Software Engineering 2

Requirements Engineering

Dr. Petr SPACEK[1]

[1]petr.spacek@fit.cvut.cz
Department of Software Engineering
Faculty of Informatics
Czech Technical University in Prague

Winter Semester, 2020

Outline

Outline

Connection to the previous course

Software engineering

The 4 main activities of Software engineering:

- Specification
- Development
- Validation
- Evolution & maintenance

Connection to the previous course

Software engineering

The 4 main activities of Software engineering:

- **Specification**
- Development
- Validation
- Evolution & maintenance

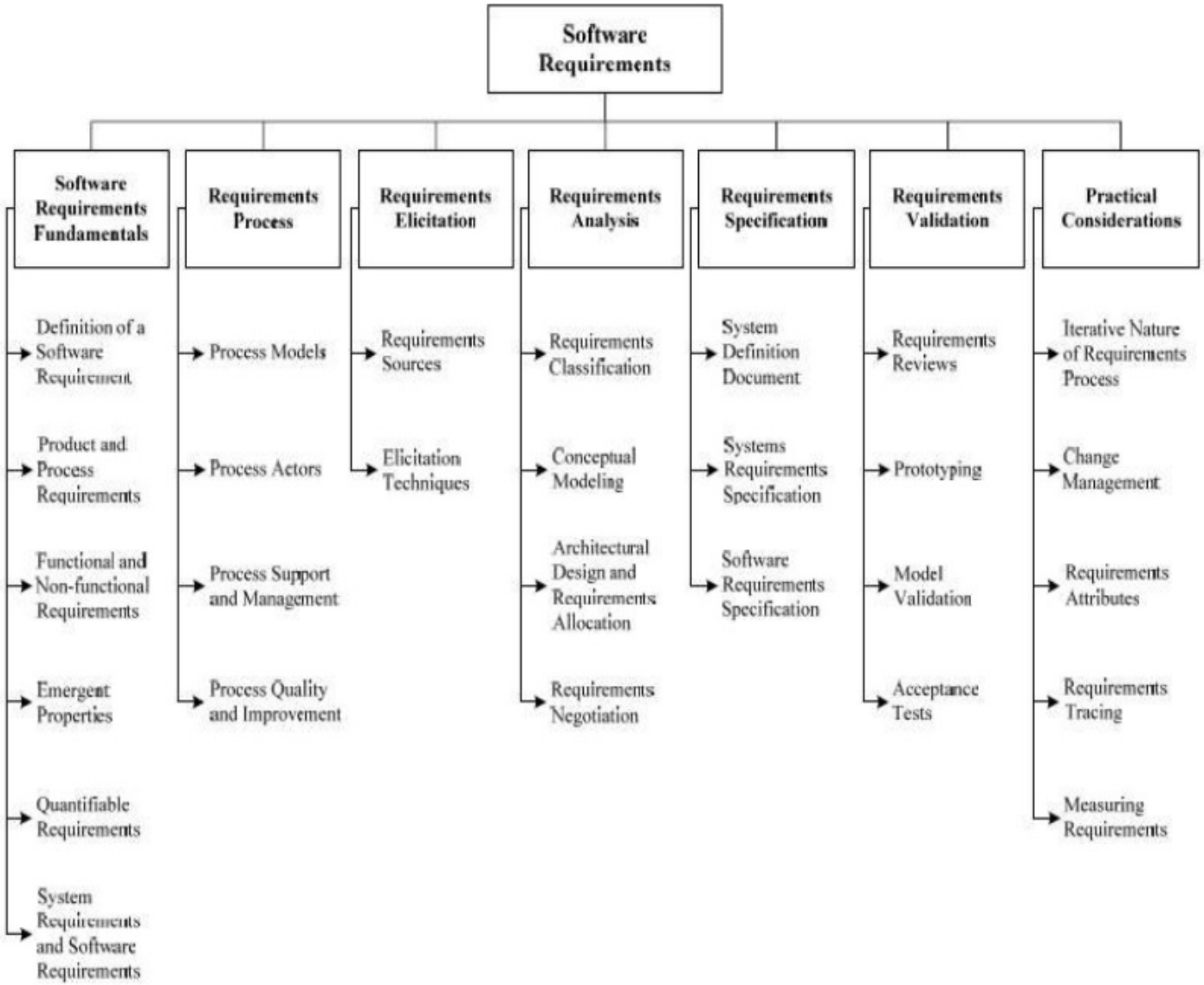
Outline

Book-Definition

Requirements engineering is a systems and software engineering process which covers all of the activities involved in discovering, documenting and maintaining a set of requirements for a computer-based system [1].

