Software Engineering Lecture Notes WS 24/25

Igor Dimitrov

2024-10-14

Table of contents

Preface

Lecture notes for the course “Software Engineering” at Heidelberg University WS24/25.

1 Introduction

1.1 Chapters

1. Introduction
2. Communication in a Project link
3. Requirements Engineering (Communication with the Users) link
4. Design (Communication with Developers) link
5. Quality Management link
6. Evolution link
7. SWE-Process (Summary and Project Management) link

2 Communication in a Project

Following topics relate to and determine communication within a project:

1. Number of participants and their roles in the project
2. Type of the contractual relationship
3. Team Organization: The way developers communicate within the project
4. Collaborative Coding

2.0.1 Projects and Participants

terms relating to project and process:

- process:
- project
- process model:

characteristics of a project:

- limited time
- creator
- purpose
- client
- results
- means and tools
- organization and planning

participants:

- client
- user
- manufacturer

2.0.2 Contractual Relationship

project types:

- EP (Entwicklungsprojekt) -> development project
- AP (Auftragsprojekt) -> Commissioned Project
- EDP (EDV-Projekt, EDV = Elektronische Datenverarbeitung) -> IT Project
- SP (Systemprojekt) -> System Project

2.0.3 Team Organization

types of team organization:

- single person
- 2-person team
- anarchic team
- democratic team
- hierarchical team
- chief-programmer team
- agile team

types of secondary organization:

- functional
- project-based
- matrix

2.0.4 Collaborative Coding

- Pair programming
- Distributed development

3 Requirements Engineering

requirements engineering can be understood as corresponding to the communication with the users / clients. Deals with the following topics

1. Introduction to communication with users/clients.
    1. Clients and Requirements
    2. Description and specification of requirements
    3. Defining Requirements Engineering
    4. Outcome of Requirements Engineering
    5. Benefit of specification
    6. Complexities of RE
2. Usage modelling / description
    1. Introductory Example
    2. Introduction
    3. Tasks, Roles, Persona
    4. Domain Data
    5. Functions, UI-Structure
    6. GUI
3. Documentation Quality
    1. Introduction and Templates
    2. Characteristics and style guide
4. Usability
5. Quality assurance with the client
    1. acceptance test
    2. usability test
6. Quality requirements
    1. Motivation
    2. Quality attributes
    3. QR-description
    4. QR-test
7. Use-cases (not relevant to the exam)

1. Description of Uses Cases
2. Use for system testing

8. RE procedure

    1. Introduction
    2. Gathering requirements
    3. Specificying requirements

3.0.1 Communication with Users / Clients

- requirements engineering:
  - collection of the requirements from the client
  - specification / formalization of the requirements
  - testing / examination of the requirements
  - management of the requirements
- requirements engineering result:
  - document:
    - description using system functions
  - prototype
- Advantages and Uses of Specification:
- Disadvantages of a missing specification
- Difficulties related to RE:

3.0.2 Usage Modelling

- Task-oriented Requirements Engineering:
  - Task level: tasks, roles, persona
  - Domain level: subtasks (as-is & to-be), domain data
  - interaction level: system functions, ui-structure
  - system level: gui, screen-structure (virtual window)

Roles, Persona, Tasks

3.0.2.1 Roles, Persona

UDC (User-centered design)

Roles and Persona

- User role:
- User profile:

How are personae described:

- name
- biographic facts: age, gender, etc
- knowledge and attitude with respect to the tasks and technology:
- needs: main use-cases in which the user wants to apply the software -> Tasks
- frustrations:
- ideal features:

3.0.2.2 Tasks

How tasks are described:

- Goals
- Decisions
- Causes
- Priority
- Execution profile (frequency, continuity, complexity)
- Precondition
- Info-in (input)
- Info-out (output)
- resources (means, participating roles)

sub-tasks:

Persona-task correspondence:

- needs <=> sub-tasks => combination of system-functions
- frustrations <=> problems in sub-tasks
- ideal features <=> ideas, system functions

3.0.2.3 Domain Data

Domain data explain the terms used in the task descriptions.

Goal: describe/model the entities of real world, including their relationships / associations to each other, in order to understand the tasks.

- domain data describe **entities** that are relevant within the context of the sofwtare. They correspond to the terms used in the task description => independent from the software.
- described with simple **class diagrams** => **domain data diagram** (no operations, no aggregation, no inheritence, only associations)
- sometimes **glossary** is sufficient.

sometimes not sufficient, because there additional data necessary on the UI level => **interaction data**. (not relevant in our MMAP case)

3.0.2.4 Functions & UI-Structure (Interaction Level)

Goal: implementation of the User/Machine boundary with respect to the task descriptions.

consists of 2 parts:

1. System functions: which functions are provided by the system?
2. UI-Structure:
    - in which context can the user call which functions,
    - which data is available/visible in those contexts?
    - how are functionalities divided among the sub-parts?

