# Using Satisfaction Arguments and Rich Traceability in Requirements Prioritisation

Praveen Kumar Motupally

A Dissertation submitted to Auckland University of Technology in partial fulfilment of the requirements for the degree of Master of Computer and Information Sciences (MCIS)

2008

School of Computing and Mathematical Sciences

Primary Supervisor: Dr. Andy Connor

## Abstract

*Requirement Engineering* (RE) is a distinct subset activity of *Systems Engineering. Eliciting* and *Specifying requirements* are the sub processes of RE. *Eliciting* and *Specifying* correct requirements, that meet the customer's needs contributes to the project's *Quality* and *Success*. However determining the "*Candidate Requirements*" is challenging for a number of reasons. *Requirement Prioritisation* helps to cope with this problem.

A number of *Requirement Prioritisation* methods exists. This dissertation aims to investigate a better prioritisation technique by subjectively assessing the "effort" between prioritising requirements with the *Analytical Hierarchy Process* (AHP) and prioritising *"Satisfaction Arguments"* (SA) with AHP and subjectively assessing the "effort" again.

The results of the experiment show a similar set of priorities produced by both attempts, however, the perceived effort of prioritising SAs is less compared with prioritising requirements with AHP due to "*Propagation of Priorities*". The results of the experiment show that "*Propagation of Priorities*" is possible with both the approaches, however "*Propagation of Priorities*" was found to be *bi-directional* when prioritising SA with AHP and *unidirectional* when prioritising requirements with AHP.

## Table of Contents

## Attestation of Authorship

"I Praveen Kumar Motupally hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person (except where explicitly defined in the acknowledgements), nor material which to a substantial extent has been submitted for the award of any other degree or diploma of a university or other institution of higher learning."

Signature:

Date:

Praveen Motupally
Auckland, New Zealand

## Acknowledgments

My gratitude goes to my Professor, Dr Andy Connor for giving me the opportunity to work on a very unique idea, he encouraged me to discover a unique requirement prioritization technique, his help during my dissertation project and advice were immeasurable. Dr Andy Connor is a visionary, a man of technical excellence, someone committed to delivering excellence, it was truly my privilege to work with him.

I take this opportunity to thank AUT staff, for their help and support over the last three years, they play major role in my academic achievements.

I am thankful to my family for their support and encouragement; I thank my wife Ana, my son Karun and my sweet daughter Gautami for bearing with me during the last few months, without their support and understanding it would not have been possible for me to complete this research work.

**Praveen Motupally**
Auckland, New Zealand

# 1 Introduction

Software Requirements Engineering (RE) normally comes to fore in the early phases of software engineering (SE). Though we have made significant progress in the area of software development, challenges experienced in the past 20 years persist. A large percentage of this can be attributed to a poor RE practice. Using appropriate RE techniques can make software development projects more successful (Jiang, 2005).

"*The hardest single part of building a software system is deciding precisely what to build. Therefore, the most important function that the software builder performs for the client is the iterative extraction and refinement of the product requirements*" (Brooks, 1987).

The following are some perceived scenarios from the industry which show how requirement engineers, software developers and customers perceive system requirements.

Inefficient communication between team members, and the stakeholders leads to flaws in the product, increased cost and delay in delivery (Brooks, 1987). Conflicting requirements may arise as a result of multiple expectations by the stakeholders, or due to limited available resources (J. Karlsson, Olsson, S., & Ryan, K. , 1997). User requirements constantly change, therefore they "*evolve*" as a result of a greater understanding of the system as the project progresses (Kaindl, 2002) and unrealistic timelines for the delivery of the system are just a few notable factors which can lead to a great discrepancy between customer requirements and the final delivered system. RE is therefore an essential practice which ensures that the delivered system meets the client's expectations.

Prioritising requirements a key area of RE, plays a crucial role in the decision making of system development, in time and resource constraints and in meeting user requirements. Prioritising requirements is complex and challenging for a number of reasons, for large scale requirements, the complexity is multiplied (P. Berander, & Andrews, A., 2005).

The purpose of this research is to investigate whether a more efficient prioritisation of requirements can be achieved by using the concept of "*Satisfaction Arguments"* (SA) (Attwood, 2004; Dick, 2000; Hull, 2005) in "*Requirement Prioritisation*" as a means of implicitly grouping together requirements, with a relatively low number of factors in pair-wise comparisons, and then prioritising these using techniques such as the *Analytical Hierarchical Process* (AHP) to ensure a fine-grained prioritisation.

Chapter 2 outlines the background of this research, a literature review includes an overview of the software requirements process, the key phases and activities involved in RE, current major research issues in software requirement engineering (SRE), a discussion of questions such as; What is prioritisation and why is it important? What prioritisation methods are used and what are their characteristics. A comparison of these methods, challenges in requirement prioritisation and prioritising large requirements sets, and some potential solutions are also reviewed. The concept of "*Rich Traceability and Satisfaction Arguments*" that allows system level requirements to be automatically generated and given a priority by applying requirement prioritisation methods is introduced. This chapter concludes by summarising SRE, Requirement Prioritisation and how people try to resolve issues in dealing with large requirements sets. Chapter 3 presents the research approach, solution construction and experimental design and the results of the experiment as carried out on a set of user and system requirements are given in chapter 4. Finally, section 5 concludes the research with concise discussion.

# 2 Literature Review

## 2.1 Software Requirements Engineering

What are *Systems Engineering*, *Requirement Engineering* and *Software Engineering* and how are they related? *Systems Engineering* defines systems requirements at an abstract level, defining overall architecture and then integrating different parts of the system into a finished product, the important activities of this process being hardware development, policy and process design and system deployment. This process is less concerned with engineering the system components, whereas *Software Engineering* deals with this aspect in detail (Holt, 2001; Sommerville, 2007; R. Stevens, Brook, P., Jackson, K., & Arnold, S., 1998).

The process of identifying, modelling, communicating and documenting stakeholder's requirements and constraints is called "*Requirement engineering*" (RE) (Sommerville, 2007). RE is a specific subset activity of *Systems Engineering* as illustrated in Figure 1, with a primary goal of creating and maintaining *Systems Requirement Specifications*. This process involves feasibility studies to assess the viability of the system being developed, *Elicitation and Analysis*, and *Specification* of requirements and *Validation* (Sommerville, 2007). RE plays a crucial role in software design and development, with its primary focus being the development of the right software for the stakeholders (A. Aurum, & Wohlin, C., 2005).

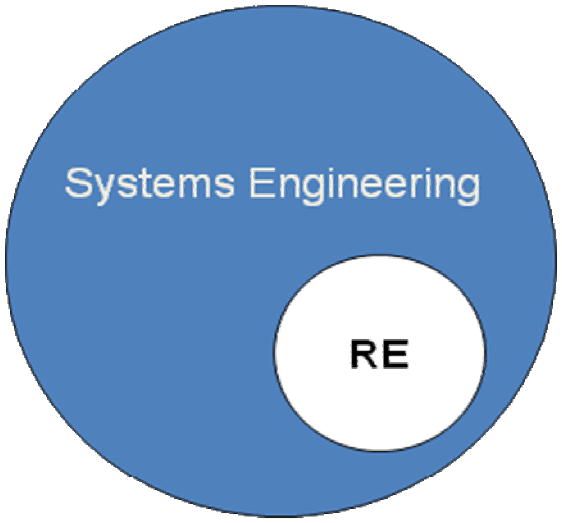


*Figure 1: Requirement Engineering in relationship to Systems Engineering*

*Software Engineering* deals with all aspects of software production for example, engineering software components to work within organisational and financial constraints, technical process and project management, and spans the entire process from the early stage of *System Requirements Specification* to maintaining the system, post production (Sommerville, 2007). As illustrate in Figure 2, there is a clear overlap between systems engineering, requirement engineering and software engineering.

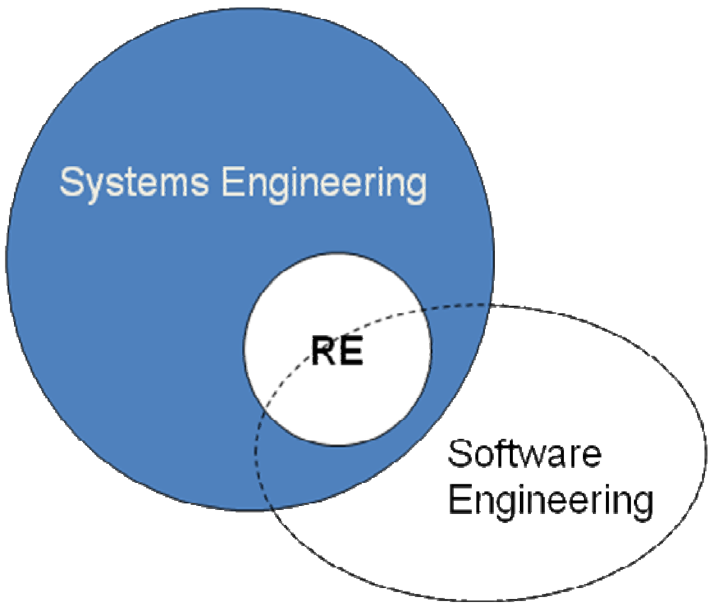


Figure 2: Software Engineering

According to Weigers (2003), the RE process consists of two sub processes:

- Requirement Development
- Requirement Management

Requirement development consists of four sub processes. *Requirements elicitation* involves identifying software requirements from a number of sources such as interviews, documents, legacy systems, workflow analysis etc. *Requirements Analysis* is the process of classifying requirement information. *Requirement specification* is the process of writing a system requirements document in a structured manageable format which can communicate to its various readers. Finally *Requirements Validation* evaluates the requirements on the basis of satisfaction of users' needs.

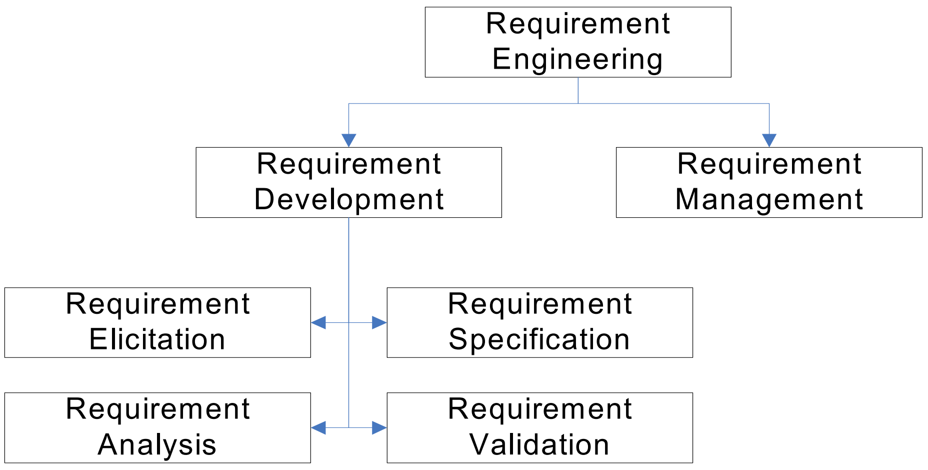


Figure 3: Requirement Engineering Processes (Sommerville, 2007)

Sommerville proposes *Feasibility Studies* prior to the four key RE activities defined by Weigers, in the RE process, i.e. *Requirements Elicitation* and *Analysis*, *Requirements Specification* and *Requirements Validation.* Figure 4 illustrates the relationship between these activities and also shows documents produced at each stage of the RE process(Sommerville, 2007).

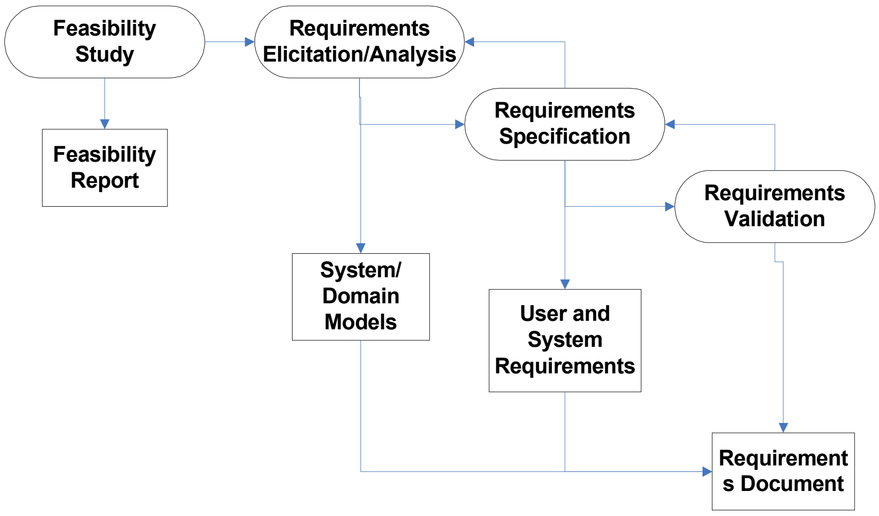


*Figure 4: Linear Requirements Development Process (Sommerville, 2007)*

An alternative model of RE process is a three stage iterative activity around a spiral as illustrated in Figure 5, the amount of time spent on each stage varies with the scale of the project. In the later phase efforts are devoted to system requirement engineering and system modelling as shown in the outer rings of the spiral (Sommerville, 2007).

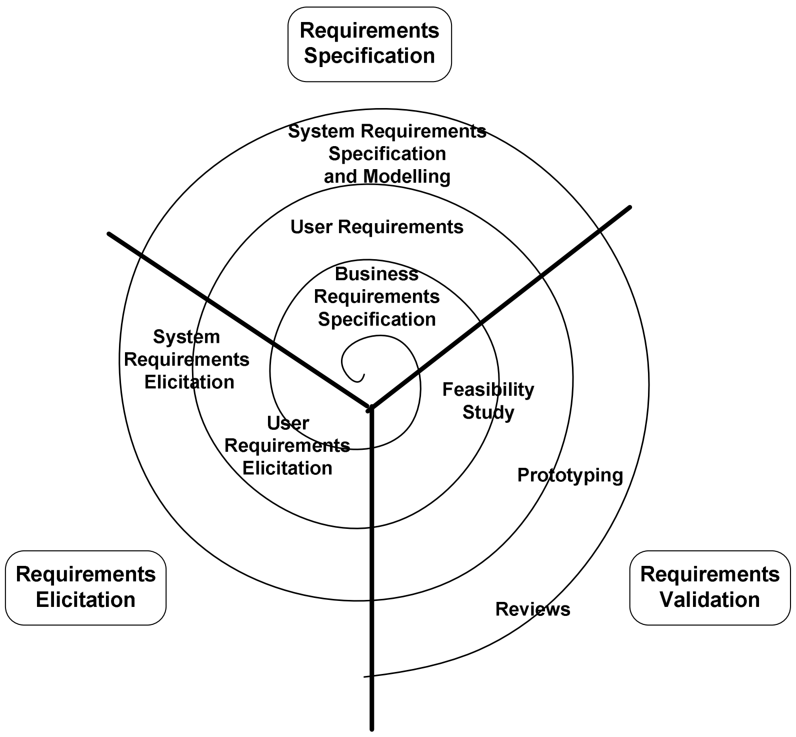

Requirements Specification
System Requirements Specification and Modelling
User Requirements
Business Requirements Specification
System Requirements Elicitation
Feasibility Study
User Requirements Elicitation
Prototyping
Requirements Elicitation
Reviews
Requirements Validation

*Figure 5: Spiral Requirements Development Process (Sommerville, 2007)*

The key RE activities are briefly discussed in the following sections.

### 2.1.1 Feasibility Studies

*A* Feasibility study is recommended prior to undertaking *Requirements Elicitation,* the sources for this process are business requirements and an outline of the system. This process is an enquiry to determine if the system being developed meets the overall objective of the organisation, within the given cost/schedule constraints and possible integration with any existing systems (Sommerville, 2007).

The output of the feasibility study is a viability report that outlines whether the system would contribute to the business objectives and whether or not it is worthwhile to developing the intended system. (Sommerville, 2007).

### 2.1.2 Requirements Elicitation and Analysis

After a feasibility study the next stage of the RE process is *Requirement Elicitation*. The meaning of "elicitation" is to draw forth or bring out (something latent or potential), therefore *Requirements Elicitation* is just not about collecting or capturing requirements but is a process of discovering requirements for a software system, with the sole purpose of communicating these needs to the developers (Zowghi, 2005).

Eliciting the right requirement is a critical part of RE, a case study by Hofmann and Lehner (2001) shows that key RE practices leads to project success (Hofmann & Lehner, 2001). Many requirement errors surface after the system is implemented and are found to be extremely costly to fix. Many RE problems originate with elicitation issues yet, research conducted on requirement engineering has chiefly ignored elicitation (Christel, 1992).

Requirements need to be "elicited" as the requirements may be spread across multiple stakeholders, existing documentation and systems. *Requirements Elicitation* is an intensively communicative phase of software development for which software engineering does not provide comprehensive techniques and for this reason effective techniques from social sciences, organisational theory, group dynamics, knowledge engineering and practical science can be used (Zowghi, 2005).

*Eliciting Requirements* is a complex and difficult process for several reasons such as, problems of scope, understanding and volatility (Christel, 1992). Brooks (Brooks, 1987) distinguishes requirements elicitation problems between; inherent difficulties in what one is trying to accomplish and those created through inadequate practice (Hatley, 1987; Sommerville, 2007; Stuart, 1997).

*Requirements Elicitation* activities comprise; communication, prioritization, negotiation, and collaboration with the relevant stakeholders. This process involves understanding the application domain, identifying requirement sources, analysing stakeholders, selecting appropriate techniques and finally eliciting requirements (Zowghi, 2005). The three interleaved processes identified by Christel (1992) are elicitation, specification and validation. Sommerville (2007) defines four aspects of the requirements elicitation and analysis phase of the overall requirement engineering process. These are; requirements discovery, classification and organisation, prioritisation and negotiation and finally documentation, as illustrated in Figure 6.

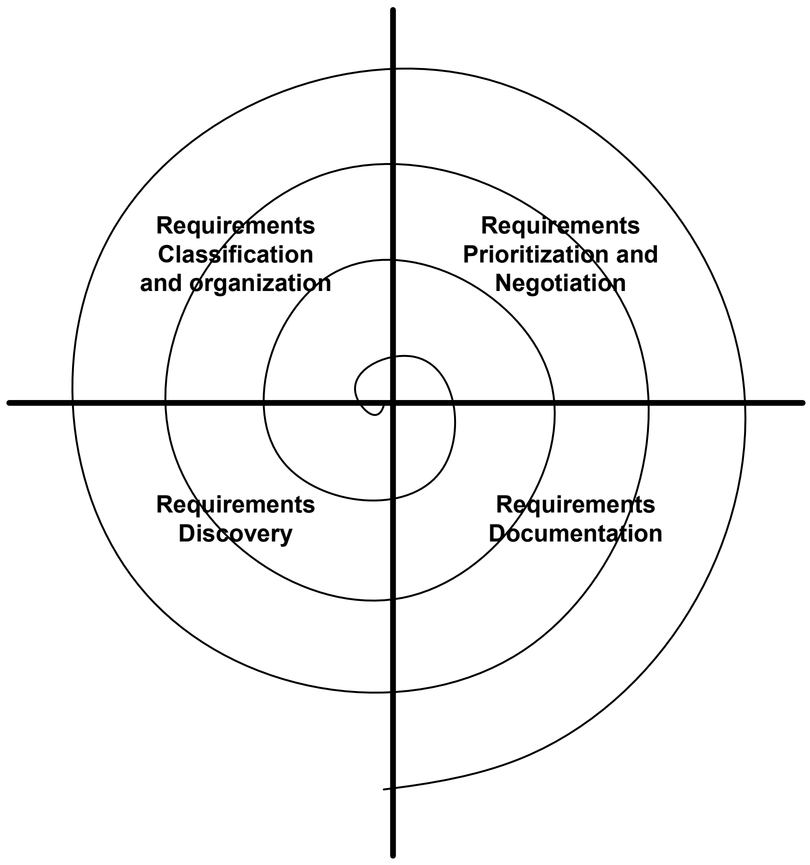

Requirements Classification and organization
Requirements Prioritization and Negotiation
Requirements Discovery
Requirements Documentation

*Figure 6: Requirements elicitation and analysis process (Sommerville, 2007)*

During the requirement elicitation and analysis phase, requirement engineers explore the problem domain and endeavour to, understand stakeholders needs and other existing systems, for the purpose of identifying system functionality and hardware requirements (Sommerville, 2007; Zowghi, 2005).

