SMART Requirements

Mike Mannion, Barry Keepence
Software Engineering Research Group.
Napier University
Department of Mechanical, Manufacting
and Software Engineering.
10 Colinton Road, Edinburgh
EH10 5DT, United Kingdom

Tel: +44- (0)31-447-9241
Fax: +44 (0)31-447-8046
email: m.mannion@central.napier.ac.uk

Abstract

Systems Analysis, or as it is increasingly known as today, Requirements Engineering, is a time consuming, expensive but critical phase in software (and system) development. The "perfect" Requirements Specification should exhibit a number of qualities including correctness, completeness and consistency. Within a Requirements Specification individual requirements at the microscopic level should be justified, clear, unambiguous and verifiable. However, in many cases Systems Analysts or Requirements Engineers describe requirements which fall short of these demands. In addition, outside reviewers faced with presenting an independent qualitative assessment of a Requirements Specification have few guidelines to assist them. In this paper we present a simple technique, borrowed from objective setting in Management Psychology, to assist the construction and evaluation of individual requirements.

1. Introduction

Systems Analysis, or as it is increasingly known as today, Requirements Engineering, is a time consuming, expensive but critical phase in software (and system) development. In broad terms it can be divided into three areas of activity: elicitation, analysis and specification. In practice elicitation and analysis are performed iteratively and often in parallel. They are notoriously difficult to regularly perform well because they depend on the ability of two people to communicate to each other whether by spoken word, written word or image. For all the methods and techniques that cognitive psychologists have provided us with over the last twenty-five years, including lateral thinking, visual hooks, and picture chains, elicitation and analysis remains an art in which the following skills are demonstrated in abundance:

- filtered listening;
- the ability to describe and explain;
- the ability to grasp new and abstract concepts quickly;
- a genuine interest and enthusiasm for solving other people's problems.

These are inherent human skills which can be taught to many but are mastered by few. In many cases the requirements engineer has a correct understanding but fails to document this accurately in formal documentation.

The skill of specification has improved over the same period of time with the development of a number of specification modelling techniques including Structured Analysis,[1,6], Object Oriented Analysis [7] and Formal Methods [11]. These techniques have evolved to overcome the difficulties of communicating a large amount of detailed and complex information precisely. Each has its own notation which is annotated to a greater or lesser extent by natural language. In general terms the more precise the notation the less annotation is needed, but the greater the learning curve which is required to understand and master the notation. It is this point in particular which has caused the use of these techniques to reside primarily within the domain of the Developer and it remains the case that a set of requirements is written in the first instance in natural language because this is the common language of the Client and the Developer.

There are a number of standards for developing Requirement Specification documents, for example [8,9,10]. Most of these standards make the distinction between a User Requirement Specification (URS) which describes the set of services required by a Client and the Software Requirement Specification (SRS) which describes the set of technical requirements necessary to provide those services, and which is used by the Developer. Typically each standard recommends guidelines on how to partition each document into different types of requirements, including functional, non-functional and external interface. It is also worth noting that the URS is usually in the form of natural language and

the SRS usually includes some form of modelling technique (e.g. dataflow diagrams).

In addition each standard offers in its guidelines for the development of the URS and SRS that the requirements should have a number of attributes including:

- maintainability;
- verifiability;
- completeness;
- correctness;
- consistency;
- clarity;
- traceability;
- modifiability;
- readability;
- ease of use.

Despite these checklists a large number of requirement specifications are being produced which fall short of these demands.

In practice, whilst each of the above characteristics are conceptually understood, their application proves difficult. The problem is twofold, first, although some textbooks and standards provide formal definitions for each characteristic few provide examples or guidelines to illustrate good or bad practice. Secondly, the sheer number of characteristics to remember, each of apparent equal importance and desirability, can be overwhelming to the author of the requirements specification, often with the result that some characteristics are heavily emphasised whilst others are overlooked. Although this imbalance may be partially rectified at formal review meetings when the various participants (customer, managers, designers, maintainers, testers, quality control) display their different interests in the document and argue their separate cases. An improvement would be to have achieved a better balance in the first place.

What is required is a more practical and easily remembered framework which can be used during the development and examination of a requirements specification, be it a URS or an SRS. Such a framework can be useful for both Clients and Developers, experienced and inexperienced, and for independent reviewers, such as the Quality Control team, whose role is to offer a qualitative independent assessment on the Requirements Specification. Quality Assurance personnel in particular often face the difficult challenge of not being familiar with the project but being asked to provide a qualitative judgement on the specification before them. This is made more difficult as the size of the document grows or in cases where documents contain a large amount of acronyms and domain specific jargon.