> ⚠ Warning
>
> UI-Structure is **not** GUI-structure, i.e. the concrete layout is not yet determined.

3.0.2.4.1 System Functions

how is a system function described:

- name: nomenclature verb-object, describe what will be achieved on the user data (e.g. unlinkMovie => movie will be unlinked)
- input:
    - context (i.e. workspace) in which the SF accessible

- concrete input: data that is gathered during the interaction with the user. Such data is not yet determined when the SF is first called on the GUI, but first provided by the user during the interaction.
- output: changes of the UI
- description:
- exceptions: cancel/discard by the user
- rules:
- quality requirements:
- precondition:
- postcondition:

3.0.2.4.2 UI-Structure

Consists of

- workspaces:
  - bundle related system functions and data similar to a class (but only from an abstract user point of view. Actual structure in code can be completely different)
  - only system functions that can be triggered by the user are listed
- navigation links between workspaces

UI-structure abstracts from a concrete screen-layout. Logical represents a logical view of the interaction structure.

> **!** Important
>
> UI-structure is created concurrently with the System functions, because their close interrelation. (Workspaces contain System functions and data)

3.0.2.5 Design of System Functions and UI-Structure

- how is the system function specification template filled in?
- how are ui-structure decision made concurrently to SF specifications?
- initial test considerations?

There's still lots of wiggle room for specific design decisions.

TODO:

3.0.2.6 GUI

GUI= concrete layout of the UI:

- data representation
- function representation
- window structure
- dialogue description (how user controls the execution of functions)

Design principles:

- law of proximity
- law of closure
- law of good continuation
- law of similarity

Types of functions:

- semantic functions: actual data manipulation in the system, e.g. save, open, calculate something etc
- help functions / auxillary functions: data manipulation on the screen, e.g. text size
- search
- navigation

How functions are represented:

- buttons
- checkbox
- menu-selection
- shortcut
- icon
- scroll-bar
- drag-and-drop

Views (Window / Screen): concrete version of Workspaces:

- how is data represented
- how are functions made available / represented

Documentation of a view: Virtual Window (Mock-up)

3.0.3 Documentation Quality

following topics relate to it:

- document templates
- features and sytle-guide

many documents exist:

- Software context-design:
  - problem-description
  - contract
  - acceptance test plan
- Requirements Engineering:
  - Client-requirements
  - Usage test plan
  - Software specification
  - System test plan
- architectural specification:
  - architectural specification / definition
  - sub-system specification
- Detailed design:
  - component specification
  - integration test plan
- Implementation:
  - Code
  - Component test plan

Communication happens via / is facilitated by documents

3.0.4 Usability

The degree to which a product can be used efficiently and adequately for specific tasks / goals, in a specific usage context:

- effectivity
- efficiency
- satisfaction

7 interaction (dialogue) principles :

- Task appropriateness: no unnecessary repetitive actions must be taken bythe user
- Self-descriptiveness: user knows what the actions achieve, what's the input, output and the systems response.
- Controllability: user can decide the order of the operations, can terminate the operation and resume at any given moment without loss.
- Expectation-conformity: system is consistent, confirms to users real-life experience and other software conventions
- Error tolerance: even at the hands of an incorrect input the intended result can be achieved with minimum correction effort.
- User engagement: system must be appealing and inviting and provide users a positive experience.
- Learnability: users are supported and guided during the learning of the software

3.0.5 Quality Assurance with the Clients

- verification: whether the software being built confirms to the specifications derived from the requirements and whether the requierements documents meet quality standards
- validation: whether the specifications actually correspond to clients requirements ⇒:
  - usability test
  - acceptance test

3.0.5.1 Acceptance Test

- Tests whether the client accepts the system ⇒ validation.
- Uses **system tests** provided by the client
- includes particularly also **usability tests**.
- tests are carried out in the production environment (live system where the software is fully deployed and used by the clients)

3.0.5.2 Usability Test

A representative group of users from the target demographic of the product take part in the test to determine to what extent the usability criteria is met.

Consists of:

- users
- observers

how:

- develop a test plan:
  - goal
  - description of the problem
  - description of the users
  - test design
  - list of tasks
  - test environment
  - evaluation criteria
- prepare the test:
  - recruiting a representative group
  - recruiting observers
  - distribution and review of the test plan with the participants
  - prepare scenarios
  - prepare surveys: backgroung, pretest, posttest
  - prepare test env
- execute the test:
- analyse the results

3.0.6 Quality requirements

3.0.6.1 Motivation

- functional requirements ⇒ what?
  - tasks
  - system functions
  - gui
- non-functional requirements (NFR) ⇒ how good?
  - Quality of the system
- QR = quality requirements: describe product considerations
- QA = quality attributes

3.0.6.2 Quality Attributes

QA describe various types of software quality.