The techniques and approaches employed during requirements elicitation often depend on a number of factors such as time and cost, the availability of resources, the safety criticality of the system, and any legal or regulatory constraints. Some examples of techniques are; interviews, ethnography, observation, brainstorming (Sommerville, 2007), though this is not an exhaustive list. Understanding the application domain in a series of steps could be an approach in addition to using interview techniques to elicit (Zowghi, 2005).

In a *Structured Analysis and Design* (SAD) methodology, a combination of techniques could be used, for example data flow diagrams (DFD) to detail the function decomposition and Entity Relationship Diagrams (ERD) to represent the system entities, other requirement methodologies used by SAD are Data Dictionaries and Event Lists (Zowghi, 2005). The agile development methodology gives little emphasis to requirement elicitation and advocates incremental development (Frauke, Armin, & Frank, 2003).

Complexity and difficulties persist in requirements elicitation due to contextual, human, economic and educational factors. The most common problems encountered during this process are discussed below (Zowghi, 2005).

No single project is similar to any other therefore replicating a successful requirements elicitation process from another project may not be possible. Communication is fundamental difficulty due to problems such as articulating requirements (Sommerville, 2007), the analyst and the stakeholder may be looking at the same problem from different perspectives, stakeholders suggesting a solution rather than the requirements (Zowghi, 2005).

The quality of the requirements may suffer for reasons such as insufficient domain knowledge, incorrect, incomplete, inconsistent requirements or requirements that are not easy to validate. Requirements evolve as the project progresses hence there is a volatility of requirements (Zowghi, 2005).

Conflict in requirements is bound to occur due to multiple stakeholders with multiple interests and the likely changes in preference (Sommerville, 2007). A lack of experience by the analyst has an impact in effective elicitation in terms of the ability to select appropriate techniques and understand the problem domain (Zowghi, 2005).

The available techniques are not very useful due to a lack of empirical research and an effective transfer of knowledge from research into practice. Practitioners tend to repeat mistakes due to a lack of experience or insufficient awareness, and organisational, financial and time constrains restrict them from implementing appropriate techniques despite the need for the project to succeed (Sommerville, 2007; Zowghi, 2005).

Some of the trends and challenges in research and practice are discussed as follows. Research has identified that the requirement elicitation phase has some unique complex characteristics, yet in practice elicitation is not considered as a distinct phase. The primary challenge for researchers is to reduce the gap between research and practice, by reducing complexity. In practice the level of experience of senior and junior practitioners differ and organisational, time and financial constraints remain (Zowghi, 2005).

Reducing the gap between research and practice emerges as a crucial factor for researchers, to increase awareness and education in industry for the selection of appropriate techniques, collaboration and reuse of knowledge and exploration of the possible applications of requirement elicitation in emerging fields of software engineering such as agent based systems, agile development methodologies, and web systems (Zowghi, 2005).

### 2.1.3 Requirements Specification

Typically a *Software Requirement Specification* (SRS) is written early in the software lifecycle and aids communication between the potential user and the developer (Pohl, 1994). A SRS is a comprehensive description of the intended purpose and environment for the software under development it contains the description of the systems function and its constraints. (Sommerville, 2007). The requirements specification documents the "what," and the design description documents the "how" (Thayer, 2008).

Writing an SRS is a technical process therefore it differs from writing a book or even a technical document such as an instruction manual or user guide. Two aspects of the SRS which need to be carefully balanced are "*Readability*" meaning enabling the reader to place statements in context and "Process ability" which ensures quality i.e. clarity, language, precision and traceability (Hull, 2005). The use of diagrams and pictures in an SRS, enhances communication.

A clear distinction has to be made between high and low level requirements descriptions in an SRS. *User Requirements* (UR) are high level, abstract requirements whereas *system requirements* (SR) are a detailed description of what the system should do. Problems arise during the RE process due to a lack of a clear distinction between UR and SR (Sommerville, 2007).

User Requirements describe system functions, so technical details such as system design characteristics and, software jargon should be avoided in documenting user requirements as customers generally do not have advanced technical understanding (Sommerville, 2007).

SR's can be classified into functional, non-functional or domain requirements. *Functional requirements* are detailed descriptions of the system function, how input and output is processed. The essential attributes of functional requirements are completeness and consistency. Completeness defines the required service and the consistency ensures that requirements are not contradictory. To achieve completeness and consistency in large scale projects is a challenge (Sommerville, 2007).

Non-functional requirements attributes are system properties such as security, response time, availability and reliability and usually apply to the system as a whole. These requirements are constraints on the system functions and specify system performance therefore they are more critical than the functional requirements, depending on the nature of the system, for example, these requirements are highly critical in an aircraft system, the system will not function correctly, if the reliability requirement is not meet (Sommerville, 2007).

Generalised non-functional requirements such as ease of use or ability to recover from failure are vague goals which leaves scope for interpretation and can lead to incorrect implementation of requirements which may in turn lead to disputes after the delivery of the system and contribute to difficulties in verifying non-functional requirements (Sommerville, 2007).

For a large scale system, a separation of functional and non-functional requirements in the requirements document is helpful to manage the requirements; however this separation in two separate documents presents visibility of relationship between functional and non-functional requirements. For small scale systems these requirements can be merged into a single document (Sommerville, 2007).

*Domain requirements* originate from the business domain rather than the stakeholders and are difficult to understand in terms of relating these requirements to other system requirements. An example of such a requirement is the automation of traffic signal controls for a train system, an understanding of how train deceleration is computed and the characteristics of the train are essential. (Sommerville, 2007).

The SRS has a multiple and diverse sets of users such as stakeholders, developers, testers and support / maintenance staff, therefore there is a challenge in documenting requirements for communication and ensuring precise requirement precision (Sommerville, 2007).

Most SRS are still being written in natural language (Chantree, 2006; Kaindl, 2002) and there seems to be a little use of the RE research findings in practice. Using simple (Sommerville, 2007) and consistent language to write SRS will make it easier to identify different kinds of requirements (Hull, 2005). Written requirements vocabulary should be simple language with simple tables and intuitive diagrams. Requirement engineers have realised the challenge of managing requirements with a word processor (Kaindl, 2002) or in database (Hull, 2005; Stuart, 1997).

### 2.1.4 Requirements Validation

To deliver a system that meets the customer's requirements, the functionality of the system is checked against the specification at every stage of the development life cycle, beginning with the requirement review, design and code review to quality assurance. Boehm distinguishes between software verification and software validation (Boehm, 1988).

- Validation
- Verification

Validation ensures that the right system is built and meets the customer's expectations and verification ensures the functional and non-functional requirements are met. The goal of this process is to ensure that the software system is "*fit for purpose*". (Sommerville, 2007).

Some of the methods employed by the V&V process for checking and analysing the system are inspection or peer reviews which are carried out at all stages of the process and may be supplemented by some automatic analysis (Sommerville, 2007).

### 2.1.5 Requirements Management

Quality of the software is dependent on quality of the development process, there are two of the many possible reasons for software project failure are; software project failure, flaws in requirement elicitation and requirements becoming out of date (British Computer Society, 2004). Therefore managing requirements is crucial for project success. The key requirement management practices are in the areas of *Requirements Elicitation*, *Specification* and *Modelling*, *Prioritization, Requirements Dependencies* and *Impact Analysis*, *Requirements Negotiation* and *Quality Assurance* (A. Aurum, & Wohlin, C. , 2005). Requirements management has a critical effect on an organisation's development costs and software quality (Sawyer, 1999).

Understanding of requirements improves progressively during the development life cycle, after the user experiences the system, new needs and priorities may arise, therefore software requirements evolve and require a process to monitor these changes. (B. H. C. Cheng, & Atlee, J. M., 2007; Sommerville, 2007). Therefore the task of requirements management is to identify requirements which may possibly change (Bush, 2003). These changing requirements may have an impact on their dependent requirements. (J. Cleland-Huang, Settimi, R., BenKhadra, O., Berezhanskaya, E., & Christina, S., 2005; J. Cleland-Huang, Zemont, G., & Lukasik, W., 2004; Hayes, 2006; Marcus, 2003; Mehrdad & Steve, 2005) Managing links between these dependent requirements is required for analysing the impact of change (Sommerville, 2007).

Tracking requirements to their source is essential to ensure that the original requirement purpose has been met therefore during the elicitation stage, quantified, measurable acceptance criteria need to be defined (British Computer Society, 2004).

In a geographically distributed (De Neve, 2001), large scale (Alspaugh, 2001) or even in small scale organisations, having a diverse set of cultures, requirements and priorities, requirement contradiction and conflicts are bound to occur, prioritising requirements helps to resolve such conflicts in requirements (A. Aurum, & Wohlin, C. , 2005; Sommerville, 2007).

Changes to the business as a result of priorities, legislation & regulation or environment will have an impact on the systems which are in use or in development stage. Therefore, requirement management is crucial to control the changing system requirements and to minimise the impact of these changes (Sommerville, 2007).

## 2.2 Requirements Prioritisation

Making a choice from a couple of options is often difficult and with multiple options this problem is multiplied. Prioritisation is the process which is designed to help cope with this problem (P. Berander, & Andrews, A., 2005). Prioritisation is an important part of RE activity and an integral part of decision making (S. Hatton, 2008).

Prioritization helps decision makers analyse requirements in order to assign priority which reflect their importance. Due to time and resource constraints, fast and reliable prioritization methods are keenly sought by practitioners (J. Karlsson, Wohlin, C., & Regnell, B., 1998).

Requirements prioritization is a fundamental activity for project success. Prioritizing requirements is a strategic process which drives development expense and delivery. This process assigns priority to requirements to establish their relative order in a collection of requirements to determine which requirements are most important. (P. Berander, & Andrews, A., 2005). Prioritisation of requirements is useful for negotiating with and helping stakeholders to resolve conflict and reach an agreement (Moisiadis, 2002)

The key challenge in prioritising requirements is to determine optimal requirements, which meet the technical constraints and preferences of stakeholders, and contribute to the overall business objectives (P. Berander, & Andrews, A., 2005; Firesmith, 2004).

In the world of commercial software development, all requirements cannot be met in a time and resource constrained environment (J. Karlsson, Wohlin, C., & Regnell, B., 1998). The purpose of requirement prioritizing is to determine the essential set of requirements (Gonzales-Baixauli, 2004; Liaskos, 2006; Moreira, 2005; Regnell, 2003). The benefits of such an exercise are creation of realistic project schedules based on the available budget and resources, improvement in customer satisfaction (Firesmith, 2004) and identification of requirement defects (J. Karlsson, Wohlin, C., & Regnell, B., 1998). Prioritization also helps to minimize rework and schedule slippage (P. Berander, & Andrews, A., 2005).

The requirements prioritization processes can be broadly classified into *Methods and Negotiation*. The Methods approach assigns quantitative values to different aspects of the requirements while negotiation focuses on resolving conflicts by brokering agreement between different stakeholders, through prioritised requirements (L. Lehtola, & Kauppinen, M., 2004; L. Lehtola, Kauppinen, M., & Kujala, S., 2004).

Some of the requirement prioritisation methods used in practice are; Analytical Hierarch Process (AHP) Hierarch AHP, priority groups, Binary Search Tree, Bubble Sort and Spanning Tree Matrix (J.

Karlsson, Wohlin, C., & Regnell, B., 1998). These method can be used to negotiate and arrive at consensus on the priority of requirements Choosing the most suitable method is often quite difficult (S. Hatton, 2008).

More discussion and analysis of requirements prioritisation is given in section 2.2.4

### *2.2.1 Aspects of Requirements Prioritisation*

An aspect is an attribute or a property of a projects requirement which is used to prioritize requirements. Often there are numerous candidate requirements in a project that cannot be met with the available time and resources, prioritisation helps to determine the most essential requirements (P. Berander, & Andrews, A., 2005).

Some examples of aspect are importance, penalty, cost, time, and risk. For example, when buying a new car, it would be easy to make a choice when a single aspect exists, for example speed; in this case, we need only evaluate which car is the fastest. If a decision has to be made on multiple aspects such as cost, safety and comfort, the choice now becomes a bit more difficult compared with the previous attempt (P. Berander, & Andrews, A., 2005). Prioritizing requirements is done based on different aspects.

#### *2.2.1.1 Importance*

The importance aspect is a multifaceted concept and could mean urgency of implementation, importance of a requirement for product architecture, strategic importance for the company, etc, therefore its essential to specify the nature of importance, when prioritising the importance aspect (P. Berander, & Andrews, A., 2005; L. Lehtola, Kauppinen, M., & Kujala, S., 2004).

#### *2.2.1.2 Penalty*

The penalty aspect is an implication that is a result of not implementing a requirement (Wiegers, 1999), this aspect is not the opposite of importance, since there might be severe impact of not implementing a low priority requirement (P. Berander, & Andrews, A., 2005).

#### *2.2.1.3 Cost*

The cost aspect is often expressed in terms of man hours, the number of hours spent developing the software, the cost is determined by considering the complexity of the requirements and the quality required (P. Berander, & Andrews, A., 2005; Wiegers, 1999).

#### *2.2.1.4 Time*

Cost is often calculated in terms of man hours which are directly related to time. The time aspect is influenced by factors such as degree of parallelism in development, training needs, need to develop support infrastructure, complete industry standards, etc (P. Berander, & Andrews, A., 2005; Wiegers, 1999) (A. Aurum, & Wohlin, C. , 2005)

#### *2.2.1.5 Risk*

Every project has a degree of risk involved. Risk management is a process used for planning to manage those risks which may cause difficulties in development. Some of the risk factors are performance risks, process risks, schedule risks etc., calculating the risk per requirement enables engineers to forecast the possible project level risk (P. Berander, & Andrews, A., 2005; Wiegers, 1999).

#### 2.2.1.6 Volatility

Requirements tend to change as the project progresses for reason such as the market changes, business requirements changes, legislative changes or users change therefore requirements are volatile. In some cases volatility of requirements is handled as part of the risk aspect. The impact of this aspect is an increase in project cost and timeframe of the project (P. Berander, & Andrews, A., 2005; Ruhe, 2003).

#### 2.2.1.7 Other Aspects

The above list though fundamental is not considered an exhaustive list. Some examples of other aspects are; financial benefit, strategic benefit, competitors, competence/resources, release theme, ability to sell, etc. Requirements aspects help in decision-making, multiple aspects are usually being prioritised and lack of such guidelines would make proceeding difficult (P. Berander, & Andrews, A., 2005; L. Lehtola, Kauppinen, M., & Kujala, S., 2004).

### 2.2.2 Requirements Prioritisation Methods

A number of requirement prioritisation methods exist, such as; analytic hierarchy process (AHP) (Vargas, 1990), hierarchy AHP, spanning tree matrix, bubble sort, binary search tree, priority groups (J. Karlsson, Wohlin, C., & Regnell, B., 1998), MoSCow, simple ranking (S. Hatton, 2008), and planning game for extreme programming (Lena et al., 2007). Some of the decision making frameworks are; EVLOVE (Greer, 2004), cost-value approach (J. Karlsson, & Ryan, K., 1997) and Quantitative Win-Win (Gunther, 2002).

Requirement prioritisation methods can be grouped into two categories, methods using an ordinal or ratio measurement scale, these methods can be used as stand-alone utilities or use cost-value frameworks within these methods (J. Karlsson, Wohlin, C., & Regnell, B., 1998). The cost-value approach takes into account cost and the customers value and gives the best return on investment (ROI) in terms of customer satisfaction (L. Karlsson, Höst, M., & Regnell, B., 2006). Ratio scale are considered to be a richer scale over ordinal scale, however richer scale are time consuming and slow over the ordinal scale (L. Karlsson, Höst, M., & Regnell, B., 2006).

Some of the most common measurement scales are; nominal, ordinal, interval and ratio scale (S. S. Stevens, 1946). The nominal scale uses numerals as labels, or a combination of words and numbers, an example of this scale is the numbering of soccer players for identification. The ordinal scale uses a range of numbers in rank-order, for example, assigning student grades between 1-5, where 5 is best. The difference between the 5 and 4 is not necessarily the difference between 4 and 3 meaning the ordinal scale is not invariant when it comes to equality of differences however the interval scale is, for example, the difference between 20°C and 21°C is the same as 29°C and 30°C. A ratio scale is richer, resulting in a higher level of information than the other scales since it satisfies equality of intervals and equality of ratio, for example the distance between two cities is 10 miles, this distance is twice the distance of 5 miles, but we cannot claim that 20°C is twice as warm as 10°C (J. Karlsson, Olsson, S., & Ryan, K. , 1997).

Business aspects, customer satisfaction or technical aspects are some of the factors which influence the definition of priority criteria. A case-based ranking approach is inspired by the case-based reasoning approach. In a case-based ranking approach, elicitation of priority and requirement analysis is done in parallel, the prioritisation criteria is acquired by examples rather than explicitly encoded, meaning solutions are based on examples. AHP can be considered to use a case-based ranking approach; however it would be impractical for relative comparisons when the number of requirements increases (Avesani, 2005).

Value-oriented prioritization (VOP) is a quantitative method and takes into account the impact of the specific business values of an organisation. This method eliminates arguments and discussions on individual requirements by emphasising the core business values. VOP provides a higher granularity to ordering the requirements (Azar, 2007).

In an investigation of requirement practices in six companies, information was seldom provided as to why the requirement was relevant. Reasons for this are often not documented for fear of possible rejection. All the companies did perform prioritisation, however they did not use any standardised methods e.g. AHP / one hundred dollar($100)method etc. Some kind of attributes were used to indicate priority and in some cases the person with authority took the decision (P. Berander, & Andrews, A., 2005) In a field study of 10 organisations requirement prioritisation was undertaken (M. Lubars, Potts, C., & Richter, C., 1993).

A survey of software development practices in New Zealand showed that development organisations do use tools and spend time on feasibility, design and testing to provide a solution to the client's needs. However there was no explicit evidence of prioritization of requirements in practice in this case study (Groves, 2000). In determining the software requirement practice the size of the development project and the software development team need to be considered (Phillips, 2005). Some of the most important requirement prioritisation methods are discussed in detail below.

