# ECON 0XA2E5: SOFTWARE ENGINEERING FOR ECONOMISTS

*C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do it blows your whole leg off.*

— Bjaren Stroustrup

*Train your weaknesses and race your strengths*

— Mike Walden

| | | | |
|---|---|---|---|
| **Instructor:** | Benjamin S. Skrainka | **Email:** | **skrainka@uchicago.edu** |
| | | **www:** | `home.uchicago.edu/~skrainka` |
| **Course:** | ECON 0xA2E5 | **Offices:** | Harris School 206b |
| **Class times:** | MW 10:30-11:50 AM | | Searle 228 |
| **Location:** | RO 329 | **Office hours:** | By appointment |
| **TA:** | Mathilde Almlund | **Email:** | **malmlund@gmail.com** |

## CONTENTS

## 1 COURSE DESCRIPTION

This course provides the essential computational skills needed to solve complex, scientific problems and to use computers more productively. Students will learn how to design and implement numerical software using C++ and Unix/Linux, the current standard for scientific computing. We also focus on using modern software development tools (version control, debugger, make, profiler), common numerical libraries, and open source software (OSS) to write better code more quickly. The course concludes with an introduction to parallel programming.

Note: this course is a complement to ECON 41800 Numerical Methods in Economics and explains how to implement these numerical methods in software.

## 2 MY BIASES

Like many disciplines, there is a split between theoretical and practical people in computer science. I have always used computers to get stuff done, and my goal is to teach you how to do the same, i.e. I teach an 'industry' perspective. If you want to solve important problems, then Unix[1] is the platform of choice. It is much more robust and facilitates the development of reliable software. When combined with C or C++, you have the power to take on most computational challenges.[2] Furthermore, supercomputers and clusters use some Linux variant, so you need to know Linux if you need massive computing power. Unix's design philosophy makes you more productive. Finally, knowledge of Unix and C/C++ will help you master and exploit other computational languages and platforms more quickly and effectively.

I also believe that it is important to learn how to use the command line interface. Not only will it make you more productive, but it also ensures that you master the concepts and techniques, which can be obscured by a graphical user interface (GUI). Later, you can always use an application with a GUI, but then you will know what you are doing and why you are doing it.

## 3 COURSE OBJECTIVES: C++, VI, AND UNIX

At the conclusion of this course, you will be able to design, implement, debug, test, optimize, and execute industrial-strength software in C++ using the Unix operating system. I may also digress and discuss MATLAB, R, FORTRAN, Python, and Stata, as appropriate. All examples will use problems in Economics to the extent possible.

The course is centered around developing the skills to implement a Monte Carlo simulation to show that OLS is consistent.

### 3.1 *Benefits of Taking this Course*

The key learning objectives are:

- Develop software more professionally and productively
- Design and implement numerical applications in C++ to improve performance and tackle larger problems (time, data)
- Exploit massive computational resources such as parallel programming in order to solve large problems or work with large data sets

1 Unless otherwise specified, I will use Unix to refer to the family of Unix-style operating systems, including Unix, Linux, and Mac OS/X.

2 In fact, C was designed for writing operating systems, which means C (or C++) is the language of choice when trying to write extensions to R, MATLAB, Python, etc.

- Configure, build, and use third-party and OSS solvers and other (numerical) libraries

### 3.2 *Explanation of the Course Goals*

Unix is an obvious choice because it is the language of professional computation. Unix makes you more productive in many ways and is much more reliable than Windows. Also, supercomputers run Linux: if you need massive computational muscle, you will need some Unix knowledge.

Different systems are administered differently. Consequently, your favorite editor may not be available, unless your favorite editor is vi or emacs. I like vi because it is fast, powerful, and ubiquitous. Emacs is also a good choice. Both will make you very productive if you invest in the necessary human capital. That said, vi is easier to learn but less powerful.

