FEWER FAULTS DUE TO REQUIREMENTS ENGINEERING

Mgr. Zdeněk MARTÍNEK, Doctoral Degree Programme (1)
Dept. of Information Systems, FIT, BUT
E-mail: martine@fit.vutbr.cz

Supervised by: Dr. Jitka Kreslíková

ABSTRACT

This paper proposes requirements engineering as one of many approaches, which have positive influence upon software development. Underestimation of this initial phase has tremendous impact on the whole software development process. Several types of requirements are described and some their difficulties are shown. Then the whole requirements engineering process is described and some its sub-tasks described. Some interconnections to other areas of software development are proposed. Finally, formal approach to this subject is mentioned.

1 INTRODUCTION

Computer based systems surround us more and more. They control more activities and more often critical systems, which may insult human lives. That's why many researchers try to improve influence upon developing fault-free systems. In many books and works, it can be found examples of abandon projects. In many cases, the roots of these failures are very often in lack of precise built and documented requirements of future system. In short, these projects had inadequate requirements engineering.

Very large number of methods and approaches must be mastered to develop system with as low level of errors as possible. Many methods are excessive time-consuming, that's why many non critical systems are developed without their usage. In the worst case, developer doesn't know about them or how to use them at all.

One approach is for example formally specified design of system from which can be partly derived implementation. The major advantage is that formal specification can be verified for particular characteristic. When the design of system is verified, it guarantees lower level of errors (after verification there are no errors in investigated area, but attention must be targeted to verified characteristics).

Another approaches concern with special methods of coding, testing and so on. Some methods can be applied to the whole project or its part or to the managing of whole project. Plan of a large project can be described with a formal language and then verified. Usage of algebra of CSP (Communicating Sequential Processes) for this purpose is described in [1].

2 TYPES OF REQUIREMENTS

Generally, we can say that there is not a big difference between a requirement and a problem. Some definition of requirements (from [2]) may be helpful: “the effects that the client wishes to be brought about in the problem domain”. The problem domain can be defined as: “that part of the universe within which the problems exist” or can be referred to as the operating environment since it includes the environment within which the new system will operate. There exist various types of requirements and some of them are not included at the first definition at all.

2.1 FUNCTIONAL REQUIREMENTS

Appropriate behaviour constitutes the functionality of a system and there is often a tight correspondence between particular requirements and particular functions of the solution system. Some functions may well be referred to as a requirement, but very often they are only functions of the solution system. It may seem subtle, but where the terminology is lost, confusion usually arises.

The identification of functional requirements is also somewhat complicated by the fact that they may be expressed at various levels of abstraction. It is more the thing of detail’s level of description of future system than anything else.

The lowest level of functional requirements is often called the input/output level. This is the realm of the interaction between the problem domain and the solution system. Even if this kind of requirement exists, it is usually best left it to the specification phase.

2.2 PERFORMANCE REQUIREMENTS

Performance is observable and performance requirements relate closely to the required functionality. Performance requirements may be regarded as parameters of functionality in that they determine how quickly, how reliably, within how much occupied space, etc. function must operate.

The performance requirements tend to be particularly volatile requirements. That is why they are separated from other requirements. The cost and timescale of a system can be greatly affected by the required performance. The typical categories of performance requirements are: speed, capacity, reliability, usability. Many other categories exist, but they are quite rare.

Most of performance requirements can be viewed from many points. That is very important when describing them. For example speed can be throughput (usually relevant to off-line systems) or response time (mainly relevant to interactive or real-time systems).

2.3 DESIGN CONSTRAINTS

Design constraints are the true non-functional requirements. They affect how the system is built but not what it does (not directly). The ideal situation, from the developer’s perspective, is that there are no design constraints. For various reasons clients may wish to impose constraints.

There are two main categories. The first one directly constrains the resultant structure of the system (for example: system must be divided into 5 main modules, one for each of the main functions). The second one affects the structure indirectly or not at all (for example: system must run on specific hardware).

2.4 COMMERCIAL CONSTRAINTS

The most important questions for many clients are: “When will I get it?” and “How much will it cost?”. Very often they have firm ideas about the answers and any such requirements form the commercial constraints.

The relationship between timescale and cost and the eventual functionality, reliability, usability, etc. are, to say at least, complex. This problem is studied under project management and software engineering management.

