Computer Science 50

Software Design and Implementation - or, how to be a hacker

In what follows, we discuss the course overview, grading, books, weekly schedule, laboratory assignments, group projects and class reading.

Course Overview

"How can I really call myself a CS major and not be a hacker?" - CS50 is your answer.

The ORC: Techniques for building large, reliable, maintainable, and understandable software systems. Topics include UNIX tools and filters, programming in C, software testing, debugging, and teamwork in software development. Concepts are reinforced through a small number of medium-scale programs and one team-programming project.

More specifically: The course teaches Linux, C and shell programming, the use of GNU development tools (gcc, gdb, make, valgrind, gprofile, cvs) and design and implementation techniques. As part of the course you will design and implement the TinySearchEngine as part of a series of programming assignments - then you can take on google! The course includes a group project based on programming embedded Linux robots. You will also be exposed to thread, socket and embedded systems programming.

Note, there is a significant amount of programming in this course requiring a significant time commitment on the part of the student.

The course breaks down into seven/eight weeks of lectures and labs with the remaining time held over for group projects. The projects will be organized around a design review, code review and demo or die event. Something similar to what you would experience in the software industry when you have to ship a beta. What you will learn

Some Linux, bash shell programming, C programming, programming tools, how to design, implement and debug a complex software system by working together in small groups. We hope that the skills and experience gained will reflect what you would expect in the software industry when working in small teams. The two main programming challenges are the TinySearchEngine specifically developed for this Dartmouth CS class - which represents a complex standalone software system - and real-time distributed client/server programming using the robots. You will learn about real-time distributed programming issues: you will programming embedded Linux robots (the Garcia below) and learn about the sensors that the robot interacts with as it roams around Sudikoff on a treasure hunt, searching for what? Yes, we like to race robots in Sudikoff.

This is a fun, exciting but challenging class. Note, there are no midterm or final exams - just design, programming, and a lot of debugging; you will learn all about segfaults, memory leaks, and core dumps. Also, we don't use any fancy IDEs (e.g., the popular Eclipse integrated development environment) in this class - everything is on the command line. We expose details to gain knowledge rather than promote abstraction.

In brief, C50 is the class I never had as an undergrad - it's a crash course in design, C/Unix programming, process/threads, GNU tools, socket programming and the implementation of complex standalone and embedded software systems - it's a course where you will write approximately 3000 lines of C and 500 lines of bash script for the programming assignments and1000 lines for the joint project; as a result, you will gain tangible programming skills; it will turn you into a systems hacker or your money back!

Grading

75% - Laboratory exercises

There are 7 weekly laboratory assignments over the first 8 weeks. These labs are designed to help you learn the languages, tools, and design skills you will need for your final project. Only six labs are graded. Some labs are harder than others and we grade as shown below. These assignments are to be done individually. The schedule is online - plan a head. You need to be organized to get through the labs and stay on schedule.

0% Lab1 - Shell Commands (Not graded)
10% Lab2 - Shell Programming
10% Lab3 - C Programming
15% Lab4 - Crawler
15% Lab5 - Indexer
15% Lab6 - Query Engine
10% Lab7 - Socket Programming

We will provide source code solutions to all labs. The TAs/section leaders will grade your solutions and write up a grade sheet on the correctness, simplicity, and clarity of your code. The instructor will review the TA's grading and grade sheets. Your grade and grade sheet and our source code solution will put in your svn repository (which is an open-source revision control system) one week after the lab assignment is turned in. If you have questions about the grade or grade sheet please talk to the TA/section leader first. If you are still have concerns see the instructor. Please do not distribute the source code solutions we provide you with (see honor code).

25% - Team project

The project is made up of a small team (three or four people) and requires strong collaboration and a problem solving mindset to get the job done. The instructor will put the teams together with each member being responsible to deliver against a part of the overall system design, implementation, testing and integration. The goals of this activity are to help you develop the confidence, skills, and habits necessary to write large computer programs while part of a multi-person team. You will become conversant in software engineering paradigms, and be exposed to various public-domain and open source tools that make the software development process easier. In addition, you will develop vital skills in self-directed learning, problem solving, and communication. The project will have a design and code review as well as the demo. A project report that captures the design and implementation will be submitted as part of the assessment.

