CSCI2340: Software Engineering

Course Overview
This source provides an overview of particular topics relevant to software engineering with a strong emphasis on research directions and where the field is going. It is designed both to familiarize the student with current research directions and where the field is going, and to lay the foundations for students wishing to do research in the area.

The emphasis this year, which tracks my particular interests in the field, is on understanding future directions in software development. Programming has always been changing, in large part due to advances in software engineering. This can be seen in the scale of programs which tends to increase by an order of magnitude each decade. Software engineering has typically addressed questions of how to enable programmers to write code that is maintainable, of high quality, secure, flexible, adaptable, portable ... The question we will attempt to address in this course is what software engineering is doing for the next generation of programming and programmers.

Course Mechanics
The course will meet Monday from 3:00 to 5:30 in CIT 316. Classes will be a mixture of group discussions, paper presentations, and student project presentations. Grading will be based on the project as well as on presentations and class participation. Students who wish to only audit the course are expected to do paper presentations and partake in the group discussions.

Dr. Reiss will have office hours Monday from 10:00am to 12:00pm and Thursday from 12:00 to 2:00pm. TA hours will be arranged as needed.

Project
Much of the course will revolve around a student-defined project relevant to the course. Projects can be done individually or in small teams (<= 3). The goal of the project is to develop or explore a technology designed to help the programmer in some way. This can be something new; it can be something you’ve always thought you’d love to have; or it can duplicate existing work, validating or expanding on past efforts. It can target the programming language and/or integrated development environment of your choice. If possible, it should be designed to handle “real” programs (i.e. it should scale, handle multiple threads, and generally be something you would actually use). Note that it can be designed to use as a capstone, Sc.M. research, or even Ph.D. research project. It can also be something related to your research in other areas (e.g. a domain-specific language).

We will discuss directions and possibilities for projects in the first class. You can follow these directions or choose a project that is quite different as long as it fits the theme of the course. The project can be related or relevant to your other research if that is appropriate.
The project schedule is:

9/17/2018 Project Proposal
- A one-paragraph description of what you plan to do and why (hand in)
  - Who is working on this
  - What language, environment, etc. will be targeted
  - A set of 2-5 relevant technical readings that provide appropriate background
- A short (2-4 minute) in-class presentation

10/15/2018 Project Specifications
- Requirements document (hand in)
- Specifications document (hand in)
- A short (5-10) minute in-class presentation centering on background and directions

11/12/2018 Project Prototype
- Initial implementation expected by this time (relative incomplete?)
- Status report (hand in)
- Short in-class presentation (5 minutes)

12/3/2018 and 12/10/2018 Project Presentations
- Full presentation (20 minutes)
- Demonstration if possible

Classes
The first class (9/10) will involve a discussion of what software engineering can do to help the programmer. This will look at what tools should be provided to the programmer and what techniques should be developed to assist the programmer. The course assumes that everyone has a significant programming background and can provide some information from personal experience. (It is difficult to understand the purpose of software engineering without such a background.) We will have a general discussion where everyone can provide their thoughts and ideas. Then we will briefly look at a variety of possible technologies include program synthesis, code repositories, user interfaces for programmers, program security, collaboration tools, automatic error correction, program understanding, design checking, and automatic refactoring.

The second class (9/17) will involve the initial project presentations and then a group discussion of the five papers listed below on the topic of how programmers work. Each student should have read these papers and come prepared to discuss them. In particular, do you believe the conclusions, does it apply to you, and is the result actually valid.

Most of the remaining classes, when there are no project-related presentations, will consist of presentations of 5 papers followed by a discussion of the papers. A student will be chosen in advance
for each of the papers to be presented. If you want to work in teams on the presentation, that is acceptable, but this will not reduce the number of papers assigned to you. Each paper will have a 20 minute slot for the presentation. (You should plan on 15 minutes of presentation and 5 minutes of questions.) Once the presentations are over, we will have a group discussion of the papers and how they relate to one another. Each presenter (or team) should have questions on the other papers as a starting point for this discussion. Note that, especially for the more recent papers, you might need to do some additional reading and give a more general presentation to lay the necessary foundation. Within a class we will go through the chosen papers in chronological order.

In preparation for each class, each student should look at each of the 5 papers to get an understanding of what they are about. You should learn how to do a 10 minute ‘read’ of a technical paper to get sense of what it involves and decide whether it is worth spending more time on. (Note that this will probably take more than 10 minutes at first.)

