AN EXPLORATION OF ASSUMPTIONS IN REQUIREMENTS ENGINEERING

By Sujatha Bulandran

This thesis is presented as partial fulfillment of the requirements for the degree of Doctor of Engineering in Information and Communications Technology in the School of Electrical, Electronic & Computer Engineering,
The University of Western Australia

January 2012

DECLARATION

I submit the thesis entitled 'An Exploration of Assumptions in Requirements Engineering' in partial fulfilment of the requirements for the award of Doctor of Engineering in Information and Communication Technology (DEICT) and declare that it is my own account of my research and contains as its main content work which has not previously been submitted for a degree at any tertiary education institution.

_____

Sujatha Bulandran

Perth, Western Australia January 2012

ABSTRACT

The aim of this thesis is to explore the issue of assumptions made during Requirements Engineering (RE). As the initiating phase of a software development process, RE involves activities which are expected to fulfil the needs of the user. The defects which originate during RE are particularly expensive to rectify when uncovered in the later stages of development. Assumptions made in RE, particularly during requirements analysis, are a significant source of defects and contribute to the total rework cost of the software. Therefore, there is a need to make visible and verify these assumptions in order to reduce the overall development cost.

This research examines the adaptation of a standard defect detection technique for revealing assumptions during requirements analysis. This is an extension of the previous literature which largely emphasizes the importance of detecting assumptions in software projects via automated tools. A process model for the research, termed the Exploration of Assumptions in Requirements Engineering (EAiRE) has been constructed by defining assumptions in the context of RE. In support, there was a need for a Taxonomy of Assumptions in Requirements Engineering (TARE) to enhance this investigation. Several important principles for detecting and inserting artificial assumptions are defined and explained. Further, two experimental trials were designed (a Scenario Based Experiment and an Assumptions Seeding Experiment).

The results of the experiments demonstrated that assumptions can be detected using the suggested approach. The number of the assumptions detected, particularly in relation to the size of the requirements documents used in this study, exceeded expectations. It is clear that it is worth investing greater effort on the detection and

measurement of assumptions in RE since this is where many defects originate. The discovery of assumptions at this initial stage of system development has the potential of significantly enhancing the quality of the delivered software.

AKNOWLEDGEMENT

Foremost, this thesis would not have been possible without my supervisor, Dr. Terry Woodings. I would like to express my sincere gratitude for his continuous support and encouragement of my doctorate studies. He had guided and helped me to conduct my research. I appreciate his time and constant support in assisting me to complete my thesis. I am also grateful to my co-supervisor A/Prof Gary Bundell for validating and suggesting ideas to improve my thesis. Next, I would like to thank my fellow colleagues undertaking PhD studies and staffs of School of Electrical and Electronic Engineering for their help and support. I also would like to take this opportunity to thank the students of UWA who have participated in the experiments conducted in this study for their involvement and feedbacks. Finally, I thank my husband, dad, mum and siblings for their help and motivation during my studies. They have always believed in me to accomplish my doctorate studies.

Table of Contents

List of Tables

List of Figures

Chapter 1 Introduction

1.1 Issues Regarding Requirements in Software Development

The process of developing software incorporates various tasks and participants that expected to fulfil the software requirements. Despite successful software development projects, there are studies reporting a significant number of project failures. For instance, a survey done by Oxford University and Computer Weekly in the United Kingdom on the status of IT project management revealed that only 16% of software projects were regarded as successful [1]. Another exploration carried out by The British Computer Society stated that less than 1% of the 500 developments projects were considered to have fulfilled the criteria for success [1].

The size and complexity of requirements in software projects are increasing. The definition of the requirements early in the software development life cycle is meant to cope with this complexity. Hence, the Requirements Engineering (RE) field is becoming an increasingly important aspect of the software development process.

Multiple studies have revealed that major causes of errors reported in software development projects are either directly related to requirements, or can be derived from requirement issues:

- By referring to several case studies, Leffingwell and Widrig [2] have stated that one-third of the total defects delivered in software projects are derived from requirements errors;
- Requirements processes are considered as the source of most (50 percent or more) critical quality problems in software development [3];

- The top five causes of poor software cost estimation are related to issues with requirements, mainly involving frequent requirements changes, missing requirements, insufficient user communication, poor specifications, and insufficient analysis [4];
- Embedded real-time software has similar error patterns to other types of software. The major sources (36%) of errors found are related to requirement problems [5].

The 2003 Standish Group report showed that the success rate of software projects was approximately 34%. There was an improvement by 50% for this rate compared to the first survey which recorded 16% of software project success reported in 1995 [1]. In the 1995 report, the Standish Group [6], which has surveyed 8,000 software projects reported by 352 companies, explained that issues related to requirements were the critical factor for project failures. The survey, which required the participants to respond on the reasons that cause projects to be challenged, confirmed that more than one-third (as depicted in Figure 1.1) are issues related to requirements.

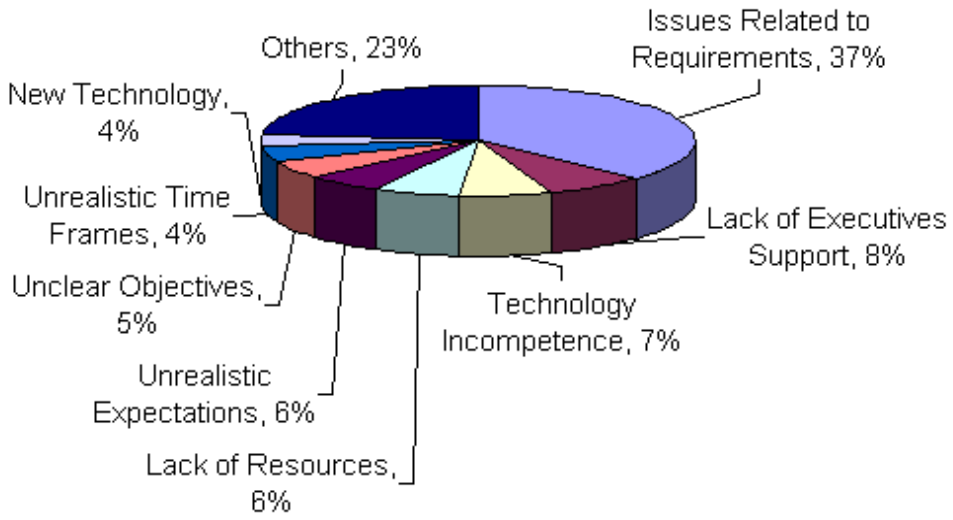


FIGURE 1.1 – *The factors that causes projects to be challenged in Standish Group (derived from [6])*

This result is backed up by another investigation concerning a United States Air Force project [7]. Requirements defects (including issues related to requirements translation and incomplete requirements) consisted of 41% of the total error count. In this study, issues were classified in terms of causes of defects in order to assist the developers to decide the necessary effort and resources needed to tackle the causes.

Errors originating from requirements are one of the highest percentages in most software projects. These errors can be categorized in several groups. A survey gathered from an aircraft project showed the types of requirement errors with the respective frequency of occurrence in percentages – incorrect facts (49%), omissions (31%), inconsistency (13%), ambiguity (5%) and misplaced (2%) [8, 9].

The major consequence of requirements problems is the amount of rework. This requires developers to work on an issue which they thought was already completed. Rework can expend up to 40% to 50% of the total software development budget [10, 11]. It is important to note that the normal cost for developers to find and fix a requirement defect can easily stretch to 100 times more than tackling the same defect during requirements development [12]. As a result, shortcomings in requirements during software developments can incur large overruns in budget and schedule. Therefore preventing requirements errors and identifying them early has a huge influence on reducing rework cost.

As discussed in the previous paragraph, errors created at the initial stage of development are the largest contributor to the total rework cost. Assumptions in requirements can be classified as a special type of factor which will lead to defects if not

overtly discussed and validated. It is likely that there is a relationship between the assumptions and rework costs in RE.

This research involves the exploration of a special type of factor, assumptions made during requirements gathering, which have a non-zero probability of being false. Therefore, requirement statements which have a zero probability of being false are not assumptions. As we had described in the previous paragraph, the rectification of these assumptions will lead to additional cost when they are detected at the later stages of the development. In order to illustrate the relationship between assumptions and rework cost during RE, we characterize these following elements in the form of sets and functions:

- *E* represents the set of all types of issues or elements which cause errors in requirements statements;
- *e* represents elements of that set – which cause errors in requirements statements;
- *A* represents the set of all the assumptions made during requirements analysis;
- *a* represents a particular assumption made during requirements analysis;
- *c* represents the function giving the cost of repair for *e* ; and
- *v* represents the function which signify the validity of *a* and has the values 'true' or 'false'.

Now, we illustrate the following correlations:

$\forall e \in E, \exists c\,(e) > 0$

$\forall a \in A, \exists\ v\,(a) > 0$

As asserted earlier and by definition, all assumptions have the potential to be incorrect.

So, $\forall a \in A$, *the probability (v (a) = false)* > 0

$\therefore A \subset E$

$\therefore \forall a \in A, a \in E$

$\therefore c(a) > 0$

Following the above relationships, it is possible to quantify for each assumption:

Risk exposure (a) = probability (v (a) = false) * c (a)

Therefore, unknown (and project dependent) size of the risk is given by:

Total Risk = $\Sigma_{\forall a \in A}$ probability (v (a) = false) * c (a)

Some other elements of *E* derived from the studies done by Vinter [13] on the bugs taxonomy and statistics developed by Beizer [14]. The elements includes incorrect requirements, issues related to requirements logic, requirements ambiguity, requirements completeness, requirements verifiability, issues related to requirements presentation and requirements changes. In other research, analysing requirements bugs, Vinter and Lauesen [15] found that all the subcategories related to the requirements errors found by Beizer, contributes 51% of the total bugs. Vinter and Lauesen have grouped the requirements related bugs into four major groups. These categories are misunderstood requirements (38%) as the highest contributor for requirements bugs, followed by missing requirements (29%), changed requirements (24%) and other requirements related bugs (9%). Hence, assumptions can be one of the significant causes of misunderstood requirements.

Assumptions are principally an important issue in RE since it is being a part of the cause for software defects. Hence, we urge that, issues related to assumptions needs serious attention – identification and resolution of this issue is essential in software

development. If efforts are allocated to validate assumptions during RE, we believe there will be a significant change in the number of requirements errors originating from invalid assumptions. There is a potential saving of cost and time if the proportion of requirements errors caused by invalid assumptions is treated at the earliest stage as possible during development.

Assumptions may be made at any stage of development by the project stakeholders who are involved in the development. Most of them are made during the requirements analysis or elicitation phases. RE has been defined as the branch of systems engineering which is concerned with the desired properties and constraints of software-intensive systems, the goals to be achieved in the software's environment and assumptions about the environment [16]. The idea of a software-intensive system describes an interrelated set of human activities supported by computer technology. The purpose of this system is expressed by requirements. The constant involvement of human activities allows assumptions to sneak in, mainly through the interpretation of requirements besides other factors during system implementation (design decisions, operational domain, environment and characteristics of input data).

For the purpose of our research, we use the analogy of noise in communication for describing assumptions in requirements engineering (as shown in Figure 1.2). Assumptions in this context are represented as three sources of noise. The quality of a message delivered in communication system is distorted by noise. The same phenomena occurs during requirements gathering when stakeholders from various background interact with each other. The client might convey a particular message *P* (for instance, their needs or requirements about the system) to the developer. In most cases, the developer on the other side would have heard the message as *Q* from the client or

message that developer wants to hear or expect to hear [17], but in actual fact the client did not intend to communicate *Q* (developer's assumptions about what client intended to convey). In this case, the sum of assumptions which influences message *P* produces *Q*. These assumptions which influence the communication between stakeholders during requirements elicitation can be either valid or invalid. The differences between valid and invalid assumptions are a source of noise which disrupts the quality of message intended to be delivered by the clients to developers. The degree of distortion shown in Figure 1.2 describes how messages can vary from the sender's and receiver's point of view respectively.

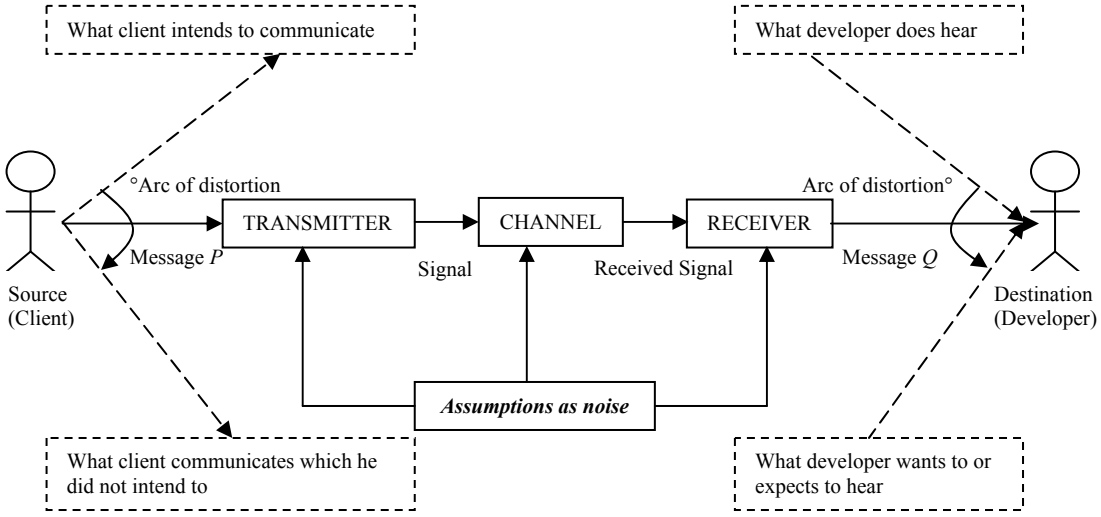


FIGURE 1.2 – *Assumption as an element of communication*

In communication, the senders might presume that the receiver had received the message correctly. The receiver may assume that the senders had sent the correct message. There are available mechanisms in communication to detect and resend the messages which may have been dropped during the transmission period. Hence, we can check the message which was sent but not the messages which was not sent. Thus, in this case, the messages which were not sent are assumptions.

The invalid assumptions generated during RE are a threat to the requirements development process. Thus, frequently eliciting, validating and documenting them in earlier stages of software projects will be highly recommended. The investigation carried out in this thesis intends to reveal the rate for occurrences of assumptions during the analysis of Requirements Specification Documents. A higher degree of invalid assumptions will have an increasingly significant impact on the quality of software. Therefore, the information gathered from our research will enable us to evaluate the importance of managing assumptions in RE.

1.2 Assumptions Management in Requirements Engineering

Case studies have shown examples of projects failure caused by invalid assumptions. Some of the observation, as illustrated by Lehman [18], have revealed the impacts of assumptions which led to disastrous failures (e.g. case studies of Ariane V, CERN Accelerator, Sinking of HMS Sheffield and London Ambulance System). Let us look at the Ariane V case study. Ariane V is a space rocket which changed direction from its flight path, disintegrated and exploded about 40 seconds after initiation of the flight sequence during its first launch. We realize that there were number of assumptions which contributed to the explosion of Ariane V as stated in the failure report [19]. The standard design for flight control system had been used for Ariane V. The rocket was equipped with Inertial Reference System (IRS) which was responsible for measuring the attitude of the launcher and its movements in space. The data gathered in IRS is transferred to On-Board Computer (OBC), which is relied upon to execute the flight program and to control the engine. The design of IRS which was being used in Ariane V was inherited from the design used in Ariane IV. Part of the

time sequence used for alignment function was based on the requirements of Ariane IV. However, it was not required in Ariane V, but it was still being implemented. An internal IRS operand error occurred when converting the horizontal bias variable from 64-bit floating to 16-bit signed integer value. This floating point had a value greater than the value which could be represented by 16-bit signed integer, causing an unexpected high value of an internal alignment function. Further, the data conversion instruction, which was programmed in ADA code, was not protected. The internal events which happened in the IRS unit led to the failure of Ariane V. According to the facts reported, during the designing process for the IRS used for Ariane IV and Ariane V, a decision was made that it was not necessary to protect the IRS computer from being made inoperative by an excessive value of the horizontal bias variable. No clear justification was found for this decision in the source code. It was believed that there should be some kind of agreement made during the development of the code; in this case the agreement was implicitly assumed though it was in fact not done. No doubt that the operand error was detected but it was managed inefficiently due to being dependent on the assumption that software should be considered to operate correctly until it is shown to be at fault. Therefore it was a definite risk when using critical equipment such as IRS to assume that it had been validated by previous use on Ariane IV. Hence, in this example of Ariane V, we learn that there was a gap in identifying the implicit assumptions made with regards to the software used for Inertial Reference System computer, which led to its failure during launching.

Khosrow-Pour [20] had elaborated the work of Schneider and Sarker in their case study of pre-implementation failure of an Information Systems development project, Computerised Maintenance Management System (CMMS) at a large public university. The main cause of this failure was overlooking the key stakeholders' interest

and needs during the selection of a vendor for development of CMMS; this caused dissatisfaction among the stakeholders. The managers who were in charge of the CMMS project assumed that, during the vendor selection criteria, they had taken into consideration the relevant factors with regards to the employees' needs. They had assumed that the purpose of some employees' participation in the project was merely for their attendance to be noted as important in the organisation. The preceding assumption was made by the managers due to some of the employees continued supporting the project even though they were unclear about the direction the project. On the other hand, from the employees' point of view, they were assuming that CMMS was mainly being developed for managements activities purposes. They had also assumed that their Information System department (which had worked together with the managers) was capable of selecting a reliable vendor for the project. However, these assumptions turned out to be wrong once the team realized that the roles of relevant but relatively hidden decision makers were overlooked during the pre-implementation of the project. This project was terminated at pre-implementation stage by the State Department of Information Services due to procedural errors.

The previous example was one where scenarios can exist in which invalid assumptions are not being clarified thus causing failure of a project even before the actual implementation of the project. The issue of assumptions is also a concern in the area of real time system development [21]. The elaboration of more case studies related to assumptions will be discussed in the next chapter, showing factors contributing towards project failures.

Assumptions, that were not well defined, validated and documented during software development, were identified as potential trauma. Assumptions affect the

actions or behaviors of stakeholders. In this case, we are referring to stakeholders as people who are directly or indirectly involved in a software development project. These individuals will be responsible for making decisions during the development process. Assumptions represent one of the factors which influence how a software project is planned and executed by its stakeholders. Identification and the resolution of conflicting assumptions remains a critical problem. This is an important but difficult challenge. Thus, research related to issues arising from assumptions is important.

A short survey among the major reference software engineering and project management books showed minimal coverage of assumptions management in requirements engineering. However, there are authors who have covered assumptions management in their writings. The details will be discussed in Chapter Two. This confirms that the importance of assumptions has been acknowledged by academics thus ensuring a gateway to an area of further research.

Despite the fact that existence of assumptions is a form of risk which propagates defects in the development process, there is no method defined to deal with assumptions at the initial stage of development. On the other hand, a method has been developed to recover assumptions related to software architecture from an existing software product in order to use that knowledge to improve the future projects [22]. Hence, there exists a need to develop a framework to research this issue and efforts taken to detect and confirm the correctness of these assumptions as early as possible. We believe that these efforts should be taken from the very first stage of software development, RE. In a comparison with assumption correction at other stages of software development, there are significant economic benefits of time, effort and increase in the users' satisfaction level.

1.3 Approaches for Assumptions Issues

According to our recent studies, current approaches tend to focus on the aspects of extracting and monitoring assumptions by developing tools (this will be further discussed in Chapter Two of this thesis). These efforts ultimately will have less involvement from stakeholders. In software development, the role of the human factor earns greater attention because, if only the technological factors are considered, the production of complete and efficient software is not compromised [23]. The research about assumptions needs to be more focused on stakeholders. They can be identified as any individual who contributes to the project. For instance, people from various backgrounds, such as technical (development team), business (client) and end-users, need to work as a team for developing the software. As such, the emphasis on stakeholders' involvement is essential because they are the entity which performs the action of assuming. Hence, the issue should be tackled by ensuring that the stakeholders take responsibility for their actions.

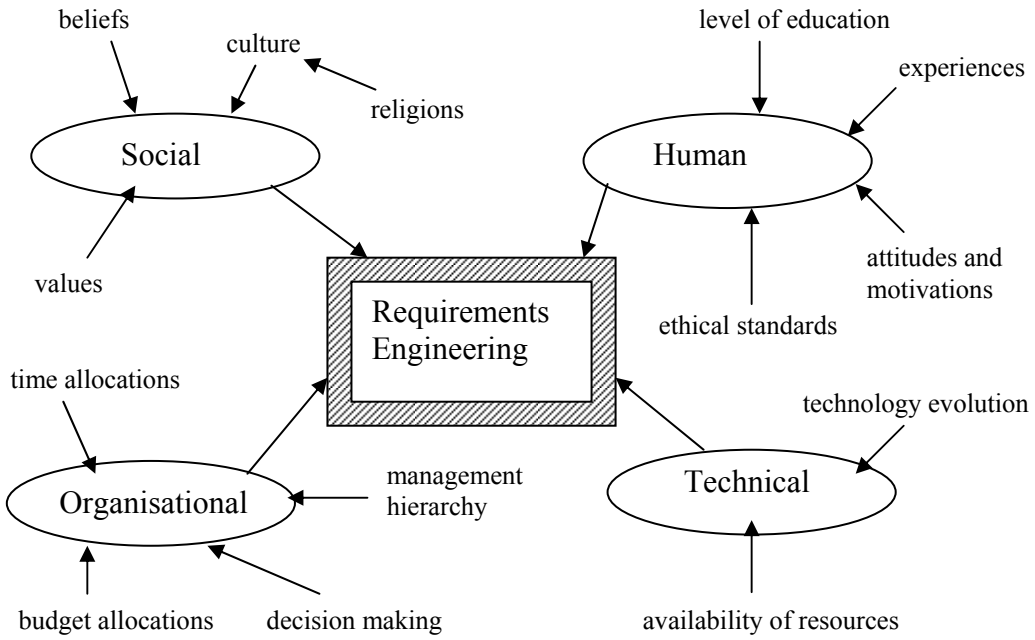


FIGURE 1.3 – *The elements which influence requirements engineering*

People have different needs that change over time and so do the stakeholder in software projects. The stakeholders come from different backgrounds, experience, levels of education, and, even more broadly, a diversity of culture, values, attitudes, ethical standards, and religions. These have a great impact in the way stakeholders make decisions and solve problems in projects. Figure 1.3 is recreated by using the information gathered from Sommerville [24] that elaborates the RE process encompasses of various stakeholders with the nature of human-centric activities. It is considered as the branch of software field which is the most manipulated by human, social, organizational and technical factors [24].

Therefore, decision has been made to focus the research on the initial stage of RE, the starting point which often seriously impacts on the outcome of the software projects. An exploration of assumptions in Requirements Engineering has been suggested and a workflow model called 'Exploration of Assumptions in Requirements Engineering' (EAiRE) has been developed in order to systematically tackle the issues of assumptions.

The cases of project failures as described in Section 1.2 indicates the minimal acknowledgment of the importance of assumptions to stakeholders during RE. This gap is addressed by strategies planned to elicit or determine assumptions during requirements elicitation based on the hypothesis below.

Standard defect detection techniques can be adapted for revealing assumptions in Requirements Engineering.

Defect detection techniques are used in software development to identify errors. Rework is required upon discovering the errors in order to produce a revised version

of the software. Assumptions are categorized as errors if it is proven to be invalid. Hence the hypothesis of this research is derived by using preceding argument.

There is definitely a need to manage the assumptions created during RE. A process model suggesting a method to manage these assumptions is illustrated in Figure 1.4. The assumptions are managed by frequently identifying, validating and documenting them during RE. Since, the software evolves rapidly, the assumptions tracing process needs to be done iteratively. The continuous effort of identifying, validating and documenting the assumptions in early stage of development will reduce the occurrence of incorrect assumptions. Hence, the focus of this research will be purely on the identification of assumptions in the initial phase of project, RE. Methods are developed to identify assumptions in RE and it will be further described in Chapter Three of this thesis.

The process model and concepts of this research will fit into most software development methodologies such as the Agile, Waterfall, Spiral and Iterative approaches.

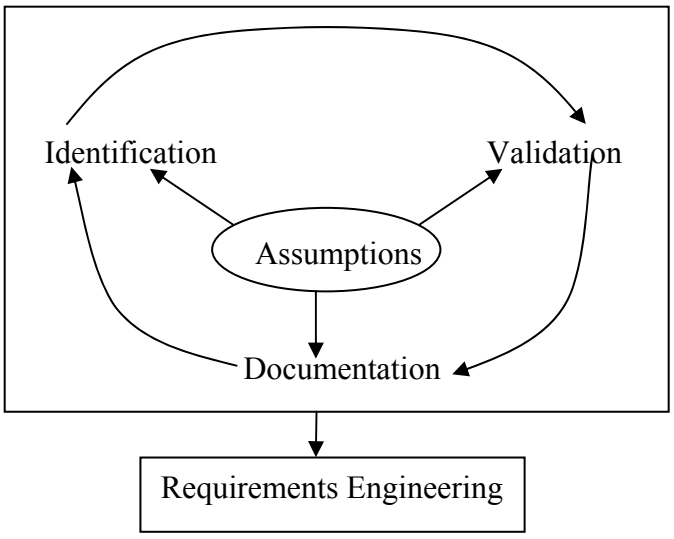


FIGURE 1.4 – *Assumptions Management in Requirements Engineering*

1.4 Contribution of this Research

This research is expected to provide a number of measurements and insights into the degree by which the occurrence of assumptions in RE influences the software development project. The primary goals of this research are described below:

(1) To tackle the issue of assumptions by attacking them at the initial phase of development process, Requirements Engineering;

(2) To indicate the size of assumptions (number of assumptions) revealed with regard to the size of the requirements specifications observed;

(3) To develop a classification of assumptions made during Requirements Engineering – taxonomy to enable us to systematically attack the issue of assumptions.

1.5 The Organisation of the Thesis

The thesis is organized into six chapters including the current introductory chapter which explains the problem. Chapter Two outlines previous research done in software engineering pertaining to assumptions management, while Chapter Three presents the methodology used in this research to tackle the issue of assumptions in Requirements Engineering. The proposed experiments, which have been developed and executed to gather data about assumptions, are elaborated in Chapter Four. The results and analysis from the observation of the experiments are presented in Chapter Five. Chapter Six is the concluding chapter of this thesis: it discusses the research findings and suggests further possible work which can be done in this area. Each chapter in this thesis is further detailed as below.

Chapter Two mainly discusses the realization of assumptions as an issue in software development. The previous work carried out on the assumptions issue focused more on the development of tools to extract, validate and document the assumptions during the designing and testing phases. Furthermore, history has shown that the significance of assumptions has been studied in numerous disciplines. This is further explained by the survey on interdisciplinary work concerning assumptions in some areas such as psychology and sociology and software architecture.

Since we believe that there is a strong need to resolve assumptions issues in software development, we have carried out a formal process of 'Exploration of Assumptions in Requirements Engineering' (EAiRE) in Chapter Three. EAiRE workflow model has been designed to suggest a systematic approach to handle assumptions as described below:

- First, assumptions are defined in the context of software development in order to learn the nature of assumptions during the development cycle;
- Second, a classification (taxonomy) of assumptions in Requirements Engineering is presented. The interaction between various elements of the taxonomy is explored;
- The chapter concludes with suggestions of experiments to study the size of assumptions generated with regards to the size of requirements specification.

Chapter Four outlines the framework for developing two experiments, Scenario Based and Assumptions Seeding based on a Requirements Specification Documents of a Mine Drainage Control System. Then, the method used for observing students' projects is described. Further, the principle of analysing assumptions used in Assumptions Seeding Experiment is illustrated.

Chapter Five presents insights on observations of students' projects. Two projects were monitored during the initial stage of development. The observation of these projects is discussed in detail, mainly explaining how the role of assumptions influenced the development process. A report is provided, the analysis of data gathered from both the experiments defined in Chapter Four by using principle of analysing assumptions. The results are presented to support my thesis argument and the importance of discovering assumptions at the earliest stage of development process.

In Chapter Six, the concluding remarks and the knowledge gained upon completion of this project is presented. I also suggest possible areas of further research to tackle assumptions issues in software developments.

Chapter 2 Literature Review

The necessity of a particular research subject is ascertained by the contributions of knowledge gained from that topic. In order to discover the contributions of research, it is essential to determine the gaps found in past investigations. As such, in Chapter Two, we will discuss and evaluate the concepts, theories and methodologies for studies of assumptions in multidisciplinary fields gathered from the appropriate literature. However, our primary focus is in the areas of Software Engineering.

2.1 Realization on Issues of Assumptions in Software Engineering

I discovered an early acknowledgement about assumptions in a panel discussion chaired by Greenspan [25]. He claimed that one presumes certain background information, such as the domain knowledge and environment, in order to define the requirements of a system which was being developed. This background information was referred to as assumptions and rationale. The main aim of the panel was to debate the ability to manage assumptions and not to highlight the importance of managing them in a project. This was the initial effort taken to venture into the assumptions topics in development. Research work which extended this effort to tackle further issues arising from assumptions was discovered later.

The panel brought forward several points: (1) there are existing tools available to capture and record assumptions; (2) there is a need to develop new techniques and tools to handle assumptions; (3) there are mechanisms available in Artificial Intelligence field

to manage assumptions and; (4) it is important to identify the right information to tackle assumptions, and tools are not necessary.

However, we understand that the importance of using automated tools, techniques and efficient modes of detecting assumptions was purely dependant on the type of scenarios encountered during the development phase. For instance, in embedded system, it is more efficient to utilize automated tools to record assumptions rather than to detect those assumptions using manual or other methods [21, 26, 27].

The efforts for tackling assumptions were challenged by raising questions such as those listed below (which I paraphrased from Greenspan et al [25]).

- Is the requirements statement consistent with the assumptions we have made? The preceding question leads to the importance of analyzing the effect of variations in assumptions
- Who needs to keep track of assumptions? And why?
- How do we elicit assumptions?
- How much reasoning can be done by tools?
- How do assumptions affect the overall system development methodology?
- One of the difficulties of recording assumptions is they rely on delayed gratification/satisfaction. Hence, how does the recording of assumptions clarify our thinking at the time?
- Cost incurred in eliciting and managing additional information. Hence, what are costs and benefits? Are these benefits worth their cost?
- What kind of information is needed to capture assumptions?
- How can assumptions be integrated into the requirements process? The preceding question can be tackled by using the existing tools which already have traceability mechanisms and/ or assumptions can be integrated into requirements documents.

The conclusion drawn from the above discussion suggests that exploration on assumptions issues is crucial and detecting them at the Requirements Engineering stage is an advantage.

The early observations by Lehman [28, 29] on assumptions related to software evolution and maintenance led to “Lehman's Uncertainty principle”. This says, “No E-type program can ever be relied upon to be correct” in the sense that while developing software, we cannot know whether all the spoken and unspoken assumptions are still valid, even though they may be valid at the point of time we took them. E-type software refers to software that addresses some real-world application, problem, or activity in some real-world domain. Thus the ever-changing environment of software itself can invalidate previously valid assumptions.

In addition, Lehman et al [30] discussed the importance of a feedback mechanism as “one of the eight laws” of software evolution in their paper. A feedback mechanism can contribute to studies related to the assumptions management. Lehman’s studies [18] on the role and impact of assumptions in software projects claimed that assumptions detected during development were probably the dominant cause and driver of software evolution. Hence, assumptions management is listed as one of the rules for software evolution, management and control. He urged that methods and tools to assist this process be developed or adapted from existing mechanism in Software Engineering. Therefore, as noted through Lehman’s works, the need for revealing and managing assumptions in software development is an important task. However, there is no concrete evidence found, in his papers, of methods that detect and manage assumptions.

Seacord [31], who studied Lehman’s work, stated that changes or invalid assumptions can contribute to insertion of defects. Hence, assumption management was suggested as a solution to tackle the problem. In this method, assumptions are recorded in source code and other software artefacts during the development process. Then, these assumptions are extracted into searchable database. The stored assumptions should

allow the system architects to assess the assumptions in order to determine the consistency of the assumptions with the system design. I suggest that the efforts to identify assumptions are essential, as agreed by Seacord, but this should be emphasized from the Requirements Engineering phase. The outcomes of this practise will be tremendous in early detection and removal of defects. I also second his idea of emphasizing the necessities of the assumptions management concepts by elaborating assumptions in structured English format; this is what has been done in experiments, Scenario Based and Assumptions Seeding, as elaborated in Chapter Five.

Parnas [32] has viewed software as similar to aging in humans, and suggested that we can not prevent this phenomenon. However, at least actions can be taken to minimize the effects of aging. He claimed that the failure of a new version of a software product was inherited from the old version of that software (assuming that the old version was faulty). In this case, a classic example will be the Ariane V case study, elaborated in Chapter One. The developers can not assume a new version of software should work as smoothly as the old version (assuming that the old version was working well without any faults). Therefore, in Ariane V case study, there is an urge to find the embedded assumptions in order to detect the root cause of the software failure.

Duboc et al [33] discussed assumptions issues in his scalability requirements studies. They described, in his research article, the usage of a goal-oriented requirements engineering (GORE) technique to elicit the scalability requirements of a financial fraud detection system. The requirements are often written so that the focus is only in the terms of application domain boundaries, which depend on assumptions that often change. Hence, it is significant to discover assumptions about the application domain of the current and estimated future system environment. Furthermore, I agree

with Duboc that the lack of methods to tackle assumptions has varied over time. Duboc claimed that there were no taxonomies for assumptions found by him during his studies. The claim for the importance of taxonomy for assumptions was made clear when Duboc urged that such taxonomy be developed for the purposes of future modelling efforts.

2.2 Development of Methods and Tools for Tackling Assumptions

As a continuation of work done by Seacord [31], a tool called Assumptions Management System (AMS) was created by Lewis et al [34] to beneficially manage assumptions. Assumptions are grouped into several categories such as environment, data, control, usage and convention. AMS records and extracts assumptions from Java source code into a repository and manages these assumptions in a web-based format.

The authors [34] claim that normally, assumptions are not documented and validated by people with the knowledge to verify their appropriateness. These people can be identified as system designer, system architect, tester, domain expert or any person that has the knowledge to determine whether an assumption made (by developer or stakeholder) is valid or invalid [34]. However, the methods of defining the appropriate people are not obvious in this paper.

It is uncovered that most of the invalid assumptions detected using AMS corresponded to the interpretation of requirements. The documentation of assumptions through AMS allowed detection of defects before the implementation phase. However, it will be pragmatic to discover the inconsistencies at the earlier phase, RE, rather than

during coding, and this will be the main issue or idea which I would like to bring into attention.

The urge for requirements monitoring has been suggested by Fickas et al [26] who initiated the importance of tracking and monitoring assumptions in software development. They manually established a linkage between requirements, assumptions and remedial action needed for the system if the assumptions were violated. I have adapted a similar concept to that used by the authors of the above paper, deriving assumptions from system requirements. However, unlike the work done by Fickas, I am not using the derived assumptions to generate specifications for monitoring assumptions. Alternatively, I have developed a checklist of assumptions for detecting the final magnitude of assumptions from a specifications document.

As continuation from [26], Cohen et al [27] illustrated a tool called automatic run-time monitoring of software systems', AMOS (Automatic Monitoring System). They extend the idea of resource monitoring, which is practiced by administrators of computer networks, to a broader class of requirements and assumptions made by the designers, purchasers, installers and users of software systems. AMOS allows users to enter assumptions or requirements as expression into FLEA (Formal Language for Expressing Assumptions). These expressions will be converted into run time monitoring code and will be triggered by a database if the triggering conditions' becomes true.

Some of the constructs provided by FLEA are illustrated in Figure 2.1. The scenarios below explain the usage of the codes in FLEA.

A developer enters an assumption, "The staff can assume that the monthly Project Progress meeting will not be held if they did not receive any emails or memos

regarding the meeting by first week of each month" into the Automatic Monitoring System (AMOS).

Then, the system detected that none of the staffs received memos or emails notifying the monthly Project Progress meeting.

However, the Project Progress meeting was scheduled on the last week of the month at 3pm in the system.

logical combination of events e.g., *eventA* OR *eventB*
sequences of events, e.g., *eventA* THEN *eventB*
counting and other simple statistical operations on events, e.g., COUNT [occurrences of] *eventA*
parameterized events, e.g., *eventA(x)*
time sensitive events, e.g., *eventA* WITHIN 60 seconds of *eventB*
threshold events, e.g., START count *eventA* > 10

FIGURE 2.1 – *Constructs used in FLEA*

(Taken from [27])

In the above example, the assumption was violated and this can be detected by using the code *logical combination of events e.g., eventA OR eventB.*

Though FLEA is used to record assumptions, it is a coded structured language which will not assist the development team to manually detect the assumptions made during the requirements gathering process. Thus, there are limitations for using FLEA to identify assumptions during the early stages of Requirements Analysis.

While developing a tool for managing system requirements, Han [35, 36] has described assumptions as constraints which a system has to respect or live with. He

categorized assumptions and included them as one of the components in an information model which he designed, the Tool for Requirements and Architecture Management (TRAM). In software development, system architecture acts as a base for a system's design. Thus, the traceability between system requirements and system architecture displays whether the architecture designs have satisfied the requirements of a system. TRAM is used as a tool for managing system requirements, system architecture and the traceability between them.

Assumptions management is also essential in large real-time systems consisting of tens or hundreds of software components exposing interfaces. In practice, assumptions can be encoded in a format which can be verified using programs. However, current software engineering practices do not provide a clear method to verify assumptions. Tirumala et al [21] presented Assumptions Management Framework (AMF) for capturing non-syntactical assumptions and requirements. The framework allows tracking dependencies based on assumptions made on software interfaces. AMF also allows capturing dependencies between modules that do not interact through the notion of property interfaces. The authors also believe that the manual process of tackling assumptions is error prone. Hence, AMF is developed to reduce a large proposition of testing time and the number of product defects by automatically tracking these dependencies. Further, assumptions during requirements verification process were used to rationalize the modelling activities for embedded system [37].

Classification of assumptions also plays an important part in the effort of managing assumptions. Taxonomy of assumptions will enable a systematic way of tackling issues related to assumptions. The development of taxonomy of assumptions in the area of requirements engineering is elaborated in Chapter Three. There are also

several researches done on taxonomies in software development [34, 36, 37, 38, 45]. The gaps discovered in the preceding taxonomies contribute to the development of the taxonomy of assumption. These gaps are further elaborated in Section 3.3 followed by the detail explanation of the taxonomies in software development.

Explorations were carried out on faults in the natural language used in development which assisted in extracting requirements faults [38]. This research is also related to assumptions studies.

In my research, I emphasize the importance of manual detection techniques of assumptions. However, automated tools that detect and record assumptions, such as AMOS, TRAM and AMF as illustrated above, are equally essential in the effort of tackling assumptions issues in development. These tools play an important role during the verification phase where some of the recorded assumptions can be verified and turned into requirements.

Several research programs which have discussed the RE aspects in software development have progressed in the past few years. These attempts were taken to develop tools and methods [21, 27, 34, 35] either to manage the existence of assumptions in SE or RE, or even to manage requirements effectively. However the methods that actually define, measure and manage these assumptions are still scarce. The potential benefit resulting from defining, measuring and managing assumptions may be tremendous as the earlier identification and elimination of defects. This also contributes to improved change analysis for more predictable and cost-effective software development. Hence, assumptions directed research and development activities is unquestionably a challenge but also an urgent need.

2.3 Coverage of Assumption Issues in Software Engineering Books

A short survey of software engineering and project management text books revealed a minimal coverage of assumptions management in Requirements Engineering. The issue of assumptions was not included in some text books [39, 40, 41]. Other authors only covered a minor area of assumptions management [16, 42, 43, 44].

2.4 Reviews on Interdisciplinary Work done on Assumptions

The studies of assumptions were reviewed in other fields apart from RE. I believe there is a need to explore assumptions in other disciplines, in order to better understand the similarity and differences about discovering assumptions in multidisciplinary areas. For instance, there is awareness of assumptions in Project Management area. A study of Assumptions Based Planning (ABP) is described by Dewar [45]. ABP is used to make clear distinctions among types of assumptions in order to focus on activities related to plans which have already been developed. The findings in other areas pertaining to assumptions are described in the following section.

2.4.1 Software Architecture

Issues pertaining to assumptions were discussed in software architecture [46, 47, 48]. The current practise of producing new applications by merging available parts of software has the possibility of facing issues with interoperability among the components. The problems were detected within the low level system interoperability. According to Garlan et al [46], this issue was caused by architectural mismatches which

depended on the assumptions made of a reusable part on the construction of the application. The root cause for most of the serious problems regarding an application originates when these assumptions were in conflict. Hence, the authors have described these mismatched assumptions from an architectural viewpoint. There were four categories of architectural mismatches identified during an application's system integration which are explained as below.

- Assumptions about the nature of the components.
- Assumptions about the nature of the connectors.
- Assumptions about the global architectural structure.
- Assumptions about the construction process.

We understand that assumptions were known regarding the risk associated with them in software architectural field. Hence, I suggest studies pertaining to assumptions in software development are important, particularly in Requirements Engineering.

2.4.2 Psychology and Sociology

Assumptions were explored broadly in Psychology and Sociology field [49] where the assumptions phenomenon was associated with feelings and behaviour of an individual. In Cognitive Therapy Techniques, assumptions were elicited using a series of questions, in the form of a dialog, to discover the problems faced by patients who had negative feelings towards themselves [50]. In addition, forms were designed and used for the purpose of assumptions elicitation. These forms allowed the patients to express their thoughts (which were believed to be assumptions) that created feelings and controlled the behaviour of an individual. I understand that assumptions dictate our

daily life by influencing our actions and decision. There might be incidence where wrong assumptions lead to incorrect decisions which we might regret in the future. Therefore, it is not abnormal for assumptions to create problems in the area of my research if the assumptions are violated.

2.5 Conclusion of the Topic

Research carried out regarding assumptions management mainly describes the importance of recording assumptions that are made during the development process [31, 34]. Efforts taken to develop tools to manage assumptions effectively are based on extracting, recording and monitoring the assumptions [21, 26, 27, 34, 35].

Since we know the existence of assumptions is a form of risk which might propagate defects in the development process, efforts must be taken to detect and confirm the correctness of these assumptions as early as possible. We suggest that these efforts should be taken from RE, the very first stage of software development. In comparison with assumptions correction at other stages of software development, there is a tremendous saving of cost, time, effort, and also an increase in the users' satisfaction level. Tackling assumptions can be a challenging task. Hence, it is necessary to have a systematic approach to classify assumptions in order to efficiently tackle the issues and consequences related to them. Therefore in the next chapter, definition of assumptions in the context of RE and development of a taxonomy of assumptions will be presented.

Chapter 3 Methodology

Assumptions are associated with risks when they are not clarified and documented. Though it is not trivial to solve issues originating from assumptions in development, I have taken the effort to minimize the risk by exploring the past research on assumptions.

There are automated solutions available to tackle issues concerning assumptions as described in Section 2.2. However, I suggest that identifying assumptions should be tracked manually as well as by using automated tools. I argue such since assumptions are a special source of defects that are caused when the assumptions are violated. Therefore, I strongly believe that a manual method of identifying assumptions will be an advantage, especially in the initial stages of development. These manual techniques will support the discovery of assumptions by understanding the frequent source of assumptions during early development process.

I claim that equal effort is required both to deal with assumptions and to realize the issues related with their causes. This leads to my primary aim in Chapter Three: – to develop concepts for handling assumptions made during the early stages of software development. This chapter defines assumptions from a Software Engineering (SE) perspective and builds a taxonomy of assumptions made during RE.

3.1 The Workflow Model

Assumptions made during software development have been identified as a potential issue if they are not well defined, validated and documented [18, 19, 20].

Hence, assumptions must be tackled at the initial phase of development process, RE. A workflow model called 'Exploration of Assumptions in Requirements Engineering' (EAiRE) was developed as illustrated in Figure 3.1 to guide my research.

