Software Engineering 1

Requirements Engineering

Arkadiusz Chrobot

Department of Computer Science, Kielce University of Technology

Kielce, November 16, 2025

Notes

Outline

Motto

Introduction

Requirements Elicitation And Analysis
- User Stories
- Scenarios
- Ethnography And Prototyping

Requirements Validation

Requirements Management

Notes

Motto

"The marketing division of the Sirius Cybernetics Corporation defines a robot as «Your Plastic Pal Who's Fun to Be With». The Hitchhiker's Guide to the Galaxy defines the marketing division of the Sirius Cybernetic Corporation as «a bunch of mindless jerks who'll be the first against the wall when the revolution comes»…"

— Douglas Adams, *The Hitchhiker's Guide to the Galaxy*

Notes

Definitions

Definition (Requrements Engineering)

Requirements Engineering (RE) is an early activity in Software Engineering, concerned with elicitation, analysing, validation and management of software requirements.

Definition (Requirement)

Requirement is a statement that identifies a product or process operational, functional, or design characteristic or constraint, which is unambiguous, testable or measurable, and necessary for product or process acceptability (by consumer or internal quality assurance guidelines). [1]

Definition (Stakeholder)

Stakeholder is an individual, group of people, organization or other entity that has a direct or indirect interest (or stake) in the software. [1]

Notes

Sources of Requirements

There are several sources of requirements:

domain is the target area of the software,

regulations can be either internal or external (i.e. enforced by the law),

stakeholders (including software developers) are the main source of requirements.

Notes

Requirements Classification

There are many way of classifying requirements. The most basic classification is:

Functional Requirements are these requirements, that define what kind of functions or services the software should provide (in other words *what* it should do).

Nonfunctional Requiremntes define constraints for the solution (in other words they describe *how* the software should perform its functions). Nonfunctional requirements directly influence the software quality.

Functional Requirement

The multimedia system should allow the user to set the brightness of the screen (scale: 1–100).

Nonfunctional Requirement

The reaction time of Anti-Lock Braking System should be less than 200 ms.

Notes

Requirements Classification

Requirements may be classified according to how detailed they are:

Business Requirements are the most important and generic (high-level) software requirements. Their source is the domain of the software and they change rarely.

User Requirements define the features and services of the software that are necessary for its users. They are more detailed than the business requirements.

System Requirements are the most low-level software requirements. They specify details necessary for the stakeholders to understand and approve what the software should provide.

Notes

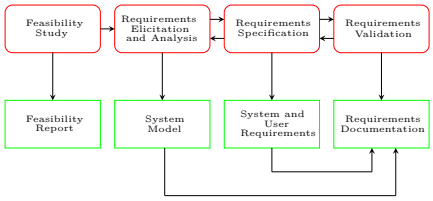
Complexity of Requirements Engineering

The Requirements Engineering is difficult because [2]:

- Stakeholders often don't know what the software should do.
- Stakeholders may have unrealistic expectations.
- Stakeholders define the requirements in the language of their domain, which may have some hidden meanings or cannot be immediately understandable.
- Different stakeholders may have different expectations, resulting in conflicts of the requirements.

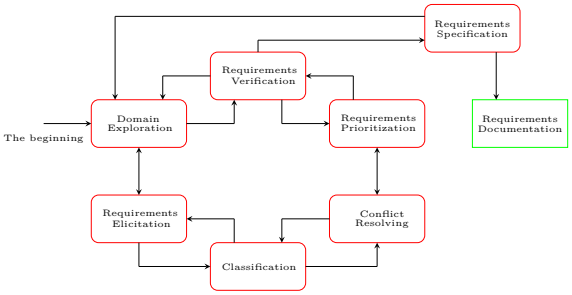
Notes

Overall Requirements Engineering Process



Notes

Elicitation And Analysis of Requirements



Notes

User Stories

User Stories are the main tool for requirements Elicitation and Analysis in agile methods. However, this process starts with an *Initiative* (akin to the Product Goal), which is later broken down into *Epics* (big User Stories) that are then simplified to *User Stories*. These stories are source of *Tasks* that should be performed in order to implement User Stories [3].
Formally, a User Story captures a description of a software feature from an end-user perspective [4]. However, the User Story is not the whole requirement. It is just an entry point for discussing with the Product Owner (or similar representative of stakeholders) the details of requirements, and for specifying the *Acceptance Criteria* that allow the Developers and/or Stakeholders to check if the requirement was implemented correctly.
The next slide summarizes the entire description of a single requirement as defined in agile methods. The User Stories originate from Extreme Programming, but may be applied in Scrum and other agile approaches.

Notes

User Stories

Card	Conversation	Confirmation
User Stories are written down on cards and may contain estimations, notes etc.	The details of a User Story are discovered and specified in a conversation with the Product Owner.	Acceptance Tests verify the implementation.
Requirement		

Notes

User Story

User Story Structure

As a <specific user, Persona, role>, **I want** <a need>, **so that** <problem to solve, goal to achieve>.
Acceptance Criteria/Conditions of Satisfaction

User Story Alternative Structure (5W)

As a <who> <when> <where>, **I want** <what>, **so that** <why>.
Acceptance Criteria/Conditions of Satisfaction

User Story Alternative Structure (BDD)

As a <who> <when> <where>, **I want** <what>, **so that** <why>.
Scenario 1: <name of the scenario>
Given <first initial condition>
(**And** <second initial condition>)
When <scenario trigger>
Then <expected result>

Notes

Scenarios

Scenarios are another tool for requirements Elicitation and Analysis in both traditional and agile software development methods. The scenario describes a user interaction with the software. Note, that the user is not necessary a human. It can be an external system or organization. The scenario has to meet several conditions:

1. It must specify the state of the system at the beginning of the scenario.
2. It has to describe the normal flow of events in the scenario.
3. It has to identify exceptions and how to handle them.
4. It should contain the information about other activities that may happen at the same time.
5. It must describe the state of the system after the scenario is finished.

