605.401
Foundations of Software Engineering

Joel Coffman

Johns Hopkins University
Engineering for Professionals

Fall 2017

Some slides adapted from *Software Engineering: A Practitioner's Approach* [Pressman, 2010].

Introduction

Outline

Contact Information

Johns Hopkins University
Applied Physics Laboratory
240 228-0544 / 443 778-0544
http://pages.jh.edu/~jcoffma2
joel.coffman@jhu.edu (preferred)
joel.coffman@jhuapl.edu

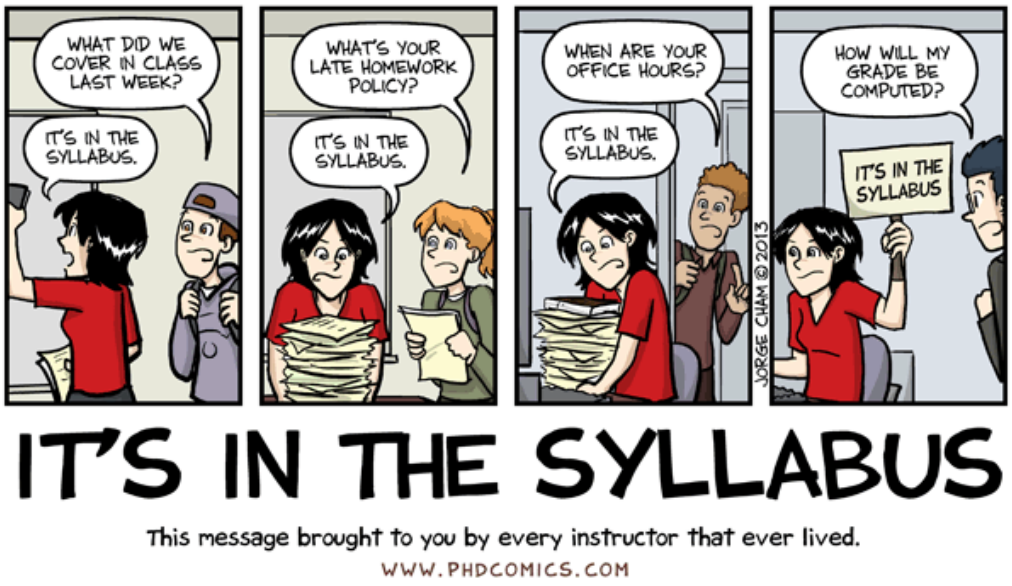
Please use email to contact me. I do my best to be responsive—I usually respond within 24 hours.

Biographical Sketch

- PhD from the University of Virginia (2012)
- Senior Cyber Engineer at JHU/APL
  - OpenStack technical lead responsible for open source security features
  - PI for internally-funded R&D
  - Involved with many other software projects
- Lecturer in JHU's Whiting School of Engineering

Outline

The beginning of wisdom for a programmer is to recognize the difference between getting his program to work and getting it right. A program which does not work is undoubtedly wrong; but a program which does work is not necessarily right. It may still be wrong because it is hard to understand; or because it is hard to maintain as program requirements change; or because its structure is different from the structure of the problem; or because we cannot be sure that it does indeed work. [Jackson, 1975]



"Piled Higher and Deeper" by Jorge Cham
www.phdcomics.com

Course

Foundations of Software Engineering

Fundamental software engineering techniques and methodologies commonly used during software development are studied. Topics include various life cycle models, project planning and estimation, requirements analysis, program design, construction, testing, maintenance and implementation, software measurement, and software quality. Emphasized are structured and object-oriented analysis and design techniques, use of process and data models, modular principles of software design, and a systematic approach to testing and debugging. The importance of problem specification, programming style, periodic reviews, documentation, thorough testing, and ease of maintenance are covered.

Textbook

Required

Brooks, Jr., F. P. (1995). *The Mythical Man-Month*.
Addison-Wesley, anniversary edition
ISBN: 0-201-83595-9 / 978-0-201-83595-3

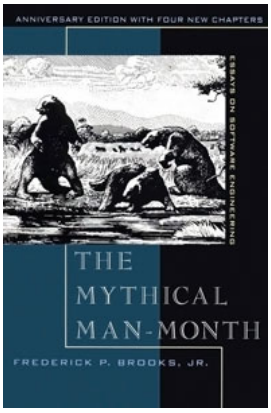


Image source: http://en.wikipedia.org/wiki/File:Mythical_man-month_(book_cover).jpg

Course Policies

- Special arrangements must be made in advance
  - Special needs
  - Late assignments
- Schedule (as well as other policies) subject to change with prior notice to class
- Assignments
  - Late assignments will generally not be accepted
  - Clearly state any assumptions and show your work

Grading

Weighting

Item	%
Weekly Quizzes	40
Project / Assignments	40
Final Exam	20

Scale

Grade	%
A	90–100
B	80–89
C	70–79
F	<70

± may be assigned at the instructor's discretion

Project

An Experiment. . .