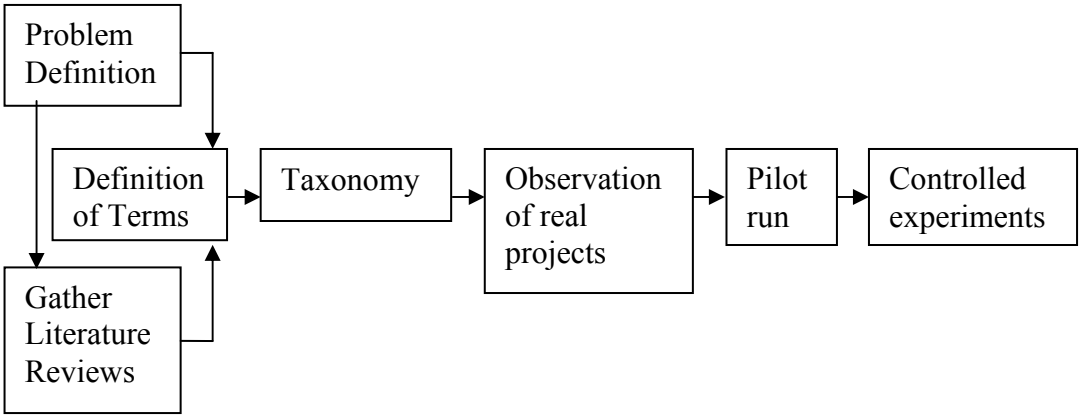


FIGURE 3.1 – *A Workflow Model*

Why do we need to study issue of assumptions in SE? This question is answered by my first task in the workflow model which outlines the problem statement and research hypothesis which have been illustrated in Chapter One. Next, I researched and studied the past histories and case studies related to assumptions in Software Engineering as discussed in the previous chapter. These past research allows to determine the importance of work done related to assumptions and allows to look at the gaps exist in the research. Simultaneously, I derived a series of definitions, clarifying assumptions as is essential for this study.

I understand that there are many kinds of assumptions related to the development phase of SE. It is important to assess these groups according to the various aspects and impacts of assumptions. There is a need for taxonomy of assumptions. I gathered several alternatives classifications of assumptions from the literature. However, there are gaps in these taxonomies which urged me to move to my next step which was to build a superset for classifications of assumptions. Though, my literature

search includes multiple phases of development projects, I narrowed down my research to focus on classification of assumptions during RE. Subsequently, these efforts lead to the development of a 'Taxonomy of Assumptions made during Requirements Engineering' (TARE). I emphasize that it is important to make clear distinctions among the different categories of assumptions in RE. With clear distinction it is possible to understand the behaviour and impact of the assumptions in software projects.

I observed how assumptions made during the requirements gathering process influenced the documentation of the requirements as well as the overall development project. Controlled experiments were designed and a pilot run was carried out to test the validity of the experiments. This leads to the execution of these controlled experiments in classroom based environment to study further the issue of assumptions in requirements development.

3.2 The Theory of Assumptions

Past research on assumptions in development projects as described in Chapter Two explains the necessity of exploring this issue in the software industries. Although the previous surveys involved assumptions management during SE, I claim that assumptions should be attacked at the initial stages of development. In this section, I will define and describe the nature of assumptions in the context of SE.

3.2.1 Definition of Assumptions

In some software development projects, the word “assumption” is taken to mean the conditions under which the system is guaranteed to operate correctly, commonly known as constraints [36]. However, according to IEEE Guide for Developing System Requirements Specifications [51] and IEEE Standard for Software Life Cycle Processes–Risk Management [52], constraints and assumptions are classified as separate entities. Although the definitions of assumptions have been included in these papers [51, 52], there are no specific guidelines for handling them during the RE stage.

Assumption is a common word that is frequently encountered in our daily life; yet it can be defined from various aspects as below:

- Definition 1 – In a layman’s terminology, an assumption is defined as ‘what we believe we know and accept as the truth without proofs’ [49].
- Definition 2 – Assumptions are defined as the conscious and unconscious invention of information that affects our actions or behaviours [53].
- Definition 3 – In the context of software engineering (SE), assumptions are described as an existing behaviour within the system’s environment that is unchanged by the proposed system [16, 42]. Therefore, the assumptions act as elements which belong to the system and will not cause any changes to the system until the assumptions are tested.
- Definition 4 – In real-time systems, assumptions are defined as formally unexpressed yet necessary conditions which are required for the components in embedded and real-time systems to function correctly [21].

- Definition 5 – In project management's framework, assumptions are defined as factors which are considered to be true, real, or certain for planning purposes [54, 55].
- Definition 6 – In the area of Assumptions Based Planning (ABP), a [planning] assumption is described as judgements or evaluation about some characteristic of the future which underlies the plans of an organization [45].

As a whole, assumptions represent knowledge created by humans without discussing the correctness of the knowledge. Hence they can be true or false. For the purpose of our research, we define assumptions in terms as below.

> *In the context of Requirements Engineering, assumptions are defined as unwritten needs or decisions which have been made but have not been validated.*

Assumptions in software projects can be treated as an issue where in the resolution has not been explicitly discussed, agreed, validated and documented. Thus, there exists a risk that stakeholders have different understandings of the necessary actions required for that issue. As such, assumptions can be looked at as potential defects.

3.2.2 Nature of Assumptions

Whenever stakeholders rely on any decision based on an assumption, they are actually taking a risk of making either a valid or invalid judgement. As depicted in Figure 3.2, at any point of time during development, implementation of a valid assumption does not change the correctness of the software (does not return defects) unless their validity changes as the software requirements evolves [18, 28, 29].

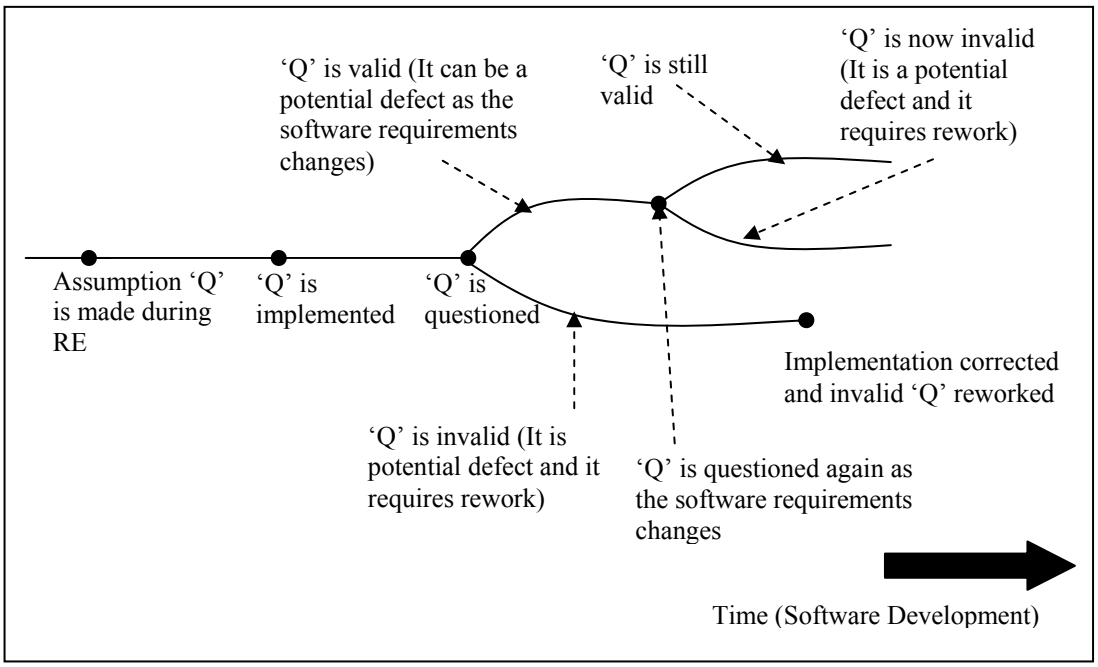


FIGURE 3.2 – *Valid and Invalid Assumptions in Software Development*

Conversely, if implementation of an invalid assumption is being challenged, it may return either a defect and require rework or contribute to defect formation. However, both valid and invalid assumptions need to be tested since they are potential sources of defects. Thus, revisiting them frequently is essential as the process of detecting and analysing those works in iterative manner.

We also define the validity of assumptions below:

- *Valid assumptions will not influence the correctness of the system until their validity changes over the evolution of that system.*
- *Invalid assumptions are defects if the correctness of the system is influenced by them.*

The impact of accrued invalid assumptions can lead to extra cost, time and effort for resolving them in the later stages of development cycle. Furthermore, when a chain of assumptions interacts, the impact is greater and may require significant rework. Some

of the observations, as illustrated by Lehman [28], have revealed the impacts of assumptions which led to disastrous failures (e.g. case studies of Ariane V, CERN Accelerator, Sinking of HMS Sheffield and London Ambulance System).

Assumptions also affect the actions or behaviour of stakeholders. They represent one of the factors which influence how a software project is planned and executed by its stakeholders. Thus, research related to issues arising from assumptions in RE is important. In EAiRE, I will be working on identification and resolution of assumptions during the RE phase. An initial foundation for my research is a taxonomy to enable me to systematically attack the assumptions problem.

3.3 Alternatives Taxonomies in Software Development

There were several studies conducted in the area of assumptions in software development in the past which discussed taxonomies of assumptions, pertaining to their areas of explorations as depicted in Table 3.1. I believe that there are gaps in the previous work done on the taxonomy of assumptions as described in the following sections.

TABLE 3.1 – Diversity among Taxonomies of Assumptions

Name of the Taxonomy	Descriptions	Author(s)	Year
Faults in natural language	Used to extract requirements faults	Lanubile, Shull and Basili [38]	1998
Types of assumptions	Used to manage assumptions extracted from source code	Lewis, Mahatham and Wrage [34]	2004
Assumptions in Assumptions Based Planning	Used to make clear distinctions among types of assumptions in order to focus on activities related to plans which already have been developed	Dewar [45]	2002
Assumptions in development of Tool for Requirements and Architecture Management	Used to record risk associated with the assumptions	Han [36]	2001
Assumptions for Embedded Systems during Requirements Verification	Used to rationalize the modelling activities for embedded system	Marincic, Mader and Wieringa [37]	2008

3.3.1 The Taxonomy of Faults in Natural Language Requirements

Perspective-Based Reading (PBR) is a method which uses a set of rules to detect faults in requirements documents [56]. In order to extract the requirements' errors, a taxonomy of faults in natural language requirements were used [38]. The relevant classification from this taxonomy were matched with the categories developed in our study called Taxonomy of Assumptions and shown in the table below.

TABLE 3.2 – A Taxonomy of Faults in Natural Language Requirements

(The numbers in the third column of the table corresponds to the eventual Taxonomy of Assumptions made during Requirements Engineering in Figure 3.6)

Classifications	Descriptions	Relevancy with TARE
Missing Information	Referring to a group of important requirements which are either missing or not defined	Yes (Covered in Dimension 4.1)
Ambiguous Information	Referring to requirements which can be explained in more than one way due to the usage of multiple words for describing the same software feature, or even multiple meanings of a word in a specific environment	Yes (Covered in Dimension 2.1, 2.2, 2.3, 4.4 and 4.5)
Inconsistent Information	Referring to situation when there are contradiction between two or more requirements	Not covered
Incorrect Fact	Referring to requirements which include information, which will not be true under some particular circumstances for the system	Not covered
Extraneous Information	Referring to information which are not required or utilized but has been included for developing the software	Yes (Covered in Dimension 4.1)
Miscellaneous	Referring to a group of requirements which are organised incorrectly, such as requirements which are placed in the wrong section	Not covered

For instance, there are possibilities for defects caused by 'Missing Information' and 'Extraneous Information' categories being contributed by assumptions which originate from 'Dimension 4.1: Lack of Domain Knowledge'. The term 'Dimension' refers to the classifications depicted in the final Taxonomy of Assumptions made in Requirements Engineering (TARE) defined in the next section.

Though the taxonomy used in PBR method helps to better understand the categories of assumptions, it focuses solely on the usage of language to categorize the

assumptions. This type of assumption will enable developers to group assumptions detected mainly from requirements documents and might not be able to extend the usage to study other aspects and impacts of assumptions during RE process.

3.3.2 Types of Assumptions

As mentioned in Chapter 2, Lewis et al [34] designed a prototype tool called Assumptions Management System (AMS) which beneficially manage assumptions and are expected to improve software quality. AMS records and extracts assumptions from Java source code into a repository and manages these assumptions in a web-based format. These assumptions were characterized into several types as below:

- Control Assumptions, captures control flow in a program.
- Environment Assumptions, records the expected behavior of an environment in which the application will be operating.
- Data Assumptions, depicts the expectation of the input or output data used in the program.
- Convention Assumptions, captures the conventions used by the developer during the program construction.
- Usage Assumptions, shows the expected way of using the application.

The categories above describe the taxonomy used by the authors to group the assumptions gathered from the program code. This research specifically focused on the development of a classification of assumptions during coding phase. Though the taxonomy assists the developers in the debugging process, I claim that the cost for debugging defects derived by assumptions at coding phase is higher than tackling them at the beginning of development. Thus this taxonomy will be less helpful to understand the

root causes and impacts of assumptions made from the early stages of software development. Therefore there is a need to design a taxonomy which will enable us to explore assumptions issues during the Requirements Engineering Process.

3.3.3 A Taxonomy of Assumptions in Assumptions Based Planning

Assumptions Based Planning (ABP) is a planning tool developed by Dewar [45] for the United States army. ABP was used during military post-planning and concentrated on activities pertaining to '*assumptions in an already-developed plan*'. The previous activities are considered to be the most important and uncertain part for a plan.

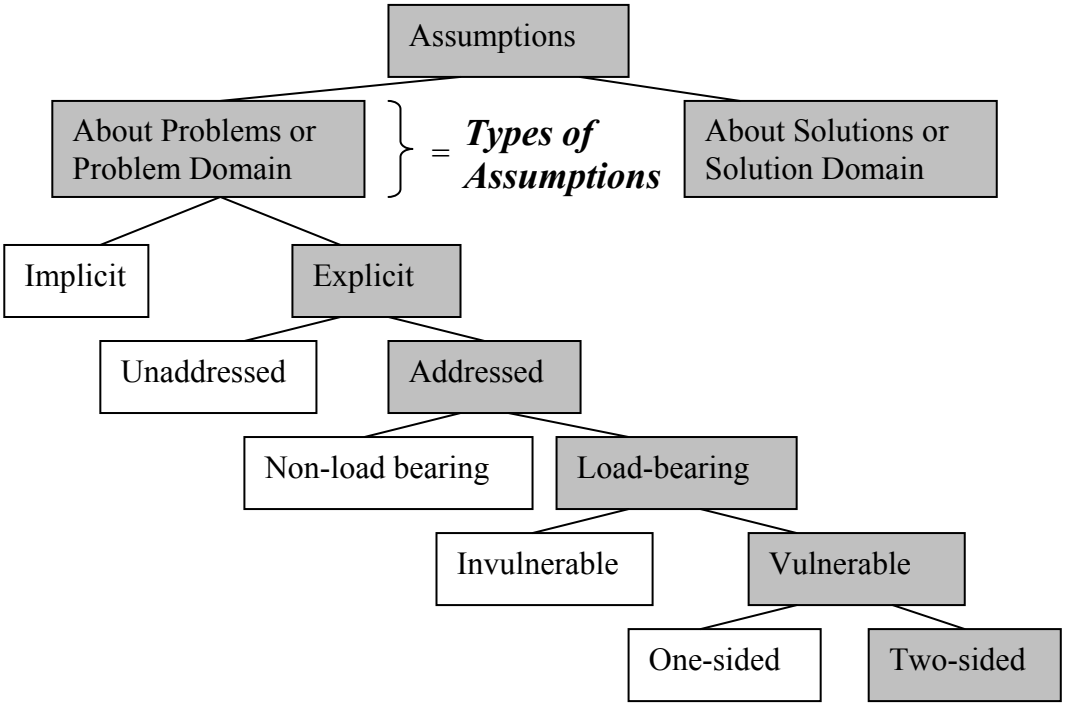


FIGURE 3.3 – *Assumptions Based Planning: Taxonomic Tree of Assumptions (Taken from [45])*

The goal of ABP is to identify types of assumptions in order to make clear distinctions among them. The taxonomy of assumptions for ABP is depicted in Figure

3.3, highlighting the main branches of the tree. ABP classifies assumptions into two domains – About Problems and About Solutions.

The primary focus is the 'Problem Domain' compared to the 'Solution Domain' because in ABP, the planners are aware of the correct actions to handle assumptions they make for solutions. 'About Problems' is similar to 'Types of Assumptions' in the taxonomy developed in our research, TARE. The descriptions of the main branches of the taxonomy tree are explained below:

- Explicit Assumptions

  Those assumptions which have been discovered and documented are known as Explicit Assumptions. In TARE, Explicit Assumptions are known as Conscious Assumptions [45].

- Addressed Assumptions

  These refer to assumptions which have been validated and documented [45].

- Load-bearing Assumptions

  If a particular assumption's failure requires significant changes in the organization's plan, then, it is classified as a Load-bearing Assumption. In TARE invalid assumptions are similar to the Load-bearing Assumptions in ABP [45].

- Vulnerable Assumptions

  These assumptions are closely related to the lifetime of a particular plan. An assumption is vulnerable if it fails within the plan's lifetime due to any possible event [45].

- Two-sided Assumptions

  During an organization's plan lifetime, these types of assumptions are vulnerable to failure in both directions, either being favourable or unfavourable towards the organization [45].

The taxonomy developed for ABP specifically addresses the types of assumptions made during planning stage. Despite the difference in the area which we focused in our research, the taxonomy which we developed, TARE can be considered as the superset for the taxonomy shown in Figure 3.3.

3.3.4 Tool for Requirements and Architecture Management

While developing a Tool for Requirements and Architecture Management (TRAM), Han [35] had classified assumptions according to their nature including system interaction, user interaction, system resources, and standards and regulations. Further, he had categorized the authorities for asserting assumptions which included management, domain experts, some stakeholders, and standards and documentation used during system development.

We have merged and illustrated his observations in Figure 3.4. The taxonomy found in Han's literature was used to associate risk with assumptions discovered. However the detection and recording of assumption was by using an automated tool, TRAM. Therefore he did not need to elaborate his classification to the level we require for our study, EAiRE.

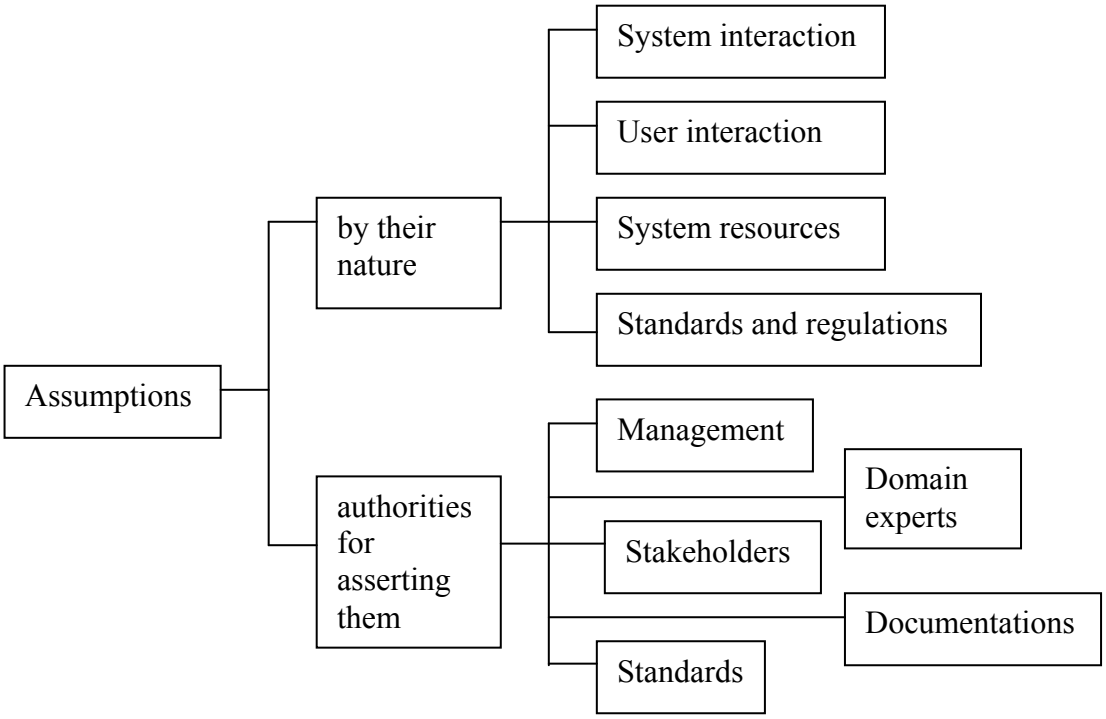


FIGURE 3.4 – *Classification of Assumptions*

3.3.5 Classification of Assumptions

Marincic et al [37] had carried out an investigation for improving the modelling process of embedded system during formal verification. They had documented some of the modelling decisions in the form of a list of the system assumptions made during modelling. The previous list derives a classification of assumptions for an embedded system which was made during the Requirements Verification process. The goal of this taxonomy was to rationalize the modelling activity itself. The taxonomy is presented in Table 3.3.

TABLE 3.3 – Classification of Assumptions for Embedded Systems

(The numbers in the fourth column of the table corresponds to the eventual Taxonomy of Assumptions made during Requirements Engineering in Figure 3.6)

Classifications	Name of the Classifications	Descriptions	Relevancy with TARE
C1	Assumptions about components	Referring to the descriptions of assumptions for the characteristics of the system	Yes (Covered in Dimension 1.1)
C2	Assumptions about aspects		Yes (Covered in Dimension 4.1)
C3	Necessary and contingent assumptions	Referring to the changeability criteria of the system	Not covered
C4			
C5	Plant and environment constraints	Referring to the upfront constraints of the system which are relevance to the system users but not part of the model for the system	Not covered

There are five classes of assumptions – C1, C2, C3, C4 and C5 in the taxonomy developed by Marincic [37]. These classes are matched with TARE wherever there are possible similarities. C1 and C2 refer to the assumptions pertaining to the descriptions of the system. As such, C1 and C2 are matched with 'Dimension 1.1: Functional' and 'Dimension 4.1: Lack of Domain Knowledge' respectively. 'Dimension' refers to the classifications depicted in Taxonomy of Assumptions (TARE) in Figure 3.6. C3 and C4 describe the essential assumptions which are needed for the changeability aspects of the system. On the other hand, C5 refers to the system constraints that are listed upfront for system users. I have a different viewpoint in EAiRE, about the concept used in C5, whereby assumptions are defined as a separate entity from constraints and assertions. However, assumptions can produce constraints or assertions upon verification. Hence we did not group constraints as one of the categories in our taxonomy, TARE. The classifications indicated in Table 3.3 are useful for grouping the assumptions. However

I believe this taxonomy is developed specifically for embedded systems. I require a broader range for grouping assumptions.

3.3.6 Several other Classifications

Durrenberger [53] defines an assumption in projects as – '*the conscious or unconscious invention of information that affects our actions or behaviors'*. His definition contributes for my effort of categorizing assumptions. In addition, he had also emphasized that knowledge is related to assumptions by claiming that knowledge is equivalent to assumptions. However I understand that though knowledge is an essential element which contributes to an assumption, but both knowledge and assumptions are not similar.

In the process of developing TARE, I have gathered several taxonomies, requirements [42, 57, 58] and levels of requirements [16], besides simply observing classification of assumptions. I expanded the search on taxonomies pertaining to requirements because I strongly claim that requirements and assumptions are closely inter-connected.

In software development, both requirements and assumptions are derived from the users' needs. For instance, in a library system, the need for a user to loan books develops the requirement for the user to be registered with the library system. At the same time, a similar need may also probably derive the assumption that the user can loan as many books as he or she wishes (if there is no constraint on the number of books that can be borrowed by the users). According to the system's point of view, requirements and assumptions are derived from users' needs. Even though both of these

terms originate from the same source, they represent discrete definitions; requirements are documented decisions of a system while assumptions are undocumented decisions of a system. Therefore we studied several taxonomies of requirements and classification for levels of requirements; this also partly contributed to the efforts for designing TARE.

3.4 A Taxonomy of Assumptions in Requirements Engineering

3.4.1 The Aims of the Taxonomy

In the process of developing the Taxonomy of Assumptions in Requirements Engineering (TARE), questions arise: (1) Why do we need classification of assumptions during RE? (2) Is it essential to design TARE? (3) What are the benefits of developing TARE? (4) How does TARE contribute to the current state-of-the-art knowledge about assumptions in software development?

Fortunately my past data gathered on explorations relating to assumptions, as explained in Section 3.3, demonstrates that there is a need for a broad classification of assumptions made specifically during RE phase. I emphasize that the first stage of software development, requirements gathering, will be the initial point where many assumptions creep in. Therefore the effort, of developing a classification of assumptions made during RE, is significant. Issues pertaining to the assumptions made at the early stages of software development will be supported by the use of a taxonomy.

> *The primary goal of TARE is to make clear distinctions for assumptions made during RE in order to choose the necessary actions to manage this issue.*

There are several benefits for classifying assumptions as described below:

- Clarification of the relationship between the requirements and the occurrence of the assumptions related to these requirements. For instance, if a high frequency of assumptions were derived from a particular requirement, then, there is a possibility that this requirement needs to be rephrased.
- Assessment of the categories of assumptions and the multidimensional interactions among them. For instance, the correlations between assumptions made during RE might have an implication down the track for the development process.
- Identification of root causes of assumptions made during RE and understanding of how they impact the overall development process. For instance, besides stakeholder, assumptions can also be initiated by other factors such as knowledge, documentations, data and others factors involved during development.
- Detection and recording of the frequency of defects derived from assumptions. For instance, if there is a particular category of assumption that contributes to high frequency of defects; this will raise awareness among the developers. They will then analyse and question this group of assumptions to minimize the occurrences of defects for future development activities.

3.4.2 Framework for the Taxonomy

A framework was developed to classify the assumptions made during RE, as illustrated in Figure 3.5. In order to simplify the process of deriving the taxonomy, I use the characteristics, which the assumption inherits as it was made. For instance, if an

assumption describes the functionality of a system, then this assumption is classified as 'Functional Assumptions'.

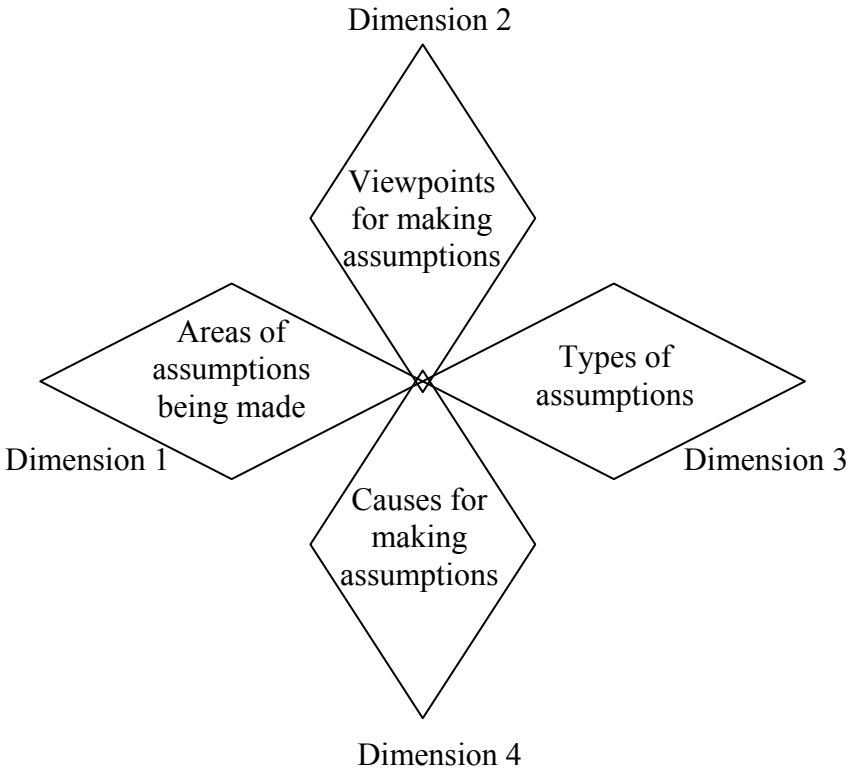


FIGURE 3.5 – *A Framework for Classification of Assumptions in Requirements*

Firstly, the assumptions were classified into four dimensions: (i) areas of assumptions being made; (ii) viewpoints for making assumptions; (iii) types of assumptions and; (iv) causes for making assumptions. Secondly, the interactions or relationships between these categories were assessed for impacts on the development of the software. This led to the investigation of further methods for tackling assumptions as each of these categories could be further subdivided, as described in the next section.

3.4.3 Explanation for the Taxonomy

The taxonomy of assumptions in RE, TARE is depicted in Figure 3.6. TARE

was developed after examining and extending the previous works conducted in [16, 34, 35, 37, 38, 42, 45, 53, 57, 58] in Section 3.3.

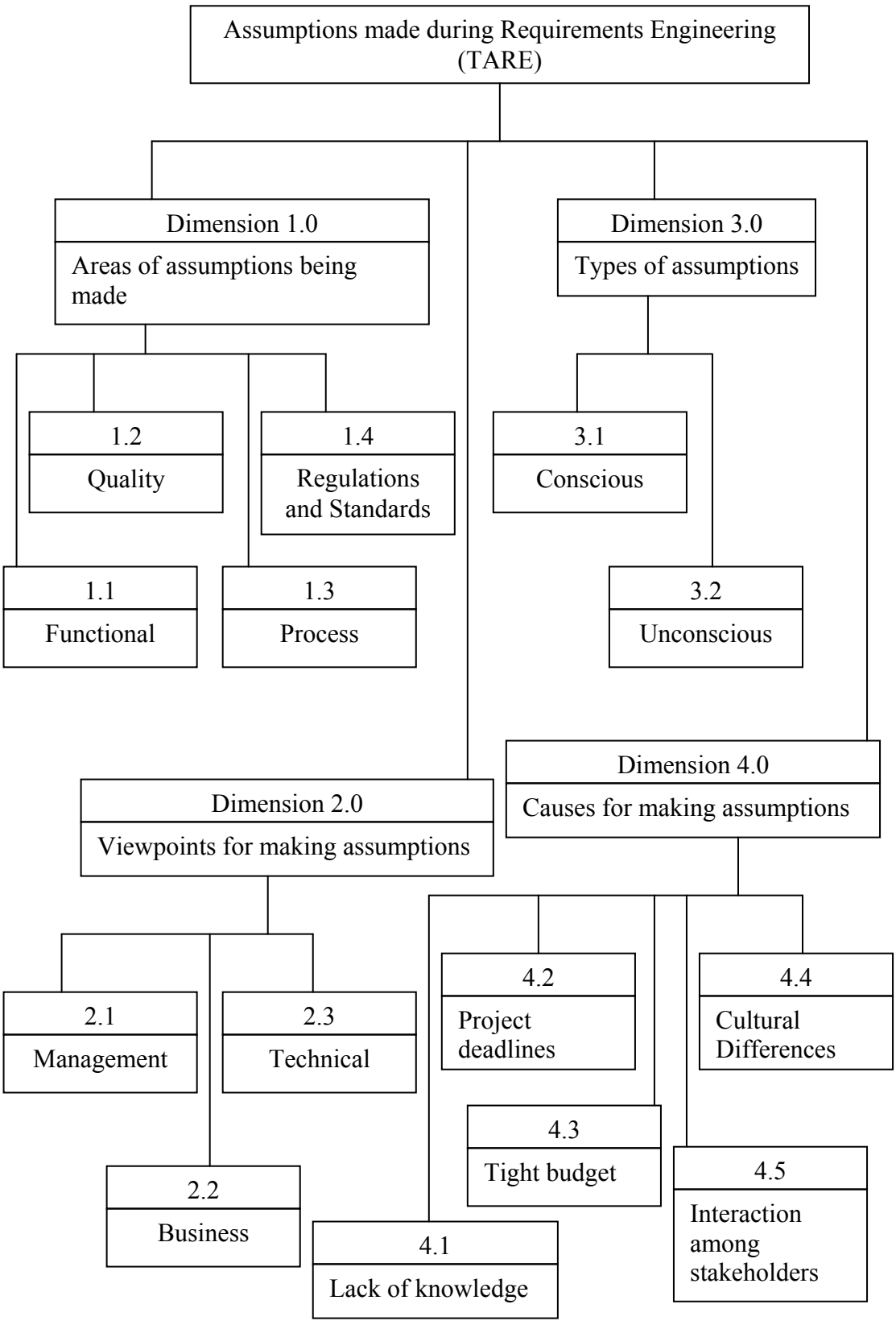


FIGURE 3.6 – *A Taxonomy of Assumptions made in Requirements Engineering*

Dimension 1.0: Areas of assumptions being made

The assumptions can be grouped according to 'Areas of assumptions being made', where assumptions were made pertaining to the system during the requirements capturing phase. 'Areas' is not limited to the subsets – 'Functional', 'Quality', 'Process' and 'Regulations and Standards' as depicted in Figure 3.6 since they vary according to the type of software being developed. Several requirements are depicted in Invisible Meeting Scheduler (IMS) [59] to illustrate examples of assumptions in 'Dimension 1.1: Functional' and 'Dimension 1.2: Quality'. IMS is a software tool that helps individuals to schedule meetings in an online calendar to display their schedules.

- 1.1: Functional Area

  The 'Functional' areas of software are referring to any requirements that specify a required behaviour of the software to be present. The following example illustrates assumptions in functional requirements.

  Example 1: If no rooms are vacant during the selected time, IMS shall choose the next feasible time [59].

  Assumptions: The system assumes that the users will be available for the meeting at the time allocated by the system. The users assume that the rooms are available during the time allocated by the system.

- 1.2: Quality Area

  Assumptions pertaining to this area specify the quality attributes of the software such as the degree of reliability, performance, dependability, security and safety. Unlike the 'Functional' area, assumptions made in 'Quality' are harder to identify, describe and measure. For instance, a client can assume that the new system which is being built will inherit qualities similar to or better than those of

the old system. Example 2 and Example 3 illustrate some of the assumptions derived from quality requirements.

Example 2: Users will be able to understand the layout and options of the IMS's user interface (UI) [59].

Assumption: A user can assume that other users will be able to understand the UI since he or she can understand it without realizing the understanding capacity differs from one user to another user.

Example 3: IMS will be implemented using modern programming practices that maximize the maintainability and reusability of the design and code [59].

Assumption: Users can assume that the design and code for Meeting Scheduling System are compatible with any new modern application (or even those) developed in the future.

- 1.3: Regarding Process

  Assumptions regarding software development processes are made when there is insufficient information provided about the processes. For example, assumptions may be made in the preparation of a Software Quality Assurance Plan (SQAP) on the need for a formal validation and verification in the software process [60].

- 1.4: Regulations and Standards

  Assumptions can also be made on standards, rules or laws (perhaps a safety consideration) which control the software development process.

Dimension 2.0: Viewpoints for Making Assumptions

Who are making the assumptions? The three primary categories for 'Viewpoints for Making Assumptions' are 'Management', 'Business' and 'Technical'. However, the

viewpoints category can vary, subject to the hierarchy of an organization. This dimension refers to the knowledge difference between the business and technical people of any problem domain. For instance, the groups use different languages to describe the same concept and this can lead to assumptions about requirements during the development.

- 2.1: Management Viewpoint

  In software development projects, the management's standpoint differs from the development team and other stakeholders. For instance, a scenario can exist where people who belong to the management group can become very influential for the development since they are the primary decision makers. This group of people includes Chief Executive Officer (CEO), general managers, financial managers and other dominant individuals.

- 2.2: Business Viewpoint

  The group of business people think in terms of the commercial benefits that the new system will promise. The assumptions which origins from this group are influenced by the thinking inherited from the business perspectives. Examples of people whom belong to this group are project manager, client and end users.

- 2.3: Technical Viewpoint

  Technical people think in term of the implementation details. It is hard to describe the technical terms in a manner which business people can understand unless they have technical knowledge. Those people who grouped under this category are development manager, system analyst, programmers and testers.

Dimension 3.0: Types of Assumptions

How are assumptions made? Though assumptions are decisions made without proofs, there are situations where we are not aware that we are making assumptions. Therefore, there are two mode of making assumptions which grouped in 'Types of Assumptions' described below:

- 3.1: Conscious Assumption

  Stakeholders are aware that they are making a decision which they believe to be correct. The motivation for 'Conscious' assumptions is often based on previous knowledge (assumptions are made using knowledge).

- 3.2: Unconscious Assumption

  Stakeholders are not aware that they are acting upon some assumptions. 'Unconscious' assumptions are driven by insufficient knowledge.

Dimension 4.0: Causes for Making Assumptions

The causes for stakeholders to assume are not limited to this taxonomy as human behaviours are very complex. We illustrate a subset of them below:

- 4.1: Lack of knowledge

  In development, when decisions are made due to lack of knowledge, there are high probabilities that assumptions will occur. Assumptions originates from this category can be further subdivided to lack of problem and domain knowledge.

- 4.2: Project deadlines

  When there are any delayed tasks affecting progress with development or there are tight deadlines, then there are increased chances of assumptions being made to speed up development. For an example, if a project deadline is due and there are still large number of task needs to be finished, then there is a likelihood that the number of assumptions made by the team members will increase due to the urgency of completing the project.

- 4.3: Tight budget

  As with problems related to time deadlines, if budget becomes a constraint in development, there is a probability for assumptions to be made in order to save cost. For instance, if a project's constraint is cost, then there are chances for the team members to make assumptions in order to conclude the project within the allocated budget.

- 4.4: Cultural differences

  Assumptions derived from cultural differences are particularly applicable for multi-site software development. One of the lessons learned from multi-site development is the existence of cultural gap. In this case, there are potentials for the stakeholders to assume, that the work processes in their own country are similar to other countries. For instance, end users from a country where English is not their first language may assume that the user interfaces for the software that is being developed for their use will be in their local language. Whereas, the software development team (an international service company) may assume that the end users can understand the user interfaces programmed in English [61].

- 4.5: Interactions among stakeholders

  Assumptions are made when there is unclear communication among stakeholders. For instance, if clients fail to describe their requirements to the developer or if the developers are poor at verifying and confirming specification, then, there are chances for them to assume that the software which is being developed will fulfil their needs. Such a description on communication is illustrated in Figure 1.2.

3.5 Contributions of the Taxonomy

There are several differences between TARE and other taxonomies described in Section 3.3. TARE expands on the knowledge gained from other classifications as described below:

- The primary difference is that TARE focuses on Requirements Engineering aspects of Software Development. The identification and correction of issues of assumptions can be made at an early stage of development, thus saving the cost and time of rectification at later phases.
- TARE builds on the combination of multiple classifications; assumptions and requirements, gathered from the previous research, enable me to develop a superset of taxonomy for assumptions. Hence there are additional categories of assumptions featured in TARE compared to other taxonomies.
- I did not narrow down the designing of TARE for a specific type of software, embedded or non-embedded system. TARE encompasses a broad range of aspects pertaining to assumptions in development process without emphasizing the type of the software.

- The categories in TARE are grouped in multiple dimensional categories which allow the user to study the interaction between them. As such, TARE allows one to understand the nature of assumptions and to detect defects derived from invalid assumptions.

3.6 Conclusion of the Topic

Assumptions made during a project, inherit multiple aspects of development. The classification of assumptions assists the efforts used to correct errors that have been caused by these same assumptions. Since defects originated from RE are the most costly to rework, the taxonomy for assumptions, made during RE, TARE has been developed to be an additional piece of information, which assists in the effort of designing methods to detect assumptions. The next chapter discusses in detail the design of experiments to identify assumptions.

Chapter 4 The Experimental Design

There are several approaches being used as a guidance to carry out research in Software Engineering (SE). A short survey investigated the various methods currently used by researchers. The results show three main categories of research methods in SE as illustrated in Figure 4.1. In this section, the plan is to elaborate only several types of research method because it is not one of the main element that contributing to the conclusion of this project.

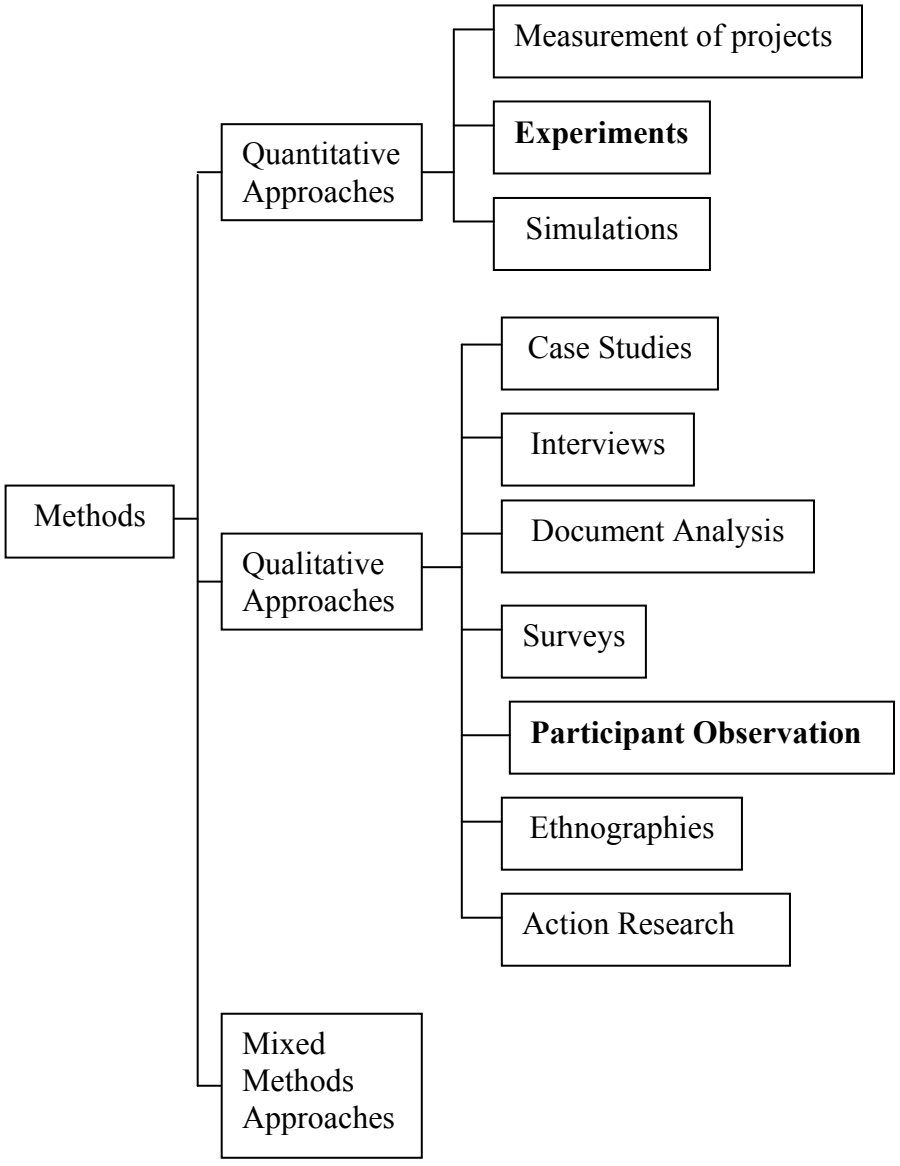


FIGURE 4.1– *Research methods in Software Engineering*

(derived from [62, 63, 64, 65 and 66])

Quantitative research aims to achieve a numerical relationship between several variables which are being examined. Hence, it allows elimination of irrelevant variables by discovering the dependent and independent variables in order to reduce the complexity of the research problem [63, 64]. This enables the research hypothesis to be verified. The data gathered in quantitative approaches are in numerical form. The collected data are usually analysed by statistical methods.

