An introduction to Unified Modeling Language (UML) in Software Engineering

Nils-Olav Skeie
Overview

- Introduction,
  - Software development process
- Analysis,
  - Use Case,
- Design,
  - Interaction diagrams,
  - Class diagram,
- Testing,
- Conclusion.
Why is developing software so hard?

• Users don’t know what they want until they see it,
  – They only think that they know what they want.
  – How to understand the need(s) of the user(s)?

• Lack of user input,
  – Management decisions, users too busy, poor user involvement,…

• Programming is easy,
  – Easy to learn how to write code, but huge gap between that and making great software,
  – Software coding is probably the easiest part of being a software engineer,

• Every line of code is a potential point of failure,

• The software industry is young,

• Estimating time is an art, not science,

• Software application is like an iceberg – 90% of the complexity is not visible!
Software Development Process

- Coding;
  - only a part of software development,

- Developing software using UML;
  - UML is only a tool!
  - Used in a development process,
  - Select the UML development tool;
    - Tool depend on the methodology,
    - Software or paper based,

- Use case
  - Main functions of the software

- Practice;
  - UML and tool,
  - ”small steps”
Object-oriented development

- **Class;**
  - An abstract definition of some sort of function in the problem domain,
  - Consists of a name, data and methods.

- **Object;**
  - An instance of a class in computer memory, with valid data,

- **OOAD;**
  - Using Object-Oriented methods for analyzing and designing applications,
  - Assigning responsibilities to each object,
  - Let objects cooperate to solve a specific task or a set of tasks.
UML diagram tools

- Handmade or computer based diagrams,
  - Visio
  - StarUML;
    - A Delphi based open-source UML platform for MS-Windows,
    - Support C#, Java, and C++,
  - Rational Rose (Rational, IBM);
    - License fee, expensive,
  - ArgoUML;
    - a Java-based open source free UML modeling tool,
    - Support Java,
  - Umbrello;
    - part of the KDE user interface of Linux,
    - Support C++/Java/Perl/PHP/Python++,
  - Rhapsody;
    - embedded and real-time systems.
- ++++

- Umbrello;
- Rhapsody;
Business Area / Specification

- Specification,
  - An oral or written description of a challenge,

- Work,
  - Business activities of the product owner,
  - Activities that the owner wants to improve,
  - Business use cases (BUC),

- Requirements,
  - Make a document describing the requirements for the system,
  - Testable!
Exercise: Business Area / BUC

- Software for a Washing Machine
  - Specification?
  - Business Use Cases?
  - Requirements?
Analysis

- Collect requirements,
- Make use case diagram,
- Make use case document,
- Use case diagram,
  - Use case; main functions of the software,
  - Actors; I/O of the software,
- Use case document,
  - Text description of each use case.
Requirements

• Use the FURPS+ letters, make a text document;
  – F is Functionality;
    • Main functions of the software, the use cases, starts with a verb,
  – U is Usability;
    • How to interact with the software, human factors, help, documentation,
  – R is Reliability;
    • Predictability, Accuracy, Mean time to failure,
  – P is Performance;
    • Speed, Resource consumption, Throughput, Response time,
  – S is Supportability;
    • Testability, Adaptability, Maintainability, Configurability,
  – + (extra)
    • Implementation, licenses, administration, interface to external systems,
Use case diagram

- How software will fulfill the requirements of the external actors,
- Functional section from the requirements,
- Consists of a set of actors and use cases,
- Actor
  - "something" requires a function or service of the software,
  - Often a person, hardware device, software function (OS) or another computer system,
- Use case
  - Main functions of the applications,
  - The functions required to/from the actors,
  - Use a verb in the use case name,
  - More details for each use case in a use case document.
Exercise: Use case diagram

- Requirements?
- Main functions of software?
- Any actors?
Use case document (iterations)

- Select the most important use case,
- Use case documents (FDUCD):
  - Describe in a text document,
  - Brief, Casual, or **Fully** dressed.
  - Different templates available,
    - Preconditions,
    - Success Guarantee,
    - Main Success Scenario (basic flow),
      - Actor events.
    - Extensions (conditional/branches),
      - Actor events.
  - Important sections; (6, 7) 8, 9 and 12.

