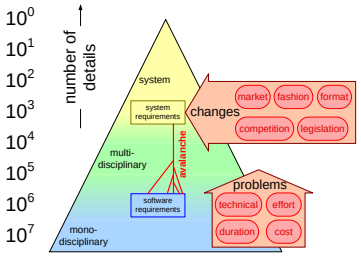
What is a Good Requirement Specification?

-



Gerrit Muller
University of South-Eastern Norway-NISE
Hasbergsvei 36 P.O. Box 235, NO-3603 Kongsberg Norway
gaudisite@gmail.com

Abstract

Requirements play a driving role during product creation. The requirements are captured in a requirements specification. How can we assess the requirements specification? What are the criteria for a good specification?
We discuss these aspects by positioning the requirements specification in the broader context of customers, market, product creation and product life-cycle. We zoom in to the software requirements specification, to discuss the criteria for this mono-disciplinary specification.

version: 0.2 status: concept September 6, 2020

1 Introduction

Software and System Engineering education puts a lot of emphasis on requirements engineering. A key success factor for product creation is to start with valid requirements. But how do we validate our requirements? What makes a requirements specification into a *good* requirements specification? Looking again at the education we see that the formalization aspects is emphasized: requirements must be SMART (Specific, Measurable, et cetera) and requirements describe *what* not *how*. In practice we have to add a number of *common sense* criteria to assess a requirements specification. Paul Davies[1], for instance, formulates ten questions to ask about the requirements specification.

In this article we will discuss a Personal Video Recorder (PVR) as an example product. We will discuss the difference between software and system requirements. The uncertainty and dynamics of requirements heavily influences the product creation approach. We will explain why the waterfall model fails in most environments, and why incremental development models provide better means to validate requirements.

This article reuses material from several earlier Gaudí articles:

- Requirements `http://www.gaudisite.nl/RequirementsPaper.pdf`
- From the soft and fuzzy context to SMART engineering. `http://www.gaudisite.nl/FromFuzzyToSmartPaper.pdf`
- Architecting for Humans; How to Transfer Experience? `http://www.gaudisite.nl/ExperienceTransferPaper.pdf`
- Industry and Academia: Why Practioners and Researchers are Disconnected. (to be published in 2005)

2 Personal Video Recorder case

Many modern appliances cause an alienated feeling for (less-technical) consumers. Figure 1 shows a multiple choice set of feelings for programming the well known Video Cassette Recorder (VCR). This task of programming the VCR is often delegated to the family member with sufficient technical feeling.

We use *time shift recording* as an example of desired user functionality. Figure 2 shows the concurrent activities that occur when straightforward time shifting is used. In this example the user is watching a movie, which is broadcasted via conventional means. After some time he is interrupted by the telephone. In order to be able to resume the viewing of the movie he pauses the viewing, which starts invisible the recording of the remainder of the movie. Sometime later he resumes

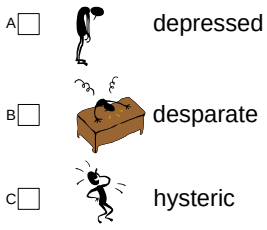


Figure 1: Did you ever program a VCR?

viewing where he left of, while in the background the recording of the not yet finished movie continues.

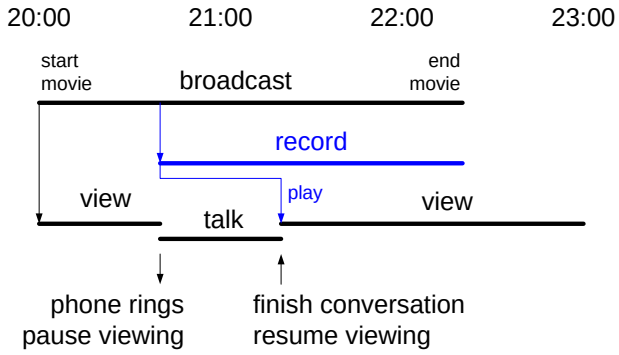


Figure 2: Example Time Shift recording

In this simple form (pause/resume) this function provides freedom of time to the user. This appears to be very attractive in this interaction modus. However when such an appliance is designed limits out of the construction world pop up, which intrude in the user experience. The list below shows a number of construction limits, which are relevant for the external behavior of the appliance.

