CONSTRUCTING CONSISTENT USER REQUIREMENTS
Lessons Learnt from Requirements Verification

Petra Heck
Laboratory for Quality Software, TU Eindhoven, Postbus 513, 5600 MB, Eindhoven, The Netherlands

Keywords: User Requirements, Use Cases, Verification, Consistency.

Abstract: The *user requirements* specify what functions an information system has to fulfil. The user requirements serve as the basis for system implementation and test specification. In this paper we present a number of guidelines that improve the quality of the user requirements. .
If the guidelines we present are obeyed during requirements construction, certain types of inconsistencies will not be present in the resulting requirements. Better quality requirements lead to fewer errors in the other system development phases and during system changes.

1 INTRODUCTION

The *user requirements* specify what functions an information system has to fulfil. We assume that before the user requirements are created a context *or* environment description is produced. The context description specifies the business processes the information system plays a role in, and the stakeholders and users of the system. The user requirements should detail each step in the business processes where the information system is involved.

For construction and analysis of user requirements, many methods, techniques and tools have been introduced in the past. Examples of tools are modelling tools for system context or processes and requirement management tools. Methods and techniques have been introduced for the entire requirements engineering life cycle from requirements elicitation to requirements validation.

However, no tool can automatically generate the user requirements for a new system. The elicitation of requirements remains human work. No matter in what form the requirements are written down, the human part of the work leaves room for many mistakes. The type of mistake that we consider in this paper is inconsistency within the requirements. This type of inconsistency can be found by only looking at the requirements specification itself.

To argue about consistency of requirements, the elements they consist of and the relations between the elements must be clear. For this we present an example structure in the next section.

We have used this structure to perform case studies, in which we have checked the requirements of different information systems for consistency. The many inconsistencies discovered during those case studies show that there is not enough awareness of the basic relations between requirements elements among requirements engineers.

In this paper we provide a number of simple guidelines that can prevent (or make it easy to detect) a number of commonly encountered inconsistencies. The guidelines have been discovered during case studies, in which we have manually checked the requirements of different information systems for consistency.

2 GUIDELINES

From the specific findings in the previous section, we can abstract a number of guidelines. Obeying these guidelines in the construction of requirements will reduce the number of consistency defects.

[G1] Structure the Requirements
Before the requirements are created, the structure of the requirements should be clear.

The easiest way to obtain a consistent structure for the requirements is to use templates (these can be e.g. Word templates, but also template projects in a requirement management tool). Good examples of

Heck P. (2007).
CONSTRUCTING CONSISTENT USER REQUIREMENTS - Lessons Learnt from Requirements Verification.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - ISAS*, pages 502-505
DOI: 10.5220/0002345305020505
Copyright © SciTePress

requirements structures can also be found in standards like IEEE Std. 830 (IEEE, 1998). When the template is fixed, the relationships between the different elements in the template should be documented. These relationships lead to simple checks that the users of the templates can perform during requirements creation.

If requirements are not structured, the relationships between the different elements will never be clear and cannot easily be checked for inconsistencies.

[G2] Use (Semi)-Formal Models

Many consistency checks can be automated if the requirements are written down in a formal (i.e. mathematically-based) language. However, in practice these formal languages are not often used. This is due to the training that is required to use them and the difficulty for business users to understand them. Much effort is now put into the creation of languages that are readable for business users (mostly picture based) but also translatable into formal languages for verification purposes. An example is the BPMN notation for business processes (OMG, 2006).

While these languages are not widely available yet, a second-best alternative is to use semi-formal elements (e.g. use case scenarios or data dictionaries) to describe the requirements. These semi-formal elements have less ambiguity than natural language only and thus are easier to validate by hand. Moreover, semi-formalness means that there are a few options to translate (parts of) the elements into formal descriptions and perform formal verification.

[G3] Structure with Use Cases

Among the semi-formal elements, the use cases deserve a guideline of their own.

Use cases (Leffingwell, 2003) are an effective way of writing down functional requirements because their story can be understood by users, developers and testers.

A use case describes sequences of actions a system performs that yield an observable result of value to a particular actor. The particular actor is the individual, system or device that initiates the action.

A use case has four mandatory elements:

- unique name;
- brief description;
- actors;
- flow of events: basic flow and alternate flows (optional situations, odd cases, variants, errors, blocked resources, non-occurring events, etc.).

