Software Engineering Summary

Moritz Gerhardt

Inhaltsverzeichnis

1 General

1.1 What is Software?

Software can describe a lot of things, some examples include:

- Executable programs
- Configuration files
- System documentation
- User documentation
- Support environment
- etc.

In general Software can be divided into three categories:

- Application Software
  - Interacts directly with the end user
  - General purpose software (To be used in other applications: Word processing, image processing, etc.)
  - Customized Software (Software specifically for a specific purpose: CAD, IDE, BIM, etc.)
- System Level Software
  - Does not interact directly with the end user
  - Responsible for keeping systems running (Operating System, firmware, drivers, etc.)
- Software as a Service (SaaS)
  - Runs on a server
  - Indirectly accessed via client (browser, remote shell, etc.)

Furthermore, some characteristics of Software include:

- Software does not wear out, its environment does
  - Software is subject to continous change in hardware, needs to be able to adapt
  - Software should be able to support new requirements, use cases etc.
- Software often lives longer than anticipanted
  - Almost impossible to know use cases in advance as it can be in use for years or even decades (Excel used in biology geneology $\rightarrow$ lead to unexpected behaviour)
- Software properties are hard to measure
  - How does the code relate to software quality?
  - How do we measure progress?
  - How do we measure resilience?

1.2 Software Engineering

Typically a software is designed to solve different needs of different groups involved in the development of the software.

- Customer / Client
  - Often the person / organisation that'll pay for the development
  - Sets a budget, timeframe, requirements etc.
  - → Requirements analysis
- User
  - Usually the person / organisation that'll use the software
  - Defines what the software is used for and subsequently what requirements this sets
  - → Use Case Analysis
- Manufacturer
  - Usually the person / organisation that'll design and develop the software
  - Is concerned with how to build the software in a way that satisfies the customers and users
  - → Domain Modelling, Architecture, Quality Assurance, Design Practice, Verification
- Maintainer
  - Usually the person / organisation that'll maintain the software during its lifetime
  - Responsible for maintenance of the software and updates to make it usuable for new demands and requirements
  - → Maintenance and Evolution

After all these aspects are consideres the software system is the built with the specific requirements in budget and time.

There are quite a few problems that can happen with Software:

- Unexpected Errors:
  - Few errors are obvious
  - Most of them are near impossible to test for and detect (Algorithmic error, arithmetic overflow)
  - Often go undetected for a long time as they're usually the result of very specifc inputs for complex computations
- Altough errors can occur, as long as they do no violate the requirements they are not considered errors:
  - INABIAF: It's not a bug, it's a feature
- Most errors are caused by missing verification, validation or documentation.
  - This usually indicates an insufficient match between requirements and implementation

Errors can also occur as a result of social aspects:

- Insufficient validation
- Inadequete Specification
- Constantly changing requirements
- Insuffciently trained software engineers
- Managment with lacking grasp on software development
- Unsuitable methods, languages, tools etc.

2 Requirements Engineering

In the following we are gonna look at requirements engineering using the case study of a car sharing service. The main roles and functionalities of a car sharing service are:

Main Roles & Functionalities

- Role-Independent
  - Authentication
- Administrator
  - Add / change new cars, rental locations
  - Biling
- User
  - Check availability
  - Request booking
  - Change booking
- Service Staff
  - Take out vehicle for service

Requirements Analysis is concerned with building a system of what the product *needs* to fulfill in terms of budget, time and surrounding criteria.

So the objectives are akin to:

What has to be developed?

- Need to understand the problems that arise in the requirement elicitation phase
- The different kinds of requirements
- The requirement engineering workflow
- Modelling requirements
  - Scenarios & Use cases
  - Notations: Textual and Graphical

Although the objectives seem pretty straightforward, requirements analysis can be tricky due to how ambigous language can be. Thorough communication is important to understand fully understand what the client wants.

What is Requirements Engineering?

The process of
- finding
- analysing
- documenting
- validating

software requirements.

2.1 What are Requirements?

Definition

- Requirements as descriptions of the services provided by the system
  - Car booking
  - Service booking
  - Location tracking
  - etc.
- Requirements as the operational constraints of the system
  - Database throughput
  - System memory
  - Navigation systems
  - etc.