<table>
<thead>
<tr>
<th>Use case section</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  Use case name</td>
<td>Start with a verb</td>
</tr>
<tr>
<td>2  Scope</td>
<td>The system under design</td>
</tr>
<tr>
<td>3  Level</td>
<td>“user goal” or “sub function”</td>
</tr>
<tr>
<td>4  Primary actor</td>
<td>The main user of the function</td>
</tr>
<tr>
<td>5  Stakeholders and Interests</td>
<td>Who cares, and what do they want</td>
</tr>
<tr>
<td>6  Preconditions</td>
<td>What must be fulfill before starting</td>
</tr>
<tr>
<td>7  Success Guarantee</td>
<td>What must be fulfilled at a successful completion</td>
</tr>
<tr>
<td>8  Main success scenario</td>
<td>A typical set of events</td>
</tr>
<tr>
<td>9  Extensions</td>
<td>Alternative scenarios of success or failure</td>
</tr>
<tr>
<td>10 Special requirements</td>
<td>Related non-functional requirements</td>
</tr>
<tr>
<td>11 Technology list</td>
<td>Different I/O methods and data formats</td>
</tr>
<tr>
<td>12 Frequency of occurrence</td>
<td>Investigation, testing and timing of implementation</td>
</tr>
<tr>
<td>13 Miscellaneous</td>
<td>Such as open issues</td>
</tr>
</tbody>
</table>
Analysis: Documents

- Specification and Requirements,
  - FURPS+,
  - Text,
- Use case diagram,
  - UML,
  - Main functions and actors,
- Use case document,
  - Text,
  - Description of each use case,
  - One document for each use case.

Documentations – part of:
- (SRS: Software Requirements & Specifications)
- **SRD: Software Requirements and Design**
Analysis to Design

- Design:
  - How the software is going to do the work,
  - Structure,
    - Classes and objects,
  - Give responsibility to objects,
Giving responsibility to objects

Two types of responsibilities:

- **knowing**
  1. private encapsulated data,
  2. related objects,
  3. “things” it can derive or calculate

- **doing**
  1. doing something itself (creating an object or doing a calculation),
  2. initiating action in other objects,
  3. controlling and coordinating activities in other objects.
Design Model

• Describe the scenario of a use case with collaborating objects,
• The system operation starts with a controller object,
• Use UML diagrams
  – interaction diagrams,
  – class diagrams,
Interaction diagrams
- Sequence diagram

• Shows the sequence of the messages in time,
• Simple notation,
• Takes a lot of horizontal space,
• Notation ("UML" Coding):
  – No answer: message()
  – With answer: ans:=message()
  – Condition: [x<10]:ans:=message()
  – Loop: *[i=1..n]: ans:=message()
Sequence diagram

• Use several objects for cooperation,
• Use responsibility for naming of objects,
• Include the overview cooperation,
• Do not include details!
Exercise: Sequence diagram

- Washing cloth at 40 deg. C
Class and Object diagrams

- Class and Object diagrams from the interaction diagrams,
  - One interaction diagram for each use case,
  - Common class and object diagrams.

- Get all the information from the interaction diagrams,
  - Class diagram and object diagram with names, attributes, and methods
Class and Object diagram

- Exercise: Make a class diagram
Design: Documents

- Interaction diagram,
  - Sequence diagram, focus on time,
  - One diagram for each use case document,
  - Dynamic model of your software,
- Class diagram,
  - Static model of your software,
  - One common diagram,
- Object diagram,
  - Static model of your application,
  - One common diagram.

Documentations – part of:
- (SDD: Software Design Document)
- SRD: Software Requirements and Design
Testing

• More than 50% of errors may arise before coding,
• Testing in each iteration,
• Deploy an early version for the customer,
• For testing:
  – Requirements,
  – Use Case Diagram,
  – Use Case Documents (Fully Dressed Use Case Document),
• Make a test plan; Description/Setup and test cases,
• All test documents produced before coding.
Conclusions

• Use a development process
• Use time in the analysis phase,
  – Collect requirements,
  – Make use case diagram and use case documents,
• Design phase,
  – Assign responsibility to objects,
  – Sequence diagram; dynamic information,
  – Class diagram; static abstract information,
  – Object diagram; static runtime information,
  – Testing is based on requirements.