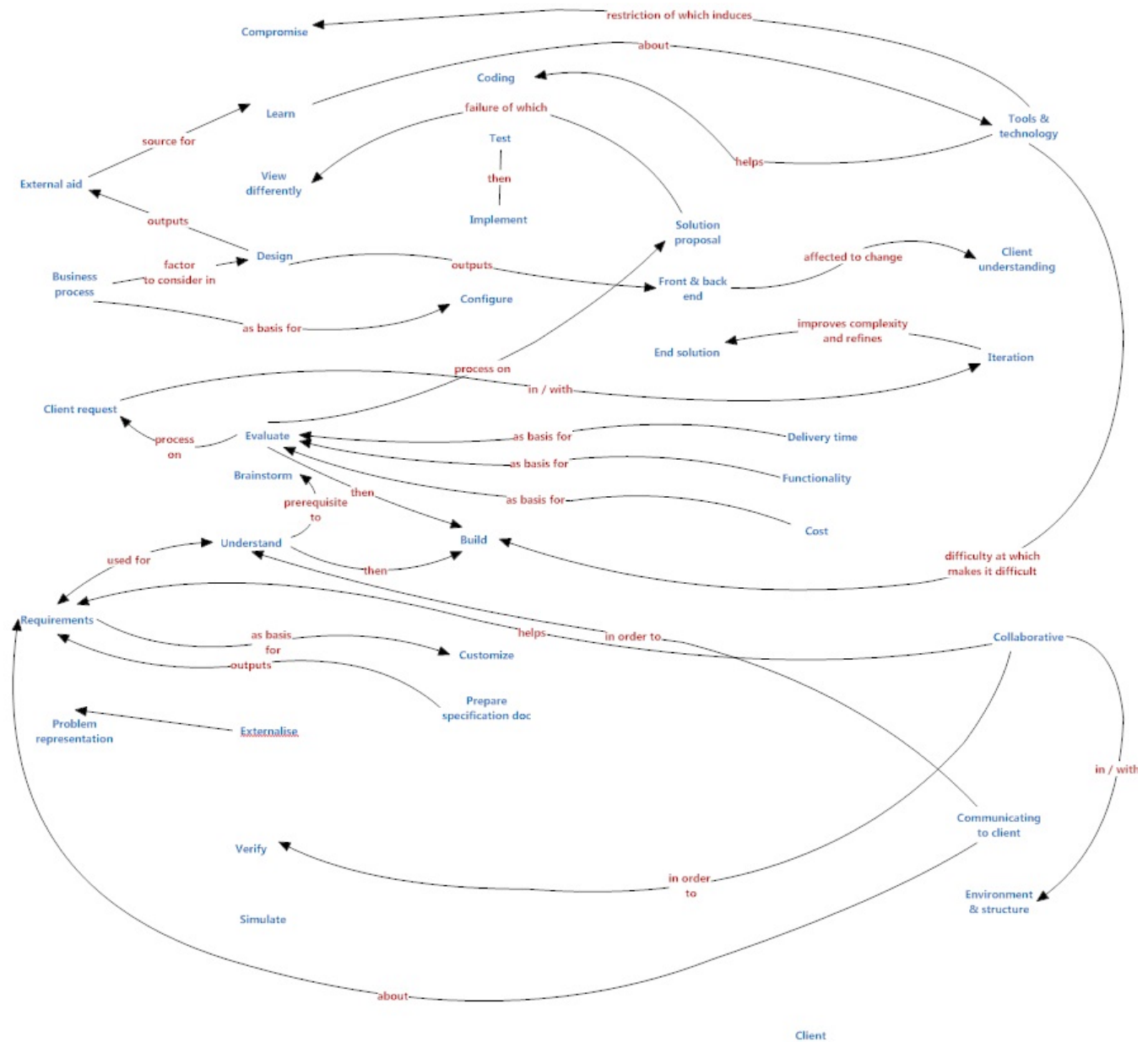
C.2 Jack



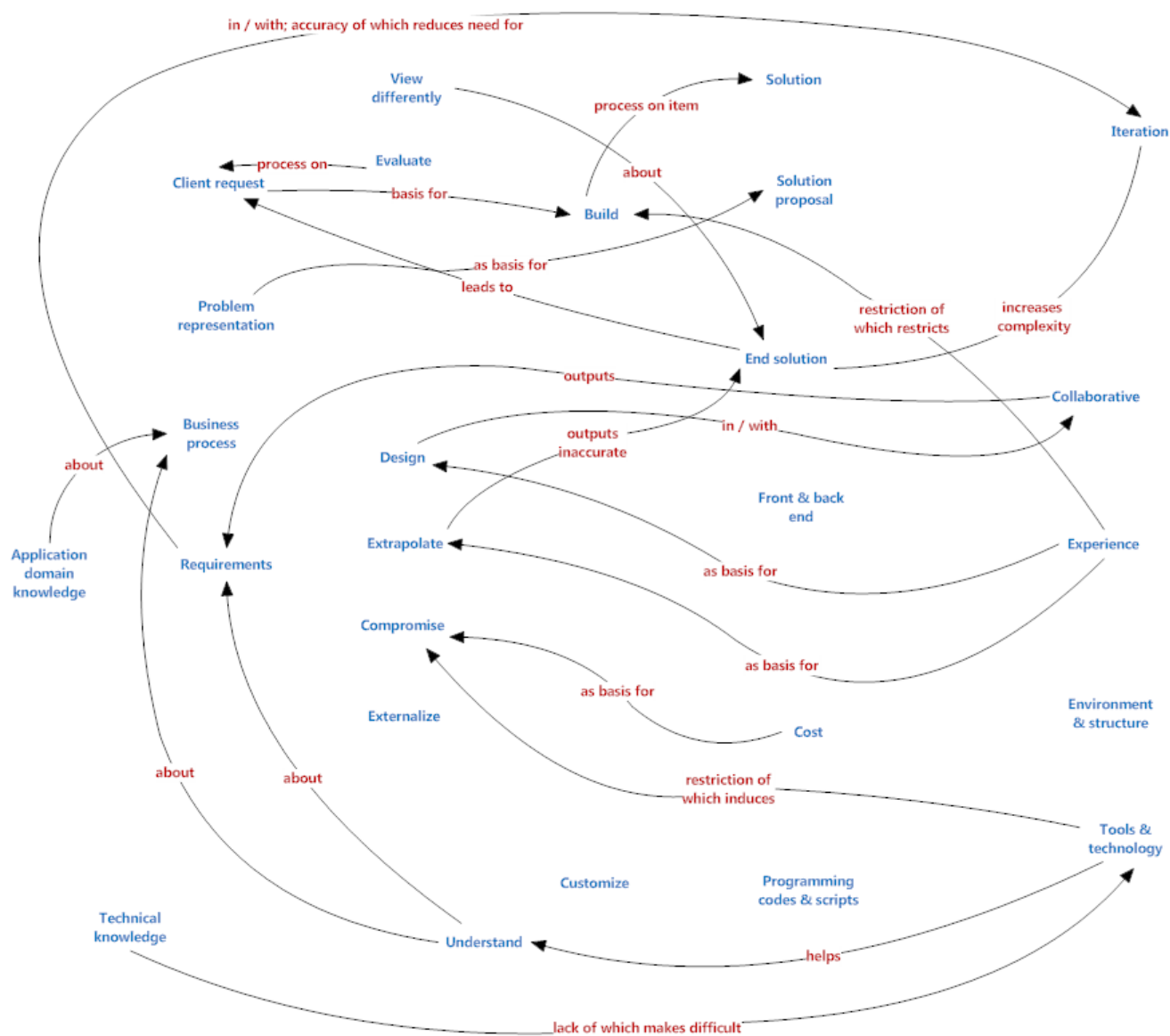
C.3 Jasmine



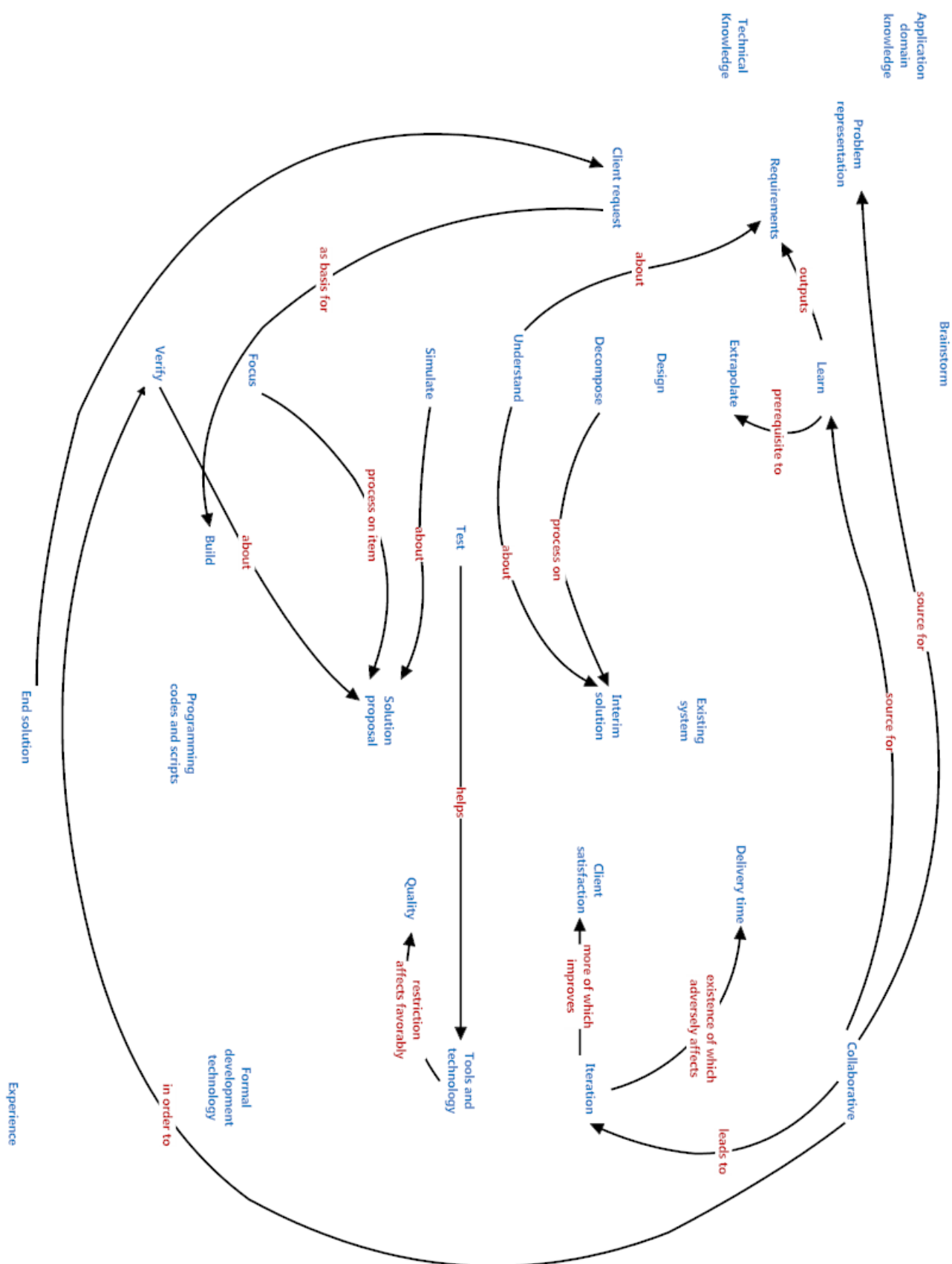
C.4 Jessica



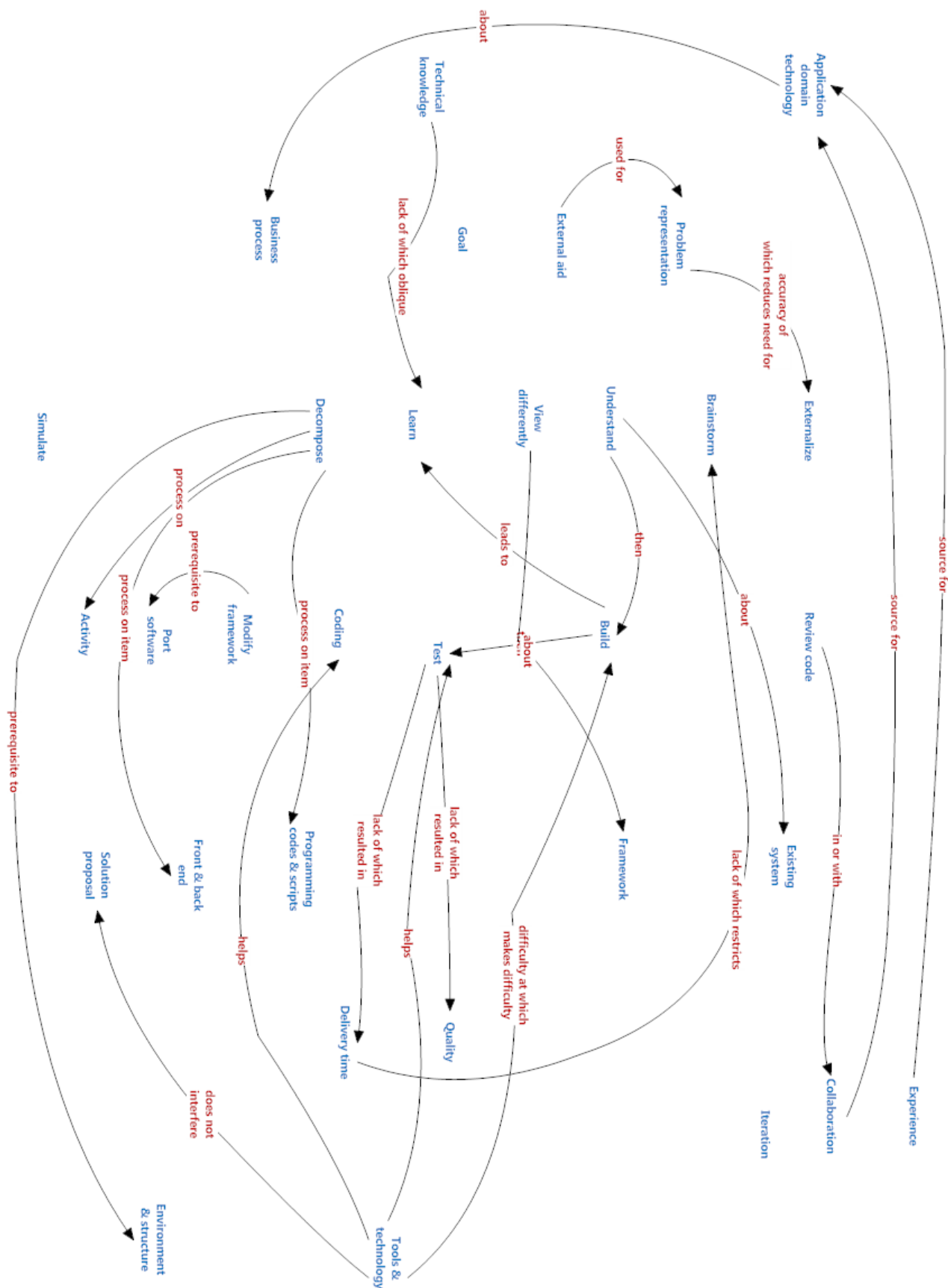
C.5 John



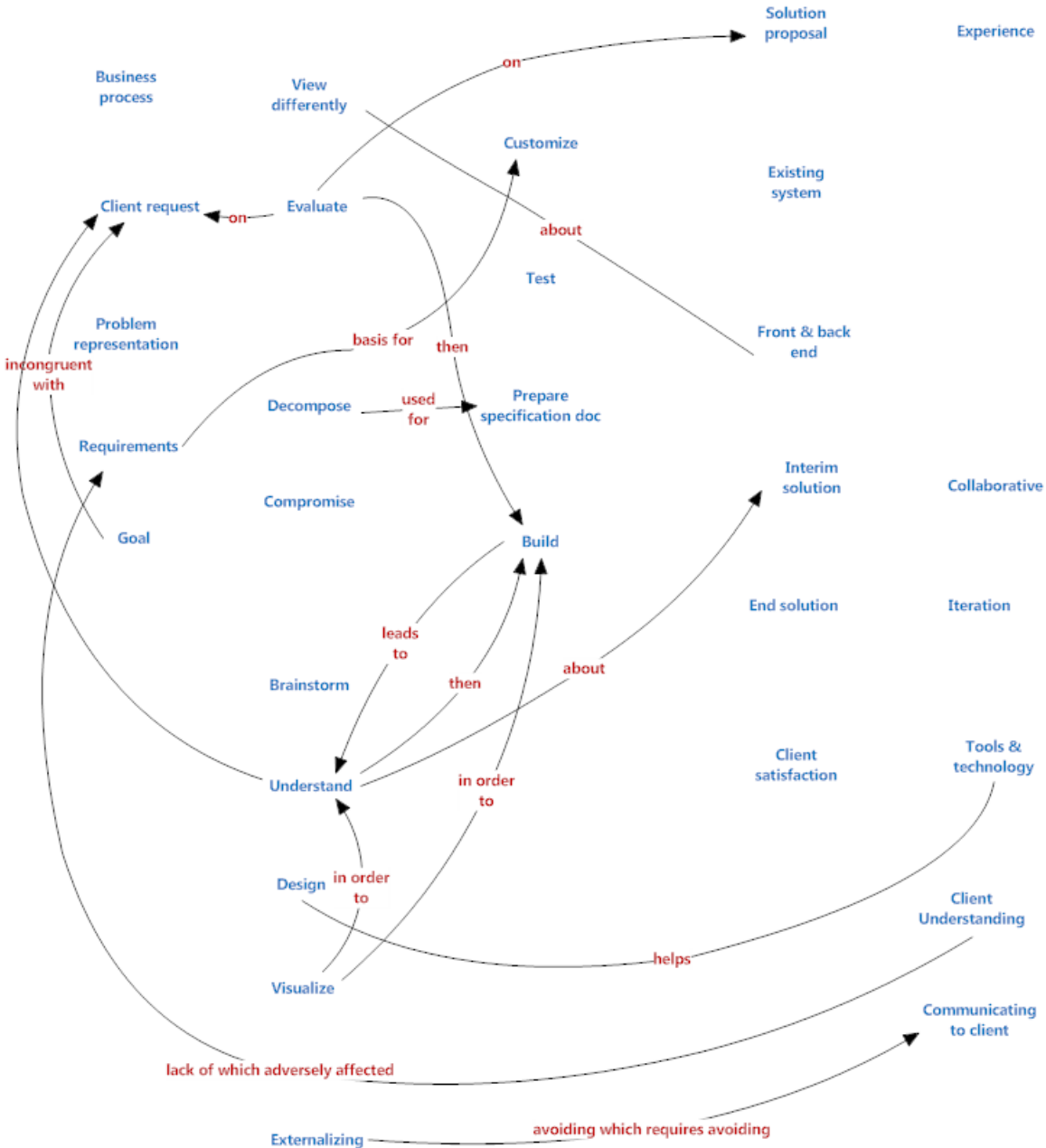
C.6 Kyle



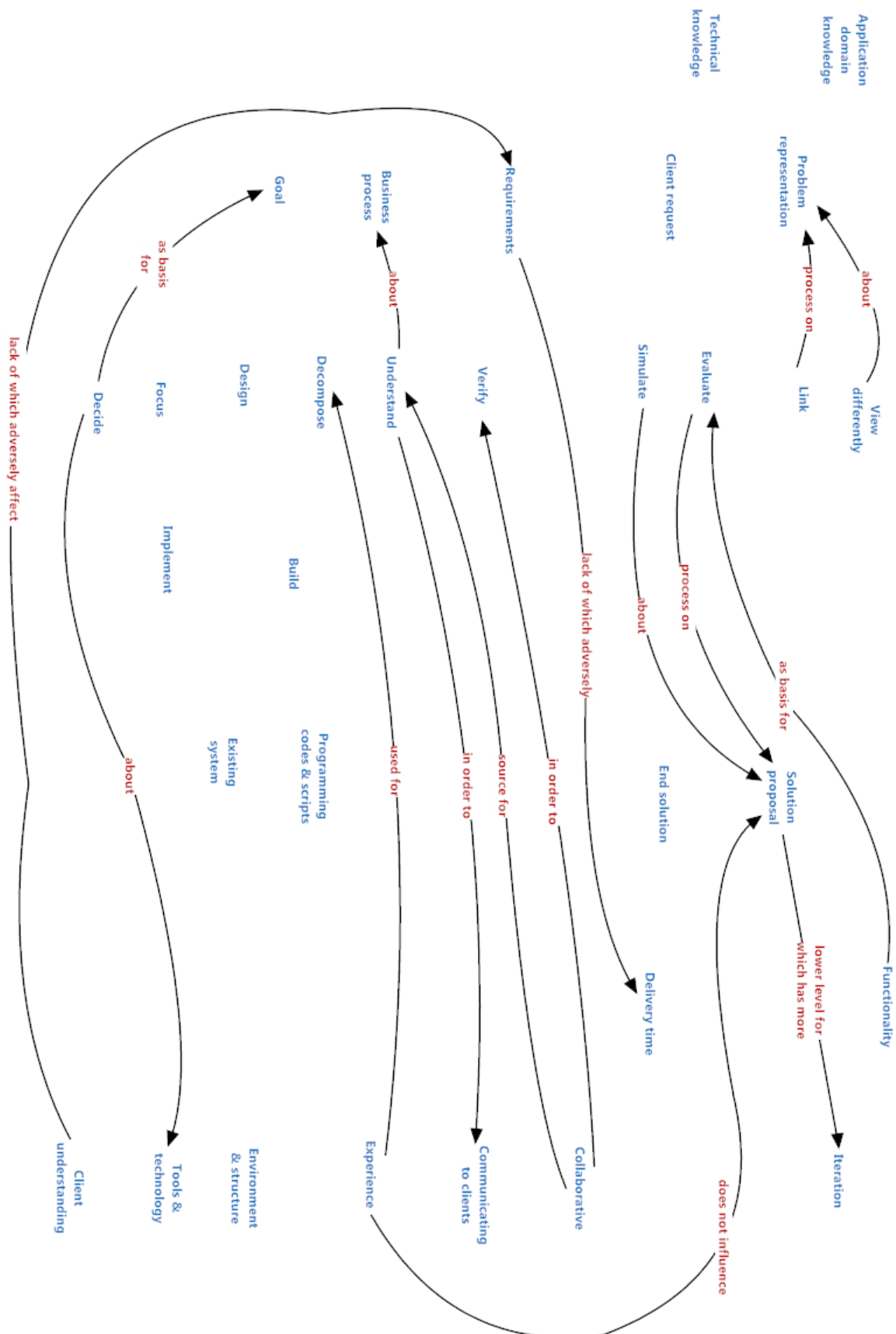
C.7 Patrick



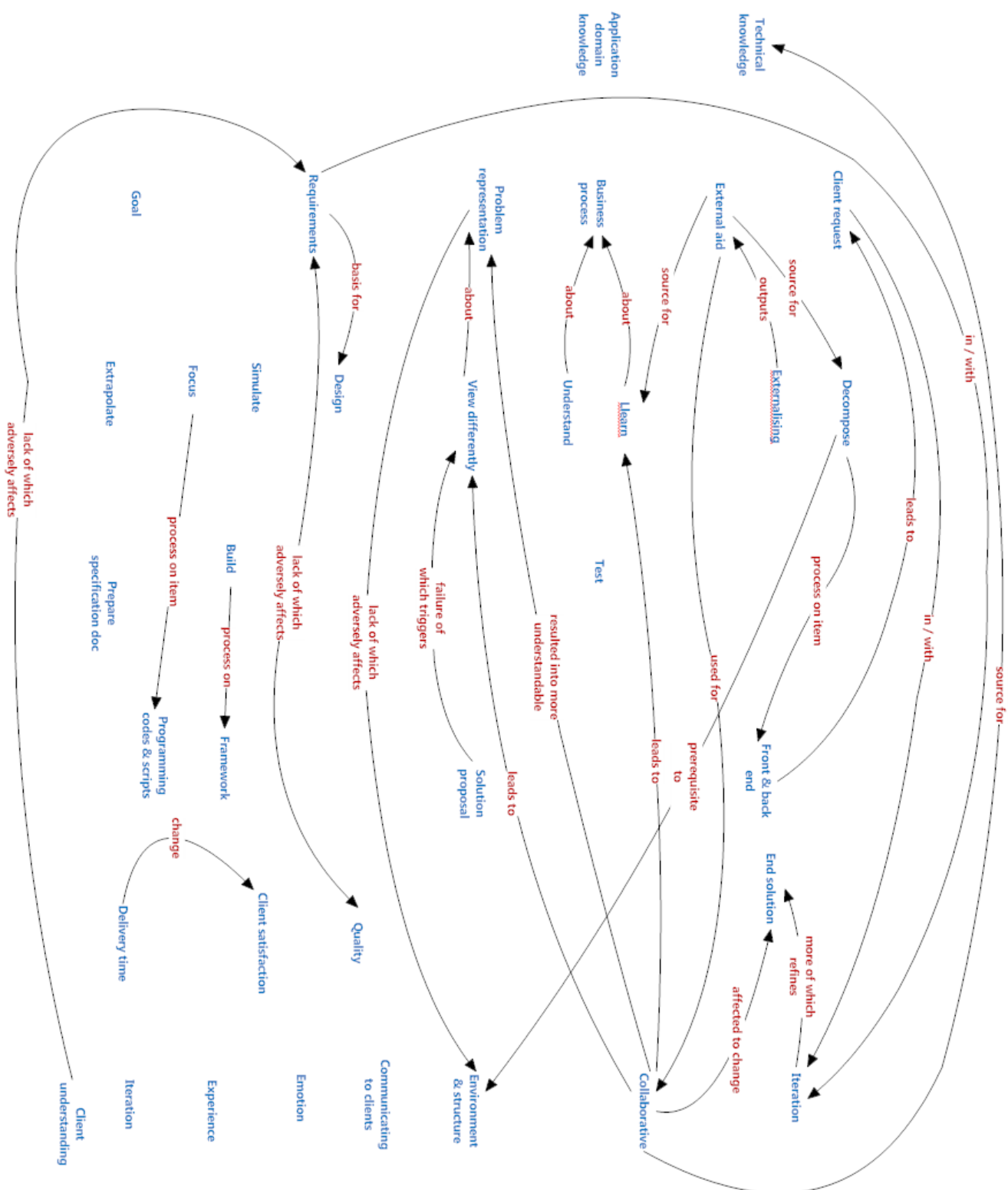
C.8 Rachel



C.9 Rick



C.10 Sam



Appendix D

LIST OF INITIAL CODES

Name	Sources	References	Created On	Created By
a possible database design requires additional business process	1	1	14 11 2012 11:09 AM	KA
acknowledge importance of knowing what clients want	1	2	16 11 2012 4:00 PM	KA
acknowledge importance of understanding the business	1	1	15 11 2012 12:44 PM	KA
actual coding done from identified computer steps	1	1	15 11 2012 4:25 PM	KA
actual development occurs after internalising	1	1	15 11 2012 12:30 PM	KA
actual software usage not as perceived	1	1	16 11 2012 3:05 PM	KA
advantages of tool	1	1	27 12 2012 3:07 PM	KA
alternative external solution - reason for not using	1	3	02 11 2012 9:46 AM	KA
although restricted by the tool, solution still satisfied client needs although not exactly as wanted	3	4	15 11 2012 2:47 PM	KA
although tools restricted, but actually forced for solution that is more in-line with user	1	1	14 11 2012 11:24 AM	KA
ask expert coder to understand what existing codes do	1	3	14 11 2012 1:56 PM	KA
asking hypothetical questions on changes of business processes	1	1	02 11 2012 3:42 PM	KA
