CS 520
Theory and Practice of Software Engineering
Fall 2017

Course introduction

September 05, 2017
The CS 520 team

Instructors

● Prof. Yuriy Brun
  ● Office: CS 346
  ● Office hours: by appointment
  ● brun@cs.umass.edu

● Prof. René Just
  ● Office: CS 358
  ● Office hours: Thursdays 12:30pm -- 1:30pm, by appointment
  ● rjust@cs.umass.edu

Teaching assistant

● Ben Kushigian
  ● Office: TBD
  ● Office hours: TBD
  ● bkushigian@umass.edu
Today

- What is Software Engineering?
- Why is Software Engineering important?
- Your expectations
- Course overview
- Our expectations
- Logistics
What is Software Engineering?
What is Software Engineering?

- Developing in an IDE and software ecosystem?
- Coding and debugging?
- Deploying and running a software system?
- Empirical evaluations?
- Modeling and designing?
What is Software Engineering?

- Developing in an IDE and software ecosystem?
- Coding and debugging?
- Deploying and running a software system?
- Empirical evaluations?
- Modeling and designing?

All of the above -- much more than just writing code!
What is Software Engineering?

More than just writing code
The complete process of specifying, designing, developing, analyzing, deploying, and maintaining a software system.

- Common Software Engineering tasks include:
  - Requirements engineering
  - Specification writing and documentation
  - Software architecture and design
  - Programming
  - Software testing and debugging
  - Refactoring
What is Software Engineering?

More than just writing code
The complete process of specifying, designing, developing, analyzing, deploying, and maintaining a software system.

- Common Software Engineering tasks include:
  - Requirements engineering
  - Specification writing and documentation
  - Software architecture and design
  - Programming
  - Software testing and debugging
  - Refactoring

Just one out of many important tasks!
What is Software Engineering?

More than just writing code
The complete process of specifying, designing, developing, analyzing, deploying, and maintaining a software system.

- Common Software Engineering tasks include:
  - Requirements engineering
  - Specification writing and documentation
  - Software architecture and design
  - Programming
  - Software testing and debugging
  - Refactoring

Why is Software Engineering important?
Why is Software Engineering important?

Software is everywhere...
Why is Software Engineering important?

Software is everywhere...and buggy!
Why is Software Engineering important?

Software is everywhere...and buggy!

Unfortunately, WhatsApp has stopped.
Why is Software Engineering important?

Software is complex!

- Aircraft: ~15 million lines of code
Why is Software Engineering important?

Software is complex!

- Aircraft: ~15 million lines of code
- Let's say 50 lines per page (0.05 mm)
- 300000 pages
- 15 m (49 ft)
Why is Software Engineering important?

Infrastructure is software, too!

Example: Design space exploration

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.34</td>
<td>0.81</td>
</tr>
<tr>
<td>2</td>
<td>0.52</td>
<td>0.32</td>
</tr>
<tr>
<td>3</td>
<td>0.21</td>
<td>0.53</td>
</tr>
<tr>
<td>4</td>
<td>0.81</td>
<td>0.22</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Why is Software Engineering important?

Infrastructure is software, too!

Example: Design space exploration

- 150 configurations, 1000+ benchmarks
- 1-85 hours per execution
- 200,000+ CPU hours (~23 CPU years)
Summary: Software Engineering

What is Software Engineering?
The complete process of specifying, designing, developing, analyzing, deploying, and maintaining a software system.

Why is it important?
● Software is everywhere and complex.
● Software defects are expensive (and annoying).

Goals
● Decompose a complex engineering problem.
● Organize processes and effort.
● Improve software reliability.
● Improve developer productivity.
Your expectations

Introduction and a brief (5 minute) survey

- Why are you taking this course?
- What do you expect from this course?
- What are your learning goals (theory and practice)?
Course overview: the big picture

- **Software architecture and design**
  - Software modelling and UML crash course.
  - Best practices and OO design principles.
  - Architecture and Design patterns.
  - Very brief intro to functional programming.

Goal: no more spaghetti code!
Course overview: the big picture

- **Software architecture and design**
  - Software modelling and UML crash course.
  - Best practices and OO design principles.
  - Architecture and Design patterns.
  - Very brief intro to functional programming.

- **Empirical Software Engineering**
  - Reasoning about experimental designs and studies.
  - Understanding and reasoning about threats to validity.

Anything wrong with the following conclusions?
- **Not using Internet Explorer** makes the world a safer place/*reduces murder rates*.
- **Spending more time on learning** a programming language *makes* you a *worse programmer*.

Goal: properly reason about research studies and findings
Course overview: the big picture

● **Software architecture and design**
  ○ Software modelling and UML crash course.
  ○ Best practices and OO design principles.
  ○ Architecture and Design patterns.
  ○ Very brief intro to functional programming.

● **Empirical Software Engineering**
  ○ Reasoning about experimental designs and studies.
  ○ Understanding and reasoning about threats to validity.

● **Software testing, debugging, and repair**
  ○ Learning about cutting-edge research.
  ○ Hands-on experience, using testing and debugging techniques.

● **Class project**
  ○ Design, development, and testing of a research prototype, etc.
Course overview: the big picture

- **Software architecture and design**
  - Software modelling and UML crash course.
  - Best practices and OO design principles.
  - Architecture and Design patterns.
  - Very brief intro to functional programming.
  - 1 homework assignment
  - 1 in-class exercise

- **Empirical Software Engineering**
  - Reasoning about experimental designs and studies.
  - Understanding and reasoning about threats to validity.
  - 2 paper reviews
  - 2 opportunities for extra credit

- **Software testing, debugging, and repair**
  - Learning about cutting-edge research.
  - Hands-on experience, using testing and debugging techniques.
  - 3 in-class exercises

- **Class project**
  - 2-month group project
  - Design, development, and testing of a research prototype, etc.
Course overview: rough timeline

September
- Software architecture and design

October
- Empirical Software Engineering
- Software testing
- Class project

November
- Software debugging and repair
- Collaboration and teamwork
- Class project

December
- Reasoning about programs
- Class project
Course overview: grading

Grading

● 30% Class project
● 30% In-class exercises
● 30% Homework and paper reviews
● 10% Participation
Course overview: grading

Grading

- 30% Class project
- 30% In-class exercises
- 30% Homework and paper reviews
- 10% Participation

Questions?
Our expectations

● Programming experience.
● Familiarity with one OO programming language (Java, C++, ...).
● Reading and reviewing 2 research papers.
● Active participation in discussions and group work.
Logistics

- LGRT 123, Tu/Th, 10am – 11:15am.
- Lectures, tutorials, and in-class exercises.
- Course material, policies, and schedule on web site: http://people.cs.umass.edu/~rjust/courses/2017Fall/CS520
- Submission of assignments via Moodle: https://moodle.umass.edu