Introduction to Software Engineering
(ESE : Einführung in SE)

Prof. O. Nierstrasz
# ESE — Introduction

| **Lecturer** | Prof. Oscar Nierstrasz  
| | scg.unibe.ch/oscar |
| **Assistants** | Erwann Wernli  
| | Aaron Karper, Joel Krebs |
| **Lectures** | Engehaltenstrasse 8, 001,  
| | Wednesdays @ 13h15-15h00 |
| **Exercises** | Engehaltenstrasse 8, 001  
| | Wednesdays @ 15h00-16h00 |
| **WWW** | scg.unibe.ch/teaching/ese |
Roadmap

- Course Overview
- What is Software Engineering?
- The Iterative Development Lifecycle
- Software Development Activities
- Methods and Methodologies
Roadmap

> Course Overview
> What is Software Engineering?
> The Iterative Development Lifecycle
> Software Development Activities
> Methods and Methodologies
Principle Texts


Recommended Literature


**Course schedule**

<table>
<thead>
<tr>
<th>Week</th>
<th>Date</th>
<th>Lesson</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21-Sep-11</td>
<td>Introduction: The Software Lifecycle</td>
</tr>
<tr>
<td>2</td>
<td>28-Sep-11</td>
<td>Requirements Collection</td>
</tr>
<tr>
<td>3</td>
<td>05-Oct-11</td>
<td>The Planning Game</td>
</tr>
<tr>
<td>4</td>
<td>12-Oct-11</td>
<td>Responsibility-Driven Design</td>
</tr>
<tr>
<td>5</td>
<td>19-Oct-11</td>
<td>Modeling Objects and Classes</td>
</tr>
<tr>
<td>6</td>
<td>26-Oct-11</td>
<td>Modeling Behaviour</td>
</tr>
<tr>
<td>7</td>
<td>02-Nov-11</td>
<td>Software Validation</td>
</tr>
<tr>
<td>8</td>
<td>09-Nov-11</td>
<td>User Interface Design</td>
</tr>
<tr>
<td>9</td>
<td>16-Nov-11</td>
<td>Project Management</td>
</tr>
<tr>
<td>10</td>
<td>23-Nov-11</td>
<td>Software Architecture</td>
</tr>
<tr>
<td>11</td>
<td>30-Nov-11</td>
<td>Software Quality</td>
</tr>
<tr>
<td>12</td>
<td>07-Dec-11</td>
<td>Software Metrics</td>
</tr>
<tr>
<td>13</td>
<td>14-Dec-11</td>
<td>Guest lecture: SE in practice</td>
</tr>
<tr>
<td>14</td>
<td>21-Dec-11</td>
<td>Software Evolution</td>
</tr>
<tr>
<td>15</td>
<td>12-Jan-12</td>
<td>Final Exam: ExWi A6 @ 10h00-12h00</td>
</tr>
</tbody>
</table>
Roadmap

> Course Overview
> **What is Software Engineering?**
> The Iterative Development Lifecycle
> Software Development Activities
> Methods and Methodologies
Why Software Engineering?

A naive view:

Problem Specification  coding  Final Program

But ...

— Where did the specification come from?
— How do you know the specification corresponds to the user’s needs?
— How did you decide how to structure your program?
— How do you know the program actually meets the specification?
— How do you know your program will always work correctly?
— What do you do if the users’ needs change?
— How do you divide tasks up if you have more than a one-person team?
Some Definitions and Issues

“state of the art of developing quality software on time and within budget”

> Trade-off between perfection and physical constraints
  — SE has to deal with real-world issues

> State of the art!
  — Community decides on “best practice” + life-long education
What is Software Engineering? (II)

“multi-person construction of multi-version software”
— Parnas

> **Team-work**
  — Scale issue (“program well” is not enough) + Communication Issue

> **Successful software systems must evolve or perish**
  — Change is the norm, not the exception
“software engineering is different from other engineering disciplines”

— Sommerville

> Not constrained by physical laws
   — limit = human mind

> It is constrained by political forces
   — balancing stake-holders
Roadmap

> Course Overview
> What is Software Engineering?
> The Iterative Development Lifecycle
> Software Development Activities
> Methods and Methodologies
Software Development Activities

