

CMSC 838P: Research in Software Engineering

Instructor: Vibha Sazawal

Spring 2006

This reading list is tentative and is subject to change. The list is divided into ten topics. Some topics are larger than others and will span multiple weeks.

Books

There are two books for this course:

- *Secure Coding: Principles and Practices* by Mark Graff and Kenneth van Wyk.

I like this book because it takes a lifecycle approach to security. In this course, we'll be taking a lifecycle approach to everything.

- *Software Fundamentals: Collected Papers* by David L. Parnas.

This is a collection of papers by David Parnas. You can find most of his papers online, so this book is optional. However, the typesetting of this book is far superior to that used in the original papers, so it is worth it to purchase the book. In addition, I will be using the section numbering and page numbering that the book uses and not the original papers.

Topic 1: "Why bad code happens to good people"

- Mark Graff and Kenneth van Wyk. *Secure Coding: Principles and Practices*. Chapters 1-4.
- Fred Brooks. *No Silver Bullet - Essence and Accidents of Software Engineering*. IEEE Computer, April 1987

Topic 2: Software processes

No matter what, everyone has a process. It just might not be a consciously chosen process.

- Barry Boehm. *A Spiral Model of Software Development and Enhancement*. IEEE Computer, vol.21, #5, May 1988, pp 61-72.
- Kent Beck. *Embracing Change with Extreme Programming*. IEEE Computer. 32(10), p. 70-77, 1999.

Topic 3: Requirements and formal specification languages

We'll look at formalisms for requirements, both old-skool and new-skool. Writing requirements in English is easy but ambiguous. If you don't believe me, just look at the instructions for your homework assignments (from other classes, of course). They may just be vague and ambiguous about what precisely is required.

- Kathryn Heninger. *Specifying Software Requirements for Complex Systems*. Chapter 6 of *Software Fundamentals*.
- Daniel Jackson, Ilya Shlyakhter and Manu Sridharan. *A Micromodularity Mechanism*.

Topic 4: Parnas on design

You will soon learn that I really like David Parnas' work.

- David Parnas. *Software Fundamentals*. Chapters 7, 10, 14, 15, 16. (I will give out a full bibliography for those who don't have the book later.)

Topic 5: Other people on design and architecture

We'll look at how designs are categorized, and also languages and tools created to offer the designer more power. How easy is it to use these mechanisms for good and not evil?

- Gregor Kiczales et al. *An Overview of AspectJ*. 15th European Conference on Object-Oriented Programming (ECOOP 2001), pp. 327-353, June 2001
- Harold Ossher and Peri Tarr. *Using multidimensional separation of concerns to (re)shape evolving software*. Communications of the ACM, October 2001.
- Karl Lieberherr et al. *Aspect-oriented programming with adaptive methods*. Communications of the ACM, October 2001.
- Michael Van Hilst and David Notkin. *Using role components to implement collaboration-based designs*. In Proc. of the Intl. Conf. on Object-Oriented Programming Systems, Languages, and Applications, 1996.
- David Garlan and Mary Shaw. *An Introduction to Software Architecture*, Advances in Software Engineering and Knowledge Engineering, 1993.
- Erich Gamma et al. *Design patterns: Abstraction and reuse of object-oriented design*. Proceedings of ECOOP '93: 7th European Conference Proceedings, (707), 406-431.

Topic 6: Development tools

There are debuggers, there are profilers, and then there are all sorts of other crazy stuff you can plug into your development environment.

- Reid Holmes and Gail C. Murphy. *Using Structural Context to Recommend Source Code Examples*. In Proc. of the 27th Intl Conf. on Software Engineering (ICSE-27). 2005.
- Vibha Sazawal and David Notkin. *Supporting Ease of Change in the Context of Code*. Technical Report UW-CSE-2004-09-01.
- William Griswold et al. *Tool Support for Planning the Restructuring of Data Abstractions in Large Systems*. In Proc. of the ACM SIGSOFT Conf. on the Foundations of Software Engineering (FSE-4), 1996.

Topic 7: Debugging, testing, and inspection

Bugs happen! But there are lots of interesting ways to find them.

- Junfeng Yang et al. *Using Model Checking to Find Serious File System Errors*. Operating System Design and Implementation (OSDI) 2004.
- Chandrasekhar Boyapati et al. *Korat: Automated testing based on Java predicates*. Intl. Symp. on Software Testing and Analysis (ISSTA), 2002.
- Atif Memon et al. *Hierarchical GUI test case generation using automated planning*, IEEE Transactions on Software Engineering, Feb 2001.
- Mike Ernst et al. *Dynamically Discovering Likely Program Invariants to Support Program Evolution*. In Proc. of the 21st Intl. Conf. on Software Engineering, 1999.
- Holger Cleve and Andreas Zeller. *Locating Causes of Program Failures*. In Proc. of the 27th Intl. Conf. on Software Engineering, 2005.
- E. Ann Myers and John Knight. *An Improved Software Inspection Technique And An Empirical Evaluation Of Its Effectiveness*. Communications of the ACM, November, 1993.

Topic 8: Hot topics: aspects and modularity

One of my favorite things to talk about. Read up and then join the debate!

- Jonathan Aldrich. *Open Modules: Modular Reasoning about Advice*.
- Gregor Kiczales and Mira Mezini. *Aspect-oriented programming and modular reasoning*. ICSE 2005.
- Kevin Sullivan et al. *Non-modularity in aspect-oriented languages: integration as a crosscutting concern for AspectJ*. In Proc. of the 1st Intl. Conf. on Aspect-oriented Software Development, 2002.
- Kevin Sullivan et al. *Information hiding interfaces for aspect-oriented design*. SIGSOFT FSE/ESEC 2005.

Topic 9: Hot topics: economic software engineering

We'll be looking at how to apply economic techniques to *technical* decision making. Can you put a price on modularity? What's the cost of crappy design? Read to find out.

- Kevin Sullivan et al. *The structure and value of modularity in software design*. SIGSOFT FSE/ESEC 2001.
- Neeraj Sangal et al. *Using Dependency Models to Manage Complex Software Architecture*. OOPSLA 2005.
- Mary Shaw et al. *In Search of a Unified Theory for Early Predictive Design Evaluation for Software*. Technical Report CMU-CS-05-139.
- excerpt from Paul Clements et al. *Evaluating Software Architectures*. 2002.

Topic 10: Hot topics: software evolution analysis

So current I can't even finish the reading list yet!

- Miryung Kim et al. *An Empirical Study of Code Clone Genealogies*. SIGSOFT FSE/ESEC 2005.
- others tbd