Categories:

- Quality in Use: Direct validation with the users
  - Beneficialness: how well are users supported
    - usability
    - accessibility
    - suitability
  - freedom from risk: general impact
  - acceptability (how good is the system from the point of view of the user)
    - support experience
    - trustworthiness
    - compliance (regulations, laws)
- Software Product Quality: verification (internally), validation (externally), continuously during the whole development process
  - functional stability
    - functional completeness
    - functional correctness
    - functional appropriateness (helpful)
  - performance efficiency
    - time behavior
    - resource utilization
    - capacity
  - compatibility
    - co-existence with other software
    - interoperability
  - interaction capability (usability)
    - recognizability
    - learnability
    - operability
    - user error protection
    - user engagement
    - inclusivity
    - user assistance
  - reliability
    - faultlessness
    - availability
    - fault tolerance
    - recoverability
  - security
    - confidentiality

        * integrity
        * non-repudiation
        * accountability
        * authenticity
    - maintainability
        * modularity
        * reusability
        * analysability
        * modifiability
        * testability
    - flexibility
        * adaptability
        * scalability
        * installability
        * replaceability
    - safety
        * operational constraint
        * risk identification
        * fail safe
        * hazard warning
        * safe integration

3.0.6.3 Describing QR

- QR should be made as measurable and as concrete as possible.
- QR are defined in parallel with the FR with as much detail as possible, for all levels:
    - Tasks
    - Domain data
    - Functions
    - GUI

3.0.6.4 Testing QR

- Different QA categories require different testing methods.
- Usually only possible at the level of system testing, especially acceptance testing.
- Usually conducted by acting out scenarios

3.0.7 Use Cases

So far we described the requirements via:

- (Sub)-Tasks
- Domain Data
- Roles / Persona
- System functions
- Interaction data
- UI Structure

At the system level this is further refined via:

- concrete views
- interaction models

But this is only an indirect description of users interaction with the system.

Use Cases describe the specifics of users interactions with the system, specifically the particular execution sequence of system function to complete a certain (sub-)task. Thus, they are derived from **Tasks**

use cases vs TORE:

- use cases cover only a subset of TORE. Particularly
  - No domain data model
  - No UI-Structure (which SF & Data are available in which view, how do you navigate between views)
- UC integrate SF within the execution flow, in TORE SF are contained in the subtask description (although without the execution flow)

3.0.7.1 Description of a UC

Example: manage movie or performer ratings in MMAPP.

- Name: Manage Rating
- Actor: Person
- Goal: to rate movies or performers
- Precondition: at least one movie exists. Depending on the current state wither A1 or A2 is the starting point:
  - Flow of events:
    * Actor: A1) W0.1 Movie Master View … System: System shows the
    * …

- Rules:
- Quality requirements:
- Data, System Functions:
- Postcondition:

3.0.7.2 Use of System Tests

Scenario vs UC:

- Scenario: a specific interaction flow
- UC: an abstract description of a set of scenarios

A choice of a typical set of scenarios can be used as a template when creating a UC. On the other hand a UC can be validated by a subsequent creation of a set of test scenarios.

What is a typical flow?

- normal flow
- coverage of branches / exceptions
- dealing with large input (stress test)
- dealing with complex execution flows (many functions are called)

3.0.8 RE Procedure - Method of RE

So far we only concerned ourselves with describing the requirements. Here we explain how to gather and manage them.

3.0.8.1 Introduction

Stakeholder (client) requirements: the requirements stipulated by the client to achieve a certain goal or solve a certain problem.

1. requirements are gathered
2. requirements are formulated / formalized in the specification
3. specifications are validated and verified
4. specifications are translated into a design
5. the design is implemented
6. the implementation is tested
7. the implementation is deployed to the clients

3.0.8.2 Gathering Requirements

what information should be gathered:

- previous method to solve the problem
- problems related to the previous method
- goals for the new system / method to solve the problem
- success criteria
- rough system architecture (how many components, distribution)
- realistic solutions
- consequences and risks

how it is gathered:

- observation
- user surveying
- ethnographic studies
- use cases
- workshops
- object oriented analysis
- entity relationship diagrams
- conceptual modelling
- data flow diagram
- formal specification
- prototypes
- observations of other products
- literature research
- studying standard software

requirements can also be gathered via user feedback

3.0.8.3 Specifying Requirements

contents and types of requirement specification:

- functional requirements: how should the software support the users?
- non-functional requirements:
  - quality requirements: how well should the software support the users?
  - edge conditions
- rationales: reasoning behind the decisions?

requirement specification is textual as well as schematic (UML, ER etc)

There are two types of requirements:

- client (user) requirements
- system (developer) requirements: requirements from the point of view of the developer with more technical detail.