An ongoing class project that spans individual class offerings

- This class is the second so you're inheriting existing code

Open Source

The project will have an open source license

- Speak to me if you cannot contribute to an open source project

Office Hours

Questions?

Email is best, but I can also meet before class if necessary.

I will make every effort to respond to email within 24 hours, but I rarely check or respond to email on Sundays

Class Reviews

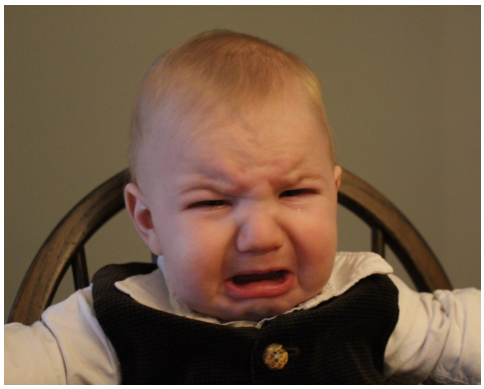
Most classes include a review of prior material. This period is great for asking questions or clarifying details.

Outline

Student Feedback

It's one of the best ways for me to improve as an instructor!

- Comments, suggestions, criticisms welcome



What have students said in the past?

- You can find evaluations from prior courses on my website: http://pages.jh.edu/~jcoffma2
- . . . but a few highlights appear next

The Mythical Man-Month

> I felt that the readings from The Mythical Man Month were interesting but outdated. The best readings for me were web or print articles like those from Joel Spolsky. (Fall 2014)

The Mythical Man-Month is a classic software engineering text, but several of the most outdated chapters have been dropped from the reading assignments. New articles have been added from additional authors. (changed Fall 2016)

> The book (Mythical Man Month) was a great read; highly applicable in many settings, not just software engineering. (Fall 2016)

Weekly Quizzes

> Weekly quizes [*sic*] are one of the best ways to reinforce the material because it forces the student to study each week's material well. (Spring 2014)

> The weekly quizzes were a great way to keep on top of the reading and worked as a knowledge check for topics you may need to focus more on. I definitely prefer weekly quizzes to a midterm as it is much less stressful and daunting. (Fall 2014)

> Weekly exams kept me honest and ensured I was retaining material. They were a challenging, but effective (and welcome) element of the course. (Fall 2016)

Assignments

> [. . .] there was not enough direction for most of the assignments. It was not very easy to decipher exactly what the professor wanted as a finished product. (Spring 2014)

> The project expectations could have been better communicated, and the project requirements could have been more explicitly related to the course content. (Fall 2016)

Incomplete requirements are part of software engineering, but I'm continually revising the assignments to make the expectations clearer.

Project I

> Project archival system very cumbersome and difficult to use. Majority of workload frustration spent sidestepping archive system. (Fall 2016)

> [. . .] the collaboration tool (Phabricator) only worked with a particular version of Mercurial, many found it difficult to do reviews and manage revisions, it had its own command line tool and markup language that I didn't think were worth learning [. . .]. A less esoteric tool may have been more effective. (Fall 2016)

I've created a longer tutorial introduction to Phabricator along with scripts that automate much of the configuration for users. (changed Fall 2017)

Project II

I believe that using a version control system that is common in the industry would have 1) given us more actionable skills from the class and 2) would have cut down on confusion and bugs. Using [. . .] Git would have been a whole lot easier in my opinion. Also due to the wide use of Git, the tools to track code/review code would not have been so finicky. (Fall 2016)

Klingon Code Warriors embrace Git, we enjoy arbitrary conflicts. Git is not for the weak and feeble. TODAY IS A GOOD DAY TO CODE. (@leted)

Mercurial is widely used (e.g., Facebook and Mozilla) but isn't as popular as Git. Mercurial was original chosen for cross-platform compatibility and simplicity, but given the compatibility issue with Phabricator, I've switched to Git instead of Mercurial. (changed Fall 2017)

Quote: https://twitter.com/leted/status/232588721181052928

Project III

The fact the whole team was working on a single project made the lessons taught by the professor very accessible since we could all observe the concepts at plain during our design, implementation, and integration of the software project. (Fall 2016)

Course

I appreciated how the course was focused more on the doing rather than the theory. (Fall 2014)

I like how the class helped us with memory retention of all in-class material over time. (Fall 2014)

Course was very well put together. Professor made sure it was very interactive, and applicable to real world scenarios. (Fall 2016)

Challenges like the marshmallow and ping pong ball event built the class as a team, and demonstrated SE / Management principles so effectively I was taken by surprise. (Fall 2016)

Outline

Introductions

Icebreaker

You will be assigned to team and must construct, with your team, the specified item. Some team members may have special roles in addition to contributing to the team's effort.