2. Specifying SMART Requirements

Individual requirements can be compared to personalised objectives. They are goals which are desired to be achieved. In many time management courses and leadership courses, the acronym SMART [12] is used to assist people in setting down good objectives. A SMART objective is:

S pecific
M easurable
A ttainable
R ealisable
T ime bounded.

For personalised objective setting T stands for time-bounded in that the objective must be achieved by a specified date. In the context of software requirements however the dates by which requirements must be achieved is usually specified in the Project Plan (and ultimately the contract). The Project Plan is not usually developed by the author(s) of the Requirements Specifications and can not be produced accurately until the SRS is completed. Consequently it would not be appropriate for a Requirements Engineer to specify time boundaries for individual requirements.

In addition, many requirements in a Requirements Specification are dependent on other requirements or are part of a higher level requirement. A common criticism of some requirements is that the original justification is lost. It would be better for a Requirements Engineer to think of T as standing for **Traceable**. If it is not possible to envisage how a particular requirement is related to other requirements and to know where it came from, then it is not a SMART requirement.

Hence in specifying software requirements we define SMART to be:

S pecific
M easurable
A ttainable
R ealisable
T raceable.

The objective of developing SMART requirements is not to prove that the requirements document is correct in the technical sense (i.e. the requirements state what is actually needed). Using the SMART framework a document can be checked and every requirement can be verified as correct in terms of expression (but not content). However, it is worth noting that a badly expressed requirement is usually a case of incorrect or incomplete analysis. Therefore it is expected that a SMART document is more likely to be technically correct.

The following sections give examples and guidelines for each of the areas in SMART. It is worth noting at this

point that if a requirement fails one of the criteria of SMART it is sometimes because of a failure of another criteria. As an example, a requirement may not be measurable because it is not specific.

2.1 Specific

All requirements techniques have a criteria in this area. A requirement must say exactly what is required. Specificity actually comprises several areas as follows:

- clear i.e. that there is no ambiguity;
- consistent i.e. that the same terminology has been used throughout the specification to describe the same system element or concept;
- simple i.e. avoid double requirements e.g. X and Y;
- of an appropriate level of detail.

A requirement can be tested for this by its reading by a reviewer. There are a number of words and phrases which are first rate indicators of an unspecific requirement. Consider the following requirement:

> "The Mission Planning System shall support several planning environments for generating the mission plan."

In this example, it is not clear what is meant by "several". In addition the terms "planning environment" and "mission plan" may not have been defined.

In general terms the following guidelines are recommended:

- avoid phrases such as: "obviously", clearly", "certainly";
- avoid ambiguities such as "some", several", "many";
- avoid list terminators such as: "etc", "and so on", "...such as";
- ensure pronouns are clearly referenced eg "When module A calls B its message history file is updated";
- when numbers are specified identify the units;
- ensure all possible elements in a list are described;
- use pictures to clarify understanding;
- ensure all system or project terms are defined in a glossary;
- consider placing individual requirements in a separate paragraph and individually numberered;
- ensure verbs such as "transmitted", "sent", "downloaded", "processed" are qualified by precise explanations;
- only use the word "details" , "information", "data" in a requirement when you can describe or refer to precisely what they will be;
- if the requirement is described by a prototype program ensure that specific program is documented;
- when a term is defined in a glossary, substitute the definition in the text and then review the requirement;
- no "To Be Defineds".

Some requirements may seem at first sight to be specific. However they often require a specific definition of a term in order to be specific. Consider:

> "The system shall support 50 simultaneous users."

This requirement is specific in itself but the definition of what the users would be doing is required. Often a certain interpretation is assumed which can be very dangerous.

2.2 Measurable

In the context of Requirements Engineering, by measurable we mean is it possible, once the system has been constructed, to verify that this requirement has been met. In some software engineering methodologies, the Requirements Engineer is instructed to determine the tests which must be performed in order to satisfy the requirement. This is a good discipline. The level of detail required to describe and set up the corresponding test is itself a strong indicator of whether the requirement should be broken down into sub-requirements.

Assuming that a requirement is specific, non - measurable requirements fall into two categories:

> a) those which cannot be instrumented (or instrumentation interferes);
>
> b) those which are specific but for which there is no yardstick available.