- number of tuners
- number of simultaneous streams (recording and playing)
- amount of available storage
- management strategy of storage space

Construction limits, but also more extensive user stories, see figure 3, show how the intrinsic simple model can detoriate into a more complex interaction model. Interference of different user inputs and interference of appliance limitations compromise the simplicity of the interaction model.

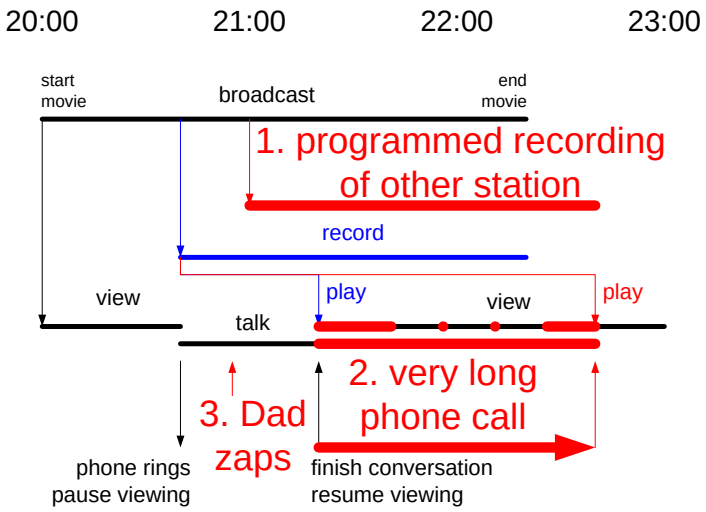


Figure 3: What if conflicting events happen during the pause interval?

In the Post doctoral education for computer science designers at the technical university Eindhoven, the students have to design such a time shift appliance. In the function of ”requirement expert” I was involved in this design workshop. The initial effort of the students was heavily focused on creating a requirement specification, full with tables defining requirements. This thick stack of paper did not really help the students to understand the essence of the appliance, nor did it help to identify the critical or difficult issues. After challenging them the students build a functioning prototype on a PC, which immediately surfaced a number of critical issues and enabled discussion and feedback on the user interaction model, see figure 4.

One of the important means to achieve successful products is the abundant use of feedback. Figure 5 shows some important aspects of obtaining feedback; get architects and designers out of the development laboratory; use short development cycles and observe and listen to users.

3 Assessing the Requirements Specification

A requirements specification drives the project plan and the product creation process. A “good” requirements specification facilitates a focused and smooth development process. But how to determine if a requirements specification is “good”?

Figure 6 shows criteria that can be used to assess a requirement specification. The product to be created should satisfy the stakeholders. This translates into a criterium for the specification: the specification must reflect the real needs of all

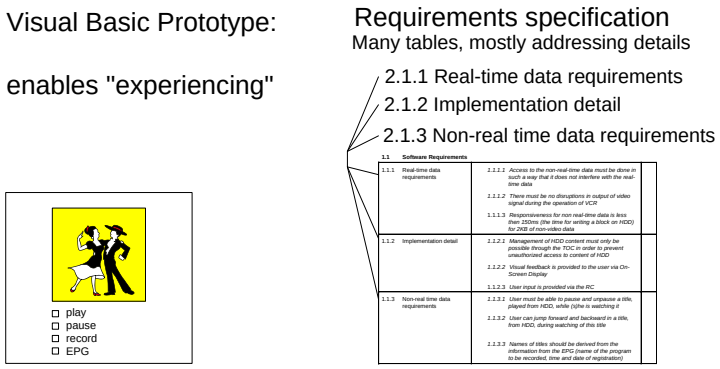


Figure 4: OOTI workshop 2001: “Requirements Engineering”