Aspects of SW requirements

according to SWEBOK [2]



Schematic view

(Software System) Requirements Engineering

- Elicitation
  (appointments, meetings, documents commenting and versioning, users observation, ...)

Schematic view

(Software System) Requirements Engineering

- Elicitation
  (appointments, meetings, documents commenting and versioning, users observation, ...)
- Analysis
  (thinking, inventing, debates, remarks, ...)

Schematic view

(Software System) Requirements Engineering

- Elicitation
  (appointments, meetings, documents commenting and versioning, users observation, ...)
- Analysis
  (thinking, inventing, debates, remarks, ...)
- Specification
  (decomposition, writing, using the notation, ...)

Schematic view

(Software System) Requirements Engineering

- Elicitation
  (appointments, meetings, documents commenting and versioning, users observation, ...)
- Analysis
  (thinking, inventing, debates, remarks, ...)
- Specification
  (decomposition, writing, using the notation, ...)
- Verification
  (texts reading, appointments, meetings, screenings GUI, scope, argues ...)

Schematic view

(Software System) Requirements Engineering

- Elicitation
  (appointments, meetings, documents commenting and versioning, users observation, ...)
- Analysis
  (thinking, inventing, debates, remarks, ...)
- Specification
  (decomposition, writing, using the notation, ...)
- Verification
  (texts reading, appointments, meetings, screenings GUI, scope, argues ...)

... **all this several times, mixed in time, people and specialties**

Outline

Why do we care?

- Poorly defined requirements cause the failure of projects
  - see article: Statistics over IT projects failure rate.
  - see article: CIO analysis: Why 37 percent of projects fail?

Why do we care?

- Poorly defined requirements cause the failure of projects
  - see article: Statistics over IT projects failure rate.
  - see article: CIO analysis: Why 37 percent of projects fail?
- Specification of requirements
  = instructions for developers

Why do we care?

- Poorly defined requirements cause the failure of projects
  - see article: Statistics over IT projects failure rate.
  - see article: CIO analysis: Why 37 percent of projects fail?
- Specification of requirements
  = instructions for developers
- Specification of requirements
  = contract (a part of) what you deliver $\Rightarrow$ scope

Why do we care?

- Poorly defined requirements cause the failure of projects
  - see article: Statistics over IT projects failure rate.
  - see article: CIO analysis: Why 37 percent of projects fail?
- Specification of requirements
  = instructions for developers
- Specification of requirements
  = contract (a part of) what you deliver $\Rightarrow$ scope
- Specification of requirements
  = basic part of the system's documentation

Why do we care?

- Poorly defined requirements cause the failure of projects
  - see article: Statistics over IT projects failure rate.
  - see article: CIO analysis: Why 37 percent of projects fail?
- Specification of requirements
  = instructions for developers
- Specification of requirements
  = contract (a part of) what you deliver $\Rightarrow$ scope
- Specification of requirements
  = basic part of the system's documentation
- The scope is one of the parameters determining the price of work $\Rightarrow$ over-scoping = low profit

Why do we care?

- Poorly defined requirements cause the failure of projects
  - see article: Statistics over IT projects failure rate.
  - see article: CIO analysis: Why 37 percent of projects fail?
- Specification of requirements
  = instructions for developers
- Specification of requirements
  = contract (a part of) what you deliver $\Rightarrow$ scope
- Specification of requirements
  = basic part of the system's documentation
- The scope is one of the parameters determining the price of work $\Rightarrow$ over-scoping = low profit

Outline

Project Scope

Definition

The work that needs to be accomplished to deliver a product, service, or result with the specified features and functions.