#### 2.2.2.1 *Analytical Hierarchy Process (AHP)*

AHP was developed by Thomas Saaty in 1980. AHP is a structured approach for dealing with complex problems. AHP helps to determine solutions rather than prescribing corrective decisions. (L. Karlsson, Höst, M., & Regnell, B., 2006; Saaty, 1980). AHP is considered to be a powerful and flexible mathematical decision making tool as it provides structure to the decision making process (Forman, 2008; Hatton, 2008).

AHP is powerful in dealing with competing criteria, when decisions need to be weighed up against multiple criteria and has found application in decision theory. This tool is useful when there are conflicting requirements (Vargas, 1990). A wide range of problems from simple to complex and capital intensive decisions, an important feature of AHP is its application in measurement of vague criteria along with the real ones through Ratio scales (Vargas, 1990).

AHP is highly sophisticated and suitable for complex and large scale projects. AHP is the only method which organises the requirements into hierarchies and then uses a pair-wise comparison, which is a promising approach, since it's based on a ratio scale, is fault tolerant and includes consistency checking (L. Karlsson, Höst, M., & Regnell, B., 2006; Vargas, 1990).

AHP and $ 100 methods use a ratio measurement scale and both these techniques scale well with small sets of requirements. Unlike other techniques, AHP can be used iteratively. AHP transforms the ordinal scale information into a ratio scale through a chain of mathematical operations. A numerical weight is given for each requirement in the hierarchy. This ability distinguishes AHP from the other prioritisation methods.

AHP uses pair-wise comparison to set priority against the objectives or alternatives. The requirements are ordered using an ordinal scale (see Table 1) to show the relative importance to one another to signify the importance of one requirement over the other. (S. Hatton, 2008).

A ratio scale measurement is richer than an ordinal scale, as it provides more information about the priority due to relative comparisons. Thus there is a cost of using simple techniques which provide ordinal ranks and complex techniques providing relative importance of requirements. In a multi-criteria decision-making problem, AHP does a pair-wise comparison within a hierarchy to determine the "Relative Importance" of the requirement. The $ 100 technique (P. Berander, & Andrews, A., 2005) uses a ratio scale measurement and provides a "Relative Difference" between the different requirements (Vargas, 1990).

AHP is comparatively difficult and time consuming when compared with other methods; however the results of this technique are more reliable and accurate than other techniques. AHP offers a precise analysis of customer requirements but the cost of doing pair-wise comparisons is higher (S. Hatton, 2007b). Automated tool support is recommended for AHP to reduce the clerical overhead involved in application of this approach.

Techniques using ratio scales of measurement cannot be compared with techniques using ordinal scales since the measurement scale differs. In one study it was concluded that AHP was superior to numeral assignment in regards to time consumption, while studies by Karlsson et al. and Ahl (L. Karlsson, Berander, P., Regnell, B., & Wohlin, C., 2004) showed that Planning Game (which basically is an extended way of doing numerical assignment where ranking is also introduced) was superior to AHP (P. Berander, Khan, K.A., & Lehtola , L., 2006). AHP & $ 100 techniques yield more accurate results over other techniques.

However this technique is best with a small set of requirements, when the number of requirements increase pair-wise comparisons increase thereby increasing the load on this technique, making the calculation complex (S. Hatton, 2008). Pair wise comparisons are time consuming to perform for large sets of requirements (L. Karlsson, Höst, M., & Regnell, B., 2006). Techniques such as "Local Stopping Rule" (LSR) & "Global Stopping Rule" (GSR) (J. Karlsson, Olsson, S., & Ryan, K. , 1997) are used to reduce the number of comparisons, and a cost-value framework enhances the process, thus making the effort efficient. However, AHP is still difficult and time consuming, but has been found by some authors to be the best technique (J. Karlsson, Wohlin, C., & Regnell, B., 1998).

At a high level, AHP has two phases, organising requirements into a hierarchy and then evaluating the priorities. Domain knowledge is essential in designing the hierarchy, and the results of this exercise may differ when carried out by two different people due to user preferences.

AHP is a decision making method carried out in three prioritization stages; in the preparation stage all exclusive pairs of requirements are outlined; during the execution stage all the outlined requirements are compared; during the evaluation phase, using the scale in table 1 requirements, requirements are determined in priority over others and to what extent. In the presentation stage the results of the execution are presented. (J. Karlsson, Wohlin, C., & Regnell, B., 1998).

*Table 1: Fundamental scale used for pair-wise comparison in AHP* (J. Karlsson, Wohlin, C., & Regnell, B., 1998)

| Intensity of Importance | Description |
|---|---|
| 1 | Of equal importance |
| 3 | Moderate difference in importance |
| 5 | Essential difference in importance |
| 7 | Major difference in importance |
| 9 | Extreme difference in importance |
| Reciprocal | If requirement i has one of the above numbers assigned to it when compared with requirement j, then j has the reciprocal value when compared with i. |

After the "*Intensity of Importance*" scale is assigned to each of the *n* requirements of a software project, the *n (n-1)/2* formula is used to determine the priority of the requirement. To illustrate AHP let us assume that we are developing an ATM which has four candidate requirements:

*Table 2: System Requirements*

| Requirement | Description |
|---|---|
| **R1** | ATM will allow withdrawal of cash |
| **R2** | ATM will allow balance checking |
| **R3** | ATM will accept other eftpos cards |
| **R4** | ATM will allow transfer of cash |

The requirements are then placed into an AHP matrix table as shown in Table 3 which allows us to assign "*intensity of importance*" values from Table 1. The highlighted R1's are the same requirement and the relationship between these requirements is equal, therefore we insert 1 "*Of equal importance*" therefore we insert 1 at all diagonal positions.

*Table 3: AHP Pair-wise comparison matrix*

| Requirement | R1 | R2 | R3 | R4 |
|---|---|---|---|---|
| **R1** | 1 | 5 or 0.50 | | |
| **R2** | | 1 | | |
| **R3** | | | 1 | |
| **R4** | | | | 1 |

Value 5 "*Essential difference in importance*" is inserted into R2 to indicate the relationship between R1 and R2 which means that R1 is 5 times more important than R2. For example if we replace value 5 from R2 to 0.50 which means that R1 is 5 times less important than the R2 requirement.

In Table 4, decimal value 0.50 is inserted to indicate the relative importance between R2 and R1, which means that R2 is half as importance than R1. Further averaging is done by totalling the column values.

*Table 4: Column Total*

| Requirement | R1 | R2 | R3 | R4 |
|---|---|---|---|---|
| **R1** | 1 | 5 | 5 | 8 |
| **R2** | 0.50 | 1 | 3 | 3 |
| **R3** | 0.50 | 0.30 | 1 | 7 |
| **R4** | 0.80 | 0.50 | 0.50 | 1 |
| **Totals** | **2.90** | **6.80** | **9.5** | **19** |

The total value of R1 i.e. 2.90, is then divided by the R1 value 1, for example 1/2.90 = 0.34 derives the normalized value.

*Table 5 : cell percentages*

| Requirement | R1 | R2 | R3 | R4 |
|---|---|---|---|---|
| **R1** | 1/2.90=0.34 | 5/6.80=0.73 | 5/9.5=0.52 | 8/19=0.42 |
| **R2** | 0.50/2.90=0.17 | 1/6.80=0.14 | 3/9.5=0.31 | 3/19=0.15 |
| **R3** | 0.80/2.90=0.27 | 0.30/6.80=0.04 | 1/9.5=0.10 | 7/19=0.36 |
| **R4** | 0.60/2.90=0.20 | 0.50/6.80=0.07 | 0.50/9.5=0.05 | 1/19=0.05 |
| **Totals** | **2.90** | **6.80** | **9.5** | **19** |

The cell values are transferred into Table 5. The rows are totalled and then divided in the row total column, the resulting value is the priority of the requirement.

*Table 6: AHP Pair-wise comparison matrix*

| Requirement | R1 | R2 | R3 | R4 | Row Total |
|---|---|---|---|---|---|
| **R1** | 0.34 | 0.73 | 0.52 | 0.42 | 2.01/4=0.50 |
| **R2** | 0.17 | 0.14 | 0.31 | 0.15 | 0.77/4=0.19 |
| **R3** | 0.27 | 0.04 | 0.10 | 0.36 | 0.77/4=0.19 |
| **R4** | **0.20** | **0.07** | **0.05** | **0.05** | **1.45/4=0.36** |

The final prioritized requirements are shown in Table 7:

*Table 7: Requirements Priority*

| Requirement | Description | Priority |
|---|---|---|
| R1 | ATM will allow withdrawal of cash | 50% |
| R2 | ATM will allow balance checking | 19% |
| R3 | ATM will accept other eftpos cards | 19% |
| R4 | ATM will allow transfer cash | 36% |

This method has proved to be precise, effective, and produces informative and reliable results (J. Karlsson, Wohlin, C., & Regnell, B., 1998). Despite its effectiveness, AHP has essential downsides which hinders its usage in large scale development projects since all exclusive pairs are to be compared, the required effort could be substantial, however, AHP scale well with small sets of requirements therefore is useful in small scale projects (J. Karlsson, Wohlin, C., & Regnell, B., 1998).

Pair-wise comparison makes the process fairly insensitive to judgemental errors and the resulting priorities are based on a relative ratio scale, which allows for a useful assessment of requirements (J. Karlsson, Wohlin, C., & Regnell, B., 1998).

#### 2.2.2.2 *Hundred-dollar test*

AHP first groups the hierarchies and then evaluates them whereas the $ 100 approach involves only one step i.e. it only distributes the available dollars among the requirements, yielding a "Relative difference" between the requirements. Therefore $ 100 is not as complex as AHP, but it is still powerful since it uses a ratio scale for measurement. This approach is highly suitable for individuals or groups of stakeholders (S. Hatton, 2008), unlike AHP which is built for large projects with multiple stakeholders. This technique will scale well with small sets of requirements. The results of this technique are reliable. Unlike AHP, however iterative use of the $ 100 approach in the life cycle of the project will not scale.

The $ 100 techniques is useful when there are numerous ideas and some them need to be carried to the next stage, or when there is limited time or is to be used in the early stages of a project e.g. to reach a consensus on the "mission statement" (Cowley, 1997).

This method is simple and straight forward however it does not work in an iterative process in a project because once the results are known, participants will bias their input the next time around or allocate more dollars to their favourite feature which didn't make it into the list in the first iteration. Sometimes we may have to limit the amount spent on a single feature otherwise participants may influence some of the requirements which will ultimately acquire a high priority. Further, we may allow higher limits so long as we have the opportunity to understand where the really big votes came from. They may represent high-priority needs from a limited stakeholder community (Leffingwell, 2003).

#### *2.2.2.3 Cumulative Voting*

"*Cumulative voting*" and the $ 100 method are fundamentally the same, except that this techniques has $20 to spend between the requirements. This technique is straight forward and simple, stakeholders are given imaginary units or 20 dollars to distribute between requirements. The outcome of this process is requirements are prioritized on a ratio. This technique is well suited to cases where there are few requirements, if you have 4 requirements, there are on average five points to distribute for each requirement (P. Berander, & Andrews, A., 2005)

A requirement is said to have higher priority when more units or dollars are assigned to it. However when there are more groups of requirements to prioritize, representing them on a ratio scale becomes a challenge (P. Berander, & Andrews, A., 2005). In a large scale prioritization there is a possibility of miscalculation if the allocated units or 20 dollars do not total to 20$. Limiting the allocation of units or dollars per requirement will eliminate the risk of stakeholder's tendency to heavily allocate on a requirement which might influence the result. However stakeholders may be forced to not prioritize according to their real priorities (P. Berander, & Andrews, A., 2005).

To illustrate the "Cumulative voting" technique (Ayad, 2008) let's consider the previous example in Table 2 (section 2.2.2.1) with our four candidate requirements. The requirements units or dollars are assigned based on the "*Importance*" aspect, for example, if a requirement is twice as important as another requirement then this requirement gets the appropriate priority.

*Table 8 : Cumulative Voting*

| Requirement | Units/$$ | Priority |
|---|---|---|
| R1 | 0 | 1 |
| R2 | 0 | 1 |
| R3 | 0 | 1 |
| R4 | 0 | 1 |
| **Total Units/$$** | **20** | |

Table 9 displays requirements with the assigned values, R1 has the highest units/$$ therefore has the 1st priority and R4 has the least units/$ assigned, meaning it has the least priority amongst the requirements.

*Table 9 : Allocated units / $*

| Requirement | Units/$$ | Priority |
|---|---|---|
| R1 | 8 | 1 |
| R2 | 6 | 2 |
| R3 | 4 | 3 |
| R4 | 2 | 4 |
| **Total Units/$$** | **20** | |

The total units/$$ cannot exceed more than 20, zero can be assigned to requirement to indicate that it is not important.

The final prioritized requirements are displayed in Table 10 below, its recommend to sort the requirements by priority for visibility and to ensure that requirements are not over assigned.

*Table 10 : Requirements Priority*

| Requirement | Description | Units/$$ | Priority |
|---|---|---|---|
| R1 | ATM will allow withdrawal of cash | 8 | 1 |
| R2 | ATM will allow balance checking | 6 | 2 |
| R3 | ATM will accept other eftpos cards | 4 | 3 |
| R4 | ATM will allow transfer of cash | 2 | 4 |

#### 2.2.2.4 *Numerical Assignment*

This approach is very easy to use, and less time consuming than ratio scale and ranking techniques. This approach uses an ordinal scale and groups requirements into priority groups. Therefore requirements grouped in one category appear to have equal priority and there is no information regarding the relationship between the requirements. Unlike AHP & $ 100, requirements in this approach are prioritised based on the "*perceived importance*" rather than the "*relative importance*" (P. Berander, & Andrews, A., 2005).

Restricting the number of requirements per group is advisable since the stakeholder may think that 85% of the requirements are critical. This approach can be used on less complex or non-sensitive data (S. Hatton, 2007b).

This technique assigns symbols to each requirement representing perceived importance and classifying requirements into priority groups such as mandatory, desirable or inessential, or essential, conditional or optional (IEEE, 1998; J. Karlsson, 1996).

It's possible to assign numerical values between 1-5, with the 5th ranked requirement being the most important, however this approach does not provide information about the relationship between the requirement therefore requirements in a priority group appear equally important. (L. Karlsson, Höst, M., & Regnell, B., 2006). The values 1-5 are assigned as follows:

Table 11 : Numeric Assignment Scale Range (J. Karlsson, 1996)

| Scale Range | Description |
|---|---|
| 5 | Mandatory (the customer cannot do without it). |
| 4 | Very important (the customer does not want to be without it |
| **3** | Rather important (the customer would appreciate it). |
| 2 | Not important (the customer would accept its absence). |
| 1 | Does not matter. |

We will apply the scale range 1-5 to our example requirements to illustrate the numeric assignment technique as shown in Table 12.

Table 12 : Requirements priority according to the numeral assignment technique

| Requirement | Priority |
|---|---|
| R1 | 5 |
| R2 | 5 |
| R3 | 4 |
| R4 | 3 |

Requirements R1 & R2 have higher priority over R3 & R4 and R3 has higher priority over R4.

#### 2.2.2.5 Ranking

This method is simple and easy to use, like numerical assignment, and top ten approaches, when compared to ratio scale techniques. Technique scales well with small sets of requirements (S. Hatton, 2008) therefore it can help sophisticated techniques such as AHP create hierarchies. Unlike the numerical assignment approach, this technique ranks requirements without ties. It's not possible to see the relative difference between ranked elements as in AHP. Research shows that there is difficulty in remembering more than seven elements (S. Hatton, 2007b). Mathematical calculations such as addition and multiplication on the ordinal ranked requirements do not produce meaningful results, however this ranking supports statistical operation, for example, taking the mean priority of the requirement, which might result in a tie between requirements (A. Aurum, & Wohlin, C. , 2005). This techniques yields medium level granularity in prioritising requirements when compared to AHP and $ 100 approach.

Numerical assignment, groups requirements into priority groups whereas the ranking technique uses a unique ordinal scale to rank requirements without any ties in rank. The list of ranked requirements could be achieved in a numbers of ways, for example, by using the bubble sort or binary search tree algorithms. This technique suits single stakeholder situations however there might be problems aligning several different stakeholders' ideas using this method, since this method doesn't work if a single requirement is ranked the same way twice (P. Berander, & Andrews, A., 2005; J. R. Swisher, & Jacobson, S. H. , 1999; J. R. Swisher, Jacobson, S.H., & Yücesan, E., 2003).

Table 13 : Requirements priority according to the Ranking method

| Requirement | Priority |
|---|---|
| R1 | 5 |
| R2 | 4 |
| R3 | 3 |
| R4 | 2 |

Unlike numeric assignment, this method does not allow duplicate ranking priority as shown in the Table 13.

#### 2.2.2.6 Top-Ten

This approach is extremely easy to use compared with all the above approaches. This approach uses neither the ordinal nor ratio scale of measurement. In the ranking approach the requirements are prioritised 1-n and in the numerical approach groups of requirements are created, in this approach the top-ten requirements are picked by multiple stakeholders, without assigning any order of priority, therefore this technique is useful when multiple stakeholders of equal importance exist were each stakeholder can pick their top ten requirements. However the downside of this approach is that unnecessary conflict arises when some stakeholders get support for their top priorities and others only for their third priority (P. Berander, & Andrews, A., 2005).

This approach yields neither "*Relative*" nor "*Perceived*" importance of the requirements (as opposed to AHP, $ 100, Ranking, Numerical) rather this approach tries to satisfy multiple stakeholder, therefore, there is a possibility of missing crucial requirements. This approach might be useful when used in the early stages of the project (P. Berander, & Andrews, A., 2005; S. Hatton, 2008).

*Table 14 : Requirements priority according to the Top 10*

| Requirement | Priority | Stakeholder |
|---|---|---|
| R1 | 1 | SH1 |
| R2 | 2 | SH1 |
| R3 | 3 | SH1 |
| R4 | 4 | SH2 |

Stakeholder (SH 1) has managed to get the top three requirements into the product whereas SH 2 managed to get only one requirement into the product, this might spark conflict in the organisation.

### 2.2.3 Comparison of Requirements Prioritisation Methods

Requirement prioritisation methods use ordinal or ratio scales. Ordinal scales where requirements are ordered is the least powerful, ratio scales are considered to be a higher level of measurement and are more powerful since they are able to measure the priority of one requirement over another based on a relative comparison, making an highly sophisticated evaluation and calculations possible (P. Berander, & Andrews, A., 2005; Fenton, 1998). Table 15 summarises the methods based on scale, granularity and sophistication. More sophisticated techniques are richer and more time consuming therefore they are better used when dealing with sensitive analysis for conflict resolution or to support critical decisions (P. Berander, & Andrews, A., 2005).

*Table 15 : Prioritisation techniques (P. Berander, & Andrews, A., 2005)*

| Techniques | Scale | Granularity | Sophistication |
|---|---|---|---|
| AHP | Ratio | Fine | Very Complex |
| Hundred-dollar test | Ratio | Fine | Complex |
| Ranking | Ordinal | Medium | Easy |
| Numerical Assignment | Ordinal | Coarse | Very Easy |
| Top-ten | -- | Extremely Coarse | Extremely Easy |

Combining some of the prioritization techniques (see table 15) is possible in order to make the requirement prioritization process easy and efficient. An example of this is the Planning Game in eXtreme programming (XP), where a combination of numerical assignment and ranking techniques is used, and the requirements are grouped by priority and then ranked within each group. Studies compared the extreme programming Planning Game technique with pair-wise method, and showed planning game to be superior to pair-wise comparisons (P. Berander, & Andrews, A., 2005; L. Karlsson, Höst, M., & Regnell, B., 2006).