Use cases have a number of advantages over summing up the functional requirements item-wise:

- Test cases for acceptance testing can be easily derived from the use cases (the same interaction sequence can be used).
- Completeness of the user requirements can be more easily assessed by the walkthrough of these use cases because the interaction is easy to imagine by the users.
- Developers of the system can better understand what each system function comprises because of the coherent way of describing each step in it.

[G4] Create Summary Use Cases

For many information systems, the number of user functions is quite large.

A good solution for structuring the use cases is the introduction of so-called ‘summary use cases’ (Cockburn, 2001). A summary use case describes the user-system interaction on a high-level where each step is itself a use case. If summary use cases are not used, another type of overview must indicate the relationships between the different use cases.

If an overview does not exist, the readers of the requirements specification will have a very hard time to grasp the functionality of the entire system. If the functionality is not clear, readers will overlook items and create inconsistencies in their follow-up work.

[G5] Do Not Forget Literature

Many publications have been written about requirements engineering. All these books, articles and standards contain numerous guidelines on requirements engineering. To read all these would require too much effort.

A good solution is to create a guideline document that collects the relevant guidelines. The reading effort is divided over more people and the new-comers can be informed by reading this document. The guideline document will grow as more people read new literature.

There are for instance many publications on use cases [e.g. (Cockburn, 2001), (Leffingwell, 2003)], but it suffices to take the most important guidelines from books, standards or articles and collect them in the use case template that is used in the company projects.

If literature is entirely ignored the risk of reinventing the wheel exists. This paper already summarizes a number of simple guidelines.

[G6] Specify Functions after User Analysis
Users of the system that are often forgotten are e.g. the system administrator or the manager who only uses the system for monthly reports.

Before the requirements are created, a good analysis of all system user roles during the entire life cycle of the system must be made. If the user roles are known, the system functions for each role can be listed.

If a proper user analysis is left out the final system may miss important user interfaces such as a report facility for managers or a configuration screen for administrators.

[G7] Document the Picture Techniques
Although pictures often contain less ambiguity than natural language, the symbols that are used in pictures can still mean different things to different people: what is the meaning of the shaded symbols, the arrows, the dotted lines vs. the full lines, etc. Most modelling tools, even if they are based on standards like UML, do not put any restrictions on the symbols in a certain diagram type.

To explain the meaning of each symbol and colour used in the picture, a legend must be added to the picture. Another good option is to document the standard meaning in a company reference and only include exceptions with the pictures themselves.

If no explanation of picture symbols (including arrows and lines) and colours is included, different interpretations are bound to be made. The different interpretation will lead to inconsistencies in the implementation or other actions that are based on the pictures.

[G8] Relate Pictures to Text
When both pictures and text are used to explain the same concept (e.g. steps in a use case scenario) they must be consistent with each other.

To make it easy to verify the consistency of the text and the picture, the same wording must be used. If steps are pictured, the steps should be numbered the same as in the text to make it easy to relate the steps. Any other relations between the pictures and text should be marked in the picture.

If text and pictures are not clearly related, it is difficult to compare them. The inconsistencies between text and picture might be overlooked.

[G9] Maintain Cross-References
A cross-reference indicates for a certain type of item in which requirements or use cases a specific item is addressed. Cross-references allow for forward and backward traceability of the requirements.

The cross-reference is useful during update actions of the requirements. If e.g. a new type of printer is attached to the system, the 'peripheral equipment' cross-reference indicates which requirements or use cases are influenced by this. It takes some extra time to maintain the cross-reference unless the requirements are stored in a requirements management tool that automatically creates the indexes.

If no cross-references are maintained during requirement development, an update action will take a long time because all documents need to be searched for references to the updated item. If multiple names were used for the item, references can easily be missed.

[G10] Maintain a Glossary
A glossary must indicate the meaning of each ambiguous or project-specific term in the user requirements. Moreover, if the requirements are written in a non-native language a list of translations should be included. A glossary is not only intended for the lookup of abbreviations. It must provide a unique source for the terminology in the project (e.g. simple things like "do we use 'customer' or 'client' to indicate the buyer of our services?").

A picture of the relations between the different entities can be included to make the glossary more informative. A less informal version of the glossary would be a data dictionary, which also defines data representations.

