Final Look on UML, Requirements Elicitation

Object-Oriented Software Construction

Prof. Dr. Armin B. Cremers, Sascha Alda & Tobias Rho
(based on Bruegge & Dutoit)
Where are we right now?

♦ Introduction
♦ Java
♦ UML
  ♦ Use Case Diagrams
  ♦ Class Diagrams

♦ Today: Consider Dynamic Diagrams (continuation of last lecture)
Sequence Diagrams

- Used during requirements analysis
  - To refine use case descriptions
  - to find additional objects ("participating objects")
- Used during system design
  - to refine subsystem interfaces
- **Objects** are represented by columns
- **Messages** are represented by arrows
- **Activations** are represented by vertical rectangles
- **Lifelines** are represented by dashed lines
Sequence Diagrams: Nested messages

- Concrete Sequences visualize one possible interaction
- The source of an arrow indicates the activation which sends the message
- An activation is as long as all nested activations
- Horizontal dashed arrows indicate data flow
Sequence Diagrams: Iteration & Condition

- Abstract Sequences visualize *all possible* interactions
- Iteration is denoted by a * preceding the message name
- Condition is denoted by a boolean expression in [ ] before the message name

...continued from previous slide...

...to be continued...

Dynamic Models
Sequence Diagrams: Creation and Destruction

- Creation is denoted by a message arrow pointing to the object
- Destruction is denoted by an X mark at the end of the destruction activation
- In garbage collection environments, destruction can be used to denote the end of the useful life of an object
Sequence Diagrams: Concurrency

- Asynchronous messages
- Both objects stay active, can communicate with each other
- Half arrow marks messages without transmitting control flow
  - New thread is created
  - Objects can kill themselves or can be killed
Sequence Diagrams: Summary

- UML sequence diagrams represent behavior in terms of interactions
- Useful to find missing objects
- Time consuming to build but worth the investment
- Presentability might be difficult
- Complement the class diagrams (which represent structure)

- Easy understandable Control Flow
- Concurrency can be presented
Where are we right now? (1/2)

♦ Introduction

♦ UML
  - Use Case Diagrams
  - Class Diagrams
  - Dynamic Diagrams (State Chart, Activity, Sequence)

♦ Java

♦ → Back to the Software Lifecycle
Where are we right now? (2/2)

♦ Three ways to deal with **complexity**
  - Abstraction
  - Decomposition (Technique: Divide and conquer)
  - Hierarchy (Technique: Layering)

♦ Two ways to deal with decomposition
  - Object-orientation and functional decomposition
  - Functional decomposition leads to un-maintainable code
  - Depending on the purpose of the system, different objects can be found

♦ What is the right way?
  - Start with a description of the functionality (Use case model). Then proceed by finding objects (object model).

♦ What activities and models are needed?
  - This leads us to the software lifecycle we use in this course
Software Lifecycle Definition

♦ Software lifecycle
  ♦ Set of activities and their relationships to each other to support the development of a software system

♦ Typical Lifecycle questions
  ♦ Which activities should I select for the software project?
  ♦ What are the dependencies between activities?
  ♦ How should I schedule the activities?
  ♦ What is the result of an activity?
Example: Selection of Software Lifecycle Activities for a specific project

♦ The Hacker knows only one activity

♦ Activities used in this lecture

♦ Each activity produces one or more models
Software Lifecycle Activities

- Requirements Elicitation
- Analysis
- System Design
- Object Design
- Implementation
- Testing

Use Case Model

- Application Domain Objects
- Subsystems
- Solution Domain Objects
- Source Code
- Test Cases

Expressed in Terms of
Structured by
Realized by
Implemented by
Verified by

Armin B. Cremers, Sascha Alda & Tobias Rho (based on Bruegge & Dutoit)
Object-Oriented Software Construction
Introduction: What are Requirements?