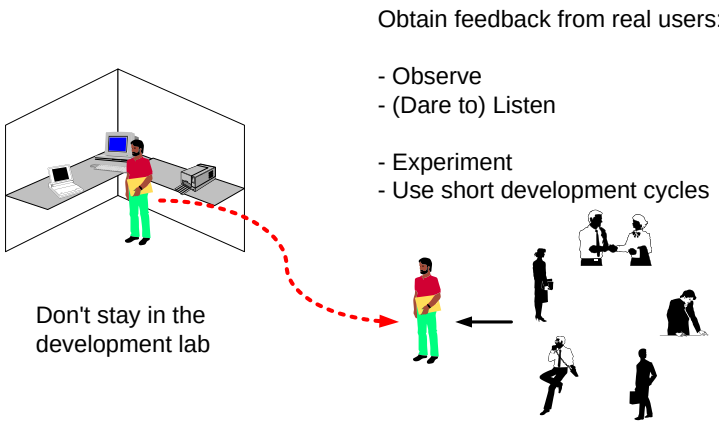


Figure 5: Key Success Factor: Feedback

stakeholders. The difficult part of this criterium is that *real needs* are unknown. The demands of stakeholders are often formulated in terms of solutions. These solutions are based on limited know-how and lots of assumptions about constraints.

The group of stakeholders is heterogeneous and asymmetric. Satisfaction of the (external) customer is very important. Satisfaction of internal stakeholders is also important, but some of the internal stakeholder processes are the subject of the product creation process itself. For instance, the manufacturing stakeholder will always oppose changes, because changes cause disruptions of the manufacturing. However, innovation requires changes. The smooth ramp up of manufacturing is part of the product creation process.

Many complementary viewpoints are required to collect the requirements. Figure 7 shows a useful number of viewpoints when collecting requirements. The viewpoints

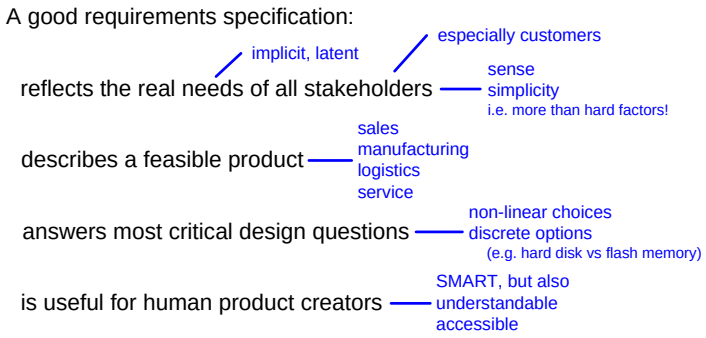


Figure 6: Criteria for a Good Requirements Specification

are classified in *top-down* and *bottom-up* viewpoints.

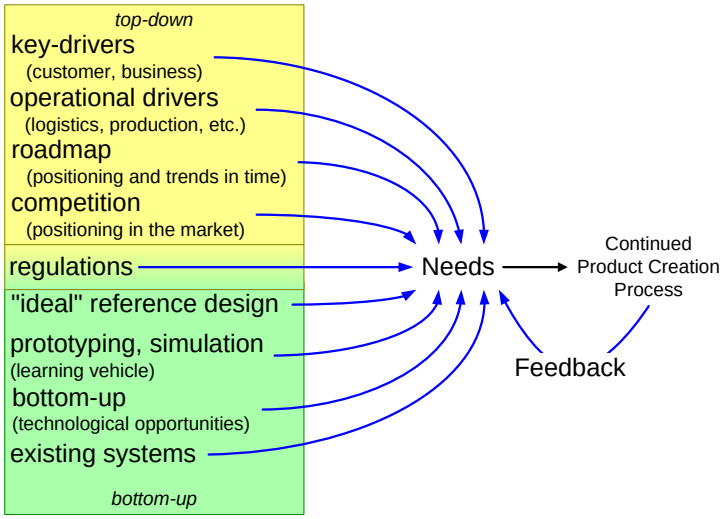


Figure 7: Multiple Viewpoints to Understand Needs and Feasibility

Top-down viewpoints. The **key-driver** viewpoint and the **operational** viewpoint are the viewpoints of the stakeholders which are "consuming" or "using" the output of the product creation process. These viewpoints represent the "demanding side". The **roadmap** and the **competition** viewpoint are viewpoints to position the requirements in time and in the market. Those viewpoints are important because they emphasize the fact that a product is never made in isolation, but in a rather dynamic and evolving world.