Examples of the first category can occur when detailed timing or performance information is required. It may be impossible to measure such values without introducing extensive intrusive software. A common example of this is ensuring that there are no memory leaks in a real-time program. The software to test for the leaks changes the characteristics of the program and therefore the operation of it.

The second category (b) is slightly more subtle. Consider the following requirement:

> "The system shall produce a plan optimised for time."

The only way to measure a plan to see if it is optimal is to compare it to an absolute optimum; which may not be available. Even if the requirement was 90% of optimum the same would apply. This is of course dependent on the test cases used in the acceptance testing. To be measurable the requirement must specify a fixed performance against a predefined set of test cases for which the absolute optimum is known. It is also worth noting that requirements which are in category b need to be made more specific in order to be measuarable.

In general terms the following guidelines are recommended.

- What other requirements need to be verified before this requirement?
- Can this requirement be verified as part of the verification for another requirement ? If so, which one?
- How much data or what test cases are required?
- How much processing power is required?
- Can the test be conducted on one site?
- Can this requirement be tested in isolation?

2.3 Attainable

By an attainable requirement we mean it is possible physically for the system to exhibit that requirement under the given conditions. Some requirements may be beyond the bounds of human knowledge. Others may have theoretical solutions but be beyond what is currently achievable. For example:

> "The system shall be 100% reliable and 100% available".
>
> "The system shall have a minimum response to a query of 1 second irrespective of system load".

These examples are not atypical. The consequence of attempting to meet these requirements is that the system will never be accepted or prohibitively expensive or both. In general terms the following guidelines are recommended:

- Is there a theoretical solution to the problem ?
- Has it been done before ? If not, why not ?
- Has a feasability study been done ?
- Is there an overiding constraint which prohibits this requirement ?
- Are there physical constraints on the size of the memory, processor or peripherals ?
- Are there environmental constraints such as temperature, compressed air?

It is often the case that the attainable and realisable criteria are often considered in parallel. This does not however make them synonymous.

2.4 Realisable

In the context of software requirements, by realisable we mean is it possible to achieve this requirement given what is known about the constraints under which the system and the project must be developed.

Determining whether a requirement is realisable or not is the most difficult part of creating a SMART requirement. The difficulty is twofold in nature:

- can we satisfy this requirement given the other system and physical constraints that we have?
- can we satisfy this requirement given the project resource constraints which we must work to?

For example, if there is a requirement to have 99% reliability but the project budget does not permit the inclusion of the extensive defensive programming needed to satisfy that requirement then that requirement is not realistic.

In general terms the following guidelines are recommended.

- Determine who has responsibility for satisfying the requirement.
  - Can they deliver?
  - Can we afford to manage them?
- How badly is it needed?
- Are there sufficient resources?
  - staff with the right skill set;
  - space and desks;
  - hardware and software for development;
  - hardware and software for testing.
- Is there sufficient time?
- Is there sufficient budget?
- Are we constrained to a particular package which does not support this requirement?
- Will we have to develop it ourselves?
- Can we reuse from other projects?

During the first iteration of the Requirements Specification requirements are often placed into one of two categories:

- essential;
- desirable.

If an analyst is not sure about a requirement then it is often marked as desirable. This does not change the requirement. Desirable requirements should only be left in requirements documents when there is a clear choice in the development stage. For example:

> "The system must have ten operator chairs" - essential
>
> "The operator chairs should be red" - desirable

If the were any other colour, the system would still be acceptable.

2.5 Traceable

Requirements Traceability is the ability to trace (forwards and backwards) a requirement from its conception through its specification to its subsequent design, implementation and test. It is important for the following reasons:

- so that we can know and understand the reason for each requirement's inclusion within the system;
- so that we can verify that each requirement has been implemented;
- so that modifications are made easily, consistently and completely.

Most software development projects which can demonstrate evidence of traceability have been driven to do by the second of these three reasons. This applies also to CASE tools which support traceability. From our own personal industrial experience and from a recent study of practitioners' experiences by Gotel and Finkelstein [5] whilst such a view of traceability is essential it does not help us understand why individual or combinations of requirements have been included, nor explain hidden requirements inter-relationships such as dependency or implication. We believe that in the specification of a requirement the provision of the following supplementary information where appropriate should be made:

- originators of requirements (institutions or people);
- underlying assumptions

  These are particularly important. Often an underlying assumption applies to many requirements but the assumption is stated once, often several pages adrift from a requirement which is dependent on it. It is also vital to ask what will happen when (not if) the underlying assumption is not true[1].
- business justifications;
- inter-relationships such as subsumption, dependency or implication;
- These sorts of relationships are vital in determining the impact of any changes brought about to the requirements specification
- their criticality.