- Functional requirements
  - Describe the interactions between the system and its environment independent from implementation

- Non-functional requirements A (Quality Requirements)
  - User visible aspects of the system not directly related to functional behavior.
  - Reliability, Performance, Supportability, Usability, Tailorability

- Non-functional requirements B (Pseudo requirements)
  - Imposed by the client or the environment in which the system operates
  - Interfaces, Legal requirements

- Non-functional requirements C (Project Management)
  - Budget, Release Date
Introduction:
What is usually not in the requirements?

- System structure
- Development methodology
- Development environment
- Implementation language
- User Interface Design
- Reusability

- It is desirable that none of these above are constrained by the client.
Introduction: What is Requirements Engineering? (1/2)

♦ Defining Requirements is a very challenging activity

♦ Requirements Elicitation needs Collaboration between different groups
  ♦ Developers (design knowledge, implementation knowledge)
  ♦ Clients
  ♦ Users (domain knowledge)

♦ Two questions need to be answered
  ♦ How can we identify the purpose of a system?
  ♦ What is inside, what is outside the system? Very hard to decide!

♦ Errors during Requirements Elicitation
  ♦ System fails to support users’ work: Missing or incorrect functionality
  ♦ Corrections are expensive
Introduction: What is Requirements Engineering? (2/2)

Requirements Engineering

Requirements Elicitation
- Basis for Discussions
- Definition of the system in terms understood by the customer ("Problem Description")

Requirements Analysis
- Design basis for developers
- Technical specification of the system in terms understood by the developer ("Problem Specification")
Process of Requirements Elicitation: Products of Requirements Process

- **Problem Statement**
- **System specification:**
  - functional and non-functional requirements
- **Analysis Model:**
  - dynamic model
  - Analysis object model
Starting with the Problem Statement

♦ The problem statement is developed by the client (in cooperation with developer) that briefly describes the scope of the system

♦ A good problem statement describes
  ♦ The current situation
  ♦ The functionality the new system should support
  ♦ The environment in which the system will be deployed
  ♦ Deliverables expected by the client
  ♦ Delivery dates
  ♦ A set of acceptance criteria
Starting with the Problem Statement: Ingredients

♦ Current situation: The Problem to be solved
♦ Description of one or more scenarios
♦ Requirements
  ♦ Functional and Non-functional requirements
  ♦ Constraints ("pseudo requirements")
♦ Project Schedule
  ♦ Major milestones that involve interaction with the client including deadline for delivery of the system
♦ Target environment
  ♦ The environment in which the delivered system has to perform a specified set of system tests
♦ Client Acceptance Criteria
  ♦ Criteria for the system tests
Starting with the Problem Statement: Current Situation - The Problem To Be Solved

- There is a problem in the current situation
  - Examples:
    - The response time when playing online-chess is far too slow
    - I want to play Chess, but cannot find players on my level
- What has changed? Why can we address the problem now?
  - There has been a change, either in the application domain or in the solution domain
  - Change in the application domain
    - A new function (business process) is introduced into the business
    - Example: We can play highly interactive games with remote people
  - Change in the solution domain
    - A new solution (technology enabler) has appeared
    - Example: The internet allows the creation of virtual communities
The Goal:
Analysis Model (vs. System Specification)

♦ Both models focus on the requirements from the user’s view of the system.

♦ *System specification* uses natural language (derived from the *problem statement*)

♦ The *analysis model* uses formal or semi-formal notation (for example, a graphical language like UML)

♦ The starting point is the problem statement
Process of Requirements Elicitation: Activities during Requirements Elicitation

- Identifying Actors
  - Types of users, roles, external systems

- Identifying Scenarios
  - Interactions between users and the systems (one possible case)
  - \(\rightarrow\) Later

- Identifying Use Cases
  - Abstractions of Scenarios
    (Many possible cases)
  - \(\rightarrow\) Later

- Refining Use Cases
  - Refinements, adding exceptions, etc.