asking questions to allow understanding among the questionee	2	2	12 11 2012 6:36 PM	KA
assumption based on prior task was incorrect	1	2	16 11 2012 4:12 PM	KA
attempted to collect requirements, user not clear enough	3	4	12 11 2012 11:59 AM	KA
aversion to know about project	1	1	12 11 2012 5:22 PM	KA
avoid difficulty by sticking to correct database design rather than user request	1	1	27 12 2012 11:09 AM	KA
aware of the mental process required	1	1	02 11 2012 3:08 PM	KA
Back end works, front end ugly	1	1	12 11 2012 10:28 AM	KA
Back-end oriented new system	1	1	12 11 2012 10:26 AM	KA
basis for breaking up	3	3	01 11 2012 2:50 PM	KA
basis for design concept includes user demographic	1	1	13 11 2012 12:18 PM	KA
basis for selecting between options	1	2	15 11 2012 3:05 PM	KA
Being anal	1	1	12 11 2012 12:25 PM	KA
big change from existing system	1	1	02 11 2012 4:08 PM	KA
bits that are overlooked include the back-end part	1	1	14 11 2012 11:55 AM	KA
break down tasks	3	3	15 11 2012 3:47 PM	KA
break existing module or program	1	1	16 11 2012 4:59 PM	KA
break up program pieces so that people can each work on their own	1	2	14 11 2012 3:42 PM	KA
breaking down in iterations by avoiding dependencies - more ideal strategy	1	1	12 11 2012 6:13 PM	KA
breaking down tasks in iterations using less-than optimal strategy	1	2	12 11 2012 6:11 PM	KA
breaking up based on steps	1	2	14 11 2012 3:37 PM	KA
breaking up by looking at user actions	1	1	14 11 2012 11:39 AM	KA