After each class, each student who did not present should do a 1-3 page write-up that discusses what you got out of the papers, with a particular emphasis on what were the good (and bad) ideas, and what are the future directions in the area. This should be handed in before the next class.

Papers are available in PDF form from the course web site. There is no additional text book for the course. Note that the following list is subject to change.

Topics and Papers

9/17: How Programmers Work

- How effective developers investigate source code: an exploratory study
- How are Java software developers using the Eclipse IDE?
- An exploratory study of how developers seek, relate and collect relevant information during software maintenance tasks
- How we refactor, and how we know it
- Exploring exploratory programming

9/24: User Interfaces for Programming

- Using task context to improve programmer productivity
- Software development with code maps
- Code Bubbles: rethinking the user interface paradigm of integrated development environments
- Flower: navigating program flow in the IDE
- SEEDE: simultaneous execution and editing in a development environment

10/1: Using Code Repositories

- Sourcerer: mining and searching internet-scale software repositories
- Semantics-based code search
• Novel and applied algorithms in a search engine for Java code snippets
• Code relatives: detecting similarly behaving software
• FaCoY - a code-to-code search engine

10/8: NO CLASS

10/15: Project Specifications

10/22: Code Suggestion

• Jungloid mining: helping to navigate the API jungle
• Active code completion
• Augmenting API documentation with insights form Stack Overflow
• CSNIPPEX: automated synthesis of compilable code snippets form Q&A sites
• API code recommendation using statistical learning from fine-grained changes

10/29: Automatic Error Correction

• GenProg: a generic method for automatic software repair
• SEFMIX: program repair via semantic analysis
• Towards practical program repair with on-demand candidate generation
• Leveraging syntax-related code for automatic program repair
• Context-aware patch generation for better automated program repair

11/5: Memory Problems

• Exterminator: automatically correcting memory errors with high probability
• Object ownership profiling: a technique for finding and fixing memory leaks
• Precise memory leak detection for Java software using container profiling
• A dynamic approach to locating memory leaks
• A framework for automatic memory leak detection designed for identifying and analyzing memory leaks with its statistics

11/12: Lightweight Program Analysis (discussion only)

• FLAVERS: A finite state verification technique for software systems
• Finding bugs is easy
• Finding security vulnerabilities in Java applications with static analysis
• Checking event-based specifications in Java systems
• A lightweight code analysis and its role in evaluation of a dependability case

11/19: Program Verification and Checking Tools

• A static analyzer for finding dynamic programming errors
• Extended static checking for Java
• SLAM and static driver verifier: technology transfer of formal methods inside Microsoft
• Verification of object-oriented programs with invariants
• Stacy: static code analysis for enhanced vulnerability detection

11/26: Program Synthesis

• Model-based software synthesis
• Formal derivation of concurrent garbage collectors
• Oracle-guided component-based program synthesis
• RobustFill: neural program learning under noisy I/O
• Recent advances in neural program synthesis

12/3, 12/10: Project Presentations

Time Commitment and Grading
Brown courses involve a minimum of 180 hours of work over the course of the semester and this course is no different. Expect to spend at least 6+ hours a week (60 total) on the readings in preparation for the class and on preparing your summary; 20 hours preparing two 25 minute talks on papers; and 10 hours a week (120 total) working on your project.

Grading will be done accordingly, with 2/3 of the grade based on your project (which includes all the hand-ins and presentations) and 1/3 on your presentations, summaries, and in-class participation.

Diversity
Brown University is committed to full inclusion of all students. Please inform me early in the term if you have a disability or other conditions that might require accommodations or modification of any of these course procedures. You may speak with me after class or during office hours. For more information, please contact STUDENT AND EMPLOYEE ACCESSIBILITY SERVICES at 401-863-9588 or SEAS@brown.edu. Students in need of short-term academic advice or support can contact one of the deans in the Dean of the College office.

Collaboration
You are responsible for whatever work you hand in – it should be your work not someone else’s. Any outside references, quoted material, etc. should be properly cited. If you collaborate on a project or presentation, make clear who did what. Note that this includes code where copyrights and citations should also be included as appropriate.

As noted in the academic code: “A student’s name on any exercise (e.g., a theme, report, notebook, performance, computer program, course paper, quiz, or examination) is regarded as assurance that the
exercise is the result of the student’s own thoughts and study, stated in his or her own words, and produced without assistance, except as quotation marks, references, and footnotes acknowledge the use of printed sources or other outside help." (ACADEMIC CODE, p. 5)