The approach is generally used to study human behaviour factor is known as qualitative research. This method uses detailed observation and involvement of the researcher in the studies undertaken. The non-numeric data which is gathered from this technique is in the form of text, graphics and or even images. The data gathered from the qualitative method is analysed by using techniques such as auditing, coding, constant comparison method, cross case analysis and member checking [62].

As adopted in other disciplines, the qualitative techniques have been largely used in SE research areas. One of the frequently used methods is ethnography. Ethnography is a strategy used to study and collect information about human behaviour in a society or organisation. An ethnographer acts as an observer and studies the environment chosen for the research. For instance, a System Analyst may take the role of a librarian and immerse himself or herself into a library system thus observing the routine work undertaken by the stakeholders of the library. However, in most scenarios, the behaviour of the participants who are being observed is modified due to the presence of the observer.

Another method, which belongs to the qualitative approach, is known as Action Research. This technique is used frequently in research topics in Social Science field to

understand and take actions for an issue being investigated. In this approach, steps such as planning, acting, observing and reflecting critical points are practised and so validated by the collected and evaluated data for the study [66].

The mixed method approach consists of the combination of the qualitative and quantitative method. In this technique, a wider range of coverage towards the types of approaches used for the research can be achieved [64, 65]. Therefore, it enables researchers to collect multiple kinds of data which can be used to test their research hypotheses.

The mixed method approach was used to undertake the research topic. Below is a walkthrough of the type of approaches which has been chosen for this research. The initial stage of the research commenced with analysis of some case studies (one of the qualitative methods as listed in Figure 4.1) which were described in Chapter Two and was followed by another qualitative method, the participant observation technique, described in more detail in Section 4.2.1. Empirical studies, being essential to reveal the potential benefits gained from the research, were indicated that the quantitative approach would be the next stage of undertaking the project. Two experiments were designed to gather data for the research. To conduct a well-controlled quantitative experiment in real-world of Software Engineering, it is not a trivial task since there is significant cost in replicating the projects. Hence, the experiments were conducted in classroom environments.

4.1 Framework for Developing the Experiments

Figure 4.2 depicts the workflow model developed to drive our exploration in the design of the experiments. Studies to build a taxonomy of assumptions in requirements engineering were commenced by classification which assisted in recognition of the types of assumptions that would be encountered in the research.

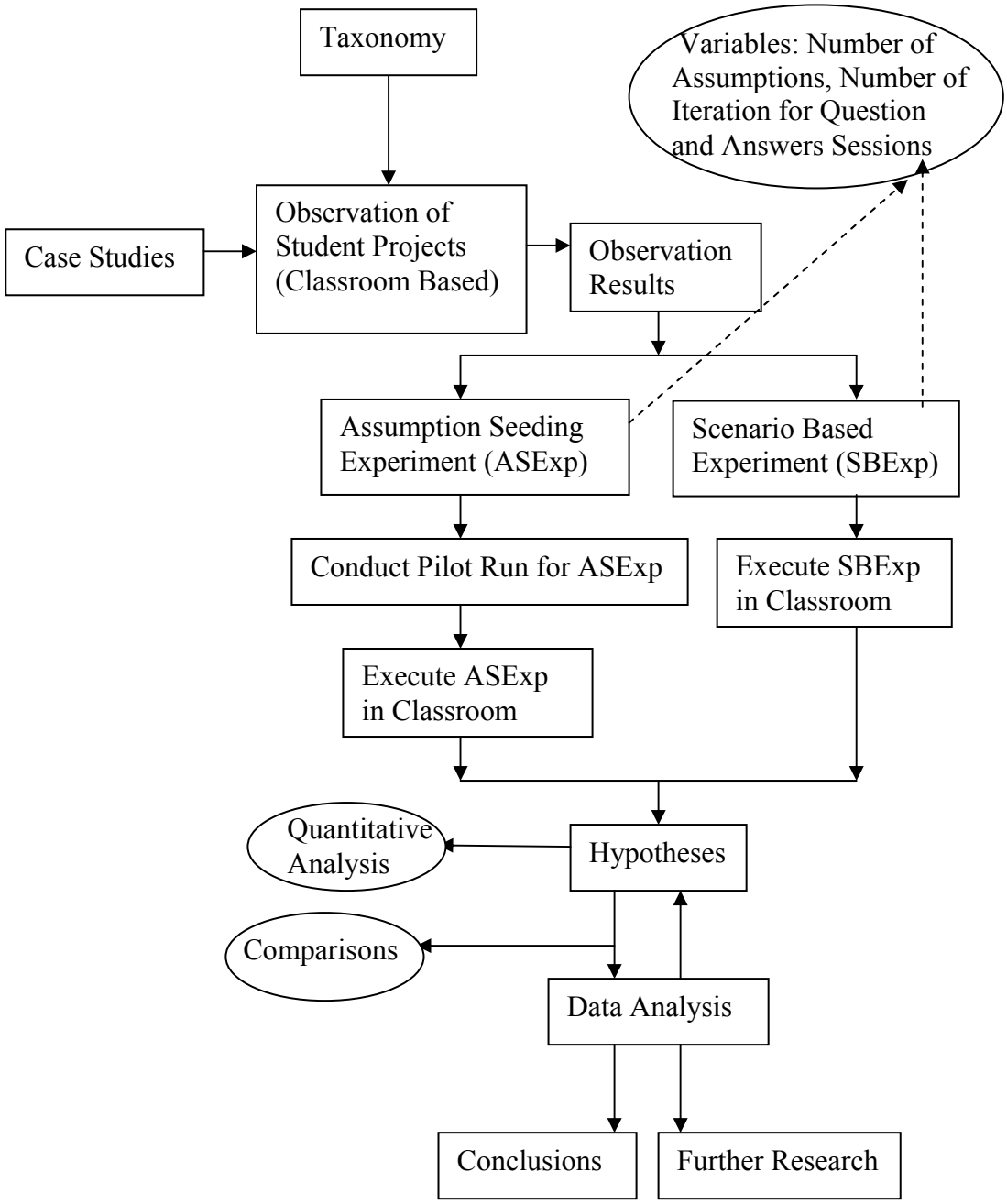


FIGURE 4.2 – *A Workflow Model for Designing Experiments*

The research effort was continued by conducting observation on students (as the subjects), involved in software development projects. The subjects are required to carry out a project as a part of their unit requirements for which they have enrolled. The outcomes of the ground work done in this research led to the next step, the design and execution of two classroom based experiments, Scenario Based and Assumptions Seeding, to discover assumptions in requirements documents.

As in social science disciplines, the human factor is also considered as the prominent factor in SE. Experimentation in SE fields can be conducted to explore the behaviour of people in their day-to-day situations; this behaviour influences the software development outcomes. Relationships between the variables under investigation were noted; for example, in a survey to determine reasons affecting the time needed in the requirement elicitation process, it was noted that one new technique needed a shorter time for this process. Data recorded verified this finding [67].

The techniques and tools required in designing the experiments were carefully analysed. During the designing process of the experiments, three items were defined as requirements for the implementation are described below:

- A sample Requirements Specification Documents was required. This document was extracted from a suitable source (without having any preference for a particular type of case study) from the literature. The chosen requirement specification document was then adapted to fit the experiment's goals.
- Participant was required to carry out the experiment. The characteristics of the participants were an important item for conducting the experiments. The decisions of choosing the participants had to be done carefully as the type of participant may influence the results of the experiment. The criterion that had to

be considered to decide the type of participants is scale of the experiment, anticipated outcomes and nature of the experiment.

- The data contributed by the participants was required to analyse the experiment. The experimental data were an important output for the research analysis. In addition, data needs to be retrieved, analysed and translated into a readable form. There are several forms to represent data such as text, pictures and numbers. At the end, conclusions were drawn by analysing the gathered data and matching them against the experiments' hypotheses.

The Workflow model for designing the experiment also considers some further exploration on assumptions during RE. Though this research topic involves eliciting assumptions which commonly is not a trivial task, the experimental designs were adapted to suit the research hypotheses.

4.1.1 Mine Drainage Control System

In this research, a Mine Drainage Control System (MDCS) was chosen from a case study to fit into our experiments' design. MDCS includes many features of a real-time embedded system and is commonly used as an example in numerous SE studies [68, 69, 70, 71, 72, 73]. The system concerns the software required to manage a simplified pump control system for a mining environment. It consists of a simple pump controller which is used to pump mine water from a sump at the bottom of the shaft to the surface. However, the pump should not be operated when the level of methane gas in the mine reaches a critical threshold. All the system events which occur during the mining process will be monitored via an operator console.

In general, the system is composed of four main subsystems: *PumpController*, *EnvironmentMonitor*, *OperatorConsole* and *DataLogger*. A simple schematic diagram illustrating the system is given in Figure 4.3. For the purpose of our experiment, we have extracted the system requirements of MDCS and designed a Requirements Specification Documents (RSD) as illustrated in the experiments documents (Appendix A-1 and Appendix B-1). The requirements specification is divided into two types: the functional requirements and the quality requirement.

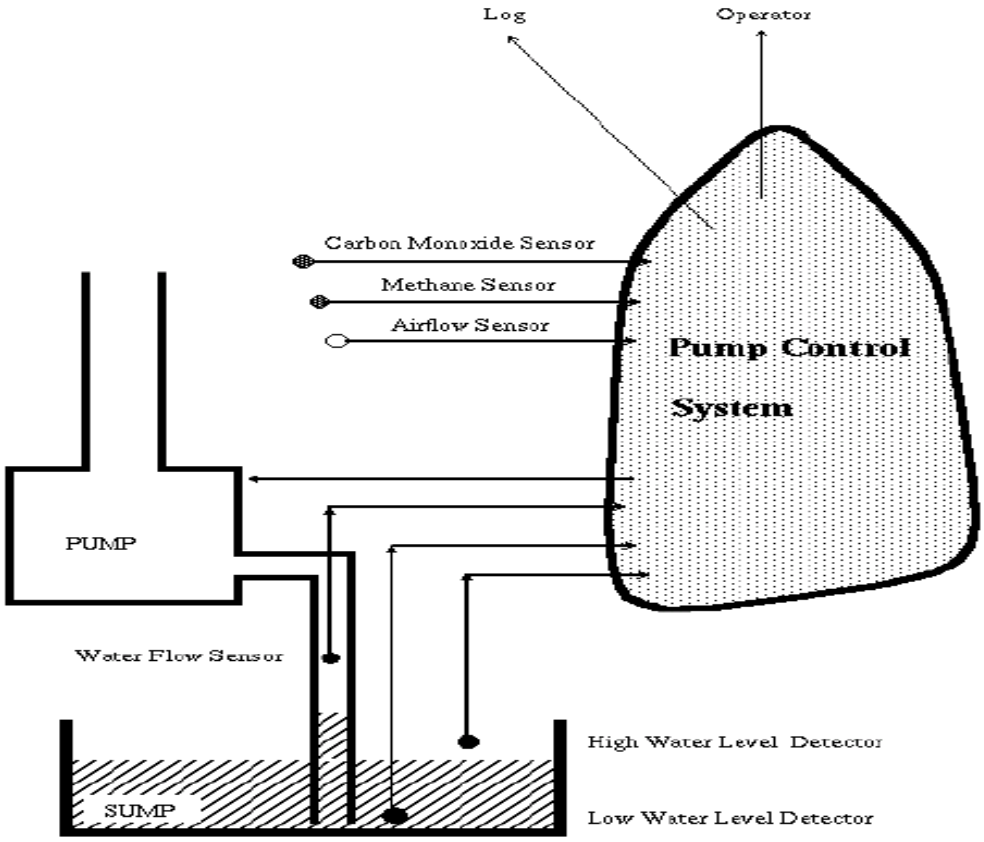


FIGURE 4.3 – *A schematic diagram for Mine Drainage Control System (Taken from Hooks [8])*

This system is chosen since it possesses characteristics which are suitable for the experimental design. The requirements illustrated for the system is brief and this was considered to be an essential feature which enabled development of a short RSD (two pages in length). The length of the requirements document was one of the deciding factors for selecting the MDCS because of the limited time available for conducting the

experiments. Since the time allocated to conduct the experiments depends on the tutorial sessions, with the duration of one hour, only brief requirements will be suitable for the experiments. In addition, the mining system was chosen, being a well understood common example in many literatures and, that enabled the gathering of sufficient information for the design of the experiments. The mining system is believed to be a non familiar system among the participant in our experiments.

In actual fact, the probability is higher to make decisions based on assumptions for a particular system which we are not familiar. This is supported by the fact that, it is human nature to use their imagination to seek answers for situations or issues which are new to them due to lack of experiences. The act of imagining forms thoughts and these thoughts are hypothesis or guesses which can be proven to be true or false. Thoughts activate the action of assuming, 'imagination at the end of the day is the road to assumptions' [74]. Therefore, the mining system was chosen for the experiment since provides room for the participants to presume the workflow and process of the system.

4.1.2 Use Case Diagrams

A Use Case Diagram (UCD) is used to display the overview of a system. From the viewpoint of an external observer, a UCD helps to describe what a system does rather than how it does it. Therefore it shows the system level functionality in a horizontal way by illustrating all the available functionality of a system rather than purely representing the details of the system's individual features. However, the UCD can not fully explain the interaction and the total system design details.

A system is built of many scenarios. Use Case Diagrams are closely related to scenarios. A scenario is an example of what happens when someone interacts with the system. A series of steps which explain the relationships between an actor and the system is called scenario. An actor initiates a use case. A use case defines a summary of scenarios for a particular task or event. The actors and use cases are connected via lines called 'communications'. An example of a simple Use Case Diagram consists of Actors, Use Cases and the Communication between both of them is illustrated in Figure 4.4.

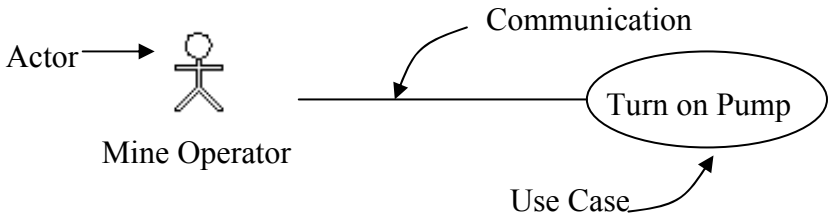


FIGURE 4.4 – *An example of a simple Use Case Diagram*

A UCD is helpful in several ways during system development. UCD were chosen as a tool or feature which can assist in the identification of assumptions related to requirements in this experiment. Use cases capture functional requirements of a system from the users' perspective. However, UCD and requirements are not similar articles and it can never serve as system requirements documents. UCD denotes the top-level behaviour which reflects on some of the requirements of the system. Furthermore, during the experiment, the participants are allowed to go beyond the requirements stated in the requirements specification and they may determine new requirements which they believe to be true. Their action can lead to the generation of assumptions. Hence, UCD were seen as an appropriate tool in the experiment where the participants needed to identify early mismatches between the requirements provided to them and assumptions which they make pertaining to the requirements.

The communication between Actors and Use Cases provides the basis for identifying the system's interactions. Hence, the participants are assisted in deciding how the system should work. Their understanding of the requirements provided to them is tested. Further, this determines whether the participants clearly understand the requirements or, if unclear, that they then translate requirements into assumptions.

Besides UCD, a Sequence Diagram (SD) is also widely used in requirements modelling processes. The SD can be used for the conversion of requirements expressed as use cases into meaningful system documents by verifying and refining them. Hence, SD helps in the requirements verification processes. The SD also can be used for assumptions verification processes. However, in the experiments the scope of this research to use the UCD for requirements clarifications is deliberately limited; there may be missing actors, use cases and communications. Further, the UCD can be used as a quality measure for capturing requirements for a system [75]. Therefore, it is believed that by using use case diagram in the experiment, some further details about the participants' understanding about the mining system will be gathered.

4.1.3 Research Participants

The subjects chosen as participants for this research studies were students taking computer science and engineering courses at The University of Western Australia (UWA) who enrolled in Professional Computing (CITS3220), Software Quality and Measurement (CITS4220), and Software Processes (CITS8220). The preceding units were chosen for this research project because the students enrolled in these units were believed to be equipped with relevant knowledge that required in participating in the observations and experiments designed to study the issue of assumptions. The

Professional Computing (CITS3220) unit was taught during the second semester in the teaching year which was July 2007 until November 2007. The number of students enrolled for this unit was thirty one. Both the other units are Software Engineering units which were taught during the first semester in the teaching year which was March 2008 until June 2008. Approximately sixty students were enrolled in both these units. All participants were at least eighteen years old. The following section discusses briefly the topics and content of the units which the participants were enrolled.

Professional Computing is a third year unit offered to undergraduate students. Students enrolled in this unit are required to undertake project work. Some of the unit aims were to enable students to gain experience using professional practices by working in a team environment to replicate industrial system in developing software. The majority of time assigned for this unit is for work on the project. Thus, there is no allocated lab or tutorial sessions for this unit. However, there are several compulsory lectures on software development practices. Hence, the students have been equipped with knowledge of Requirements Engineering. This unit was chosen to conduct the observation on students' projects.

The Software Quality Measurement unit is offered to both honours and post graduate students. The prerequisite to enrol in this unit is to have a fundamental knowledge about software development. Hence, students enrolled in CITS4220 were equipped with the knowledge of how the Requirements Engineering process works with regards to analysis of requirements documents. One of the objectives of this unit is to prepare students to understand software development as a process. Further, this unit exposes them to a broad range of issues pertaining to software development and demonstrates the critical problem solving skills needed to address these issues. The unit

was composed of lectures and tutorial or lab sessions which allows students to utilize their tutorial session for conducting the experiments.

The Software Processes unit emphases the underlying principles of software processes, their analysis, measurement and improvement and is only for post graduate students. The precondition to enroll in this unit is CITS4220. CITS4220 covers topics such as software process components, development life cycles, processes modeling, process methodologies, measuring and assessing process methodologies, the meta-process and methodology verification. One of the expectations for the students who are enrolled in CITS4220 is to acquire understanding about all aspects of software development life cycles and processes. As a result, they have been briefed about the requirement engineering processes and issues related to this topic. The nature of the unit permits the use of students enrolled in this unit as the participant for our experiment; the unit was composed of lectures and tutorials.

Convenience sampling method is used to determine the participants of the experiments. The nature of ease of use and low cost involved in conducting the project made convenience sampling as the preferred method for this research. Students were used as the participant to conduct the experiments. The logic behind choosing students as the subjects for the experiment is, the belief that the data gathered from students will be a reliable source for our initial investigation about assumptions in requirements engineering. Port and Klappholz [76] argue that the data acquired from classroom-based experiments are honest in the sense that there is little evidence towards biased outcomes. In fact, the SE units offered in universities can serve as laboratories to perform experiments which indeed will be beneficial to both the academics and industrial SE associates. Furthermore, the outcomes gained from these experiments can

impart knowledge to students in later parts of the course as well in their undertakings in industries after graduation. The attempt of conducting experiments in classrooms also motivates students to take interest in SE topics as they can foresee the potential room for research topics and, hence, motivates the research culture among the undergraduate students.

A survey reported that out of 5453 scientific articles on Software Engineering published from 1993 to 2002, 103 articles were controlled experiments in which one or more software development tasks performed by individuals or teams [77]. One of the criteria highlighted in this article for conducting experiments in software engineering studies was the necessity to explore the relevance to industry, of the experiment. Experiments which are being conducted must be able to be evaluated regarding their value for industrial SE communities. In addition, if there is a possibility of applying similar type of experiments in the industry, then, it is worth conducting the study as an experiment in a classroom setting. The use of students as the participants for the experiments initiated in this research was seen to be correct as confidence of the observations carried out on these experiments was gained. It therefore appears that similar types of research work (experiments) can be conducted in industries. It is not a trivial task to gain approval from industry to run such experiments in their premises since the time (which is equivalent to cost) allocated for the purposes of the study needs to considered.

4.1.4 Assumptions for the Experiments

Assumptions regarding the research must be considered to the experiments. Several such assumptions are listed below:

- Subjects' prior knowledge for developing Use Case Diagrams

  The subjects of the experiments were students. They were third year undergraduates and postgraduates who had prior knowledge of Software Engineering. The participants were assumed to be well versed in designing and developing Use Case Diagrams.

- Familiarity handling mine drainage system

  It was assumed that the subjects had neither work experiences nor industrial background related to mine drainage system prior to the experiments. In addition an assumption was made that the instructors for the experiments were well versed in the mine system.

- Subjects' background and skills

  The majority of the subjects involved in the experiments were assumed to have similar cultural background and level of maturity. The ability to understand, analyse, interpret and solve the problem varied according to the individual. No special criteria for forming the groups were used during the experiments. It was presumed that the individuals' skills would not influence the outcome of the experiments.

- Subjects' knowledge on the detection of assumptions

  Subjects were assumed to have same level of knowledge and skills for detecting assumptions. They were given approximately four hours of training in Requirements Engineering topics prior to the execution of the experiments.

4.2 Data Collection Instruments and Protocols

Preliminary research was done by carrying out observations on students' projects to study the occurrence and issues related to assumptions during the

Requirements Elicitation Process. Further empirical studies were carried out to gather further results for detecting assumptions during Requirements Analysis.

4.2.1 Examination of Students' Projects

Observation was made on students carrying out software projects to conduct the preliminary studies regarding assumptions in the early phase of software development. As described in the Section 4.1.3, CITS3220 unit was chosen as the unit in which to conduct observation. This particular unit was the only unit which replicated a model of software development project in an industrial setting. There were thirty one students enrolled in this unit. They were divided into three groups of students, Team *A*, *B* and *C*. The maximum number of students in each group was eight and the minimum was six. However these numbers did not remain constant until the end of the project as there were students who dropped the unit. There were seven projects offered by several clients and each group was required to choose one project. The students were the software developers. The groups were allowed to choose their projects. Hence, the team members were anticipated to discuss the available expertise in their groups and choose a suitable project which allows them to utilize their team members' capabilities.

Each team had to submit four deliverables and an individual summary report. The deliverables were a Requirements Analysis Document, design documents, coding documents and the project presentation respectively.

Two of these teams were selected for observation. These were Team *A* and *B*. The reason for not selecting Team *C* was simply because the meeting time for Team *B* clashed with Team *C*, the choice for only two of the three teams. There were meetings

held between the team members and clients as well as meetings among the team members for the entire duration of the project. Observations during the meetings are illustrated in Figure 4.5. The observer did not interact with any of the participants in the meeting. The observers only joined the participants during the meetings to listen to their discussions. The observations conducted were used to study the issue of assumptions during software requirement development by the students. This scenario was taken as a prototype of real world software projects developed in industries.

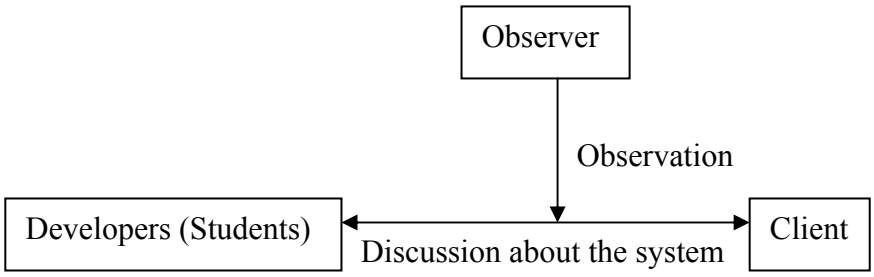


FIGURE 4.5 – *Observation on Students' Projects*

4.2.2 Scenario Based Experiment – Process and Rationale

Scenario based techniques have been used in software development process to elicit, define and model requirements. The scenarios found in software development projects are used as the examples for designing this method. The scenarios simulate events in which a user would experience performing the tasks that constitute the operation of the system [78]. An early requirements prototype had been developed using scenarios which were based on operational examples. The purpose of choosing scenarios as a method to develop the prototype was to allow fast extraction of requirements in a timely manner. This happens to serve as an advantage to allow early analysis and validation of the requirements [79].

This approach is used to design the first trial, the Scenario Based Experiment (SBExp). The experiment's goal is to identify assumptions from a requirements document which provides the participants with minimum information based on the scenario given to them. SBExp is executed in a classroom-based environment. There are instructors who are responsible to disseminate the directions to carry out the experiment and therefore the experiment process will be observed. Figure 4.6 illustrates a brief overview of the SBExp.

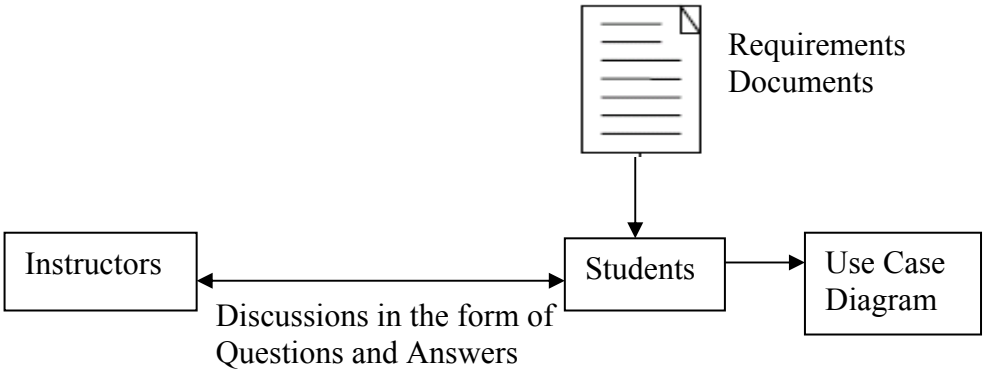


FIGURE 4.6 – *Overview of the Scenario Based Experiment*

SBExp was conducted during a tutorial session. The subjects had been briefed on requirements engineering topics prior to the experiment session. The time taken to complete the experiment was 40 minutes. The number of subjects who participated in the experiment was 32; two instructors were required to conduct the experiment. The subjects were teamed into seven groups, *A, B, C, D, E, F* and *G*.

The hypotheses of this experiment were:

(i) *A larger number of validated assumptions contributes to higher degree of accuracy in the requirements documents. If larger numbers of questions were asked by the participants, then we expect to see a higher degree of accuracy in designing the Use Case Diagrams.*

(ii) Requesting participants to question assumptions aloud reduces the number of assumptions made.

The questioning method is believed to elicit assumptions when it is used wisely. By holding question and answer (Q&A) sessions within both the developer's team and within the client's team and also between developers and clients allows the ironing out of the differences. There is an opportunity to resolve unclear requirements as well as clarify assumptions by carrying out frequent Q&A sessions.

The process flow of the experiment is depicted in Figure 4.7. At the beginning of the experiment, the instructors will distribute the experiment papers (as illustrated in Appendix A-1) and brief the subjects about the goal and method to complete the experiment. The subjects are required to form a group of maximum four before the experiment documents were distributed to them.

The subjects were given a scenario which required them to take the role of a System Analyst. They had received an email from their manager who had appointed them to develop a Mine Drainage Control System (MDCS). The email consists of a brief forwarded message from the client who has requested a MDCS; this step has been deliberately taken in order to reduce the amount of information given to the subjects.

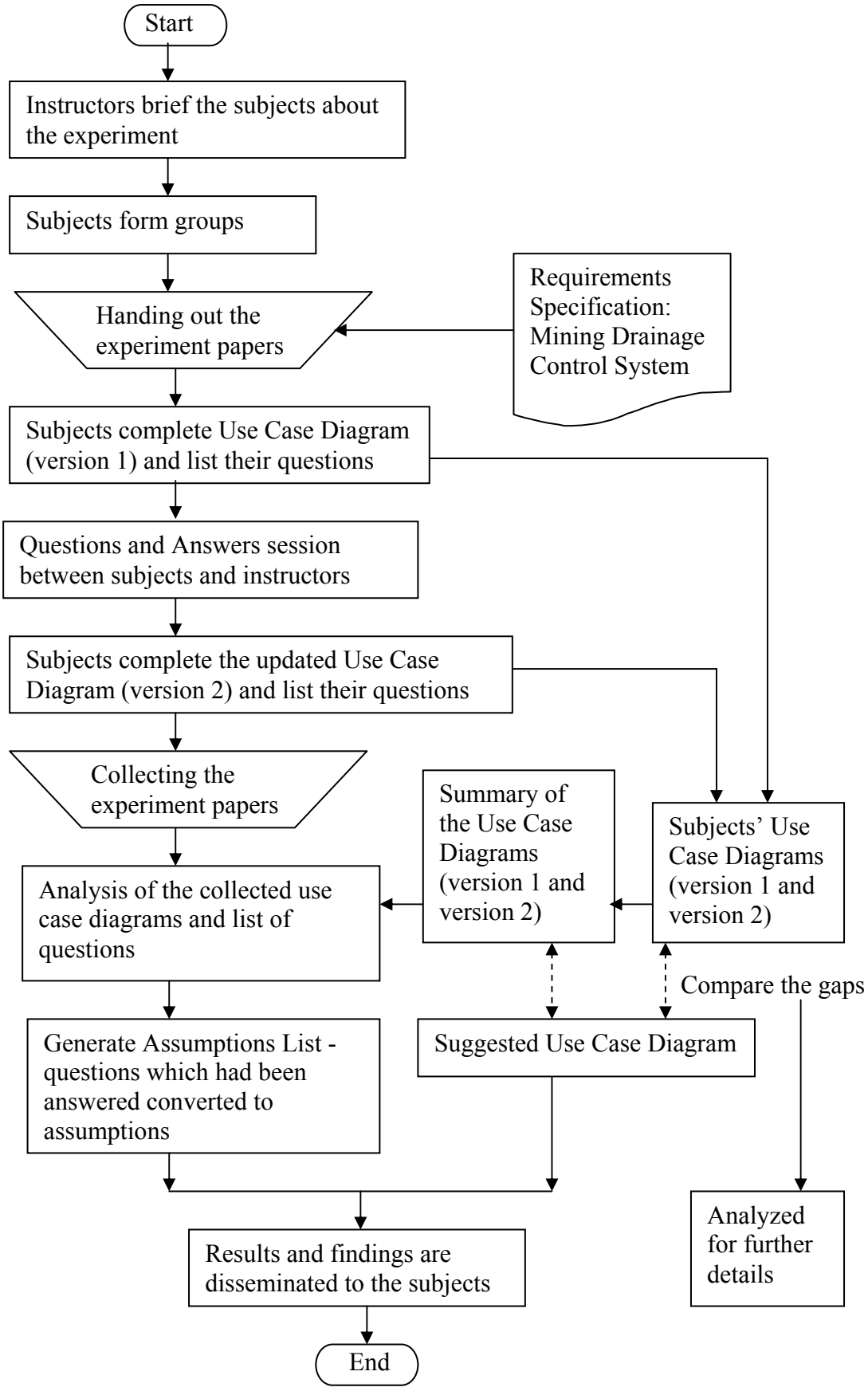


FIGURE 4.7 – P*rocess flow for Scenario Based Experiment*

The experiment consists of two iterations. They were given a duration of fifteen minutes to complete the first iteration. The first iteration requires them to understand the message from the client and list all the possible questions which they would like to follow up with the client in the next meeting. The subjects were instructed to list their questions in 'Form A' in their experiment paper (as described in Appendix A-1). They were also supposed to design Use Case Diagram (UCD) which best illustrates the required behaviour of the system according to the minimum information provided to them in the client's mail. This UCD was represented as version 1.

Following on the completion of the task, the instructors will request the subjects to stop their work and to commence an open discussion session which takes place for the next ten minutes. During this session, the subjects can raise their questions which they have listed previously and answers will be provided by the instructors.

Prior to executing this experiment, a checklist of questions and answers, called 'My Checklist of Questions', as depicted in Appendix A-3, had been prepared by the instructor by referring to possible questions which had been predicted to be asked by the subjects. This checklist was used to provide answers for any relevant questions during this open discussion session with the subjects. Note that the checklist of questions listed in Appendix A-3 is merely a predicted number of questions. If any of the questions asked by the subjects were not found in this checklist, then an ad hoc answer will be given to the students according to the instructor's knowledge with regards to MDCS. Here it is assumed that the instructors know the system thoroughly and are able to answer subjects' questions. Each one of the group take turns to ask questions and the rest of the group were required to listen to these questions. If any one of the group found that any of the questions which being is raised is similar to the questions which they

have listed, then they are required to record the answers for it and not to repeat that particular question again.

After the open discussion ends, the second iteration begins where the allocated time was fifteen minutes. In this iteration, the subjects were instructed to design the UCD for the second time and this will be the updated version of the UCD. This updated UCD was represented as version 2. The subjects were welcomed to list any further questions if they believe that they have more queries which they would like to clarify with the client. At the end of the experiment, the documents consist of both UCD (version 1 and version 2) and list of questions were collected from the subjects. These documents were analyzed.

Prior to conducting the experiment, a 'Suggested UCD', which is actually the superset of the UCD (as depicted in Appendix A-2), was designed by the instructors. This 'Suggested UCD' was developed by referring to the literature [68, 69, 70, 71, 72, 73]. This UCD was not claimed to furnish the perfect design for the mining system, yet should be acceptably close to the intended UCD.

The analysis for the UCD was divided into two parts. The first part compared both the UCD, version 1 and version 2 developed by each group with the 'Suggested UCD' respectively. The aim of the first part of the analysis was mainly to capture the subjects' capabilities in individual groups for creating a UCD with the minimum information provided to them. As a result from these comparisons, both UCD (version 1 and version 2) were encapsulated and constructed as a single design which is known as 'Composite Use Case Diagrams' (Composite UCD). This 'Composite UCD' was built

by encompassing all the elements of UCD from each of the groups participated in the experiments by eliminating duplicates.

In the second part of our analysis, we compared the 'Composite UCD' with the 'Suggested UCD'. The second part of the analysis was done to measure how accurate the collective information gathered from each groups in the experiments were able to match with the 'Suggested UCD'. The gaps between these Use Case Diagrams in both parts of the analysis were further studied.

The questions related to building the mining system, collected from the groups were listed in document called 'Students' Initial List of Questions' (depicted in Appendix A-4). These questions were merged with 'My Checklist of Questions' into a single list called 'Superset of Questions' (depicted in Appendix A-5). This 'Superset of Questions' is used as the "reference document" to analyze the questions collected from the students. Each answer for the question is converted to an assumption. Here, we will elaborate the logic of converting these questions to relevant assumptions. In the process of decision making, one is likely to assume if one is unsure about certain information when making a decision. The probability for the subjects in this experiment to make assumptions would be decreased if they had a chance of acquiring the answers for their questions. Therefore questioning will be an appropriate action to gain the information. In this case, if and only if they had not obtained the answers or information for their questions, there will be probabilities for the answers to each question to be assumed. Thus the act of questioning can be used as a motivation for eliciting assumptions. The matching answers obtained for the questions were converted to assumptions. We believe that the possible cure to avoid an action of making assumptions is by trying to question them aloud; this is one of our experiment's hypotheses.

The rationale of SBExp states that if a larger number of assumptions made by the subjects were validated, then we expect to see a higher degree of accuracy in the UCD which is being developed. Consequently, the questions asked by the subjects were used to measure the accuracy of the UCD which they had built. Hence, the correctness of the UCD is interlinked with the questions raised by the subjects. The lists of questions collected from the subjects were used to capture subjects' capabilities in acquiring relevant information to build the mining drainage system by referring to the client's message. The outcomes of the experiment were reported and disseminated to the subjects.

4.2.3 Assumption Seeding Experiment

The Error Seeding approach established by Mills [80] is one of the techniques grouped under the software verification and validation methods and currently being used in the software development processes. Figure 4.8 illustrates how a simple error seeding model works. According to this technique, a known number of faults are seeded in a program code which is assumed to have an unknown number of original faults. The program code is tested and the number of detected seeded and original faults is counted. From these counted faults, an estimated number of the fault content in the program code prior to seeding is obtained. This can be used to assess software reliability and other relevant measures. A standard definition from IEEE [81] states that error seeding is:

> *"the process of intentionally adding known faults to those already in a computer program for the purpose of monitoring the rate of detection and removal, and estimating the number of faults remaining in the program"*

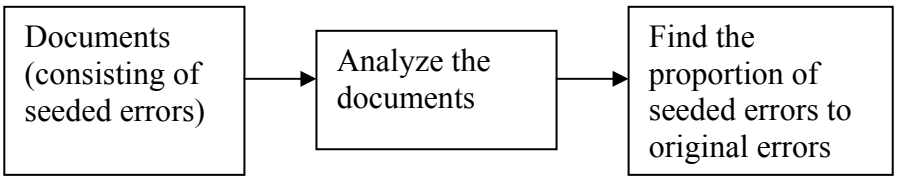


FIGURE 4.8 – *Error Seeding Model*

Let us define the detected seeded errors as E_{ds}; total seeded errors as E_{ts}; detected original errors as E_{do} and total original errors as E_{to}. The formula to obtain the total original errors remaining in a program code is described in Equation (1).

$$\frac{E_{ds}}{E_{ts}} = \frac{E_{do}}{E_{to}}$$

$$E_{to} = \frac{E_{do}\ E_{ts}}{E_{ds}} \qquad (1)$$

The defects seeding approach prove to have gained the confidence of many researchers as it was being embedded into their research [38, 82, 83]. Halling [82] studied the benefits of using inspection method from economical perspectives for requirements negotiation process. He had seeded defects into the inspection documents to appropriately assess the performance of the inspection technique.

Upon completion of the Assumptions Seeding Experiment (ASExp) and Scenario Based Experiment (SBExp), some research done by Basili et al [38] was noted. Basili's work described that the defects seeding method was used to evaluate the benefits and costs of the process of abstracting errors from faults in requirements documents. In that research, a classroom based experiment had been carried out in two groups using an ad hoc and Perspective-Based Reading (PBR) reviews respectively to examine error abstraction process. PBR is a method for detecting defects in natural language requirements documents. The requirements documents being reviewed had

been seeded with a number of known faults. The percentage of faults found in the requirements documents is used as a measure of the error abstraction process. The results of Basili's studies showed that PBR method did contribute to a certain extent to the process of extracting errors even though the process was not effective in the way the authors' had expected.

It was concluded that the research noted in the previous paragraph is the closest in content to this research. However the idea of seeding assumptions which had been adopted in the ASExp had not been practiced in research carried out by Basili et al. They have adopted defect seeding in their experiment to detect errors from requirements documents using PBR method. It is useful to compare the two studies.

Table 4.1 illustrates the similarities and differences between the criteria used in the research performed by Basili et al and this study. The similarities and differences are respectively denoted by symbol '≡' and '≠' at the end of each statement.

Basili et al in the previous literature believed that revealing the number of faults would not be a good measure for defect detection effectiveness. Therefore they decided to concentrate on the underlying errors of these faults. There were questions developed to uncover each type of fault. These questions were built upon using the help of a suggested taxonomy of faults in natural language requirements by the authors [38].

TABLE 4.1 – Similarities and Differences between EAiRE and Basili et al

Descriptions of the research criteria	**Research carried out for EAiRE**	**Research done by Basili et al**
main interest of the experiments	to reveal the number of assumptions in the requirements documents (≠)	to focus on the causal errors for the defects by considering the benefits of this effort (≠)
Experiment's objective	Detect assumptions (≠)	Abstract errors (≠)
participants of the experiments	Students (≡)	Students (≡)
method used in conducting the experiments	reviewed requirements documents (≡)	reviewed requirements documents (≡)
	no specific technique were used (≠)	used a technique called Perspective Based Reading (≠)
	seeded assumptions in requirements documents (≠)	seeded faults in requirements documents (≠)
	scenarios were used to help the participants to complete the tasks assigned to them in the experiments (≡)	operational descriptions or scenarios to support the participants throughout the reading process for the experiment (≡)
	classroom open discussion used in SBExp for the purpose of eliciting assumptions (≠)	Focused in-classroom discussion were performed at the end of the second session of the experiment to measure participants' response on the key dependent variables in this study (≠)
taxonomy	taxonomy for assumptions in Requirements Engineering built to find the inter-relationship between the elements of the taxonomy (≠)	taxonomy of faults in natural language developed to assist in developing questions to reveal each type of fault (≠)
type of requirements documents	one RSD for Mine Drainage Control System which 2 pages in long (≠)	two RSD for an Automated Teller Machine which 17 pages in long and Parking Garage Control System which 16 pages in long (≠)

On the other hand the experiments in EAiRE were designed to identify the number of assumptions. When dealing with assumptions, it is not a trivial task to identify them as there is always a possibility for unconscious assumptions (issues which are knowingly assumed) as mentioned by Armour's Second Order Ignorance [83]; "we do not know that we do not know". It can be a challenging job to detect the underlying errors for the defects propagated by false assumptions. Hence, the first step which is appropriate at this stage is to discover the number of assumptions pertaining to the requirements documents.

Seeding errors into programs has also been used as a method to find the limitation for software testing processes. Knight and Ammann [84] stated that the usual criteria to terminate testing processes are either the allocated resources for the process had been exhausted or the process deadline had been reached. However, they claim if the exact number of faults in a program created unconsciously by the programmers during development process was known, the termination for testing could be estimated. Therefore, they have applied the error seeding technique for determining when a program has been sufficiently tested using functional and random testing.

It seems possible that one of the methods of extracting the correct results from a problem is by adding more issues into the problem in order to extract a correct outcome. From this viewpoint, the ASExp has been designed in this research by adopting the error seeding technique.

The objective of this experiment is to elicit assumptions from a requirements document. The experiment is designed to be conducted in a classroom-based

environment. The directions to carry out the experiment will be given by the instructors and therefore the experiment process will be observed as described in Figure 4.9.

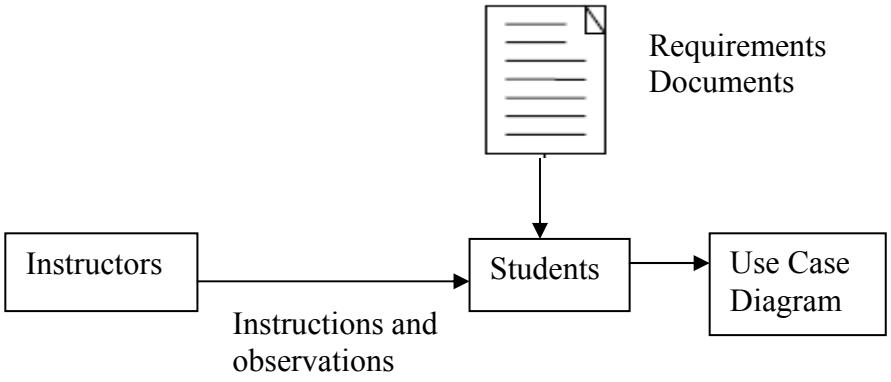


FIGURE 4.9 – *Overview of the Assumptions Seeding Experiment*

The hypothesis of this experiment is:- *the number of assumptions in a Requirements Specification Document can be usefully derived using the assumptions seeding technique*.

4.2.3.1 The Early Pilot Trial

Colleagues were selected to take part in the ASExp. The experiment documents were attached with a Requirements Specification Document (RSD) for Mine Drainage Control System (as illustrated in Appendix B-5) was distributed to four individual participants and they were allowed to complete these at their own time. They were required to complete two tasks. First, they were assigned to design a use case diagram (UCD) by referring to the RSD. Second, they were required to list all the assumptions which they believe they have made during the process of developing the UCD. The subjects participated in this experiment were students. The selection criteria for participants were they must have background knowledge in Software Engineering as well as being graduate students.

The subjects were required to provide some personal information such as their age, occupation, the time they started and ended the experiment. The reason for acquiring the age and occupation of the subjects were simply to verify the relevancy of their feedbacks on the experiments with regards to their experiences. The participants did not communicate with each other while completing this experiment. Hence, we believe the participants were not able to discuss the outcomes of the experiments. There was no time limit given to them as they were required to finish the task at their own pace. This reason is simply to discover the average time duration used by the four participants.