Last year, the course used C, but that made linear algebra difficult. Modern template libraries such as Eigen make linear algebra as easy as writing MATLAB code and as fast as Intel MKL, the gold standard for performance. This year course will switch to C++ because the productivity payoffs are worth the extra complexity. Also, if you know C++ it is easy to program in C, which is basically a subset of C++, or Java. In addition, C/C++ is like Latin: if you master it, you will be able to master other languages quickly, even those which have not yet been written.

Mastering the soft skills in addition to C++ and Unix will allow to you prove that you exercised professional diligence in verifying you work and help diffuse criticisms of models which are considered too complex and untransparent.

### 3.3 *Disclaimer*

Unfortunately, I don't have time to teach you everything I know about computing and you probably wouldn't enjoy it if I did. I will focus on the most important skills you need to be productive during your career, which includes knowing the fundamental concepts needed to master new technologies . . . or technologies which are not covered in class.[3] In particular, I will only teach the core of C++, especially essential functionality and templates. I will ignore most of the OO technology because it is not useful for most numerical work. You will need to RTFM[4] to use additional libraries/packages which may be relevant for your work but which I do not have time to cover.

## 4 ATTENDANCE

You can take the course one of two ways. The best option is to make the commitment to attend all the lectures, do the assignments, and get a grade. The only way to learn programming is to write code. . . and more is better. Just dive in and be fearless. I have found that using the best tool for the job saves time in the long run. Initially, it will be frustrating, but the second (and third) time, you will be much more productive.

However, you can also attend only the lectures that interest you as an auditor. You will still get something out of the class and gain productivity, but you probably will not master the skills to develop good C++ code quickly.

3 For example, students who attended last year's class figured out how to write R extensions without my help.

4 RTFM is standard acronym in industry, meaning *R*ead *t*he *F****ing *M*anual.

## 5 COMPUTER SOFTWARE AND EQUIPMENT

### 5.1 *Standard Class Environment*

To complete this class, you need access to a Unix operating system, a relatively recent version of the GNU G++ compiler, and vi or emacs. There are several options you can use:

1. Install Linux. Ubuntu (`www.ubuntu.com`) is an easy to use and install distribution. Make sure that you also install gcc-4.x via the Synaptic Package Manager.

2. Use OS/X. Mac's let you have your cake and eat it too. Underneath that slick exterior lies Unix. To enable all the Unix goodness you could want, install MacPorts (See `www.macports.org`.), which is a Unix-style package manager which will download, build, and install most Unix facilities. Once you have installed MacPorts, install the g++ compiler with (this may take several hours) by executing the following in a Terminal window:

   ```
   $ sudo port -v install gcc45
   ```

   Also install either MacVim (`http://macvim.org/OSX/index.php`) or AcquaMacs (`http://aquamacs.org/`).

3. Use Cygwin to emulate Unix on a PC. This is the least invasive, but worst option. See `www.cygwin.com` for installation instructions. Make sure you install gcc and vi.

4. Get an account on a Unix machine

Before you do any of these things, back up your computer... On OS/X, there is no excuse for not using Time Machine.

### 5.2 *Athens Account*

Athens is the reference platform for all homework until the Office of Research Computing's new machine is available. The grader should be able to build and run your homework on this machine. To request an account on Athens, go to `https://iota.src.uchicago.edu/user`. Connect to Athens via SSH:

```
$ ssh YourUsername@athensx.uchicago.edu
```

or via VNC (`http://sscs.uchicago.edu/pages/vnc-guide.html`). If you do not know your username, it is probably your CNetID.

### 5.3 *Software*

For linear algebra, download Eigen from `http://eigen.tuxfamily.org/index.php?title=Main_Page`.

Use Mersenne Twister to generate random numbers, use Richard Wagner's implementation which I have uploaded to Chalk. I can no longer find it on the Internet.

