COMP1023 Software Engineering

Spring Semester 2019-2020

Dr. Radu Muschevici

School of Computer Science, University of Nottingham, Malaysia

Lecture 1
2020-02-05

Welcome to Software Engineering

Q: How many software engineers
does it take to change a light bulb?

Welcome to Software Engineering

Q: How many software engineers does it take to change a light bulb?

A: None, that's a hardware problem.

Welcome to Software Engineering

Q: How many software engineers
does it take to change a light bulb?

A: None, that's a hardware problem.

Note: This is an old joke. Nowadays, more and more light bulbs include software! (think "smart" home appliances.)

Course Infrastructure

- **Lecturer:**
  Radu Muschevici, Ph.D.

  Email: radu.muschevici@nottingham.edu.my
  Office: BB03

- **Lecture:** Wednesdays 9.00–11.00, Room F3A08

- **Computing Lab:**
  - starting on the second teaching week (i.e., Week 23)
  - Tuesdays 9.00–11.00, Room F2TCR1
  - **or** Wednesdays 13.00–15.00, Room F2TCR2

Course Infrastructure (cont.)

- **Course website (Moodle)**

  `https://moodle.nottingham.ac.uk/course/view.php?id=60322`

  Course materials, lecture recordings, discussion forum …

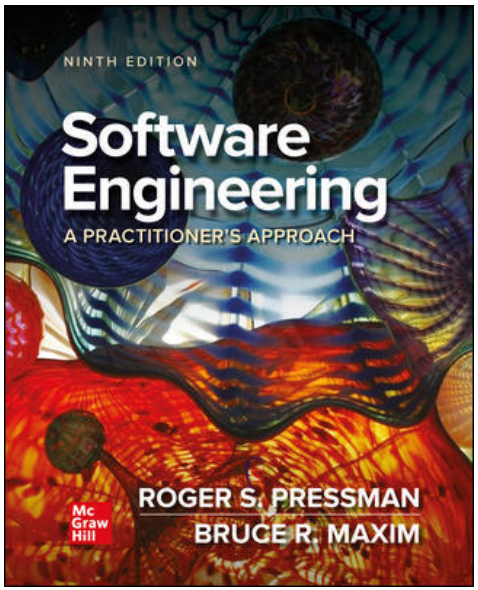
- **Lecture slides** will be made available before each lecture (usually in the same week on Monday).

- **Assessment**
  - Practical assignment based on weekly lab sessions: 50%
  - Written exam (1 hour): 50%

Recommended Books

Ian Sommerville, **Software Engineering** 10th ed., 2015, Pearson.

The course is based on this book.



Pressman and Maxim, **Software Engineering: A Practitioner's Approach**, 9th ed., 2019, McGraw-Hill.

Course Topics Overview

1. Introduction
2. Software processes
3. Agile software development
4. Requirements engineering
5. System modelling
6. Architectural design
7. Design and implementation
8. Software testing
9. Software evolution

Course Objectives

You will learn about…

- Phases of the **software lifecycle**:
  Requirements, Specification, Software Design, Implementation, Testing, Maintenance, Evolution
- Software development methodologies: **Waterfall**, **Agile**

You will gain…

- A general **understanding** of Software Engineering
- Ability to apply Software Engineering methodologies **in practice**
- Skills to master the various software development projects undertaken throughout your studies and beyond (i.e. your professional life).

Software Engineering
– an Introduction –

What is Software Engineering?

Software Engineering (SE) is an **engineering** discipline that is concerned with all aspects of **software** production.

[Sommerville 2015]

Engineers make things work.

- Apply appropriate theories, methods, and tools.
  — **knowledge & skills**
- Look for solutions to problems even when there are no applicable theories and methods. — **curiosity & creativity**
- Must work within organizational and financial constraints.
  — **pragmatism, realism**

Programming vs. Software Engineering

What does Programming involve?

- Knowing (the syntax and semantics of) a programming language
- Designing algorithms
- Structuring code and data
- Writing code
- Making the code work

Programming vs. Software Engineering

What does Software Engineering involve?

- Finding out precisely what your customer needs.
- Finding the best solution to the problem (most efficient, simple, cheap…)
- Producing the software, i.e. **programming**.
- Delivering the software to your customer within the agreed time frame and at the agreed price.
- Ensuring the software runs without errors and failure.
- Ensuring the software solves the customer's problem.
- Documenting how to use/operate the software.
- Changing and extending the software as your customer's needs change and grow.
- Making sure the software works for as long as intended (years—decades).

Computer Science vs. Software Engineering

Computer Science (CS)

- is concerned with the theories and methods that underlie computers and software systems.

