THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

DEPARTMENT OF
COMPUTER SCIENCE
Te Tari Rorohiko

2025

Contributors

J. Turner

R. Mercado

V. Moxham-Bettridge

J. Kasmara

THE SOFTWARE DEVELOPMENT LIFECYCLE: ANALYSIS

Last week we briefly discussed the planning phase of the SDLC, this included determining exactly what it is that a client wants from a new piece of software. This week we will move on to the analysis phase. More than just knowing what the client needs or wants from a piece of software, we must clearly define the requirements that the software must adhere to.

What is Analysis?

Recall from last week that the analysis phase consists of two sub-processes:

- What are the requirements of the system?
- Out of those requirements, which are important, redundant etc?

Specifically, we consider two types of requirements, functional and non-functional requirements. Functional requirements describe the functionality that is required of a software system while non-functional requirements are criteria the system and user interface must satisfy to be acceptable. To quote the Software Engineering Body of Knowledge (SWEBOK): "Software requirements express the needs and constraints placed on a software product that contribute to the solution of some real-world problem."[1]

Consider the following video: https://www.youtube.com/watch?v=zCX-N1H8Vps which reiterates the differences between functional and non-functional requirements.

[1] http://swebokwiki.org/Chapter_1:_Software_Requirements

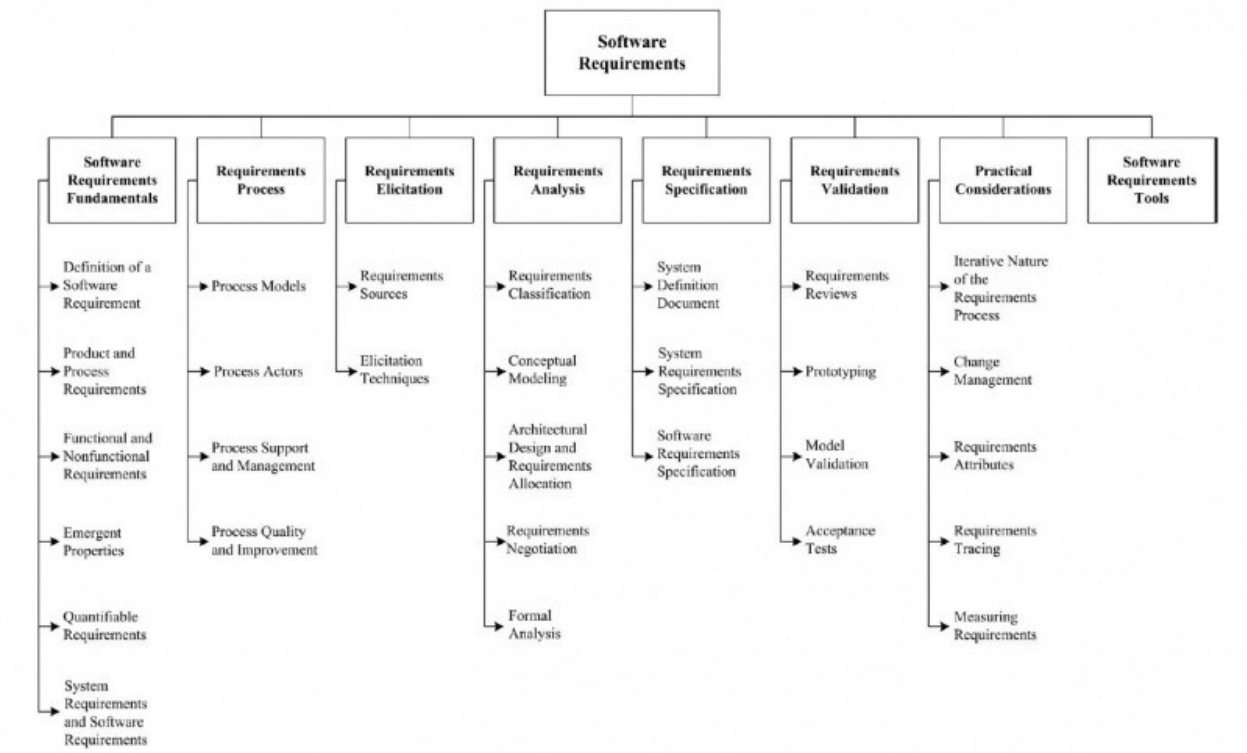

Figure 3: Software Requirements Fundamentals[2]

Figure 3 depicts the broader area and different tasks involved in the software requirements stage. For this session we will only focus on the first area "Software Requirements Fundamentals". By briefly reading over each of the different areas you can see how necessary software requirements are to the development process. They are used as the basis for what the software should do. If a system fails to adhere to its requirements we can say that it does not work as expected. In order to determine this, verification, validation and/or testing is carried out to ensure that the software has been implemented correctly (we will look at this in more detail later).

Software Requirements

Naveda et al. in their book "IEEE Computer Society Real-World Software Engineering Problems, 2006, Vol. 6"[3] detail various requirements problems through a series of questions. Let's take a look at some of these problems and their explanations next to help gain a better understanding of software requirements.

[2] http://swebokwiki.org/File:Software-requirements.jpg
[3] Naveda, J. (2013). Software Requirements. In IEEE Computer Society Real World Software Engineering Problems (pp. 15-78). Hoboken, NJ, USA: John Wiley & Sons.