- Identifying Relationships among Use Cases

- Identifying Non-Functional Requirements
  - Reliability issues, Performance, etc.
Process of Requirements Elicitation: The Requirements Elicitation Cycle

Observing users

Interviewing users and clients

As-Is Scenarios

Visionary Scenarios

Use Cases + Refinements

Prototypes

Validation

Validation

Tests

Validation

Stable Requirements Specification (System Specification)

- Functional Requirements
- Non-Functional Requirements
- Use Cases
- Scenarios
Process of Requirements Elicitation: Types of Requirements Elicitation

♦ Greenfield Engineering
  ♦ Development starts from scratch, no prior system exists, the requirements are extracted from the end users and the client
  ♦ Triggered by user needs
  ♦ Example: Develop a game from scratch: Asteroids 4

♦ Re-Engineering
  ♦ Re-design and/or re-implementation of an existing system using newer technology
  ♦ Triggered by technology enabler
  ♦ Example: Re-Engineering an existing game

♦ Interface Engineering
  ♦ Provide the services of an existing system in a new environment
  ♦ Triggered by technology enabler or new market needs
  ♦ Example: Interface to an existing game
Process of Requirements Elicitation: How to elicit Requirements?

♦ Sources of information
  ♦ Documents about the application domain
  ♦ Manual and technical documents of legacy systems

♦ User Participation
  ♦ Interviews
    ♦ Closed Interviews: User answer a predefined set of questions
    ♦ Open Interviews: No predefined agenda
  ♦ Topics:
    ◦ Description of current work practice
    ◦ Brainstorming on future system’s functionality
  ♦ Problems: Specific domain terminology
  ♦ Observing Work Practice

→ Describing Scenarios
**Process of Requirements Elicitation: The Requirements Elicitation Cycle**

- Observing users
- Interviewing users and clients
- As-Is Scenarios
- Visionary Scenarios
- Use Cases + Refinements
- Prototypes
- Validation
- Validation
- Tests

**Stable Requirements Specification (System Specification)**

- Functional Requirements
- Non-Functional Requirements
- Use Cases
- Scenarios
Scenarios

♦ “A narrative description of what people do and experience as they try to make use of computer systems and applications” [M. Carrol, Scenario-based Design, Wiley, 1995]

♦ A concrete, focused, informal description of a single feature of the system used by a single actor.

♦ Scenarios can have many different uses during the software lifecycle
  ✷ **Requirements Elicitation**: As-is scenario, visionary scenario
  ✷ **Client Acceptance Test**: Evaluation scenario
  ✷ **System Deployment**: Training scenario.
Scenarios: Different Types

- **As-is scenario**
  - Used in describing a current situation
  - Usually used in re-engineering projects
  - The user describes the system

- **Visionary scenario**
  - Used to describe a future system
  - Usually used in greenfield engineering and reengineering projects
  - Can often not be done by the user or developer alone
    → brainstorming sessions, future workshop

- **Evaluation scenario**
  - User tasks against which the system is to be evaluated

- **Training scenario**
  - Step by step instructions that guide a novice user through a system
Scenarios: How do we find scenarios?

- Don’t expect the client to be verbal, if the system does not exist (greenfield engineering)
- Don’t wait for information even if the system exists
- Engage in a dialectic approach (evolutionary, incremental engineering)
  - You help the client to formulate the requirements
  - The client helps you to understand the requirements
  - The requirements evolve while the scenarios are being developed
Scenarios: Heuristics for finding Scenarios

♦ Ask yourself or the client the following questions
  ◆ What are the primary tasks that the system needs to perform?
  ◆ What data will the actor create, store, change, remove or add in the system?
  ◆ What external changes does the system need to know about?
  ◆ What changes or events will the actor of the system need to be informed about?
♦ However, don’t rely on questionnaires alone.
♦ Insist on task observation if the system already exists (interface engineering or reengineering)
  ◆ Ask to speak to the end user, not just to the software contractor
  ◆ Expect resistance and try to overcome it
Scenarios: Example - Accident Management System