3 SUB-TASKS OF REQUIREMENT ENGINEERING

After describing types of requirements, it must be better specified the whole requirements engineering process. Many publications define it as: “all activities up to but not including the decomposition of the software into its actual architectural components” (for example in [3]). Better one is in [2]: “investigating and describing the problem domain and requirements and designing and documenting the characteristics for a solution system that will meet those requirements”.

Requirements engineering is a process as itself and it may help to understand the whole by considering the various sub-tasks, their products and the interaction between them. The following sub-tasks may be identified:

- elicitation
- analysis
- specification
- human to machine interface (HMI) design
- validation

3.1 ANALYSIS

Analysis can be designated as: “through study of a problem domain, the achievement of understanding of and the documentation of the characteristic of that domain and the problems (requiring solution) that exist within that domain”. It might be more precisely referred to as “problem domain analysis”. In literature there is a lot of synonyms or near synonyms: “systems analysis”, “problem analysis”, requirement analysis”. Within the context of business applications, “business modelling” is often tantamount to the same thing as well.

3.2 ELICITATION

Elicitation concerns the gathering of information and it is also known as requirements capture or requirements acquisition. At first, very little information is known about the problem. Generalised approach must be adopted within domain to find problems. Analysis of the early elicited information will characterise the problem domain and this can then guide the subsequent elicitation. So, there is a feedback loop via analysis and, as requirements engineering progresses, this feedback loop will extend to include specification. Output of elicitation is distinct from analysis output because it is largely unprocessed, unstructured and may contain many irrelevancies.

3.3 SPECIFICATION

Specification is more creative process than analysis, because the new system does not exist and its behaviour must be invented. Usually clients dictate what problems need to be solved by the new system (what requirements must be met), but often they do not dictate the precise system behaviour that will best meet those requirements. Where they do, then specification is simply the matter of documentation. Specification may be defined as: "the invention and definition of behaviour of a solution system such that it will produce the required effects in the problem domain".

3.4 HUMAN TO MACHINE INTERFACE

Sometimes, the detailed external design is largely divorced from specification. This almost always corresponds to having a complex HMI (also called the human to computer interface (HCI)). The essential behaviour of the new system is still designed in the specification but the design of the low-level detail of the HMI is produced too. Another reason for separating is that detailed specification of the HMI can be large and, if included in specification, can mask the essential behaviour of the system.

3.5 VALIDATION

The final sub-task of requirements engineering process is validation. To err is human there is no reason to suppose that errors will not be made during requirements engineering. Problems can arise from misunderstanding between elicitor and client, equivocation in documentation and so on. Mistakes during this phase are often the most pervasive and expensive ones. It is, therefore, particularly important to take steps to minimise errors and to detect and correct them. Validation attempts to ensure that the correct functionality for the solution system has been defined and it follows these simple reasoning:

- if the problem domain behaves as described
- and if the requirements are correctly recorded
- and if the new system behaves as described
- then, provided that the external design is correct

  the requirements will be met.

4 FORMAL APPROACH

Formal methods can be used in many tasks within requirements engineering. One of the most frequent is formal specification. It is based on formal methods which have underlying mathematical notation. This approach try to take advantage from the idea that, if a system could be specified using a formal, mathematical notation then, not only would the specification be amenable to various proofs of logical consistency, but it would also be possible to derive implementation that could be shown to produce the specified behaviour. In other words, it could be proved that, provided only that the specification were correct, there were no bugs in the final programs.

Unfortunately such proof has, despite big effort of many researchers, proved impracticable. It is also the case that the expertise in predicate calculus required of the formal methods practitioner has remained relatively rare, and the approach has not proved particularly accessible for clients or potential users. Furthermore there are limitations as to what can be specified using certain formal method. Usage of these methods is within specification at the logical level and the physical detail of interfaces is almost impossible to address.

Nevertheless, it is now widely accepted that formal methods do have a role (larger than that currently practised) within specification. The precision that they provide can help eliminate ambiguity and misunderstanding in this area.

REFERENCES

[1] Martínek, Z.: How to utilise algebra of Communicating Sequential Processes in modelling of software product development, In: Netss, Ostrava, 2005

[2] Bray, I. K.: An Introduction to Requirements Engineering, Pearson Education Ltd., ISBN 0201 767929

[3] Davis, A. M.: A Taxonomy for the Early Stages of the Software Development Life Cycle, The Journal of Systems and Software, Vol. 8, pp. 297-311

[4] Davis, A. M.: A Comparison of Techniques for the Specification of External Behaviour of Systems, Communications of the ACM, Vol. 31, No. 9, pp. 1098-1115