breaking up computer steps relied on internal knowledge	1	1	15 11 2012 4:27 PM	KA
breaking up functionalities	1	1	12 11 2012 11:50 AM	KA
breaking up into interface, database and web service	1	1	14 11 2012 11:40 AM	KA
breaking up 'methods' makes it more difficult to understand	1	1	14 11 2012 2:14 PM	KA
Breaking up problem representation	1	1	01 11 2012 2:28 PM	KA
breaking up using diagrams	1	1	12 11 2012 12:43 PM	KA
bring developers into same type of understanding	1	1	31 12 2012 4:36 PM	KA
business process	2	2	14 11 2012 10:52 AM	KA
by being transparent, tool aids developer to understand how tool or software works	1	1	16 11 2012 3:48 PM	KA
capability of system despite lack of requirements	1	1	12 11 2012 5:20 PM	KA
challenge; how to turn concepts into process	1	1	12 11 2012 12:20 PM	KA
change in design done for things not covered in concept	1	1	14 11 2012 11:16 AM	KA
change request is a factor for iteration	2	2	15 11 2012 12:36 PM	KA
changing nature to requirements	1	1	12 11 2012 11:56 AM	KA
cleaning a design, after concept is agreed	1	1	14 11 2012 11:36 AM	KA
clear understanding of technical aspect of general solution	1	1	01 11 2012 3:38 PM	KA
client asked for improved interface only after software seems to work	1	1	12 11 2012 5:28 PM	KA
client changes	1	1	31 12 2012 4:19 PM	KA
client dissatisfaction about time	1	1	12 11 2012 5:04 PM	KA
client instructions	1	1	12 11 2012 10:27 AM	KA
client pleasantly surprised with software despite problems	1	1	12 11 2012 5:25 PM	KA
