

Requirements and specifications

Paolo Burgio
paolo.burgio@unimore.it



UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

High Performance
Real Time **Lab**

Expectations



vs reality



From specifications, to requirements

Specifications are a **contract** between us and the customer

- › At a first step, we define what the system must do, and we derive a set of requirements
 - Specification requirements
 - We speak of software functionalities

Then, for each phase of the development flow, we create a more detailed/tech set of requirements

- › System specs&reqs
- › Module/component specs&reqs
- › From each of them, we create tests

These are agreed between designers and developers, the customer has nothing to do with this!

- › Unless he has some constraint / expertise in the field
- › Beware: often, the customer thinks that SW development is easy!
- › *“Lo fa anche ammiocuggino”*



What vs How

Specs tells us **what** the system should do

- › So, functional requirements do

Implementation tells us **how** the system does things

- › So, system/module/component requirements do
- › Typically, start with a top-down (*divide-et-impera*) approach, and include existing modules, if necessary

Every step of the development process defines a contract between its sub-parts

- › This contract can be, e.g., a Middleware protocol in the system design phase, or Java interfaces of the sub-modules of a single module/component

..and so on, and so on...

- › The deeper we go in the process, the more in detail we go with technology



What vs How: example

What: users shall authenticate in the system

How

- › Users shall enter their data in a form
- › Design: is it a custom webform? Is it a google form?

More deeper

- › Users shall enter the password in an obfuscated field in the webform, and click "submit"

Even more deeper

- › The data shall be transmitted to a webservice that returns 400 in case age is null

...and so on, and so on...



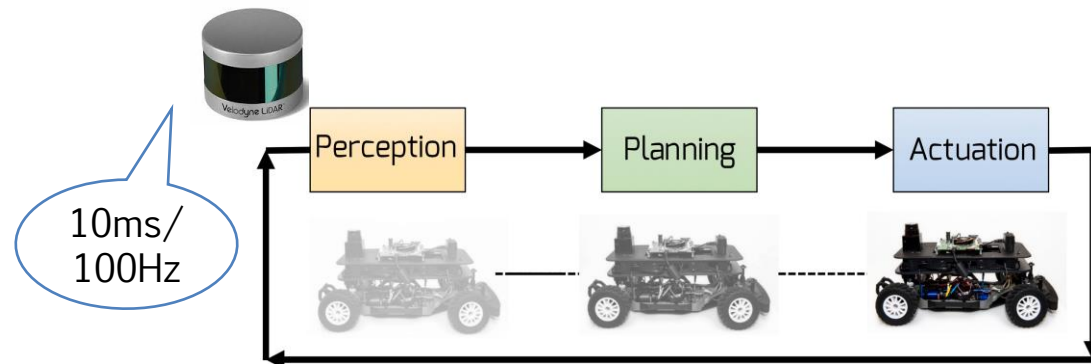
Highest-level: system requirements

Requirements describe of system functionalities

- › Every requirement describes a specific use-case / scenario
- › Functional requirements to describe how the system reacts to input, which output it produces and how its state changes (e.g., a DB)
- › Non-functional requirements describe properties such as performance, reliability, etc

Domain-specific requirements

- › Often, implicit, and the customer won't tell you!
- › Ex: Real-time systems, deal with worst case execution time, not with average case execution time, and it is a **functional** requirement





FURPS model

Requirements are classified in:

- › **Functionality**
- › Usability
- › Reliability
- › Performance
- › Supportability



Communicating the requirements

*"The hardest single part of building a software system
is deciding precisely what to build" (Fred Brooks)*

- › Often, not even customers know what they want

Specifications must be

- › Clear, non-ambiguous
- › Consistent among them
- › Complete
- › Incrementally build with customer



Communicating the requirements

We use formalism

- › E.g., Finite State Machines, Use-case charts, E/R diagrams
- › Might depend on the specific application domain (e.g., FSM are good for RT systems)
- › We'll see them later on....