## 6 ASSESSMENT

Grading will be based on:

| | |
|---|---|
| Homework | 25% |
| Midterm Take Home Exam | 25% |
| Final Project | 50% |

### 6.1 *Final Project*

Pick a subject which interests you and is manageable in a quarter. One option is to reproduce a journal article. Most likely, there is some way you can improve on what was done in the paper and/or the published work will not be reproducible. If you play your cards right this could become a paper, a note, or a chapter in your thesis. Projects need to be approved by the end of week 4. In terms of marking, economic content is less important than showing mastery of the skills taught in class and the ability to apply them to a concrete problem.

### 6.2 *Coding Convention*

All code must conform to the coding convention. See the coding convention handout.

### 6.3 *Homework Submission*

Homework is due at the start of class each Monday unless stated otherwise. If you expect you will need to hand in work late, please discuss it with me ahead of time because it is hard to fix something ex-post. In general, late homework will be marked at the convenience of the grader. Please submit work via Chalk as a complete zipped file which includes a script or Makefile to build your code on the reference platform for homework (Athens, an Office of Research Computing machine, or OS/X.).

## 7 READING

### 7.1 *Required*

The required text is Deitel and Deitel [5], but you will probably want to purchase Sobell [19] as well. The former covers C++ and how to program effectively; the later focuses on basic Unix skills including key commands, concepts, regular expressions, and vi. Chacon [4] explains how to use git) for version control and is available for free at `http://progit.org/ebook/progit.pdf`.

### 7.2 *Recommended*

Kernighan and Pike [8] provides practical advice about how to design and build high quality software. Scott Meyer's books, especially Meyers [17], provide invaluable real-world advice about how to use C++ without blowing off your leg.

### 7.3 *Helpful*

Koenig and Moo [10] is an immersion-based approach to learning C++ which is much less wordy than Deitel and Deitel [5], but will be unsatisfying if you want a more rigorous treatment. Agans [1] provides advice on how improve your skills at debugging software. Learning how to debug is a key skill which will save you a lot of time over your career. McCullough and Vinod [13] shows how unreliable some commercial packages are as well as how few papers are reproducible. Bryant and O'Hallaron [3] provides more detail on how computer systems work so that you can optimize your code more effectively. Consult Lakos [11] if you ever have to develop extremely large systems. Kerrisk [9] has replaced Stevens et al. [20], long the classic reference, as the book on using the Unix system programming interface to develop Unix applications. Brooks [2] is a classic description of pitfalls awaiting software developers, especially on larger projects, and still is often surprisingly relevant.

## 7.4 *References*

Josuttis [7] covers the Standard C++ Library in gory detail and is an excellent reference – probably not worth purchasing unless you use the Standard Library regularly.

## 7.5 *Articles*

Goldberg [6] explains everything you need to know about floating point and more. McCullough and Vinod [13] will make you suspicious of most computational results in Economics.

| Lecture | Subject | Objectives | Reading |
|---|---|---|---|
| 0 | Intro to Unix | Unix process model, basic commands, configuration, and vi | S 2-6 |
| 1 | Intro to C++ Programming | Fundamental hardware and software concepts; Compiling and linking a program; basic programming concepts; hello world; | DD 1 (skim), 2 & D |
| 2 | Control Statements I & II | Looping and branching + goto | DD 3 & 4 |
| 3 | Software Engineering | How to write better software | KP 1, 3, 4, 8 |
| 4 | Functions | Using functions to improve software quality, reuse, performance, and maintainability; Top-down & Bottom-up design; linking & multiple modules; C Preprocessor | DD 5 & E & F |
| 5 | Version Control | Source code management with git | ProGit |
| 6 | Arrays, vectors, strings, file I/O | Key data types; basic file I/O | DD 6 & 8 |
| 7 | Debugging & testing | How to debug software; Using GDB, unit tests, and valgrind | DD I, KP 5 |
| 8 | Pointers & memory management | Pointers; call by value vs reference; const | DD 7 |
| 9 | TBD | | |
| 10 | Classes | Classes | DD 9 & 10 |
| 11 | OO Concepts | OO concepts: operator overloading; polymorphism; encapsulation; inheritance; isA vs. hasA (inheritance vs. aggregation) | DD 11-13 |
| 12 | Templates | Templates ; Standard C++ Library (vector, map, set) | DD 14 |
| 13 | Linear Algebra & Random Numbers | Using Eigen for linear algebra; Mersenne Twister | Eigen |
| 14 | Exception Handling & Advanced Templates | Error recovery, additional template libraries (containers, string, streams) | DD 16, 15, 21 |
| 15 | Building Software | Using make to build software | |
| 16 | Intermediate Unix Skills | Shell programming; regular expressions; bash; sed; awk; Python | Sobell |
| 17 | Parallel Programming | TBD | |
| 18 | Data | Cleaning, provenance, and databases | |
| 19 | C Legacy Issues | Preprocessor, typedef; casts; sizeof; namespaces; POSIX APIs and system calls | DD F |