The feedback gained from the trial was used to improve the experimental methodology before conducting the ASExp in a larger experiment involving 12 undergraduate students. For instance the average time duration taken to complete the tasks by the subjects was used as a guideline for allocation of time to complete the experiment in the tutorial.

4.2.3.2 The ASExp Process and Rationale

The experiment was executed during a tutorial session. The duration of the experiment was an hour. A total number of 12 subjects participated in the experiment and two instructors were required to conduct the experiment. The execution of the experiment is as illustrated in Figure 4.10. The instructors briefed the subjects about the objective and manner to complete the experiment. They were required to form a group of a maximum four. After they have decided their groups, the experiment documents (as depicted in Appendix B-2) were distributed to them. The subjects were required to act as System Analysts to develop the Mine Drainage Control System. The Requirements

Specification Documents (RSD) for the system was attached with the experiment documents. A total number of ten assumptions were deliberately inserted or seeded into this RSD.

The experiment consisted of two tasks, assigned to the subjects to be completed. First, the subjects were required to analyse the RSD and design a use case diagram for the system. Second, they were expected to list all the possible assumptions which they have made in the process of developing the use case diagram. However, before they began their tasks, a partially completed UCD for the mining system was displayed on the screen for two minutes for the subjects to view it. The purpose of this step was to provide an example of UCD for the mining system to help the subjects to initiate their tasks.

In order to ensure a smooth execution of the experiment, the instructors observed the subjects while they carried out their assignment. The instructors will help the subjects by attending to any questions in the case where subjects have doubts about completing the experiment. However, there would not be any assistance given by the instructors to subjects on designing the UCD and listing the assumptions. The subjects are demanded to contribute their own ideas on finishing the experiment tasks. The reason for avoiding discussion between the subjects and instructors about the answers for the tasks is to maintain the outcomes of the experiments to be merely from subjects' effort without any influences from the external parties.

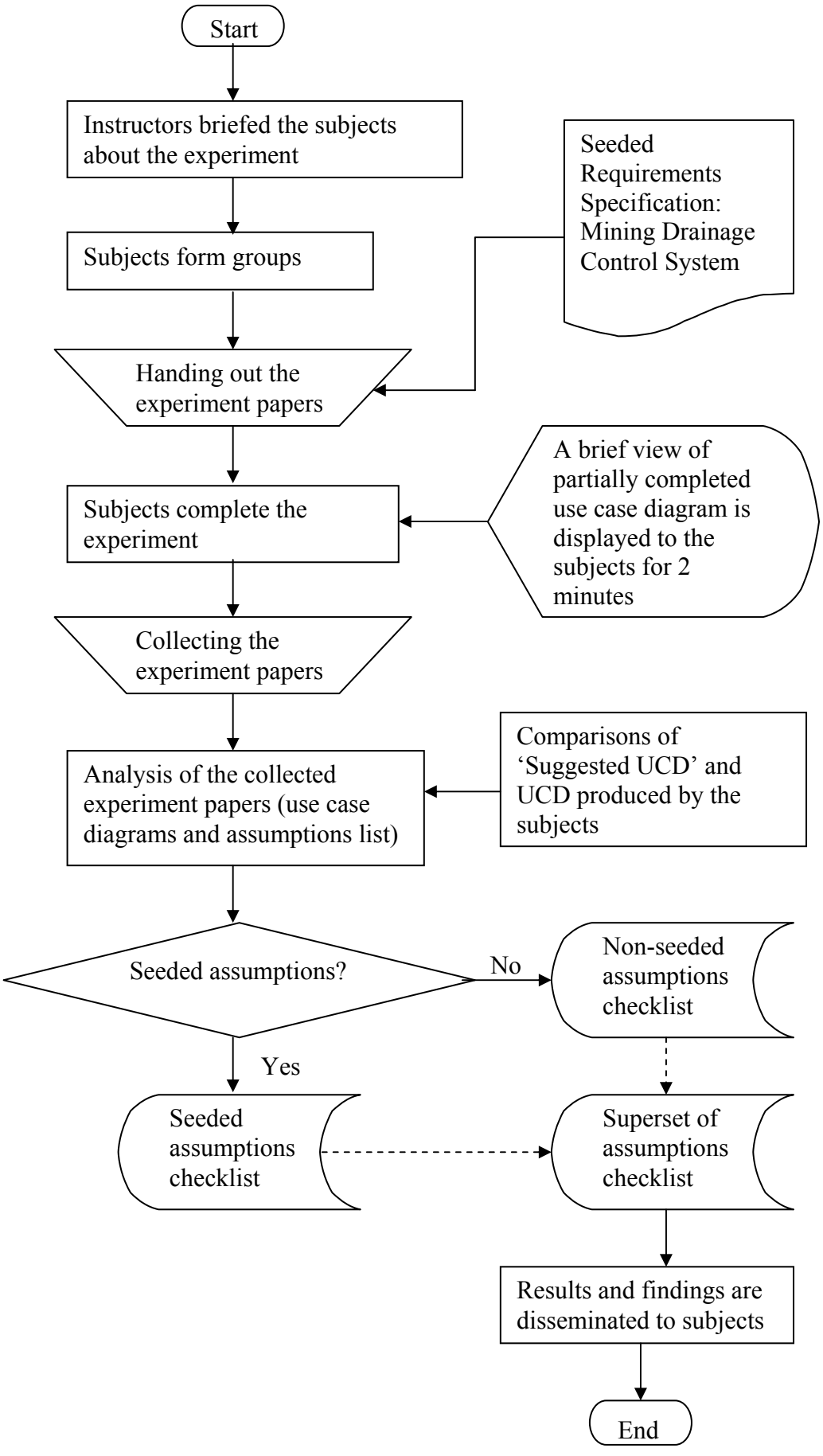


FIGURE 4.10 – *Process flow for Assumptions Seeding Experiment*

The experiment was scheduled for an hour and subjects were informed once they have reached the time limit. The documents were collected from the subjects upon completion of the assignment and analysed. The outcome from the experiment's first task, the UCD is compared with the 'Suggested UCD' as illustrated in Appendix A-2. There are two reasons for comparing the UCD described as below.

- This step assists us to measure the correctness of the UCD designed by the subjects.
- We are able to determine their understanding about the Mine Drainage Control System with regards to the content of the requirements documents provided to them.

The correctness of the UCD is interlinked to the subjects' understanding of the requirements. As a result the subjects' misconception about the requirements of the mining system was considered revealed in the UCD designed by them.

The second task of the experiment required the subjects to list all the assumptions generated during the designing process of the UCD. The list of assumptions were collected and reviewed.

As we have mentioned previously, the requirements documents for the Mine Drainage Control System have been intentionally seeded with a number of known assumptions. Therefore, the assumptions lists collected from the subjects are examined to find the seeded assumptions. The original or non-seeded assumptions embedded in the requirements documents are also discovered during this process. A checklist of seeded assumptions and original assumptions (as shown in Appendices, B-3 and B-4

respectively) had been developed prior to execution of the experiment. These checklists were used as the baseline for the assumptions discovered by the subjects.

The rationale of ASExp is to measure the number of non-seeded assumptions generated from the requirements analysis process. The number of original assumptions remaining in the requirements documents can be measured by the proportion of seeded assumptions discovered. Consequently, the number of undiscovered seeded assumptions will be used as the indicator of the number of total indigenous assumptions remaining in the requirements. The relationship between detected seeded assumptions (DSA), total seeded assumptions (TSA), detected original assumptions (DOA) and total original assumptions (TOA) from the requirements is described in Equation (2). The total original assumptions can be the factor which leads to potential defects in requirements if it is not validated.

$$\frac{DSA}{TSA} = \frac{DOA}{TOA}$$

$$TOA = \frac{(DOA)\ (TSA)}{DSA} \qquad (2)$$

The gathered assumptions from ASExp were compiled to form a 'Superset of Assumptions' (as shown in Appendix B-9). This superset consists of seeded and original assumptions (as listed in Appendices, B-3 and B-4), and original assumptions derived by the students which is not found in Appendix B-4. The idea of expanding the assumptions list is merely for the interest of uncovering the total number of assumptions. The outcomes of the experiment is reported and disseminated to the students to enable us to share the insights gained from their tutorial session with them.

4.2.3.3 The generation of Seeded Assumptions

Deliberate elimination of segments of statements from the Requirements Specification Documents (RSD) was used to seed assumptions as illustrated in Appendix B-1. The elimination of the sections of requirements was made by referring to the 'Suggested UCD' as depicted in Appendix A-2. For instance two statements were removed from the RSD pertaining to the pump operations which contributed to Assumption 1 (*A1*) as shown in Table 4.2. In addition another statement also related to pump operation, was removed from the RSD, which reflected on Assumption 5 (*A5*). Consequently, both seeded assumptions, *A1* and *A5* were created by removing the relevant statements from the requirements document.

TABLE 4.2 – The Statements Eliminated from the Requirements Specification

Seeded Assumptions		**Statements eliminated from the Requirements Documents contributing to the respective assumptions**
No.	**Descriptions**	
A1	*The pump should be only operated when the level of methane is below a critical level.*	the main safety requirement is that the pump should not be operated when the level of methane gas in the mine reaches a high value due to the risk of explosion.
		the pump should be only allowed to operate if the methane level in the mine is below a critical level
A5	*The environment monitors the level of methane and there is a level beyond which is unsafe to mine or operate the pump.*	there is a level beyond which it is unsafe to mine or operate the pump

The descriptions of both the seeded assumptions, *A1* and *A5* are extracted from Appendix B-3 and illustrated in the above table. Both these seeded assumptions state the

required status of the pump in order for it to be operated. In addition, it also explains the conditions where the pump should not be operated.

The information obtained from both these seeded assumptions was matched with the 'Suggested UCD'. As shown in Figure 4.11, *A1* and *A5* reflected on the communication between the *Pump Motor* and *Check Methane Level*. Thus, during the process of designing the UCD, if the subjects fail to identify the communication between *Pump Motor* and *Check Methane Level*, it is possible that they are potentially unable to discover both the seeded assumptions, *A1* and *A5*. However, these will not be the only possibilities as the subjects may identify the *A1* and *A5* but yet did not state the communication between *Pump Motor* and *Check Methane Level* in their UCD.

Though it is not a trivial task to design the seeded assumptions, the design in this research is believed to be optimal and no better specific guidelines, used in order to design the seeded assumptions, were discovered.

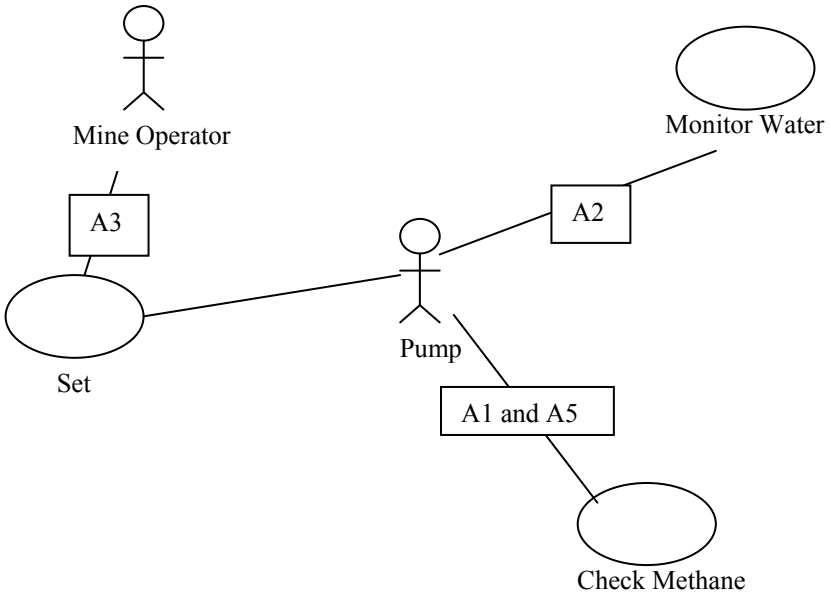


FIGURE 4.11 – *A Segment from the Use Case Diagram with Seeded Assumptions*

4.3 Principle for Assumptions Analysis

The principle below has been used to design the basis of this research, Exploration of Assumptions in Requirements Engineering (EAiRE).

> *When there are two documents referring to the same subject matter at the same level of detail, where one document contains more information than the other, then, the missing information from the second document is defined to be an assumption - which is presumed to be true for the purpose of the subject matter in that document.*

The above principle explains that if there are two documents with similar content and if one of them has missing information; then, it could be said that the missing information were the assumptions which could be taken for granted to be true in order to work out the missing information. The missing information could be used for measuring the assumptions. In ASExp, statements from the Requirements Specification Documents have been removed deliberately. These statements are represented as the missing information. Therefore, the missing statements are the seeded assumptions which need to be detected by the subjects.

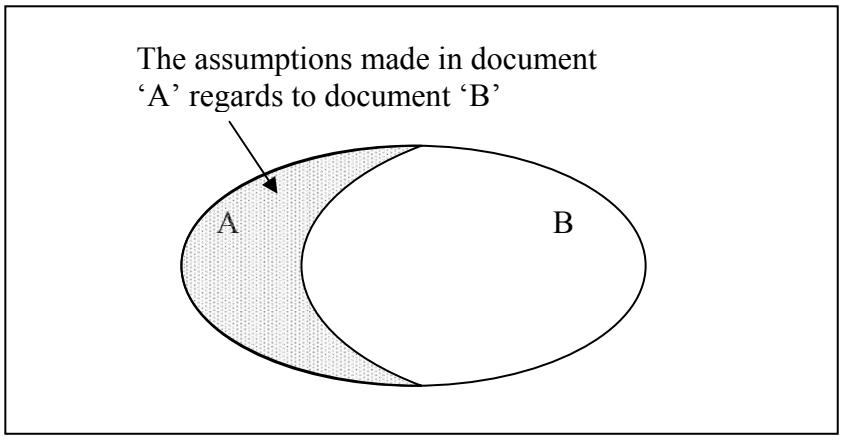


FIGURE 4.12 – *The Principle of Analysing Assumptions*

The concept described in the previous paragraph is also depicted in the Figure 4.12. This principle has been used to derive observations and results of the Assumptions Seeding Experiment; this will be explained in Chapter Five.

A detail analysis of the data gathered for Scenario Based Experiment and Assumptions Seeding Experiment will be described in Section 5.2.1 and Section 5.4.1 of this thesis.

4.4 Conclusion of the Topic

The design has been elaborated above for both the experiments, Assumptions Seeding (ASExp) and Scenario Based (SBExp) conducted for the purpose of this research. The experiments were designed to reveal the number of assumptions found in the requirements documents for a Mine Drainage Control System. The goal of the SBExp was achieved by clarifying doubts pertaining to the system by using questioning method. Further, the ASExp was used to find the remaining original assumptions in the requirements documents. The number of the assumptions uncovered is considered with reference to the size of the requirements document used in these experiments. In the next chapter, the outcomes of these experiments will be considered.

Chapter 5 Results

In this chapter, the data gathered from the three experimental activities carried out as a part of this research will be detailed. The data comprises: (1) observations on students' projects; (2) the Scenario Based Experiment; and (3) Assumptions Seeding Experiment (as described in Chapter Four). The analysis of the collected data revealed support for the thesis arguments. The results presented confirm the importance of discovering assumptions early in Requirements Engineering for software development.

5.1 Observation on Students' Projects

This section will describe the examination of assumptions as an issue during Requirements Engineering (RE). The observation on the students' projects was the initial attempt to study this issue. Although the scale of the students' projects is small compared to most projects in industry, they are a subset of real world software development projects. Thus the outcomes of the observation can be used as a potential guideline in industries to improve the effectiveness of the Requirements Engineering Processes.

Observation of Team *C* was not possible as their meetings clashed with the meetings of other groups, as described in Section 4.2.1. Therefore only the observation on two of three groups has been reported. The observations on Team B are presented before those of Team *A* as Team *B* provided more basic information.

5.1.1 Outcomes from Team B's project

Team *B* were assigned the task of designing software that enabled an owner of a Personal Digital Assistant (PDA) to request and build online quotation for computer. The pricing and configuration options for the computers should be supplied and updated regularly by the system. The system should also incorporate a function which automatically sends an email to the requester which records the estimated price of the computers.

The development team consisted of five members (who were students) and a client representing company named *X*. The document Meeting Timetables (refer Appendix C-1), showed that the project commenced with six members. Later one of the team member withdrew from the unit leaving only five members in the team. A similar scenario occurs in industries if a member of a development team resigns from the organisation or transferred to another team.

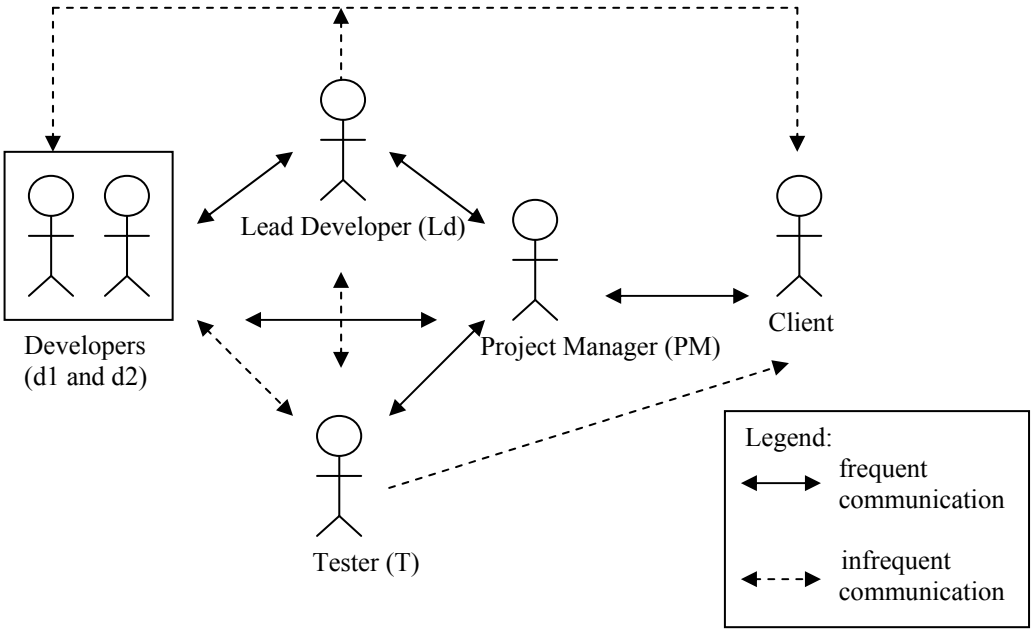


FIGURE 5.1 – *Team B's communication activities*

The roles for each of the member and their relationships with the client are presented in the Figure 5.1. The team consisted of a Project Manager (*PM*), a lead developer (*Ld*), two developers (*d1* and *d2*) and a tester (*T*). The *PM* was responsible for coordinating the project. He managed the work load distributions of the project, arranged and chaired the meetings among team member and with the client. In addition, he was also the main channel of communication that liaised with the client during development. It was important to take note of the communication pattern for Team *B* because this was one of the factors which contributed to the success of their project.

Team B had attended two meetings with the client. Significantly the client emphasized to the developers that they should not work on assumptions during the development and requested them to enquire and clarify any information which they encounter. He provided his contact details and availability to the developers to allow them to reach him easily. The client's previous experience with the IT industry enabled him to give valuable advice to the team not to use assumptions and be alert when making decision and moving forward in development. A simple warning as described in this scenario can reduce the risk of propagating assumptions during early phases of development.

The team arranged group meetings where they discussed the progress of the project. There were a total of four team meetings. The Project Manager (*PM*) emphasized the importance of communication among team members during the project. He also advised the team to contact him with regards to the development matters when they need further information about the requirements. It was assumed, other than the team's initial interaction with the client during the first meeting, that the *PM* would be the sole mode of communication between the group and the client.

The first requirements draft developed by Team *B* after their discussions with the client is shown in Table 5.1. The analysis and discussion were on the functional requirements since the draft focused on these aspects. These requirements items have been numbered for easy references. The analysis on the requirements shows that the use of natural language for constructing the requirements contributes as the main probable cause of propagating assumptions.

TABLE 5.1 – Initial List of Requirements developed by Team B

(italics are referring to the words, unclear and vague, and missing information which leads to potential assumptions)

Requirements Number	Requirements Descriptions
R01	***All*** constraints on building a configurable system will be enforced.
R02	The system must be able to be viewed ***correctly*** by the intended PDA devices.
R03	An email quote will be produced if the user selects to 'email' the quote by clicking the 'done' button.
R04	The ***system*** should reflect the total price of the ***system***, as specified by the ***internal data***, as the user selects or deselects different components.
R05	***Each section*** will calculate the price of the selected components of that section.
R06	The web page should ***correctly*** display the components the user has selected or deselected.
R07	Administrators can view the components in the system including ***elements not available to the user***.
R08	There will be ***an appropriate mechanism*** to change the ***internal data*** without affecting users already accessing or trying to access the data.
R09	There will be ***appropriate interface*** so as to allow administrators to change the internal data and provide integrity constraints.
R10	The user should not be able to edit the ***internal data***.
R11	The user must be able to restart choosing a system and all options should be set to default.
R12	The user should receive confirmation of the system they have 'built' and be able to return to initial page.
R13	The ***internal*** must be secure and free from becoming corrupt.

For instance, the word '*All constraints*' in *R01* is generalizing the system constraints without listing them individually. Hence the constraints, according to the

client's viewpoint, might contradict with the developers' viewpoint. The difference in opinion arises because the list of constraints was not discussed overtly and documented. The previous scenario can contribute to the possibility of generating assumptions. There are probabilities for these assumptions to create errors if the assumptions are invalid. The outcomes of the process for validating assumptions are explained below.

> *If an assumption is invalid, then there are two possibilities: (1) it either can be an error and requires rework or (2) it does not affect the development process at present but can represent a potential error as the requirements evolve.*
>
> *If an assumption is valid, then it does not affect the development process at present but can represent a potential error as the software requirements evolve.*

Therefore the use of natural language in *R01* creates the chances for one to assume invalid constraints for building a configurable system for the software which might lead to errors in the development process.

The use of the word '*correctly*' in *R02* in Table 5.1 is vague. It allows the client and developers to define the word '*correctly*' from their point of view. The definition can be invalid because of the differences in opinion which may lead to inaccurate information. The use of the word '*system*' was inconsistent in *R04*. The first occurrence of this word referred to the software, while the second occurrence referred to the computer system which a user intended to purchase. The incoherency of the words used in the requirements may lead to errors. The word '*internal data*' in *R04* was not clearly defined in any part of the requirements and, again, those concerned one can assume their own definition of '*internal data*'. There are other unclear words used for

constructing the requirements as represented in italic bold in Table 5.1. These words can contribute to misconceptions among the developers and client. Hence natural language is one of the causes for initiating assumptions from the requirements documents.

Several researchers [38, 85, 86, 87] have discussed the manipulation of natural language for detecting issues in Requirements Engineering. For instance, Basili et al [38] have illustrated the use of an error abstracting process by using taxonomy of faults in natural language requirements. The taxonomy consists of types of information that are missing, ambiguous, inconsistent, incorrect facts, extraneous and/or miscellaneous. The taxonomy was utilized by the authors to construct questions which were used to discover faults. The authors did not underline the fact that the use of natural language in requirements documents has the probability of generating assumptions. However, they did highlight the fact that natural language is one of the factors which makes the requirements document vulnerable to faults. It can be argued that a subset of these faults was caused by errors contributed by incorrect assumptions.

However the role of diagrams can be considered important for clarification and verification of natural languages in software projects. Most research has been based on using Unified Modeling Language (UML) diagrams for formalizing the requirements model [88]. UML is an object modeling and specification language used in software engineering. Therefore using some of the UML model, such as UCD, will ultimately reduce the confusion and ambiguity in software requirements' descriptions caused by the use of natural languages. This will also contribute to a reduction of the generation of assumptions caused by the use of unclear words in requirements documents.

The functional requirements were taken from the Requirements Analysis Document (RAD) produced by Team B. The use of natural language in developing their requirements is analysed as shown in Table 5.2. This list of requirements is the revision of the first requirements draft shown in Table 5.1. The column '*Requirements Name*' have been added by the developers to name the functional requirements in RAD.

TABLE 5.2 – Functional Requirements List by Team B

Requirements		
Number	Name	Descriptions
R01	Integrity Constraints	All constraints on building a configurable computer system will be enforced.
R03	Quote Verification	An email quote will be produced if the user selects to 'email' the quote by clicking the 'done' button.
R04	Pricing Integrity	The system should reflect the total price of the computer system, as specified by the internal data, as the user selects or deselects different components.
R07	Full Internal Access	Administrators can view the components in the system including elements not available to the user.
R08	Non Interference	There will be an appropriate mechanism to change the internal data without affecting users already accessing or trying to access the data.
R09 and R10	Permission Access	The user should not be able to edit the internal data. Administrators can approve new components to be available in the configuration and remove components that are obsolete.
R11	Re-enterable System	The user must be able to restart choosing a system and all options should be set to default.
R14	System Logic	System interactivity should progress linearly during system components selection to avoid accidental omitting of components. However, users are able to modify previous selections.
R15	Smart selection	Only relevant system components should be displayed for user selection with respect to previous user selections.
R16	Automation	The component price list will be uploaded regularly (weekly) and software should identify and timestamp the changes.
R17	Operation	The software will enable a user to select a computer system and configure its optional features.
R18	Compatibility Warnings	System should provide a warning message to tell users that their modified selection might have compatibility issues. Alternatively, in the case of linear progression, bring the user back to their last incompatible selection.

In order to refer and discuss the requirements, the '*Requirements Number*' column, used in the initial requirements draft, has to be matched with the '*Requirements Name*' and '*Requirements Descriptions*'. Continuous discussion and negotiation between the developers and client having taken place, the developers have removed *R02*, *R05*, *R06*, *R12* and *R13* from their first requirements draft (refer to Table 5.1) in order to produce the RAD.

No changes were made for the word '*all constraints*' in *R01*, and the developers have justified the constraints for the system by stating them in the RAD. In accordance to this decision, reference point for the word '*all constraints*' exist in *R01*. The word '*system*' in *R01* has been changed to '*computer system*' and similarly the word '*system*' in *R04* has been amended to '*computer system*'. The previous changes reflect that the inconsistent use of the word '*system*' may lead to ambiguity. Thus it was verified and corrected in RAD, leaving no room for potential assumptions pertaining to the word.

R09 and *R10* have been merged and rewritten as '*The user should not be able to edit the internal data. Administrators can approve new components to be available in the configuration and remove components that are obsolete*'. The removal of the words '*appropriate interface*' from the previous merged requirements, *R09* and *R10* shows that these words did not assist in clarify the requirement.

In the RAD, the requirements, *R03*, *R07*, *R08* and *R11* remained unchanged. New requirements *R14*, *R15*, *R16*, *R17* and *R18* were added to the RAD. There are no assurances that the previous requirements are free from the risk of propagating assumptions. For instance: the words '*appropriate mechanism*' still remains in the *R08* and are not well defined. Therefore probabilities of misconceptions remain due to the

use of natural language between developers and clients while the requirements are being developed.

A total of 11 assumptions, described in Appendix C-2 were uncovered from the observation made from the meetings. These assumptions were grouped as 'conscious or unconscious'. These categories are discussed in the 'Taxonomy of Assumptions made during Requirements Engineering' (TARE) in Section 3.4.3.

As an example, the client being unsure of certain design criteria for the webpage, had left the task of outlining the webpage to the developers. A conscious assumption was made by the client, *A03*, '*The client is assuming that the developers will be able to design the webpage correctly according to their judgments*' (refer to Appendix C-2). This statement allows the developers to work out the design themself without client's suggestions. However the designs, which will be produced later by the developers, might not be completely accurate; there is the possibility for the assumptions to remain until they are made explicit during the validation process for the webpage designs. The process of tracking the unconscious assumptions was challenging; it is not a trivial task to understand and define the unknown future knowledge. This remains the most dangerous type of assumptions in RE. Only one assumption, in *A05*, was detected from the observation: '*The developers assumed that the system needs to generate and display all the options for an item chosen by the user*'.

The assumptions gathered from the observation are potential misunderstandings between both the client and developers, and among the developers themselves. If not detected and verified during the early phases of development, these assumptions have the possibilities of leading to potential errors.

Approximately four out of eleven assumptions, which contributed to potential defect in the process of RAD development, were detected. These defects were reflected on the transition of the 'Initial Requirements' developed by Team *B* in Table 5.1 to Requirements Analysis Document depicted in Table 5.2. The requirements, *R14*, *R05*, *R15* and *R04* (listed in Tables, 5.1 and 5.2) were affected by the assumptions, *A05*, *A08*, *A10* and *A11* (described in Appendix C-2) respectively. Hence 36% of the assumptions found led to problems during requirements development. Even though only less than half of the uncovered assumptions contributed to problems, they influenced the quality of the system during the initial stage of project. This was proven when the developers had to restart the prototype design after realizing that there were certain mismatches between the client's needs and their design. The mismatches originated from several assumptions pertaining to the systems' logic and interfaces. A discussion was held between the client and only two developers to rectify the problems and brainstorm the new prototype.

Even though there were still potential assumptions existed in the development, the client signed off the project. The acceptance of the project by the client was dependent on the criteria where the client believed that the developers had outlined the necessary requirements for the system through Acceptance Testing. However, these assumptions may end up as potential defects in the future as the needs of the software changes.

5.1.2 Outcomes from Team A's project

Team *A* consisted of six members who opted to build a car-pool software for the use of staff and students of a university. Two members of the team represented the

client's company. This system needed to encompass a database system which enabled the users to locate others who wished to car-pool. The aim of the software was to allow the users to calculate the amount of savings to them that could be made through the car-pooling system. Other system components were also required by the clients.

The Requirements Analysis Document (RAD) developed by Team *A* is illustrated in Table 5.3. The RAD from Team *A* shows a similar pattern to that of Team *B*'s RAD; the usage of natural language is one of the factors which can mislead the development. For instance, the word '*keeping*' in the requirement '*Encryption*': '*Security in keeping with the main UWA web infrastructure*' was not defined clearly and can be misleading.

TABLE 5.3 – *Functional Requirements by Team A*

Requirements	
Name	Descriptions
Encryption	Security in ***keeping*** with the main UWA web infrastructure
Database	Storage of client records, accessible to the matching system
Matching System	Algorithms for providing matches between users.
Costing Module	Allows a user to determine how much money they save by carpooling, requires a map of the surrounding area to calculate distances
Webpage	Miscellaneous web content including links, maps of parking areas etc.
Registry	Registry to search engines allowing the page to be found from outside UWA

The assumptions gathered from Team *A*'s meetings were also caused by similar issues as found in Team *B*. Some of the clients' needs were not transparent. There were no explicit discussions regarding this matter. This was illustrated in *A02* as '*We will look at the progress of the webpage development as the project evolves. We have intentions to use the software permanently since there is a possibility for*

commercialization in the future. So, it has to look good' (refer to Appendix C-3). The developers and clients might have a different viewpoint of the word 'good' in *A02*, which has the possibility of leading to the occurrence of assumptions. The previous assumption belongs to the categories in TARE, 'Unconscious', 'Business' and 'Technical' (refer to Figure 3.6).

According to the observations gathered from the meetings, it is understood that the communication between the developers and the client of Team *A* was less efficient compared to Team *B*. Although this is a qualitative analysis, this fact appears true to an extent. During the meetings, the clients would describe their needs and request the developers to ask questions. However in some of the meetings held by Team *A*, the clients could not answer some of the developers' questions related to the system if the person in charge for that particular item was absence at the meeting. Alternatively if there were several developers not present at meetings, and a representative attempted to clarify their doubts with the clients, in these cases, in spite of minutes of the meeting being available, to which the developers absence during the meetings could refer, the message transmitted to them might distort the information which the clients wished to convey. A similar scenario was also illustrated in Figure 1.2. This type of scenario contributes to the propagation of assumptions during requirement analysis process.

In conclusion, using the qualitative analysis, Team *B* manage to communicate more efficiently and produce a better end results compared to Team *A* during the requirement analysis process. This was supported in the final grade obtained by each of the groups for the unit – Team *B* achieved better grade than Team *A*.

5.1.3 Lesson Learned from the Observations

There are two main factors which contribute to generation of assumptions: time and experience/knowledge.

It is human nature to be extra cautious if one is aware of the consequences of a particular scenario compared to the situation when there is no information about the consequences. It cannot be denied that if a team is well informed about the consequences of wrong assumptions during the early stages of development process, then the probability of them generating assumptions will be less compared to the situation where they are not informed. Therefore previous experiences and knowledge of managing issues pertaining to assumptions are essential for a development team to reduce the risk of propagating assumptions.

Time, as a factor, has a strong influence on propagating assumptions in software projects. If a development team have been assigned to build a system in a short duration, for instance four months, the probability to propagate assumptions might be high since the team must make decisions rapidly to move forward during the development process. On the other hand, if the team is aware of the short time period allocated for development, they may have a tendency to be more vigilant in making decisions to avoid assumptions. The team will not be likely to operate, based on assumptions (being corrected at a later stage) because they are aware of the time factor; not having sufficient time to correct invalid assumptions, which can lead to errors in the later part of the development process, they will be more vigilant to avoid assumptions.

5.2 Observations on Scenario Based Experiment

This section describes the data analysis for the SBExp. This step allows us to validate our experiment's hypotheses - previously described in Chapter Four as below.

(i) *A larger number of validated assumptions contribute to higher degree of accuracy in the requirements documents. If larger numbers of questions were asked by the participants, then, we expect to see a higher degree of accuracy in designing the use case diagrams.*

(ii) *Requesting participants to question assumptions aloud reduces the number of assumptions made.*

The aim of the experiment was to identify assumptions from a requirements document of a Mine Drainage Control System as described in Section 4.2.2. The participants were assigned to detect these assumptions based on a given scenario. We gathered the size of the assumptions or potential assumptions generated during this experiment by referring to the questions posed by the participants. These assumptions can be represented as the missing information required for developing the Mine Drainage Control System, described in Section 4.1.1.

5.2.1 Process for Data Analysis

Figure 5.2 illustrates the overview process for data analyses for the Scenario Based Experiment (SBExp). The task of the experiment was to allow the students to develop a UCD according to the information provided to them (refer to Appendix A-1). Students were expected to develop the UCD by clarifying their understanding of the Mine Drainage Control System (MDCS).

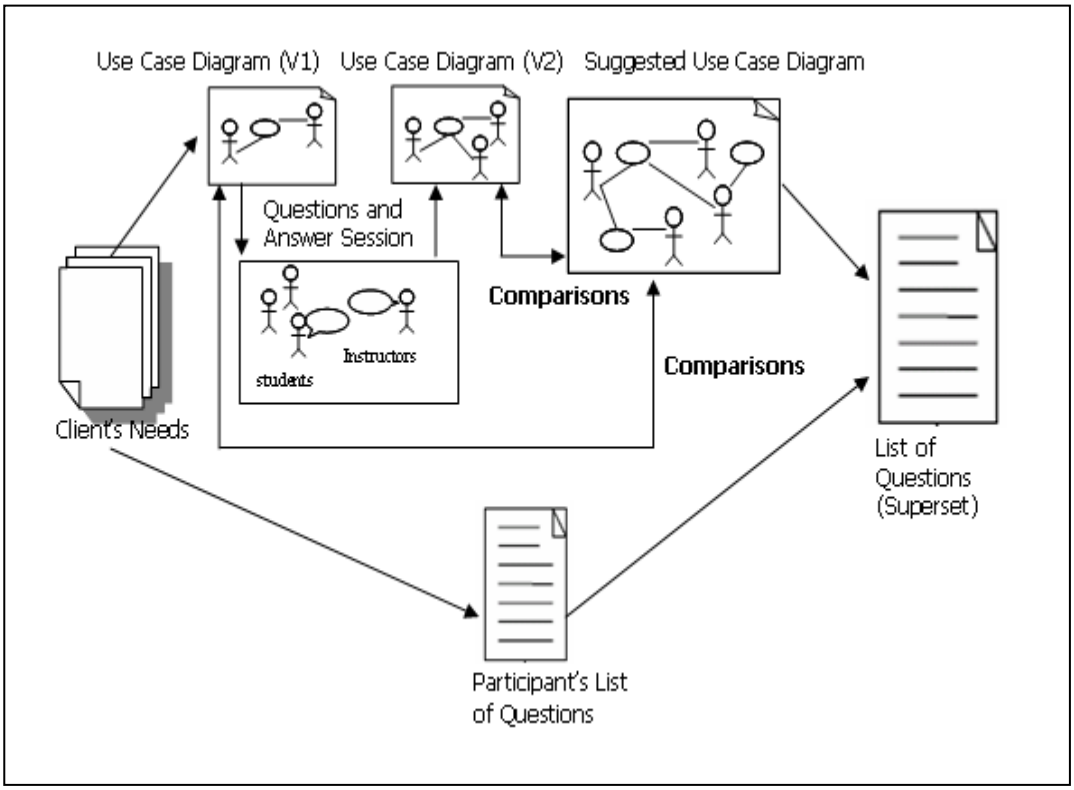


FIGURE 5.2 – *Overview of the Data Analyses for Scenario Based Experiment*

The accuracy of the developed UCDs was determined by comparing them with the 'Suggested UCD' (as shown in Appendix A-2). At the end, the questions posed by the students were translated into assumptions. This is actually the primary goal of the experiment: finalizing the number of assumptions captured by the students and how assumptions can influence the early stage of Requirements Engineering.

Figure 5.3 illustrates the detailed process for analysing the raw data accumulated from the experiment. After collecting the data from the students, it was segregated into two sections: the list of questions and the Use Case Diagrams (UCD). Prior to commencing the experiment, a checklist of questions had been prepared and it had been named as 'My Checklist of Questions' (represented as *Q* in Figure 5.3), depicted in Appendix A-3. *Q* was used as a reference to answer the questions posed by the students during the tutorial.

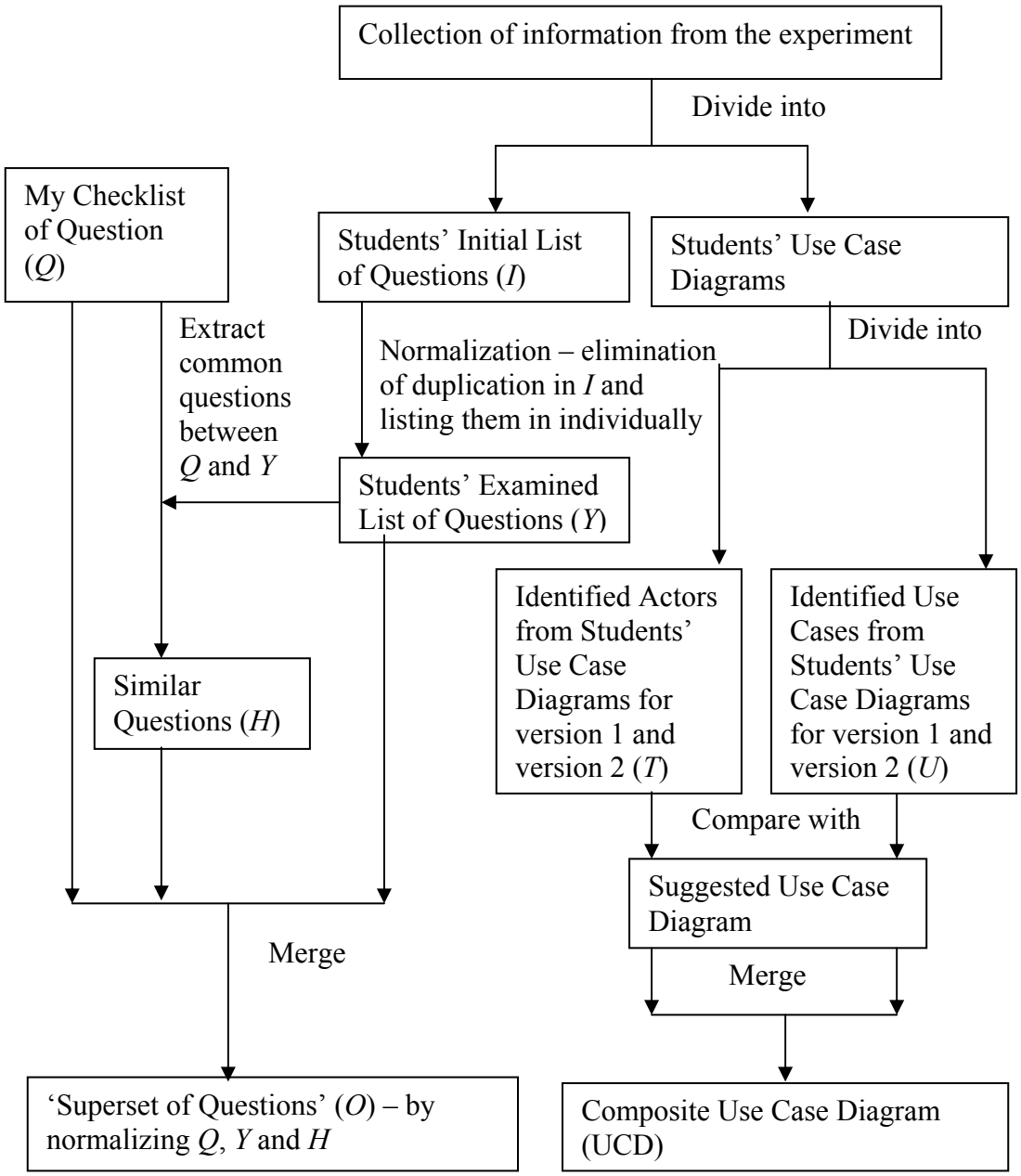


FIGURE 5.3 – A *Detailed Process Flow of the Data Analyses for SBExp*

All the questions gathered from the participants were listed. These questions were listed in a file which has been named as 'Students' Initial List of Questions' (represented as *I* in Figure 5.3), shown in Appendix A-4. The questions in *I* had been carefully inspected, and duplicate questions were removed. Those questions which consist of more than a single query were split and listed individually. In order to describe the previous steps, an example of questions extracted from *I* is depicted in Table 5.4.

TABLE 5.4 – Questions extracted from Students' Initial List of Questions (I)

No.	Questions	Answers
:		
12	What is a high water level?	Assume that the term 'high' is referring to a particular critical water level which will be specified later in the requirements documents.
13	What is a low water level?	Assume that the term 'low' is referring to a particular critical water level which will be specified later in the requirements documents.
:		
29	What are the actual values of "high" and "low" levels?	Assume that the term 'high' and 'low' is referring to a particular water level and methane level which will be specified later in the requirements documents.
:		
33	Who monitors the level of gases – the operator or another system?	The carbon monoxide sensor and methane sensor.
:		

Question 12 and *question 13* are similar to *Question 29* in the above table. Hence, these questions can be merged to form one single query. As for *Question 33*, must be split into two individual questions such as: 'Does the operator monitor the level of gases?' and 'Does another system monitor the level of gases?' This process contributes to 'Students' Examined List of Questions' (represented as *Y* in Figure 5.3).