Software Engineering (SE)

- is concerned with the practical problems of producing software.
- SE $\subset$ CS

System Engineering vs. Software Engineering

System Engineering is concerned with the development and evolution of complex systems (where software plays a major role):

- hardware development,
- policy and process design,
- system deployment,
- **software engineering**.

Learning Outcomes – Knowledge and Understanding

What you should know by the end

- The different approaches to managing the software engineering process.
- The practice of producing specifications from informal briefs.

In other words…

- Software processes (Waterfall, incremental) and their activities (specification, design, implementation, validation, evolution)
- Requirements engineering practice

Learning Outcomes – Intellectual Skills

What you should know by the end

- Understand how to determine formal software requirements.
- Understand how to create and deploy an effective plan for testing software systems.

In other words…

- Requirements elicitation, analysis, specification, validation
- Know how to make sure the requirements are met. Various kinds of testing (component, system, customer).

Learning Outcomes – Professional Skills

What you should know by the end

The ability to …

- apply software engineering methodologies in practical scenarios.
- understand how good software is closely related to the needs of users.
- evaluate, select and deploy appropriate tools and techniques.

In other words…

- Know the differences between SE processes and how to choose & use the best approach for each scenario.
- Learn to talk to and understand people who have a different background/perspective than yours.
- Have the skills to use a wide range of SE methods and tools.

Sumary of Content

What we will cover

- Software development methodologies and the software lifecycle (Waterfall model, Agile).
- Formal requirements and specification — how to turn an informal design brief into a formal specification.
- Software testing, evaluation and debugging — including practical use of modern debugging toolkits.
- Software evolution and maintenance — including version control and collaborative development systems.

Putting things in perspective

Writing software for a **study assignment** is – in many respects – different from industrial software engineering practice:

- You have to work on a piece of code only once.
- Maximum size of your code: a few thousand lines
- Your errors are unlikely to cost anyone money.
- Your bugs are unlikely to kill anyone.
- ~~You can sit down on your own and do your thing.~~

Putting things in perspective

In industry, working as a software engineer means:

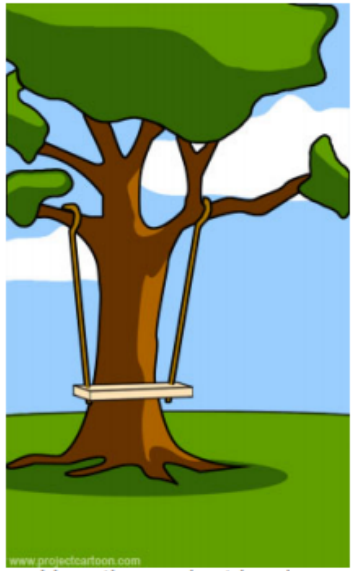
- You have to work on piece of code **many times**.
- Enterprise code can be many **millions of lines**.
- Your errors will cost your company **money**.
- Your bugs may **kill** your users.
- You will be expected to work as a **team**.

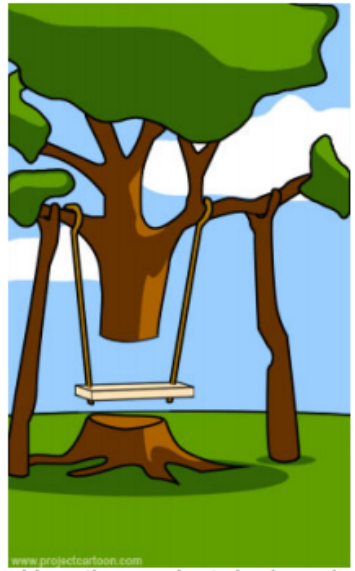
Why you should care about this?

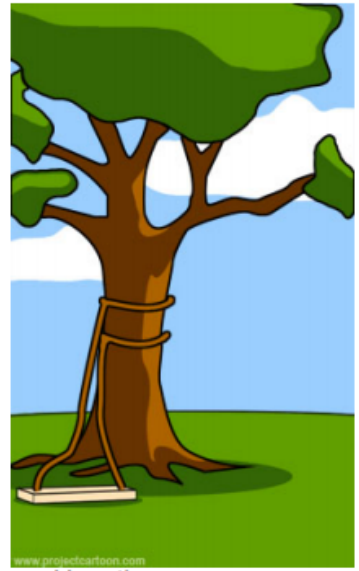
- Majority of graduates get job as software engineers.
- A lot of software projects are failures; we would like to do better in future.
- The principles taught in this course apply to large systems, much bigger than anything you will write for your degree.
- Essential module for job seekers as well as job creators (entrepreneurs).