Reading key: DD$\rightarrow$ Deitel & Deitel; KP$\rightarrow$ Kernighan & Pike; S $\rightarrow$ Sobell;

## 9 CHALK

See `chalk.uchicago.edu` for the latest information, announcements, etc.

## EXTRAS

Some of you may find Software Carpentry (www.software-carpentry.org) a useful compendium of lectures and best practice, especially for learning core software development skills as well as Python.

## REFERENCES

[1] D.J. Agans. *Debugging: the nine indispensable rules for finding even the most elusive software and hardware problems*. Amacom Books, 2002.

[2] F.P. Brooks. The mythical man-month: Essays on software engineering. 1995.

[3] Randal E. Bryant and David Richard O'Hallaron. *Computer Systems: a Programmer's Perspective*. Prentice Hall, 2003.

[4] S. Chacon. *Pro Git*. Springer, 2009. http://progit.org/ebook/progit.pdf.

[5] P. Deitel and H. Deitel. *Program C++ How to Program: Late Objects Version*. Prentice Hall, 7 edition, 2010.

[6] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991.

[7] N.M. Josuttis. *The C++ Standard Library: a Tutorial and Handbook*. Addison-Wesley Professional, 1999.

[8] B.W. Kernighan and R. Pike. *The Practice of Programming*. Addison-Wesley Professional, 1999.

[9] M. Kerrisk. *The Linux Programming Interface: A Linux and Unix System Programming Handbook*. No Starch Press, 2010.

[10] A. Koenig and B.E. Moo. *Accelerated C++: Practical Programming by Example*. Addison-Wesley, 2000.

[11] John Lakos. Large-scale c++ software design. *Reading, MA*, 1996.

[12] S.B. Lippman and Safari Tech Books Online. *Inside the C++ object model*. Addison-Wesley, 1996.

[13] B.D. McCullough and H.D. Vinod. The numerical reliability of econometric software. *Journal of Economic Literature*, 37(2):633–665, 1999.

[14] M.K. McKusick and G.V. Neville-Neil. *The Design and Implementation of the FreeBSD Operating System*. Addison Wesley, 2004.

[15] S. Meyers. *More Effective C++: 35 New Ways to Improve Your Programs and Designs*. Addison-Wesley Longman, Amsterdam, 1996.

[16] S. Meyers. *Effective STL*. Addison-Wesley, 2001.

[17] Scott Meyers. *Effective C++*. Addison-Wesley, 2005.

[18] W.J. Savitch. *Absolute C++*. Addison Wesley, 2009.

[19] Mark G. Sobell. *A Practical Guide to Linux Commands, Editors, and Shell Programming*. Prentice Hall, 2nd edition, 2009.

[20] W.R. Stevens, S.A. Rago, and D.M. Ritchie. *Advanced Programming in the UNIX Environment*. Addison-Wesley New York., 1992.