Bottom-up viewpoints. The **"ideal" reference design** is the challenge for the architect. What is in his vision the perfect solution? From this perfect solution

the implicit requirements can be reconstructed and added to the rest of the requirements. **Prototyping or simulations** are an important means in communication with customers. This "pro-active feedback" is a very effective filter for nice but impractical features at the one hand and it often uncovers many new requirements, which do not appear with a pure paper approach. The **bottom up** viewpoint is the viewpoint which takes the technology as the starting point. This viewpoint sometimes triggers new opportunities which are overlooked by the other viewpoints due to an implicit bias by today's technology. The **existing system** is one of the most important sources of requirements. In fact it contains the accumulated wisdom of years of practical application. Especially the large amount of small but practical requirements can be extracted from existing systems.

The requirement specification is a dynamic entity, because the world is dynamic: the users change, the competition changes, the technology changes, the company itself changes. For that reason the **Continuation of the Product Creation Process** will generate input for the requirements as well. In fact nearly all viewpoints are present and relevant during the entire Product Creation Process.

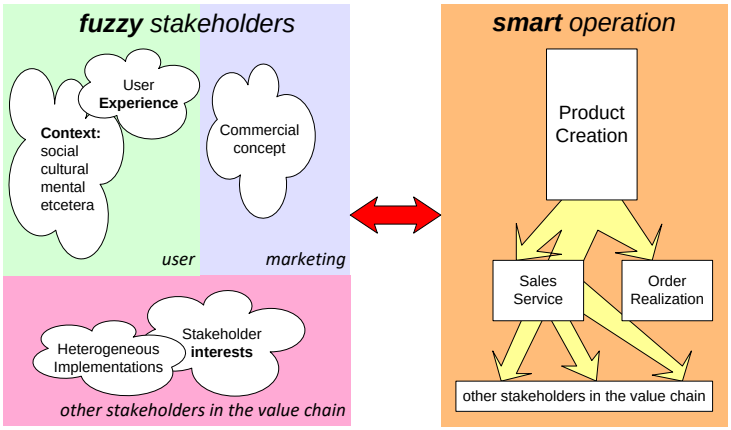


Figure 8: How SMART can requirements be described?

All the stakeholders involved in this supply chain need specific and verifiable data to order, assemble, test and sell the product. (Did you ever try to tell a sales manager: "don't worry the product will be fast, will have a nice image quality and it will be very fashionable", without any further hard facts?)

Figure 8 shows the problem statement by visualizing all the fuzzy needs at the one hand and the SMART facts at the other hand.

Standards like ISO 9000 or methods like CMM prescribe the requirements for the requirement management process. These requirements are shown in the column "smart operation" in Figure 9: specific, unambiguous, verifiable, quantifiable, measurable, complete, and traceable.

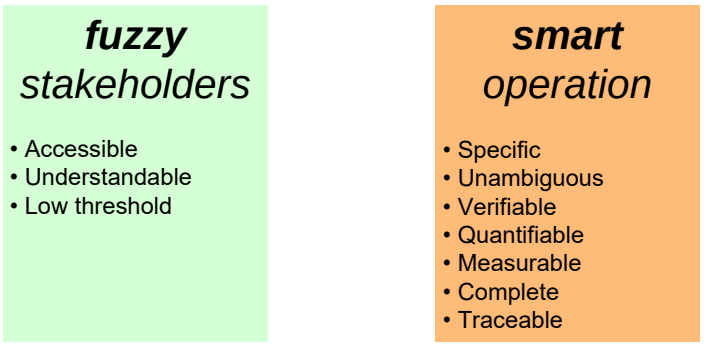


Figure 9: Requirements must be SMART and Usable

Unfortunately these requirements are always biased towards the formal side. A process which fulfills these requirements is from technical point of view sound and robust. However an important aspect which is forgotten quite often is that product creation is a human activity, with all their human capabilities and constraints. The Human point of view adds a number of requirements, which are required for **every** stakeholder, shown in the column *fuzzy stakeholders*: accessible, understandable, and low threshold.