client request	2	2	16 11 2012 10:37 AM	KA
client seldom dissatisfied with requirement not fulfilled due to limiting tool	1	1	15 11 2012 2:49 PM	KA
clients rectify what they said	1	1	12 11 2012 12:36 PM	KA
co-design	1	1	13 11 2012 11:46 AM	KA
compare simulated in mind with tool test	1	1	02 11 2012 10:23 AM	KA
Comparing current task and previous task	1	1	31 10 2012 3:11 PM	KA
comparing with hypothetical case	1	1	02 11 2012 9:42 AM	KA
Concept solution was formed at initial view of problem representation	3	3	01 11 2012 2:52 PM	KA
conducive working environment	1	1	16 11 2012 11:19 AM	KA
confidence in manual rather than automated scripting	1	2	02 11 2012 10:06 AM	KA
consequence for not having formal requirements	2	2	12 11 2012 6:31 PM	KA
constant awareness of current situation	1	1	15 11 2012 3:48 PM	KA
contained in internal memory	3	4	02 11 2012 3:18 PM	KA
content of external aid	1	1	02 11 2012 3:15 PM	KA
content of requirements document	1	3	13 11 2012 12:05 PM	KA
could not break down plan because lack information	1	1	12 11 2012 5:49 PM	KA
critical source of domain representation	1	1	31 12 2012 4:23 PM	KA
current task	6	10	14 11 2012 1:33 PM	KA
customize interface	1	1	16 11 2012 3:53 PM	KA
defining the problem	2	2	15 11 2012 12:23 PM	KA
delving into an idea	1	1	16 11 2012 6:06 PM	KA
description of technical details of software	1	1	15 11 2012 4:00 PM	KA
design based on computer processes	1	1	15 11 2012 12:29 PM	KA