Exercise #1

Jo is gathering the requirements for a software-controlled furnace. After interviewing several users, Joe obtained the following requirements list:
R1. Gas inlet valves should always be open when furnace is heating.
R2. Heating stops when furnace temperature reaches 150 degrees celsius.
R3. Furnace temperature should increase gradually when heating.
R4. The gas inlet valves should be close when the temperature goes about 200 degrees celsius.

Which requirements defects (ambiguous, design dependent, incomplete, unverifiable) can be identified in Jo's requirements list?

R3 is ambiguous and unverifiable because of the use of the word "gradually" because we cannot quantify its meaning. That is, we cannot know what amount to increase the temperature by in order for it to be gradual. To fix this we would need to specify the rate of increase e.g. "1 degree celsius per second".

The requirements Jo has specified are also incomplete because they do not explain what happens when the heating is complete (over 150) or when the valves are closed (over 200). For example, we could assume that valves remain open but we do not know for certain.

Exercise #2

During a software development project, 2 similar requirements defects were detected. One was detected in the requirements phase, and the other during the implementation phase. Which of the following statements is most likely to be true?

a. The most expensive defect to correct is the one detected during the requirements phase.
b. The most expensive defect to correct is the one detected during the implementation phase.
c. The cost of fixing either defect will usually be similar.
d. There is no relationship between the phase in which a defect is discovered and its repair cost.

The answer here is b, the longer it takes for a defect to be found the more expensive it is to correct the design. Empirical evidence shows that the cost to correct a defect found in the design is generally 2.5 to 5 times the cost of fixing a defect found during the requirements phase.

Similarly, a defect during implementation will cost 5 to 10 times as much, and one found in testing costs 10 to 20 times as much.

Exercise #3

Which of the following statements is not a valid requirements specification?

a. Software shall be written in C.
b. Software shall respond to all requests within 5 seconds.
c. Software shall be composed of the following 23 modules.
d. Software shall use menu screens whenever it is communicating with the user.

The answer here is c as it is a statement describing the structure of the software. Software requirements should not include project requirements, design information or product assurance plans. Conversely, A is a valid non-functional requirement, while B and D are valid functional requirements.

Exercise #4

You are a requirements engineer working on a project to enhance a course-registration and payment system for a large public university system. Although you are new to the tasks needed to complete the project's requirements phase, you are asked to lead the team.

You need to elicit requirements for the system. Which elicitation technique will BEST allow you to understand both the typical and atypical activities and tasks involved in course registration and payment?

a. Observation (watching people use the existing system)
b. Prototypes (building mock ups of the software)
c. Interviews (sitting down and asking people who use the existing system questions)
d. Questionnaires (sending out questions and asking people who use the existing system to respond)

The answer here is a, observation. While best practice will require you to use multiple requirements elicitation techniques, observation is the best to uncover the way people actually perform their work (atypical) as opposed to the documented process (typical).

The other options may also be used but each has its own specific benefits in terms of the requirements. Developing a quick prototype provides a starting point for discussion and will help

to get a sense of initial requirements. Interviews assist with finding conscious and concrete requirements. Lastly, questionnaires are the most limited, as they tend to address items we already know about.

Exercise #5

You are a requirements engineer working on a project to enhance course-registration and payment system for a large public university system. You are the requirements analyst. You are also a recent university graduate, so you have experience in the problem domain. Consequently, less time is allocated for understanding the problem domain.

Which of the following is MOST LIKELY consequence of conducting the requirements specification phase under these conditions?

a. Subtle mismatches between your conceptual understanding and the proper meaning of concepts within the domain are present and will require rework.
b. The time saved during the elicitation process is allocated to the requirement validation process, where more defects can be detected.
c. Time is saved during the requirements specification process since you do not need to spend time asking follow-up questions to the stakeholders.
d. Mismatches in your conceptual understanding and the proper meaning are less likely to occur. The quality of the requirement specification is maximised.

The answer here is again A. Your prior understanding may conflict with what the client actually wants. Van Vliet states, “Subtle mismatches between the analyst’s notion of terms and concepts and their proper meaning within the domain being modelled can have profound effects. Such mismatches can most easily occur in domains we already ‘know’, such as a library.” Therefore, it is unlikely that any prior knowledge you have will actually be useful, and if you rely on that prior knowledge you may actually introduce defects and increase time costs.

Furthermore, keep in mind that software developers are not “typical” users of software, as they are more likely to be able to work around any problems that they might encounter. We need to be able to design software for non-technical users and it is important that we take this into account throughout the entire SDLC process.

Exercises: Analysis

On the Slack channel there is a completed example proposal for the software that you began designing last session. Using this proposal, define a list of what you think are the functional and non-functional requirements needed for the software. Here are some examples to get you started:

Functional Requirement:	R1. Filter data carbon emissions by company.
Non-functional Requirement:	R2. Results from carbon emissions data must be returned within 2 seconds.

Summary

In this session we covered functional and non-functional requirements as part of the Analysis phase of the SDLC. Next week we will begin discussing the design phase where we will introduce personas and scenarios, which are useful methods when determining and clarifying a system's intended purpose and audience.

Useful Resources

- Naveda, J. (2013). Software Requirements. In IEEE Computer Society Real World Software Engineering Problems (pp. 15-78). Hoboken, NJ, USA: John Wiley & Sons. https://ieeexplore.ieee.org/book/5989366
- P. Bourque and R.E. Fairley, eds., Guide to the Software Engineering Body of Knowledge, Version 3.0, IEEE Computer Society, 2014; www.swebok.org