In another approach, requirements are grouped into *"Must", "Optional"* and *"More Attention"* categories. In this approach all requirements do not require prioritisation using more sophisticated techniques, simple techniques can be applied to the *optional category* and more sophisticated techniques such as AHP, $ 100 and ranking can be applied to the "*More Attention*" group. The numerical assignment approach can be used in AHP to create the groups within the hierarchical structure and in the $ 100 approach to create the different groups within the hierarchy (P. Berander, & Andrews, A., 2005).

Using a combination of methods at different stages of the software project is likely to yield useful results, however selecting the right method for a particular stage of the project require experience. Each of the prioritization method outputs different types of information e.g. simple ranking provides order of preference while $ 100 and AHP magnitude of preference and the others, hierarchical order of preference of groups, some methods can be used iteratively while the others cannot (S. Hatton, 2007a).

A combination of methods at different stages of a software development project is illustrated with the following example. At the early stage, high level requirements do not require highly sophisticated methods such as AHP or $ 100 as there would be a limited understanding of these requirements in this stage of the project. MoSCoW is highly suitable at this stage for grouping requirements into Must have, Should have and Could have categories, and is an excellent starting point and easy to use (S. Hatton, 2008).

In the middle stage, the stakeholder and the development team have a greater understanding of the requirements, must and should have requirements can be further investigated using $ 100 or simple ranking techniques. Simple ranking is feasible on small groups of requirements within the must, should and could have requirements (S. Hatton, 2008).

At a later stage, when the requirements are fully fleshed out and the design specification is being worked out, there would be a higher degree of requirement understanding. In large and complex projects there is possibility of conflicting requirements surfacing during the design phase, which indicates that the early prioritization efforts are not sufficient. Therefore the rich and highly sophisticated approach AHP, is recommended for this stage, though it is time consuming, it is known for its effectiveness when decisions need to be weighed up against multiple criteria. It is highly recommended for resolving conflicting requirements and yields accurate and fine grained results (S. Hatton, 2008).

#### *2.2.3.1 AHP*

Propagation of priorities is only possible with AHP. This is the only method which decomposes the problem area into a hierarchy of criteria and alternatives. AHP states the objective, defines the criteria and picks the alternatives. The judgement of priority is determined by the relative ranking, similar to $ 100, this is the only method which uses both qualitative and quantitative criteria to derive priority.

Unlike other prioritisation methods, AHP can be employed iteratively. Most of the prioritisation methods employ human judgment to determine the priority of the requirements, for example numeric assignment, $ 100 etc prioritise by perceived priority rather than relative importance, while AHP offers relative importance between alternatives, and helps stakeholders to visualise the problem in a hierarchy. The simple ranking methods provides order of preference while AHP and $ 100 provide the magnitude of preference.

AHP is recommended for multiple decision makers, while the other prioritisation methods require appropriate judgement to suit the context. Though AHP yields fine granular results which are based on a relative scale, like $ 100 method, large sets of requirements make the pair-wise comparisons cumbersome, making this method very complex.

#### 2.2.3.2 Hundred-dollar test

This method is simple compared to AHP since its uses human judgement to distribute the available dollars between the requirements. While this technique does a comparative analysis for prioritisation like AHP, it does not structure the problem area into hierarchy and alternatives.

The results of the prioritisation of two stakeholders may differ due to personal preferences, therefore the results of prioritisation is based on the perceived importance rather than relative importance. $ 100 yields relative difference and fine granular results like AHP. This technique is straight forward and simple compared to AHP. Like AHP, this technique does not scale with large sets of requirements. This technique cannot be used iteratively on the same set of requirements as AHP can.

#### 2.2.3.3 Ranking

AHP and $ 100 methods use a ratio measurement scale while this method uses an ordinal measurement scale. Requirements are ranked without a tie, meaning, the relative difference is not available as in AHP or $ 100 method. This technique does not scale when multiple stakeholders exist. The operational overhead is low compared to AHP and $ 100 since this method orders requirement by assigning priority, this is a simple and easy method.

#### 2.2.3.4 Numerical Assignment

Like Ranking, this technique uses an ordinal measurement scale. Priority groups are created based on the individual perceived importance, therefore the relative importance is missing in this approach, as in the ranking technique. The priority of requirements in each group seems to be of equal importance.

#### 2.2.3.5 Top-ten

This technique picks the top-ten requirements without using an ordinal or ratio scale, unlike any of the prioritisation techniques discussed so far. This technique is suitable for multiple stakeholders with equal importance, therefore it does not offer any conflict resolution like AHP. This technique might be useful in the beginning of the project to define high level requirements, while AHP or $ 100 methods could be used in the later stage of the project.

### 2.2.4 Challenges in Requirements Prioritisation

Requirements prioritisation is a complex communication and negotiation process, the challenges in requirements prioritisation stem from informal practice and the ambiguous nature of the process since "*Requirements Prioritisation*" and "*Priority*" can, in practice, imply several different meanings (L. Lehtola, Kauppinen, M., & Kujala, S., 2004).

Though requirements prioritisation is a crucial aspect of product development, this process is often not well done for reasons such as difficulty in assigning / modifying priority, selecting an appropriate method for a particular stage of the project and communicating priorities. A possible reason for this is a lack of proven technique (Davis, 1990; L. Lehtola, Kauppinen, M., & Kujala, S., 2004).

Determining an optimal set of requirements for a release of a software system is difficult as requirements may depend on each other in complex ways, therefore prioritising the dependent requirements presents a challenge (Carlshamre, 2001)

The scope of large scale projects present challenges in comprehension of the problem domain, integrating different legacy systems, understanding the requirements of multiple stakeholders who have diverse culture, socio-economic backgrounds and may even be globally distributed (Feiler, 2006).

In a large system with multiple stakeholders distributed across the world, and requirements abstracted at different levels, gathering, resolving and prioritising such requirements is challenging (B. H. C. Cheng, & Atlee, J. M., 2007; Feiler, 2006).

Requirements prioritisation involves a great deal of invisible decision making, stakeholders may possibly avoid this process for fear of not implementing their favourite requirements. This is more of a political process than a technical one (Andriole, 1998; P. Berander, & Andrews, A., 2005; L. Lehtola, Kauppinen, M., & Kujala, S., 2004).

#### 2.2.4.1 Assigning Priority

Prioritising requirements is an important activity in requirement engineering, due to high customer expectations, a time and resource constrained environment, and pressure to produce the best return on investment and customer satisfaction(L. Karlsson, Höst, M., & Regnell, B., 2006; Port, 2008). However assigning priority is difficult for several reasons (M. Lubars, Potts, C., Richter, C., 1993) .

The qualitative and the quantitative aspects may be considered for assigning priority, in some cases assigning a quantitative value as *priority* to the requirement may mean that the qualitative aspect is missing from the process (L. Karlsson, Höst, M., & Regnell, B., 2006).

Assigning priority to requirements with multiple stakeholders and their personal expectations is challenging, since some stakeholders may not want to compromise. On the other hand, it gets even more complex when stakeholders know that low priority requirements may never be implemented (Wiegers, 2003).

Requirements are fully fleshed out as the project progresses (Hatton, 2008). Requirements change in time, that means that the priority of the requirements may change with a change in business requirements (Kaindl, 2002).

A lack of business domain knowledge or a lack of understanding of the requirements may present challenges in assigning requirement priority (M. Lubars, Potts, C., & Richter, C., 1993).

Large sets of requirements present challenges in assigning priority to requirements (P. Berander, & Andrews, A., 2005), as understanding the scope of the project, multiple stakeholders, heterogeneous cultures and decentralised organisations (Feiler, 2006) make the prioritisation process difficult.

Assigning the *real* priority to requirements may not be possible at all times, since requirement engineering is more of a political process than a technical one (Andriole, 1998; Sommerville, 2007),

There is lack of support tools for prioritising requirements. There is a clerical overhead in managing even a modest number of requirements, for a large-scale project, automated computer based support is essential (J. Karlsson, Olsson, S., & Ryan, K. , 1997)

#### *2.2.4.2 Selecting the Right Prioritisation Methods*

There are numerous methods for prioritizing requirements, choosing the appropriate method can pose challenges, one of the reason for this is a lack of research in this area, often requirement engineers learn RE practice on the job (Jiang, 2005).

The numerous prioritization methods, each produce a different degree of information about the stakeholders' preferences. The most difficult task is to pick a prioritization method which suits the stage of the project and the amount of information required. The criteria for selecting a prioritization method is based on criteria such as, the project development methodology being used, the time available, the amount of information known about the requirements, the stage the project is at and the amount of information known about priority of the requirements. Therefore choosing the most appropriate method is quite difficult (S. Hatton, 2007b).

Methods like AHP can be used iteratively during the life cycle of the projects unlike the $ 100 method. Some development methodologies requirement prioritisation is done only once, while an iterative prioritisation approach may help to realise the full benefits. However selecting a method for iterative use or selecting appropriate prioritisation methodology which would suite the stage of the project is a challenge (S. Hatton, 2008).

Selecting a particular prioritization method for a given circumstance or iterative approach is not very clear. A method for prioritization could be selected simply based on the personal liking of the developer, rather than the characteristics of the project. Very little research has been done to help in technique selection based on project attributes (Jiang, 2005).

The stage of project development determines the amount of information known by the stakeholder and the developer; meaning the requirements are better defined and understood as the development process progresses. When the delivery schedule is a major issue, faster methods for prioritization are required, and the details of the requirements are sacrificed. However, selecting faster prioritization methods to suit a given stage of the project is a challenge (S. Hatton, 2008).

There is a lack of flexibility when development methodologies have built-in prioritisation techniques, which may not be suitable for a particular stage of the project (S. Hatton, 2008), in this case, there is no opportunities to select any other prioritisation method.

In previous studies on prioritisation methods, AHP was found to be difficult and time consuming, but was found to be the best technique (S. Hatton, 2007b). When there is demand for delivery in a limited time, application of a robust and time-consuming method might prove fatal to the project, however, application of much faster methods may mean sacrificing quality. A lack of knowledge of both the simple and complex prioritisation methods will add to the challenge.

Some positive results from combining various requirement prioritisation methods for application at different stages of a project have been gained (Jiang, 2005; Mishra, 2008). However, an in-depth understanding of the various methods to be combined poses a challenge.

#### 2.2.4.3 *Prioritising Large Sets of Requirements*

Prioritising a large set of requirements requires robust methods, such as AHP, which defines the problem at a high level, since it decomposes the problem into a hierarchy of criteria and alternatives, and this is a step towards structuring the problem domain and providing visibility to the stakeholders. The pair-wise comparison presents the best alternative (E. W. L. Cheng, & Li,H., 2001; Vargas, 1990).

Structuring the requirements into hierarchies has several benefits; firstly in large scale or complex projects, sets of requirements are likely to be elaborated as a layered hierarchy. Secondly these layers of hierarchy provide an appropriate level of abstraction to the stakeholders. Finally the layered hierarchy reduces the number of comparisons in the evaluation phase (Saaty, 1980). The process of comparison produces a relative scale of measurement of the priority of each requirement.

Though AHP is robust and suitable for large scale projects, it cannot be used at every stage of the project. Combining requirement prioritisation techniques can make the process easier and more efficient. The application of AHP iteratively, in the later stages of the project, when the requirements are fully fleshed out has benefits (P. Berander, & Andrews, A., 2005; S. Hatton, 2008)

Tools for consistent checking can be highly effective for detecting errors in requirements specifications. Modelling large, complex systems from stakeholders' different viewpoints gives rise to consistency behaviour (Heitmeyer, 1996) which can be classified into: horizontal, meaning the different viewpoints must not have contradictions, and vertical, meaning, refining or creating new versions of specifications consistency with previous version is maintained. Inconsistency may be the result of logical errors in the requirements (Heimdahl & Leveson, 1996).

There are some approaches to managing the inconsistency in requirements which help in large scale requirements sets (Engels, 2001). Yet in practice lack of appropriate tools which can cope with the situation (Nentwich, 2003).

AHP eliminates inconsistent responses. According to Saaty (1980), Inconsistency refers to a lack of transitivity of preference (Saaty, 1980). Transitive consistency means, when A is better than B, and B is better than C, then A is better than C. AHP eliminates any inconsistencies in requirements and therefore provides a higher consistency (E. W. L. Cheng, & Li,H., 2001) it helps to select consistent requirements and eliminate conflicts.

AHP is the recommended method for prioritising large scale requirements at a granular level. This method combines the qualitative and quantitative approaches into a single methodology. AHP decomposes the unstructured problem into a decision hierarchy and then employs a iterative pair-wise comparison (E. W. L. Cheng, & Li,H., 2001). This method is therefore, appropriate for dealing with large sets of requirements since it groups the requirements into manageable hierarchical groups.

## 2.3 Potential Solutions

We cannot rely on a single technique to solve RE problems. Methodology for Requirement engineering Technique Selection (MRETS) is one process which helps in the selection of an appropriate combination of RE techniques. The results of combining RE techniques look promising (Jiang, 2005).

AHP clusters requirements into a hierarchy, which abstracts the problem in a layered hierarchy which helps the stakeholders and developers conceptualize a complex problem. AHP has proved to be a sound basis for prioritising requirements. The prioritised requirements are determined on a ratio scale based on their relative importance (J. Karlsson, Wohlin, C., & Regnell, B., 1998). Since AHP may not scale with large sets of requirements, using techniques to reducing the number of comparisons has been suggested (J. Karlsson, Olsson, S., & Ryan, K. , 1997). Combining different prioritization techniques at various stages of the project improves the process (S. Hatton, 2008).

### *2.3.1 Requirements Clustering*

A system can be decomposed into less complex, manageable, functional components by applying a "*divide-and-conquer*" concept which helps to decompose the system, making it easier to manage its complexity (P. Hsia, Hsu, C.T., Kung, D.C., & Holder, L.B. , 1996).

Conceptualizing requirements for a large scale project is a challenge. Thus it is important to decompose the systems into a set of modules or clusters to manage a large-scale complex application. Building modular software systems is an on going design issue (Al-Otaiby, 2005).

Requirements clustering allows for the decomposition of large scale-systems into user recognizable components, where each component is able to satisfy parts of the system independently (P. Hsia, & Yaung, A. T., 1988).

Some requirement clustering techniques are discussed in this section. Requirements scenarios are expressed in English and are clustered with those with which it has a strong functional relationship within logical groups based on the quantitative attribute of the scenario, and as having a weak relationship with requirements in other cluster groups (Al-Otaiby, 2005). Rapid prototypes are created in *Scenario-Based Prototyping*, where requirements are clustered based on the scenario (P. Hsia, & Yaung, A. T., 1988).

Some of the standard data mining clustering algorithms such as K-Means, agglomerate hierarchical clustering, bisecting, and probabilistic techniques can be used to cluster requirements, however the drawback to data mining are the highly dimensional, sparse, noisy data sets, that arise from ambiguity in requirements (Duan, 2008).

In an incremental development and delivery approach, requirement clustering is done using Entity Relation (ER) modelling, scenarios and formal specification notation Z. Firstly all the possible scenarios are prepared to cover the full system operations, then each scenario is presented as a scenario tree. Using an ER diagram, the systems data model is created. Secondly, this ER model is mapped onto a Z state schema. Finally a six-step requirement clustering algorithm is applied to further refine the clusters (P. Hsia, Hsu, C.T., Kung, D.C., & Holder, L.B. , 1996).

### 2.3.2 Proposed Clustering Technique

*Requirements traceability* in requirement engineering is about understanding how high level requirements are transformed into low level requirements. *"Requirements Traceability"* deals with relationships between high and low level requirements as shown in Figure 7, linking user requirements (UR) to system requirements (SR).

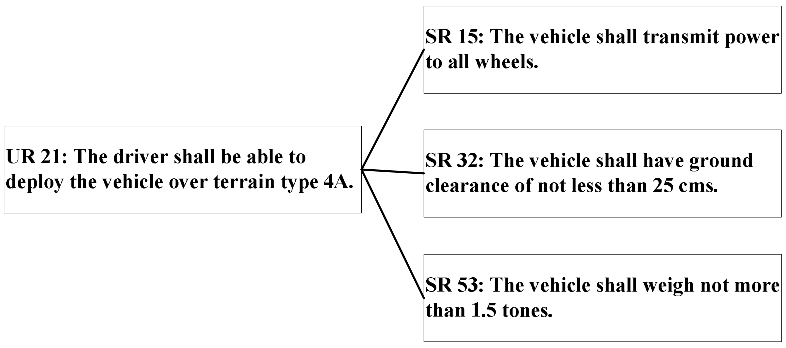

UR 21: The driver shall be able to deploy the vehicle over terrain type 4A.
SR 15: The vehicle shall transmit power to all wheels.
SR 32: The vehicle shall have ground clearance of not less than 25 cms.
SR 53: The vehicle shall weigh not more than 1.5 tones.

*Figure 7: Elementary Traceability (Hull, 2005)*

Figure 7 illustrates elementary traceability; a single UR is linked to three SRs implying that the three SRs are required to satisfy the linked UR. The purpose of this activity is to create relationships between layers of information to ensure that the stakeholders requirements are met by the system requirements (Hull, 2005; Lamsweerde, 2001).

Figure 8 illustrates traceability link relationships between lower levels and higher level requirements, tracing back to the stakeholders' requirements. The traceability links flow downwards through the different layers of requirements and across to the test information (Hull, 2005).

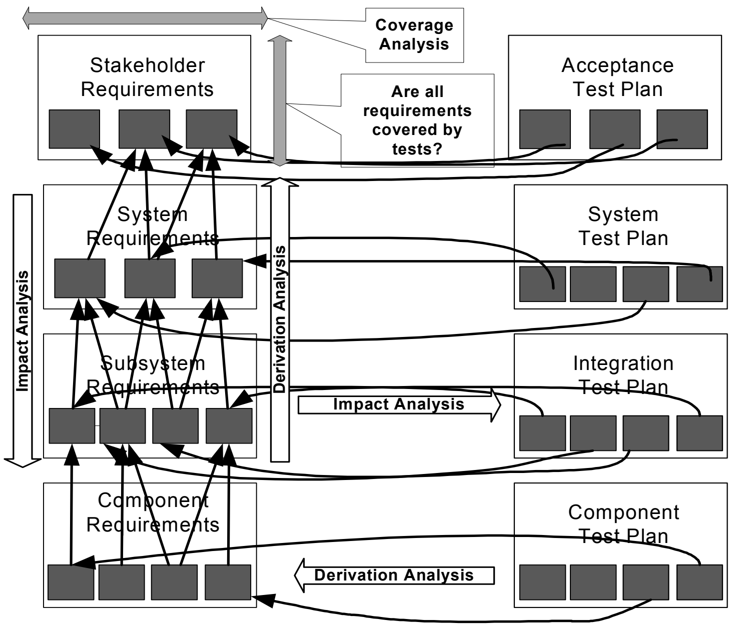


*Figure 8 : Requirements Traceability (Hull, 2005)*

Traceability linking can be used for various kinds of analysis. Creating traceability links between paragraphs of a document or between objects in a requirement database has value for system engineering, traceability links help analysis of impact, coverage and derivation to be performed and enhances the requirement management process (Dick, 2000) .