Upon completing these steps, *Q* and *Y* were compared, and similar questions extracted. These similar questions are listed separately and named as *H*. Next, *Q*, *Y* and *H* are normalized to create a 'Superset of Questions' labelled *O* as shown in Figure 5.3 (as described in Section 4.2.2). In this experiment, the normalization process consists of merging the questions in *Q*, *Y* and *H* as illustrated in the Equation (3) and Figure 5.4.

$$O = Q \cup Y$$
$$O = (Q - H) + (Y - H) + H \quad (3)$$

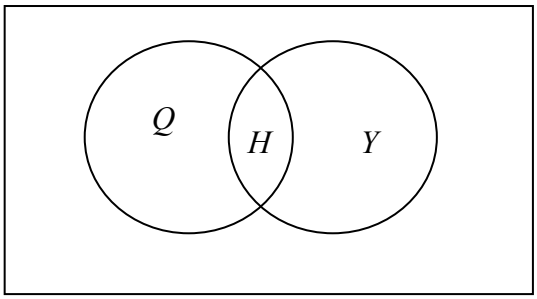


FIGURE 5.4 – *Representation of Questions collected upon Normalization*
(Q – My Checklist; Y – Students' Examined List of Questions; H – Similar Questions)

A brief description of the second process follows. The collected UCDs from the students were compared with the 'Suggested UCD' which was created prior to conducting the experiment as mentioned in Section 4.2.2. For the purpose of this experiment, the 'Suggested UCD' serves as a superset of the UCD. There will be two versions of UCDs collected from each group of students which are version 1 (V1) and version 2 (V2). V1 and V2 are referring to the UCDs before the 'Question and Answer' session, and after the 'Question and Answer' session respectively. As for the process of comparison, we had identified the actors in the UCDs of each group and recorded them in *T*. The process was repeated for the use cases and they were grouped in *U*. The reason for separating the actors and use cases found in the students' UCDs was to evaluate the number of actors and use cases discovered by each group. These details will be used to derive the accuracy of UCDs created by each group compared to the 'Suggested UCD'. Furthermore, this step will also simplify the process of combining the use case diagrams gathered from the students. Next, both *T* and *U* were merged to develop the 'Composite Use Case Diagram' which serves as the representative for UCD collected from the students. This merging process comprises the participated groups' understanding pertaining to the mine system which reflects in the production of the

single ‘Composite UCD’. This was used to compare with the ‘Suggested UCD’ in order to assess the degree of accuracy of the students UCD.

5.2.2 Distribution of Data Pertaining to UCD

The data relating to the UCD will be examined in this section. There were seven groups participating in the experiment which each took forty minutes to complete. These groups were labelled as *A*, *B*, *C*, *D*, *E*, *F* and *G* as described in Section 4.2.2.

Table 5.5 presents the data from *T* (the actors identified from the student’s UCDs). There were nine actors described in the ‘Suggested UCD’ which are ‘*Mine Operator*’, ‘*Pump Motor*’, ‘*Water Flow Sensor*’, ‘*Water Level Sensor*’, ‘*Methane Sensor*’, ‘*Carbon Monoxide Sensor*’, ‘*Air Flow Sensor*’, ‘*Data Logger*’ and ‘*Alarm*’. The actors identified by the groups were verified against these nine actors.

TABLE 5.5 – Actors identified in Version 1 and Version 2 (V1 and V2)

Groups / Actors	A		B		C		D		E		F		G	
	V1	V2	V1	V2	V1	V2	V1	V2	V1	V2	V1	V2	V1	V2
Mine Operator	√	√	√	√	√	√	√	√	√	√	√	√	√	√
Pump Motor			√	√	√								√	√
Water Flow Sensor		√											√	√
Water Level Sensor														
Methane Sensor													√	√
Carbon Monoxide Sensor														
Air Flow Sensor														
Data Logger		√												√
Alarm		√		√		√								√

TABLE 5.6 – Use Cases identified in Version 1 and Version 2 (VI and V2)

Groups / Use Cases	A		B		C		D		E		F		G	
	V1	V2	V1	V2	V1	V2	V1	V2	V1	V2	V1	V2	V1	V2
Request Motor Status														
Set Pump	√	√	√	√	√	√	√	√	√		√	√	√	√
Check Methane Level					√	√								
Monitor Water Level	√	√	√	√	√	√	√	√	√	√	√	√		√
Data Log														√
Check Carbon Monoxide Level					√	√								
Check Air Flow														
Set Alarm								√						√

There were eight use cases according to the 'Suggested UCD' and these are '*Request Motor Status*', '*Set Pump*', '*Check Methane Level*', '*Monitor Water Level*', '*Data Log*', '*Check Carbon Monoxide Level*', '*Check Air Flow*' and '*Set Alarm*'. The groups have managed to identify the use cases as shown in Table 5.6. The information in Table 5.6 presents the data from *U* (the use cases identified from the student's UCDs).

The data in Table 5.5 and Table 5.6, which depict the UCDs designed by the students, showed that none of the groups had managed to identify the actors: '*Water Level Sensor*', '*Carbon Monoxide Sensor*' and '*Air Flow Sensor*'. For the use cases, none of the groups had identified '*Request Motor Status*' and '*Check Air Flow*'. It was expected that there might not be any questions posed by the students regarding actors and use cases: '*Water Level Sensor*', '*Carbon Monoxide Sensor*', '*Air Flow Sensor*', '*Request Motor Status*' and '*Check Air Flow*'. Hence, the lack of information pertaining to the actors and use cases did not allow the students to identify them.

However, we discovered that students did enquire about two of the missing actors which were '*Water Level Sensor*' and '*Carbon Monoxide Sensor*'. They also enquired about '*Request Motor Status*' and the relevant questions for both the actors and use cases shown in Table 5.7.

TABLE 5.7 – Questions extracted from Students' Initial List of Questions (X)'

No.	Questions	Relevancy
: :		
8	How will the pump be able to monitor the water level in the sump?	Water level sensor
: :		
11	Are there any other functions for an operator besides operating the pump (turning on the pump when the water reaches high level and the water is drained until it reached low level)?	Request Motor Status
: :		
22	How can the information about the level of carbon monoxide emitted from mining be obtained by the system?	Carbon monoxide sensor
: :		
33	Who monitors the level of gases – the operator or another system?	Carbon monoxide sensor and methane sensor.
: :		

On the other hand, the students did not comprehend the need for the '*Air Flow Sensor*' and '*Check Air Flow*' as no questions were noted from students pertaining to these actors and use cases. This lack of questions will be classed as 'unconscious assumptions' from this scenario. Despite the fact that the students managed to find out relevant information with regards to some actors and use cases, this scenario revealed that they still failed to identify others. The rationale is that the information which they

had found through their questions was assumptions and that they believed these assumptions to be true in the process of developing their UCDs. These assumptions might not have contributed the correct information that they needed to design the accurate UCD and this therefore, could result in false assumptions being barrier for developing a correct system (which in our experiment was the UCDs).

5.2.3 Accuracy of the UCDs Developed by the Participants

For the purpose of the experiment, the 'Suggested UCD' (as shown in Appendix A-2, also see Section 4.2.2) was defined as the most complete and correct Use Case Diagram for the mine drainage system. This document serves as the "reference" document against which the experimental data will be judged. In order to measure the accuracy of each group's UCD, the UCDs generated by the groups in Version 1 and Version 2 were compared against the 'Suggested UCD'. The accumulated number of actors and use cases in the 'Suggested UCD' is 17. The total number of actors and use cases identified in Version 1 and Version 2 of the UCD are added for each group and compared against the expected value of 17, in order to derive the accuracy of each group's UCD. This is listed in Table 5.8.

TABLE 5.8 –Accumulation of Actors and Use Cases for SBExp

Descriptions	Student Group						
	A	B	C	D	E	F	**G**
Version 1	3	4	6	3	3	3	**5**
Version 2 (new contribution)	**3**	**1**	**1**	**1**	**0**	**0**	**5**
Total actors and use cases	**6**	**5**	**7**	**4**	**3**	**3**	**10**
The accuracy of the developed UCD	35%	29%	41%	24%	18%	18%	**59%**

For instance, the accuracy of the UCD developed by group *A* was calculated as (6/17) * 100% or 35%. So group *A* can be interpreted as having developed a UCD which was 35% accurate and complete compared to the 'Suggested UCD'.

According to the data illustrated in Table 5.8, group *G* managed to produce the most accurate UCD with 59% of the requirements given. In fact, when the UCD of each group was examined group *G* was seen to have managed to design a UCD was is closest to the 'Suggested UCD'. Therefore the total number of actors and use cases discovered by the students is understood to be important in designing an accurate UCD. The group which asked the highest number of questions related to the UCD development was expected to produce the UCD with the highest accuracy. In order to verify this, it was necessary to examine the questions collected from the students; these provided insights of the students' understandings for the mine drainage system.

5.2.4 Analysis on the Gathered Questions

The number of questions found in the 'My Checklist of Questions' generated prior to the execution of the experiment was 35 (the value for *Q* in Figure 5.3). The initial number of questions identified by students was 63 (the value for *I* in Figure 5.3). The previous number was reduced to 50 (the value for *Y* in Figure 5.3) after removing the duplicate questions. The final number of questions derived from the experiment upon normalization was 65 (the value for *O* in Figure 5.3). Using Equation (3) from Section 5.2.1,

$$O = (Q - H) + (Y - H) + H$$
$$O = (35 - 20) + (50 - 20) + 20$$
$$= 65$$

The number of duplicated questions can be obtained manually when eliminating and segregating the questions. Value *H* mainly depicts the degree of common or redundant understanding among the students. If the Scenario Based Experiment were to be executed among a set of participants with different background and knowledge, it would be possible to compare how the *H*' (or new value of *H*) could differ with value *H* of the present experiment. It may be possible to discern some patterns or correlations between knowledge (from types of participants) and the level of redundant understanding from the participants.

The questions accumulated in the 'Superset of Questions' or *O* were segregated according to the relevancy or type of question related to the Mine Drainage Control System (MDCS). The categories are divided as below:

- 'Question relevant to actors' represented as '$\boldsymbol{\alpha}$',
- 'Question relevant to use cases' represented as '$\boldsymbol{\beta}$',
- 'Question relevant to both use cases and actors' represented as '$\boldsymbol{U}$',
- 'Question not relevant to both actors and use case but relevant to MDCS' represented as '$\boldsymbol{\eta}$'; and
- 'Question not relevant to actors, use cases and MDCS' represented as '$\boldsymbol{\lambda}$'.

Table 5.9 consists of the number of questions for types or categories as described above for each group. The 'Total missed questions' in the Table 5.9 refers to the information which the students have missed during the development of the UCDs.

TABLE 5.9 – Number of questions gathered according to categories

Questions according to Categories	**Student group**						
	A	**B**	**C**	**D**	**E**	**F**	**G**
Questions not relevant to MDCS (***λ***)	0	0	0	0	0	0	0
Questions relevant to Actors only (***α***)	3	3	1	3	2	0	0
Questions relevant to Use Cases only (***β***)	1	1	1	4	1	0	1
Questions relevant to both Actors and Use Cases (***U***)	2	3	2	3	3	4	3
Questions neither relevant to Actors nor Use cases but Relevant to MDCS (**η**)	4	2	1	1	3	9	2
*Questions relevant to UCD (**R = α + β + U**)*	*6*	*7*	*4*	*10*	*6*	*4*	*4*
Total missed questions (**δ = 65 - *R***)	59	58	61	55	59	61	61

The data for 'Total missed questions' were derived using the equation below.

$$R = (\alpha) + (\beta) + (U)$$

$$\delta = \#\,(\text{Superset of Questions}) - R \qquad (4)$$

where 'R' represents Questions relevant to UCD for each group; 'δ' represents Total Missed questions and '#' represents the cardinality of the set

For the purpose of this experiment, the 'Total missed questions' have been used to quantify the amount of missing information which is needed by each group to develop their UCD. The principle defined in Section 4.3 is used to derive the number of assumptions made by the students is shown below.

Level of Assumptions $\propto$ *Missing Information*

$N \propto$ *# (Set of missing questions)*

where 'N' is the number of assumptions made

$\therefore N = k$ *# (Set of missing questions)*

where 'k' is a constant of proportionality

$\therefore N = k\,\delta$
$= k$ (*# (Superset of Questions)* $- R$)

The collected questions consist of the information which is needed by the students in order to complete the task assigned to them in the experiment which was developing the UCD. If the students were not given the opportunity to verify their doubts, then these doubts (questions) have a high probability of turning into decisions which the students need to assume to be true. Therefore the 'Total missed questions' was interpreted as the measure of potential assumptions made by the students for the purpose of developing the UCD.

There were also sets of questions gathered from the experiment which were not relevant to UCD development but relevant to the mine drainage system. This data set is denoted by $\boldsymbol{\eta}$. Although these questions did not influence the UCD development, they contributed to the process of understanding the overall mine drainage system. The students did not pose any questions which were not relevant to the mine drainage system, represented by $\boldsymbol{\lambda}$. This indicates that the student were aware of the issues included in the task assigned to them in the tutorial.

5.2.5 The Potential Assumptions' Influences

In this section the potential assumptions gathered during the SBExp and impacting the quality of the UCD produced by the students will be examined. This can be further explained by using several observations as follows.

5.2.5.1 Results for Observation 1

The size of the potential assumptions made by the students was derived from the number of questions missed by the students during the UCD development. Figure 5.5 shows that the lowest number of potential assumptions originated from group *D* which was 55. Group *D* managed to verify more doubts pertaining to the UCD. Thus group *D* was expected to produce a UCD which would be the most accurate UCD, compared to the other groups (refer to the accuracy of the UCDs generated by all the groups in Table 5.8). However group *D* did not produce the most accurate UCD. Although the questions asked by group *D* was valid, the questions might not be high quality to enable group *D* to design the most accurate UCD. Therefore, the poor quality of the questions asked by group *D* was one of the possible factors affecting the accuracy of the UCD being produced.

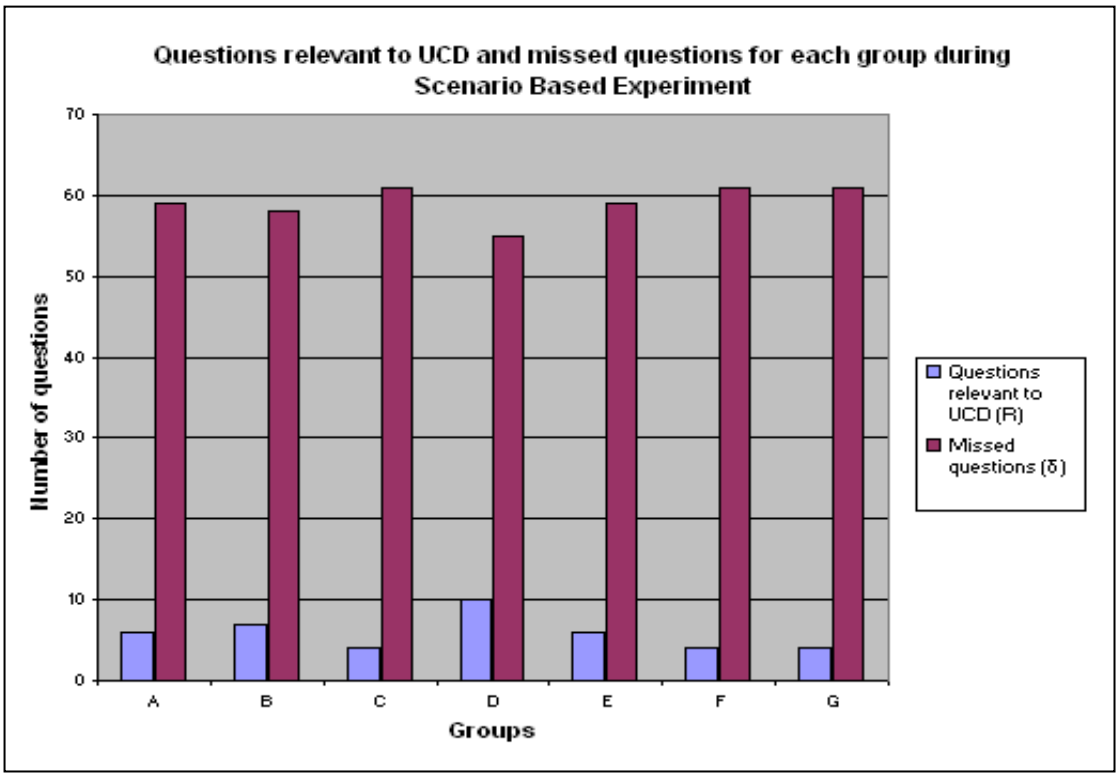


FIGURE 5.5 – *Questions relevant to UCD and missed questions*

On the other hand, the highest numbers of potential assumptions comes from groups *C*, *F* and *G* which were 61. Although these groups only managed to verify minimum number of assumptions, the UCD generated by one of these three groups, group *G* was the most accurate UCD compared to the other groups.

There is no sufficient evidence to support the argument that the number of assumptions dictates the accuracy of the UCD developed by the participants. This could be because there are other factors which may influence the UCD development besides assumptions. Therefore the number of potential assumptions relevant to the matter which is being verified could not be used as sole factor (or in this case, as the only variable) to predict the quality of the UCD being produced.

The statement in the previous paragraph can be supported by comparing the expected model for the relationship between assumptions and quality of the UCD with

the outcomes gathered from this experiment. For the purpose of this experiment, the quality of the UCD is determined by the accuracy of the UCD. The 'Suggested UCD' was used as the benchmark to obtain the accuracy of the UCD developed by the students.

> *The quality of the UCD developed by the students is proportional to the accuracy of their UCD.*

Thus, the expected model is a negative relationship between the two variables, quality of the UCD and number of assumptions. The quality of the UCD is expected to decrease as the assumptions (number of missed questions) increase.

However, the result from our experiment, which is depicted in Figure 5.6, differs from the expected model. The two variables shown in Figure 5.6 have a non-negative relationship (slope). Table 5.10 represents the information in Figure 5.6.

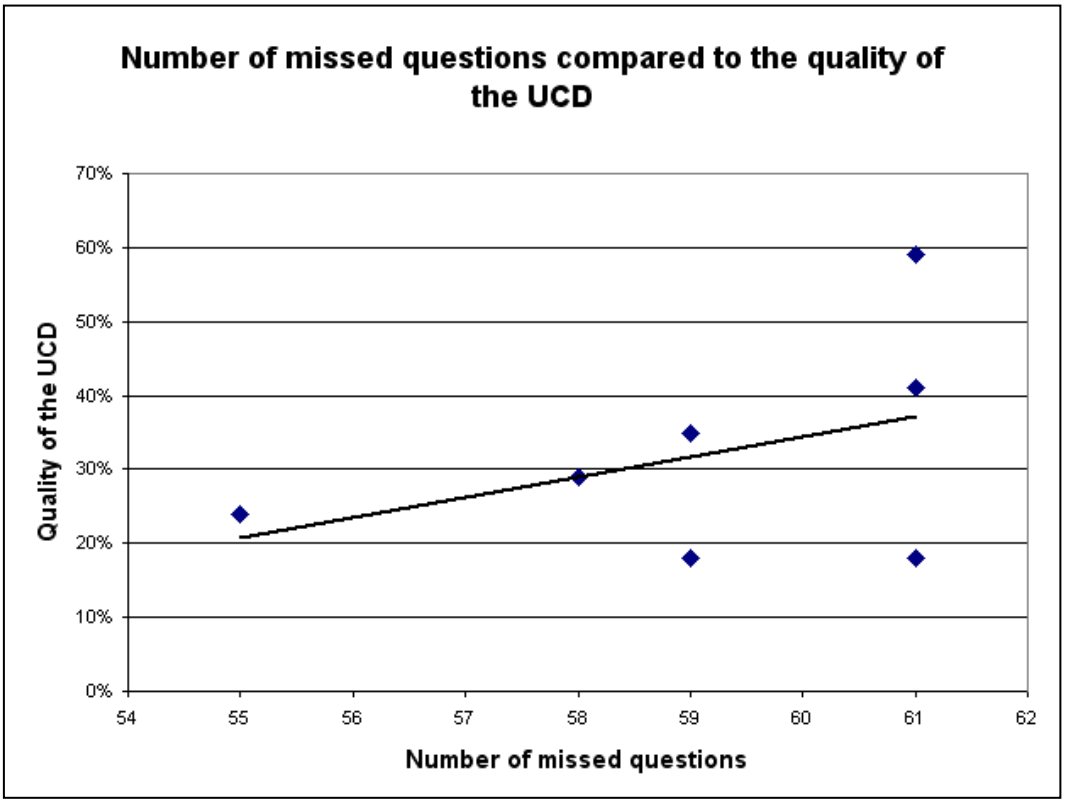


FIGURE 5.6 – *Number of missed questions and UCD's quality*

TABLE 5.10 – Number of missed questions compared to the quality of the UCD

Descriptions	Student Group						
	A	B	C	D	E	F	G
Number of Missed Questions (δ)	59	58	61	55	59	61	61
The accuracy or quality of the developed UCD (%)	35	29	41	24	18	18	59

The significance of the graph (shown in Figure 5.6) will be reviewed before comment is made on the relationship between the two variables. How established was the interrelationship between the variables in this experiment? The Standard Error of Estimate was used to measure the accuracy of the graph [89]. For the purpose of this calculation, we define the 'Number of Missed Question' as *Y* and 'The accuracy of the developed UCD' as *X*. The Standard Error of Estimate for accuracy of the developed UCD,

$$S_{Y.X} = 0.12$$

However there are an insufficient number of points to be able to make any claims about the distribution of the variables, normal or otherwise. Even though the number of points was insufficient to distinguish clear claims, the SBExp were not repeated due to time constraint. The participants of this experiment were students enrolled in Software Quality and Measurement (CITS4220) unit. The waiting period to conduct the second set of SBExp involving the students from CITS4220 will be six months. Although it is worth for further experimentation, it is not viable to extend the research for another six months.

The graph in Figure 5.6 is not statistically significant. Hence there is insufficient evidence to be able to make a clear statement about the relationship between the variables. From these results it cannot be claimed that the number of assumptions dictates the accuracy of the UCD developed by the participants.

5.2.5.2 Results for Observation 2

In addition, Figure 5.7 (which depicts the data in Table 5.11) did not illustrate satisfactory evidence to support the argument that the number of questions related to the UCD does not guarantee the accuracy of the UCD being generated.

TABLE 5.11 – Number of questions relevant to UCD

Descriptions	**Groups**						
	A	**B**	**C**	**D**	**E**	**F**	**G**
Improvement in UCD	3	1	0	1	-1	0	5
Questions relevant to UCD	6	7	4	10	6	4	4

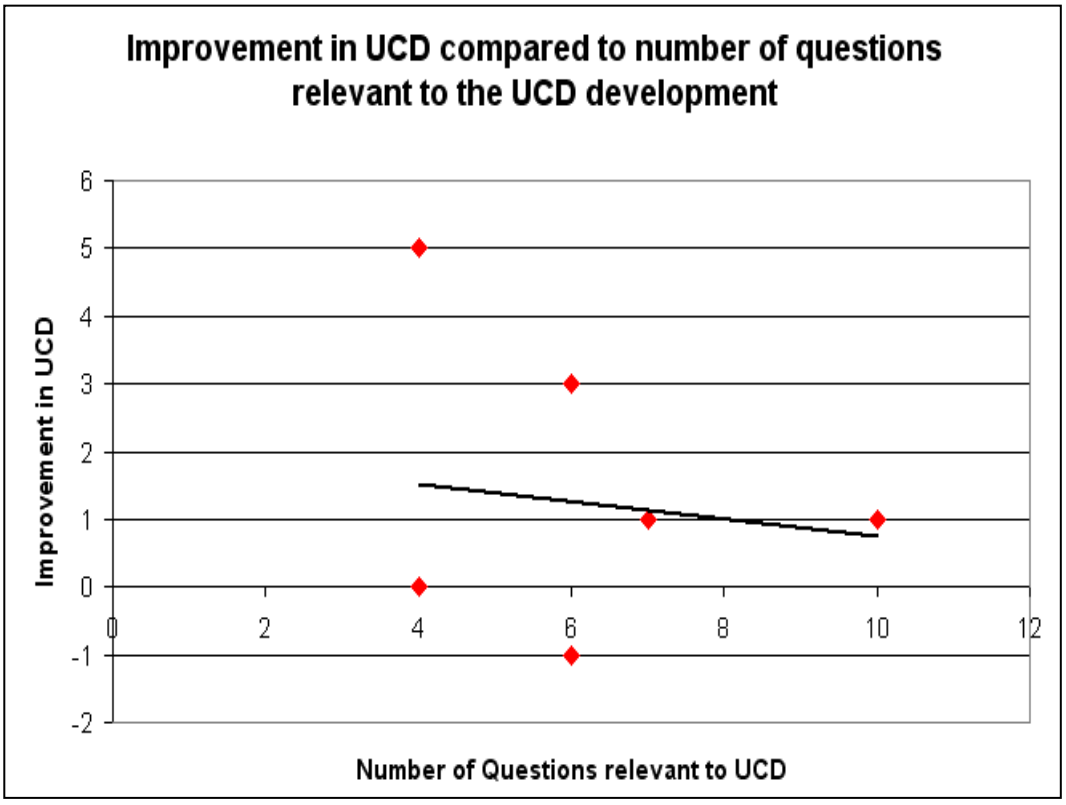


FIGURE 5.7 – *Number of questions relevant to UCD and UCD's improvement*

The value for 'improvement in UCD' was derived by subtracting the accumulation of actors and use cases in Version 1 from Version 2 as shown below.

$I = A_2 - A_1$

where 'I' represents the Improvement in UCD; 'A_2' represents Accumulation of Actors and Use Cases in Version 2 and 'A_1' represents Accumulation of Actors and Use Cases in Version 1

Group *D* with 10 questions relevant to the UCD only managed to achieve a minor improvement in the UCD. However, group *G*, which is one of the groups which asked the lowest number of questions with regards to the UCD managed to achieve the highest improvement for the UCD. There is a possibility that group *G* has asked more relevant questions pertaining to the development of the UCD compared to the other groups. This has allowed group *G* to achieve the highest improvement for the UCD.

Using the same method used in Section 5.2.5.1, the Standard Error of Estimate for accuracy of the developed UCD,

$S_{Y.X} = 1.89$

There is an insufficient number of points to be able to make any claims about the distribution of the variables – Normal or otherwise. Therefore, the weak correlation between the variables (shown in Figure 5.7) indicates that there are factors other than assumptions which influence the development of the UCD. These include the students' past experience and the limited time given to complete the tasks.

As a result of Observations, 1 and 2, the conclusion was as below:

> *There was inadequate evidence to make a clear statement about the correlation between the qualities of the UCD produced by the students with the number of assumptions made by them during the development of the UCD.*

However, the quality of the UCD appears to be related to the validity of the assumptions which were used to develop the UCD rather than the size of assumptions as mentioned in our experiment hypothesis.

This segment of the experiments explains that by being aware or clarifying doubts on (conscious) assumption made during the initial stages of Requirements Engineering, is not sufficient to guarantee a satisfactory outcome for the RE process. In order to solve this dilemma, further investigation is required.

5.2.5.3 Results for Observation 3

Data on the types of questions posed by each group during the experiment were examined. This data (as illustrated in Appendix A-4) consists of redundant questions whereby one questions can be referred by more than one group. However, in order to retrieve the value for the questions revealed in the experiment, the questions, relevant to UCD development asked by the students without referring to the individual groups, are further analysed. This data is depicted in Table 5.12 and the value for each of the categories derived by referring to Section 5.2.4.

TABLE 5.12 – Number of questions gathered from Scenario Based Experiment

Questions according to categories	Questions	
	Amounts	Percentages
Questions Not relevant ($\eta + \lambda$) to UCD development	19	29
Questions Relevant (R) to UCD development	25	39
Total Missed Questions (δ)	21	32
Total	65	100

The above table shows that 39% (derived by 25/65 * 100%) were questions relevant to the UCD development while 29% (derived by 19/65 * 100) of the questions were not relevant to the development of the UCD. These percentages represent the amount of information gathered by the students which enabled them to design their use case diagrams. Hence, we consider that 39% of the information discovered by the students represents the amount of information gathered by the students which enabled them to design their use case diagrams. It is hoped that the students used 39% of relevant information for UCD development in order to design the 'Composite UCD' as illustrated in Appendix A-6.

The 'Composite UCD' should resemble a UCD that which encompasses and merges the use case diagrams from all the groups. The accuracy of the 'Composite UCD' can be computed in the same manner which is used to derive the accuracy of each group's UCD as illustrated in Section 5.2.3. The accumulated number of actors and use cases in the 'Composite UCD' was 11. Thus the accuracy of the 'Composite UCD' was 11/17 or 65% compared to the 'Suggested UCD' where 17 refer to the total number of actors and use cases in the 'Suggested UCD'. A linear relationship between amounts of information needed to develop the UCD with the accuracy of the UCD was expected. This suggests a result whereby the students were expected to attain 58% of information related to the UCD in order to design a UCD similar to the 'Suggested UCD'.

However data presented in Table 5.13 showed that this expectation was incorrect. The details in Table 5.13 were derived by merging the analysis done in Section 5.2.3 and data illustrated in Table 5.8. It was expected that the group which manages to obtain 58% of information relevant to UCD would be able to produce a UCD similar to 'Suggested UCD'. But according to the illustration in Table 5.13, group *G* which designed a UCD with 59% of accuracy only discovered a small amount of information related to UCD, 6%. In addition, other groups as well did not gather information close to 58% in order to generate their UCD with respective accuracies.

TABLE 5.13– The relevancies of questions with the accuracy of UCD

Descriptions	Student Groups						
	A	B	C	D	E	F	G
Questions relevant to UCD	6	7	4	10	6	4	4
Relevancy of the questions pertaining to UCD development (%)	9	11	6	15	9	6	6
The accuracy of the developed UCD (%)	35	29	41	24	18	18	59

Hence, the observation showed that a linear relationship cannot be applied between the information needed to develop UCD and the accuracy of the UCD. Further this observation also reemphasizes the fact that there is insufficient evidence to determine the quality of the UCD development by only considering the level of information gathered (or assumptions clarified). There are other factors which can contribute to the quality of the UCD development besides assumptions.

5.2.5.4 Results for Observation 4

The number of missed questions (or assumptions) is shown in Table 5.14. For the purpose of this experiment (as explained in Section 4.2.2), the 'Superset of Questions' is represented as the "reference" document against which the experimental results will be judged. The document consists of all the questions (assumptions) needed by students to verify in order to complete the experiment tasks. However, in reality, the size of potential assumptions raised during the Requirements Engineering Process was not able to be realised. The limitation of the size of the assumptions will merely depends on the scope of the RE process.

TABLE 5.14 – Number of missed assumptions from the 'Superset of Questions'

Descriptions	Questions	
	Amounts	Percentages
Questions (or potential assumptions) verified by the students	44	68
Questions (or potential assumptions) missed by the students	21	32
Total	65	100

The results of the experiment showed that the number of missed questions (or potential assumptions) was 32%. There were 21 potentials assumptions which students did not verify. Although, 68% of the assumptions from the 'Superset of Questions' were verified, this did not guarantee the development of an accurate UCD from the students.

The results gathered from Observations, 3 and 4 states as follows.

> *There was insufficient evidence to show that the quality of the UCD, produced by the students, correlates with the number of potential assumptions relating to the development of the UCD as verified by the students.*

The statement above supported the previous conclusion in Observations, 1 and 2 in Section 5.2.5.2. There may be other factors which contribute to the UCD development besides the number of assumptions made by the students.

The outcome of verification process might allow the students to absorb either a valid or invalid assumption into the process of their UCD development. Therefore the quality of the developed UCD would depend on the quality of assumptions made by the students pertaining to the UCD development, which can be projected by the number of valid assumptions. The validity of these assumptions can only be justified by interrogating the students on their list of assumptions.

5.2.6 Estimation for the Number of Iterations

Table 5.15 (a) and (b) represents the total number of actors and use cases accumulated in Version 1 and Version 2 of the experiment respectively.

TABLE 5.15 – Summary of Actors and Use Cases identified by the Students

(a) Number of Actors

Descriptions	Groups							Total for each version
	A	B	C	D	E	F	G	
Version 1	1	2	2	1	1	1	4	12(/63)
Version 1 + Version 2	4	3	3	1	1	1	6	19(/63)

(b) Number of Use Cases

Descriptions	Groups							Total for each version
	A	B	C	D	E	F	G	
Version 1	2	2	4	2	2	2	1	15(/56)
Version 1 + Version 2	2	2	4	3	2	2	4	19(/56)

The data illustrated in Table 5.15 had allowed the consideration of the number of sessions that might be required by the students in order for them to produce an accurate UCD (which is the 'Suggested UCD' in our experiment). In the ideal case, the total number of actors and use cases need to be identified by all the 7 groups. This would be 63 (derived by (7 * 9) where each group identifies 9 actors) and 56 (derived by (7 * 8) where each group identifies 8 use cases) accordingly. These details are represented in the vertical column named 'Total for each version' in Table 5.15 (a) and (b).

As illustrated in Figure 5.2, the students had been given two chances to derive their UCD. First, they were required to draft the first version of their UCD by referring to the limited information (client's requirements) provided to them (refer to Appendix A-1). Next they had to develop the second version of their UCD after the question and answers session where they were given the opportunity to clarify any doubts. So, there

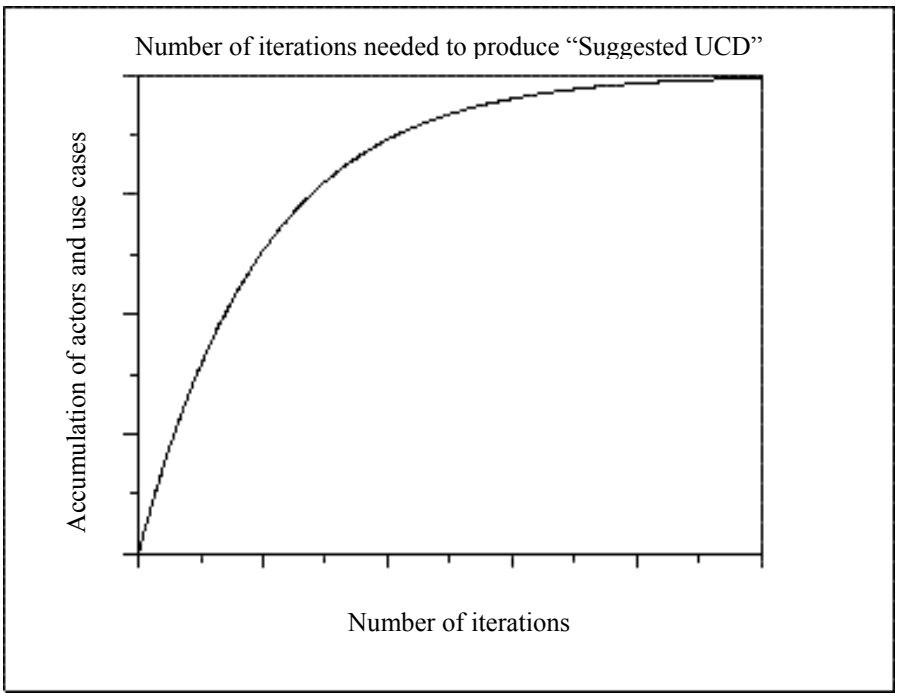
were actually two iterations used to obtain the final product in this experiment, the student's UCD. Therefore next the number of sessions (or iterations) was noted that the students needed to design an accurate UCD (which will be referred to as 'Suggested UCD' in our experiment).

TABLE 5.16 – Summation for number of Actors and Use Cases

Versions	Groups							Summations of actors and use cases, ∑	Average of ∑
	A	B	C	D	E	F	G		
Version 1	3	4	6	3	3	3	5	27	27/7 = 3.86
Version 1 + Version 2	6	5	7	4	3	3	10	38	38/7 = 5.43
: :			: :					: :	: :
Version n								**119**	**119/7 = 17**

We had taken the average for the summation of actors and use cases for each group, ∑ and their respective averages as depicted in Table 5.16. Ultimately, the ideal number of actors and uses cases expected to be discovered by the students from the seven groups will be 119 (summation of 63 actors and 56 use cases) divided by seven groups. These previous values have been listed to help in the process of estimating the number of iterations required by the students in order to produce the 'Suggested UCD'. Therefore there will be '*n*' iterations needed in order to develop the 'Suggested UCD' is defined. The value for '*n*' can be derived by using a cumulative model.

The expected result of the cumulative model from our experiment should be as Figure 5.8 where we predicted the value for summation of actors and use cases to increase as the number of versions increase.



(Taken from [90])

Using the total number of actors and use cases identified in two iterations (versions) as depicted in Table 5.16, we can estimate the number of iterations needed by the students in order to generate the "Suggested UCD". This is achieved by using a common function with the appropriate shape given by the formula below.

$$y = ae^{-bt} \qquad (5)$$

First, we derive the above model with an assumption that the students will discover a given proportion of the problem in each of the iteration. This is also known as learning curve. The amount of new materials uncovered by the students is given as a proportion either of the remaining problem which need to be solved or the amount discovered in the previous iteration (version).

$$Thus\ y_t = \omega\, y_{t-1} = \omega^2\, y_{x-2}$$

We derive the equation as below.

$$y_t = \omega^t\, y_0 \qquad (6)$$

by setting $a = y_0$ *and* $b = -\ln \omega$ *which gives* $y(t) = ae^{-bt}$

Note that this assumes $0 \leq \omega < 1$ (For instance, $0 \leq y_t < y_{t-1}$)

Second, using the function in Equation (5), we obtain the cumulative amount learnt as illustrated below.

$$Y(T) = \int_0^T y(t)\, dt$$

$$= \left(-\frac{a}{b}\, e^{-bt} \right)_0^T$$

$$Y(T) = \frac{a}{b}\left(1 - e^{-bT}\right) \qquad (7)$$

From Table 5.16, we have two averages for values of the total number of actors and use cases for iteration 1 (Version 1) and iteration 2 (Version 2) respectively.

$Y(1) = 3.86$

$Y(2) = 5.43$

Third, using the above values, two equations were derived consisting of two unknowns as follows.

$$Y(1) = \frac{a}{b}\left(1 - e^{-b}\right) = 3.86 \qquad (8)$$

$$Y(2) = \frac{a}{b}\left(1 - e^{-2b}\right) = 5.43 \qquad (9)$$

Fourth, the value for '*b*' was found by dividing Y(2) with Y(1).

$$\frac{Y(2)}{Y(1)} = \frac{5.43}{3.86}$$

$$\frac{\frac{a}{b}\left(1 - e^{-2b}\right)}{\frac{a}{b}\left(1 - e^{-b}\right)} = \frac{5.43}{3.86}$$

$$\frac{(1 - e^{-b})\,(1 + e^{-b})}{(1 - e^{-b})} = \frac{5.43}{3.86}$$

$$1 + e^{-b} = \frac{5.43}{3.86}$$

$$e^{-b} = \frac{5.43}{3.86} - 1$$

$$e^{-b} = 0.41$$

$$b = 0.89$$

Now, the value of '*b*' is replaced in Equation (8) to obtain value '*a*'.

$$\frac{a}{0.89}\left(1 - e^{-0.89}\right) = 3.86$$

$$a = \frac{(3.86)\,(0.89)}{\left(1 - e^{-0.89}\right)}$$

$$a = 5.83$$

Thus, the asymptote value for this function is

$$\frac{a}{b} = \frac{5.83}{0.89} = 6.55$$

Lastly, the cumulative model is given by the function as below.

$$Y(T) = 6.55\left(1 - e^{-0.89T}\right) \qquad (10)$$

Table 5.17 depicts the average values of ∑ (or summation of actors and use cases) for the respective iterations.

TABLE 5.17 – Average values for ∑ in each iteration

(∑ is summation of actors and use cases)

Versions /Iterations (T)	Average of ∑
0	0.00
1	3.86
2	5.45
3	6.10
4	6.36
5	6.47
6	6.52
7	6.54
8	6.55
9	6.55
10	6.55

This gives a cumulative curve as in Figure 5.9. This result indicates that the students will require eight iterations in order to derive the 'Suggested UCD'. This is an estimated figure derived from the calculation for number of iterations needed by the students. However, the students would have only detected an average of seven for summation of actors and use cases – compared to the average of 17 detected by all the groups in the perfect document.

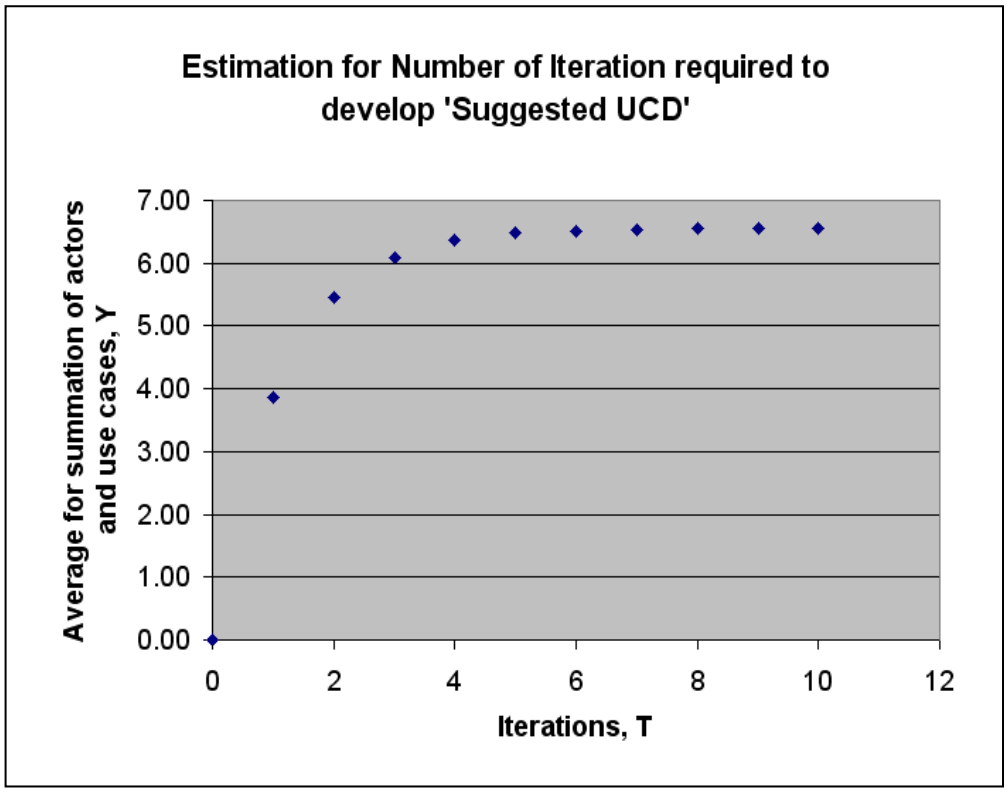


FIGURE 5.9 – *Estimated number of iterations to generate the 'Suggested UCD'*

The computation for the number of estimated iteration can also be derived using the method as below.

$$\textit{The asymptote} = \textit{Limit}_{T \to \infty} \left(\frac{a}{b} \left(1 - e^{-bT} \right) \right) = \frac{a}{b}$$

$$\textit{and } \frac{a}{b} = \frac{b\, Y(1)}{(1 - e^{-b})\, b}$$

$$\textit{but } e^{-b} = \frac{Y(2)}{Y(1)} - 1 = \frac{Y(2) - Y(1)}{Y(1)}$$

$$\textit{Hence, asymptote} = \frac{a}{b} = \frac{Y(1)^2}{(2Y(1) - Y(2))} \qquad (11)$$

In Table 5.16, the average values have been taken for total number of actors and use cases. Actually, the estimated number of iteration should not be derived from the average for all the groups, rather the calculation should be done for each group separately. For instance, using Equation (11), we can work out the number of iterations required by the individual group as below.

Group A : Y(1) = 3, Y(2) = 6

Therefore, $\frac{a}{b} = \frac{3^2}{(2*3) - 6} = \infty$

Group B : Y(1) = 4, Y(2) = 5

Therefore, $\frac{a}{b} = 5.33$

Group C : Y(1) = 6, Y(2) = 7

Therefore, $\frac{a}{b} = 7.20$

Group E : Y(1) = 3, Y(2) = 3

Therefore, $\frac{a}{b} =$ *3 (they did not find any additional actors and use cases)*

However the cumulative model in Equation (11) was not applied to work out the number of iterations for each group because groups, A, E, F and G did not satisfy the assumption used to derive the formula which is as below.

$0 \leq \omega < 1$ (For instance, $0 \leq y_t < y_{t-1}$)

Therefore the average values for total number of actors and uses cases for overall groups was taken to estimate the number of iteration needed to develop the 'Suggested UCD'.

5.3 Observation on Early Pilot Trial

The data collected from Assumptions Seeding Experiment were expected to reveal the number of assumptions in a Requirements Specification Document (RSD) using assumptions seeding method. This technique is elaborated in Section 4.2.3.3.