Usually two different documents are created.

TORE again:

1. task level:
    - tasks
    - roles, persona
2. domain level:
    - sub-tasks
    - domain data
3. interaction level:
    - system functions
    - interaction
    - ui-structure
4. system-level: GUI
    - screen-structure (virtual window)

above levels can be categorized as follows:

- 1, 2: client requirements
- 3, 4: system requirements

Two different specification documents:

- lastenheft (client specification document)
- pflichtenheft (developer specification document)

4 Design

Design can be understood as communication with and within the developers.

Goal: The software system can be further developed efficiently, and is easy to understand for new developers.

Belongs to the development and specifically detailed design category of software development.

Deals with the following topics:

1. Introduction to Modelling
2. Class diagrams
3. Interaction diagrams (sequence diagrams)
4. State Diagrams
    1. UML State diagrams
    2. Dialog models
5. Class design with OOAD
    1. OOAD introduction
    2. OOAD: Analysis Class diagram
    3. OOAD: Design Class Diagram
6. Design Patterns
    1. Introduction
    2. Creational patterns
    3. Structural patterns
    4. Behavioral patterns
7. Rationales (Communication of decisions)
8. Summary of modelling techniques

4.0.1 Introduction to Modelling

A given team (approx 7 people) developes an understanding of the structure of the Software System, albeit without using code, since code is too detailed and is not conducive to understanding the overall structure of the software.

Instead an appropriate abstraction level is necessary ⇒ modelling languages.

Model: Abstraction of a system, expressed in a formal language / notation where irrelevant details are omitted.

Characteristics of models:

- mapping
- incompleteness
- pragmatic (a model is created with a certain purpose / goal)

Characteristics of a formal notation:

- syntax
- semantics
- pragmatics
  - analysis techniques: type checking, consistency checks
  - simulation techniques
  - transformation techniques (refactoring, (algebraic) simplification)
  - generation techniques

A notation is a 'theory' that enables to reason about and manipulate representations of objects from the domain.

4.0.1.1 UML

defines:

- Structure diagrams → statics of the system
- Behavior diagrams → dynamics of the system
  - interactions
  - flows
  - state transitions

4.0.1.1.1 Structure Diagrams

- Design:
  - Class Diagram (analysis and design)
  - Object diagram (special cases)
  - package diagram (bundles of classes)
- Architecture:

  - compositional structure diagram
  - logical component diagram (internal and external view)
  - distribution diagram (over the physical components)

4.0.1.1.2 Behavior Diagrams

- Flows:
  - Use case diagram (overview of the use case)
  - activity diagram (sequences of activities)
  - state diagram (sequences of states)
- interaction:
  - sequence diagram (sequence of messages)
  - communication diagram (focus on a component)
  - time diagram (communication between automata)
  - interaction overview diagram (interaction of multiple interactions)

In the lecture especially:

- class diagrams
- object diagrams
- interaction diagrams
- state diagrams

4.0.2 Class Diagrams

important components of class diagrams:

- classes (objects)
- associations between classes:
  - aggregation
  - komposition
- attributes
- operations
- generalization / specialization relationship (inheritance)
- interfaces: a view on a class (a contract)

4.0.3 Interaction Diagrams (Sequence Diagrams)

4.0.4 State Diagrams

4.0.5 Class Design with OOAD

4.0.6 Design Patterns

4.0.7 Rationales (Communication of Decisions)

Documents contain only the last decision. Communication of all sorts of decisions and history of discarded decisions via Rationales.

How are rationales described:

- questions: concrete problems that don't have an obvious solution
- options: describe alternative solutions to a problem
- criteria: quality requirements
- arguments: condensate and summarize discussions
- decisions:
  - relates to one or more open questions
  - summarizes the chosen options and arguments that support it
-

5 Quality Assurance

Quality: Software satisfies the requirements

topics:

1. Introduction
2. Organizational quality assurance
3. Testing:
    1. Intro
    2. Test-case specification
    3. Black-box component testing
    4. White-box component testing
    5. System testing
    6. Integration testing / overall-component testing
4. Static testing
    1. Static Analysis
    2. Metrics
    3. Inspection
5. Analytical Quality assurance at large

6 Evolution

All activities that facilitate re-use and further development. (All activities that take place after the initial development phase)

topics:

1. Intro
2. Architecture
3. Re-use
4. Further development and change management
5. DevOps & IT-Governence
6. Re-engineering

7 SWE Process & Project Management

Making sure that the Software system is developed withing the time money constraints.

topics:

1. Project management
2. SWE-process models & methods