*Impact analysis:* This is change management process for determining the impact on artefacts when the requirements are changed.

*Derivation Analysis:* Provides reasons for the existences of artifacts.

*Coverage Analysis:* Ensures that all requirements are covered.

The meaning of these relationship links is used to explain how one object may be impacted on by changes to another, however the semantics of these links lack rational association with the relationship, therefore they are not descriptive enough to explain if one or more SR's are required to satisfy the UR. There is therefore a need for deeper and richer semantics in elementary traceability relationships.

It is difficult for a non-technical person to determine the validity of the traceability relationship links, therefore it is important to present "*Satisfaction Arguments"* as a conjunction between the UR and its corresponding SRs describing how each of the SRs could satisfy the UR. *Satisfaction Arguments* have appeared in the literature in several forms. For example, correctness arguments appear in (Jackson, 2001) and (Hall, 2005), satisfaction arguments in (Attwood, 2004; Hammond, 2001) and (Hull, 2005).

Rich Traceability introduces satisfaction argument*s* into elementary traceability, with a detailed description of how an SR will meet its corresponding UR (Attwood, 2004) as illustrated in Figure 9.

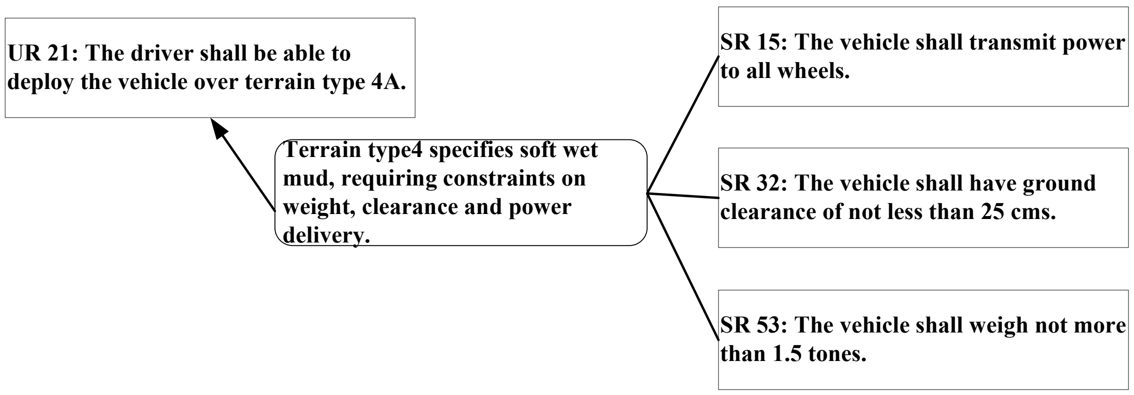

UR 21: The driver shall be able to deploy the vehicle over terrain type 4A.
Terrain type4 specifies soft wet mud, requiring constraints on weight, clearance and power delivery.
SR 15: The vehicle shall transmit power to all wheels.
SR 32: The vehicle shall have ground clearance of not less than 25 cms.
SR 53: The vehicle shall weigh not more than 1.5 tones.

*Figure 9: Rich Traceability (Hull, 2005)*

Rich traceability has advantages over elementary traceability, since the links in the elementary traceability structure lack a relationship rationale between the UR and the SR's (Attwood, 2004).

Satisfaction arguments (SA) use conjunction (&) or disjunction (Or) operators as illustrated in Figure 10, to describe the relationship between the UR and its SR's. The "&" operator indicates that all SR's are required to hold SA and the "*Or*" operator indicates that any one of the SR's is sufficient for the SA to hold (Attwood, 2004; Dick, 2000).

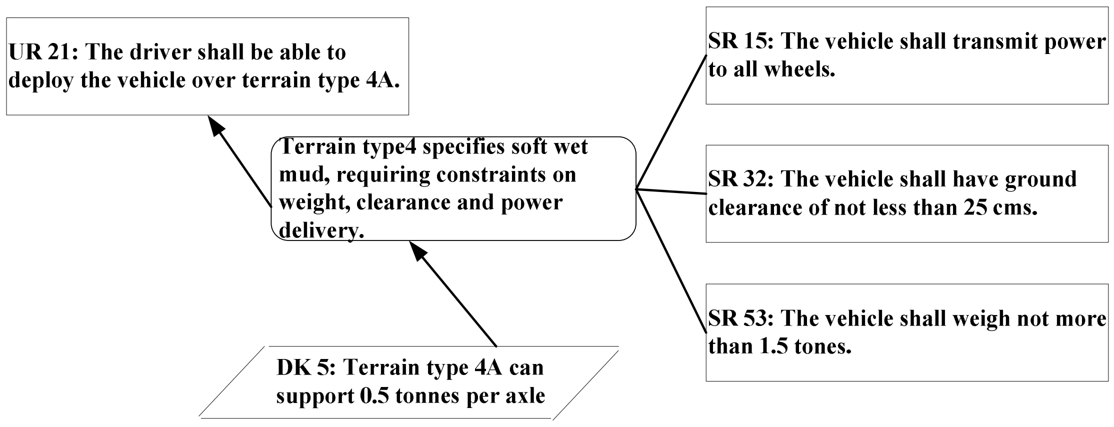

UR 21: The driver shall be able to deploy the vehicle over terrain type 4A.
Terrain type4 specifies soft wet mud, requiring constraints on weight, clearance and power delivery.
DK 5: Terrain type 4A can support 0.5 tonnes per axle
SR 15: The vehicle shall transmit power to all wheels.
SR 32: The vehicle shall have ground clearance of not less than 25 cms.
SR 53: The vehicle shall weigh not more than 1.5 tones.

*Figure 10: Rich Traceability Conjunction (Hull, 2005)*

Though SA's capture the relationship rationale they sometimes lack sufficient evidence to satisfy the requirements, therefore SAs depend on domain knowledge (DK) (see Figure 11 in slanted box) for validity. DK is an argument to support the SA, which is essentially a fact or assumption of the real world and does not constrain the solution (Attwood, 2004; Dick, 2000).

The focus of the requirement phase is to comprehend the problem domain, which may possibly involve a diverse range of people of heterogeneous culture, skill, knowledge and status (Coughlan, 2002). The complexity of the task is multiplied by the diversity (Stuart, 1997). Satisfaction arguments enhance communication (Hull, 2005). The systems requirement specification is written for a specific audience, therefore complexity should be considered while developing satisfaction arguments.

Rich traceability is a many-to-many relationship, the highlighted system requirements flow into more than one SA, meaning that these system requirements satisfy more than one UR and need special attention or, in technical terms, these are reusable software components of high priority.

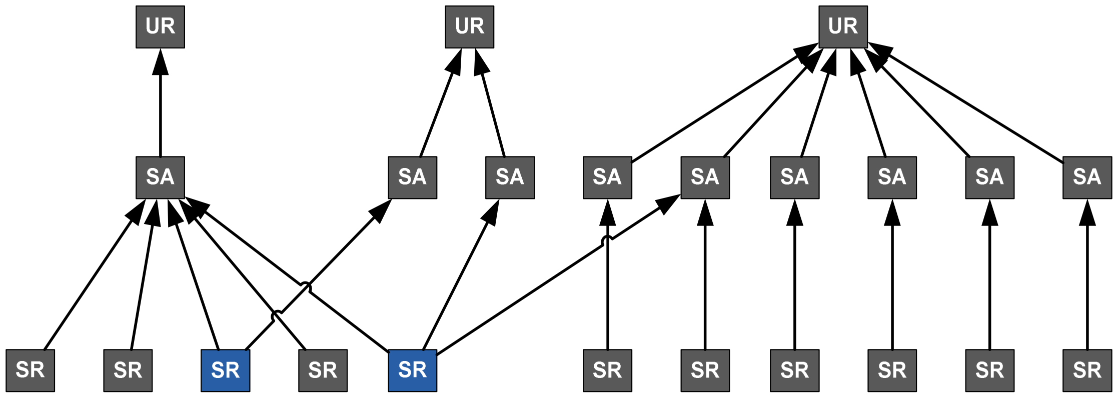


*Figure 11: Rich Traceability as a clustering technique (Hull, 2005)*

As illustrated in Figure 11, SRs are clustered by their corresponding SA's, therefore rich traceability provides us with an inherent clustering ability, which then allows the priority assigned to an SA to propagate to its SR's below and UR above, it further propagates to its dependent SA.

Both elementary and rich traceability provide an inherent clustering technique enabling propagation of priorities from URs or SA to the SR as illustrated in Figure 11. However, prioritising SAs with AHP has benefits over prioritising UR’s with AHP in elementary traceability, due to the rationale associated with the relationships of the SA.

## 2.4 Summary

Systems engineering is about creating solutions for problems, while RE is a subset of system engineering. Software engineering deals with production of software from the early phase of the RE process to maintenance of the system, post production. Software engineering spans across system and requirement engineering, therefore there is an overlap.

RE consists of two processes, requirements development and requirements management. Requirements elicitation, analysis, specification and validation are subsets of requirements development. Feasibility studies are carried out as a first phase of the RE process, in a linear Requirement development process. Feasibility studies are carried out prior to requirement elicitation which is in turn followed by requirement specification and validation. The spiral requirement development process is a three stage iterative model, revolving around requirements elicitation, specification and validation.

The requirements elicitation process is again a, four stage iterative process, beginning with requirement discovery, classification & organisation, prioritisation & negotiation and documentation. A software requirement specification is written early in the software lifecycle and aids communication between the potential stakeholders and the developer. It contains the description of the system functions and constraints.

User Requirements are defined at a high level, their detailed description are the system requirements, which are further classified into functional, detailed descriptions of the user requirements and non-functional, constraints on the service of the system. Domain requirements are derived from the application domain.

The elicited requirements need to be validated to make sure that the software system meets customers’ expectations. The purpose of requirement management is to manage the key RE activities; elicitation, analysis, specification, validation and requirement change management. The major research issue in RE is to transfer RE research into practice.

In a resource constrained development environment, all requirements cannot be met therefore prioritisation is required to determine an optimal set of requirements. Making a decision between requirements is difficult, with tens and thousands of requirements it’s even more complex. However the challenge is selecting the right prioritisation method from the numerous prioritisation methods. Prioritizing requirements is a strategic process which drives development expenses and delivery. Requirements prioritization is a fundamental activity for project success.

An aspect is an attribute or a property of a project requirement that can be used for prioritisation. Some examples of aspect are; importance, penalty, cost, time, and risk. For requirement prioritisation the aspect of the requirement can be considered.

Requirement prioritisation methods can be broadly divided into methods using ordinal or ratio scale measurement. These methods can operate as a stand-alone process or use a cost-value framework within these methods. Prioritising requirements is important in regard to using the limited resources in software development projects. Assigning priorities in a group forces the participants to share their particular knowledge on which priority could be decided, and aids communication, sharing of knowledge, agreement and uncovering potential problems in requirements. The positive side effects of prioritisation are conflict resolution.

Amongst requirement prioritisation methods, AHP is found to be a robust and scalable method, it has a hierarchical problem solving and relative comparison approach that has at least three benefits; Firstly the problem area is elaborated, secondly the grouping aids comprehension / visibility and finally it reduces the number of comparisons within each group which enables in-depth analysis. A hierarchical requirement structure aids implicit prioritisation enabling determination of the most candidate requirements.

Though AHP is considered to be highly sophisticated and ensures granular prioritisation, the drawbacks to using it are high complication and time consumption. Combining requirement prioritisation methods at different stages of the project will possibly reduce complexity. There is lack of tools to support to visualisation of AHP's hierarchies and reduce the clerical burden of comparison in the hierarchy groups.

Combining multiple methods at different stage of the project yields useful results. In an example by Hatton (2008), MoSCoW is used in the beginning stage of the project to group the requirements into Must, Should and Could have, in the mid stage there is greater understanding of the requirement therefore the $ 100 and ranking method is used, finally, when the requirement are fully fleshed out, richer and more sophisticated methods such as AHP are recommended.

It is challenging to assign priority, and select the appropriate prioritisation method, but even more so when, prioritising large scale requirements, distributed geographically with multiple stakeholders. The current state of the art is not fully geared to meet these new challenges.

Conceptualizing large scale systems is challenging, decomposing the system into logical groups helps to manage the complexity better, and requirement clustering provides a new approach to system decomposition which breaks them down to cohesive and loosely coupled modules.

*Requirement Traceability* is the process of creating links between the user requirements and system requirements. This process establishes relationships between requirements in different layers. The benefits of traceability links are *Impact, Derivation* and *Coverage Analysis*. However these traceability links make it difficult for non-technical people to understand the validity of the relationship.

Rich traceability introduces the concept of satisfaction arguments (Dick, 2000) between the UR and SR's supported by additional information in the form of *domain knowledge*. Prioritising *satisfaction arguments* with AHP provides an inherent clustering technique that allows system level requirements to automatically generate a given priority and therefore help multi-directional propagation of priorities between SA and UR, SA and SR and between SA's themselves. This means prioritisation of requirements in large and small scale projects will be automated, reducing the cost of prioritisation.

# 3 Methodology

## 3.1 Research Methodology

The methodology used for this research is the constructive research method (Lukka, 2003) which is defined as "*a research procedure for producing innovative constructions, intended to solve problems faced in the real world and, by that means, to make a contribution to the theory of the discipline in which it is applied*". The goal of this research approach is to develop novel solutions to existing problems.

Constructive research output is "*constructions*", according to Kasanen, Lukka & Siitonen (Kasanen, 1993), which means construction of a solution to the problem which had never been thought of up to now. An element of innovation is clearly evident in constructive research. Another important charactestic of this research approach is that the solution produced can be verified by its implementation. Kasanen, Lukka & Siitonen (1993) list the following six phases that usually occur in constructive research: Problem definition, Literature review, Solution construction, Solution verification, Theory augmentation and Exploration of the solution scope.

Constructive research is extensively used in change impact analysis in software development. This approach is also used for testing the impact of change using Lee, Offutt and Alexanders' (2000) algorithmic methods. O'Neal & Carver (2001) used this method for analysing evolving requirement impacts using requirements traceability.

What is needed for requirements to be prioritised, based on the benefit and cost of requirements, is being researched using a grounded theory approach by Strauss and Corbin (Maya, 2008) which follows a constructivist research paradigm and a systematic review of literature. This research investigates how requirements prioritisation can be undertaken based on cost and benefit. The results of this analysis are "*Activities*" and "*Requirement Properties*" (Daneva & Herrmann, 2008).

Activities are the core RE practices, while requirements properties are (1) *Type* (2) *Estimated Benefit* (3) *Estimated Size* (4) *Estimated Cost* (5) *Priority* (6) *Requirement Dependency*. The type distinguishes primary and secondary requirements. Primary requirements satisfies the customer while secondary requirements are derived from the primary requirement and constrain them (Daneva & Herrmann, 2008).

It was found that "*priority*" is an ambiguous concept and requirement prioritization methods do not define what priority means and prioritization criterion do not offer any support for benefit or cost estimation. Due to the multi-fold dependencies in requirements, determining the benefit, size and cost is a challenge. One way of studying requirement dependencies is the "*benefit function*" commonly used in mathematical economics, which is under-utilised in software engineering. Requirement hierarchies were found to be specific, easier to conceive and refine when they could support cost or benefit estimation (Daneva & Herrmann, 2008).

It was found that RE activities only consider some of requirement properties and some of the requirement prioritisation methods are only specific to some RE activities (Daneva & Herrmann, 2008).

The research outlined in this dissertation follows the six phases outlined by Kasanen, Lukka & Siitonen (1993). This involves:

- Problem definition – In this case, the objective of the research is to provide a simple and precise model for improving the ease with which large requirements sets can be prioritised with a high degree of granularity. (Chapter 1)
- Literature review – In order to fully understand the solution scope, a detailed literature review covering existing models as well as potential techniques for prioritising large requirements sets will inform the research. (Chapter 2)
- Solution construction – This step involves creation of the actual model to be used for impact analysis. One potential technique to be investigated is that of the using a traceability method, known as Satisfaction Arguments (Dick, 2000) as a means of aggregating system requirements and then prioritising them using techniques such as AHP. (Chapter 3.2)
- Solution verification – The model will be verified by application to a dummy set of requirements found on the web titled "*General Purpose Source Particle Module for GEANT4/SPARSET*". As these are dummy requirements, there is no need for ethical approval. (Appendix 1)
- Theory augmentation – This study will help to devise an innovative model for prioritising large requirements sets in software development that would be more easily integrated in the industry than existing approaches and, at the same time, be more precise.
- Exploration of the solution scope – The applicability of the model for prioritising large sets of requirements in the software development process will be explored.

## 3.2 Solution Construction and Experimental Design

For the purpose of this experiment, complexity is implanted into the UR's & SR's by creating dependencies in requirements for testing prioritisation of user requirements and satisfaction arguments with AHP in two different attempts. The experiment is carried out in two phases on the same set of requirements, which involve three *User Requirements and their eight dependent System Requirements*:

***Phase 1:***

a) User Requirements (URs) are prioritised with AHP in a 3 x 3 matrix
b) System Requirements (SRs) are prioritised with AHP in 8 x 8 matrix
c) Test propagation of priorities with AHP
d) Observations

***Phase 2:***

a) Satisfaction arguments (SA) are prioritised using AHP to test propagation of priority.
b) Observations

The purpose of Phase 1.a & 1.b is to subjectively assess the effort involved in prioritising the three user requirements with AHP and then in phase 1.c to test propagation of priority with AHP.

The purpose of Phase 2.a, is to prioritise SAs with AHP and then subjectively assess the effort involved and test possibilities of priority propagation from the SA to its URs and SRs and between SA's.

The efforts of phase 1.a, 1.b are then compared with phase 1.c only. This is done to show the advantages and disadvantages of the way AHP is used.

The effort required in phase 1.c and phase 2.a are then compared to see if prioritising satisfaction arguments has any benefit over prioritising URs, and test the propagation of priorities in both the attempts, and finally assess the efforts / benefits and the similarities and/or differences of each approach with the other, and recommend the best possible solution.

# 4 Results

**Phase 1:** In Sections 2.1 & 2.2, (See Appendix 1), three UR's and eight SR's were prioritised with AHP, with a total of 219 comparisons involved in this effort. This demands a lot of manual calculation and clerical man hours.

The system requirements (SR) are prioritised in an 8 x 8 matrix, meaning all the system requirements were prioritised simultaneously, however of the eight SRs 3 belong to UR1, two belong to UR2 and the remaining three belong to UR3 as illustrated in the Figure 12.