Books

There is no one book that caters for the material in the course. However, we do recommend a book on the C language - this is a very good book:

A First Book of ANSI C, Fourth Edition by Gary J. Bronson

We do not recommend you buy any other books for the course. But If I were to recommend a book on design and programming for Unix and C it would be this classic text:

The Practice of Programming (Addison-Wesley Professional Computing Series) by Brian W. Kernighan, Rob Pike

If I where to recommend a hands on book on Linux and shell programming it would be this one (lots of good stuff in this):

A Practical Guide to Linux Commands, Editors, and Shell Programming by
Mark G. Sobell

Another really good book covering, debugging, processes, threads, and
socket programming in clear and easy manner to grasp:

Beginning Linux Programming, 4th Edition by Neil Matthew, Richard Stones

Just to make things clear. Buy the ANSI C book for the course. But if you are interested in hacking C, shell scripts, Linux you might like the other books for your library. They are all really nice books – I love them.

Schedule

We plan to use all x-hour periods for demos, recitations, quizzes and project tutorials and lectures. Note, all x-hour periods will be in Sudikoff Lab 01. The last two weeks of the course are held over for group projects - there will be no lectures during this period but there will be group design reviews and code reviews. See the schedule below.

Week 1
Lecture 1 Getting Started
Lecture 2 The Linux Shell and Commands
Lab Work through "Getting Started" lecture notes, demo of Garcia robot
Lecture 3 The Linux Shell and Commands (continued)

Week 2
Lecture 4 Shell Programming
Lecture 5 Shell Programming (continued)
Lecture 6 Shell Programming (continued)
Lecture 7 The C Programming Language

Week 3
Lecture 8 Preprocessor, Functions, Data Structures, Arrays and Strings
Lecture 9 Standard IO Lib and C/OS Interface - Oh and bad IO too!
Lecture 10 Pesky Pointers
Lecture 11 Pesky Pointers and Dynamic Memory Allocation

Week 4
Lecture 12 Searching the Web and Design Methodology
Lecture 13 TinySearch Engine: Crawler Design
Lecture 14 TinySearch Engine: Crawler Data Structure Design and
Lecture 15 The Make Utility

Week 5
Lecture 16 TinySearch Engine: Indexer Design
Lecture 17 The Art of Debugging
Lab Pizza and Pointers
Lecture 18 The Art of Testing (and Writing Good Code)

Week 6
Lecture 19 The Art of Testing: Unit Testing
Lecture 20 (A comment on the design of the query engine)
Building your very own C library

Lecture 21 Concurrent Versions System (CVS) - source code management

Week 7
Lecture 22 Socket Programming
Lecture 50 Processes, threads, mutex
Lab PROJECT: Garcia Tutorial Part I
Lab PROJECT: Garcia Tutorial Part II
Lab PROJECT: GTK Tutorial

Week 8
No classes: group project work
Design Review

Week 9
No classes: group project work
Code Review

Week 10
No classes: group project work
Demo or die day

Lab Assignments

Typically, labs are given out after the lecture on Monday and are returned the following Sunday at 24.00 hours. Labs are graded and returned one week later. Each student is given two free passes for 48 hour extensions with no penalty. But you do not want to use these if possible since you will fall behind.

Lab1 - Shell Commands
Lab2 - Shell Programming
Lab3 - C Programming
Lab4 - Crawler
Lab5 - Indexer
Lab6 - Query Engine
Lab7 - Socket Programming

Labs 4-6 are the TinySearchEngine set. You will code the crawler and indexer. We will design these components in class together. You will do the design for the query engine and code it. Then you will put all three components together and you'll have your TinySearchEngine. Now you are ready to take on Google! This is an extremely cool set of assignments. It is, however, very challenging but you will learn how to design, implement, test and debug a complex software system. You will gain significant coding chops doing this. It will be painful but we promise you will get there.

You will write a lot of C code in this course: approximately 500 lines of bash scripts and 3000 lines of C for the assignments and 1000 lines for the joint project - these are counts of C code, excluding comments. Note, there is some code reuse/ refactoring between the TinySearchEngine set of labs.