A trial experiment as described in Section 4.2.3.1 was executed before this experiment, using a larger group involving 12 participants. There were four participants selected for the pilot study, Subject *1*, *2*, *3* and *4*. The results gathered from these subjects are illustrated in Table 5.18.

TABLE 5.18 – Observation on the results collected from the pilot trial

Subject	Duration to complete the experiment	Working Experience (Academic/ Industry)	The amount of		
			Use cases and actors identified	Seeded Assumptions	Non Seeded Assumptions
1	30 minutes	Academic	5	0	0
2	120 minutes	Industry	6	3	3
3	45 minutes	Academic	4	2	5
4	60 minutes	Not available	4	0	4

The participants' comments were of the interested rather than the data, mainly to improve the experimental methodology. Despite having a small group of participants taking part in the pilot trial, comments to improve the design of the experiment were gained.

According to the feedbacks from these subjects, the subjects' background was understood to be an important criterion that needed to be considered. It was realized the majority of the future participants would encounter problems to start up the UCD design

process as their experience towards understanding the problem stated in the experiment was lower than the subjects from the pilot study. Thus, it was decided to allow the students to view a brief diagram of partially completed use case diagram, which would be displayed to them for two minutes as illustrated in Figure 4.9 before they began the tasks stated in the experiment. This step was included in the next experiment where the collection of the relevant data was expected to be the most.

The average time used by these participants in completing the tasks was one hour. This was used as a guide for setting the timeframe for the next experiment. Further the seeded assumptions were matched with the respective scenarios in the UCD as described in Section 4.2.3.3. Upon completing this update for the experiment's design, the Assumptions Seeding Experiment session was conducted with a larger group of participants.

5.4 Observation on Assumptions Seeding Experiment

The Assumptions Seeding Experiment was executed as explained in Section 4.2.3.2. The goal of the experiment was to detect the number of original assumptions remaining in the seeded requirements document. The results from the Assumptions Seeding Experiment were analysed and used to verify the experiment's hypothesis; that the size of assumptions in a Requirements Specification Document (RSD) can be derived using assumptions seeding technique.

5.4.1 Process for Data Analysis

The overview of the data analyses process for the participants is described in Figure 5.10.

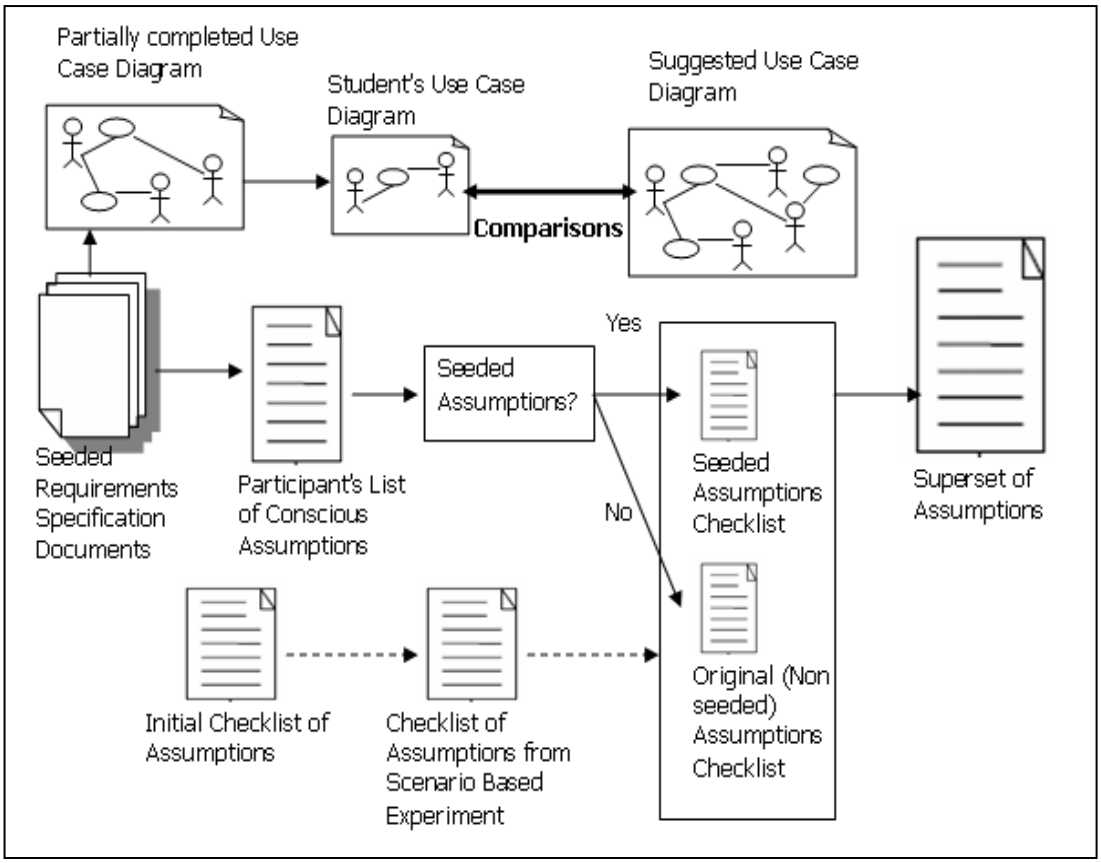


FIGURE 5.10 – *Overview of the Data Analyses for ASExp*

There were two tasks assigned to the participants as elaborated in Appendix B-2:

- to elicit assumptions from a seeded Requirements Document Specification (RSD) for Mine Drainage Control System
- to develop a use case diagram (UCD) for the system.

The UCD generated by the participants were compared with the 'Suggested UCD', depicted in Appendix A-2 in order to determine the accuracy. The proportion of seeded assumptions discovered was used to determine the number of the original assumptions remaining in requirements document.

There were two conclusions to be drawn from the analyses of this experiment:

- determine whether the assumptions seeding technique can be a reliable approach to discover assumptions in requirements documents.
- comparing the size of the discovered assumptions with regards to the size of the requirements document.

The detailed process for the data analysis is depicted in Figure 5.11.

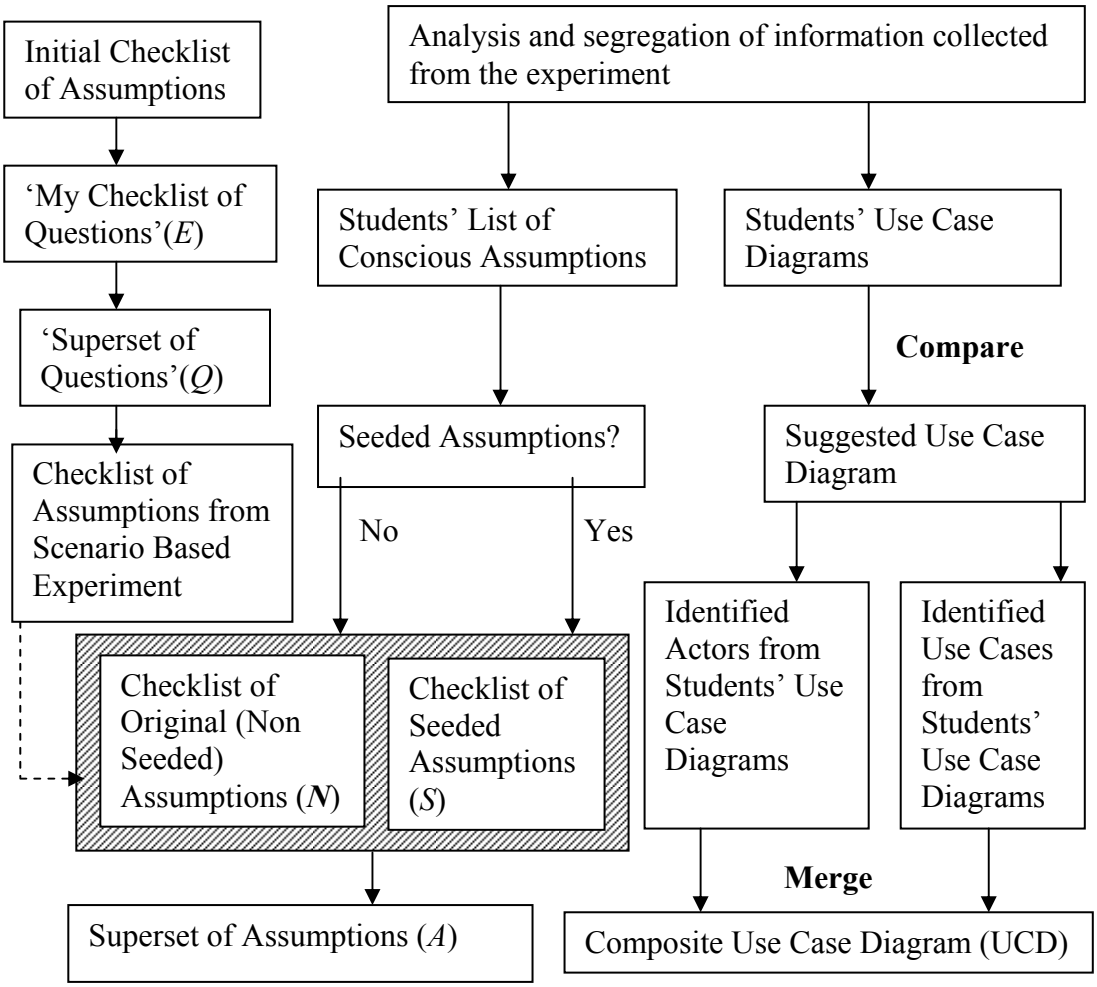


FIGURE 5.11 – *A Detailed Process for Analysing data for ASExp*

The analysis of the data was separated into two parts:

- the list of assumptions
- the Use Case Diagrams (UCD)

The ‘Checklist of Seeded Assumptions’ (*S*), described in Appendix B-3 and ‘Checklist of Original (Non Seeded) Assumptions’ (***N***), described in Appendix B-4. Those lists were designed for the purpose of our experiment to act as the reference documents to compare with the assumptions collected from the students. The previous checklists were developed using the ‘Checklist of Assumptions from Scenario Based Experiment’ which was derived from ‘My Checklist of Questions’ and ‘Superset of Questions’.

The process used in Scenario Based Experiment (listing the questions individually) described in Section 5.2.1 was used in this experiment for listing the participants’ assumptions individually. The assumptions were compared with *S* and ***N*** in order to identify the number of seeded assumptions and original assumptions gathered from the students. The seeded assumptions found by the student were matched with *S* and the non-seeded assumptions were matched with ***N***. If the students’ non seeded assumptions were not found in ***N***, then, they will be added into ***N***. The previous process was used to develop the ‘Superset of assumptions’ (*S*) illustrated in Appendix B-9 which will consists the final number of the assumptions detected for the mine drainage system.

The collected UCD from the participants were compared with the ‘Suggested UCD’ which was created prior to conducting the experiment as illustrated in Section 4.2.2. The ‘Suggested UCD’ serves as a superset of the UCD for the purpose of our experiment. A preview of partially completed UCD depicted in Appendix B-6 were screened to the participants for two minutes in order for them to have a brief idea of the system which needed to be developed before they began their UCD designs.

The actors and use cases identified by the participants were listed in files named 'List of Actors for Assumptions Seeding Experiment' and 'List of Use Cases for Assumptions Seeding Experiment' respective. The process used in Scenario Based Experiment in Section 5.2.1 was adopted to derive the 'Composite Use Case Diagram'. The previous UCD corresponded to the participants' understanding of developing a UCD for the mine drainage system while completing the task of eliciting assumptions from the requirements document. The 'Composite Use Case Diagram' for Assumptions Seeding Experiment was compared with the 'Suggested UCD' in order to assess the degree of accuracy of the UCD designed by the participants.

5.4.2 Analysis of the UCDs Developed by the Participants

There were three groups of participants: these groups were labelled as *P*, *Q* and *R*. The time given to complete their experiment was an hour. The data shown in Table 5.19 presents the number of actors and use cases detected by the groups. The 'Suggested UCD' (as shown in Appendix A-2) is used as the "reference" document to judge the produced data.

TABLE 5.19 – Accumulation of Actors and Use Cases for ASExp

Descriptions	Student Group			Composite Use Case Diagram
	P	Q	R	
Actors	6	5	4	6
Use Cases	2	2	4	4
Total actors and use cases	8	7	8	10
Accuracy of the developed UCD	47%	41%	47%	59%

The accuracy of the UCD developed by the participants was calculated using the same method as that used in Section 5.2.3. The total number of actors and use cases found in 'Suggested UCD' was 17. Hence, the accuracy of the UCD developed by

group *P* is given by (8/17) * 100 which results in 47.06%. Group *R* developed an UCD which was equal in accuracy to with group *P* whereas group *Q* developed an UCD with a lower accuracy compared to the other two groups.

The 'Composite Use Case Diagram' (or Composite UCD) illustrated in Appendix B-7 was developed by merging and removing the duplicate actors and use cases found by the three groups during ASExp. The accuracy of the 'Composite UCD' is 59%. Note that this value is lower than the accuracy of the 'Composite UCD' developed during the SBExp, 65% as explained in Section 5.2.5.3. Thus, the quality of the 'Composite UCD' developed during SBExp is better than ASExp. However, the quality of the UCD does not totally depend on the accuracy. It must be admitted that there are other factors which can influence the quality of the 'Composite UCD'.

The type of activity undertaken by the students during the experiment sessions is another factor that determines the quality of the UCD. For instance the seeded requirements documents distributed to the ASExp's participants contained more information about the mine drainage system. In contrast, the SBExp's participants were furnished with requirements document which deliberately lacked information about the system. However, the SBExp's participants managed to develop a 'Composite UCD' of a higher accuracy compared to the UCD designed by the ASExp's participants. This observation was the result of the activity undertaken by the students to design the UCD. In SBExp, the students were given the opportunity to clarify their doubts pertaining to the system during the Question and Answer session (refer to Section 4.2.2) in order to build their UCD. The ASExp's participants were assigned to design the UCD by detecting the seeded assumptions from requirements documents. Therefore some methods, such as questioning, have a probability to clarify more assumptions in order to

design a UCD with a better quality. This hypothesis is also supported in Section 4.2.2. The approach used to design a system has influence on the quality of the system.

5.4.3 The Influences of the Discovered Assumptions

A partial list of examples of assumptions discovered by the participants was extracted from Appendix B-8 and is shown in Table 5.20. The evaluation of this list shows that the assumptions elicited by the groups influence the design and quality of the UCD. For instance, the assumption identified by group *R* '*The alarm will be activated if there is an accident'* in Table 5.20 refers to the participants' understanding for designing an actor to activate the alarm in the cases of accidents. The previous assumption also allows participants to consider the causes which may trigger the accidents in the mine. This will assist the participants to determine the actors and use cases related to the accidents. Conversely, the assumption found by group *Q*, for instance, '*We assume that console is always working and only one console is present*' though relevant to the system, it neither contributed towards designing the actors nor use cases. Therefore the assumptions discovered by the participants can be a factor assisting in the design of their UCD.

TABLE 5.20 – Examples of assumptions discovered during ASExp

Descriptions of Assumptions	Groups
As per requirement 2.2.7, we assume that operator is alerted of low air flow reading on console only.	Q
We assume that console is always working and only one console is present.	Q
From 2.1.1, assume that pump will be turned off manually or automatically in case of critical conditions.	R
The alarm will be activated if there is an accident.	R
The operator can access the database for monitoring.	R
Exact values for low and high are not mentioned, we assume.	P
We assume that low air flow means either high methane or high carbon monoxide flow (non-functional 2.2.7).	P

The relationship below describes the dependency of the quality of the UCD developed by the participants. Put simply:

> *Accuracy of the UCD depends on the discovered assumptions*
>
> *Quality of the UCD depends on the accuracy of the UCD*
>
> *Hence, quality of the UCD also depends on the discovered assumptions*

Furthermore, the above observation supports the conclusion in Section 5.2.5 which states that the quality of the UCD correlates to the quality rather than the quantity of the detected assumptions.

5.4.4 Results from Assumptions Seeding Technique

The list of assumptions gathered from the groups is depicted in Appendix B-8. The number of assumptions collected from the experiment is illustrated in Table 5.21. Group *R* managed to detect the most assumptions compared to the other groups.

TABLE 5.21 – Type of Assumptions identified during ASExp

Descriptions	Student Group		
	P	Q	R
Seeded Assumptions	1	1	1
Non-Seeded Assumptions	6	8	10
Total detected assumptions	7	9	11

The data depicted in the above table shows that none of the groups managed to detect more than one of the ten seeded assumptions. The number of detected seeded assumptions by each of the group is lower than the non-seeded or original assumptions.

The previous observation can highlight the need for considering the method and criterion used to design the seeded assumptions.

The above data is used to measure the number of original assumptions found in the Requirements Specification Documents (RSD). This is measured by using the proportion of seeded assumptions discovered. For instance, the data collected for group *P* is used below to find the original assumptions remaining in the requirements documents.

Group P

Seeded Assumptions	*Non-Seeded Assumptions*
DSA = 1	DNA = 6
9	Unknown

Total: TSA = 10 TNA = ?

Hence, TNA = (10) . (6) = 60
Hence, the number of potential assumptions = TSA + TNA = 10 + 60 = 70
Hence, there are 60 - 6 = 54 assumptions remaining.

The number of detected seeded assumption (DSA) and original assumptions (DNA) found were one and six respectively. The total number of assumptions seeded (TSA) into the requirements document was ten. Hence, participants have missed nine seeded assumptions. The previous values, one, six and ten are replaced in Equation (2) from Section 4.2.3.2 to calculate the size of original assumptions (TNA) remaining in the requirements document which is 60. This value is used as an indicator to estimate the number of other potential assumptions detected by group *P* which results in 70. The above steps were used to derive the assumptions estimated to be discovered by the groups, *Q* and *R* which was 90 and 110 correspondingly. Table 5.22 depicts the

estimated values for assumptions which can be detected from the requirements documents by each of the group.

TABLE 5.22 – Estimated number of potential assumptions

Descriptions	Student Group		
	P	Q	R
Total detected assumptions (DNA + DSA)	7	9	11
Potential Assumptions	70	90	110
Percentage of assumptions discovered	10%	10%	10%

The actual number of assumptions discovered by individual group from the controlled experiment was gathered and the values for the estimated number of assumptions made by each group as below were derived.

Estimated number of assumptions made by each group is {70, 90, 110}

The actual number of assumptions made by each group is {7, 9, 11}

Each of the three groups actually managed to obtain only 10% of the total estimated number of assumptions made. There is a huge variance between actual number of assumptions detected by the groups and estimated number of assumptions discovered by the groups using assumption seeding approach. Here the validity of the assumptions seeding technique could be questioned with regards to the number of assumptions made. This suggests that more work is required to enhance the assumptions seeding method as a tool to discover assumptions from requirements documents.

Although the ASExp resulted in a highly unexpected number of assumptions based on assumptions seeding method, an alternative approach to show that the technique might not be appropriate for assumption detection is not explored in this project due to time constraint. However, a research looking into an alternative interpretation of assumption seeding technique is worth for further exploration.

5.4.5 The Estimated Size for the Superset of Assumptions

In this section, the number of assumptions is estimated that can be gathered from the requirements document of the mine drainage system. There are two approaches to estimate this value described below.

5.4.5.1 Estimation Method 1

In this section, the number of assumptions estimated to be discovered is stated using the Assumptions Seeding Technique. The observation gathered from ASExp shows that group *R* (as illustrated in Table 5.21) has discovered the maximum number of assumptions compared to the other groups. The number of assumptions detected by each group dictates the estimated values of the potential assumptions discovered. Hence the estimated value for maximum number of assumptions discovered from the requirements document among the three groups is 110.

5.4.5.2 Estimation Method 2

The number of assumptions derived from the requirements document can be estimated through multiple independent group analysis. The analysis of the requirements document to detect the seeded assumptions during the experiment was conducted independently. The students were asked to work within their own group and not to discuss with the other groups. The data collected from the experiment is presented in the form of "Set" to display the analysis of the independent group as depicted in Figure 5.12.

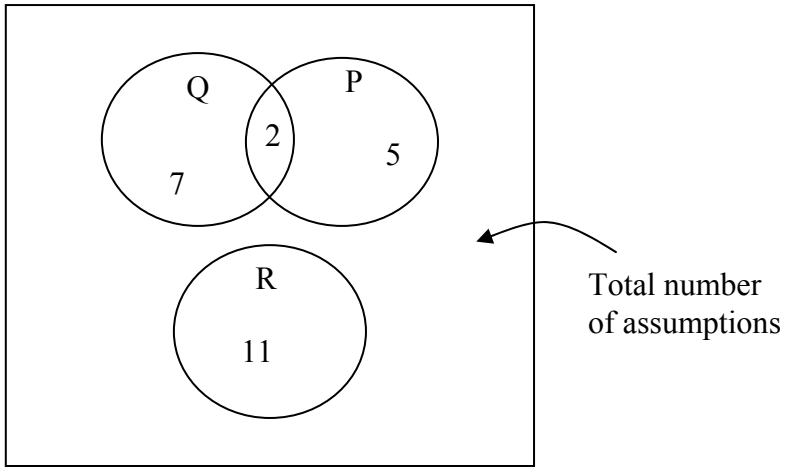


FIGURE 5.12 – *Multiple Independent Group Analysis for ASExp*

The "Set Theory" was used to estimate the common assumptions among the groups. For instance, the value for intersection is denoted by ∩ between groups, *P* and *Q* are two whereas groups, *P*, *Q* and *R* are zero. Hence, the common assumptions for the groups are derived by using the method as below.

Size of Assumptions accumulated from groups, P and Q

$$\approx \frac{\#P \,.\, \#Q}{\#(P \cap Q)} \approx \frac{7 \,.\, 9}{2} \approx 32$$

Since there are no intersections between the groups, the size of assumptions accumulated from Groups, P, Q and R is ∞.

The above observation shows that the estimated number of assumptions from groups, *P* and *Q* is 32 and groups, *P*, *Q* and *R* is possibly infinite. These values suggest a bound for the estimated number of assumptions gathered by the groups, which is as below.

$x \geq 32$ *whereby 'x' is the number of assumptions*

Therefore, the estimated number of assumptions gathered from the students should be at least 32.

5.4.6 The Size for the Superset of Assumptions

The Superset of Assumptions consists of the assumptions gathered by the students. It was developed by carefully merging the list of assumptions as described in Figure 5.13.

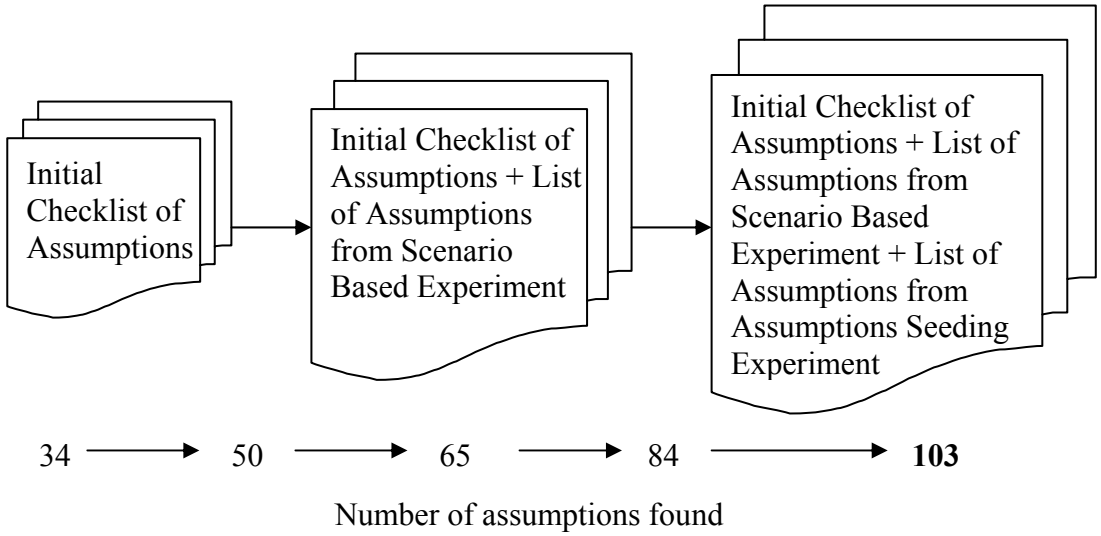


FIGURE 5.13 – *The process of deriving the Superset of Assumptions*

The 'Initial Checklist of Assumptions' was developed prior to conducting the experiments and it consists of 34 assumptions. This was expanded to 50 when assumptions were gathered during the execution of Scenario Based Experiment. The size of assumptions continued to increase when attempts to merge and remove duplicates of assumptions were found in the 'Initial Checklist of Assumptions' and 'List of Assumptions from Scenario Based Experiment'. The final size of assumptions accumulated in the 'Superset of Assumptions' is 103. It is noted that the number of assumptions derived from the experiments is close to the size of estimated potential assumptions produced in 'Estimation 1', which is 110, and that it also falls within the recommended range of 'Estimation 2'.

5.4.7 Total Number of Assumptions Gathered from the Experiments

The outcome for the size of assumptions for mine drainage system from the experiments was unexpected. At the experimental design stage, only 34 assumptions were listed in 'Initial Checklist of Assumptions'. It was presumed that a requirements document with a length of two pages could be based upon the assumptions that were listed in the initial checklist. However this assumption was incorrect. In fact the final number of assumptions, 103, was not expected.

The final size of assumptions was evaluated with regards to the size of the requirements document. It was discovered that two pages of requirements document could lead to the possibility of producing more than 100 potential assumptions. Therefore many requirements documents produced during Requirements Engineering Processes were vulnerable to assumptions. This finding was presented in a paper [91] presented at the symposium.

It is essential to discover assumptions from requirements document because these assumptions can be a factor that leads to potential defects in requirements, if the document is not validated. Consequently efforts should be taken to identify and validate assumptions during the early stages of software development, where savings from the later cost for reworks can be presented.

5.4.8 Factors Contributing to the Outcomes of the Experiments

There exists a weak correlation between the variables observed during the experiments, the number of potential assumptions, and the quality of the UCD

developed by the participants, as described in Section 5.2.5. There are several factors contributing to these results which described below:

- Assumptions about the subjects' skills for developing Use Case Diagrams

  The subjects were assumed to be competent in developing UCD. However the participants' skills for UCD development were not verified and hence there were possibilities that this assumption could have been invalid.

- Assumptions about the subjects' knowledge on mine drainage system

  There was potential for some of the participants to have prior knowledge regarding the mine drainage system. Though the subjects were graduates, who were assumed to have no industrial background, it does not stop them from acquiring information related to mine drainage system through research, readings and assignments from other course units prior to these experiments. Hence some, who were familiar with the mine systems, might have an advantage in the process of analysing the system requirements for developing UCD compared to the others.

- Level of the individual capability

  Even though a special procedure to group the subjects was not used during the experiment, there were possibilities that one of the teams would perform better, compared to the rest, due to the skill of members of this particular group.

In addition to the above issues, there were constraints influencing the outcomes of this investigation. They are elaborated below:

- The participants' experience in identifying assumptions

  Although the subjects were furnished with an approximate four of hours for lectures on Requirements Engineering issues, there was the probability that this session was not sufficient to equip the subjects with the assumptions detection skills through requirements analysis. Hence participants' experiences in identifying assumptions could have been a constraint for our investigation.

- Limited time for identifying assumptions

  The maximum time given for the execution of the experiments was forty five minutes. Therefore, time may have been one of the limiting factors for identification of assumptions in this research.

- Limited number of iterations given to the participants for detecting assumptions

  Participants were given a maximum of 2 trials for analysing the requirements documents, this might also be the limiting factor for the participants to detect assumptions and redesign their UCD.

- Conditions of the seeded assumptions

  Since the error seeding technique had been adopted for assumptions seeding, there was a need to consider whether the seeded assumptions had the same criteria as the original assumptions in the requirements documents. Hence this factor has to be taken into account to maximize the effectiveness of the assumptions detection procedures.

5.5 Conclusion of the Topic

In this chapter, the analysis of the results collected from the observation of the students' projects and experiments has been elaborated. The factors which could contribute to assumptions propagation in projects include both the knowledge of the development team and the time duration assigned for the project. The outcomes of the experiments were unexpected as a two pages requirements document managed to generate 103 assumptions. However, the size of assumptions is not limited and in fact depends on the scope of the requirements analysis process. The issues related to assumptions in development are commonly encountered in most of the software projects. The results gained from these experiments will assist software developers by emphasizing the need for early observation to tackle assumptions during Requirements Engineering phase.

Chapter 6 Conclusions

Chapter Six will summarize this thesis by highlighting the key points discovered during this research, "Exploration of Assumptions in Requirements Engineering (EAiRE)". Some suggestions have been included for possible areas of studies to further explore assumptions made during Requirements Engineering (RE). Lastly some concluding remarks and lesson learned upon completion of this study are presented.

6.1 Research findings

The literature reviews illustrate that the current approaches in assumptions detection, during software projects, are mainly tools oriented. However this thesis claims that it is importance to understand and explore the non-automated approaches to discover assumptions. Thus this research comprises the investigation of assumptions detection techniques, based on human-centric activities, made during the initial stage of software development.

If the research hypothesis described in Section 1.3 is revisited, it is defined as:
The standard defect detection techniques can be adopted in order to reveal the extent of assumptions made during Requirements Engineering.

This thesis follows the above hypothesis by using an assumptions-seeding technique (which was modified from defect seeding approach) to discover the original assumptions remaining in a requirements documents. The defect seeding method is used to assist the error debugging activities during software verification processes.

This research is conducted to present a number of measurements and insights into the degree by which the incidence of assumptions in RE influences the software development process. The goals of this investigation are as elaborated in Section 1.4. This was justified by the number of assumptions gathered from both the experiments, Scenario Based (SBExp) and Assumptions Seeding (ASExp) conducted during the investigation. The participants were assigned to design Use Case Diagrams for a Mine Drainage Control System which was described in a two pages Requirements Specification Document (RSD). The outcome of the experiments was unexpected since it was not predicted that 103 assumptions would be discovered (from the Superset of Assumptions) with regards to the size of the RSD. This confirms that assumptions made during RE need serious attentions. In the software development industries, the stakeholders encounter large numbers of requirements documents. These documents are vulnerable to assumptions. Even though the number of assumptions will not serve as the ultimate factor that determines the quality of software, it does contribute to a great extent. In this research the UCD produced by the participants was analogous to software. Thus investigations regarding issues related to assumptions should be strongly encouraged in both academic and industrial fields.

As discussed in Section 1.1, false assumptions are a factor that may lead to requirements defects. The number of assumptions gathered during RE will be an alert to the stakeholders. This could be used as a potential estimation to derive requirements defects originated from invalid assumptions. Further, false assumptions also contribute to rework cost in development. The cost of rework for requirements defects during RE is significantly lower compared to other stages of development. Therefore discovering and rectifying assumptions in the early stages of projects will enable us to treat them early and thus reduce the rework cost.

Even though it is realized that the identification of the nature of assumptions in RE is a difficult task, it is still crucial. Therefore, a 'Taxonomy for Assumptions made during RE' was constructed to assist the investigation. Assumptions in RE appear not to be only influenced by the human factor, which were made either consciously or unconsciously, but also by other factors such as (1) the area where the assumption was made pertaining to system (Functional, Quality, Process, Regulations and Standards), (2) viewpoint for making assumptions (management, business and technical) and (3) causes for making assumptions (knowledge about the system, allocation of cost and time for the software project, cultural differences and types of interaction among stakeholders such as face to face, teleconferencing and other types of approaches).

The development of this taxonomy will allow enhancement of understanding regarding the occurrences of assumptions during the early development phase. This can assist the stakeholders to plan and utilize suitable techniques to identify and validate assumptions in order to reduce the risk of defects originating from them. For instance if a software development project involves geographically distributed stakeholders, then there needs to be awareness of the possibilities for assumptions to be made due to the cultural differences such as beliefs, languages, religion and values of life. In this case, it is necessary to consider appropriate methods to capture assumptions while eliciting the requirements for the system. Further the type of interactions used in this case may also contribute to the gap between each other. Therefore, the development team need to consider these factors in order to reduce the assumptions being made during the requirements analysis process. The ethnography technique will be suitable in this scenario.

The classification of assumptions will also enable the evaluation of the root causes of the misconceptions in RE. For instance if the highest number of assumptions gathered related to functional areas of the system, then the functional requirements need to be revisited, clarified and refined.

6.2 Lesson Learned from the Outcomes of the Experiments

6.2.1 The Scenario Based Experiment

The SBExp was executed by providing the subjects with a requirements document which was deliberately presented with a lack of information. The decisions made by the subjects were assumed to be correct in order to complete the assignment's task which was to design the Use Case Diagram (UCD). The hypothesis for SBExp is illustrated below as stated in Section 4.2.2.

(i) The larger the number of validated assumptions, then contributes to a higher degree of accuracy in the requirements documents; if a larger number of questions were asked by the participants, then a higher degree of accuracy will be expected in designing the Use Case Diagrams.

(ii) Requesting participants to question assumptions aloud reduces the number of assumptions made.

The conclusion drawn from the outcomes of the SBExp is as below.

"We could not make a clear statement for the correlation of the variables involved in the experiment; the quality of the UCD produced and the number of potential assumptions related to the development of UCD verified by the

subjects. However by questioning the doubts aloud during requirements analysis, the subjects avoided making some assumptions."

In the conclusion stated as above, '*quality of the UCD*' is analogous to 'accuracy of the UCD' and '*potential assumptions*' is analogous to 'missed questions'. There exist a weak correlation between the variables observed during the experiment, the number of potential assumptions and the quality of the UCD developed by the participants. Further, a clear statement could not be made saying that the number of assumptions impacted the quality of the UCD developed by the subjects, since there was not sufficient evidence gathered from the experiment, as described in Section 5.2.5.

It must be agreed that assumptions are not the sole factor for the quality of artefacts delivered through the requirements analysis process. In these experiments, the artefact was the UCD. There were several factors besides assumptions which contribute to the quality of the UCD, as illustrated in Section 5.4.8.

6.2.2 The Assumptions Seeding Experiment

The hypothesis for ASExp, as described in Section 4.2.3, is that the size of assumptions in a Requirements Specification Document (RSD) can be derived using assumptions seeding technique.

The results gathered from ASExp demonstrated that

"The original assumptions in a Requirements Specification Document can be detected through the assumptions seeding method. However, the distribution and the prediction of the original assumptions were not clear in this case"

The hypothesis of ASExp was demonstrated by the experiments results. However the reliability of this approach must be further validated in industry since we have yet to validate them. The assumptions seeding methodology used in our research comes with an underlying assumption that is taken from defect seeding. In defect seeding, it is assumed that the distribution of seeded defects is the same with the original or non-seeded defects. This analysis was done by examining the types of defects found for seeded defects and original defects thus producing a catalogue of defect types. This catalogue will be used to study the distribution of the defects. Hence, this entire work, as mentioned above, was used to justify the reliability of defect seeding method to find the number of defects which still remain in a program or code.

Similar to defect seeding method, the evaluation criterion for assumptions seeding approach is the assumptions detection effectiveness, the ratio between the numbers of assumptions discovered by the participants and the numbers of assumptions originally in the document. In this context the distribution of the seeded assumptions and original assumptions must be equal. The outcome of these research activities did not show that this criterion was achieved because the distribution of the assumptions was not evaluated.

Even though we have developed a taxonomy for assumptions, a single experiment is insufficient to examine the distribution of the assumptions in the experiment. Further, there is a huge variance between actual number of assumptions

detected by the groups and estimated number of assumptions that could be discovered by the groups using assumption seeding approach during the experiment. Therefore the conclusion drawn from the ASExp's observation is that the assumptions seeding approach, to predict the size of original assumptions remaining in a requirements document, must ensure that the distribution of seeded and original assumptions has been taken into consideration.

In addition, there are several factors influencing the results of the ASExp, as illustrated in Section 5.4.8. These issues can be treated as the constraints for the experiment and further improvement can be made in this area.

6.3 Possible Future Research Directions

This thesis has indicated a gateway for further research about occurrences of assumptions during RE. It is strongly recommended that there should be similar initiatives on exploration of assumptions in the software industries:

- Efforts should be taken to further survey the possibilities of designing and implementing methods to identify assumptions early in development process.
- There must be attempts to designed techniques to measure the rate of requirements defects propagated by invalid assumptions.
- Exploration of the relationship between size of invalid assumptions and rework cost caused by them.
- Relationship between the taxonomy of assumptions with the assumptions discovered and how they impact the overall performance of the Requirements Engineering Process.

6.4 The Research Conclusions

The insights gained from this thesis support the current understanding that the issue of assumptions needs attentions in software development. This survey claims that assumptions should be attacked from the initial phase of development. In this investigation, Exploration of Assumptions in Requirements Engineering, an approach has been introduced to detect assumptions during requirement analysis process and to classify them into a systematic taxonomy.

Even though both the experiments conducted in this research did not produce unexpected conclusions, it is strongly claimed that assumptions in RE are critical and the requirements documents are vulnerable to assumptions. Time was a constraint that limited further exploration of both the experiments to achieve improved results to support the claims of this research. However, it is worth for further work to be carried out in this field.

However there is a need to further manage assumptions. This effort will include four steps: (1) frequent identification, (2) validation, (3) documentation of assumptions and (4) tracing them iteratively as described in Chapter 1. This will improve the Requirements Engineering Process through reducing the rework cost for defects originated by incorrect assumptions.

Appendices

Appendix A Scenario Based Experiment (SBExp)

A-1 Scenario Based Experiment

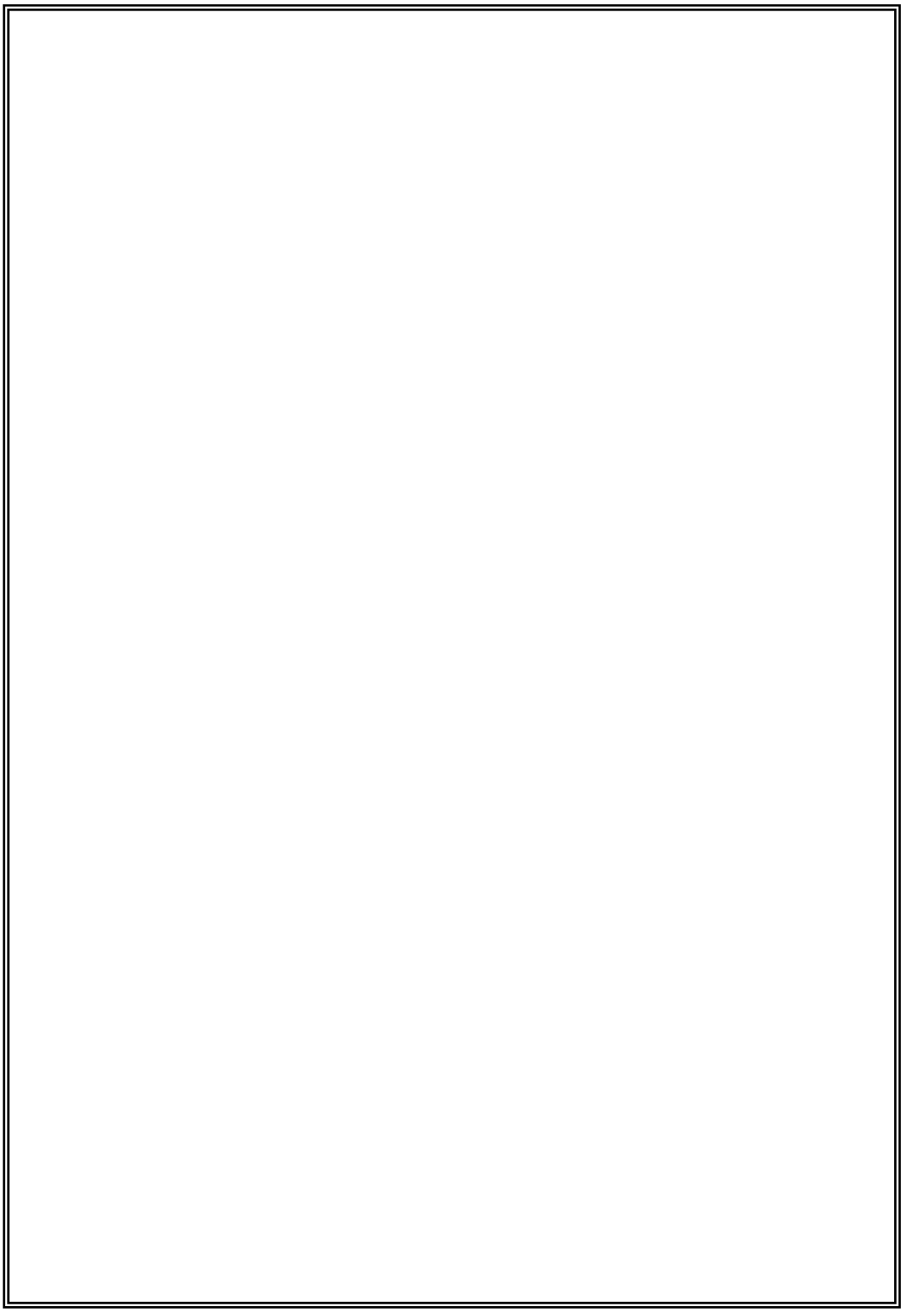
TUTORIAL: REQUIREMENTS ELICITATION ACTIVITY

This tutorial is designed to gather data about underlying assumptions being made during the requirements elicitation process. You should work in groups of a maximum of **three** students. Please choose a distinctive and easy to remember group name. The work includes drawing a use case diagram for the proposed system. If you believe there are issues requiring further clarification, please write a set of questions that you would need answered and complete the use case diagram as best you can. Thank you for your participation. The outcomes of this tutorial will be returned to you and discussed in your next lecture or tutorial session.

Select a group name: [ ]

Scenario

You have been appointed by Jim Atkins, your manager as a System Analyst for a new project to develop a mine drainage control system. Jim has forwarded to you the mail which he has received from the client describing the system. You are required to accomplish two tasks as stated in Jim's mail.