design based on requirement	1	1	15 11 2012 12:27 PM	KA
design front end as asked by user	1	2	01 11 2012 3:28 PM	KA
design on paper, to help internalise	1	1	15 11 2012 12:28 PM	KA
design rough interface until fairly agreed, before cleaning and actual back end	1	1	14 11 2012 11:37 AM	KA
design solution together with another who talked to the clients	1	1	16 11 2012 3:20 PM	KA
Design test to strain requirements	1	1	12 11 2012 12:48 PM	KA
design triggered client to realise their need	1	1	15 11 2012 12:38 PM	KA
design was based on previous knowledge	1	1	16 11 2012 3:30 PM	KA
determine action related to item in problem representation	1	1	01 11 2012 3:23 PM	KA
developer decided to develop system after finding no suitable package	1	1	31 12 2012 11:04 AM	KA
developer pro towards methodology	1	1	31 12 2012 11:28 AM	KA
developer request for representation not met	1	2	12 11 2012 5:07 PM	KA
developer understands application domain more than clients	1	1	12 11 2012 5:32 PM	KA
development approach difficult because lack familiarity	1	1	16 11 2012 5:38 PM	KA
development approach, contain difficulties	1	1	16 11 2012 5:37 PM	KA
difference between customise and develop	1	1	15 11 2012 2:52 PM	KA
difference in current and previous module developed	1	1	16 11 2012 5:35 PM	KA
differences between design task and implementation	1	1	16 11 2012 10:22 AM	KA
different conceptualisation among developers - due to working alone	1	1	31 12 2012 4:42 PM	KA
different documents changes hands in between stakeholder	1	2	15 11 2012 2:56 PM	KA
difficult to find information from clients	1	2	12 11 2012 5:36 PM	KA
difficulty in getting a design to work	1	1	14 11 2012 11:28 AM	KA
difficulty of using tools because no-standarad and lack training on it	1	1	16 11 2012 3:56 PM	KA
difficulty to make tools do what developer wanted	1	2	14 11 2012 11:25 AM	KA
discussion between stakeholders regarding requirement	2	3	16 11 2012 3:04 PM	KA
disputing problem representation	1	4	02 11 2012 9:30 AM	KA
dissimilar previous tasks	1	1	12 11 2012 10:17 AM	KA
dissimilarity between current and previous task	1	2	13 11 2012 12:00 PM	KA
divert from original porting task, to modify framework to suit the task	1	1	14 11 2012 2:30 PM	KA
do not need to understand the big picture of the software to execute small tasl	1	1	16 11 2012 5:48 PM	KA
do own script, but use tool to test the script	1	1	02 11 2012 10:18 AM	KA
do program housekeeping to facilitate other developers understanding	1	1	31 12 2012 11:06 AM	KA
Done many times makes familiarity of the domain material	2	3	01 11 2012 2:54 PM	KA
done similar before	2	2	31 10 2012 11:15 AM	KA
draw database tables to find peculiarities	1	1	27 12 2012 3:01 PM	KA
draw diagrams to understand	3	5	14 11 2012 1:52 PM	KA
drawing aids understanding	1	3	26 12 2012 10:49 AM	KA
earlier representation of solution	2	2	02 11 2012 11:12 AM	KA
early problem representation	2	3	02 11 2012 9:33 AM	KA
early process	1	1	02 11 2012 9:28 AM	KA
easy request, do immediately; difficult request, documnt first.	1	1	27 12 2012 11:18 AM	KA
educate user on the lack of understanding of their own request	1	2	27 12 2012 10:53 AM	KA
efficiently solved problem using own solution rather than requested by client	1	1	16 11 2012 5:58 PM	KA
enlightening clients on how system works	2	2	12 11 2012 6:34 PM	KA
establishing relations between concepts identified during understanding	2	3	12 11 2012 12:21 PM	KA
evaluate feasibility and time of specific client request	2	2	15 11 2012 12:21 PM	KA
evaluate options in the mind, choose one that matches, and then proceed to develop	2	3	15 11 2012 3:11 PM	KA
exception handling not built into system	2	2	14 11 2012 12:09 PM	KA
exclusive cases where one works on other person's part	1	1	15 11 2012 3:01 PM	KA
existing system	2	2	15 11 2012 3:38 PM	KA
expected users to be happy because their close involvement	1	1	16 11 2012 5:43 PM	KA
experience guides task performance	1	1	13 11 2012 12:07 PM	KA
experience in technical aspect of problem	1	1	31 12 2012 10:55 AM	KA
external aid for low level problems, math functions	1	2	12 11 2012 2:27 PM	KA
external aid helps uncover things	1	1	12 11 2012 2:30 PM	KA
external aid of lower level problem is for other people	1	1	12 11 2012 2:29 PM	KA
extra functionality built after implementation	1	1	14 11 2012 12:11 PM	KA
extra functionality is to account for user error	1	1	14 11 2012 12:13 PM	KA
extrapolate problem with clients to economic symptoms	1	1	12 11 2012 5:40 PM	KA
failure in doing something unfamiliar	1	1	02 11 2012 10:39 AM	KA
feedback of user to refine design	1	1	14 11 2012 11:46 AM	KA
finalised probelm representation results in less iteration of solution	1	1	01 11 2012 3:44 PM	KA
finally gets the required documentation of processes	1	2	16 11 2012 10:18 AM	KA
find out whether users aware consequences of their request	1	1	27 12 2012 10:41 AM	KA
find out why solution did not match expected	1	1	16 11 2012 10:19 AM	KA
'finding' for a compromise solution	1	1	13 11 2012 12:36 PM	KA
focus diverted to one particular purpose of solution	1	1	31 12 2012 4:05 PM	KA
forming questions regarding existing system	1	1	02 11 2012 3:07 PM	KA
forming solution is 'obvious' with clear problem representation	1	1	01 11 2012 3:04 PM	KA
frequent delivery improved satisfaction	1	1	21 12 2012 3:45 PM	KA
front-end different from back-end	1	2	27 12 2012 11:01 AM	KA
generalising or making assumption	1	1	16 11 2012 4:09 PM	KA
get further info if given is not enough to execute internal rule	1	1	01 11 2012 3:10 PM	KA
getting familiar with requirements and concept design	1	1	15 11 2012 3:44 PM	KA
goal helps to guide how to achieve goal	1	1	14 11 2012 11:32 AM	KA
goal of concept solution	1	1	31 12 2012 10:59 AM	KA
guide other developers to focus and isolate from political noise	1	2	12 11 2012 5:56 PM	KA
hands-on as way to get familiar with system or tool	1	1	15 11 2012 12:16 PM	KA

how business process knowledge links with technical aspect	1	1	14 11 2012 2:02 PM	KA
how system works	1	1	13 11 2012 11:31 AM	KA
How working 2 people together is good	1	4	12 11 2012 3:10 PM	KA
html level programming	1	1	02 11 2012 10:10 AM	KA
hypothesize reduce iteration if requirements were more detail	1	1	16 11 2012 3:45 PM	KA
hypothetical case - change in process or strategy	1	2	02 11 2012 9:41 AM	KA
idealistic situation	1	1	02 11 2012 3:09 PM	KA
identified problem from viewing existing half-baked solution	1	1	26 12 2012 11:01 AM	KA
identify problem and work on same understanding of problem among developers	1	1	31 12 2012 4:39 PM	KA
implemebtation took longer time because changing requirement	1	1	21 12 2012 3:52 PM	KA
implement what client asked for, without realising what they need	1	2	15 11 2012 12:39 PM	KA
incorporate other work into solution of this particular problem	1	2	31 12 2012 11:27 AM	KA
increase complexity from one iteration to another	1	1	31 10 2012 3:32 PM	KA
incremental understanding among developers of specifics in design, after general concept design	3	3	14 11 2012 11:52 AM	KA
info oevrload with book-type external aid	1	1	02 11 2012 10:54 AM	KA
info refined from user iteration meeting	5	6	12 11 2012 6:19 PM	KA
initial selling stage is to get client to 'feel' the system is what they need	1	1	14 11 2012 11:45 AM	KA
interested in similar task	1	1	31 10 2012 11:18 AM	KA
interface design aided database design	1	1	13 11 2012 11:45 AM	KA
internal representation covered most needed in programming task	1	1	15 11 2012 4:22 PM	KA
internal rule for solution	1	2	01 11 2012 3:08 PM	KA
issue of lack of support for the developers	1	1	12 11 2012 5:18 PM	KA
issue of not telling enough to client	1	1	12 11 2012 5:03 PM	KA
iteration ends up in scope creep	1	1	31 12 2012 5:59 PM	KA
iteration to suit specific user requirement	1	1	16 11 2012 4:03 PM	KA
iteration was based on comparing the codes with internal knowledge	1	1	14 11 2012 2:48 PM	KA
iterations contain knowledge	1	1	12 11 2012 6:15 PM	KA
iterations keep improving system	2	2	12 11 2012 6:17 PM	KA
iterations to get agreement on how solution should look	3	4	14 11 2012 11:43 AM	KA
justify chosen development methodology	1	1	16 11 2012 5:44 PM	KA
justify strategy of base technology layer	1	1	12 11 2012 11:57 AM	KA
justify strategy to gain info for internal representation	1	1	12 11 2012 2:25 PM	KA
justify why normal startegy is not chosen	1	2	01 11 2012 3:39 PM	KA
justifying designing test	1	1	12 11 2012 2:16 PM	KA
knowing a certain relevant technology or tool	1	1	13 11 2012 12:08 PM	KA
knowledge sharing in segregated task	1	1	15 11 2012 12:45 PM	KA
lack of formal analysis and design stage	1	2	12 11 2012 11:42 AM	KA
lack of program documentation reason for difficulty in unde4rstanding	1	1	27 12 2012 11:39 AM	KA
Lack of proper organisation of collaborative work	1	1	31 12 2012 11:30 AM	KA
learn by working with other people	2	2	12 11 2012 6:09 PM	KA
less change in representation through deduction from previous implementation	1	1	21 12 2012 3:43 PM	KA
less formal development process	1	1	31 12 2012 11:16 AM	KA
less need for documentation because code is already self explanatory, by using sensible names	1	1	14 11 2012 2:08 PM	KA
less understanding of business process	1	1	14 11 2012 1:41 PM	KA
lesson learned from previous task	1	1	13 11 2012 12:12 PM	KA
limited involvement in design	1	1	15 11 2012 3:50 PM	KA
link front end design to back end	1	1	01 11 2012 3:25 PM	KA
long time looked at similar task	2	3	31 10 2012 11:17 AM	KA
look for work-around if client request is not feasible	1	2	15 11 2012 12:22 PM	KA
looking external guide for something not done before	1	2	02 11 2012 10:37 AM	KA
major info from lead develpr was regarding business logic	1	1	15 11 2012 4:20 PM	KA
make changes based on request	2	2	16 11 2012 3:18 PM	KA
make client understand changes to how system works	2	2	12 11 2012 6:38 PM	KA
make client understand how business process would change	1	1	15 11 2012 12:13 PM	KA
many instances of doing similar task	3	5	31 10 2012 11:28 AM	KA
match user and lead developers design and reconcile	1	1	15 11 2012 3:53 PM	KA
minor change to existing system	1	1	27 12 2012 10:43 AM	KA
missing elements uncovered while building system	1	2	14 11 2012 11:49 AM	KA
mixed complexity of task	1	1	15 11 2012 2:53 PM	KA
mixed success of reviewing differnt people's codes	1	1	14 11 2012 2:36 PM	KA
mixed understanding of technical aspects of system	1	1	14 11 2012 1:40 PM	KA
modify database design with using external aid	1	1	13 11 2012 11:53 AM	KA
module worked on was very small compared to the whole software	1	1	16 11 2012 5:02 PM	KA
more than expected amount of framework modification	1	1	14 11 2012 3:25 PM	KA
multiple viewof single item	1	1	01 11 2012 2:56 PM	KA
name (verb) item in problem representation	1	1	01 11 2012 3:22 PM	KA
need for proper project management	1	2	31 12 2012 11:07 AM	KA
need to learn new programming language for the task	1	1	14 11 2012 2:06 PM	KA
need to make assumptions in order to start on task	1	1	16 11 2012 5:50 PM	KA
need to understand business processes	5	5	02 11 2012 3:12 PM	KA
need to understand half-baked solution by previous developer	2	3	26 12 2012 10:56 AM	KA
negative approach to build supplier-client relationship	1	1	12 11 2012 5:08 PM	KA
new system developed to interface the actual system solution	1	1	31 12 2012 4:03 PM	KA
New system to process transactions differently	1	1	12 11 2012 10:25 AM	KA
new to programming language was factor for not looking at several possible solutions	1	1	14 11 2012 2:52 PM	KA
no formal requirements - but obtained via prototype	1	1	31 12 2012 4:14 PM	KA
no need for external guide for something familiar	2	2	02 11 2012 10:40 AM	KA
no need to refine front-end, all in head from previous task	1	1	02 11 2012 4:04 PM	KA