3. SMART Users

SMART as a technique applies to all aspects of specifying a requirement. However not everyone who has an interest in a requirements specification will be concerned with each aspect of SMART. Consider for example the various interested parties of an SRS. Typically, on the Client's side there will be the Procurer and End User, and on the Developer's side, there will be the Project Manager, the Requirements Engineer, the System Designer, the Test Engineer, the Maintainance Engineer and the Quality Engineer.

At a formal review meeting of the SRS, each of these parties will be reading and reviewing the SRS from their own viewpoint (Table 1.1). A Procurer is most interested in requirements being correct, complete and easily changed if necessary ie Specific and Traceable. An End User is interested only in whether the requirements are correct and complete ie Specific. The Project Manager, whilst having a healthy interest in all aspects, is focused on whether the implementation of these requirements can be achieved on time, within budget and to such a level of quality that corrective maintenance does not become a contractual issue ie Attainable, Realisable and Traceable.

The Requirements Engineer, being the author of the SRS, has attempted to construct a document in which all requirements are SMART. The System Designer reviews a requirement asking the questions: "do I understand what is required?" and "is it possible to achieve?". That is, is it Specific and Attainable? Similarly the Test Engineer asks: "do I understand what is required?" and "is it possible to verify the requirement?". That is, is it Specific and Measurable? The Maintenance Engineer needs to understand each requirement, where it came from and what the impact is if it needs changing: that is, Specific and Traceable. The Quality Engineer faces the most difficult task, having to assess whether each requirement is SMART without having been directly involved in its construction nor having responsibility for its subsequent development and implementation.

This matrix helps in the following way. It helps to clarify for individuals during their preparation for a formal review, what their responsibilities are and what their objectives are. This means their own reading of each requirement will be sharper. This in turn will bring about less overlap and deeper coverage by the group as a whole.

---

[1] Typically an assumption is removed from a system (i.e. becomes invalid) but all the requirements which depended upon it are not removed. This often leads to features which are not required still being implemented.

Users of SMART	S	M	A	R	T
Procurer	✓	×	×	×	✓
Analyst	✓	✓	✓	✓	✓
Designer	✓	×	✓	×	×
Project Manager	✓	✓	✓	✓	✓
Quality Engineer	✓	✓	✓	✓	✓
Test Engineer	✓	✓	×	×	×
Maintainer	✓	×	×	×	✓
End User	✓	×	×	×	×

Table 1.1. Interests of different users in SMART

4. Conclusion

Communication between different groups in requirements engineering is very important, and it cannot be relied upon for one group (the user) to learn the craft of another (the analyst). Natural language will continue to be the expression medium for requirements for a long time. However it is by its nature highly dependent upon assumption and definition (even disregarding ambiguity).

The requirements development and evaluation technique we have developed and presented here is a technique which can be used by all people involved in requirements engineering. In SMART we have presented a simple and straightforward system for ensuring that requirement documents are smart.

Most importantly it is independent of any analysis or design methodology which has been used. It is also independent of the requirements extraction method used.

In a forthcoming paper we will presenting SMARTRe in which we extend the SMART acronym to cover Reusable requirements.

References

1. Modern Structured Analysis
E Yourdon, Prentice-Hall, 1989

2. Software Engineering, 4th edition
I Sommerville. Addision-Wesley, 1992

3. Software Engineering: A Practitioner's Guide
R Pressman, McGraw-Hill, 1992

4. Software Engineering: Principles and Practice
H van Vilet, John Wiley, 1993

5. An Analysis of the Requirements Traceability Problem
O Gotel, A Finkelstein, Dept of Computing, Imperial College, 180 Queen's Gate, London, SW7 2BZ, 1993

6. A Practical Guide to Real-Time Systems Devlopment
Sylvia Goldsmith, Prentice-Hall, 1993

7. Object-Oriented Design with Application, 1st Edition G Booch, Benjamin-Cummings Publishing Company, 1991

8. IEEE Standard 830: Software Requirements Specifications

9. European Space Agency Software Engineering Standards
PSS-05-0 Issue 1, January 1987

10. The STARTS Guide
2nd edition, Vol 1, 1987, ISBN 0 85012 G193

11. Z: An Introduction to Formal Methods
A Diller , John Wiley 1990

12. Study Skills,
P Crisfield, L Sollars,
National Coaching Foundation, 1992
Leeds, ISBN 0 947850 872