If not at least a basic glossary is provided for the project the different possible interpretation of terms will lead to inconsistencies in follow-up activities. Moreover, the use of multiple terms for the same entity (because there was no glossary that uniquely defined the term to use) will lead to items being easily overlooked.

[G11] Create Templates Pre-filled with Examples
A template that only outlines the structure of the document can cause different writers to fill in fairly distinct contents for the sections.

To make the template more indicative each section should contain comments on what is expected from the writer (are pictures mandatory? is there a standard way of numbering items? when can this section be left empty? etc.), and an example of the contents. Checks that the author can perform on the contents can also be included in the template.

If a template only contains section headings the contents of the documents will vary from author to author. It is hard to find inconsistencies between documents that are not similar. Guidelines, examples and checks in the templates will improve the quality of the produced documents in general.

[G12] Do Not Use 'N/A'
When templates are used to produce requirements documents, some sections may not apply to all projects. E.g. for a system without peripheral equipment a section describing the equipment is not needed.

When a template is the basis for a document, no sections can be left out otherwise the structure among different projects will become inconsistent. However, it does not suffice to simply mark the section as 'N/A' or 'Not applicable", because it is not clear why the section is not applicable. E.g. is there no peripheral equipment in the system or is it not used for this use case?

Each occurrence of 'N/A' must be clarified with a reason, otherwise people can not verify that the 'N/A' was appropriate (it could have been e.g. a copy-paste error) or when it should be updated (when the reason becomes invalid). 'N/A' will lead people to ignore the section in all future uses of the document.

An incorrectly specified 'N/A' is a major inconsistency in the document.

[G13] Keep Track of Open Points
Requirements are usually written in more than one cycle. During the earlier cycles, not all information might be available. The missing information leads to so-called 'issues' or 'open points'.

At the final delivery of the requirements, all open points must be solved. The best way to make sure that none of the documents contain open points is to keep track of the open points on a central list with referrals to the related documents. An open point may only be closed after it has been updated in the corresponding documents.

If open points are left in the documents they are incomplete and some parts cannot be used for follow-up actions.

3 RELATED WORK

Certain standards provide guidelines for requirements, e.g. IEEE Std 830 (IEEE, 1998), IEEE Std 1233 (IEEE, 1998) or the ESA standards (ESA, 1991). Our guidelines supplement these standards. Our guidelines are on a greater level of detail than the standards. The standards specify a format for the requirements, some guidelines for the contents of the requirements, and processes to create requirements.

The idea of using formal models for requirements engineering is not new and promoted by many authors, e.g. (Parnas, 1995), (Lamsweerde, 2001). The use case-based approach is also not new, e.g. (Cockburn, 2001), (Leffingwell, 2003).

The guidelines we have presented in the previous section are not new, but the overview we have presented can serve as a quick-reference for requirements engineers.

4 CONCLUSIONS

Our paper aims to serve as a practical overview of simple guidelines that companies can incorporate in their requirements engineering processes in order to reduce the chance for inconsistencies.

If the guidelines we present are obeyed during requirements construction, certain types of inconsistencies will not be present in the resulting requirements. Better quality requirements lead to fewer errors in the other system development phases and during system changes.

REFERENCES

Cockburn, A., 2001. Writing Effective Use Cases, Addison-Wesley. London.

ESA Board for Software Standardisation and Control (BSSC), 1991. ESA software engineering standards, Issue 2.

IEEE Computer Society, 1998. IEEE Recommended Practice for Software Requirements Specifications, IEEE Std 830-1998.

IEEE Computer Society, 1998. IEEE Guide for Developing System Requirements Specifications, IEEE Std 1233-1998.

Lamsweerde, A. van, 2001. Building Formal Requirements Models for Reliable Software. In *Ada-Europe 2001*. Springer-Verlag.

Leffingwell, D., Widrig, D., 2003. Managing Software Requirements. A Use Case Approach, Pearson Education. Boston, 2nd edition.

OMG, February 2006. Business Process Modeling Notation (BPMN) Specification. Version 1.0, Final Adopted Specification. Retrieved from http://www.omg.org/docs/dtc/06-02-01.pdf.

Parnas, D.L., Madey, J., 1995. Functional Documentation for Computer Systems. In *Science of Computer Programming*, vol. 25, no. 1, pp. 41–61.