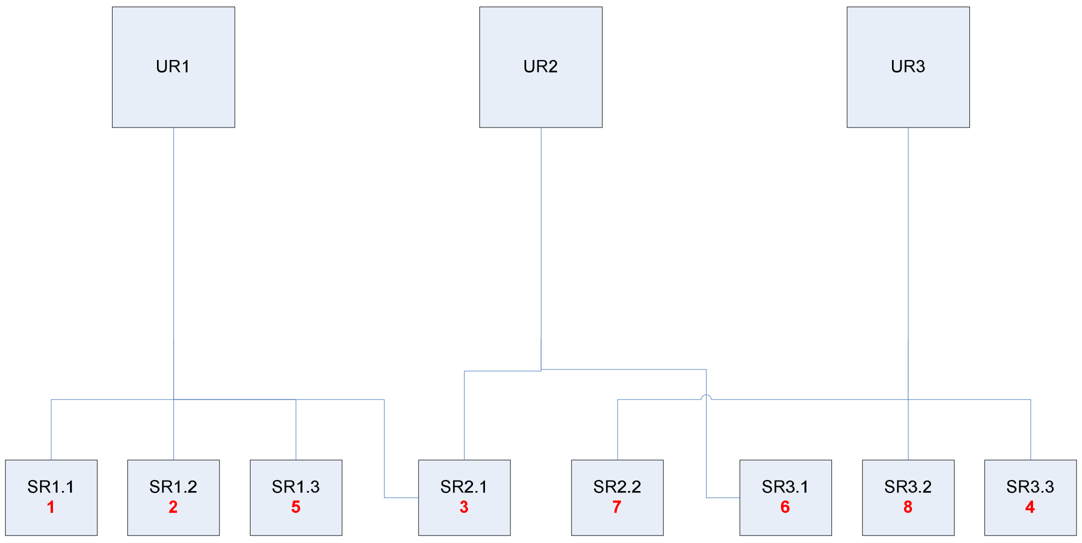


*Figure 12: Prioritizing UR's & SR's with AHP*

AHP process prioritisation in two steps, first it organises the hierarchies and then it applies pair-wise comparisons. Prioritising all SRs together goes against the rules of AHP. The SRs are required to be decomposed into logical groups and then AHP can be applied, so alternatives can be prioritised within each logical group, for example, by making a 3x3 (UR's), a 3x3 (SR's), a 2x2(SR's) and a 3x3(SR's) matrix before applying pair-wise comparisons, the complexity of comparison is reduced. Prioritising SR's in a 8 x 8 matrix was a bit difficult, therefore this calculation was made in the excel document (See Annexure 1 attached).

UR3 has a lower priority than UR1 & UR2, however if it were not implemented there would be a direct impact on UR2, since there is a dependency between UR2 and UR3, which may produce inaccurate priorities.

Though AHP organises requirements into hierarchies, due to a lack of support tools, visualisation of the hierarchy was not possible. However using Excel to prioritise helped to reduce complexity and also helped to visualise the hierarchy.

In phase 1, since UR's were first prioritised and then SR's, the propagation of priority from UR to its SR was not possible. However links between the UR and SR provide traceability, this feature provides the relationship between the UR and its linked SR's.

In section 2.4 (Appendix A) AHP was applied to the same set of three URs, this time, instead of again prioritising SR's, the priority of each UR was mapped onto its dependent SR's, meaning unidirectional propagation of priority downwards only was possible from UR's to their dependent SRs. In this case only 27 comparisons were required, reducing the complexity and computational cost of comparisons.

**Phase 2:** In the second phase, satisfaction arguments (SA) were implanted between the URs and the SRs as shown in Figure 13. This time only the SAs were prioritised using AHP. In this case only 27 comparisons were required for prioritisation. The SA priorities did propagate to their dependent UR and SRs implicitly grouping the dependent SR's, similar to the process in section 2.4. Only 27 comparisons were required in this case making this process faster. The additional information supplied by the SA itself adds richness to traceability.

In Figure 13, SAs are used. The SA for UR1 says functionality of UR1, is delivered by implementing SR1.1, SR1.2, SR1.3 and SR 2.1". In this case the SAs were prioritised and SA1 has priority over SA2 & SA3, therefore priority was propagated to SR1.1, SR1.2, SR1.3 and SR 2.1.

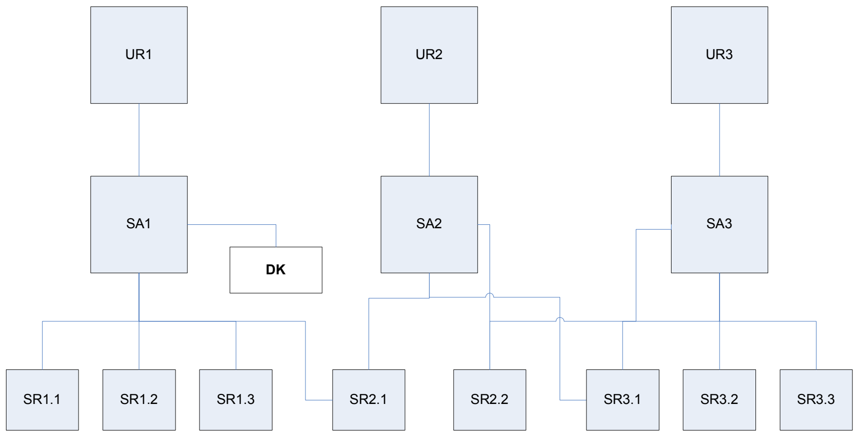


*Figure 13: SA & RT / DK with AHP*

The results of the attempts in Section 2.1 & 2.2 are inaccurate compared to those of 2.4 & 3. Propagation of priority was not possible with in 2.1 and 2.2.

The propagation of priority in phase 2 was bi-directional, for example, as a result of SA1 having priority 1, both its UR and SRs inherit its priority. Further, SA1s priority is inherited by SA2 as well since SR2.1 is required to implement UR1, therefore SA2 has priority 1 as well (see Figure 13).

**Comparison between the three attempts:** Firstly, propagation of priority was possible with both the attempts in Section 2.4 & Section 3 (see Appendix 1). The results of prioritising URs in Section 2.4 and SA in Section 3 with AHP are identical and the number of comparisons was also similar.

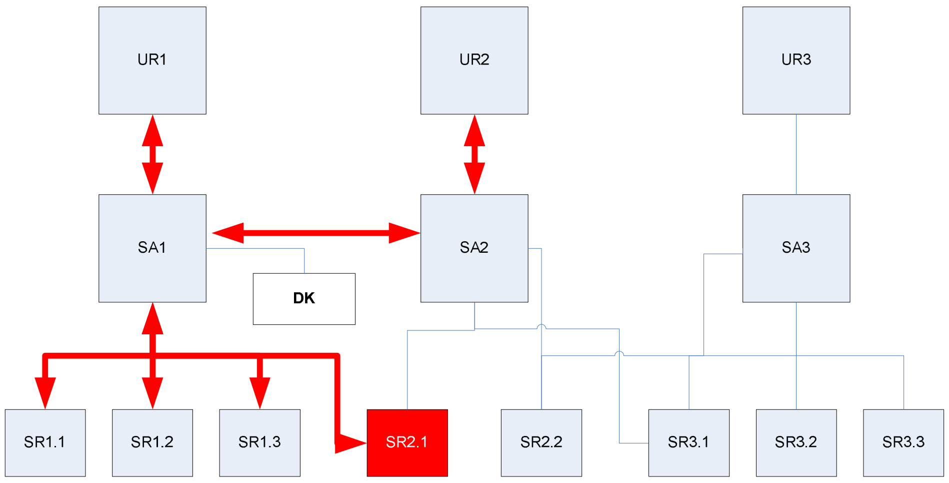


*Figure 14: Propagation of priority*

Secondly, the benefit of prioritising SA's is that the "priority" of SA1 has propagated to SR1.1, SR1.2, SR1.3 & SR2.1. Since SR2.1 is required to implement SA1, the priority of SA1 has propagated to SA2 and UR1 & UR2 above as illustrated in Figure 14. SAs provide richness to traceability. Creating links using rich traceability provides deeper semantics in traceability relationships. (Dick, 2000). Rich traceable links make prioritisation faster.

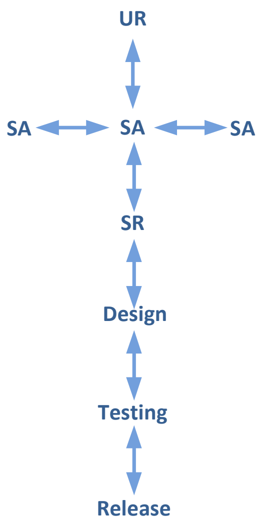


*Figure 15: Bi-Directional Propagation of priorities*

Finally, this idea of rich traceability can be pushed further by allowing relationships between design and requirements, therefore selection of requirements will induce selection of design solutions (Dick, 2000) and priorities can propagate further into testing and even up to the release of software as illustrated in Figure 15.

Conflicts may arise when the originators of URs are different stakeholders and this problem will be complex in ultra large systems or large systems. Each person has his/her own view, when multiple stakeholders exists; multiple views of the same requirement exist. This feature has a positive effect on the RE process, this difference in view can be used to validate the requirement and to detect any dissimilarities, inconsistencies, contradicting requirements and additional requirements which can be used for conflict resolution through communication, and collaboration between people.

Additional support could be provided by a graphical representation or by automated detection of differences between formal specifications to come to a common agreement on the final specifications. Techniques for requirement conflict resolution are cited in sections 3.3 & 3.5 (Pohl, 1994).

Managing a product family requires knowing which range of products or product features are best suited to a given set of requirements. Rich traceability offers the ability to represent a product family as a set of possible responses to a set of possible requirements, providing the ability to manage the product family at a requirement level rather than at component level. In a project with 328 requirements rich traceability reduces the number of choices to 9.

In both the above cases the priorities from URs or SAs are propagated to SRs, however, in the first case though, the priorities are propagated to SRs but there is no textual description for this action, whereas in the second case, the SAs have detailed comments explaining each decision which aids stakeholders' understanding. This added element of clarification provides a stronger basis from which decision can be made.

# 5 Conclusions

Requirement engineering is inherently difficult regardless of the scale of the project (Azar, 2007). There is no off-the-shelf or out-of-the-box solutions for real-life software development efforts, since these tools and techniques are developed for general application development and not for a particular context. Skill is required in selecting the most appropriate technique or tool for a particular context, which may not serve the purpose 100%, but may still be useful.

This research has outlined directions towards an improved process for prioritising software requirements using the concept of satisfaction arguments. This approach allows requirement engineers to effectively and accurately prioritise requirements by prioritising satisfaction arguments using AHP. Propagation of priorities was possible through prioritising URs and SAs with AHP in two separate attempts.

The results of the experiment show a high number of comparisons was required when prioritising three URs and eight SRs with AHP making this process complex and the resulting priorities were inaccurate compared with those prioritised using AHP on satisfaction arguments. However the results of prioritising URs and satisfaction arguments with AHP were found to be similar, the effort required of these two attempts was also similar.

Propagation of priorities was possible in both the case (see sections 2.4 and 3 in Appendix 1) when applying AHP to user requirements or satisfaction arguments. In the case of prioritising user requirements, propagation of priorities was unidirectional while prioritising satisfaction arguments propagated priorities tri-directionally from SA to UR and SRs. Further it was found that the priorities propagate between SAs. User requirement priorities propagate into the design, development and release phases. However due to lack of a visual user interface this benefit cannot be harnessed. When SAs cross multiple URs, the priority is automatically propagated

Prioritising satisfaction arguments is ideal for large scale software development projects as it is found to be a more efficient way of prioritising requirements. It implicitly groups the requirements with a low number of pair-wise comparisons, resulting in a fine grained prioritisation process that reduces time and effort considerably.

# 6 References

Al-Otaiby, T. N., AlSherif, M., & Bond, W.P. (2005). *Toward software requirements modularization using hierarchical clustering techniques.* Paper presented at the Proceedings of the 43rd annual Southeast regional conference - Volume 2, Kennesaw, Georgia.

Alspaugh, T. A., & Anton, A.I. (2001). *Scenario Networks for Software Specification and Scenario Management*: North Carolina State University at Raleigh.

Andriole, S. (1998). The politics of requirements management. *IEEE Software*, 82-84.

Attwood, K., Kelly, T., & McDermid, J. (2004). The Use of Satisfaction Arguments for Traceability in Requirements Reuse for System Families: Position Paper. *in Proceedings of the International Workshop on Requirements Reuse in System Family Engineering, Eighth International Conference on Software Reuse*, 18-21.

Aurum, A., & Wohlin, C. (2005). Requirements Engineering: Setting the Context. In *Engineering and Managing Software Requirements*.

Aurum, A., & Wohlin, C. . (2005). *Engineering and Managing Software Requirements*.

Avesani, P., Bazzanella, C., Perini, A., & Susi, A. (2005). *Facing Scalability Issues in Requirements Prioritization with Machine Learning Techniques.* Paper presented at the Proceedings of the 13th IEEE International Conference on Requirements Engineering, Trento, Italy.

Ayad, H. G., & Kamel, M. S. (2008). Cumulative Voting Consensus Method for Partitions with Variable Number of Clusters. *Pattern Analysis and Machine Intelligence, IEEE Transactions on, 30*(1), 160-173.

Azar, J., Smith, R. K., & Cordes, D. (2007). Value-Oriented Requirements Prioritization in a Small Development Organization. *Software, IEEE, 24*(1), 32-37.

Berander, P., & Andrews, A. (2005). Requirements Prioritization. In *Engineering and Managing Software Requirements* (pp. 69-94). Verlag, Berlin, Germany: Springer.

Berander, P., Khan, K.A., & Lehtola , L. (2006). *Towards a Research Framework on Requirements Prioritization.* Paper presented at the Sixth Conference on Software Engineering Research and Practise in Sweden (SERPS'06) Umeå, Sweden.

Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer, 21*(5), 61-72.

British Computer Society. (2004). The Challenges of Complex IT Projects. The report of a working group from the Royal academy of engineering and the British computer society. *The British Computer Society*, 1-45. Retrieved 17th Oct 2008, from http://www.bcs.org/upload/pdf/complexity.pdf

Brooks, F. P. J. (1987). No Silver Bullet: Essence and Accidents of Software Engineering. *IEEE Computer, 20*, 10-19.

Bush, D., & Finkelstein, A. (2003). *Requirements Stability Assessment Using Scenarios.* Paper presented at the Proceedings of the 11th IEEE International Conference on Requirements Engineering, Los Alamitos, USA.

Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., Natt och Dag, J. (2001). An industrial survey of requirements interdependencies in software product release planning. In: Proceedings of the Fifth IEEE International Symposium on Requirements Engineering. *IEEE Computer society press*, 84-91.

Chantree, F., Nuseibeh, B., de Roeck, A., & Willis, A. (2006). *Identifying Nocuous Ambiguities in Natural Language Requirements.* Paper presented at the Requirements Engineering, 14th IEEE International Conference, Minnesota, USA.

Cheng, B. H. C., & Atlee, J. M. (2007). *Research Directions in Requirements Engineering.* Paper presented at the Future of Software Engineering, 2007. FOSE '07, Washington, DC, USA.

Cheng, E. W. L., & Li,H. (2001). Information priority-setting for better resource allocation using analytic hierarchy process (AHP). *Information Management and Computer Security, 9*, 61–70.

Christel, M., & Kang, K. (1992). Issues in Requirements Elicitation. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.141.5894&rep=rep1&type=pdf

Cleland-Huang, J., Settimi, R., BenKhadra, O., Berezhanskaya, E., & Christina, S. (2005). *Goal-centric traceability for managing non-functional requirements.* Paper presented at the Proceedings of the 27th international conference on Software engineering, St. Louis, MO, USA.

Cleland-Huang, J., Zemont, G., & Lukasik, W. (2004). *A heterogeneous solution for improving the return on investment of requirements traceability.* Paper presented at the Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International, Kyoto, Japan.

Coughlan, J., & Macredie, R. D. (2002). Effective Communication in Requirements Elicitation: A Comparison of Methodologies *Requirements Engineering, 7*(2), 47-60.

Cowley, M., & Domb, E. (1997). *Beyond Strategic Vision: Effective Corporate Action with Hoshin Planning*.

Daneva, M., & Herrmann, A. (2008). *Requirements Prioritization Based on Benefit and Cost Prediction: A Method Classification Framework.* Paper presented at the Software Engineering and Advanced Applications, 2008. SEAA '08. 34th Euromicro Conference.

Davis, A. M. (1990). *Software requirements: analysis and specification*: Prentice Hall Press.

De Neve, P., & Ebert, C. (2001). Surviving Global Software Development. *IEEE Soft*, 62-69.

Dick, J. (2000). *Rich traceability.* Paper presented at the Automated Software Engineering Conference, Edinburgh, Scotland.

Duan, C., Cleland-Huang, J., & Mobasher, B. (2008). *A consensus based approach to constrained clustering of software requirements.* Paper presented at the Proceeding of the 17th ACM conference on Information and knowledge mining, Napa Valley, California, USA.

Engels, G., Kuster, J.M., Heckel, R., & Groenewegen, L. (2001). A methodology for specifying and analyzing consistency of object-oriented behavioral models. *SIGSOFT Softw. Eng. Notes, 26*(5), 186-195.

Feiler, P., Gabriel, R. P., Goodenough, J., Linger, R., Longstaff, T., Kazman, R., Klein, M., Northrop, L., Schmidt, D, Sullivan, K., & Wallnau, K. (2006). Ultra-Large-Scale Systems. The Software Challenge of the Future. *Software Engineering Institute, Carnegie Mellon*.

Fenton, N. E., Lawrence, S., & Pfleeger. (1998). *Software Metrics: A Rigorous and Practical Approach*: PWS Publishing Co.

Firesmith, D. (2004). Prioritizing Requirements. *Journal of Object Technology, 3*(8), 35-47. Retrieved 23-Oct-2008, from http://www.jot.fm/issues/issue_2004_09/column4/

Frauke, P., Armin, E., & Frank, M. (2003). *Requirements Engineering and Agile Software Development.* Paper presented at the Proceedings of the Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises.

Gonzales-Baixauli, B., Prado Leite, J. C. S., & Mylopoulos, J. (2004). *Visual variability analysis for goal models.* Paper presented at the Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International, Springs, CO, USA.

Greer, D., & Ruhe, G. (2004). Software release planning: an evolutionary and iterative approach. *Information and Software Technology, 46*, 243--253.

Groves, L., Nickson, R., Reeve, G., Reeves, S., & Utting, M. (2000). *A survey of software development practices in the New Zealand software industry.* Paper presented at the Software Engineering Conference, 2000. Proceedings. 2000 Australian, Canberra,. Australia.

Gunther, R., Eberlein, A., & Pfahl, D. (2002). *Quantitative WinWin: a new method for decision support in requirements negotiation.* Paper presented at the Proceedings of the 14th international conference on Software engineering and knowledge engineering, Ischia, Italy.

Hall, J. G., Rapanotti, L., & Jackson, M. (2005). Problem frame semantics for software development *Software and Systems Modeling, 4*(2), 189-198.

Hammond, J., Rawlings, R., & Hall, A. (2001). *Will it work? [Requirements engineering].* Paper presented at the Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on, Toronto, Canada

Hatley, D. J., & Pirbhai, I.A. (1987). *Strategies for real-time system specification*: Dorset House Publishing Co., Inc.

Hatton, S. (2007a). Prioritisation of Goals. In J.-L. R. Hainaut, E.A.; Kirchberg, M.; Bertolotto, M.; Brochhausen, M.; Chen, P.; Sisaid Cherfi, S.; Doerr, M.; Han, H.; Hartmann, S.; Parsons, J.; Poels, G.; Rolland, C.; Trujillo, J.; Yu, E.; Zimlanyi, E. (Ed.), *Advances in Conceptual Modeling – Foundations and Applications. Lecture Notes in Computer Science Vol 4802* Berlin: Springer-Verlag.

Hatton, S. (2007b). Early Prioritisation of Goals. *Springer-Verlag Berlin Heidelberg*, 235-244.