These requirements are usually handled in the form of **System Requirements Specification (SRS)** Documents (Ger: Pflichtenheft) or **User stories**, structured natural language of use cases, state diagrams etc. stored in the product backlog (ordered list of requirements)

2.2 Different Types of Requirements

Overall the requirements can be divided in the following:

- User Requirements
- System Requirements
- Functional Requirements
- Non-Functional Requirements
- Domain Requirements

2.2.1 User Requirements

State in language or diagrams:
- What services the system should provide
- What the operational constraints are

The descriptions are often high-level and abstract.

For example ”According to german law, a car sharing service must keep track of all bookings”

2.2.2 System Requirements

Precise and detailed specification of the systems
- functions
- services
- operational constraints

For example: ”After a successful booking the user must be shown an overview of their booking”

”Booking details must be stored for 10 years”

Characteristics

- Refinement of user Requirements
- Determine system interface (functional)
- Recorded as part of the SRS and part of the contract with the client
- Authored by software developer or business analyst with the client

2.2.3 Functional Requirements

Functionality that is clearly identifiable and localized in the code

- Services providede by the system
- System reactions to inputs or events
- System behaviour in specific situations like Network distruption

2.2.4 Non-Functional Requirements (NFR)

Constraints of the services or functions

- Service Level Agreement (SLA)
- Constraints from development process
- Alignment to standards (e.g. Protocols)

NFRs often apply to the whole system as they cannot be handled by simply adding a peice of code.

For example: ”The database must be able to process 1000 queries a second” ”User data must only be accessible to authorized persons”

Examples of Non-Functional Requirements

- Product requirements
  - Reliability (crashes, use cases)
  - Efficiency (performance, memory)
  - Portability (Not confined to one device or service)
- Organisational Requirements
  - Delivery mode (beta, continous)
  - Implementation (Programming language, framework)
  - Standardization (ISO standards or similar)
- External requirements
  - Interoperability (TUCaN $\Leftrightarrow$ Moodle)
  - Ethical aspects
  - Legal aspects (safety, security, privacy)

NFRs may often result in the identification of functional requirements and are often more important to adhere to strictly than individual functional requirements.

A problem with NFRs come from how subjective they are: What is ethical, what is ease of use, what is good performance etc.?

2.2.5 Domain Requirements

Are derived from the application domain rather than the needs of the user

- Often expressed in domain specific language → Hard to understand for software engineers.
- For example: Software engineers usually do not have profound knowledge of chemistry, however the client might be a chemist and needs software that can be used for very specific applications.
- Often implicitly assumed as obvious to domain experts
- Can be functional or non-functional

2.3 Feasibility Study

The objective of the Feasibility Study is to obtain a justified understanding of whether the requirements engineering and system development phases should be **started**. This is usually base on:

- Business requirements
- Outline description of the system
- Description of how the system should support the business

The resulting **Feasibility Report** then covers

- Whether the system contributes to the objective of the organization
- Whether the system can be implemented within technical, financial and schedule constraints
- Whether the system can be implemented using other systems used by the company

2.4 Requirements Elicitation and Analysis

2.4.1 Requirements Discovery

Systematic Requirement Discovery
Viewpoint-Oriented Approach

- Interactor Viewpoint
  - People or systems who interact directly with the system
  - End Users, Administrators, Service Personal, etc.
  - **Direct Stakeholders**
- Indirect Viewpoint
  - Stakeholders who influence the requirements, but won't use the system directly
  - CFO, Data protection personal, etc.
  - **Indirect Stakeholders**
- Domain Viewpoint
  - Domain characteristics & constraints that influence the requirements
  - Legal, Ethical, etc.

The goal of the requirement elicitation process is to develop more specific viewpoints and use them to discover more specific requirements.

The elicitation can be done in an interview which are usually structured as follows:

Systematic Requirement Discovery
Interviews

- Closed Interviews:
  - Predefined questions
- Open Interviews:
  - No predefined agenda
- Interviews should only be used as a supplement:
  - Interviewee can be biased
  - Interviewee can assume domain knowledge

Some further elicitation techniques are:

Systematic Requirement Discovery
Other Techniques

- Scenario Analysis
  - Analyses the sequence of interactions with the system
- Use Case Analysis
  - Analyses the use cases of the system

2.4.2 Requirements Classification & Organisation

For further structured workflow the requirements should be categorized, this can be done using the **FURPS+** Model:

- **F**unction
- **U**se
- **R**equirements
- **P**riority
- **S**cope
- +
  - Implementation
  - interface
  - Operations
  - Packaging
  - Legal

2.4.3 Requirements prioritisation & Negotiation

Another problem in the elicitation process are conflicts. Different stakeholders might have different requirements. These conflicts need to be resolves through negotiation.