These requirements, which are imposed by the human element, can be conflicting with the requirements which are prescribed by the management process. Many problems can be traced back to violation of the human imposed requirements. For instance a customer requirement which is described so abstract that no real customer can understand it anymore is a severe risk, because early validation is impossible.

4 From System to Software Requirements

When SW engineers demand "requirements", then they expect *frozen* inputs to be used for the design, implementation and validation of the software. So far, however, we have discussed *system* requirements. System requirements describe the **what** at system level. The system requirement specification can be a limited document, at least if the authors focus on the most important and relevant system functions and characteristics.

The translation of system requirements into detailed mono-disciplinary design decisions spans many orders of magnitude. The few statements of performance, cost and size in the system requirements specification ultimately result in millions of details in the technical product description: million(s) of lines of code, connections, and parts. The technical product description is the accumulation of *mono-disciplinary* formalizations. Figure 10 shows this dynamic range as a pyramid with

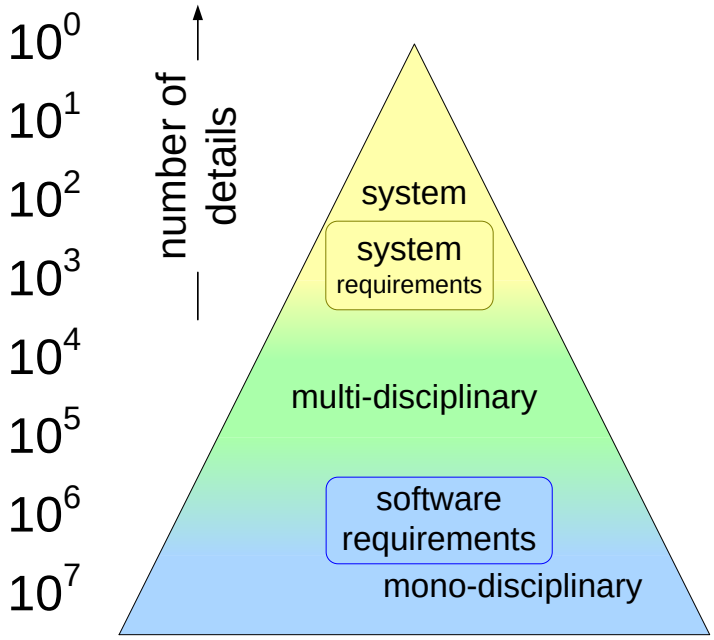


Figure 10: System versus Software Requirements

the system at the top and the millions of technical details at the bottom.

The amount of details in a software requirements specification is several orders of magnitude more than the amount of details in the systems requirements specification. Figure 11 shows the software "subsystem" in its context. All the relations of the software subsystem with its context must be reflected in the software requirements specification. The software requirements specification is part of the detailing process of the system design and implementation.

The *user interface* and *system behavior* depends on many design choices. The software in most systems is the technology that implements both *user interface* and *system behavior*. Embedded systems interact with the physical world. The software implements the control of actuators and sensors that perform the interaction with this physical world. The related hardware-software interface (HSI) is a broad interface. The HSI determines many software design choices, and becomes part of the software requirements specification. Software needs a computing infrastructure to be executed upon. The computing infrastructure is always limited, putting constraints on the software. The combination of performance and cost requirements are translated into resource management requirements for the software subsystem. The software development department and environment result in operational requirements for the software subsystems. For instance in terms of tools and languages, and in terms of programming conventions and rules.