Hatton, S. (2008). *Choosing the "right" prioritisation method.* Paper presented at the 19th Australasian Software Engineering Conference, Perth, Western Australia.

Hayes, J. H., Dekhtyar, A., & Sundaram, S. K. (2006). Advancing candidate link generation for requirements tracing: the study of methods. *Software Engineering, IEEE Transactions on, 32*(1), 4-19.

Heimdahl, M. P. E., & Leveson, N. G. (1996). Completeness and consistency in hierarchical state-based requirements. *Software Engineering, IEEE Transactions on, 22*(6), 363-377.

Heitmeyer, C. L., Jeffords, R. D., & Labaw, B.G. (1996). Automated consistency checking of requirements specifications. *ACM Trans. Softw. Eng. Methodol., 5*(3), 231-261.

Hofmann, H. F., & Lehner, F. (2001). Requirements engineering as a success factor in software projects. *Software, IEEE, 18*(4), 58-66.

Holt, J. (2001). *UML for Systems Engineering* from

Hsia, P., & Yaung, A. T. (1988). *Another approach to system decomposition: requirements clustering.* Paper presented at the Computer Software and Applications Conference, 1988. COMPSAC 88. Proceedings., Twelfth International.

Hsia, P., Hsu, C.T., Kung, D.C., & Holder, L.B. . (1996). *User-centered system decomposition: Z-based requirements clustering.* Paper presented at the Requirements Engineering, 1996., Proceedings of the Second International Conference on, Colorado, USA.

Hull, E., Jackson, K., & Dick, J. (2005). *Requirements Engineering* (2 ed.). London: Springer-Verlag.

IEEE. (1998). IEEE Recommended Practice for Software Requirements Specifications. Retrieved from

Jackson, M. (2001). Problem Frames. *Addison Wesley*.

Jiang, L., Eberlein, A., & Far, B. H. (2005). *Combining requirements engineering techniques - theory and case study.* Paper presented at the Engineering of Computer-Based Systems, 2005. ECBS '05. 12th IEEE International Conference and Workshops on the, Maryland, USA.

Kaindl, H., Brinkkemper, S., Bunenko, J.A., Farbey, B., Greenspan, S., Heitmeyer, C., Leite, J.C., Mead, N., Mylopolous, J. & Siddiqi, J. . (2002). Requirements Engineering and Technology Transfer: Obstacles, Incentives and an Improvement Agenda. *Requirements Engineering Journal, 7*(1), 113-123.

Karlsson, J. (1996). *Software requirements prioritizing.* Paper presented at the Requirements Engineering, 1996., Proceedings of the Second International Conference on, Colorado, USA.

Karlsson, J., & Ryan, K. (1997). A cost-value approach for prioritizing requirements. *Software, IEEE, 14*(5), 67-74.

Karlsson, J., Olsson, S., & Ryan, K. . (1997). Improved practical support for large-scale requirements prioritizing. *Requirements Engineering Journal, 2*(1), 51-60.

Karlsson, J., Wohlin, C., & Regnell, B. (1998). An evaluation of methods for prioritizing software requirements. *Information and Software Technology 39*, 939-947.

Karlsson, L., Berander, P., Regnell, B., & Wohlin, C. (2004). *Requirements Prioritisation: An Experiment on Exhaustive Pair-Wise Comparison versus Planning Game Partitioning.* Paper presented at the Empirical Assessment in Software Engineering Conference, Keele, UK.

Karlsson, L., Höst, M., & Regnell, B. (2006). Evaluating the Practical Use of Different Measurement Scales in Requirements Prioritisation. *ACM*, 326-335.

Kasanen, E., Lukka, K., & Siitonen, A. (1993). The constructive approach in management accounting research. *Journal of Management Accounting Research, 5*, 243.

Lamsweerde, A. V. (2001). *Goal-Oriented Requirements Engineering: A Guided Tour.* Paper presented at the Proceedings of the 5th IEEE International Symposium on Requirements Engineering, Toronto, Canada.

Lee, M., Offutt, A. J., & Alexander, R. T. (2000). *Algorithmic analysis of the impacts of changes to object-oriented software.* Paper presented at the 34th International Conference on Technology of Object-Oriented Languages and Systems, Santa Barbara, California, USA.

Leffingwell, D., & Widrig, D. (2003). *Managing Software Requirements: A Use Case Approach*: Addison-Wesley.

Lehtola, L., & Kauppinen, M. (2004). Empirical Evaluation of Two Requirements Prioritization Methods in Product Development Projects. *Springer-Verlag Berlin Heidelberg, 3281*, 161-170.

Lehtola, L., Kauppinen, M., & Kujala, S. (2004). Requirements Prioritization Challenges in Practice. *Springer-Verlag Berlin Heidelberg*, 497–508.

Lena, K., Thomas, T., Bj, rn, R., Patrik, B., & Claes, W. (2007). Pair-wise comparisons versus planning game partitioning--experiments on requirements prioritisation techniques. *Empirical Softw. Engg., 12*(1), 3-33.

Liaskos, S., Lapouchnian, A., Yu, Y., Yu, E., & Mylopoulos, J. (2006). *On Goal-based Variability Acquisition and Analysis.* Paper presented at the Requirements Engineering, 14th IEEE International Conference, Minnesota, USA.

Lubars, M., Potts, C., & Richter, C. (1993). *A review of the state of the practice in requirements modeling.* Paper presented at the Requirements Engineering, 1993., Proceedings of IEEE International Symposium on, San Diego, CA, U.S.A.

Lubars, M., Potts, C., Richter, C. (1993). A review of the state of the practice in requirements modelling. In: Proceedings of IEEE Symposium on Requirements Engineering (RE´93). *EEE Computer Society Press*

Lukka, K. (2003). The Constructive Research Approach. In O. P. H. L. Ojala (Ed.), *Case study research in logistics* (pp. 83--101): Turku School of Economics and Business Administration.

Marcus, A., & Maletic, J.I. (2003). *Recovering documentation-to-source-code traceability links using latent semantic indexing.* Paper presented at the Proceedings of the 25th International Conference on Software Engineering, Portland, Oregon.

Maya, D., & Herrmann, A. (2008). *Requirements Prioritization Based on Benefit and Cost Prediction: A Method Classification Framework.* Paper presented at the Software Engineering and Advanced Applications, 2008. SEAA '08. 34th Euromicro Conference, Parma, Italy.

Mehrdad, S., & Steve, E. (2005). *Traceability in viewpoint merging: a model management perspective.* Paper presented at the Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering, Long Beach, California.

Mishra, D., Mishra, A., & Ali, A. (2008). *Successful requirement elicitation by combining requirement engineering techniques.* Paper presented at the Applications of Digital Information and Web Technologies, 2008. ICADIWT 2008. First International Conference on the, Ostrava, Czech Republic.

Moisiadis, F. (2002, 29-12-2008). *The Fundementals of Prioritising Requirements* Paper presented at the Systems Engineering, Test & Evaluation Conference, Sydney, Australia. http://www.seecforum.unisa.edu.au/Sete2002/ProceedingsDocs/05P-Moisiadis.pdf

Moreira, A., Rashid, A., & Araujo, J. (2005). *Multi-Dimensional Separation of Concerns in Requirements Engineering.* Paper presented at the Proceedings of the 13th IEEE International Conference on Requirements Engineering, Japan.

Nentwich, C., Emmerich, W., Finkelstein, A., & Ellmer, E. (2003). Flexible consistency checking. *ACM Trans. Softw. Eng. Methodol., 12*(1), 28-63.

O'Neal, J. S., & Carver, D. L. (2001). *The impact of changing requirements.* Paper presented at the 2001 IEEE International Conference on Software Maintenance, Florence, Italy.

Phillips, C., Kemp, E.A., & Hedderley, D. (2005). Software Development Methods and Tools: a New Zealand study: Australian Computer Society.

Pohl, K. (1994). *The three dimensions of requirements engineering: a framework and its applications.* Paper presented at the fifth international conference on Advanced information systems engineering, Paris-Sorbonne, France.

Regnell, B., Karlsson, L., & Host, M. (2003). *An analytical model for requirements selection quality evaluation in product software development.* Paper presented at the Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International, Monterey, CA, U.S.A.

Ruhe, G. (2003). Software engineering decision support - A new paradigm for learning software organizations. *Advances in learning software organization, 2640*, 104-115.

Saaty, T. L. (1980). *The Analytic Hierarchy Process*. New York: McGraw-Hill.

Sawyer, P., Sommerville, I., & Viller, S. (1999). Capturing the benefits of requirements engineering. *Software, IEEE, 16*(2), 78-85.

Sommerville, I. (2007). *Software Engineering* (8th ed.). New York: Addison-Wesley.

Stevens, R., Brook, P., Jackson, K., & Arnold, S. (1998). *Systems Engineering: Coping with Complexity.*: Prentice Hall.

Stevens, S. S. (1946). On the Theory of Scales of Measurement. *Science, 103*(2684), 677-680.

Stuart, R. F. (1997). Software Requirements: A Tutorial. *IEEE Computer society press*, 1-21.

Swisher, J. R., & Jacobson, S. H. . (1999). A survey of ranking, selection, and multiple comparison procedures for discrete-event simulation *ACM, 1*.

Swisher, J. R., Jacobson, S.H., & Yücesan, E. (2003). Discrete-event simulation optimization using ranking, selection, and multiple comparison procedures: A survey *ACM, 13*(2).

Thayer, R. H., & Dorfman, M. (2008). Guide for Implementing a Software Requirements Specification. *IEEE Software Engineering Standards and Examples:*. Retrieved 31-12-2008, from http://www.computer.org/portal/site/ieeecs/menuitem.c5efb9b8ade9096b8a9ca0108bcd45f3/index.jsp?&pName=ieeecs_level1&path=ieeecs/ReadyNotes&file=thayer-dorfman-sample.xml&xsl=generic.xsl&

Vargas, L. G., & Katz, M.J. (1990). An overview of the Analytic Hierarchy Process and its applications. *European Journal of Operational Research, 48*, 2-8.

Wiegers, K. E. (1999). *Software Requirements*: Microsoft Press, Redmont, Washington.

Zowghi, D., & Coulin, C. (2005). Requirements elicitation: A survey of techniques, approaches and Tools. In *Engineering and managing software requirements* (pp. 19-46). Berlin, Germany: Springer.

# Appendix – A

## Experiment

## 1. Introduction

This experiment is performed on the real time user and system requirements found on the web titled "*General Purpose Source Particle Module for GEANT4/SPARSET*". The URL for the User Requirement (UR) Document and System Requirement (SR) Document are in the reference section of this appendix.

For the purpose of this experiment complexity is implanted into the User Requirement's (UR) & System Requirements (SR) by creating dependencies in requirements. Firstly we subjectively assess the effort and complexity involved in prioritising UR's & SR's in different variations. Secondly we test "Propagation of Priorities" by prioritising UR's & "*Satisfaction Arguments" (*SAs) with the Analytical Hierarchy Process (AHP). The main purpose of this experiment is to investigate whether more efficient prioritisation of requirements can be achieved by using the concepts of SAs.

Firstly AHP is applied to UR's in a 3 x 3 matrix and then to the SRs in an 8 x 8 matrix. Secondly AHP is only applied to the URs. Finally SAs are implanted between the URs and the SRs and this time AHP is only applied to the SAs.

In this experiment we subjectively assess the "effort", "complexity" and possibilities of "*Propagation of Priorities*" of each of the three attempts, then we can compare the priorities of the three sets of SRs and the complexity involved and look for similarities and/or differences and the reasons for these, which will be discussed in the main dissertation in detail (Chapter 4). The same URs & SRs are used for each of these attempts.

### 1.1 UR's & SR's

**User Requirements**

1. *Table A1: User Requirements*

| Requirement | Description |
|---|---|
| UR1 | Definition of source particle |
| UR2 | Definition of source position distribution |
| UR3 | Definition of source angular direction |

**System Requirements**

*Table A2: System Requirements*

| Requirement | Description |
|---|---|
| SR1 | User input interface |
| SR2 | Source position definition |
| SR3 | Source angular distribution |
| SR4 | Source energy distribution |
| SR5 | User defined biasing function |
| SR6 | Performance requirements |
| SR7 | Interface requirements |
| SR8 | Operational requirements |

## 2. Prioritising With AHP (Case 1)

In this section we will prioritise URs first then SRs and then assess the effort and the results. A 3 x 3 UR requirement matrix is constructed and an 8 x 8 SR requirement matrix.

### 2.1 Prioritise URs with AHP

*Table A3: Pair-wise comparisons matrix*

| Requirement | UR1 | UR2 | UR3 |
|---|---|---|---|
| UR1 | 1 | 7 | 7 |
| UR2 | 0.8 | 1 | 2 |
| UR3 | 0.5 | 5 | 1 |
| | | | |
| **Totals** | 2.30 | **13.00** | **10.00** |

The highlighted normalized value 0.43 in Table A4 below is derived, from, UR1's highlighted value 1 which is divided by the highlighted total value 2.30 from Table A3 above.

*Table A4: Cell percentages*

| Requirement | UR1 | UR2 | UR3 |
|---|---|---|---|
| UR1 | 0.43 | 0.54 | 0.70 |
| UR2 | 0.35 | 0.08 | 0.20 |
| UR3 | 0.22 | 0.38 | 0.10 |

The cell percentages from Table A4 are then transferred into Table A5 below. The rows are totalled and then divided by the number of columns, the resulting value is the priority of the requirement.

*Table A5: Requirement Priority in Ratio*

| Requirement | UR1 | UR2 | UR3 | Row Total |
|---|---|---|---|---|
| UR1 | 0.43 | 0.54 | 0.70 | 1.67/3=0.56 |
| UR2 | 0.35 | 0.08 | 0.20 | 0.63/3=0.21 |
| UR3 | 0.22 | 0.38 | 0.10 | 0.70/3=0.23 |

The final prioritization of the user requirements are in Table A6 below:

*Table A6: User Requirements Priority*

| Requirement | Priority |
|---|---|
| UR1 | 56% |
| UR2 | 21% |
| UR3 | 23% |

Total No of Comparisons:

| | |
|---|---|
| 1. Table 3 (Pair wise comparisons): | 9 |
| 2. Table 4 (Cell Percentages): | 9 |
| 3. Table 5 (Priority Ratio): | 9 |

**Total No of Comparisons: 27**

## **2.2** Prioritise SR's with AHP

The steps in 2.1 are repeated to prioritise the SR's in an 8 x 8 matrix below.

*Table A7: Pair-wise comparisons matrix*

| Requirement | SR1.1 | SR1.2 | SR1.3 | SR2.1 | SR2.2 | SR3.1 | SR3.2 | SR3.3 |
|---|---|---|---|---|---|---|---|---|
| **SR1.1** | 1.00 | 7.00 | 6.00 | 1.00 | 3.00 | 6.00 | 7.00 | 7.00 |
| **SR1.2** | 8.00 | 1.00 | 5.00 | 4.00 | 2.00 | 5.00 | 4.00 | 3.00 |
| **SR1.3** | 0.50 | 0.50 | 1.00 | 2.00 | 2.00 | 5.00 | 4.00 | 3.00 |
| **SR2.1** | 1.00 | 5.00 | 4.00 | 1.00 | 3.00 | 1.00 | 5.00 | 4.00 |
| **SR2.2** | 0.80 | 0.50 | 0.50 | 0.30 | 1.00 | 2.00 | 3.00 | 4.00 |
| **SR3.1** | 0.35 | 0.43 | 0.50 | 1.00 | 3.00 | 1.00 | 4.00 | 6.00 |
| **SR3.2** | 0.50 | 0.30 | 0.60 | 0.50 | 0.02 | 0.04 | 1.00 | 0.05 |
| **SR3.2** | 3.00 | 4.00 | 0.05 | 3.00 | 1.00 | 5.00 | 4.00 | 1.00 |
| **Totals** | **15.15** | **18.73** | **17.65** | **12.80** | **15.02** | **25.04** | **32.00** | **28.05** |

The normalized value is shown in Table A8 below.

*Table A8: cell percentages*

| SR | SR1.1 | SR1.2 | SR1.3 | SR2.1 | SR2.2 | SR3.1 | SR3.2 | SR3.3 |
|---|---|---|---|---|---|---|---|---|
| **SR1.1** | 0.07 | 0.37 | 0.34 | 0.08 | 0.20 | 0.24 | 0.22 | 22.00 |
| **SR1.2** | 0.53 | 0.05 | 0.28 | 0.31 | 0.13 | 0.20 | 0.13 | 0.11 |
| **SR1.3** | 0.03 | 0.03 | 0.06 | 0.16 | 0.13 | 0.20 | 0.13 | 0.11 |
| **SR2.1** | 0.07 | 0.27 | 0.23 | 0.08 | 0.20 | 0.04 | 0.16 | 0.14 |
| **SR2.2** | 0.05 | 0.03 | 0.03 | 0.02 | 0.07 | 0.08 | 0.09 | 0.14 |
| **SR3.1** | 0.02 | 0.02 | 0.03 | 0.08 | 0.20 | 0.04 | 0.13 | 0.21 |
| **SR3.2** | 0.03 | 0.02 | 0.03 | 0.04 | 0.00 | 0.00 | 0.03 | 0.00 |
| **SR3.2** | 0.20 | 0.21 | 0.00 | 0.23 | 0.07 | 0.20 | 0.13 | 0.04 |

The priority of the requirements is calculated in Table A9 below.

*Table A9: Requirement Priority in Ratio*

| Requirement | SR1.1 | SR1.2 | SR1.3 | SR2.1 | SR2.2 | SR3.1 | SR3.2 | SR3.3 | Row Total |
|---|---|---|---|---|---|---|---|---|---|
| **SR1.1** | 0.07 | 0.37 | 0.34 | 0.08 | 0.20 | 0.24 | 0.22 | 22.00 | 23.52/8=2.94 |
| **SR1.2** | 0.53 | 0.05 | 0.28 | 0.31 | 0.13 | 0.20 | 0.13 | 0.11 | 1.74/8=0.22 |
| **SR1.3** | 0.03 | 0.03 | 0.06 | 0.16 | 0.13 | 0.20 | 0.13 | 0.11 | 0.85/8=0.10 |
| **SR2.1** | 0.07 | 0.27 | 0.23 | 0.08 | 0.20 | 0.04 | 0.16 | 0.14 | 1.19/8=0.15 |
| **SR2.2** | 0.05 | 0.03 | 0.03 | 0.02 | 0.07 | 0.08 | 0.09 | 0.14 | 0.51/8=0.06 |
| **SR3.1** | 0.02 | 0.02 | 0.03 | 0.08 | 0.20 | 0.04 | 0.13 | 0.21 | 0.73/8=0.09 |
| **SR3.2** | 0.03 | 0.02 | 0.03 | 0.04 | 0.00 | 0.00 | 0.03 | 0.00 | 0.15/8=0.02 |
| **SR3.2** | 0.20 | 0.21 | 0.00 | 0.23 | 0.07 | 0.20 | 0.13 | 0.04 | 1.08/8=0.13 |

The final prioritized system requirements are in Table A10 below:

*Table A10: User Requirements Priority*

| Requirement | Priority | Priority Order |
|---|---|---|
| SR1 | 2.94 | 1 |
| SR2 | 0.22 | 2 |
| SR3 | 0.10 | 5 |
| SR4 | 0.15 | 3 |
| SR5 | 0.06 | 7 |
| SR6 | 0.09 | 6 |
| SR7 | 0.02 | 8 |
| SR8 | 0.13 | 4 |

Total No of Comparisons:

1. Table 7 (Pair wise comparisons): 64
2. Table 8 (Cell Percentages): 64
3. Table 9 (Priority Ratio): 64

**Total No of Comparisons: 192**

## 2.3 Observations

The number of comparisons of UR's in a 3 x 3 matrix is 27 and 192 for SR's in 8 x 8 matrix. The total number of comparisons in this case is 27 + 192 = 219. To prioritise three UR's and eight SR's has taken a total of 219 comparisons, therefore there is a high cost in terms of manual calculation of priority in this case.