2.4.4 Requirements Documentation

The produced requirements are then documented and used as a basis for further elicitation and analysis. These documents (SRS) can be formal or informal.

SRS Target Groups

- Client, users
- Managers: Client and Manufacturer
- System Engineers, system testers, system maintainers
- Anyone concerned with ordering, using, manufacturing or maintaining

The level of detail of the SRS depends on the system, development process, whether the product is developed in-house or external etc.

The usual format of an SRS is:

System Requirement Specification (SRS) Document Format

1. Introduction
   (a) Purpose of the SRS
   (b) Scope of the product (Also what isn't in the scope)
   (c) Glossary
   (d) References
   (e) Overview
2. General Description
   (a) Product perspective
   (b) Product functions
   (c) User characteristics
   (d) Limitations
   (e) Assumptions and dependencies
3. Specific Requirements
4. Appendices, Index, etc.

2.4.5 Requirements Validation

Requirement Validation Checklist

- Validity
  - Do the requirements capture the needed features?
  - Is additional functionality needed?
- Consistency
  - Are the requirments conflicting?
- Completeness
  - Do the requirements cover all the features and constraints?
- Realism
  - Can the requirements be implemented feasably?
- Verifiability
  - Is there criteria to check whether the requirements are met?
- Traceability
  - Is each requirement tracable to the source of the requirement?

3 Use Case Analysis

3.1 Client Side Involvement

To identify all and good use cases, it's imperative to involve the users. This is usually very expensive: Around 30-50% of development costs are allocated towards requirements and use case analysis and validation.

Use cases usually are text stories used to discover and record requirements. These use cases complement requirments analysis and provide operational requirements as a basis for system design. They do not replace requirement analysis as they do not capture non-functional requirements.

Definitions of Constituents of Use Cases

- Actor
  - Someone or something with behaviour (person, computer system, organisation, etc.)
  - **Primary Actor:** The person who initiates the use case (requests a service)
- Scenario (Use Case Instance)
  - Specific sequence of actions and interactions between actors and system
  - One particular story using a system
- Use Case
  - Collection of related success and failure scenarios
  - Describe an actors usage of a system to achieve a goal

Different Kinds of Use Cases

- White Box vs. Black Box: With whom does interaction occur?
  - White Box (Transparent): Use cases provide details on internal interaction with the system
  - Black Box: Use cases describe only interactions with external actors
- Corporate vs. System
  - Corporate: Use cases describe business process
    (Usually white box)
  - System: Use cases are described with respect to the system
    (Usually black box)

Use Case Formats

- Brief: Short, one paragraph summary. Usually outlines main succes scenario
- Casual: Informal, Multiple paragraphs that cover multiple scencarios
- Fully Dressed: All steps and variations in detail. Includes supporting sections on preconditions, success guarantees etc.

Should be precise (detailed) and accurate (correct).

Fully Dressed Use Case Template

- Use Case Name
  - Start with a verb (”Accomplish this task”)
- Scope
  - Corporate, system (name), subsystem
  - Design Scope: Boundaries of the system of the use case (whole corporation, (sub-)system name)
  - Function Scope: Limits functionality to be realized. Managed by a list of functions in and out of scope
- Level
  - User goal, summary goal, subfunction
  - User goal: Most important goal of the user
  - Summary goal
    - Multiple User Goals: Describe context of system
    - Life cycle sequence of related goals
    - Table of content for lower-level use cases
  - Subfunction: Use case that is part of user goal. Singled out on a by-need basis, reusable in multiple goals
- Primary Actor: Initiates use case
- Stakeholders and Interests: Who is interested in this and what do they want?
- Preconditions
  - What must be true or worth telling
  - Enforced by system and known to be true
  - Will not be checked again during execution
- Minimal Guarantee
  - Fewest promises the system makes to Stakeholders
  - Especially if the primary actors goal cannot be achieved
  - (MVP) Minimal Viable Product
- Success Guarantees
  - What must be true on successfull completion
  - States the satisfied interests of the stakeholders after successful completion
- Main Success Scenario
  - Representative Scenario of successful execution
  - Numbered list of steps executed
  - Each step may reference a sub use case
  - First step specifies trigger of use case
- Extensions
  - Alternative scenarios of success or failure
  - Refer to main success scenarios step, by explaining alternative scenario for each step as well as the condition or failure needed for the alternative
- Special Requirements: Related non-functional requirements
- Technology and Data Variation: Needed / Used Technology and Data formats
- Frequency of Occurence: How often does the use case occur?
- Miscellaneous: For example: open issues