Unless otherwise specified for a particular role, *you may not use other resources* (e.g., the Internet). You have 15 minutes total to complete this task.

Item to construct: castle

Grader

Kaitlin Farr

Johns Hopkins University
Applied Physics Laboratory
240 228-4338 / 443 778-4338
kaitmfarr@gmail.com

Background

- Software engineer at JHU/APL
- MS in Computer Science from Johns Hopkins University
- BS in Computer Science from Texas A&M University

Appendix

Outline

References I

[Albert et al., 2013] Albert, E., de Boer, F., Hähnle, R., Johnsen, E. B., and Laneve, C. (2013). Engineering Virtualized Services. In *Proceedings of the Second Nordic Symposium on Cloud Computing & Internet Technologies*, NordiCloud '13, pages 59–63.

[Baker, 1972] Baker, F. T. (1972). Chief programmer team management of production programming. *IBM Systems Journal*, 11(1):56–73.

[Bentley, 1985] Bentley, J. (1985). Programmimg Pearls: Bumper-Sticker Computer Science. *Communications of the ACM*, 28(9):896–901.

References II

[Bersoff et al., 1980] Bersoff, E. H., Henderson, V. D., and Siegel, S. G. (1980). *Software Configuration Management*. Prentice Hall, Englewood Cliffs, NJ.

[Boehm, 1981] Boehm, B. (1981). *Software Engineering Economics*. Prentice Hall, 1st edition.

[Boehm, 1996] Boehm, B. (1996). Anchoring the Software Process. *IEEE Software*, 13(4):73–82.

[Brooks, 1986] Brooks, Jr., F. P. (1986). No Silver Bullet—Essence and Accidents of Software Engineering. In Kugler, H.-J., editor, *Proceedings of the IFIP 10th World Computer Congress*, Information Processing 86, pages 1069–1076. North-Holland / IFIP.

References III

[Brooks, 1995] Brooks, Jr., F. P. (1995). *The Mythical Man-Month*. Addison-Wesley, anniversary edition.

[Chacon and Straub, 2014] Chacon, S. and Straub, B. (2014). *Pro Git*. Apress, 2nd edition.

[Charette, 2013] Charette, R. N. (2013). The U.S. Air Force Explains its $1 Billion ECSS Bonfire. *IEEE Spectrum*.

[Chen, 1976] Chen, P. P.-S. (1976). The Entity-relationship Model—Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36.

[Chidamber and Kemerer, 1994] Chidamber, S. R. and Kemerer, C. F. (1994). A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, 20(6):476–493.

References IV

[Constantine, 1993] Constantine, L. L. (1993). Work Organization: Paradigms for Project Management and Organization. *Communications of the ACM*, 36(10):35–43.

[Desouza and Smith, 2015] Desouza, K. and Smith, K. (2015). Why Large Scale Government IT Projects Fail and What To Do About It.

[Dijkstra, 1988] Dijkstra, E. W. (1988). On the cruelty of really teaching computer science. Technical report.

[Eggen and Witte, 2006] Eggen, D. and Witte, G. (2006). The FBI's Upgrade That Wasn't.

[Fagan, 1976] Fagan, M. E. (1976). Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15(3):182–211.

References V

[Fairley and Willshire, 2003] Fairley, R. E. and Willshire, M. J. (2003). Why the Vasa Sank: 10 Problems and Some Antidotes for Software Projects. *IEEE Software*, 20(2):18–25.

[Fishman, 1996] Fishman, C. (1996). They Write the Right Stuff. *Fast Company*, (6).

[Fowler, 2004] Fowler, M. (2004). *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Pearson Education, Inc., 3rd edition.

[Fowler et al., 1999] Fowler, M., Beck, K., Brant, J., Opdyke, W., and Roberts, D. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley.

References VI

[Gibbs, 1994] Gibbs, W. W. (1994). Software's Chronic Crisis. *Scientific American*, 271(3):72–81.

[Goldstein, 2005] Goldstein, H. (2005). Who Killed the Virtual Case File? *IEEE Spectrum*, 42(9):24–35.

[Hähnle and Johnsen, 2015] Hähnle, R. and Johnsen, E. B. (2015). Designing Resource-Aware Cloud Applications. *Computer*, 48(6).