Group Name:

FORM A

No	Descriptions of your questions

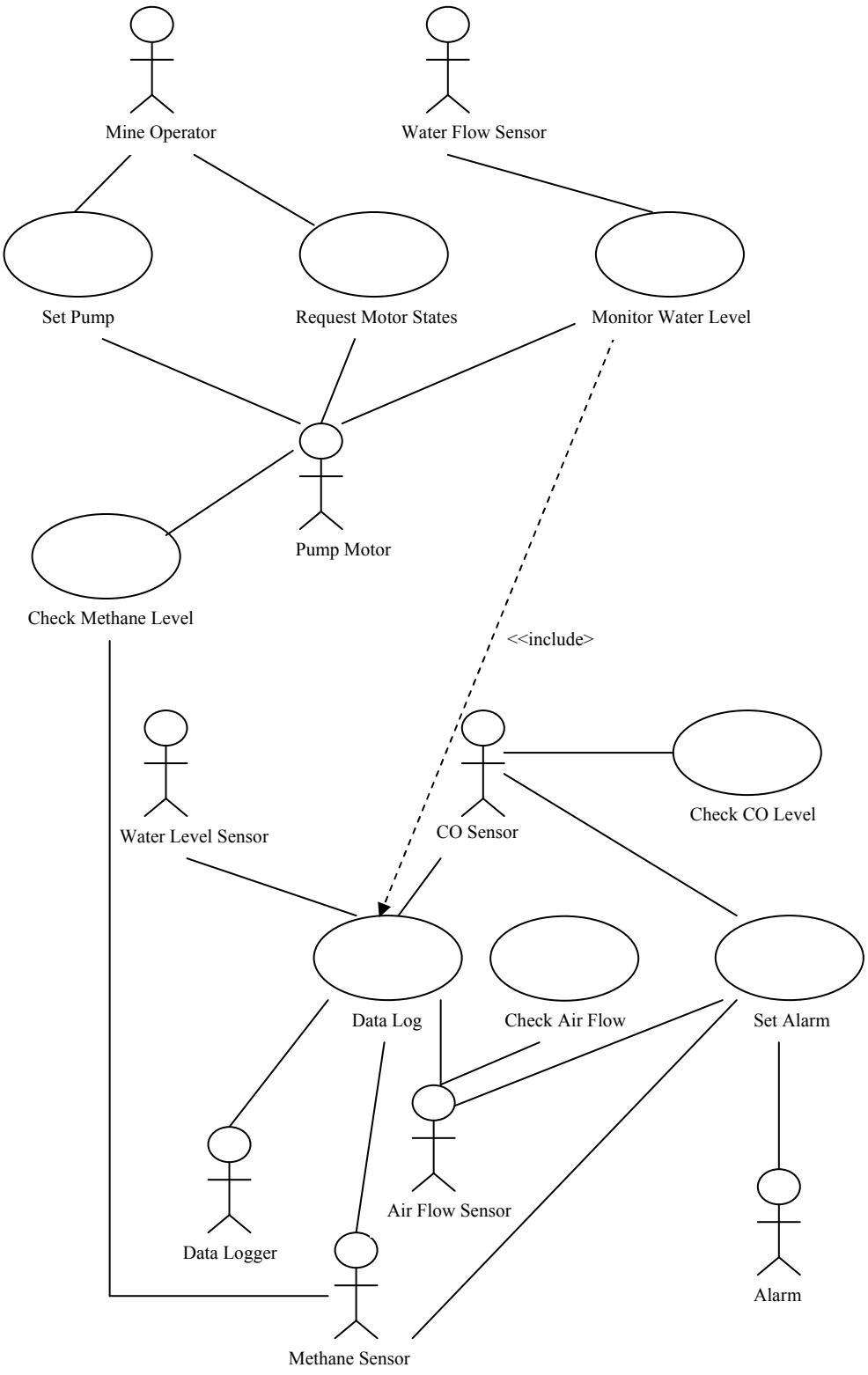
Use Case Diagram:

Group Name:

Your Updated Use Case Diagram:

A-2 Suggested Use Case Diagram

Suggested Use Case Diagram for Mine Drainage Control System



A-3 My Checklist of Questions

Check List of the questions (ALL THE QUESTIONS) which developers intend to follow up with the clients for Mine Drainage Control System (MDCS).

*R = Relevant questions for developing the UCD
*I = Irrelevant questions for developing the UCD

R	19
I	15
Total	35

No	Questions	*R/I
1	What are the hardware requirements which you are expecting for MDCS system? ***Mine Drainage Control System (MDCS) should be implemented on a single processor with a simple memory mapped input/output architecture.***	I
2	What is the scope of this system? ***The scope of the Requirements Specification Document for MDCS is only on the timing requirements.***	I
3	What are the safety requirements of this system? ***The main safety requirement is that the pump should not be operated when the level of methane gas in the mine reaches a high value.***	R
4	What are the quality (or non-functional) requirements of this system? ***The quality requirements of this system can be divided to timing, dependability and security.***	I
5	What are the security issues of this system? ***The security of the system will not be discussed at this stage.***	I
6	What are the reliability issues of this system? ***The reliability of the system will not be discussed at this stage.***	I
7	How the water level in the sump is monitored? (For instance, do we need to use any high water level sensor and low water level sensor?) ***The water level in the sump is monitored by the pump controller or also can be obtained by the operator.***	R
8	How the pump will be able to monitor the water level in the sump? ***The pump monitors the water level in the sump by using water level sensors which are high water level sensor and low water level sensor.***	R
9	How the information about the water level in the sump can be obtained by the operator. ***The operator can obtain the information about the water level in sump from the pump controller by requesting the pump motor status.***	R
10	Where the information about the water level is stored? ***The information about the water level is stored in archival database.***	R
11	Are there any other functions for an operator besides operating the pump (turning on the pump when the water reaches high level and the water is drained until it reached low level)? ***The system is controlled via an operator console and the operator***	R

	is informed of all the critical events. Besides, the operator needs to require the status of the pump.	
12	What are the critical events of the system? ***The critical events are referred to high water level, high value of methane level, carbon monoxide or inadequate flow of air in the environment.***	R
13	What is "high" referring to in the statement "when the water reaches the high level" as described in your mail? ***Assume that the term 'high' is referring to a particular critical water level which will be informed later in the requirements documents.***	I
14	What indicates the pump need to be turned on when the water reaches the high level? ***When the water level reaches high level, the pump controller/pump motor will be alerted and the water will be drained out or alternatively, the operator will be informed and the water will be drained out.***	R
15	What will happen when the pump is not turned on when the water reaches the high level? ***In this case, if the pump did not turn on automatically, then, the operator should turn on the pump. If the pump still not turned on, then, system should stop operating and alarm should be activated.***	R
16	What will happen if the operator tries to drain the water from the sump during low water level? ***In this case, the draining process strictly should be prohibited and alarm should be activated.***	R
17	What will happen if the operator tries to drain the water from the sump during high methane level? ***In this case, the draining process strictly should be prohibited and alarm should be activated.***	R
18	What is "high" referring to in the statement "when the level of methane gas in the mine reaches a high value" as described in your mail? ***Assume that the term 'high value' is referring to a particular critical methane level which will be informed later in the requirement documents.***	I
19	What is the critical level for methane emitted from the mining processes? ***Assume that the 'critical level' is referring to a particular figure which will be informed later in the requirement documents.***	I
20	Where the information about the level of methane is stored? ***The information about the methane level is stored in archival database.***	R
21	Where the information about the level of carbon monoxide is stored? ***The information about the carbon monoxide level is stored in archival database.***	R
22	How the information about the level of carbon monoxide emitted from mining can be obtained by the system? ***The information about carbon monoxide level is obtained from the carbon monoxide sensor.***	R

23	How the information about the level of methane emitted from the mining can be obtained by the system? ***The information about methane level is obtained from the methane sensor.***	R
24	What is the reading period for methane sensor? ***It is assumed that the maximum reading period for environment sensors may be dictated by legislation and in this case will be 100 milliseconds. Hence, the maximum reading period for methane sensor (is an environment sensor) is 100 milliseconds.***	R
25	What is the reading period for carbon monoxide sensor? ***It is assumed that the maximum reading period for environment sensors may be dictated by legislation and in this case will be 100 milliseconds. Hence, the maximum reading period for carbon monoxide sensor (is an environment sensor) is 100 milliseconds.***	R
26	What is the reading period for water level sensor? ***The water level sensor is event driven. Hence, the reading period between interrupts from the two water level indicators (from the water sensor) must be at least 6 seconds.***	I
27	What is the reading period for water flow sensor? ***The water flow sensor given a reading period of 1 second and it uses the results of two consecutive reading (to determine the actual state of the pump).***	I
28	What is the reading period for air flow sensor? ***It is assumed that the maximum reading period for environment sensors may be dictated by legislation and in this case will be 100 milliseconds. Hence, the maximum reading period for air flow sensor (is an environment sensor) is 100 milliseconds.***	R
29	What should be done when there is a detection of critically high methane readings? ***Then, the operator must be informed within 1 second of within 2 seconds of a critically low air-flow reading and within 3 seconds of a failure in the operation of the pump.***	I
30	What should be done when there is a detection of critically high carbon monoxide readings? ***Then, the operator must be informed within 1 second of within 2 seconds of a critically low air-flow reading and within 3 seconds of a failure in the operation of the pump.***	I
31	What will happen if the pump is operated when the level of methane in the mine reaches a high value? ***Then, the mine is faced with the risk of explosion. This situation should not occur.***	I
32	How does the pump controller interact with the methane sensor? ***The methane sensor either polled or directly controlled by the pump controller.***	I
33	How does the pump controller interact with the carbon monoxide sensor? ***The carbon monoxide sensor either polled or directly controlled by the pump controller.***	I
34	How does the pump controller interact with the water level sensors? ***The water level sensors communicate via interrupts.***	I

35	What does the water level sensor consist of? ***Water level sensor consist of High Water Sensor and Low Water Sensor.***	R

A-4 Students' Initial List of Questions'

No	Questions	Question(s) Number*	Answers
1	What is a sump?	n/a	Assuming it is a reservoir for mining sludge.
2	What is the capacity of the sump?	n/a	Will be specified later in the requirements documents
3	What is the safe level of methane?	19	Assume that the safe level is referring to a particular figure which will be informed later in the requirement documents.
4	What equipments are used to measure the gas levels?	22, 23	The information about carbon monoxide level is obtained from the carbon monoxide sensor. The information about methane level is obtained from the methane sensor.
5	What equipments are used to measure the water levels?	8	The pump monitors the water level in the sump by using water level sensors which are high water level sensor and low water level sensor.
6	Is this a new system or improvement of an old system?	n/a	New system
7	What are the operations can be performed by the operator?	11	Operator monitors the water level. He can turn on the pump when the water level reaches a high level and turn off the pump when after the water has been drained out. The system is controlled via an operator console and the operator is specified of all the critical events. Besides, the operator needs to require the status of the pump.
8	Can operator monitor the levels?	11	The system is controlled via an operator console and the operator is specified of all the critical events.
9	Assuming gas level is high and water level is high, what to do?	n/a	The pump should stop operating and the methane sensor/ carbon monoxide sensor should alert the alarm to be activated.
10	What needs to be done when gas levels is high?	n/a	The pump should stop operating and methane

			sensor/ carbon monoxide sensor should alert the alarm to be activated.
11	What will trigger the gas to turn off when the gas level is high?	22, 23	The carbon monoxide sensor and methane sensor.
12	What is a high water level?	13	Assume that the term 'high' is referring to a particular critical water level which will be specified later in the requirements documents.
13	What is a low water level?	13	Assume that the term 'low' is referring to a particular critical water level which will be specified later in the requirements documents.
14	What is a high level of methane?	18	Assume that the term 'high value' is referring to a particular critical methane level which will be specified later in the requirement documents.
15	Why do we expect an explosion?	n/a	If there occur a situation where the pump still operates when level of methane is critical.
16	How fast the pump should operates?	n/a	Will be specified later in the requirement documents
17	What "the latter action" is as stated in the client's mail?	n/a	To turn on the pump when the water reaches a high level.
18	How simplified should the system be?	2	The scope of the requirements for the system is only on the timing requirements.
19	Are there more stakeholders?	n/a	Yes.
20	Is this system automated?	n/a	Yes.
21	What level of accuracy the system require?	n/a	Arbitrary.
22	What if both water level and gas level are high?	n/a	The pump should stop operating and the methane sensor/ carbon monoxide sensor should alert the alarm to be activated.
23	What if the system fails? What precautions should we take?	n/a	Evacuation.
24	What does "should not be operated" means?	n/a	It means 'should not perform any action, such as on or off.
25	Must there be a manual	n/a	Yes.

	override for safety?		
26	What do we do when too much methane and water level is high?	n/a	The pump should stop operating and the methane sensor/ carbon monoxide sensor should alert the alarm to be activated.
27	Where do we pump water to?	n/a	Surface to a specific area
28	What happen if the methane level gets too high during the period where the water is being drained out from the sump?	17	In this case, the draining process strictly should be prohibited and alarm should be activated.
29	What are the actual values of "high" and "low" levels?	13, 18	Assume that the term 'high' and 'low' is referring to a particular water level and methane level which will be specified later in the requirements documents.
30	What sort of interaction does the user have? For instance, Windows/Linux or dial, etc?	n/a	Windows
31	Who monitors the level of gases – the operator or another system?	22, 23	The carbon monoxide sensor and methane sensor.
32	How do you define the levels of gases and liquid?	13, 18	Assume that the water level and methane level which will be specified later in the requirements documents.
33	Will the operator notified about the risks when the levels go up?	11	Yes. The system is controlled via an operator console and the operator is informed of all the critical events.
34	What is the life expectancy of the system?	n/a	Will be specified later in the requirements documents.
35	What happen if the sump overflows and the water level does not come down?	15	In this case, if the pump did not turn on automatically, then, the operator should turn on the pump. If the pump still not turned on, then, system should stop operating and alarm should be activated.
36	Is the system manual or automated?	n/a	Automated
37	What is the user interface?	n/a	Windows
38	Is the system to be self-automated?	n/a	Automated
39	What is high or low water	13	Assume that the term 'high'

	level?		and ‘low’ is referring to a particular water level which will be specified later in the requirements documents.
40	What is high or low gases level?	18	Assume that the term ‘high’ and ‘low’ is referring to a particular methane level which will be specified later in the requirements documents.
41	Should the system be operated when the level of carbon monoxide is high?	n/a	No.
42	Should the system be showing or measuring the water level at all times?	20, 21	The information about the methane level and carbon monoxide level are stored in archival database.
43	Any output for the water level?	10	The information about the water level is stored in archival database.
44	Can the user select to view the water level?	10	The information about the water level is stored in archival database.
45	Can the user select to view the gases level?	20, 21	The information about the methane and carbon monoxide is stored in archival database.
46	Can the user shut down the system?	n/a	Yes.
47	Will the alarm be enabled if too high?	n/a	Yes.
48	Is the alarm turned on by the operator?	n/a	No, by the system.
49	Are there any separate alarm for gas and water?	n/a	No.
50	What level of water is considered is high or low?	13	Assume that the term ‘high’ and ‘low’ is referring to a particular water level which will be specified later in the requirements documents.
51	What level of methane is too high or acceptable?	18	Assume that the term ‘high’ and ‘acceptable’ is referring to a particular methane level which will be specified later in the requirements documents.
52	Is it automated or manual for the on or off of the system?	n/a	Automated as well as manual to turn on and off the pump.

53	Is the system remote or onsite?	n/a	Onsite
54	What hardware are they using?	1	Mine Drainage Control System (MDCS) should be implemented on a single processor with a simple memory mapped input/output architecture.
55	Who or what will interact with the system?	n/a	Operator, sensors and pump.
56	What monitors the carbon monoxide level?	22	The carbon monoxide sensor.
57	What monitors the methane level?	23	The methane sensor.
58	What are the major functionalities of the system?	n/a	The major functionality of the system are pump operation, environment monitoring (the gases), operator interaction (via console) and system monitoring(archiving data in database)
59	What is the condition if the water level is high and at the same time carbon monoxide and methane level is also high?	n/a	In this case, the draining process strictly should be prohibited and alarm should be activated.
60	What kinds of actions are given by the operator when the water level is high and at the same time carbon monoxide and methane level is also high?	n/a	The operator should not operate the pump and the methane sensor and carbon monoxide sensor should trigger the alarm.
61	Are the operator for water and gas referring to the same operator?	n/a	The same operator is in responsible for the entire system.
62	What about carbon monoxide high level?	n/a	Pump should not operate during the high level of carbon monoxide.
63	Who has the priority command (operator or the system)?	n/a	Operator.

A-5 Superset of Questions'

The table below shows the list of questions (ALL THE QUESTIONS) collected from the students after the second trial and respective answers for each question.

*T = Relevancy of the questions (refer as below)
A = Questions relevant to Actors
U = Questions relevant to Use cases
N = Questions irrelevant to Actors and Use cases but relevant to MDCS
I = Questions irrelevant to Actors, Use Cases and MDCS
B = Questions relevant to both Actors and Use cases
R = Total Relevant questions to UCD

A	13
U	8
N	33
I	0
B	11
Total	65

No	Questions	*T
1	What are the hardware requirements which you are expecting for MDCS system? *Mine Drainage Control System (MDCS) should be implemented on a single processor with a simple memory mapped input/output architecture.*	N
2	What is the scope of this system? ***The scope of the Requirements Specification Document for MDCS is only on the timing requirements.***	N
3	What are the safety requirements of this system? ***The main safety requirement is that the pump should not be operated when the level of methane gas in the mine reaches a high value.***	B
4	What are the quality (or non-functional) requirements of this system? ***The quality requirements of this system can be divided to timing, dependability and security.***	N
5	What are the security issues of this system? ***The security of the system will not be discussed at this stage.***	N
6	What are the reliability issues of this system? ***The reliability of the system will not be discussed at this stage.***	N
7	How the water level in the sump is monitored? (For instance, do we need to use any high water level sensor and low water level sensor?) ***The water level in the sump is monitored by the pump controller/pump motor or also can be obtained by the operator.***	A
8	How the pump will be able to monitor the water level in the sump? ***The pump monitors the water level in the sump by using water level sensors which are high water level sensor and low water level sensor.***	A
9	How the information about the water level in the sump can be obtained by the operator. ***The operator can obtain the information about the water level in sump from the pump controller/pump motor by requesting the pump motor status.***	B
10	Where the information about the water level is stored? ***The information about the water level is stored in archival***	U

	database.	
11	Are there any other functions for an operator besides operating the pump (turning on the pump when the water reaches high level and the water is drained until it reached low level)? ***The system is controlled via an operator console and the operator is informed of all the critical events. Besides, the operator needs to require the status of the pump.***	U
12	What are the critical events of the system? ***The critical events are referred to high water level, high value of methane level, carbon monoxide or inadequate flow of air in the environment.***	B
13	What is "high" referring to in the statement "when the water reaches the high level" as described in your mail? ***Assume that the term 'high' is referring to a particular critical water level which will be informed later in the requirements documents.***	N
14	What indicates the pump need to be turned on when the water reaches the high level? ***When the water level reaches high level, the pump controller/pump motor will be alerted and the water will be drained out or alternatively, the operator will be informed and the water will be drained out.***	A
15	What will happen when the pump is not turned on when the water reaches the high level? ***In this case, if the pump did not turn on automatically, then, the operator should turn on the pump. If the pump still not turned on, then, system should stop operating and alarm should be activated.***	B
16	What will happen if the operator tries to drain the water from the sump during low water level? ***In this case, the draining process strictly should be prohibited and alarm should be activated.***	U
17	What will happen if the operator tries to drain the water from the sump during high methane level? ***In this case, the draining process strictly should be prohibited and alarm should be activated.***	U
18	What is "high" referring to in the statement "when the level of methane gas in the mine reaches a high value" as described in your mail? ***Assume that the term 'high value' is referring to a particular critical methane level which will be informed later in the requirement documents.***	N
19	What is the critical level for methane emitted from the mining processes? ***Assume that the 'critical level' is referring to a particular figure which will be informed later in the requirement documents.***	N
20	Where the information about the level of methane is stored? ***The information about the methane level is stored in archival database.***	U
21	Where the information about the level of carbon monoxide is stored?	U

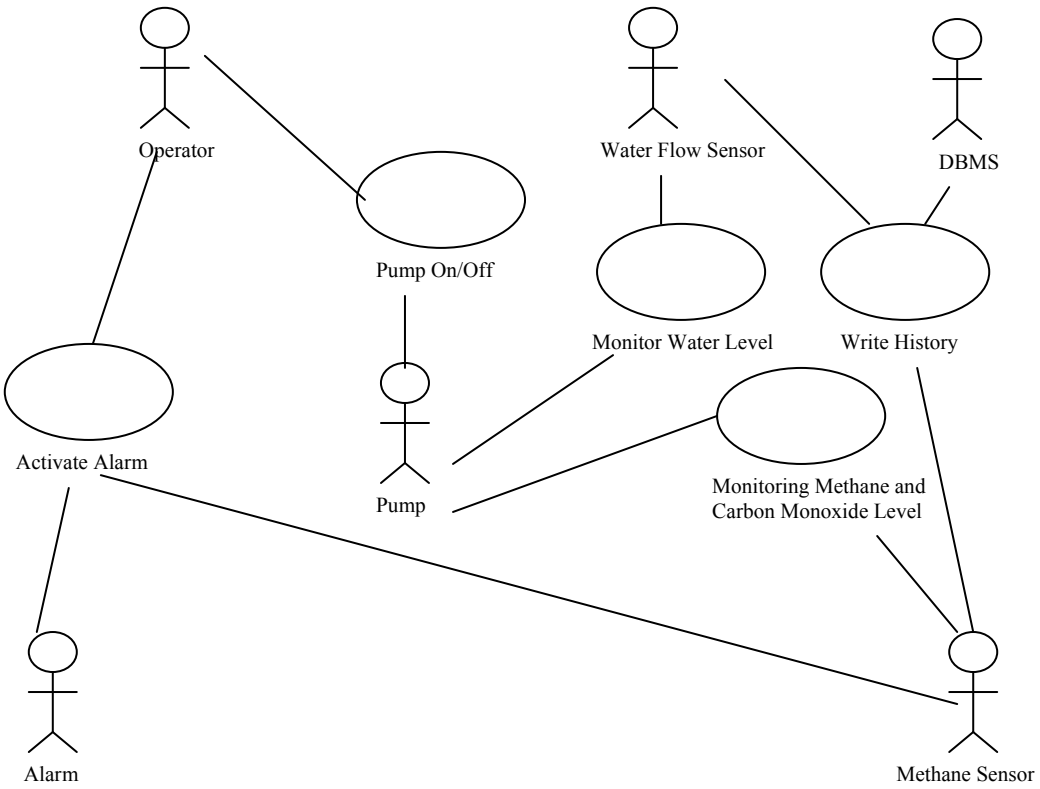
	The information about the carbon monoxide level is stored in archival database.	
22	How the information about the level of carbon monoxide emitted from mining can be obtained by the system? ***The information about carbon monoxide level is obtained from the carbon monoxide sensor.***	A
23	How the information about the level of methane emitted from the mining can be obtained by the system? ***The information about methane level is obtained from the methane sensor.***	A
24	What is the reading period for methane sensor? ***It is assumed that the maximum reading period for environment sensors may be dictated by legislation and in this case will be 100 milliseconds. Hence, the maximum reading period for methane sensor (is an environment sensor) is 100 milliseconds.***	A
25	What is the reading period for carbon monoxide sensor? ***It is assumed that the maximum reading period for environment sensors may be dictated by legislation and in this case will be 100 milliseconds. Hence, the maximum reading period for carbon monoxide sensor (is an environment sensor) is 100 milliseconds.***	A
26	What is the reading period for water level sensor? ***The water level sensor is event driven. Hence, the reading period between interrupts from the two water level indicators (from the water sensor) must be at least 6 seconds.***	N
27	What is the reading period for water flow sensor? ***The water flow sensor given a reading period of 1 second and it uses the results of two consecutive reading (to determine the actual state of the pump).***	N
28	What is the reading period for air flow sensor? ***It is assumed that the maximum reading period for environment sensors may be dictated by legislation and in this case will be 100 milliseconds. Hence, the maximum reading period for air flow sensor (is an environment sensor) is 100 milliseconds.***	A
29	What should be done when there is a detection of critically high methane readings? ***Then, the operator must be informed within 1 second of within 2 seconds of a critically low air-flow reading and within 3 seconds of a failure in the operation of the pump.***	N
30	What should be done when there is a detection of critically high carbon monoxide readings? ***Then, the operator must be informed within 1 second of within 2 seconds of a critically low air-flow reading and within 3 seconds of a failure in the operation of the pump.***	N
31	What will happen if the pump is operated when the level of methane in the mine reaches a high value? ***Then, the mine is faced with the risk of explosion. This situation should not occur.***	N
32	How does the pump controller interact with the methane sensor? ***The methane sensor either polled or directly controlled by the pump controller.***	N
33	How does the pump controller interact with the carbon monoxide	N

	sensor? ***The carbon monoxide sensor either polled or directly controlled by the pump controller.***	
34	How does the pump controller interact with the water level sensors? ***The water level sensors communicate via interrupts.***	N
35	What does the water level sensor consist of? ***Water level sensor consist of High Water Sensor and Low Water Sensor.***	A
36	What is a sump? Assuming it is a reservoir for mining sludge.	N
37	What is the capacity of the sump? Will be specified later in the requirements documents	N
38	Is this a new system or improvement of an old system? New system.	N
39	Assuming gas level is high and water level is high, what to do? The pump should not be operated or stop operating and the methane sensor should alert the alarm to be activated.	B
40	What needs to be done when gas levels is high? The pump should not be operated or stop operating and methane sensor/ air flow sensor should alert the alarm to be activated.	B
41	Why do we expect an explosion? If there occur a situation where the pump still operates when level of methane is critical.	N
42	How fast the pump should operates? Will be specified later in the requirement documents.	N
43	What "the latter action" is as stated in the client's mail? To turn on the pump when the water reaches a high level.	B
44	Are there more stakeholders? Yes.	N
45	Is this system automated? Yes.	N
46	What level of accuracy the system require? Arbitrary.	N
47	What if the system fails? What precautions should we take? Evacuation.	N
48	What does 'should not be operated' means? It means 'should not perform any action, such as on or off'.	N
49	Must there be a manual override for safety? Yes.	N
50	Where do we pump water to? Surface to a specific area.	N
51	What sort of interaction does the user have? For instance, Windows/Linux or dial, etc? Windows.	N
52	What is the life expectancy of the system? Will be specified later in the requirements documents.	N
53	Should the system be operated when the level of carbon monoxide is high? No. The carbon monoxide sensor should alert the alarm to be activated.	B

54	Can the user shut down the system? Yes. The operator as user can shut down the system.	N
55	Will the alarm be enabled if too high? Yes. The alarm must be set.	U
56	Is the alarm turned on by the operator? No, by the sensors.	A
57	Are there any separate alarm for gas and water? No. There is only one central alarm.	A
58	Is it automated or manual for the on or off of the pump? Automated by the pump motor as well as manual by the operator to turn on and off the pump.	B
59	Is the system remote or onsite? Onsite.	N
60	Who or what will interact with the system? Operator, sensors and pump.	A
61	What are the major functionalities of the system? The major functionality of the system are pump operation, environment monitoring (the gases), operator interaction (via console) and system monitoring (archiving data in database).	U
62	What kinds of actions are given by the operator when the water level is high and at the same time carbon monoxide and methane level is also high? The operator should not operate the pump and the methane sensor and air flow sensor should trigger the alarm.	B
63	Are the operator for water and gas referring to the same operator? The same operator is in responsible for the entire system.	A
64	What about carbon monoxide high level? Pump should not operate or stop operating during the high level of carbon monoxide and the carbon monoxide sensor should trigger the alarm to be activated.	B
65	Who has the priority command (operator or the system)? Operator.	N

A-6 Composite Use Case Diagram for SBExp

Composite Use Case Diagram for Mine Drainage Control System for Scenario Based Experiment



Appendix B Assumptions Seeding Experiment (ASExp)

B-1 ASExp (Original or Non-Seeded Copy)

TUTORIAL: ASSUMPTIONS ELICITATION ACTIVITY

This tutorial is designed to gather data about underlying assumptions being made during analysis of Requirements Specification Document. You should work in groups of a maximum **four** students. Please choose a distinctive and easy to remember group name. Thank you for your participation. The outcomes of this tutorial will be returned to you and discussed in your next lecture or tutorial session.

Select a group name: [ ]

Scenario

You are appointed as a System Analyst for a new project to develop a Mine Drainage Control System. You are required to analyze the Requirements Specification Document for the mentioned system as illustrated below and complete **Task 1** and **Task 2**.

Requirements Specification Document for Mine Drainage Control System

1 System Objectives

The system is used to pump mine water, which collects in a sump at the bottom of a shaft, to the surface. The main safety requirement of the pump depends on the level of methane in the mine. (The main safety requirement is that the pump should not be operated when the level of methane gas in the mine reaches a high value due to the risk of explosion). It is assumed that the system will be implemented on a single processor with a simple memory mapped Input/Output architecture.

Assumption 1

2 Functional Requirements

2.1 Functional Requirements

The functional specification of the system may be divided into four components: the pump operation, the environment monitoring, the operator interaction, and system monitoring.

Assumption 3

Assumption 2

2.1.1 Pump Operation

(The required behavior of the pump controller is that it monitors the water levels in the sump). When the water reaches a high level ((or when requested by the operator)), the pump is turned on and the sump is drained until the water reaches the low level. At this point (or when requested by the operator), the pump is turned off. (The flow of water in the pipes can be detected if required). (The pump should be only allowed to operate if the methane level in the mine is below a critical level).

Assumption 4

Assumption 5

2.1.2 Environment Monitoring

(The environment must be monitored to detect the level of methane in the air,) (there is a level beyond which it is unsafe to mine or operate the pump). The environment must also be monitored to measure the level of carbon monoxide in the mine (and detects whether there is an adequate flow of air). Alarms must be activated if the level of gases or air-flow becomes critical.

Assumption 0

Assumption 6

2.1.3 Operator Interaction

The system is controlled via an operator's console. The operator is informed of all critical events.

2.1.4 System Monitoring

All the system events are to be stored in an archival database, and may be retrieved and displayed upon request.

2.2 Non-Functional Requirements

The non-functional requirements can be divided into three components: timing, dependability, and security. This requirements document mainly concerned with the timing requirements and consequently dependability and security will not be addressed here.

2.2.1 The maximum periods of reading the environment sensors may be dictated by legislation. These periods for all the sensors are assumed to be the same, which are 100 milliseconds.

2.2.2 Both the methane sensor and carbon monoxide sensors require 40 milliseconds in order for a reading to become available. Hence they require a deadline of 60 milliseconds.

2.2.3 The water flow object has two roles and it executes periodically. (While the pump is operational, it checks that there is water flow but while it is off (or disabled), it also checks that the water has stopped flowing).

Assumption 7

2.2.4 The water flow object is given a period of 1 second and it uses the results of two consecutive reading (to determine the actual state of the pump). The object is given tight deadline of 40 milliseconds (that is two readings will be at least 960 milliseconds but not more than 1040 milliseconds apart).

Assumption 8

2.2.5 The system should respond within 200 milliseconds (since it is assumed that water level detectors are event-driven).

Assumption 9

2.2.6 A deadline (from methane going high to the pump been disabled) of 200 milliseconds is assumed.

2.2.7 When there is a detection of critically high methane or carbon monoxide readings, then, the operator must be informed within 1 second of within 2 seconds of a critically low air-flow reading (and within 3 seconds of a failure in the operation of the pump).

Assumption 10

Group Name:

Task 1

Once you have studied the requirements for Mine Drainage Control System, sketch a use case diagram to elaborate the system.

Use Case Diagram:

Group Name:

Task 2

During the analysis of the requirements for Mine Drainage Control System, you might have made assumptions about these requirements. Besides, you might have made decisions based on assumptions which you believe to be true in order to develop your use case diagram. List all your assumptions in the table below if you have made any.

(Hint: your assumptions can be normally derived from the interpretations of requirements which you believe can be true or by considering statements or words such as "if-then", "what-if", "must" and "should" when you analyze the requirements).

No.	Assumption(s) made

B-2 ASExp (Seeded Copy)

TUTORIAL: ASSUMPTIONS ELICITATION ACTIVITY

This tutorial is designed to gather data about underlying assumptions being made during analysis of Requirements Specification Document. You should work in groups of a maximum **four** students. Please choose a distinctive and easy to remember group name. Thank you for your participation. The outcomes of this tutorial will be returned to you and discussed in your next lecture or tutorial session.

Select a group name: [ ]

Scenario

You are appointed as a System Analyst for a new project to develop a Mine Drainage Control System. You are required to analyze the Requirements Specification Document for the mentioned system as illustrated below and complete **Task 1** and **Task 2**.

Requirements Specification Document for Mine Drainage Control System

1 System Objectives

The system is used to pump mine water, which collects in a sump at the bottom of a shaft, to the surface. The main safety requirement of the pump depends on the level of methane in the mine. It is assumed that the system will be implemented on a single processor with a simple memory mapped Input/Output architecture.

2 Functional Requirements

2.1 Functional Requirements

The functional specification of the system may be divided into four components: the pump operation, the environment monitoring, the operator interaction, and system monitoring.

2.1.1 Pump Operation

When the water reaches a high level, the pump is turned on and the sump is drained until the water reaches the low level. At this point (or when requested by the operator), the pump is turned off.

2.1.2 Environment Monitoring

The environment must also be monitored to measure the level of carbon monoxide in the mine. Alarms must be activated if the level of gases or air-flow becomes critical.

2.1.3 Operator Interaction

The system is controlled via an operator's console. The operator is informed of all critical events.

2.1.4 System Monitoring

All the system events are to be stored in an archival database, and may be retrieved and displayed upon request.

2.2 Non-Functional Requirements

The non-functional requirements can be divided into three components: timing, dependability, and security. This requirements document mainly concerned with the timing requirements and consequently dependability and security will not be addressed here.

2.2.1 The maximum periods of reading the environment sensors may be dictated by legislation. These periods for all the sensors are assumed to be the same, which are 100 milliseconds.

2.2.2 Both the methane sensor and carbon monoxide sensors require 40 milliseconds in order for a reading to become available. Hence they require a deadline of 60 milliseconds.

2.2.3 The water flow object has two roles and it executes periodically.

2.2.4 The water flow object is given a period of 1 second and it uses the results of two consecutive reading. The object is given tight deadline of 40 milliseconds (that is two readings will be at least 960 milliseconds but not more than 1040 milliseconds apart).

2.2.5 The system should respond within 200 milliseconds.

2.2.6 A deadline (from methane going high to the pump been disabled) of 200 milliseconds is assumed.

2.2.7 When there is a detection of critically high methane or carbon monoxide readings, then, the operator must be informed within 1 second of within 2 seconds of a critically low air-flow reading.

Group Name:

Task 1

Once you have studied the requirements for Mine Drainage Control System, sketch a use case diagram to elaborate the system.

Use Case Diagram:

Group Name:

Task 2

During the analysis of the requirements for Mine Drainage Control System, you might have made assumptions about these requirements. Besides, you might have made decisions based on assumptions which you believe to be true in order to develop your use case diagram. List all your assumptions in the table below if you have made any.

(Hint: your assumptions can be normally derived from the interpretations of requirements which you believe can be true or by considering statements or words such as "if-then", "what-if", "must" and "should" when you analyze the requirements).

No.	Assumption(s) made

B-3 Checklist of Seeded Assumptions

Check List of Seeded Assumptions elicited from Requirements Specification Document for Mine Drainage Control System (MDCS)

Seeded Assumptions		Y/N
No.	**Descriptions**	
0	According to requirement 2.1.2, we assume that the environment must be monitored to detect the level of methane in the air.	☐
1	The pump should be only operated when the level of methane is below a critical level.	☐
2	Pump controller is responsible for monitoring water level in the sump.	☐
3	Operator can turn on the pump if required (when the water level reaches a high level) and the sump is drained until the water reaches the low level.	☐
4	The flow of the water can be detected if required.	☐
5	The environment monitors the level of methane and there is a level beyond which is unsafe to mine or operate the pump.	☐
6	The detection of whether there is an adequate flow of air is done by the monitor when it measures the level of carbon monoxide in the mine.	☐
7	In section 2.2.4, there are two roles for “water flow object” which is to check whether there is water flow when the pump is on and to check whether the water has stopped flowing when the pump is off.	☐
8	The actual state of the pump is determined by using the results of two consecutive reading of water flow object. The water flow object is given a period of 1 second.	☐
9	In requirement 2.2.6, it is assumed that the water level detectors are event-driven in order for the system to respond within 200 milliseconds.	☐
10	The operator must be informed within 3 seconds of a failure in the operation of the pump when there is a detection of critically high methane or carbon monoxide readings.	☐

Check List of Original or Non-Seeded Assumptions for Mine Drainage Control System (MDCS)

No	Non-Seeded Assumptions	Y/N
11	The scope of the Requirements Specification Document for MDCS is only on the timing requirements.	☐
12	Pump controller subsystem consists of Water Flow Sensor.	☐
13	Pump controller subsystem consists of Water Level Sensor.	☐
14	The “water flow object” in section 2.2.4 and 2.2.5 is referring to Water Flow Sensor.	☐
15	Water Level Sensor consists of High Water Sensor (also known as High Water Level Detector).	☐
16	Water Level Sensor consists of Low Water Sensor (also known as Low Water Level Detector).	☐
17	Environment monitor subsystem consists of Carbon Monoxide Sensor.	☐
18	Environment monitor subsystem consists of Methane Sensor.	☐
19	Environment monitor subsystem consists of Air Flow Sensor.	☐
20	Operator console is responsible for the interaction between the system and the operator.	☐
21	Data logger is responsible for logging operational data (levels of water in the sump).	☐
22	Data logger is responsible for logging environmental data (levels of methane in the mine).	☐
23	Data logger is responsible for logging environmental data (levels of carbon monoxide in the mine).	☐
24	Data logger is responsible for logging environmental data (levels of air flow in the mine).	☐
25	The water levels should be measured and recorded at all time.	☐
26	The methane levels should be measured and recorded at all time.	☐
27	The carbon monoxide levels should be measured and recorded at all time.	☐

28	The air flow levels should be measured and recorded at all time.	☐
29	The Requirements Specification Document for MDCS is not concerned with the details of the Data Logger and Operator Console. However, it is a requirement that they only delay the real time threads for a bounded time. Hence, it is assumed that their interfaces contain protected objects. (Note: Protected objects are objects which may control when invocations of their operations are executed, but do not spontaneously invoke operations in other objects; in general protected objects may not have arbitrary synchronization.)	☐
30	The information about Methane level is obtained from the methane sensor.	☐
31	The information about carbon monoxide level is obtained from the carbon monoxide sensor.	☐
32	The information about water level is obtained from the water level sensor.	☐
33	In section 2.1.2, the phrase “the gas levels” in the last sentence is referring to methane and carbon monoxide.	☐
34	In section 2.1.3, the phrase “all critical events” is referring to high value or critical level of methane level in the mine.	☐
35	In section 2.1.3, the phrase “all critical events” is referring to critical level of carbon monoxide in the mine.	☐
36	In section 2.1.3, the phrase “all critical events” is referring to critical level of air flow in the mine.	☐
37	In section 2.1.3, the phrase “all critical events” is referring to pump fault.	☐
38	In section 2.1.3, the phrase “all critical events” is referring to the situation where the water level is high and at the same time, the methane level is also high.	☐
39	In section 2.1.3, the phrase “all critical events” is referring to the situation where the water level is high and at the same time, the carbon monoxide level is also high.	☐
40	In section 2.1.3, the phrase “all critical events” is referring to the situation where the water level is high and at the same time, the air flow level is critical.	☐
41	In section 2.1.3, the phrase “all critical events” is referring to the situation where the water level is high and at the same time, the methane and carbon monoxide is high and the there is inadequate air flow in the mine.	☐

42	In section 2.1.4, the phrase “all the system events” is referring to levels of water in the pump.	☐
43	In section 2.1.4, the phrase “all the system events” is referring levels of methane gas in the mine.	☐
44	In section 2.1.4, the phrase “all the system events” is referring to levels of carbon monoxide gas in the mine.	☐
45	In section 2.1.4, the phrase “all the system events” is referring to levels of air flow in the mine.	☐
46	High Water Sensor communicates via interrupts.	☐
47	Low Water Sensor communicates via interrupts.	☐
48	Methane Sensor either polled or directly controlled.	☐
49	Carbon Monoxide Sensor either polled or directly controlled.	☐
50	Air Flow Sensor either polled or directly controlled.	☐
51	Water Flow Sensor either polled or directly controlled.	☐
52	Assume that a sump in this requirements document referring to a reservoir for mining sludge.	☐
53	Assume that the water which is drained out from the sump to surface is diverted to a specific area.	☐
54	The capacity of the sump will be specified in the detail requirements specification. Assume that this information is not required at this stage.	☐
55	The critical level of methane is referring to a particular figure which will be specified in the detail requirements specification. Assume that this information is not required at this stage.	☐
56	This is a new system which needs to be developed.	☐
57	Operator can monitor all the critical events via the console.	☐
58	Operator can request to retrieve and view system events which stored in the database via console.	☐
59	The operator has the priority commands to operate the MDCS compared to the automated MDCS system by itself.	☐
60	Assume that the operator have the privilege to shut down the system.	☐

61	The pump should not be operated or stop operating when the water level and the methane level are high and the methane sensor should trigger the alarm.	☐
62	If the methane level gets too high during the period where the water is being drained out of the sump, then, the draining process should be stoped and the methane sensor should trigger the alarm.	☐
63	The pump should not be operated or stop operating when the water level and the carbon monoxide level are high and the air flow sensor should trigger the alarm.	☐
64	The pump should not be operated or stop operating when the carbon monoxide level is high and the air flow sensor should trigger the alarm.	☐
65	When the gases level is high (methane and carbon monoxide), the pump should not be operated or stop operating and methane sensor and air flow sensor should trigger the alarm.	☐
66	When the water level is high and at the same time, the methane and carbon monoxide levels are high and there is an inadequate flow of air the pump should not be operated or stop operating.	☐
67	If the sump overflows and the water level do not come down, then, the pump should be turned on automatically. If the pump did not turn on automatically, then, the operator should turn on the pump. If the pump is not working (pump fault), then, the system should not be operated or stop operating and the alarm should be triggered.	☐
68	The carbon monoxide sensor will alert the alarm when the carbon monoxide level is above the critical level in the mine.	☐
69	The methane sensor will alert the alarm when the methane level is above the critical level in the mine.	☐
70	We assume there will be an explosion if the pump continues operating when the level of methane is above the critical level in the mine.	☐
71	We assume there will be an explosion if the pump continues operating when the level of carbon monoxide is above the critical level or there's no adequate flow of air in the mine.	☐
72	The Mine Drainage Control system is an automated system.	☐
73	The action to turn on and off the pump is automated by the pump controller/pump motor as well as manual by the operator.	☐
74	The accuracy of the system is assumed to be arbitrary.	☐

75	If there is a system failure, alarm will be activated and evacuation should be alerted in the mine as the safety precautions.	☐
76	The operator can override manually the action of the pump.	☐
77	The console operator is Windows based system.	☐
78	The life expectancy of the system will be specified in the detail requirements document. Assume that this information is not required at this stage.	☐
79	There are more stakeholders for MDCS besides the operator.	☐
80	The high water level is referring to a particular figure which will be specified in the detail requirements specification. Assume that this information is not required at this stage.	☐
81	The low water level is referring to a particular figure which will be specified in the detail requirements specification. Assume that this information is not required at this stage.	☐
82	There is only one central alarm in MDCS system.	☐
83	The system is controlled on site.	☐
84	The person and hardware which interacts with the system are operator, pump, water level sensor, water flow sensor, methane sensor, carbon monoxide sensor and air flow sensor.	☐
85	Assume that alarm is activated when the water reaches high or low level.	☐
86	If system does not respond within 200 milliseconds. The entire system should be turned off.	☐
87	The operator can change the level of gases.	☐
88	The operator can turn off the system within 5 seconds.	☐
89	The gas sensor can sense other unwanted gases and inform operator.	☐
90	Mine Drainage Control System (MDCS) should be implemented on a single processor with a simple memory mapped input/output architecture.	☐
91	The reading periods for all the sensors are assumed to be the same which is 100 milliseconds.	☐
92	Assumed that the deadline for methane going high to the pump been disabled is 200milliseconds.	☐

93	The system should respond within 200 milliseconds.	☐
94	We assume that low air flow means either high methane or high carbon monoxide flow.	☐
95	It is assumed that water flow element respond within 1 second.	☐
96	It is assumed that the water is always flowing.	☐
97	We assume that system should respond in 200 milliseconds.	☐
98	Assume that there are always two previous readings and the different between them is used by the water flow object.	☐
99	Assume that the console is always in the operating condition.	☐
100	Assume that there is only one console.	☐
101	We assume that all events are time synchronized.	☐

B-5 Pilot Trial (Seeded Copy)

Age: _____ **Occupation: _____**
Start time: _____ **End time:_____**

Objective of Case Study: Eliciting Assumptions from Requirements Specification Document for Mine Drainage Control System

Description: You are given a sample Requirements Specification Document (RSD) for Mine Drainage Control System illustrated as below. You are required to analyze this RSD and complete task 1 and task 2.

Requirements Specification Document: Mine Drainage Control System

1 System Objectives

The system is used to pump mine water, which collects in a sump at the bottom of a shaft, to the surface. The main safety requirement is that the pump should not be operated when the level of methane gas in the mine reaches a high value due to the risk of explosion. It is assumed that the system will be implemented on a single processor with a simple memory mapped Input/Output architecture.