For developing use cases one should proceed incrementally. Meaning that first relevant use cases should be accurately identified as a high level and then add precision gradually.

Recommended Workflow and Tips

1. List supported actors and goals
   Review list for accuracy and completeness
2. Write stakeholders, triggers and main success scenario for each use case
   Validate that the system delivers to important stakeholders
3. Identify and list failure conditions
4. Write Failure handeling

- Start simple and focus on intent
- Write black box use cases
- Focus on actors and users of a system and their goals

A well defined task in general should fulfill the following requirements:

- performed by one stakeholder in one place at one time
- model a business event
- add measurable business value
- leaves data in a consistent state
- be more than a single step

This is called the **Elementary Business Process (EBP)**.

3.2 UML Use Case Diagrams

The Unified Modeling Language is a visual, precise design notation for software development.

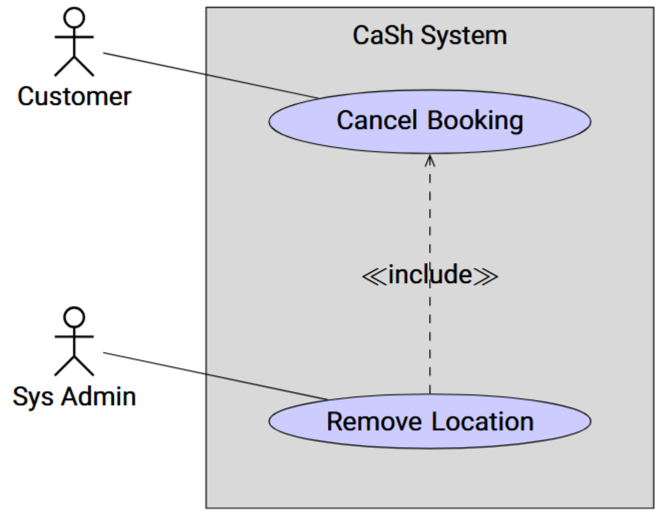
UML Use Case Diagrams Remarks

- UMLUCD is intentionally minimalist
- UMLUCD are an organizational method to improve communication and comprehension of use cases and to reduce text duplication
- UMLUCD provide a black-box view on system software
- Are only useful for early phases of use case analysis →not suitable for fully dressed use cases

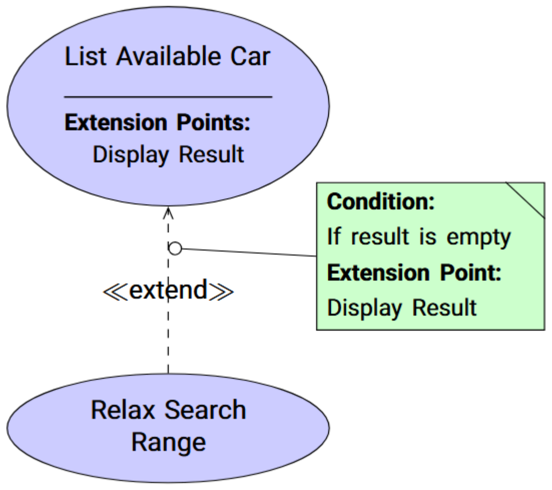
These diagrams essentially consist of system boundary (scope of the system), actors and use cases, as well as their relations.

UML Use Case Diagrams Relations

- «include»:
  - Factors out common behaviour between use cases into sub-function
  - Facilitates decomposition of large use cases and enables reuse
  - Included use cases are *always* executed
  - (Arrow goes from sub-function to base use case)
- «extend»:
  - Describes where and under what condition an extending use case extends the behaviour of the base use case
  - Most extensions do not qualify as seperate use cases →Should only be used when really justified



UML Include



UML Extend