CS 520

Theory and Practice of Software Engineering
Fall 2018

Course introduction

September 04, 2018

The CS 520 team

Instructor

- Yuriy Brun
- Office: CS 346 (for now)
- Office hours: Tuesdays 11:15-12:00
- brun@cs.umass.edu

Teaching assistant

- Manish Motwani
- Office: TBD
- Office hours: TBD
- mmotwani@cs.umass.edu

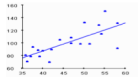
Today

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

What is Software Engineering?

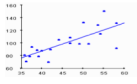
What is Software Engineering?

- 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 — Just one out of many important tasks!
  - 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

Why is Software Engineering important?

Why is Software Engineering important?

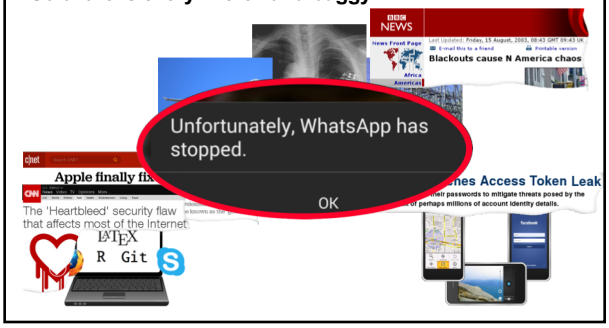
Software is everywhere...

Why is Software Engineering important?

Software is everywhere...and buggy!

Why is Software Engineering important?

Software is everywhere...and buggy!



Why is Software Engineering important?

Software is complex!

- Aircraft: ~15 million lines of code

How complex is software?

- Measures of complexity:
  - lines of code
  - number of classes
  - number of modules
  - module interconnections and dependencies
  - time to understand
  - # of authors
  - … many more

How complex is software?

- Measures of complexity:
  - lines of code — Windows Server 2003: 50 MSLoC; Debian 5.0: 324 MSLoC
  - number of classes
  - number of modules
  - module interconnections and dependencies
  - time to understand
  - # of authors
  - … many more

How big is 324 MSLoC?

- 50 lines/page ⇒ 6.5M pages
- 1K pages/ream ⇒ 6.5K reams
- 2 inches/ream ⇒ 13K inches
- 13K inches ≈ four times the height of this building

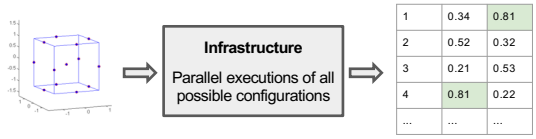
- 5 words/LoC @ 50 wpm ⇒ 32M min ≈ 61 years

And we don't just want random words, we want compiling code!

Why is Software Engineering important?

Infrastructure is software, too!

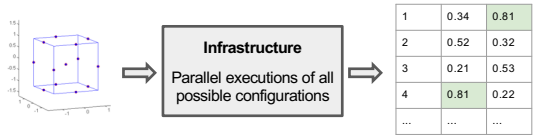
Example: Design space exploration



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.

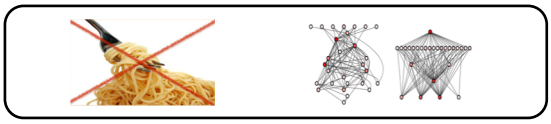
Your expectations

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.



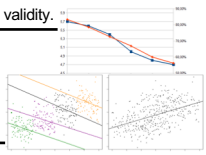
Goal: no more spaghetti code!

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** 1 homework
  - Software modelling and UML crash course. 1 in-class exercise
  - Best practices and OO design principles.
  - Architecture and Design patterns.
  - Very brief intro to functional programming.
- **Empirical Software Engineering** 2 paper reviews
  - Reasoning about experimental designs and studies.
  - Understanding and reasoning about threats to validity.
- **Software testing, debugging, and repair** 3 in-class exercises
  - Learning about cutting-edge research. 1 homework
  - Hands-on experience, using testing and debugging techniques.
- **Class project** 2-month group project
  - Design, development, and testing of a research prototype, etc.

opportunities for extra credit

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

Exposure to cutting-edge research

- We will have 4 guest lectures on research
  - These will be held out of class, most likely at 4PM. Videos will be available.
- We might have 1 guest lecture on what it's like to work in industry.

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 an OO programming language (e.g., Java, C++, etc.)
- Reading and reviewing 2 research papers.
- Active participation in discussions and group work.

Logistics

- ISB 221, Tuesday and Thursday, 10am – 11:15am.
- Lectures, tutorials, and in-class exercises.
- Course material, policies, and schedule on web site: http://people.cs.umass.edu/~brun/class/CS520/
- Submission of assignments via Moodle: https://moodle.umass.edu/course/view.php?id=49403