Introduction to Software Engineering (ESE : Einführung in SE)

Mircea F. Lungu

Based on a lecture by Oscar Nierstrasz. Selected material courtesy of Serge Demeyer.
# ESE — Introduction

| **Lecturer** | Mircea F. Lungu  
| | scg.unibe.ch/staff/mircea |
| **Assistants** | Andrea Caracciolo  
| | Andrei Chiş, Bledar Aga |
| **Lectures** | Engehaldenstrasse 8, 001,  
| | Wednesdays @ 13h15-15h00 |
| **Exercises** | Engehaldenstrasse 8, 001  
| | Wednesdays @ 15h00-16h00 |
| **WWW** | scg.unibe.ch/teaching/ese |
Roadmap

> Course Overview
> What is Software Engineering?
> Development Lifecycles
> Software Development Activities
> Methods and Methodologies
Roadmap

> Course Overview
> What is Software Engineering?
> Development Lifecycles
> Software Development Activities
> Methods and Methodologies


Recommended Literature


<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>18 Sep</td>
<td>Introduction</td>
</tr>
<tr>
<td>25 Sep</td>
<td>Requirements Collection</td>
</tr>
<tr>
<td>2 Oct</td>
<td>Guest Lecture: Processes &amp; Scrum</td>
</tr>
<tr>
<td>9 Oct</td>
<td>Usability and UI</td>
</tr>
<tr>
<td>16 Oct</td>
<td>Architecture</td>
</tr>
<tr>
<td>23 Oct</td>
<td>Good Software Design</td>
</tr>
<tr>
<td>30 Oct</td>
<td>Validation and Testing</td>
</tr>
<tr>
<td>6 Nov</td>
<td>Documenting Design with UML</td>
</tr>
<tr>
<td>13 Nov</td>
<td>Software Quality</td>
</tr>
<tr>
<td>20 Nov</td>
<td>Software Assessment Tutorial</td>
</tr>
<tr>
<td>27 Nov</td>
<td>Software Evolution</td>
</tr>
<tr>
<td>4 Dec</td>
<td>Empirical Software Engineering</td>
</tr>
<tr>
<td>11 Dec</td>
<td>TBD</td>
</tr>
<tr>
<td>18 Dec</td>
<td>Project showcase</td>
</tr>
<tr>
<td>7 Jan</td>
<td>Final Exam</td>
</tr>
</tbody>
</table>
> **Goal**: Android application
> **Customers**: ESE Staff
> **Organization**: teams
> **Target**: The Android Market
> **Deadline**: December 18
> **Worth**: 40% of final grade
Grading

Final Grade = 0.4 * Project + 0.6 * Exam

Project =
  0.1 * Cross-Project Code Review +
  0.4 * Code/Design Quality & Documentation +
  0.4 * Functionality & Usability (lack of bugs, richness of features) +
  0.1 * Final Presentation
Roadmap

> Course Overview
> What is Software Engineering?
> Development Lifecycles
> Software Development Activities
> Methods and Methodologies
Why Software Engineering?

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?
What is Software Engineering?

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
What is Software Engineering? (Def: I)

State of the art of developing quality software on time and within budget

(Anonymous)

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? (Def: 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
What is Software Engineering? (Def: III)

Everything else, besides programming, that contributes to building software systems. *Plus programming.*

This definition is mine, not of a famous person. But it is important to accentuate the fact that a successful system is a function of much more than programming.

Every one of you should be able to implement Facebook by the end of your bachelor. The reason why there is only one Facebook is everything else they did right besides programming. Plus luck.
The relationship between software engineering and computer science is the same as that between mechanical engineering disciplines and physics. One supports the other. Only that the difference between CS and SE is much smaller.

Roadmap

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

Called the “waterfall model” by Royce in 1970.

Software Development Activities

- Requirements Collection  Establish customer’s needs
- Analysis  Model and specify the requirements (“what”)
- Design  Model and specify a solution (“how”)
- Implementation  Construct a solution in software
- Testing  Validate the solution against the requirements
- Maintenance  Repair defects and adapt the solution to new requirements

The classical software lifecycle models the software development as a step-by-step “waterfall” between the various development phases.
Problems with the Waterfall Lifecycle

The waterfall model is often unrealistic for many reasons:

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

“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.”

“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

Requirements must be frozen too early in the life-cycle and the implementation is validated too late

However, this depends on the type of the application. When one is implementing the Mars rover, things might be different.
Iterative Development

How do you plan the number of iterations? How do you decide on completion?

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

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.

Show your prototype early to the user. Andrew Begel: “After the first minute of watching somebody use my tool, I know a lot of things I have to fix”.
Plan to *incrementally* develop (i.e., prototype) the system

Proposed by Millis in 1980.

Plan to *incrementally* develop (i.e., prototype) the system.

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.

Continuous integration is the keyword here.
Nowadays we do not have the same model as 50 years ago. We do not ship software in boxes. The current approach to deployment has changed. Systems like GitHub and Facebook are notorious for continuously deploying new features.

How can you be sure that your small change will not interfere with other features?

- automatic testing
- testing with small groups of users then A/B testing
Roadmap

- Course Overview
- What is Software Engineering?
- Development Lifecycles
- **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.

“Verba volant, scripta manent”. Writing remains, and helps us think.
Validation is needed throughout system evolution

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. A very good example of analysis is present in the Hyperloop documents leaked by Elon Musk. There one can see that the problem has been studied in detail, and a solution has been outlined only in order to conduct further discussions.

The result of analysis is a specification document.

Beware: what is not specified, remains at the attribution of the individual developer. And sometimes this is not what you want.

Specification is even more important when you develop software for yourself.
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

Image: [http://www.youtube.com/watch?v=TeS_U9qFq7Y](http://www.youtube.com/watch?v=TeS_U9qFq7Y)
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

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

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.

Testing and implementation go hand-in-hand
Ideally, test case specification precedes design and implementation

The last 20% of the time takes 80% of the effort.

http://blog.treefortbikes.com/index.php/posts/1327
Maintenance is the process of changing a system after it has been deployed.

Repeatable, automated tests enable evolution and refactoring

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

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

- Course Overview
- What is Software Engineering?
- Development Lifecycles
- 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

— Ghezzi et al. 1991
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 or computer science?
> Why is the “waterfall” model unrealistic?
> Why plan to iterate? Why develop incrementally?
> Why is programming only a small part of the cost of a software project?
Can you answer these questions?

> Why do requirements change?
> How can you validate that an analysis model captures users’ real needs?
> When can implementation start?
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/

All images that are not otherwise attributed, are from wikipedia.