Introduction — CS-397/8: Software Systems Development

Dr. Marouane Kessentini
Department of Computer Science
• **Instructor:**  Dr. Marouane Kessentini  
  – 324 Computer Science Bldg. (office)  
  – Office Hours: MF 11:00 – 12:00 or by appt.  
  – Office phone: 573-341-6371  
  – Email: marouanek@mst.edu
References

• **Course Webpage:** All material in this syllabus plus supplemental lecture material can be found at the following URL:
  
  http://web.mst.edu/~marouanek/classes/cs397

• **References:**


• **Course Prerequisites:** Comp Sci 206.

• **Grade Scale:**
  - A: 100-90,
  - B: 89-80,
  - C: 79-70,
  - D: 69-60,
  - F: 59-0.
Assignments and Project(s):

- **Presentations**: 20% (graded individually)
  - Mid-semester: 10%
  - End of semester: 10%
- **Monthly reports**: 15% (graded individually) - 3 reports (5% each)
- **Project (for each group)**: 65%
  - CV (resume): 5% (graded individually)
  - Project proposal: 10%
  - Requirements analysis: 15%
  - Planning: 5%
  - Project design: 15%
  - Coding and testing: 15%
<table>
<thead>
<tr>
<th>Date</th>
<th>Lecture Topics</th>
<th>Assignment/Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>08/23/2012;</td>
<td>Software engineering overview</td>
<td>CV and CL(up-to 4 pages) Due date : 08/30/2012</td>
</tr>
<tr>
<td>08/30/2012; 09/06/2012</td>
<td>UML: use cases, sequence diagram, planning, class diagrams, etc.</td>
<td>Project proposal Due date : 09/06/2012</td>
</tr>
<tr>
<td>09/13/2012;</td>
<td>Meetings/discussions about software requirements</td>
<td>Requirements analysis Monthly report Due date : 09/27/2012</td>
</tr>
<tr>
<td>09/20/2012</td>
<td>No lecture</td>
<td></td>
</tr>
<tr>
<td>09/27/2012</td>
<td>Meetings/discussions about projects planning</td>
<td>Project planning Due date : 10/11/2012</td>
</tr>
<tr>
<td>10/04/2012</td>
<td>Meetings/discussions about projects design</td>
<td>Project design Mid-semester presentations Monthly report Due date : 10/25/2012</td>
</tr>
<tr>
<td>10/11/2012; 10/18/2012;</td>
<td>Meetings/discussions about projects design</td>
<td>Project coding and testing Monthly report Final presentations Due date : 11/29/2012</td>
</tr>
<tr>
<td>10/25/2012;</td>
<td>Mid-semester presentations</td>
<td></td>
</tr>
<tr>
<td>11/01/2012;</td>
<td>No lecture</td>
<td></td>
</tr>
<tr>
<td>11/08/2012</td>
<td>Meetings/discussions about projects coding and testing</td>
<td></td>
</tr>
<tr>
<td>11/15/2012</td>
<td>No lecture</td>
<td></td>
</tr>
<tr>
<td>11/22/2012</td>
<td>Meetings/discussions about projects coding and testing</td>
<td></td>
</tr>
<tr>
<td>11/29/2012</td>
<td>Final presentations</td>
<td></td>
</tr>
<tr>
<td>12/06/2012</td>
<td>Final presentations</td>
<td></td>
</tr>
</tbody>
</table>
Roadmap

- Course Overview
- What is Software Engineering?
- The Iterative Development Lifecycle
- Software Development Activities
Why Software Engineering?

A naive view:

*coding*

Problem Specification $\rightarrow$ 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? (I)

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
Roadmap

- Course Overview
- What is Software Engineering?
- The Iterative Development Lifecycle
- Software Development Activities
# Software Development Activities

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

• Course Overview
• What is Software Engineering?
• The Iterative Development Lifecycle
• Software Development Activities
Requirements Collection

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
Requirements Analysis and Specification

**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?*
Design

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!
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.
Maintenance

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.