Project Scope

Definition

The work that needs to be accomplished to deliver a product, service, or result with the specified features and functions.

$$\text{scope} \Rightarrow \text{all your commitments}$$

- system functions, features, ...
- methodology, documents, integration with other systems, ...
- expectations regarding performance, security, ...

Project Scope

Definition

The work that needs to be accomplished to deliver a product, service, or result with the specified features and functions.

$$\text{scope} \Rightarrow \text{all your commitments}$$

- the customer wants everything what isn't explicitly excluded
- the contractor is doing only what is explicitly included
- $\Rightarrow$ carefully with the *grey* zone !!

Kinds of requirements

- it is important to think of all requirements kinds
- it is important to communicate with all stakeholders

At least the following:

- functional requirements
- interfaces reqs. UI, SW, HW, communication, ...
- non-functional reqs. performance, security, reliability, availability, scalability, ...
- internationalization reqs. legislative, multilingual ...
- accessibility reqs. blinds, deafs, ...

$\Rightarrow$ use the standard templates for software specification

Requirements on requirements

acording to IEEE

- Correct
- Unambiguous
- Complete
- Consistent
- Ranked
- Verifiable
- Modifiable
- Traceable

1. **Correct**: The SRS, or software requirements specification, should correctly describe the system behavior. It is not productive to have a requirements document that describes implausible or impossible expected system behavior or user goals.

2. **Unambiguous**: Software requirements should be written in such a manner as they are not subject to different interpretations. The use of specific and appropriate language can help avoid ambiguity in interpretation.

3. **Complete**: the software requirements document should completely describe the system's expected behaviors and feature set.

4. **Consistent**: Requirements for the system under discussion must not contradict each other.

5. **Ranked**: You must rank your software requirements for importance. Each software requirement has its own level of importance and criticality, and they are not all equal. By ranking the requirements, software designers ensure that guidance is given to the development team regarding effective prioritization.

6. **Verifiable**: If the requirement cannot be verified as having been met, then the requirement itself is written poorly. The requirements have to be testable.

7. **Modifiable**: The requirements must be easy to modify or change.

8. **Traceable**: The requirements must be traceable, and it is essential that traceability information has been provided, as the requirements document provides the starting point in the traceability chain. I have written elsewhere in this blog at length about the importance of software requirements traceability and have provided examples of software requirement traceability matrixes. Many software development organizations use proprietary CASE software tools and other methods to enforce traceability policies that stipulate how much traceability information regarding requirements must be maintained.

http://tinyurl.com/bmnnoyt

Outline

The fundamental questions that we pose

page I

- What should be the result of analysis?
- What is the content of the output of analysis?
- What is the form of the output of analysis?
- The logical and physical decomposition?
- The level of detail?
- What about the labour-intensity of analysis?
- What about the duration of analysis?
- What about the number of people involved in analysis?
- How to decompose the work between people working in parallel?
- When is the right time to begin with architecture design?
- When the engineer can already start?
- How should the analysis / specifications spread at the time from writing the offer to maintenance of the system?
- What are the differences between the primary specification and the changes specifications (during maintenance?)
- What is the relationship between the specification and architecture (what vs. how)?

The fundamental questions that we pose
page II

- How do I know that the customer provide the right people?
- What qualities should a good analyst has?
- How do I verify that the specification is a specification of what is really wanted?
- It is reasonable to fear to ask?
- It is wise to let approve something of what you are not truly conviced?
- How do I know that my specification is useful?
- What to do with analysts after the analysis?
- How to keep the results of the analysis?
- What is different about the analysis in the development and maintenance?
- Can we skip the analysis?
- How find and recognize the proper boundary conditions?
- How will the requirements change and how to prepare for that?
- How to detect the increase in the scale?
- The phenomenon of "gold-plating"?

And many others ... ...

Outline

Specification format

possible different approaches:

- a structured document - accompanied with diagrams, pictures, models, ...

