Principles of Software Engineering: Course Outline

Ethan Jackson And Wolfram Schulte,
Research in Software Engineering (RiSE)
Microsoft Research
Overview

- Motivation and Focus
- Syllabus
- Projects
i. Motivation and Focus

http://www.cs.washington.edu/csep503
Motivation

Software engineering is more important than ever, from many points-of-view:

- **Business point-of-view:** Need to sell software with few bugs in the face of ever shorter release cycles. Badly engineered software is counter-productive.

- **Consumer point-of-view:** Software should perform its functions quickly, correctly, securely, privately, using little power... and the list continues to grow.

- **Societal point-of-view:** Software helps to drive our cars, monitor our health, generate our power. How can we engineer software that lives up to these applications?
This class will be a journey through state-of-the-art techniques in software engineering.
Focus: Engineering Correct Software

Our journey has a focus.
Engineering correct software that is:

- Likely to be correct, for some properties, by extensive automated testing.

- Provably correct, for some properties, by automated deduction.

- Provably correct, for some properties, by automated synthesis.
Our journey has twists and turns.
We don’t know of a single technique that address all SE problems:

- **Programming in the Small:**
  How do we design a solitary algorithm without bugs?

- **Programming in the Large:**
  How do we orchestrate several concurrent software systems?

- **Programming in the Real:**
  How do we write software that controls a physical system?
Mission: Get to Gliese 581

Our journey has a mission.
Design a probe that travels to the nearest earth-size planet in habitable zone.

Probe must autonomously travel 20 light years, which translates to 8,645 bug-free years at 10 times fastest speed ever achieved.
ii. Syllabus

http://www.cs.washington.edu/csep503
Part 1: Programming in the Small

First two weeks focus on functionally correct sequential software modules using pre/post conditions (code contracts):

- **Class 1**: Static verification using code contracts and abstract interpretation. Experiment with the tool Clousot.

- **Class 2**: Full static verification using object/loop invariants and automated theorem proving. Experiment with the Dafny language.
Second two weeks focus on orchestrating concurrent systems using (timed) automata theory.

- **Class 3**: Automata-theoretic models - Untimed and timed automata. Build models of system orchestration using the tools SMV and Uppaal.

- **Class 4**: Specifying temporal properties with LTL, CTL, CTL*, and observers. Model checking to verify temporal properties.
Part 3: Programming in the Real

Third two weeks look at synthesizing systems from models.

- **Class 5**: Realizing system synthesis through code generation. Applying formal verification to code generators.

- **Class 6**: Design-space exploration of software/hardware architectures in the presence of resource constraints. Experiment for the FORMULA system.
Part 4: Simulating Systems

Fourth two weeks look at simulation before implementation.

- **Class 7**: Simulation of mixed-domain models. Problems of combining discrete/continuous systems. Experiment with Ptolemy II framework.

- **Class 8**: Low-cost prototyping by simulation of virtual hardware. Experiment with the Giano system.
Part 5: Empirical Software Engineering

Fifth two weeks look at empirical software engineering.

- **Class 9**: Approaches to bug predication. Applying bug data from other related projects to predict bugs new projects.

- **Class 10**: Using data analytics to make decision. Impact of organizational structure on bugs.
Projects, Grading, and Homework (I)

- **Tests**: None.

- **Projects**: Class is project based. There will be 4 two-week projects. Leaves one week of start-up time, and one week of slack in case more time is needed.

- **Groups**: Feel free to work in groups of two.
Projects, Grading, and Homework (II)

- **Time**: Expect to spend several hours to: (1) get a new tool up-and-running, (2) think through the problem, (3) solve the problem. Probably 6 – 8 hours a week is reasonable.

- **Grading**: Historically, this class focuses on the journey. A strong attempt at projects guarantees a high grade.

- **TA \ Labs**: There is no TA and not set lab times. The class is small enough that we can meet in a lab if that is helpful.
Most importantly, have Fun!
iii. Projects

http://www.cs.washington.edu/csep503
In 1995 the first extra-solar planet was definitely confirmed.
In 2009 the Kepler mission was launched to look for planets among 100,000 stars.

To date Kepler has identified over 2,326 planets.
Background (III)

Jan. 5th, 2012 (today) DARPA leaks that astronaut Mae Jemison will head the “100 Year Starship” project to develop a starship.
Project 1: Astrometrics Subsystem

Probes must determine their location in space without any help from earth.

- **Location in ICRF**: Probe must find its direction in the ICRF by matching observed radio-sources with a database of known radio-sources.

- **Design and Prove**: Write a subsystem that matches a region of the sky with a database of known markers to determine spacecraft orientation. Prove it correct.
Three probes must move in tandem separated by 1 AU. Devise a command-and-control system that preserves this requirement.

- **Command-and-control**: Model command-and-control system as a set of interacting timed automata. These handle the high-level operations of the probe (e.g. call the astrometrics subsystem).

- **Prove**: Specify temporal properties of the command-and-control system. Use Uppaal explicit state model checking to verify properties.
Project 3: Synthesize Probe System

Generate software and partition onto hardware.

- **Code generation**: Write a code generator that produces an implementation of the command-and-control-system from automata models.

- **System synthesis**: Specify software/hardware partitioning problem as a constraint system over resources and synthesize candidate architectures.
Project 4: System Simulation

Simulate behavior of synthesized system using Ptolemy II

- **Plant Model**: Build a simple model of probe dynamics using continuous-time models in Ptolemy II.

- **Hybrid Model**: Combine plant model with synthesized discrete-time system to simulate complete behavior of probes.
Project 0: Play with Code Contracts


3. Read the documentation and try some of the samples.
Thanks And Questions!

http://www.cs.washington.edu/csep503