no time target	1	1	21 12 2012 3:52 PM	KA
non-fundamental change suggested by other expert	2	3	14 11 2012 11:14 AM	KA
not difficult due to very clear goals	2	3	14 11 2012 2:41 PM	KA
not enough problem representation to excuse task	1	2	15 11 2012 4:09 PM	KA
not enough testing	2	2	14 11 2012 3:54 PM	KA
not familiar with user processes	1	1	16 11 2012 3:36 PM	KA
not sure of current status of project	1	1	16 11 2012 10:13 AM	KA
objects being designed	1	1	16 11 2012 3:29 PM	KA
obtain comments on ideas and change accordingly	1	1	12 11 2012 6:07 PM	KA
obtain understanding of tasks from lead developer	1	1	15 11 2012 4:06 PM	KA
one definition of success - achieve overall goal of project	1	1	31 12 2012 4:12 PM	KA
one definition of success - does what the software is told to do	1	1	31 12 2012 4:11 PM	KA
only become familiar with worked on aspect of the big software	1	1	16 11 2012 5:46 PM	KA
people understand abstractions differently	1	1	12 11 2012 6:00 PM	KA
perceiving what users think	1	1	26 12 2012 10:29 AM	KA
poor execution of chosen methodology	1	1	31 12 2012 4:07 PM	KA
positive perception towards seeking info on domain	1	1	12 11 2012 5:31 PM	KA
prefer user define objective of problem but not ways to solve	1	2	26 12 2012 10:39 AM	KA
preference to work alone on task to avoid communication problem	1	1	27 12 2012 11:45 AM	KA
present options to clients and allow them to choose	1	1	15 11 2012 3:07 PM	KA
presenting options to client or user	1	1	02 11 2012 10:43 AM	KA
Pressure from inside people	1	1	12 11 2012 11:40 AM	KA
previous process - seek only to work on finalised problem representation	1	3	01 11 2012 3:43 PM	KA
previous similar task	3	4	31 10 2012 11:20 AM	KA
previous task - put short term goal as basis for strategy	1	2	01 11 2012 3:30 PM	KA
prioritise issues, collaboratively	1	1	31 12 2012 4:38 PM	KA
problem of software implemented was not part of requirement	1	1	14 11 2012 12:08 PM	KA
Problem representation	6	11	31 10 2012 3:14 PM	KA
problem representation suggested solution method	2	2	01 11 2012 3:20 PM	KA
problem symptoms of implemented software	1	2	14 11 2012 11:59 AM	KA
problems sorted out early	1	1	02 11 2012 11:02 AM	KA
process into software solution	3	4	14 11 2012 10:54 AM	KA
process quite mechanical after front end agreed	1	3	01 11 2012 3:40 PM	KA
protested against lack of analysis stage, but conceded	1	1	12 11 2012 11:43 AM	KA
re-acknowledge importance of technical knowledge	1	1	14 11 2012 2:04 PM	KA
reason for port	1	2	14 11 2012 1:34 PM	KA
reason for strategy of base level technical	1	2	12 11 2012 11:54 AM	KA
reason for strategy of looking from different view	1	1	12 11 2012 12:28 PM	KA
reason for tool choice - possible vast sources of information	1	1	31 12 2012 4:27 PM	KA
reason for tool selection - support from tool founders	1	1	31 12 2012 4:26 PM	KA
reason for verifying idea with others	1	1	15 11 2012 3:03 PM	KA
reasoning for solution representation	1	2	01 11 2012 3:34 PM	KA
reexamine programming script, and test	1	1	02 11 2012 10:04 AM	KA
refactor of codes was tested	2	2	14 11 2012 2:49 PM	KA
refactor similar methods into a single method	1	3	14 11 2012 2:39 PM	KA
regurgitate issues discussed in a document	1	1	02 11 2012 4:56 PM	KA
relief for disengaging from project	1	1	12 11 2012 5:09 PM	KA
requirements facilitated task	2	5	12 11 2012 5:51 PM	KA
requirements not enough detail	1	1	16 11 2012 3:39 PM	KA
respect and execution of ideas	1	1	12 11 2012 6:08 PM	KA
review newly written codes	1	1	14 11 2012 2:35 PM	KA
rework codes because new to programming language	1	1	14 11 2012 2:45 PM	KA
run script to understand existing software and figuring out what needs to be done	1	1	16 11 2012 3:23 PM	KA
strategy of thinking a lot of ideas and working on them in turns	1	1	16 11 2012 6:08 PM	KA
scope creep resulted in more time needed	1	1	31 12 2012 6:00 PM	KA
search info on domain	1	1	12 11 2012 5:30 PM	KA
seek external guide if cannot remember	1	1	02 11 2012 10:52 AM	KA
seek further info on problem representation	2	3	01 11 2012 2:55 PM	KA
segregation of task	2	2	15 11 2012 12:44 PM	KA
segregation of unit vs whole testing	1	1	15 11 2012 2:57 PM	KA
selected tools provided part of database design	1	1	13 11 2012 12:27 PM	KA
Selecting tool used before	2	3	14 11 2012 10:58 AM	KA
share problems and solution with other developers	1	1	31 12 2012 5:54 PM	KA
short notice for a sub-task	1	1	02 11 2012 11:06 AM	KA
show interface design to users	1	1	15 11 2012 3:08 PM	KA
Similar task - build back-end after front-end agreed	1	1	31 10 2012 11:26 AM	KA
similar task - determine whether problem that arose on the system is within own control	1	1	02 11 2012 11:03 AM	KA
Similar task - not difficult	1	1	31 10 2012 11:29 AM	KA
Similar task - prototype	1	1	31 10 2012 11:24 AM	KA
Similar task - prototype 'spot on'	1	1	31 10 2012 11:25 AM	KA
Similar task - seeking explanation to the problem	1	1	31 10 2012 11:23 AM	KA
similar task - understanding beyond the surface of problem representation	1	1	02 11 2012 9:36 AM	KA
similarity of current with previous task is only in the programming language	1	2	15 11 2012 4:10 PM	KA
similarity of previous task to current task	4	5	01 11 2012 2:49 PM	KA
simple but long programming script	4	7	02 11 2012 10:03 AM	KA
simple database solution	1	1	02 11 2012 9:55 AM	KA
simple scripting	1	1	02 11 2012 9:56 AM	KA
software implemented has problems, to abide by required deadline	1	1	14 11 2012 11:58 AM	KA