Lab 2 (Shell): 445
Lab 3 (C): 387
Lab 4 (Crawler): 692
Lab 5 (Indexer): 1002
Lab 6 (Query Engine): 587
Lab 7 (Sockets): 389
Project: 1063

Group Project

There are no lectures during the last two weeks of the course. During this time you will be working in teams programming Gracia robots. The project will pull on all the skills you have developed so far but will provide a set of new skills needed to do cross development and embedded systems programming and debugging. This completes the hacker's toolkit that you can be proud of. We will do distributed programming using sockets, mutable threads for embedded Linux.

We have added a mote sensor to the bot that allows it to interact with a sensor network built across the CS department. We have also added a camera to the bot. You will write the controller that runs on your laptop and the server side on the robot - all in C and Linux. Once your ready your bot will go on a treasure hunt. The final task is the CS50 remote controlled bot race.

The project is made up of a small team (three or four people) and requires strong collaboration and a problem solving mindset to get the job done. The instructor will put the teams together with each member being responsible to deliver against a part of the overall system design, implementation, testing and integration. The goals of this activity are to help you develop the confidence, skills, and habits necessary to write large computer programs while part of a multi-person team. You will become conversant in software engineering paradigms, and be exposed to various public-domain and open source tools that make the software development process easier. In addition, you will develop vital skills in self-directed learning, problem solving, and communication. The project will have a design and code review as well as the demo.

A project report that captures the design and implementation will be submitted as part of the assessment. The report should be 10 pages max: include

1) Thread design of client and server
2) Design Specs
3) Implementation Specs
4) GUI screen dump
5) Lesson learned

The project potion of the course grade is large: 30% of the overall grade. We give the same grade for the project to all members of a team unless it is clear that people aren't equally pulling their weight. There has not been a year when we didn't give the same grade to all members of a team.

The following set of deadlines are important for the progress of the project. Please note that you need to provide documentation for the design and code reviews and the final project submission.

Design review. The project review should include requirements, Design Spec (inputs/outputs, data flow, data structures, pseudo-code) and functional decomposition. We are particularly interested in how the functions map to threads on the client (Unix machine/laptop) and server (bot); we will discuss the threaded design in the review. We are also interested in who is doing what.

Code review. The code review should include the Implementation Spec and whatever code is written up until the review point.

Demo or die day. Project presentation (design overview, lessons learnt, etc) and demo of project.

Project reports due. The report written in latex includes description of the project, the Design Spec, functional decomposition of the system, Implementation Spec, lessons learnt. The

appendix includes all code and unit tests. All materials should be signed into the teams cvs project page.

Class Reading

Throughout the course we will read a number of articles that relate to programming and the software development cycle. Most will be anecdotal based on experience working in the software industry and others more academic. Please read them and come armed with thoughts, opinions, and questions.

I like these articles for a number of reasons: they are insightful, sometimes funny, always opinionated and reflect real world experience - whether you like that or not.

Week 1
-The Tale of J. Random Newbie (2 pages), chapter 16, section 1 of The Art of Unix Programming, Eric S. Raymond.
-Biculturalism, (3 pages), Joel Spolsky

Week 2
-The Joel Test: 12 Steps to Better Code, (9 pages), Joel Spolsky
-Philosophy (15 pages), chapter 1 of The Art of Unix Programming, by Eric S. Raymond.

Week 3
-The Perils of JavaSchools, (6 pages), Joel Spolsky

Week 4
-Searching the Web, ACM Transactions on Internet Technology (TOIT), Volume 1, (39 pages), Issue 1 (August 2001), Arvind Arasu, Junghoo Cho, Hector Garcia-Molina, Andreas Paepcke, Sriram Raghavan (Stanford University)

Week 5
- Craftsmanship, (4 pages), Joel Spolsky
- Painless Functional Specifications - Part 1: Why Bother?, (6 pages), Joel Spolsky

Week 6
-Things You Should Never Do, Part I, (5 pages), Joel Spolsky
-Painless Functional Specifications - Part 2: What's a Spec?, (4 pages), Joel Spolsky

Week 7
- Fire And Motion, (5 pages), Joel Spolsky