[Harrison et al., 1998] Harrison, R., Counsell, S. J., and Nithi, R. V. (1998). An Evaluation of the MOOD Set of Object-Oriented Software Metrics. *IEEE Transactions on Software Engineering*, 24(6):491–496.

References VII

[Hartz et al., 1996] Hartz, M. A., Walker, E. L., and Mahar, D. (1996). *Introduction to Software Reliability: A State of the Art Review*. Reliability Analysis Center, Rome, NY.

[Hendershot, 2015] Hendershot, S. (2015). Facing Up to Government IT Project Failures.

[IBM, 1981] IBM (1981). Implementing Software Inspections. Course Notes.

[Jackson, 1975] Jackson, M. A. (1975). *Principles of Program Design*. APIC. Academic Press, Inc., Orlando, FL, 1st edition.

[Jones, 1991] Jones, C. (1991). *Applied Software Measurement: Assuring Productivity and Quality*. McGraw-Hill, Inc., New York, NY, USA.

References VIII

[Jones, 1996] Jones, C. (1996). Software defect-removal efficiency. *Computer*, 29(4):94–95.

[Kernighan and Plauger, 1978] Kernighan, B. W. and Plauger, P. J. (1978). *The Elements of Programming Style*. McGraw-Hill, New York.

[Koopman, 2014] Koopman, P. (2014). A Case Study of Toyota Unintended Acceleration and Software Safety. *Better Embedded System SW* (blog).

[Lorenz and Kidd, 1994] Lorenz, M. and Kidd, J. (1994). *Object-oriented Software Metrics: A Practical Guide*. Prentice-Hall, Inc., Upper Saddle River, NJ.

References IX

[Matsudaira, 2017] Matsudaira, K. (2017). Does Anybody Listen to You? *Queue*, 15(1):20:5–20:10.

[Myers, 1978] Myers, G. J. (1978). *Composite Structured Design*. Van Nostrand Reinhold.

[Myers, 1979] Myers, G. J. (1979). *The Art of Software Testing*. John Wiley & Sons, Inc., New York, NY.

[Nuseibeh, 1997] Nuseibeh, B. (1997). Ariane 5: Who Dunnit? *IEEE Software*, 14(3):15–16.

[OMG, 2003] OMG (2003). OMG Unified Modeling Language Specification.

References X

[O'Sullivan, 2009] O'Sullivan, B. (2009). Making Sense of Revision-Control Systems. *Communications of the ACM*, 52(9):56–62.

[Pfleeger and Atlee, 2006] Pfleeger, S. L. and Atlee, J. M. (2006). *Software Engineering: Theory and Practice*. Pearson Prentice Hall, 3rd edition.

[Pigoski, 1996] Pigoski, T. M. (1996). *Practical Software Maintenance: Best Practices for Managing Your Software Investment*. Wiley Publishing, 1st edition.

[Pirsig, 1974] Pirsig, R. M. (1974). *Zen and the Art of Motorcycle Maintenance*. William Morrow.

References XI

[Polya, 1945] Polya, G. (1945). *How to Solve It: A New Aspect of Mathematical Method*. Princeton University Press.

[Pressman, 2010] Pressman, R. (2010). *Software Engineering: A Practitioner's Approach*. McGraw-Hill, Inc., New York, NY, 7th edition.

[Prowell et al., 1999] Prowell, S. J., Trammell, C. J., Linger, R. C., and Poore, J. H. (1999). *Cleanroom Software Engineering: Technology and Process*. Addison-Wesley Longman Publishing, Inc., Reading, Massachusetts.

[Scharer, 1981] Scharer, L. (1981). Pinpointing Requirements. *Datamation*, 27(4):139–140.

References XII

[Schleimer et al., 2003] Schleimer, S., Wilkerson, D. S., and Aiken, A. (2003). Winnowing: Local Algorithms for Document Fingerprinting. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 76–85.

[Seacord et al., 2003] Seacord, R. C., Plakosh, D., and Lewis, G. A. (2003). *Modernizing Legacy Systems: Software Technologies, Engineering Process and Business Practices*. Addison-Wesley Professional, 1st edition.

[Thorp, 2013] Thorp, F. (2013). 'Stree tests' show Helathcare.gov was overloaded.

[Weinberg, 1971] Weinberg, G. M. (1971). *The Psychology of Computer Programming*. van Nostrand Reinhold Ltd., New York.

References XIII

[Weinberg, 1986] Weinberg, G. M. (1986). *On Becoming a Technical Leader*. Dorset House Publishing Company, Inc.

[Zeller, 1999] Zeller, A. (1999). Yesterday, My Program Worked. Today, It Does Not. Why? In *Proceedings of the 7th European Software Engineering Conference / 7th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ESEC/FSE-7, pages 253–267.

Outline

Glossary I