♦ What needs to be done to report a “Cat in a Tree” incident?
♦ What do you need to do if a person reports “Warehouse on Fire?”
♦ Who is involved in reporting an incident?
♦ Can the system cope with a simultaneous incident report “Warehouse on Fire?”
Scenario: Example - Warehouse on Fire

♦ Bob, driving down main street in his patrol car notices smoke coming out of a warehouse. His partner, Alice, reports the emergency from her car by using the SYSTEM.

♦ Alice enters the address of the building, a brief description of its location (i.e., north west corner), and an emergency level. In addition to a fire unit, she requests several paramedic units on the scene given that area appear to be relatively busy. She confirms her input and waits for an acknowledgment.

♦ John, the Dispatcher, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire unit and two paramedic units to the Incident site and sends their estimated arrival time (ETA) to Alice.

♦ Alice received the acknowledgment and the ETA.
Scenarios: Observations about “Warehouse on Fire”

♦ Concrete scenario
  ◆ Describes a single instance of reporting a fire incident.
  ◆ Does not describe all possible situations in which a fire can be reported.

♦ Participating actors
  ◆ Bob, Alice and John
Process of Requirements Elicitation: The Requirements Elicitation Cycle

Observing users → As-Is Scenarios

Interviewing users and clients → Visionary Scenarios

Use Cases + Refinements → Prototypes

Tests

Validation

Validation

Validation

Stable Requirements Specification (System Specification)

- Functional Requirements
- Non-Functional Requirements
- Use Cases
- Scenarios
Use Cases:  
Next goal, after the scenarios are formulated

- Find all the use cases in the set of scenarios that completely represent the future system.
  - Abstractions of many scenarios
  - Example: “Report Emergency” is a candidate for a use case from the first paragraph of the scenario
- Describe each of these use cases in more detail
  - Participating actors
  - Describe the Entry Condition
  - Describe the Flow of Events
  - Describe the Exit Condition
  - Describe Exceptions
  - Describe Special Requirements (Constraints, non-functional Requirements)
Use Cases: Definition

♦ A use case is a flow of events in the system, including interaction with actors
♦ It is initiated by an actor
♦ Each use case has a name
♦ Each use case has a termination condition
♦ Graphical Notation: An oval with the name of the use case

ReportEmergency

Use Case Model: The set of all use cases specifying the complete functionality of the system
Use Cases: Communication relationships between actors and use cases

- Users, Roles
- Which user can invoke which use case
Use Cases

Textual Representation

- Template for a textual use case

<table>
<thead>
<tr>
<th>Use case name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Participating Actors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flow of Events</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Entry Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Exit Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quality Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
### Use Cases Textual Representation

- **Example (Excerpt, see [Bruegge & Dutoit, 2004], p. 135)**

<table>
<thead>
<tr>
<th>Use case name</th>
<th>ReportEmergency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Participating Actors</strong></td>
<td>Initiated by <strong>FieldOfficer</strong>, communicates with <strong>Dispatcher</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flow of Events</th>
<th>1. The <strong>FieldOfficer</strong> activates the ReportEmergency function on this terminal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2. SYSTEM responds by presenting a dialog form to the <strong>FieldOfficer</strong></td>
</tr>
<tr>
<td></td>
<td>3. <strong>FieldOfficer</strong> completes and submits the form</td>
</tr>
<tr>
<td></td>
<td>4. SYSTEM receives the form, evaluates the information in interaction with a <strong>Dispatcher</strong> and sends an acknowledge</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Entry Condition</th>
<th>The <strong>FieldOfficer</strong> is logged into the SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exit Condition</td>
<td>The <strong>FieldOfficer</strong> receives an acknowledge</td>
</tr>
</tbody>
</table>