What we want to avoid…

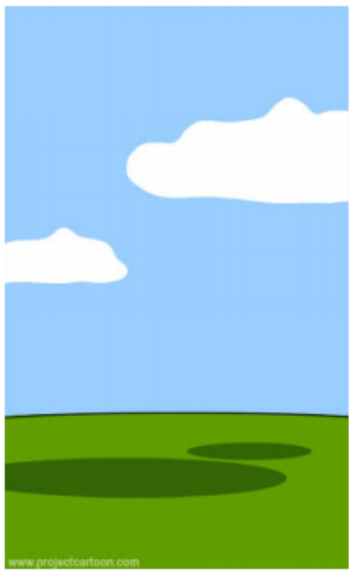

How the customer explained it

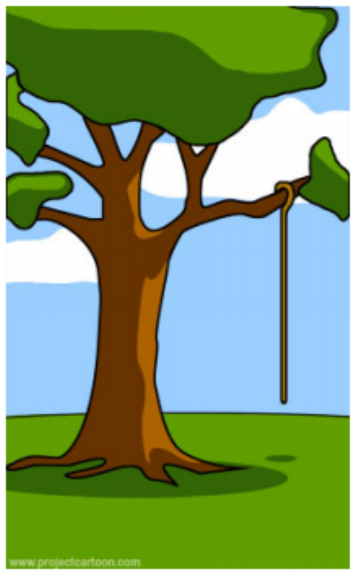

How the project leader understood it

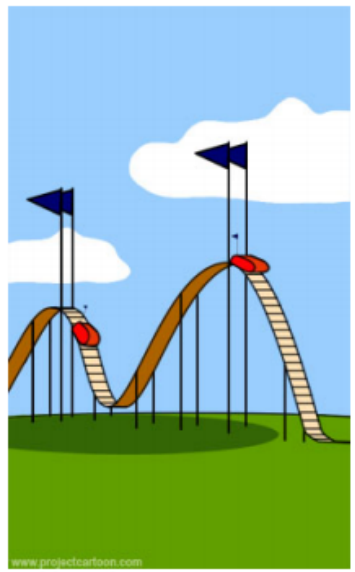

How the analyst designed it

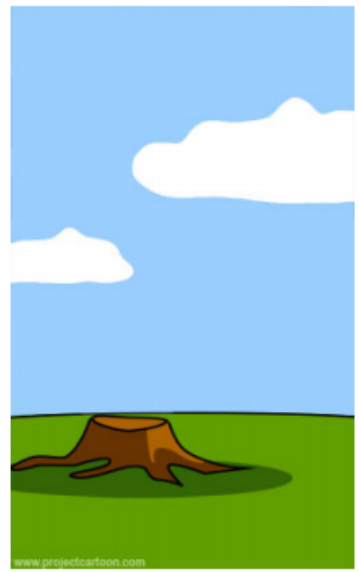

How the programmer wrote it

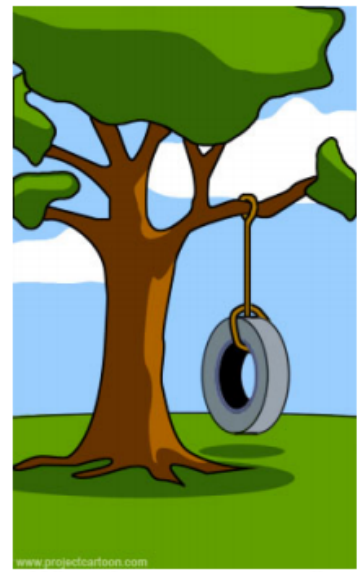

How the business consultant described it


How the project was documented


What operations installed


How the customer was billed


How it was supported


What the customer really needed

Examples of Software Engineering failures

The Ariane 5 rocket Flight 501

Ariane 5 flight 501 was launched on Tuesday, 4 June 1996, and exploded a few seconds later. The fault was identified as an improper type cast in the rocket's Inertial Reference System. There were two bits of code. One that measured the sideways velocity, and one that used it in the rocket's guidance system. The measurement side used a 64 bit variable, but the guidance side used a 16 bit variable. The code was borrowed from an earlier, slower rocket whose velocity would never grow large enough to exceed 16 bits.

The disastrous launch **cost approximately USD 370 million**, and through the destruction of the rocket's payload, **delayed scientific research** into workings of the Earth's magnetosphere for almost 4 years.

- Wikipedia
- Video by Ian Sommerville

Examples of Software Engineering failures

Many more…

- History's Worst Software Bugs (2005 Wired article)
- 20 Famous Software Disasters

In Yesterday's News…

The New York Times

App Used to Tabulate Votes Is Said to Have Been Inadequately Tested

The app was quickly put together in the past two months and was not properly tested at a statewide scale, according to people briefed on the matter.

Until next lecture

- Obtain the **Software Enginnering** book by Sommerville
- Read/skim Chapter 1 – Introduction (24 pages)

The End
(of the Beginning)

References I

Sommerville, Ian (2015). **Software Engineering**. 10th ed. Pearson. ISBN: 0133943038.