Specification format

possible different approaches:

- a structured document - accompanied with diagrams, pictures, models, ...
- the "Victorian amendment"
  - a lot of text
  - minimal structure

Specification format

possible different approaches:

- a structured document - accompanied with diagrams, pictures, models, ...
- the "Victorian amendment"
  - a lot of text
  - minimal structure
- a catalog of requirements - without deeper structures
  - frequent
  - not optimal

Specification format

possible different approaches:

- a structured document - accompanied with diagrams, pictures, models, ...
- the "Victorian amendment"
  - a lot of text
  - minimal structure
- a catalog of requirements - without deeper structures
  - frequent
  - not optimal
- specification of requirements in CASE tools

  (Enterprise Architect, Visual Paradigm, MS Visio, ...)
  - significantly more work
  - clear definition of the specification-creating rules
  - when the rules are respected = complete description of the system
  - very complex and difficult to maintain for larger systems

Outline

GUI Model

another interesting topic

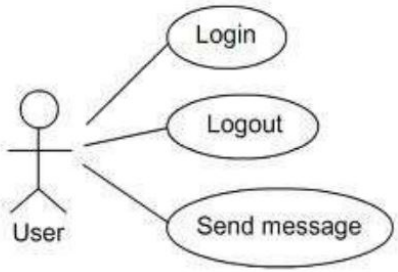
- almost everything works!!!
  - PowerPoint
  - Excel
  - HTML
  - Special languages for modeling GUI
- being able to reuse the GUI model = big advantage
- not needed in all cases (CMD, services, ...)
- aids in system understanding = it's visual
- enables partial verification of reqs. by client

Modeling

another interesting topic

UML

- a tool for representing the developed SW
  - analysis stage
  - design stage
  - and sometimes even implementation stage
- the need to know it (problem for clients)
- standardized = advantage
- USE-CASES
  - scenarios
  - cannot be used as the base for specification
  - sometimes - an escape from complications



Business vs. Requirements analysis

another interesting topic

- these disciplines have a large overlap = often mixed
- Business analysis focuses on identifying changes necessary for achieving the strategic objectives of the organization.
- It is about changes in:
  - strategy
  - organization structure
  - politics
  - processes & data flows
- many many many methodologies: ☺

  PESTLE, HEPTALYSIS, MOST, SWOT, CATWOE, MoSCoW, Five Why's, VPEC-T, ...

Outline

The practice

... the real-life experience

- labor intensity: 10 - 30%
- the work involved in maintenance: 20%
- personal qualities of participants
- the form cannot outshine the content
- the content must be "comprehensive"
- a sufficient description of the scope
- maintainability is the essential thing
- keep principles pure
- distribute expenses according to labor intensity
- be brave
- the GUI model works
- a structured text works
- the template and checklist works
- all types of requirements are handy
- the measured data are needed (screens, ...)
- life will promote what is really needed (the question of how painfully)
- dealing with changing requirements
  - is it change or misunderstanding?
  - is it change or greater level of details?
  - positive or negative definition?
  - scope-fights

Outline

Impasses

- skipping analysis
- not thinking about the architecture
- doing only a catalog or use cases
- ignore other than functional requirements
- not knowing about who is really important stakeholder
- not believing in your own prudence and intuition
- not asking about the things

  "you don't want to hear"

... and many others ...

Outline

1 Intro
- Connection to the previous course

2 Requirements Engineering
- What it is?
- Why do we care?
- Basic terms
- The fundamental questions
- Specification format
- Other interesting topics
- The practice
- Impasses

3 Conclusion
- Summary
- Literature

Requirements engineering sets the scope of yout project

scope $\Rightarrow$ all your commitments

many kinds of requirements $\Rightarrow$ be precise, well-structured & formatted

Outline

1 Intro
- Connection to the previous course

2 Requirements Engineering
- What it is?
- Why do we care?
- Basic terms
- The fundamental questions
- Specification format
- Other interesting topics
- The practice
- Impasses

3 Conclusion
- Summary
- Literature

For Further Reading I


Kotonya G. and Sommerville, I.
Requirements Engineering: Processes and Techniques.
Chichester, UK: John Wiley & Sons. 1998.


Abran, Alain and Bourque, Pierre and Dupuis, Robert and Moore, James W.
Guide to the Software Engineering Body of Knowledge - SWEBOK.
Piscataway, NJ, USA: IEEE Press. 2001.