**Quality Requirements**

- The **FieldOfficer's** request is acknowledged within 30 seconds.
- The Dialog form should be implemented by using Java SWING classes
Use Cases: Associations

- A use case model consists of use cases and use case associations
  - A use case association is a relationship between use cases
- Important types of use case associations: Include, Extends, Generalization
  - Include
    - A use case uses another use case (“functional decomposition”)
  - Extends
    - A use case extends another use case
  - Generalization
    - An abstract use case has different specializations
Use Cases: <<Include>> Functional Decomposition

- **Problem:**
  - A function in the original problem statement is too complex to be solvable immediately

- **Solution:**
  - Describe the function as the aggregation of a set of simpler functions. The associated use case is decomposed into smaller use cases

---

Reducing complexity by functional decomposition

- ManageIncident
  - <<include>>
  - CreateIncident
  - HandleIncident
  - CloseIncident
Use Cases: <<Include>> Reuse of Existing Functionality

- **Problem:**
  - There are already existing functions. How can we reuse them?

- **Solution:**
  - The include association from a use case A to a use case B indicates that an instance of the use case A performs all the behavior described in the use case B (“A delegates to B”)

---

Reusing existing behavior and avoid double definitions.

**Note:**
(a) The base case cannot exist alone. It is always called with the supplier use case. (b) The Base UC knows, when to call the supplier.
Use Cases: `<<Extend>>` Association for Use Cases

- **Problem:**
  - The functionality in the original problem statement needs to be extended (optional behaviour, exceptional behaviour)

- **Solution:**
  - An extend association from a use case A to a use case B indicates that use case B is an extension of use case A.

**Note:** (a) The base use case can be executed without the use case extension in extend associations. (b) The Extending UC knows when it should be executed.
Use Cases in a textual way
Describing \textit{<<include>>} and \textit{<<extend>>}

\begin{itemize}
  \item \textbf{\textit{<<include>>}} relationship
  \begin{itemize}
    \item Flow of Events
      \begin{itemize}
        \item 3. FieldOfficer completes and submits the form
        \item 4. SYSTEM receives the form, evaluates the information (by using Use Case \textit{EvaluateInfo}) being received and sends an acknowledge
      \end{itemize}
  \end{itemize}

  \item \textbf{\textit{<<extend>>}} relationship
  \begin{itemize}
    \item Flow of Events
      \begin{itemize}
        \item The \textit{ConnectionDown} use case extends any use case in which the connection between the FieldOfficer and the SYSTEM can be lost
        \item 1. The FieldOfficer is notified that the connection is broken
      \end{itemize}
  \end{itemize}
\end{itemize}
Use Cases: Generalization association in use cases

♦ Problem:
  ♦ You have common behavior among use cases and want to factor this out.

♦ Solution:
  ♦ The generalization association among use cases factors out common behavior. The child use cases inherit the behavior and meaning of the parent use case and add or override some behavior.

![Diagram of use cases]

- Parent Case: ValidateUser
- Child Use Case: CheckPassword, CheckFingerprint
Use Cases: How do I find use cases?

- Refine a single scenario of the system
  - Discuss it in detail with the user to understand the user’s assumption about the system
- Select a horizontal slice (i.e. many scenarios) to define the scope of the system.
  - Discuss the scope with the user
- Use illustrative prototypes (mock-ups) as visual support
- Find out what the user does
  - Task observation (Good)
  - Questionnaires (Bad)
Order of steps when formulating use cases

♦ First step: Name the use case
  ♦ Use case name: ReportEmergency (not: Report → who, what)

♦ Second step: Find the actors
  ♦ Generalize the concrete names (“Bob”) to participating actors (“Field officer”)
  ♦ Participating Actors:
    ♦ Field Officer (Bob and Alice in the Scenario)
    ♦ Dispatcher (John in the Scenario)

♦ Third step: Concentrate on the flow of events
  ♦ Use informal natural language