2 Functional Requirements

2.1 Functional Requirements

2.1.1 Pump Operation
The required behavior of the pump controller is that it monitors the water levels in the sump. When the water reaches a high level, the pump is turned on and the sump is drained until the water reaches the low level. At this point (or when requested by the operator), the pump is turned off. The flow of water in the pipes can be detected if required.

2.1.2 Environment Monitoring
The environment must be monitored to detect the level of methane in the air. The monitoring also measures the level of carbon monoxide in the mine and detects whether there is an adequate flow of air. Alarms must be activated if the gas levels or air-flow become critical.

2.1.3 Operator Interaction
The system is controlled via an operator's console. The operator is informed of all critical events.

2.1.4 System Monitoring
All the system events are to be stored in an archival database, and may be retrieved and displayed upon request.

2.2 Non-Functional Requirements

2.2.1 The maximum periods of reading the environment sensors may be dictated by legislation. These periods are 100 milliseconds.

2.2.2 The pump should not operate when the methane level is critically high.

2.2.3 Both sensors require 40 milliseconds in order for a reading to become available. Hence they require a deadline of 60 milliseconds.

2.2.4 The water flow object has two roles and it executes periodically. While the pump is operational, it checks that there is water flow.

2.2.5 The water flow object is given a period of 1 second and it uses the results of two reading to determine the actual state of the pump. The object is given tight deadline of 40 milliseconds (that is two readings will be at least 960 milliseconds but not more than 1040 milliseconds apart).

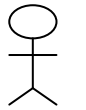
2.2.6 The system should respond within 200 milliseconds since it is assumed that water level detectors are event-driven.

2.2.7 A deadline (from methane going high to the pump been disabled) of 200 milliseconds is assumed.

2.2.8 When there is a detection of critically high methane or carbon monoxide readings, then the operator must be informed within 1 second of within 2 seconds of a critically low air-flow reading and within 3 seconds of a failure in the operation of the pump.

Task 1: Once you have analyzed this RDD, develop a Use Case Diagram to elaborate the Mine Control System. Use Case Diagram is one of the UML (Unified Modeling Language) diagrams used to identify the primary elements and processes that form the system. The primary elements are termed as "actors" and the processes are called "use cases." The use case diagram shows which actors interact with each use case. Some examples of use case diagrams are shown as below.

Use the shapes as shown below to develop the Use Case Diagram for MDCS.

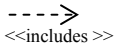


Actor: An actor portrays any entity (or entities) that perform certain roles in a given system. The different roles the actor represents are the actual business roles of users in a given system.

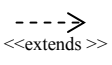
Use Case: A use case is a visual representation of functionalities in a system.

System Boundary: A system boundary defines the scope of what a system will be. (Note: a system cannot have infinite functionality).

Association: A link between an actor and a use case. It indicates which actors interact with the system to complete the various tasks.



Includes: To show that one use case includes the task described by another use case.



Extends: To show that one use case extends the task described by another use case.

Generalization: An informal way of showing that one use case is similar to another use case, but with extra functionality. One use case inherits the functionality represented by another use case and adds some additional behaviors to it.

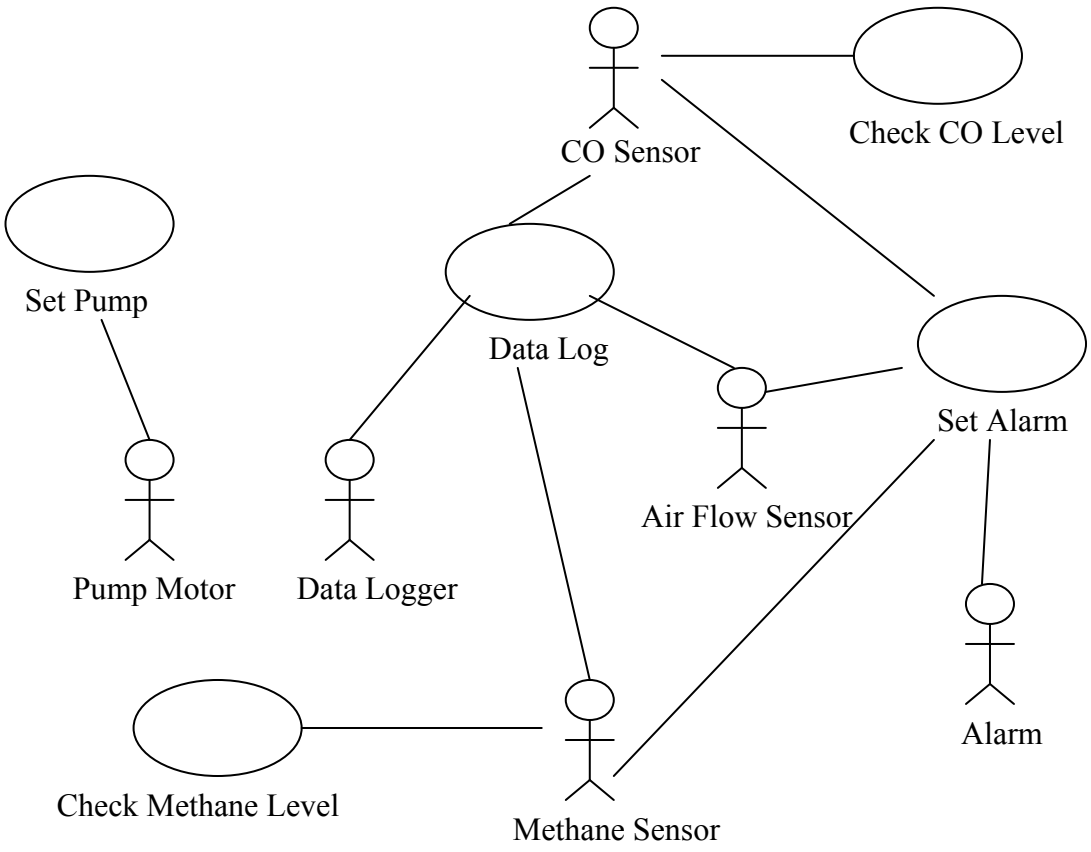
Use Case Diagram :

Task 2: List all the assumptions which you might have possibly used in order to develop this process flow chart and briefly describe why you assume such. List your answers for this task in the Table below. You can use extra pages if you need more space to list your assumptions. (Hint: your assumptions can be normally derived from the interpretations you believe can be true in this case or statements which you interpret by analyzing conditional statements such as 'if-then').

No	Description of assumptions	Why you assume such?

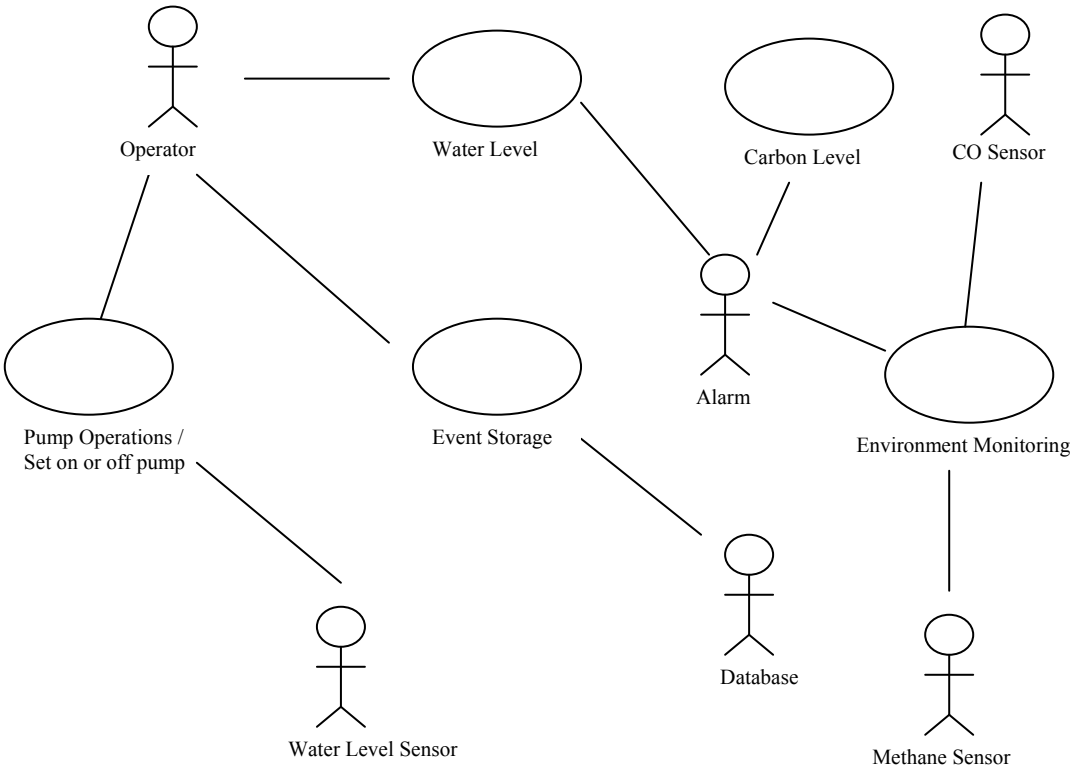
B-6 An Incomplete Use Case Diagram

An Incomplete Use Case Diagram for Mine Drainage Control System



B-7 Composite Use Case Diagram for ASExp

Composite Use Case Diagram for Mine Drainage Control System for Assumptions Seeding Experiment



B-8 List of Assumptions gathered by the Subjects

No	Descriptions of Assumptions	Seeded	Non-Seeded	
			Related	Not-related
1	As per requirement 2.1.1, water is always flowing. Was been assumed.		1	
2	As per requirement 2.1.2, we assume that all workers are alerted by hearing alarm system only.			1
3	As per requirement 2.1.3, we assume that an operator is always working on a console.			1
4	As per requirement 2.2.1, the reading periods for all the sensors are fixed.			1
5	As per requirement 2.2.5, we assume that system should respond in 200 milliseconds.			1
6	As per requirement 2.2.4, we assume that there are always two pervious readings used by water flow object.	1 (A8)		
7.	As per requirement 2.2.7, we assume that operator is alerted of low air flow reading on console only.		1	
8	We assume that console is always working and only one console is present.			1
9	We assume that all events are time synchronised.			1
	TOTAL BOYS	**1**	**2**	**6**
10	According to requirement 2.1.2, we assume that the alarm is activated when water reaches high/low level.		1	
11	From 2.1.1, assume that pump will be turned off manually or automatically in case of critical conditions.	1 (A3)		
12	The alarm will be activated if there is an accident.			1
13	The operator can turn off system when any other system stops working.		1	
14	The operator can access the database for monitoring.		1	
15	System should display status in case of critical conditions.			1
16	System should record data at time of accidents.			1
17	If system does not respond within 200 milliseconds, the entire system is turned off.			1
18	The operator can change the level of gases.		1	

19	The operator can turn off system within 5 seconds.		1	
20	The gas sensors sense other unwanted gases and inform operator		1	
	TOTAL FOR SUN	**1**	**6**	**4**
21	Assumptions stated that it is implemented with a single processor.			1
22	Periods for all sensors assumed same (Non-Functional-2.2.1).			1
23	Non-functional -2.2.6, deadlines for methane is not stated. Only stated about the assumption about carbon monoxide deadline.		1	
24	System maintains the time is assumed (non-functional 2.2.5).			1
25	Exact values for low and high are not mentioned, we assume.			1
26	We assume that low air flow means either high methane or high carbon monoxide flow (non-functional 2.2.7).	1 (A6)		
27	It is assumed that water flow element responds within time (non-functional 2.2.4).			1
	TOTAL FOR NAD	**1**	**1**	**5**

Superset of Assumptions elicited from Requirements Specification Document for Mine Drainage Control System (MDCS)

Seeded Assumptions		Y/N
No.	**Descriptions**	
1	Mine Drainage Control System (MDCS) should be implemented on a single processor with a simple memory mapped input/output architecture.	☐
2	The scope of the Requirements Specification Document for MDCS is only on the timing requirements.	☐
3	This is a new system which needs to be developed.	☐
4	The major functionality of the system are pump operation, environment monitoring (the gases), operator interaction (via console) and system monitoring (archiving data in database).	☐
5	The Mine Drainage Control system is an automated system.	☐
6	The accuracy of the system is assumed to be arbitrary.	☐
7	Assume that a sump in this requirements document referring to a reservoir for mining sludge.	☐
8	Assume that the water which is drained out from the sump to surface is diverted to a specific area.	☐
9	The capacity of the sump will be specified in the detail requirements specification. Assume that this information is not required at this stage.	☐
10	The life expectancy of the system will be specified in the detail requirements document. Assume that this information is not required at this stage.	☐
11	There are more stakeholders for MDCS besides the operator.	☐
12	There is only one central alarm in MDCS system.	☐
13	The system is controlled on site.	☐
14	The person and hardware which interacts with the system are operator, pump, water level sensor, water flow sensor, methane sensor, carbon monoxide sensor and air flow sensor.	☐
15	The security of the system will not be discussed at this stage. It	☐

	will be specified in the detail requirements specification.	
16	The reliability of the system will not be discussed at this stage. It will be specified in the detail requirements specification.	☐
17	Environment monitor subsystem consists of Carbon Monoxide Sensor.	☐
18	Environment monitor subsystem consists of Methane Sensor.	☐
19	Environment monitor subsystem consists of Air Flow Sensor.	☐
20	We assume that the environment must be monitored to detect the level of methane in the air.	☐
21	There is a level beyond (for methane in the air) which is unsafe to mine or operate the pump.	☐
22	The information about Methane level is obtained from the methane sensor.	☐
23	Methane Sensor either polled or directly controlled.	☐
24	The critical level of methane is referring to a particular figure which will be specified in the detail requirements specification. Assume that this information is not required at this stage.	☐
25	The information about carbon monoxide level is obtained from the carbon monoxide sensor.	☐
26	Carbon Monoxide Sensor either polled or directly controlled.	☐
27	In section 2.1.2, the phrase “the gas levels” in the last sentence is referring to methane and carbon monoxide.	☐
28	Air Flow Sensor either polled or directly controlled.	☐
29	The detection of whether there is an adequate flow of air is done by the monitor when it measures the level of carbon monoxide in the mine.	☐
30	We assume that low air flow means either high methane or high carbon monoxide flow.	☐
31	The gas sensor can sense other unwanted gases and inform operator.	☐
32	It is assumed that the maximum reading period for environment sensors may be dictated by legislation and in this case will be 100 milliseconds. Hence, the maximum reading period for methane sensor (is an environment sensor) is 100 milliseconds.	☐

33	It is assumed that the maximum reading period for environment sensors may be dictated by legislation and in this case will be 100 milliseconds. Hence, the maximum reading period for carbon monoxide sensor (is an environment sensor) is 100 milliseconds.	☐
34	It is assumed that the maximum reading period for environment sensors may be dictated by legislation and in this case will be 100 milliseconds. Hence, the maximum reading period for air flow sensor (is an environment sensor) is 100 milliseconds.	☐
35	Pump controller is responsible for monitoring water level in the sump.	☐
36	The speed of the pump controller/pump motor will be specified in the detail requirements specification.	☐
37	It is assumed that the water is always flowing.	☐
38	The flow of the water can be detected if required.	☐
39	The information about water level is obtained from the water level sensor.	☐
40	Pump controller subsystem consists of Water Flow Sensor.	☐
41	Pump controller subsystem consists of Water Level Sensor.	☐
42	Water Level Sensor consists of High Water Sensor (also known as High Water Level Detector).	☐
43	Water Level Sensor consists of Low Water Sensor (also known as Low Water Level Detector).	☐
44	High Water Sensor communicates via interrupts.	☐
55	Low Water Sensor communicates via interrupts.	☐
56	Water Flow Sensor either polled or directly controlled.	☐
47	The “water flow object” in section 2.2.4 is referring to Water Flow Sensor.	☐
48	The “water flow object” in section 2.2.5 is referring to Water Flow Sensor.	☐
49	In requirement 2.2.4, there are two roles for “water flow object”. The first role is to check whether there is water flow when the pump is on.	☐
50	In requirement 2.2.4, there are two roles for “water flow object”.	☐

	The second role is to check whether the water has stopped flowing when the pump is off.	
51	As per requirements 2.2.4, we assume that there are always two previous readings used by water flow object.	☐
52	The actual state of the pump is determined by using the results of two consecutive reading of water flow object.	☐
53	It is assumed that water flow object respond within 1 second.	☐
54	In requirement 2.2.5, it is assumed that the water level detectors/water level sensors are event-driven in order for the system to respond within 200 milliseconds.	☐
55	The pump should be only operated when the level of methane is below a critical level.	☐
56	The action to turn on and off the pump is automated by the pump controller/pump motor as well as manual by the operator.	☐
57	The high water level is referring to a particular figure which will be specified in the detail requirements specification. Assume that this information is not required at this stage.	☐
58	The low water level is referring to a particular figure which will be specified in the detail requirements specification. Assume that this information is not required at this stage.	☐
59	Assume that alarm is activated when the water reaches high or low level.	☐
60	Operator can request to retrieve and view system events which stored in the database via console.	☐
61	Operator can monitor all the critical events via the console.	☐
62	Assume that the operator have the privilege to shut down the system.	☐
63	The operator can change the level of gases.	☐
64	The operator can turn off the system within 5 seconds.	☐
65	The operator has the priority commands to operate the MDCS compared to the automated MDCS system by itself. (For an example, the operator can override manually the action of the pump).	☐
66	Operator can turn on the pump manually if required (when the water level reaches a high level) and the sump is drained until the water reaches the low level and then turn the pump off.	☐

67	The operator must be informed within 3 seconds of a failure in the operation of the pump when there is a detection of critically high methane or carbon monoxide readings.	☐
68	Operator console is responsible for the interaction between the system and the operator.	☐
69	The console operator is Windows based system.	☐
70	Assume that the console is always in the operating condition.	☐
71	Assume that there is only one console.	☐
72	Data logger is responsible for logging operational data (levels of water in the sump) into the archival database at all times.	☐
73	Data logger is responsible for logging environmental data (levels of methane in the mine) into the archival database at all times.	☐
74	Data logger is responsible for logging environmental data (levels of carbon monoxide in the mine) into the archival database at all times.	☐
75	Data logger is responsible for logging environmental data (levels of air flow in the mine) into the archival database at all times.	☐
76	The Requirements Specification Document for MDCS is not concerned with the details of the Data Logger and Operator Console. However, it is a requirement that they only delay the real time threads for a bounded time. Hence, it is assumed that their interfaces contain protected objects. (Note: Protected objects are objects which may control when invocations of their operations are executed, but do not spontaneously invoke operations in other objects; in general protected objects may not have arbitrary synchronization.)	☐
77	In section 2.1.3, the phrase “all critical events” is referring to high value or critical level of methane level in the mine.	☐
78	In section 2.1.3, the phrase “all critical events” is referring to critical level of carbon monoxide in the mine.	☐
79	In section 2.1.3, the phrase “all critical events” is referring to critical level of air flow in the mine.	☐
80	In section 2.1.3, the phrase “all critical events” is referring to pump fault.	☐
81	In section 2.1.3, the phrase “all critical events” is referring to the situation where the water level is high and at the same time, the	☐

	methane level is also high.	
82	In section 2.1.3, the phrase “all critical events” is referring to the situation where the water level is high and at the same time, the carbon monoxide level is also high.	☐
83	In section 2.1.3, the phrase “all critical events” is referring to the situation where the water level is high and at the same time, the air flow level is critical.	☐
84	In section 2.1.3, the phrase “all critical events” is referring to the situation where the water level is high and at the same time, the methane and carbon monoxide is high and the there is inadequate air flow in the mine.	☐
85	In section 2.1.4, the phrase “all the system events” is referring to levels of water in the pump.	☐
86	In section 2.1.4, the phrase “all the system events” is referring levels of methane gas in the mine.	☐
87	In section 2.1.4, the phrase “all the system events” is referring to levels of carbon monoxide gas in the mine.	☐
88	In section 2.1.4, the phrase “all the system events” is referring to levels of air flow in the mine.	☐
89	The pump should not be operated or stop operating when the water level and the methane level are high and the methane sensor should trigger the alarm.	☐
90	If the methane level gets too high during the period where the water is being drained out of the sump, then, the draining process should be stoped and the methane sensor should trigger the alarm.	☐
91	The pump should not be operated or stop operating when the water level and the carbon monoxide level are high and the air flow sensor should trigger the alarm.	☐
92	The pump should not be operated or stop operating when the carbon monoxide level is high and the air flow sensor should trigger the alarm.	☐
93	When the gases level is high (methane and carbon monoxide), the pump should not be operated or stop operating and methane sensor and air flow sensor should trigger the alarm.	☐
94	When the water level is high and at the same time, the methane and carbon monoxide levels are high and there is an inadequate flow of air the pump should not be operated or stop operating.	☐

95	If the sump overflows and the water level do not come down, then, the pump should be turned on automatically. If the pump did not turn on automatically, then, the operator should turn on the pump. If the pump is not working (pump fault), then, the system should not be operated or stop operating and the alarm should be triggered.	☐
96	The carbon monoxide sensor will alert the alarm when the carbon monoxide level is above the critical level in the mine.	☐
97	The methane sensor will alert the alarm when the methane level is above the critical level in the mine.	☐
98	We assume there will be an explosion if the pump continues operating when the level of methane is above the critical level in the mine.	☐
99	If there is a system failure, alarm will be activated and evacuation should be alerted in the mine as the safety precautions.	☐
100	We assume that system should respond in 200 milliseconds.	☐
101	Assumed that the deadline for methane going high to the pump been disabled is 200milliseconds.	☐
102	If system does not respond within 200 milliseconds. The entire system should be turned off.	☐
103	We assume that all events are time synchronized.	☐

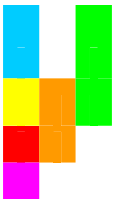
Appendix C Observation on Students' Projects

C-1 Meeting Timetables for Team B

CITS3200: Professional Computing - Group B Meeting and Availability Schedule (V1)

	Mon	Tues	Wed	Thur	Fri
9AM			P/C LC		
10AM	P		RW		
11AM	P,N	A	RW		
12PM	P,N	A			
1PM	RD,M,N	A,P	N,RD,M	RW	RW
2PM	A,N,RD,M,P	A,P	N,RD,M,P	RW	N,RW
3PM	MEETING AT MPSL	A	A,N,RD,M		N,RW
4PM	ALL	A	A,N,RD,M	N	N,
5PM	ALL	A	A,N,RD,M	N	N,

Availability Key



People Key

A	Andrew	Tester	
P	Patrick	Tester Developer	**
N	Nic	Lead	**
RD	Roland	Developer	
M	Mark	Developer	
RW	Robbie	Project Manager	**

CITS3200: Professional Computing - Group B Meeting and Availability Schedule (V2)

	Mon	Tues	Wed	Thurs	Fri
9AM			P/C LC		
10AM			RW		
11AM	N	A	RW		
12PM	N	A			
1PM	RD,M,N	A	N,RD,M	RW	RW
2PM	A,N,RD,M	A	N,RD,M	RW	N,RW
3PM	MEETING AT MPSL	A	A,N,RD,M		N,RW
4PM	ALL	A	A,N,RD,M	N	N,
5PM	ALL	A	A,N,RD,M	N	N,

Availability Key

Program Manager
Developer
Tester
All

People Key

A	Andrew	Tester Developer	**
N	Nic	Lead	**
RD	Roland	Developer	
M	Mark	Developer	
RW	Robbie	Project Manager	**

C-2 Assumptions List extracted from observation on Team B

List of Assumptions collected from the observation on Team B's meetings

Assumption(s)				Source (Statement leading to the Assumption)
No	Description	Type	Owner	
A01	The developers assumes that the client is comfortable with either one of the options he had mentioned	Conscious	Developers	"The system should either fetches or displays the descriptions from a database which keeps information for – option 1: some specified duration, or option 2: from database which updates information regularly" – by Client
A02	The developers assume that they do not need to build an authentication feature for the system as they believe the users are already authenticated	Conscious	Developers	"So, we can assume that the users already authenticated" – by Developers
A03	The client is assuming that the developers will be able to design the webpage correctly according to their judgments	Conscious	Client	Unsure of certain criteria or design for the webpage. He has left this for the developers to try and figure it out. – by Client
A04	The client is assuming that the developers will not work on assumptions during the	Conscious	Client	Client is available to be contacted by mail anytime to verify any doubts which the developers tend to come across in the

	development process			future. He had also mentioned that to the developers '*don't* work on Assumptions' – by Client
A05	The developers assumed that the system needs to generate and display all the options for an item chosen by the user.	Unconscious	Developers	'*Let me clear this*'. We don't want the system to work out all the options and display to the users because it will be too long for a menu. (ie. If there are four sockets available, then the system should list down all the option for the first socket. Then the system should ask the users whether the user want to go ahead to choose the second socket's options) – by Client
A06	The client assumes that the functional requirements named 'Automation' in the RAD which is referring to particular constraint must have been written by one of the developers.	Conscious	Developers	"I didn't write that constraint. That RAD was sent to me by your PM. So, I presume it must be written by one of you" – by Client
A07	***d2*** *might have an understanding about the client's explanation which might not be accurate but he will be passing this information to*	Conscious	Developer 1 (***d1***) and Developer 2 (***d2***)	One of the developers. ***d1*** questioned the other developer, ***d2*** whether ***d2*** understood the explanation pertaining to the examples illustrated by the client. ***d2*** answered saying that he understood. Then, ***d1*** said he will go through

	d1			that part with ***d2*** after the meeting with the client. – by ***d1*** and***d2***
A08	***d1*** *assumes that* ***d2*** *understood about the slot preference methods from the client's but* ***d2****'s understanding might not be accurate but he will be passing this information to the* ***d1***	Conscious	Developer 1 (***d1***) and Developer 2 (***d2***)	***d1*** told ***d2*** to explain to him how the slot preferences work since ***d2*** claims that he knows how it works? – by ***d1*** and ***d2***
A09	PM assumes that they do not need a large number of programmers for the project	Conscious	PM	According to the details of the requirements, the PM said that it seems like they should have more programmers for the project. However, he said from his point of view, he thinks or suggest that they might not need a large number of programmers. – by PM
A10	The developers might assume their own definition of the word 'smart selection'	Conscious	Developers	PM said that acceptance test for the prototype should contain smart selection. – by PM
A11	The word 'other data as well' allow the developers to assume the type of data used in the system	Conscious	Developers	PM said that price integrity should include other data as well. – by PM

C-3 Assumptions List extracted from observation on Team A

List of Assumptions collected from the observation on Team A's meetings

Assumption(s)				Source (Statement leading to the Assumption)
No	Description	Type	Owner	
A01	The developers assumes that they know about the module which the clients were referring	Unconscious	Developers	"Mentioned about module to the developers when explaining the requirements." – by Client
A02	The developers can assume the definition of 'good' in this context.	Unconscious, Business and Technical	Developers	"We will look at the progress of the webpage development as the project involves. We have intentions to use the software permanently since there is possibility for commercialization in the future. *So, it has been to look good.*" – by Clients
A03	The client is assuming that the development team will be able to do the programming task according to the statement they heard from the developers	Conscious	Clients	"As of now, we haven't started to discuss the details. After today's discussion, we will be able to start. I think there's one of the group member will be able to do the programming task". – by Developers

References

[1] The Challenges of Complex IT Projects (2004), The Royal Academy of Engineering. Available: http://www.bcs.org/server.php?show=conWebDoc.1167 [Accessed: 11 June 2009].

[2] Leffingwell, D. and Widrig, D. (1999) *Managing Software Requirements: A Unified Approach*, Addison-Wesley, pp. 5-13.

[3] Weinberg, G. M. (1971) *The Psychology of Computer Programming*, Van Nostrand Reinhold.

[4] Davis, A. M. (1995) *201 Principles of Software Development*, McGraw-Hill.

[5] Vinter, O., Poulsen, P. M., Thomsen, J. M., Nissen, K. and Andersen, O., "The Prevention of Errors through Experience-Driven Test Efforts", Tech. Project D-259, Delta, Horsholm, Denmark, 1996.

[6] The Standish Group Report (1995), "The Standish Group Report Chaos". Available: http://www.spinroot.com/spin/Doc/course/Standish_Survey.htm [Accessed: 28 Feb 2008].

[7] Sheldon, F. T., Kavi, K. M., Tausworthe, R. C., Yu, J. T., Brettschneider, R. and Everett, W. W., "Reliability Measurement from Theory to Practice", IEEE Software, July 1992, pp. 13-20, 1992.

[8] Hooks, I. F. and Farry, K. A. (2000) *Customer Centered Products: Creating Successful Products Through Smart Requirements Management*, Amacom, pp. 5-10.

[9] Basili, V. R. and Weiss, D. M., Evaluation of a software requirements document by analysis of change data. In Proceedings of the 5th International Conference on Software Engineering, San Diego, CA, March 1981.

[10] Boehm, B., "Improving Software Productivity," IEEE Computer, September 1987, pp43-57, 1987.

[11] Boehm, B. W., and Papaccio, P. N., "Understanding and Controlling Software Costs", IEEE Transactions of Software Engineering, Oct 1988.

[12] Boehm, B. and Basili, V. R., "Software Defect reduction Top 10 List", Software Management, Jan 2001.

[13] Vinter, O, "Bug Taxonomy and Statistics by Boris Beizer, as amended by Otto Vinter". Available: http://inet.uni2.dk/~vinter/bugtaxst.doc [Accessed: 11 October 2007].

[14] Beizer, B. (1990) *Software Testing Techniques*, Van Nostrand Reinhold New York, 2nd Edition, pp. 15-34.

[15] Vinter, O. and Lauesen, S. (2000), "Analysing Requirements Bugs. Take a closer look at your bugs – you might improve your products", STQE Magazine Volume 2 Issue 6, Nov/Dec 2000. Bug Report Department. Available: http://itu.academia.edu/SorenLauesen/Papers/746571/Analyzing_Requirements_Bugs [Accessed: 23 November 2007].

[16] Aurum, A. and Wohlin, C. (2005) *Engineering and managing software requirements*, Springer, Berlin, pp. 140.

[17] Jaques, D. and Salmon, G. (2007) *Learning in groups: a handbook for face-to-face and online environments*, Abingdon, N.Y. : Routledge, 2007, 4th Edition, pp. 69-70.

[18] Lehman, M. M., The Role and Impact of Assumptions in Software Development, Maintenance and Evolution, Proceedings of the 2005, IEEE International Workshop on Software Evolvability, Budapest, Hungary, 2005.

[19] Lions, J. L. (1996), "The Ariane 5 Failure Report". Available: http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html [Accessed: 1 May 2008].

[20] Pour, M. K. (2006) *Cases on Information Technology: Lesson Learned*, Idea Group Publishing, USA, Chap XIV, pp. 231-250.

[21] Tirumala, A., Crenshaw, T. and Sha, L., "Prevention of failures arising from assumptions on software components in real-time systems", ACM SIGBED Review, 2005.

[22] Roeller, R., Lago, P. and Vliet, H. V., "Recovering architectural assumptions", The Journal of Systems and Software 79, April 2006, pp. 552–573.

[23] Deek, F. P., McHugh, J. A. M. and Eljabiri, O. M. (2005) *Strategic Software Engineering: An Interdisciplinary Approach*, Auerbach Publications, Boston, MA, USA, pp 68-69, 120-121 and 162.

[24] Sommerville, I. (2004) *Software Engineering*, Pearson/Addison Wesley, 7th Edition.

[25] Johnson, L., Greenspan, S., Lee, J., Fischer, G. and Potts, C., Recording requirements assumptions and rationale, Requirements Engineering, 1993, Proceedings of IEEE International Symposium, San Diego, CA, USA, 1993.

[26] Fickas, S. and Feather, M. S., Requirements Monitoring in Dynamic Environments in Proc. of the Second IEEE International Symposium on Requirements Engineering (York, England, U.K., March 1995), IEEE Computer Society Press, pp. 140-147.

[27] Cohen, D., Feather, M. S., Narayanaswamy, K. and Fickas, S. S., Automatic monitoring of software requirements. Proceedings of the 1997 International Conference on Software Engineering, ICSE 97.

[28] Lehman, M. M., Software's Future: Managing Evolution. Software, IEEE 15(1), pp. 40– 44, 1998.

[29] Lehman, M. M. and Ramil, J. F., Software evolution: background, theory, practice", Inf. Process. Lett., vol 88, no. 1-2, pp. 33-44, 2003.

[30] Lehman, M. M. & Ramil, J. F., “Software Evolution in the Age of Component Based Software Engineering,”, IEEE Proceedings Software, 2000, Vol. 147, No. 6, pp. 249–255, December 2000.

[31] Seacord, R. “Assumption Management.” news@sei interactive 6, 1, First Quarter 2003. Available: http://www.sei.cmu.edu/library/abstracts/news-at-sei/cotsspot1q03.cfm [Accessed: 13 March 2007].

[32] Parnas, D. L., “Software Aging” in Proceedings of the 16th International Conference on Software Engineering, Sorrento Italy, IEEE Press, 279- 287, May 16-21/94, 1994.

[33] Duboc, L., Letier, E., Rosenblum, D. S. and Wicks, T., Case Study in Eliciting Scalability Requirements, 16th IEEE International Requirements Engineering Conference, 2008.

[34] Lewis, G., Mahatham, T. and Wrage, L., Technical Note: Assumptions Management in Software Development, Carnegie Mellon University, Aug 2004.

[35] Han, J., “Experience with Designing a Requirements and Architecture Management Tool”. Proceedings of International Conference on Software Methods and Tools, Wollongong, Australia, November 2000, pages 179-188. IEEE Computer Society Press, 2000.

[36] Han, J., “TRAM: A Tool for Requirements and Architecture Management”, Proceedings of the 24th Australasian Computer Science Conference, Gold Coast, Australia, pp. 60-68, IEEE Computer Society Press, 2001.

[37] Marincic, J., Mader A. and Wieringa R., Classifying Assumptions during Requirements Verification of Embedded Systems, Proceedings of the 14th international conference on Requirements Engineering: Foundation for Software Quality, Montpellier, France, 2008.

[38] Lanubile, F., Shull, F. and Basili, V. R., "Experimenting with Error Abstraction in Requirements Documents", Software Metrics Symposium, 1998.

[39] Sommerville, I. (1996) *Software Engineering*, Addison Wesley, 5th Edition.

[40] Perry, W. E. (2000) *Effective Methods for Software Testing*, Wiley, New York, 2nd Edition.

[41] Pressman, R. S. (1997) *Software Engineering: a practitioner's approach*, McGraw-Hill, New York, 4th Edition.

[42] Pfleeger, S. L. and Atlee J. M. (2006) *Software Engineering Theory and Practice*, New Jersey, Pearson Education Inc.Third Edition, 2006, Chap 8, Pages 148-152.

[43] Robertson, S. and Robertson, J. (1999) *Mastering the Requirements Process*, Addison Wesley, Harlow.

[44] Hull, E., Jackson, K. and Dick, J. (2002) *Requirements Engineering*, Springer, New York.

[45] Dewar J. A. (2002) *Assumption-Based Planning A Tool for Reducing Avoidable Surprises*, RAND, Cambridge University Press.

[46] Garlan, D., Allen, R., and Ockerbloom, J., "Architectural Mismatch: or Why It's Hard to Build Systems Out of Existing Parts", Proceedings of the 17th International Conference on Software Engineering, Seattle, WA, 1995.

[47] Ruhai, L. C., Bradbury, J. S. and Dingel, J., Discovering Architectural Mismatch in Distributed Event-based Systems using Software Model Checking, Technical Report 2006-524, 2006.

[48] Uchitel, S. and Yankelevich, D., Enhancing Architectural Mismatch Detection with Assumptions, 7th IEEE International Conference and Workshop on the Engineering of Computer Based Systems, 2000.

[49] Sansone, C., Morf, C. C. and Panter, A. T. (2004) *The Methodological Assumptions of Social Psychology: The Mutual Dependence of Substantive Theory and Method Choice* in The SAGE Handbook of Methods in Social Psychology, SAGE Publications, Inc., London, pp. 19-22.

[50] Leahy, R. L. (2003) *Cognitive Therapy Techniques: A Practitioner's Guide*, The Guilford Press.

[51] IEEE Std 1233-1998, IEEE IEEE Guide for Developing System Requirements Specifications. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=741940&isnumber=16016 [Accessed: 10 May 2007].

[52] IEEE Std 1540-2001, IEEE Standard for Software Life Cycle Processes-Risk Management. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?tp=&isnumber=19740&arnumber=914365&punumber=7300 [Accessed: 8 May 2007].

[53] Durrenberger, M. R. (2005), Project Management Innovations: 'Assumptions on Projects: Don't take them for granted, or, the problem of not knowing'. Available: http://dewi.lunarservers.com/~ciri03/pminpdsig/letters/NPDSIGSeptember2005.pdf [Accessed: 2 May 2007].

[54] A Guide to the Project Management Body of Knowledge (PMBOK® Guide) 2000 Edition, pp. 138-139.

[55] A Guide to the Project Management Body of Knowledge (PMBOK® Guide) Third Edition, 2004.

[56] Shull, F., Rus, I. and Basili V. R., How Perspective-Based Reading Can Improve Requirements Inspections, IEEE 2000.

[57] OPEN Process: Classifications of Requirements by OPEN Process Framework Repository Organization. Available: http://www.opfro.org/index.html?Components/WorkProducts/RequirementsSet/Requirements/Requirements.html~Contents [Accessed: 4 May 2007].

[58] Requirements Categories and Subcategories (2007): Requirements Categories and Subcategories by the Requirements Solutions Group. Available: http://www.requirementssolutions.com/RequirementsTaxonomy.pdf [Accessed: 2 May 2007].

[59] IMS: The Requirements Definition Document for Invisible Meeting Scheduler. Available: http://www.cc.gatech.edu/classes/AY2001/cs6300_fall/Handouts/reqtsExample.htm [Accessed: 11 October 2007].

[60] IEEE Std 828-1998: IEEE Standard for Software Configuration Management Plans –Description. Available: http://standards.ieee.org/reading/ieee/std_public/description/se/828-1998_desc.html [Accessed: 8 May 2007]

[61] Law W.K. and Perez K. (2006) *Cross-Cultural Implementation of Information System* in Cases on Information Technology-Lesson Learned, Idea Group Publishing, USA.

[62] Carver, J., Seaman, C. and Jeffery R., Using Qualitative Methods in Software Engineering, International Advanced School of Empirical Software Engineering", (IASESE04), August 18, 2004 -Los Angeles, CA, 2004.

[63] Shull, F., Singer, J. and Sjoberg, D. I. K. (2007) *Guide to Advanced Empirical Software Engineering*, Springer, pp. 285-311.

[64] Lazaro, M. and Marcos, E., "An Approach to the Integration of Qualitative and Quantitative Research Methods in Software Engineering Research", Philosophical Foundations on Information Systems Engineering 2006.

[65] Kaplan, B. and Duchon, D., Combining Qualitative and Quantitative Methods in Information Systems Research: A Case Study, Management Information Systems Quarterly, Vol. 12, No. 4, pp. 571-586, 1998.

[66] Argyris, C., Putnam, R., and Mclain (1985). *Action Science: Concepts, Methods, and Skills for Research and Intervention*. Jossey-Bass.

[64] Juristo, N. and Moreno, A. M. (2001) *Basics of Software Engineering Experimentation*, Kluwer Academic Publishers, Chap 2, pp. 22-27.

[68] Burns, A. and Wellings, A. J. (2001) *A case study in Ada' in Real-Time System and Programming languages*, Addison Wesley, pp. 653-657.

[69] Burns, A. and Wellings, A. J., HRT-HOOD: A structured design method for hard real-time systems, Real-Time Systems, SpringerLink, 1994.

[70] Kramer, J., Magee, J., Sloman, M. S. and Lister, A. M., CONIC: An Integrated Approach to Distributed Computer Control Systems, IEEE Proceedings (Part E) 180(1), pp. 1-10, 1983.

[71] Sloman, M. and Kramer, J. (1987) *Distributed Systems and Computer Networks*, Prentice-Hall.

[72] Shrivastava, S. K., Mancini, L. and Randell, B., On The Duality of Fault Tolerant Structures, pp. 19 - 37 in Lecture Notes in Computer Science, Springer-Verlag, 1987.

[73] Burns, A. and Lister, A. M., An Architectural Framework for Timely and Reliable Distributed Information Systems (TARDIS): Description and Case Study, YCS.140, Department of Computer Science, University of York, 1990.

[74] Leahy R. L., *Cognitive Therapy Techniques A Practioner's Guide*, The Guilford Press, 2003, pp. 11-12.

[75] Leffingwell, D. and Widrig, D., Managing Software Requirements: A Unified Approach, Addison-Wesley, pp289-292, 1999.

[76] Port, D. and Klappholz, D., Performing Empirical Software Engineering Research in the Classroom, Proceedings of the 17th Conference on Software Engineering Education and Training (CSEET'04), 2004.

[77] Sjoberg, D. I. K., Hannay, J. E., Hansen, O., Kampenes, V. B., Karahasanovic, A., Liborg, N. K. and Rekdal, A. C., A Survey of Controlled Experiments in Software Engineering, IEEE Transactions On Software Engineering, Vol. 31, No. 9, September 2005.

[78] Hsia, P. and Asur, S., Scenario-Based Modelling", Rapid System Prototyping, 1991. Shortening the Path from Specification to Prototype, Second International Workshop, USA, 1991.

[79] Hooper, J. W. and Hsia, P., Scenario-based Prototyping for Requirements Identification, *SIGSOFT Softw. Eng. Notes* 7, 5 (Dec. 1982), 88-93, 1982.

[80] Mills H. D., On the statistical validation of computer programs, IBM Federal System Division., 1972.

[81] IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology,

[82] Halling, M., Biffl, S. and Grunbacher, P., An economic approach for improving requirements negotiation models with inspection, 13 May 2003, Springer-Verlag London Limited 2003.

[83] Armour, P. G., The Laws of Software Process, Communications of the ACM January 2001/Vol. 44, No. 1, 2001.

[84] Knight, J. C. and Ammann, P. E., An experimental evaluation of simple methods for seeding program errors, International Conference on Software Engineering, Proceedings of the 8th international conference on Software engineering, 1985.

[85] Ryan, K. The Role of Natural Language in Requirements Engineering, 0-8186-3120-1/92, IEEE, 1992.

[86] Kof., L., An Application of Natural Language Processing to Requirements Engineering – A Steam Boiler Case Study, 2004. Contribution to SEFM 2004.

[87] Kof, L., Natural Language Processing For Requirements Engineering: Applicability to Large Requirements Documents, Available: http://www4.informatik.tu-muenchen.de/publ/papers/Scalability_WITSE04.pdf [Accessed: 20 December 2007].

[88] Shen, W. and Liu, S., Formalization, Testing and Execution of a Use Case Diagram. Proceedings of the 5th International Conference on Formal Engineering Methods, ICFEM 2003, Springer Berlin / Heidelberg, pp. 68-85.

[89] Spiegel, M. R., Theory and Problems of Statistics in SI Units, McGraw-Hill Publishing Co. Ltd., SI Edition, pp. 217-268, 1961

[90] Exponential Distribution (2008): Exploratory Data Analysis, Engineering Statistic Handbook. Available: http://www.itl.nist.gov/div898/handbook/eda/eda.htm [Accessed: 23 September 2008].

[91] Bulandran, S. “How Vulnerable are Specifications to Assumptions”, Progress on Software Requirements Engineering, 1st August, 2008, Perth, Western Australia

[92] Wallace, D. R., Ippolito., L. M. and Cuthill, B. B., “Reference Information for the Software Verification and Validation Process”, National Institute of Standards and Technology Special Publication 500-234, 1996.

[93] ANSI/IEEE. IEEE Guide to Software Requirements Specifications. Standard Std

830-1984, 1984.

[94] Basili, V. and Weiss, D., Evaluation of a software requirements document by analysis of change data. In Proceedings of the 5th International Conference on Software Engineering.