<table>
<thead>
<tr>
<th>Requirements Collection</th>
<th>Establish customer’s needs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis</td>
<td>Model and specify the requirements (“what”)</td>
</tr>
<tr>
<td>Design</td>
<td>Model and specify a solution (“how”)</td>
</tr>
<tr>
<td>Implementation</td>
<td>Construct a solution in software</td>
</tr>
<tr>
<td>Testing</td>
<td>Validate the solution against the requirements</td>
</tr>
<tr>
<td>Maintenance</td>
<td>Repair defects and adapt the solution to new requirements</td>
</tr>
</tbody>
</table>

NB: these are ongoing activities, not sequential phases!
The classical software lifecycle models the software development as a step-by-step “waterfall” between the various development phases.

The waterfall model is unrealistic for many reasons:
• requirements must be *frozen too early* in the life-cycle
• requirements are *validated too late*
Problems with the Waterfall Lifecycle

1. “Real projects rarely follow the sequential flow that the model proposes. *Iteration* always occurs and creates problems in the application of the paradigm”

2. “It is often *difficult* for the customer *to state all requirements* explicitly. The classic life cycle requires this and has difficulty accommodating the natural uncertainty that exists at the beginning of many projects.”

3. “The customer must have patience. A *working version* of the program(s) will not be available until *late in the project* timespan. A major blunder, if undetected until the working program is reviewed, can be disastrous.”

— Pressman, SE, p. 26
Iterative Development

In practice, development is always iterative, and all activities progress in parallel.

If the waterfall model is pure fiction, why is it still the dominant software process?
Iterative Development

Plan to *iterate* your analysis, design and implementation.

— You won’t get it right the first time, so *integrate, validate and test* as frequently as possible.

— “You should use iterative development only on projects that you want to succeed.”
  – *Martin Fowler, UML Distilled*
Plan to *incrementally* develop (i.e., prototype) the system.

— If possible, always have a *running version of the system*, even if most functionality is yet to be implemented.
— *Integrate* new functionality as soon as possible.
— *Validate* incremental versions against user requirements.
### The Unified Process

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td>Iter. #1</td>
<td>Iter. #2</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>Iter. #n-1</td>
<td>Iter. #n</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**How do you plan the number of iterations?**

**How do you decide on completion?**
Boehm’s Spiral Lifecycle

- **Planning** = determination of objectives, alternatives and constraints
- **Risk Analysis** = Analysis of alternatives and identification/resolution of risks
- **Risk** = something that will delay project or increase its cost
- **Customer Evaluation** = Assessment of the results of engineering
- **Engineering** = Development of the next level product
- **initial requirements**
- **completion**
- **alpha demo**
- **go, no-go decision**
- **first prototype**
- **evolving system**
Roadmap

- Course Overview
- What is Software Engineering?
- The Iterative Development Lifecycle
- Software Development Activities
- Methods and Methodologies
User requirements are often expressed *informally*:  
— features  
— usage scenarios

Although requirements may be documented in written form, they may be *incomplete, ambiguous*, or even *incorrect*. 
Changing requirements

Requirements *will* change!

— *inadequately captured* or expressed in the first place
— user and business *needs may change* during the project

Validation is needed *throughout* the software lifecycle, not only when the “final system” is delivered!

— build constant *feedback* into your project plan
— plan for *change*
— early *prototyping* [e.g., UI] can help clarify requirements
Analysis is the process of specifying *what* a system will do.

—The intention is to provide a clear understanding of what the system is about and what its underlying concepts are.

The result of analysis is a *specification document*.

*Does the requirements specification correspond to the users’ actual needs?*
An *object-oriented analysis* results in models of the system which describe:

> **classes** of objects that exist in the system
  — **responsibilities** of those classes
> **relationships** between those classes
> **use cases** and **scenarios** describing
  — **operations** that can be performed on the system
  — allowable **sequences** of those operations
A **prototype** is a software program developed to test, explore or validate a hypothesis, i.e. *to reduce risks*.