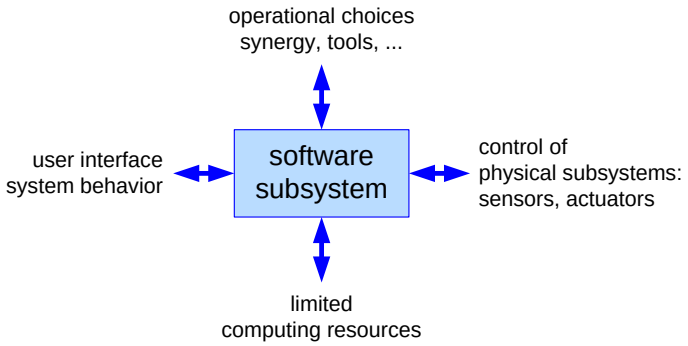


Figure 11: Why is the Software Requirement Specification so Large?

The amount of details in the software requirements specification is huge. One of the consequences is that this specification is never complete nor up-to-date, see Figure 12. The environment of the software requirements specification is in practice highly dynamic. The outside world is changing in many ways (market, competition, legislation, fashion, and format). A small change in the outside world (top-down) may cause many changes in the software requirements. Design and implementation problems (technical, cost, effort, and duration) trigger bottom-up changes that may propagate into changes of the system requirements specification. Bottom-up changes can also trigger an avalanche of changes in the software requirements.

5 Conclusions and Recommendations

We have shown that system and software requirements are part of a dynamic and complex world. Requirements are targeted at multiple audiences. Many stakeholders need readable and understandable requirements, while the product creation crew needs SMART requirements. *Software* requirements tend to have much more detail than *system* requirements.

Figure 13 shows the conclusions and recommendations based on the summary above.

6 Acknowledgements

Feedback from Teade Punter helped to shape this article.

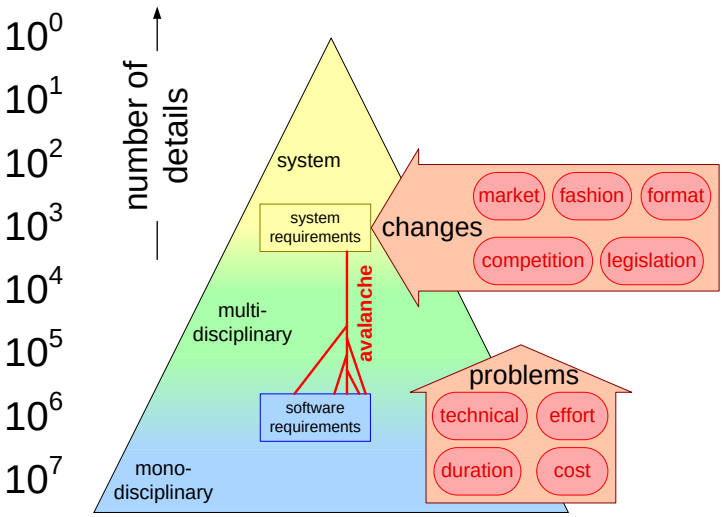


Figure 12: And why is the Software Requirement Specification never up-to-date?

References

[1] Paul Davies. Ten questions to ask before opening the requirements document. In *INCOSE 2004 14th Annual International Symposium Proceedings, Toulouse, France*, 2004.

[2] Gerrit Muller. The system architecture homepage. `http://www.gaudisite.nl/index.html`, 1999.

History

Version: 0.2, date: November 9, 2004 changed by: Gerrit Muller

- Spell checked the article
- changed status to concept

Version: 0.1, date: October 28, 2004 changed by: Gerrit Muller

- Added figures to assess requirements and to relate fuzzy needs to SMART requirements
- added text
- changed status to draft
- added acknowledgements

Version: 0, date: September 22, 2004 changed by: Gerrit Muller

- Created, no changelog yet

Never wait for the software requirements specification to be complete

1) it is never complete

2) it is never up-to-date

3) the product will be too late.

Be creative to cope with uncertainty and dynamics

for instance, use prototype as specification "WYSIWYG"

use incremental development strategies (XP, EVO, ...)

focus on most important and critical issues

Figure 13: Conclusions and Recommendations