software too large to fully understand	2	2	14 11 2012 1:37 PM	KA
software worked beyond expectation despite problems	1	1	12 11 2012 5:23 PM	KA
solution changes business process	1	1	15 11 2012 12:11 PM	KA
solution goal restricts the tools or methods used	2	2	02 11 2012 10:32 AM	KA
solution is formed based on requirements	1	2	15 11 2012 12:08 PM	KA
solution representation is fixed, how to get there can be changed	1	1	02 11 2012 10:30 AM	KA
solution to impracticality of user request	1	2	13 11 2012 12:34 PM	KA
solution to problem	3	5	14 11 2012 11:07 AM	KA
some limitation of tool, but was overcome	1	2	31 12 2012 4:31 PM	KA
some task don't require business knowledge	1	1	15 11 2012 4:02 PM	KA
some user request came after a solution representation	3	4	02 11 2012 11:15 AM	KA
source of problems of implemented software	1	1	14 11 2012 12:01 PM	KA
source to get familiar with tools	1	1	15 11 2012 12:15 PM	KA
Sources containing problem representation	5	6	02 11 2012 3:04 PM	KA
specific request cannot be met due to practical and regulatory concern	3	4	13 11 2012 12:30 PM	KA
specifics changed more than the higher level	1	2	16 11 2012 10:16 AM	KA
startegy - problem solution should work independently	1	1	31 12 2012 11:09 AM	KA
startegy of focusing on one possible solution and working on it	1	1	14 11 2012 11:41 AM	KA
strategy - solve base level technical problems	1	1	12 11 2012 11:52 AM	KA
Strategy - develop front end separately first	3	3	01 11 2012 3:16 PM	KA
strategy - get comfortable with problems not yet solved, and let them talk to each other.	1	2	31 12 2012 11:14 AM	KA
Strategy - look at the problem a different way	1	2	12 11 2012 12:27 PM	KA
strategy - to design concept, solve the more familiar details first	2	2	02 11 2012 4:13 PM	KA
strategy of assigning responsibility	1	1	13 11 2012 11:54 AM	KA
strategy of choosing alternative soonest possible before being more technical	1	1	27 12 2012 11:52 AM	KA
strategy of getting user to agree on front-end design, and only then start developing	2	2	15 11 2012 3:09 PM	KA
strategy of looking at several options after gaining understanding	1	2	15 11 2012 3:05 PM	KA
strategy of not taking too long on an idea when there are othe ideas	1	2	16 11 2012 6:05 PM	KA
Strategy of previous tasks - develop front and back end closely together	1	1	01 11 2012 3:17 PM	KA
strategy of thinking how to software conceptually executes step by step	2	4	13 11 2012 11:50 AM	KA
strategy of using one path and reworking on it	3	3	14 11 2012 2:55 PM	KA
strategy of working at in parallel on different parts, and test together	1	1	13 11 2012 11:58 AM	KA
strategy to agree on database table first and then develop application, web interface and webservices	1	1	13 11 2012 11:51 AM	KA
strategy to design general concept to get evryone at the same level of uderstanding	1	1	14 11 2012 11:51 AM	KA
strategy to gather info to stor in internal representation	1	1	12 11 2012 2:24 PM	KA
strategy to think of different ways of implementation, their pros and cons	1	1	27 12 2012 11:48 AM	KA
strategy to understand big picture first before programming	1	1	27 12 2012 11:51 AM	KA
strategy to work in small bits	2	3	14 11 2012 11:33 AM	KA
stratgy of doing small codes and test that it works, and going back to modify	2	2	14 11 2012 11:34 AM	KA
strong understanding of business processes	1	1	02 11 2012 3:23 PM	KA
structured development phases	1	1	15 11 2012 12:00 PM	KA
sufficient understanding to execute task	2	2	14 11 2012 1:38 PM	KA
synergy effect of collaborative work	1	1	31 12 2012 4:24 PM	KA
system completed late because of late requirements document	1	2	16 11 2012 10:40 AM	KA
tactic to faciliilate co-developers cooperation	1	1	12 11 2012 6:01 PM	KA
take up others responsibility to ensure project moves	1	1	12 11 2012 5:53 PM	KA
talk to other develeopr to gain domain understanding	1	1	16 11 2012 5:11 PM	KA
talk to previous developers to understand exisiting system	1	1	27 12 2012 11:26 AM	KA
task difficult because understanding previous half-baked solution	1	2	27 12 2012 11:38 AM	KA
task involves learning	1	2	14 11 2012 2:06 PM	KA
task more difficult because continuing from others	1	1	27 12 2012 11:43 AM	KA
task performed on time	3	3	02 11 2012 11:05 AM	KA
team closeness factor to not feel difficulty in development	1	1	12 11 2012 5:45 PM	KA
team member source of problem representation	1	1	31 12 2012 4:15 PM	KA
technical aspect quite ok in terms of time	1	1	12 11 2012 5:16 PM	KA
test an idea	1	1	16 11 2012 6:04 PM	KA
test the design	1	1	12 11 2012 12:44 PM	KA
the more specific the external aid, the more preferred	1	1	02 11 2012 10:55 AM	KA
think of more familiar area first to propose the less familiar	1	1	02 11 2012 4:11 PM	KA
thinking of negative effect of following user request	1	2	26 12 2012 10:35 AM	KA
thinking on how to solve a problem	1	1	14 11 2012 11:06 AM	KA
thought task was easy, maybe because he was given the low level technical part	1	1	15 11 2012 4:18 PM	KA
throw away modified version of previous module and rewrite	1	1	16 11 2012 5:36 PM	KA
time factor may play role in selecting method	1	1	02 11 2012 10:15 AM	KA
time is a factor for not looking at several possible solutions	2	2	14 11 2012 11:41 AM	KA
tool allow to generate output database tables	1	1	16 11 2012 3:52 PM	KA
tool helped	1	1	16 11 2012 5:40 PM	KA
tool helped verify code and suggest better code	1	1	14 11 2012 3:50 PM	KA
tool restricts from database table modification	1	1	16 11 2012 3:50 PM	KA
tools do not dictate solution	3	3	02 11 2012 10:28 AM	KA
tools restrict work to only go proper way	1	1	21 12 2012 3:49 PM	KA
tools used are considered the best way to achieve task	2	2	15 11 2012 4:17 PM	KA
tools used restricted solution implementation	3	5	14 11 2012 11:22 AM	KA
training to get familiar with system	1	1	15 11 2012 12:15 PM	KA
translate how one program language handle things into another language	1	1	14 11 2012 2:22 PM	KA
typical strategy as oppose to current task	1	1	12 11 2012 5:38 PM	KA
understand by looking at codes of half-baked solution	1	1	26 12 2012 10:59 AM	KA
understand by queries to isolate gaps to be filled	1	1	15 11 2012 12:18 PM	KA

understand by reading existing codes	3	3	14 11 2012 1:55 PM	KA
understand each problem leads to understand how problems link	1	1	31 12 2012 5:55 PM	KA
understand problem representation	2	3	01 11 2012 3:20 PM	KA
understand the problem	2	2	15 11 2012 4:23 PM	KA
understand user process by seeing how they do it	1	1	16 11 2012 3:36 PM	KA
understanding business process	2	3	12 11 2012 12:15 PM	KA
understanding of business processes achieved during discussion	3	4	02 11 2012 3:17 PM	KA
unfamiliar part of task	1	1	02 11 2012 10:37 AM	KA
unfamiliar with application domain	3	4	12 11 2012 5:29 PM	KA
un-utilised part of solution	1	1	31 12 2012 4:06 PM	KA
usage of the solution	1	1	16 11 2012 3:03 PM	KA
use completed requirements doc and compare with toolkit and see how to modify toolkit	1	1	15 11 2012 2:56 PM	KA
use different sequence of process to fit user request	1	1	01 11 2012 3:15 PM	KA
use of method that is understood better, even if more manual	1	2	02 11 2012 10:14 AM	KA
use tool to secure and isolate database	1	1	02 11 2012 10:20 AM	KA
use tool to understand components in software	1	1	16 11 2012 3:26 PM	KA
use tools	5	7	14 11 2012 11:18 AM	KA
user ask for database change, not understanding	1	1	26 12 2012 10:34 AM	KA
user generally satisfied	5	7	16 11 2012 4:13 PM	KA
user problem	1	1	31 12 2012 11:01 AM	KA
user testing to identify bugs	1	1	27 12 2012 11:34 AM	KA
using a tool that he aware of, but not used	1	1	14 11 2012 11:21 AM	KA
utilising a technology for a task	1	1	31 10 2012 11:19 AM	KA
verbalising aids identification of problem	1	1	16 11 2012 11:20 AM	KA
verify ideas on a task with other developers	3	3	15 11 2012 3:02 PM	KA
very complex task	1	1	31 12 2012 4:33 PM	KA
very general level of early problem representation	2	2	12 11 2012 11:37 AM	KA
very minor help needed from lead regarding programming language	1	1	15 11 2012 4:21 PM	KA
view problem in two separate perspective; user and developer	1	1	12 11 2012 12:03 PM	KA
viewed technical knowledge as less important, probably because automatic	1	1	14 11 2012 2:03 PM	KA
visual aid helps finding how to achieve	1	1	14 11 2012 11:31 AM	KA
what sprint meeting should be about	1	1	12 11 2012 6:32 PM	KA
what user request is against what they need	1	1	27 12 2012 10:54 AM	KA
when a mental model is wrong, forget about it and build new model	1	1	16 11 2012 10:57 AM	KA
wide range of project task	2	2	12 11 2012 5:42 PM	KA
with other developer, work on same task but different part of system	1	1	15 11 2012 3:00 PM	KA
without externalising, difficult to understand	1	1	14 11 2012 11:30 AM	KA
Work a way around the framework	1	1	14 11 2012 2:29 PM	KA
work on half-baked solution	1	2	26 12 2012 10:52 AM	KA
work-around the limitation of software	1	1	16 11 2012 4:18 PM	KA
working together improves task output	1	3	12 11 2012 3:09 PM	KA
working together makes easier	1	1	14 11 2012 11:27 AM	KA
write in small 'method's to allow unit-test	1	1	14 11 2012 2:10 PM	KA
write on external aid	3	6	02 11 2012 3:13 PM	KA
write up spec document based on info asked from client	1	1	15 11 2012 11:59 AM	KA