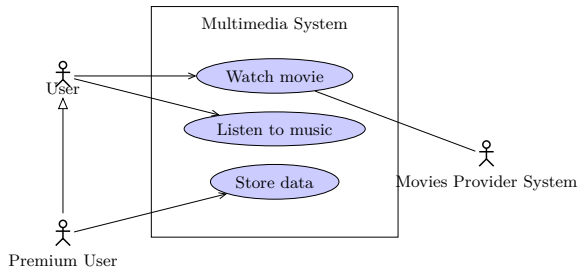
Notes

Use Cases

In traditional software development methods the scenarios may be expressed in a form of UML *Use Case Diagrams*. An example of such a diagram is shown in the next slide. The stick figures are *actors*. An *actor* represents a role that include human users, other software, hardware or other systems. The ellipses are *use cases* which define the interaction with software or the services that the software provides. Each use case represents rather a collection of scenarios than a single scenario. Some of them describe the alternative ways of preforming the same use case. The use case diagrams are not enough to describe scenarios so other UML diagrams have to be used to specify their details, like the *sequence diagram* presented in the first after the next slide.

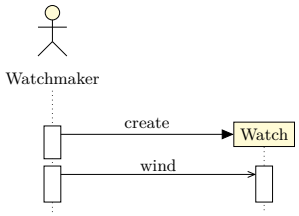
Notes

Use Cases



Notes

Use Cases



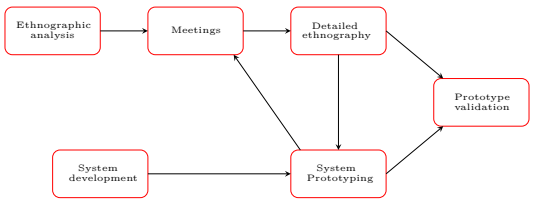
Notes

Ethnography And Prototyping

Prototyping is used both in agile and traditional software development methods to tackle the analysis of complex and ambiguous requirements. It involves preparing and delivering to the users a prototype version of the software that implements only the features that are defined by such requirements. This allows the developers to get feedback information on how well the requirements have been recognized.
The prototyping can be combined with the ethnography. In the software engineering ethnography means observing the work customs of people who will be the users of the developed software. An analyst who performs the ethnography may discover patterns of work carried out by future software users that are not described in formal documents and this can have impact on how the software prototype should work. Introduction of the prototype to the work environment of these people can have further consequences that again may be discovered by the analyst. After several such iterations a balance should be reached, that allows the developers to finish analysing the examined requirements. Please note, that the ethnography cannot be applied to elicitation and analysis of formally defined requirements.

Notes

Ethnography And Prototyping



Notes

Requirements Validation

According to the Chaos Report, issues with requirements are the most common cause of software projects failures. A mistake in requirements is more expensive to correct that the bug in the code. It is then crucial to make sure that the requirements are:

- clearly defined,
- really important,
- complete,
- feasible,
- verifiable.

This is the responsibility of the *requirements validation* process. There are several methods that can be used to reach its objective:

- Requirements Reviews,
- Prototyping,
- Acceptance Testing,
- Automated Validation (if Formal Methods are used).

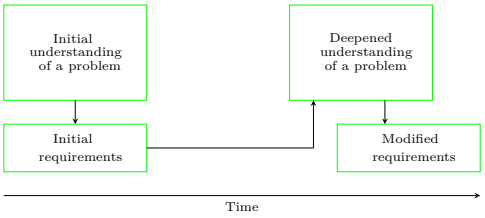
Notes

Requirements Management

As the software project progresses the requirements may, and usually change. Not all of them alternate at the same rate. These of them that steam directly from the domain are less likely to change. Others, that are dependent on external factors, like the law regulations are more fragile. The most important cause of the changes is the knowledge about the product that the software engineers gather as the time progresses (see next slide).

Notes

Requirements Management



Notes

Requirements Management

To manage the changes in requirements software engineers needs to have a method of *tracking* them and a procedure to *manage* them. Tracking the changes in requirements is relatively easy in agile methods, with short iteration time and rich feedback information. In traditional methods it may involve additional effort. However, in both cases it requires knowing what is the origin of the requirements, how they are related to each other and how their change will impact the design of the software.

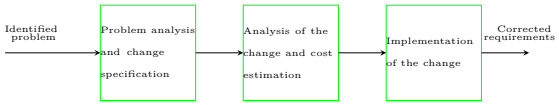
Notes

Requirements Management

The schematics of the procedure for requirement change handling is given in the next slide.

Notes

Requirements Management



Notes

Bibliography

- Jeremy Dick, Elizabeth Hull, and Ken Jackson. *Requirements Engineering*. Cham, Switzerland: Springer, 2017.
- Gerard O'Regan. *Concise Guide to Software Engineering*. Cham, Switzerland: Springer, 2017.
- *User stories with examples and a template*. 2023. URL: https://www.atlassian.com/agile/project-management/user-stories.
- *What is User Story?* 2023. URL: https://www.visual-paradigm.com/guide/agile-software-development/what-is-user-story/.

Notes

Questions

?

Notes

The End

Thank You for Your attention!

Notes