To implement UR1, we require SR1.1, SR1.2, SR1.3 & SR2.1, to implement UR3 we require SR3.1, SR3.2, SR3.3 & SR2.2 and finally we require SR2.1, SR2.2 & SR3.1 to implement UR2, as show in Figure A1.

However if the stakeholders decide not to implement the low priority UR2 then it will have an impact on UR1 and UR3 implementation because SR2.1 & SR2.2 are both required to implement UR1 and UR3 respectively. Therefore prioritising SR's in an 8 x 8 matrix together is not recommended, as the results are not accurate.

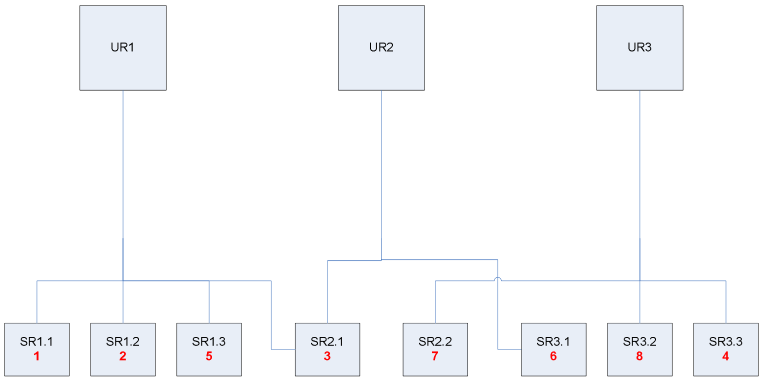


*Figure A1: Prioritisation of URs with AHP*

## **2.4** Propagating Priorities with AHP (Case 2)

The steps in 2.1 are repeated to prioritise UR's in a 3 x 3 matrix below. In this case the UR's are prioritised and the SR's are not, the priorities of the UR are inherited by their dependent SR's.

*Table A11: Pair-wise comparisons matrix*

| **Requirement** | **UR1** | **UR2** | **UR3** |
|---|---|---|---|
| UR1 | 1 | 7 | 7 |
| UR2 | 0.8 | 1 | 2 |
| UR3 | 0.5 | 5 | 1 |
| | | | |
| **Totals** | **2.30** | **13.00** | **10.00** |

The normalised value in Table A12 below.

*Table A12: cell percentages*

| **Requirement** | **UR1** | **UR2** | **UR3** |
|---|---|---|---|
| UR1 | 0.43 | 0.54 | 0.70 |
| UR2 | 0.35 | 0.08 | 0.20 |
| UR3 | 0.22 | 0.38 | 0.10 |

The priority of the requirement is calculated in Table A13 below.

*Table A13: Requirement Priority in Ratio*

| **Requirement** | **UR1** | **UR2** | **UR3** | **Row Total** |
|---|---|---|---|---|
| UR1 | 0.43 | 0.54 | 0.70 | 1.67/3=0.56 |
| UR2 | 0.35 | 0.08 | 0.20 | 0.63/3=0.21 |
| UR3 | 0.22 | 0.38 | 0.10 | 0.70/3=0.23 |

The final prioritized requirements are in Table A14 below:

*Table A14: User Requirement Priority*

| **Requirement** | **Priority** |
|---|---|
| UR1 | 56% |
| UR2 | 21% |
| UR3 | 23% |

Total No of Comparisons:

| | |
|---|---|
| 1. Table 11 (Pair wise comparisons): | 9 |
| 2. Table 12 (Cell Percentages): | 9 |
| 3. Table 13 (Priority Ratio): | 9 |

**Total No of Comparisons: 27**

## 2.5 Observations

The number of comparison involved in prioritisation UR's is only 27. In this case only the UR's are prioritised and then the priority of the UR is mapped onto its dependent SR's, therefore the priorities have propagated from the UR's to its dependent SR's. For example, to implement UR1, we require SR1.1, SR1.2, SR1.3 & SR2.1 therefore these SR's have the top priority 1, likewise to implement UR2, we require SR2.1,SR2.2 & SR3.1 therefore they have priority 2, finally to implement UR3, we require SR3.1, SR3.2 & SR3.3 and SR2.2 as it is last in the list it has priority 3.

*Table A15: User Requirements Priority*

| Requirement | Priority |
|---|---|
| SR1.1 | 1 |
| SR1.2 | 1 |
| SR1.3 | 1 |
| SR2.1 | 1 |
| SR2.2 | 2 |
| SR3.1 | 2 |
| SR3.2 | 3 |
| SR3.3 | 3 |

In this case the priorities have been propagated using AHP and the number of comparisons has been drastically reduced to 27 as opposed to 219 comparisons in section 2.2. As a result of propagation of priorities the SR's have unambiguous or clear priority.

## 3. Satisfaction Arguments (SA) (Case 3)

The steps in 2.1 are repeated to prioritise SA's in a 3 x 3 matrix below. In this case SA's are implanted between the UR's and SR's and then SA's are prioritised with AHP instead of the UR's or SR's.

*Table 16: Pair-wise comparisons matrix*

| Requirement | SA1 | SA2 | SA3 |
|---|---|---|---|
| SA1 | 1 | 7 | 7 |
| SA2 | 0.8 | 1 | 2 |
| SA3 | 0.5 | 5 | 1 |
| | | | |
| **Totals** | **2.30** | **13.00** | **10.00** |

The derived normalized values are transferred to Table A17 below.

*Table A17: Cell percentages*

| Requirement | SA1 | SA2 | SA3 |
|---|---|---|---|
| SA1 | 0.43 | 0.54 | 0.70 |
| SA2 | 0.35 | 0.08 | 0.20 |
| SA3 | 0.22 | 0.38 | 0.10 |

The cell percentages are transferred into Table A18. The rows are totalled and then divided in the row total column; the resulting value is the priority of the requirement.

Table A18: SA Priority in Ratio

| Requirement | SA1 | SA1 | SA1 | Row Total |
|---|---|---|---|---|
| SA1 | 0.43 | 0.54 | 0.70 | 1.67/3=0.56 |
| SA2 | 0.35 | 0.08 | 0.20 | 0.63/3=0.21 |
| SA3 | 0.22 | 0.38 | 0.10 | 0.70/3=0.23 |

The final prioritized requirements are in Table 19 below:

Table 19: Satisfaction Arguments Priority

| SA | Priority |
|---|---|
| SA1 | 56% |
| SA2 | 21% |
| SA3 | 23% |

Total No of Comparisons:

| | | |
|---|---|---|
| 1. | Table 16 (Pair wise comparisons): | 9 |
| 2. | Table 17 (Cell Percentages): | 9 |
| 3. | Table 18 (Priority Ratio): | 9 |

**Total No of Comparisons: 27**

## 3.1 Observations

The number of comparisons involved in prioritisation SA's is only 27 similar to prioritising the UR's with AHP in Section 2.4. The feature "*Propagation of Priority*" when prioritising URs with AHP or prioritising SA's with AHP is similar, however it is easy to prioritise SA's with AHP as they have the "*Reason*" detailed and domain knowledge contributing to richness in traceability, between the UR and the SR's.

Table A20: User Requirements Priority

| Requirement | Priority |
|---|---|
| SR1.1 | 1 |
| SR1.2 | 1 |
| SR1.3 | 1 |
| SR2.1 | 1 |
| SR2.2 | 1 |
| SR3.1 | 1 |
| SR3.2 | 1 |
| SR3.3 | 1 |

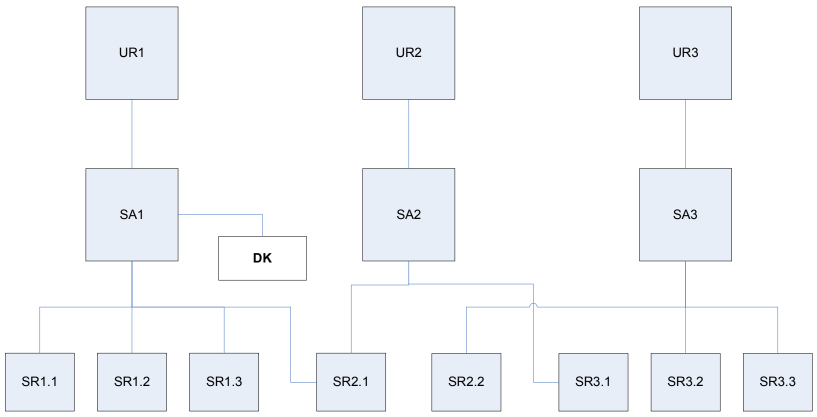


*Figure A2: Prioritisation SA's with AHP*

Since SA1 has priority 1, UR1 and SR1.1, SR1.2, SR1.3 & SR2.1 inherit priority 1, SR2.1 is required by UR1 and UR2 for implementation and SR2.1 has priority 1 and so UR2 also inherits priority 1. Similarly SR2.2 is required to implement UR3 and SR2.2 has priority 1 and so UR3 also inherits priority 1. Therefore In this case the priorities have propagated from SA to SA's vertically and SR's horizontally, respectively, as shown in Figure A2.

## 4.Results

The results of this experiment are summarised in table A21 below.

*Table A21: User Requirements Priority*

| | **Total Comparisons** | **Propagation of Priority** | **Bi-directional propagation** | **Rich Traceability** | **Complexity** |
|---|---|---|---|---|---|
| Case 1 | 219 | No | No | No | Very High |
| Case 2 | 27 | Yes | No | No | Medium |
| Case 3 | 27 | Yes | Yes | Yes | Low |
| | **Totals** | | | | |

Prioritising SA's with AHP has a multi-directional cascading effect beginning at the UR to the release of the software, as illustrated in the Figure A3.

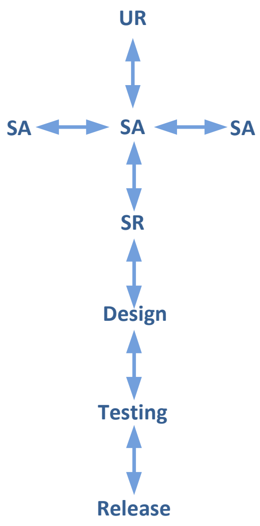


*Figure A3: Bi-Directional Propagation of priorities*

An in-depth analysis and discussion of this experiment is presented in the main dissertation.

## 5.References

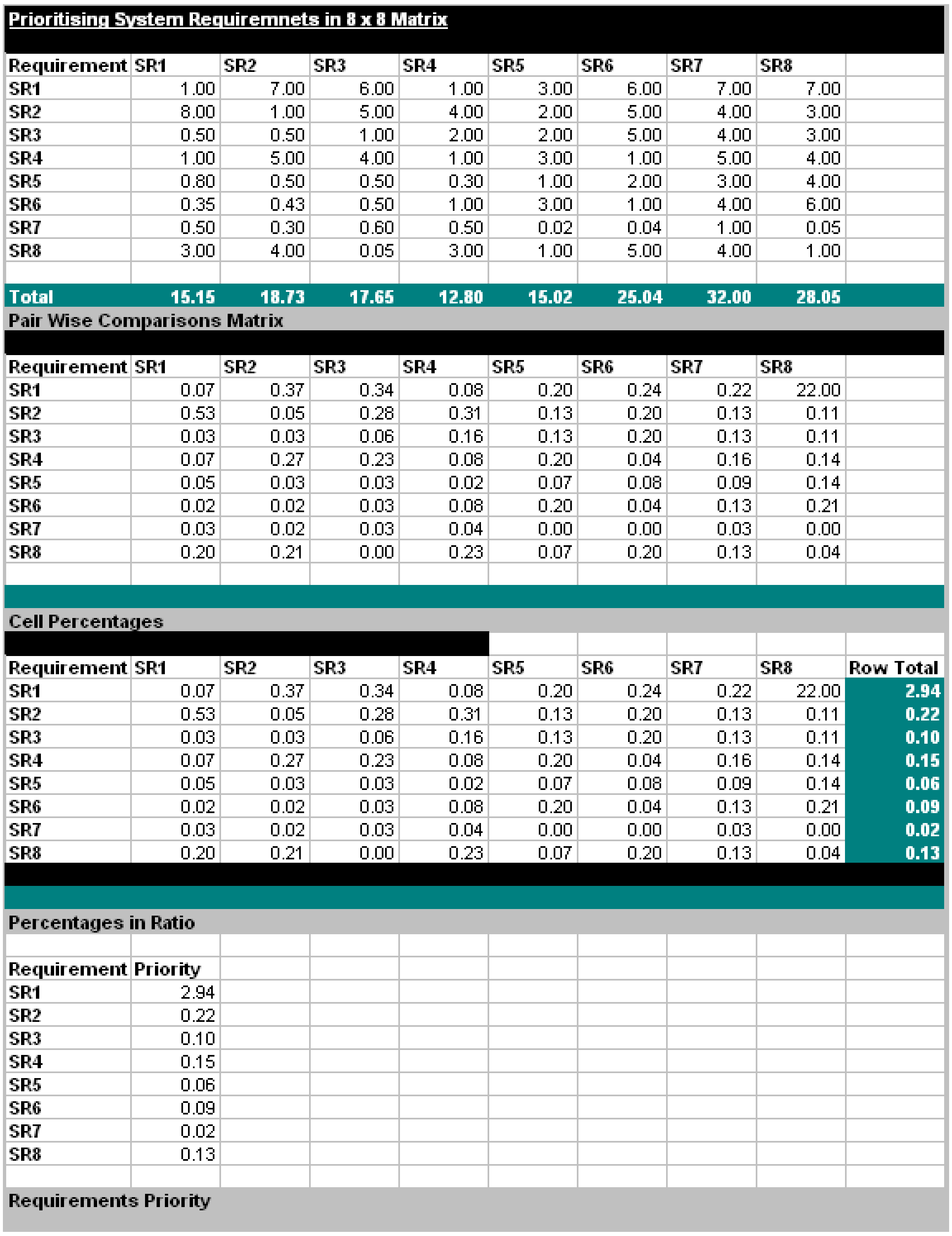
User requirements document
http://reat.space.qinetiq.com/gps/gspm_docs/gspm_urd.pdf

System requirements document

http://reat.space.qinetiq.com/gps/gspm_docs/gspm_ssd.pdf

# Appendix – B

## Annexure – I (Excel calculation sheet)



**Prioritising System Requiremnets in 8 x 8 Matrix**

| Requirement | SR1 | SR2 | SR3 | SR4 | SR5 | SR6 | SR7 | SR8 |
|---|---|---|---|---|---|---|---|---|
| SR1 | 1.00 | 7.00 | 6.00 | 1.00 | 3.00 | 6.00 | 7.00 | 7.00 |
| SR2 | 8.00 | 1.00 | 5.00 | 4.00 | 2.00 | 5.00 | 4.00 | 3.00 |
| SR3 | 0.50 | 0.50 | 1.00 | 2.00 | 2.00 | 5.00 | 4.00 | 3.00 |
| SR4 | 1.00 | 5.00 | 4.00 | 1.00 | 3.00 | 1.00 | 5.00 | 4.00 |
| SR5 | 0.80 | 0.50 | 0.50 | 0.30 | 1.00 | 2.00 | 3.00 | 4.00 |
| SR6 | 0.35 | 0.43 | 0.50 | 1.00 | 3.00 | 1.00 | 4.00 | 6.00 |
| SR7 | 0.50 | 0.30 | 0.60 | 0.50 | 0.02 | 0.04 | 1.00 | 0.05 |
| SR8 | 3.00 | 4.00 | 0.05 | 3.00 | 1.00 | 5.00 | 4.00 | 1.00 |
| **Total** | **15.15** | **18.73** | **17.65** | **12.80** | **15.02** | **25.04** | **32.00** | **28.05** |

**Pair Wise Comparisons Matrix**

| Requirement | SR1 | SR2 | SR3 | SR4 | SR5 | SR6 | SR7 | SR8 |
|---|---|---|---|---|---|---|---|---|
| SR1 | 0.07 | 0.37 | 0.34 | 0.08 | 0.20 | 0.24 | 0.22 | 22.00 |
| SR2 | 0.53 | 0.05 | 0.28 | 0.31 | 0.13 | 0.20 | 0.13 | 0.11 |
| SR3 | 0.03 | 0.03 | 0.06 | 0.16 | 0.13 | 0.20 | 0.13 | 0.11 |
| SR4 | 0.07 | 0.27 | 0.23 | 0.08 | 0.20 | 0.04 | 0.16 | 0.14 |
| SR5 | 0.05 | 0.03 | 0.03 | 0.02 | 0.07 | 0.08 | 0.09 | 0.14 |
| SR6 | 0.02 | 0.02 | 0.03 | 0.08 | 0.20 | 0.04 | 0.13 | 0.21 |
| SR7 | 0.03 | 0.02 | 0.03 | 0.04 | 0.00 | 0.00 | 0.03 | 0.00 |
| SR8 | 0.20 | 0.21 | 0.00 | 0.23 | 0.07 | 0.20 | 0.13 | 0.04 |

**Cell Percentages**

| Requirement | SR1 | SR2 | SR3 | SR4 | SR5 | SR6 | SR7 | SR8 | Row Total |
|---|---|---|---|---|---|---|---|---|---|
| SR1 | 0.07 | 0.37 | 0.34 | 0.08 | 0.20 | 0.24 | 0.22 | 22.00 | **2.94** |
| SR2 | 0.53 | 0.05 | 0.28 | 0.31 | 0.13 | 0.20 | 0.13 | 0.11 | **0.22** |
| SR3 | 0.03 | 0.03 | 0.06 | 0.16 | 0.13 | 0.20 | 0.13 | 0.11 | **0.10** |
| SR4 | 0.07 | 0.27 | 0.23 | 0.08 | 0.20 | 0.04 | 0.16 | 0.14 | **0.15** |
| SR5 | 0.05 | 0.03 | 0.03 | 0.02 | 0.07 | 0.08 | 0.09 | 0.14 | **0.06** |
| SR6 | 0.02 | 0.02 | 0.03 | 0.08 | 0.20 | 0.04 | 0.13 | 0.21 | **0.09** |
| SR7 | 0.03 | 0.02 | 0.03 | 0.04 | 0.00 | 0.00 | 0.03 | 0.00 | **0.02** |
| SR8 | 0.20 | 0.21 | 0.00 | 0.23 | 0.07 | 0.20 | 0.13 | 0.04 | **0.13** |

**Percentages in Ratio**

| Requirement | Priority |
|---|---|
| SR1 | 2.94 |
| SR2 | 0.22 |
| SR3 | 0.10 |
| SR4 | 0.15 |
| SR5 | 0.06 |
| SR6 | 0.09 |
| SR7 | 0.02 |
| SR8 | 0.13 |

**Requirements Priority**