An **exploratory prototype**, also known as a **throwaway prototype**, is intended to **validate requirements** or **explore design choices**.

- UI prototype — validate user requirements
- rapid prototype — validate functional requirements
- experimental prototype — validate technical feasibility
Prototyping (II)

An evolutionary prototype is intended to evolve in steps into a finished product.

> iteratively “grow” the application, redesigning and refactoring along the way

First do it, then do it right, then do it fast.
Design is the process of specifying how the specified system behaviour will be realized from software components. The results are architecture and detailed design documents.

Object-oriented design delivers models that describe:
— how system operations are implemented by interacting objects
— how classes refer to one another and how they are related by inheritance
— attributes and operations associated to classes

Design is an iterative process, proceeding in parallel with implementation!
Conway’s Law

—“Organizations that design systems are constrained to produce designs that are copies of the communication structures of these organizations”
Implementation and Testing

**Implementation** is the activity of constructing a software solution to the customer’s requirements.

**Testing** is the process of validating that the solution meets the requirements.

—The result of implementation and testing is a fully documented and validated solution.
Design, implementation and testing are iterative activities

— The implementation does not “implement the design”, but rather the design document documents the implementation!

> System tests reflect the requirements specification

> Testing and implementation go hand-in-hand

— Ideally, test case specification precedes design and implementation
Maintenance is the process of changing a system after it has been deployed.

- **Corrective maintenance**: identifying and repairing defects
- **Adaptive maintenance**: adapting the existing solution to new platforms
- **Perfective maintenance**: implementing new requirements

In a spiral lifecycle, everything after the delivery and deployment of the first prototype can be considered “maintenance”!
Maintenance activities

“Maintenance” entails:
> configuration and version management
> reengineering (redesigning and refactoring)
> updating all analysis, design and user documentation

Repeatable, automated tests enable evolution and refactoring
“Maintenance” typically accounts for **70% of software costs**!

*Means*: most project costs concern continued development *after* deployment

– Lientz 1979
Roadmap

- Course Overview
- What is Software Engineering?
- The Iterative Development Lifecycle
- Software Development Activities
- Methods and Methodologies
**Methods and Methodologies**

*Principle* = general statement describing desirable properties  
*Method* = general guidelines governing some activity  
*Technique* = more technical and mechanical than method  
*Methodology* = package of methods and techniques packaged

---

*Ghezzi et al. 1991*
Object-Oriented Methods: a brief history

> **First generation:**
  — Adaptation of existing notations (ER diagrams, state diagrams ...): Booch, OMT, Shlaer and Mellor, ...
  — Specialized design techniques:
    - *CRC cards; responsibility-driven design; design by contract*

> **Second generation:**
  — Fusion: Booch + OMT + CRC + formal methods

> **Third generation:**
  — Unified Modeling Language:
    - *uniform notation: Booch + OMT + Use Cases + ...*
    - *various UML-based methods (e.g. Catalysis)*

> **Agile methods:**
  — Extreme Programming
  — Test-Driven Development
  — Scrum …
What you should know!

> How does Software Engineering differ from programming?
> Why is the “waterfall” model unrealistic?
> What is the difference between analysis and design?
> Why plan to iterate? Why develop incrementally?
> Why is programming only a small part of the cost of a “real” software project?
> What are the key advantages and disadvantages of object-oriented methods?
Can you answer these questions?

> What is the appeal of the “waterfall” model?
> Why do requirements change?
> How can you validate that an analysis model captures users’ real needs?
> When does analysis stop and design start?
> When can implementation start?
> What are good examples of Conway’s Law in action?
Attribution-ShareAlike 3.0

You are free:

▪ to copy, distribute, display, and perform the work
▪ to make derivative works
▪ to make commercial use of the work

Under the following conditions:

**BY:** Attribution. You must attribute the work in the manner specified by the author or licensor.

**Share Alike.** If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

▪ For any reuse or distribution, you must make clear to others the license terms of this work.
▪ Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

http://creativecommons.org/licenses/by-sa/3.0/