Specifications can be

- › Operational: what the system shall do
 - "The system must record users' age"
- › Descriptive: how the system is made
 - "We store users' data, such as age, in a DB" – "We expose a web server"
- › Tech: how the system is implemented
 - Typically, due to existing applications/legacy codebase



Software Requirement Specification (SRS)



Software Requirement Specification (SRS)

IEEE 830 standard (circa 1998)

- › Used only for requirements, does not speak of implementation, nor about quality of code

It must be:

- › Correct
- › Unambiguous
- › Complete
- › Consistent
- › Orders requirement by their priority/level of abstraction
- › Verifiable
- › Modifiable
- › Traceable





Software Requirement Specification (SRS)

Correct

- › It correctly models system behaviour

Unambiguous

- › Has the right notations, terminology and Data Dictionary (DD)
- › No ambiguity in language: use **must/not** and **shall/not**
- › Based on the concept of entity, process, behavior

Complete

- › Covers all requirements, functional and non-functional
- › Defines a response for every possible state/input of the system (spoiler: FSMs do)
- › Enhanced with figures



Software Requirement Specification (SRS)

Consistent

- › No conflicts might arise
- › Typically, because multiple contributors to the document don't speak each other!!!
- › Defining a syllabus / DD is the first step

Orders requirement by their priority/level of abstraction

- › Assign an identifier, and a priority
- › Describe the requisite in details
- › E.g., Failure Mode and Effects Analysis (FMEA) for electronic devices

Verifiable

- › Assign metrics and Key Performance Indicators to requirements
- › «Response time under 7 secs [in 90% of cases]»



Software Requirement Specification (SRS)

Modifiable

- › Often, requirements change
- › The document must be well organized, and requirements should not be redundant

Traceable

- › We shall understand where each requirements come from
- › High level (e.g., functional) requirements must be approved by the customer
- › Tech requirements are typically agreed between the dev team



Example of requirements for svc/db modules

› Functional requirement

ID	Description	Parents
USR_568	The user must be able to modify his/her birth date in DB	

› SVC component requirement

ID	Description	Parents
WEBSVC_11	The web services returns 400 if age is null The web services handle null value errors in DB returning 400	REQ_USR_568, ...

› DBMS component requirement

ID	Description	Parents
DBMS_49784	DB shall throw an exception if age is null	REQ_USR_568, ...



Structure of a SRS

1. Introduction

- 1.1 Purpose document
- 1.2 Scope of the document
- 1.3 Definitions, acronyms and abbreviations
- 1.4 [References]
- 1.5 Overview / document structure

2. General description

- 2.1 Product perspective
- 2.2 Product functions
- 2.3 User characteristics
- 2.4 General constraints
- 2.5 Assumptions and dependencies
- 2.6 Apportioning of requirements

3. Requirements

- 3.1 System interface and frontend
- 3.2 Functional requirements
- 3.3 Non-functional requirements

Appendix / [References]

Index



1. Introduction

1.1 Purpose of the document

- › What is our goal (remember: customer might not have a complete idea, so better clarifying everything!)
- › Audience (tech people? Non tech people?)

1.2 Scope of the document

- › Name of the product
- › Goals
- › Benefits
- › What we plan to do/solve
- › What we don't plan to do/solve now
- › What we might plan to do/solve in the future



1. Introduction

1.3 Definitions, acronyms and abbreviations

- › Typically, a list + table for acronyms

1.4 References

- › Can also be in appendix

1.5 Overview / document structure

- › Briefly and concisely, go through every section
- › In case of migrations, typically they are split in *as-is* + *to-be*



2. General description

Main features of our system, and main rules constraints it shall adhere to

- › Not yet the real requirements..

2.1 Product perspective – similar products

- › 2.1.1 System interface
 - Are we using a framework/middleware? Which Web server?
- › 2.1.2 User interface
 - WebApi? WebForm?
- › 2.1.3 Hardware interface
 - How to configure it
- › 2.1.4 Software interface
 - Which libraries to use
 - Interface towards other systems and services (e.g., an existing DB)



2.1 Produce perspective (cont'd)

- › 2.1.5 Communication interface
 - TCP/IP? MQTT? ROS2?
- › 2.1.6 Memory requirements
 - Remember that I work in embedded systems...
- › 2.1.7 Initialization, backup and recovery
 -
- › 2.1.8 Installation and configuration
 - In case of micro-services, for instance, we deploy auth and comm bus first...
 - What is the security level? Do we need to set up a DMZ? An application gateway?