Appendix E

INTER-RATER RELIABILITY SUMMARY

Category No.	Categories\sub-categories	After reconcile Kappa	Before reconcile Kappa	
1	Activities\Build	0.771	0.000	
1	Activities\Coding	0.683	-0.005	
1	Activities\Configure	0.776	0.390	
1	Activities\Customise	0.578	0.578	
1	Activities\Implement	0.688	2.43E-14	
1	Activities\Modify framework	0.505	0.505	
1	Activities\Port software	0.567	0.567	
1	Activities\Prepare specification document	0.741	0.180	
1	Activities\Review code	0.809	-0.001	
1	Activities\Standardising and cleaning	1.000	1.000	
1	Activities\Test	0.716	0.078	
2	client understanding	0.670	0.199	
3	Team and collaboration	0.659	0.277	
4	Communicating to client	0.517	0.151	
5	Environment and structure	0.624	0.156	
6	Knowledge and experience\Application domain knowledge	0.732	1.000	
6	Knowledge and experience\Experience in similar task	0.502	0.502	
6	Knowledge and experience\technical knowledge	0.540	0.540	
7	Information\business process	0.685	-0.012	
7	Information\client request	0.705	0.127	
7	Information\existing system	0.787	1.000	
7	Information\external aid	0.707	-2.37E-14	
7	Information\Goal	0.591	0.591	
7	Information\Problem representation	0.510	0.185	
7	Information\requirements	0.666	0.391	
8	Iteration	0.731	0.362	
9	Mental process\brainstorm	0.631	0.185	
9	Mental process\break down	0.611	0.245	
9	Mental process\Compromise	0.746	0.746	
9	Mental process\Decide or choose	1.000	1.000	
9	Mental process\Defining the problem	0.574	0.574	
9	Mental process\design	0.669	0.317	
9	Mental process\diagnose	0.000	0.000	Not included
9	Mental process\Do and uncover	-1.34E-13	-1.34E-13	Not included
9	Mental process\Evaluate	0.561	0.273	
9	Mental process\externalising	0.721	-0.002	
9	Mental process\extrapolate	0.620	0.620	
9	Mental process\Focus	0.873	0.195	
9	Mental process\Learning	0.797	0.427	
9	Mental process\Link	-0.001	-0.001	Not included
9	Mental process\Refactor	0.108	0.108	Not included
9	Mental process\Simulate	0.531	-0.002	
9	Mental process\understand - developer	0.570	0.224	
9	Mental process\Verify	0.897	0.000	
9	Mental process\view differently	0.738	-0.006	
10	Solution object\End solution	0.467	0.017	
10	Solution object\Framework	0.716	2.53E-14	
10	Solution object\Front-end and back-end	0.686	0.686	
10	Solution object\Interim solution	1.000	1.000	
10	Solution object\Module	1.000	1.000	
10	Solution object\Programming codes and scripts	0.646	0.217	
10	Solution object\solution proposal	0.815	0.000	
11	Success\client satisfaction	0.475	0.475	
11	Success\Cost	0.973	1.000	
11	Success\Delivery time	0.597	0.373	
11	Success\Functionality	0.763	0.763	
11	Success\Quality	0.837	1.000	
12	Tools and technology	0.536	0.536	
	Emotion	0.000	0.000	Not included

Appendix F

**INFORMATION SHEET AND CONSENT
FORM**

MENTAL REPRESENTATION FOR PROBLEM SOLVING

INFORMATION SHEET FOR PARTICIPANTS

Thank you for showing an interest in this project. Please read this information sheet carefully before deciding whether or not to participate.

The Aim

The project is part of a PhD thesis. In general, the study aims to examine the thinking processes of software engineers in software design.

Your Role

The study involves software engineers in various forms of organisations. Should you agree to take part in this project as a software engineer,

- You will be interviewed with open ended questions. This may involve up to 2 interviews taking a maximum of 30 minutes each.
- The interview session may require the researcher to observe your task engagements.
- Copies of your design materials will be kept and analysed by the researcher. This may involve materials of meeting, design diagrams and other relevant documents.

The Information to be Collected

The interviews, observations and document analysis cover the method and information the software engineers gather from users regarding user requirements, how they conceptualise the software design and how this conceptualisation changes over time, and discussion material and design documentation for the development of software. This project involves an open-questioning technique where the precise nature of the questions which will be asked have not been determined in advance, but will depend on the way in which the interview develops. Consequently, although the Department of Information Science is aware of the general areas to be explored in the interview (see the Interview and Observation Protocol), the department has not been able to review the precise questions to be used. This study will also involve the use of audio recordings during interviews and observations.

Your Confidentiality

The data collected will be securely stored in such a way that only the PhD student and his supervisors will be able to gain access to it, and used solely for the purpose of this study. At the end of the project any personal information will be destroyed immediately except that, as required by the University's research policy, any raw data on which the results of the project depend will be retained in secure storage for five years, after which it will be destroyed.

The results of the project may be published and will be available in the library but every attempt will be made to preserve your anonymity. The use of such results will be limited to academic purposes and will not cause any disadvantage to you or your organisation. Reasonable precautions will be taken to protect and destroy data gathered by email. However, the security of electronically transmitted information cannot be guaranteed. Caution is advised in the electronic transmission of sensitive material.

Your Decision

Please be aware that you may decide not to take part in the project, or you may withdraw from participation in the project at any time, or decline to answer any particular question(s) during the interviews, without any disadvantage to yourself or your organisation of any kind. If you decide to participate, please sign and date the Consent Form for Participants.

Results and Questions

You are most welcome to request a copy of the results of the project should you wish. If you have any questions about our project, either now or in the future, please feel free to contact either:-

[PhD student name]
Department of Information Science
University Telephone Number:-

or

[Supervisor's name]
Department of Accountancy & Business Law
University Telephone Number:-

This project has been reviewed and approved by the Department of Information Science,
University of Otago.

MENTAL REPRESENTATION FOR PROBLEM SOLVING

CONSENT FORM FOR PARTICIPANTS

I have read the Information Sheet concerning this project and understand what it is about. All my questions have been answered to my satisfaction. I understand that I am free to request further information at any stage.

I know that:-

1. my participation in the project is entirely voluntary;
2. I am free to withdraw from the project at any time without any disadvantage;
3. the audio-tapes will be destroyed at the conclusion of the project but any raw data on which the results of the project depend will be retained in secure storage for five years, after which it will be destroyed;
4. this project involves an open-questioning technique where the precise nature of the questions which will be asked have not been determined in advance, but will depend on the way in which the interview develops and that in the event that the line of questioning develops in such a way that I feel hesitant or uncomfortable I may decline to answer any particular question(s) and/or may withdraw from the project without any disadvantage of any kind
6. the results of the project may be published and available in the library but every attempt will be made to preserve my anonymity;
7. I understand that reasonable precautions have been taken to protect data transmitted by email but that the security of the information cannot be guaranteed;

I agree to take part in this project.

.....

(Signature of participant)

.....

(Date)

This project has been reviewed and approved by the Department of Information Science,
University of Otago.

MENTAL REPRESENTATION FOR PROBLEM SOLVING

INTERVIEW AND OBSERVATION PROTOCOL

This interview protocol indicates the constructs of interest and related questions that may be asked in the interviews or gathered during the observations. As the questions are open ended, further questions may develop during the interview in pursuant to the project aims. However you may decline to answer or withdraw to participate if you feel hesitate and uncomfortable with the development of the interviews and observations.

Internal representation of the Problem Domain

- 1 How much do you know about the domain of the system?
- 2 What do you think would be required in the system?
- 3 Does the perceived application requirement change?
- 4 What changed the perceived requirement?
- 5 How does your knowledge guide you in making in making mental models of the system?

External Problem Representation

- 1 What did you ask during their user-developer meeting?
- 2 Which other references do you use for understanding user requirement?
- 3 Do you collect any material/document from the user stating their requirement?
- 4 To what extent do your team members' thoughts influence your mental model?
- 5 Do the software development tools guide aid your software design task?

Problem-solving Task

- 1 Explain the types of tasks given to you in systems design
- 2 What are the goals for your tasks?
- 3 How complex do you think the tasks are?
- 4 Are the tasks interconnected?
- 5 How do tasks assigned to you relate to those of your team members?

Mental Representation for Task Solution

- 1 What do you do to form a mental model of the system?
- 2 What tools do you use to aid in forming the mental model?
How do you rely on existing internal and external representation of the problem to form
- 3 mental model of the system?
- 4 Explain your strategy in selecting the most representable mental model.
- 5 How do you utilise previous iteration of mental models to produce a better mental model?

Problem Solving Performance

- 1 Are the system's functionalities clear and easy to be understood?
- 2 How do the functionalities address the tasks of the user?
- 3 Is the system completed in time?
- 4 Are users satisfied with the system?
- 5 What concerns are raised regarding the system?