

# Principles of Software Design Requirements

Robert Lukočka

lukotka@dcs.fmph.uniba.sk

www.dcs.fmph.uniba.sk/~lukotka

M-255

# Why do we need requirements

- Input for design and implementation

# Why do we need requirements

- Input for design and implementation
- Choice of the software developer
- Preparing contract
- Estimation, project management
- Verification
- The basis for the documentation / orientation in the project
- Risk management
- ...

All stakeholders work with requirements

# Source of requirements

- Customer

# Source of requirements

- Customer
- Domain knowledge
- Other stakeholders (architectural requirements, implementation requirements, testing requirements, ...)
- Regulations and law
- Deployment conditions
- Organizational structure / Internal regulations of the customer
- ...

# Hierarchy of requirements

- Business requirements

*We want to decrease paper consumption ...*

- User requirements

*After finishing work on ... the following parties will be informed: ...*

- System requirements

*Upon signature the document will be handled to ... using ...*

Traceability - it is important to know the relations between requirements (especially between requirements in different levels).

# Hierarchy of requirements

- Business requirements

*We want to decrease paper consumption ...*

- User requirements

*After finishing work on ... the following parties will be informed: ...*

- System requirements

*Upon signature the document will be handled to ... using ...*

Traceability - it is important to know the relations between requirements (especially between requirements in different levels).

- Which requirements should we derive the contract from?

# Requirement types

- Process requirements
- Product requirements
  - Functional requirements
  - Non-functional requirements



# Requirement types

There are various frameworks to categorize requirements. This is useful so we can be sure that each category is covered.

- [An example](#)

# Properties of good requirements

- Unitary (Cohesive)
- Complete
- Consistent
- Non-Conjugated
- Traceable
- Up to date
- Unambiguous
- Understandable
- Specified priority
- **Testable**

Focus on WHAT, not HOW (however, sometimes, especially when describing a desired process WHAT=HOW)

# Properties of good requirements

- **Complete** (for given subsystem)
  - from all stakeholders
  - no gray areas - includes non-requirements, e.g. what is not part of the system
  - Potentially useful tools: templates ([IEEE830](#)), [classifications](#).

# Examples of requirements and its implications

*We want to decrease the paper consumption ...*

- Business requirement, the source is the customer.
- Should be quantified, e.g. “by ... %”
- Priority?

# Examples of requirements and its implications

*We want to decrease the paper consumption ...*

- Business requirement, the source is the customer.
- Should be quantified, e.g. “by ... %”
- Priority?
- Is it testable?

# Examples of requirements and its implications

*We want to decrease the paper consumption ...*

- Business requirement, the source is the customer.
- Should be quantified, e.g. “by ... %”
- Priority?
- Is it testable? You need to create a system to measure this.  
The quality may depend on the priority.

# Examples of requirements and its implications

*After finishing work on ... the following parties will be informed:*

...

- User requirement - we should know which business requirements it is related with.
- We need a dictionary for the project to be able to specify the type of the document and the parties involved.
- How fast the parties should be notified? Can we measure it?
- Can anything go wrong during the process (of significant importance to be an user requirement)?

# Examples of requirements and its implications

*Upon signature the document will be handled to ... using ....*

- How long should it take?
- What happens if the document will not be signed (+ several other exceptional workflows).



# How to capture the requirements

## How to capture the requirements

- “Victorian novel”
- Using a template
- Spreadsheet / other form of list
- Issue tracking
- Minutes from a meeting (only short time validity, it should be processed into a more robust form) . . .

## Challenges:

- How much documentation we want to create **and maintain**?
- How to preserve traceability?
- How to find related requirements?
- What is the state of the requirement?

# Capturing functional requirements

The functional requirements consist of **use cases**.

# Use case

A full use case contains:

- Actor (human or external system)
- A goal - something of a value to the actor
- A complete sequence of steps how to attain the goal  
**including alternative paths**
- A lot of other stuff

Use cases can be composed, extended and generalized.

# Use case - an example

Wikipedia - use case example

# How to capture functional requirements

Various amount of detail:

- Full use case
- Scenarios - A pass through the use case.
  - We use several scenarios per use case.
- User story - As a <role> I can <capability>, so that <receive benefit>.

# How to capture functional requirements

Various amount of detail:

- Full use case
- Scenarios - A pass through the use case.
  - We use several scenarios per use case.
- User story - As a <role> I can <capability>, so that <receive benefit>.

You may proceed building you system use case by use case or scenario by scenario. You get very different outcomes.

# Requirements workflow

- Stakeholder identification
- Elicitation
- Analysis
- Specification
- Validation

The process is iterative. During each phase a dictionary is created and refined.

# Elicitation

- Discussions
- Focus Groups
- Questionnaires
- Observation
- Studying documentation, legislature ...
- Study of similar products
- ...



# Analysis and specification

- Actor identification and modeling (including new abstract roles)
- Prototyping
- Traceability matrix
- Avoid requirements smells
  - Subjective language
  - Nebulous adjectives, superlatives
  - Negative requirements ( $\neq$  Non-requirements)
  - ...
- Careful write-up helps to identify weak spots
  - Use case analysis -searching for alternative scenarios
  - Generalization

# Common problems capturing the use cases

The structure of a use case is too complex?

- UML activity diagram

The use case is repetitive / too long?

- We can divide it and use use-case composition.
- Creates dependencies between your use cases.







To give an overview of dependencies between actors and use-cases:

- UML use case diagram

# Resources I

- [Wikipedia - Requirement](#)
- [J. Mifsud: Requirements Gathering Part 1](#)
- [Techpedia - Use case](#)
- [Wikipedia - Use case example](#)
- [UML Use Case Diagrams](#)
- [UML Activities](#)

# References |

-  [SWEBOK V3 - Chapter 1](#)
-  [Wikipedia - Requirement](#)
-  [Wikipedia - Requirement analysis](#)
-  [J. Kostičová: Requirements](#)
-  [J. Mifsud: Requirements Gathering](#)
-  [uml-diagrams.org](#)