2. General description

2.2 Product functions

- › What do the system do?
- › Not too many details...

2.3 User characteristics

- › Newbie, experts in the domain, tech vs. non-tech
- › Will they require a training to use the system? If so, we must provide a cost entry for this in quotation!

2.4 General constraints

- › Functional and performance, when interacting with other systems
- › Parallel/concurrent system? I.e., a web server
- › Any criticality

2. General description

2.5 Assumptions and dependencies

- › Existing system, knowledge base, licenses

2.6 Apportioning of requirements

- › We are not omniscient! We cannot know everything (especially at this stage 😊)
- › Specify which are these aspects, and **who** shall work on them



3. (Func and non-func) requirements

3.1 System interface / frontend

- › Refines what's in 2.1
- › Input and output data formats, and protocols

3.2 Functional requirements

- › Every requirements has a dedicated table/sheet, or/and a row in a table
- › Intro: specifies the context, the functionality, and the entities/actors involved
- › Input and output
- › Description:
 - Validation /format of data
 - Operations that will be performed
 - What happens in case of errors?
 - What output should we expect?
 - Etc..



3.2 Functional requirement: example

NOTE this is just *a possible representation*. As soon as you are clear, complete, etc, you can devise your own!

USR_568	User account	Update birth date
Input	User ID, the new birth date	
Description	User wants to update the birth date with a new one; the system updates it, and, in case user does not exist, it creates a new user with empty fields	
Error messages	In case birth date is a future date, or it is empty, an error with a meaningful message is shown. The error is shown in user language, as described in requirement LOCAL_45	
Output	Information is updated in the persistent storage; all views that contain the datum are automatically updated, or get the fresh new datum when manually updated.	

› (Possible additional) compact representation

ID	Description	Parents
USR_568	The user must be able to modify his/her birth date in DB	

3.2 Functional requirement: example

NOTE this is just *a possible representation*. As soon as you are clear, complete, etc, you can devise your own!

ID | Area | Short desc

USR_568	User account	Update birth date
Input	User ID, the new birth date	
Description	User wants to update the birth date with a new one; the system updates it, and, in case user does not exist, it creates a new user with empty fields	
Errors	In case birth date is a future date , or it is empty , an error with a meaningful message is shown. The error is shown in user language, as described in requirement LOCAL_45	
Output	Information is updated in the persistent storage ; all views that contain the datum are automatically updated, or get the fresh new datum when manually updated.	

Non-tech language

Explicitly declaring additional (sub) functionalities

Refer to other req

Here, we also assume that other modules/areas have different optional functionalities

› (Possible additional) compact representation

ID	Description	Parents
USR_568	The user must be able to modify his/her birth date in DB	



3.3 Non-functional requirements

- › 3.3.1 Performance requirements
 - How many concurrent accesses?
 - How many users shall we store?
 - How many concurrent asynchronous workflows shall we have in-place?

- › 3.3.2 Database / storage
 - Do we have any tech constraint?

- › 3.3.3 General constraints
 - Shall we adhere to standards?
 - Do we have HW limitations / OS constraint?
 - Shall we use some specific language?
 -



3.3.4 System attributes

Reliability

- › Down time, faults we can/shall/must tolerate...

Accessibility

- › Checkpoints, backup, recovery time under faults...

Security

Maintainability

Portability



3.3.5. Other requirements

Development / staging / production software lines

- › CI/CD pipelines
- › Possible public releases

User groups, visibility rules, access rules

Scalability

- › Performance: “up” and “out” (replicas)
- › Number of users
- › Cost/pricing of scalability



Appendix / index

Everything that is non-essential should go in appendix

- › Ex: we use AWS. It is useful to know its structure and protocol, because we might interact with it. But we shouldn't embed a full guide to AWS in our Section 2!!!
- › Also, typically references are here

Index

- › Index of sections
- › Index of terms..?
- › ...

References



Course website

- › <http://hipert.unimore.it/people/paolob/pub/ProgSW/index.html>

Book

- › I. Sommerville, "Introduzione all ingegneria del software moderna", Pearson
- › Chapter 9 for testing

My contacts

- › paolo.burgio@unimore.it
- › <http://hipert.mat.unimore.it/people/paolob/>
- › <https://github.com/pburgio>