JOHNS HOPKINS UNIVERSITY | Engineering for Professionals

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].

Review

Outline

Why Study Software Engineering?

Large software systems are difficult to create but not due to any one problem

- Challenge is to find real solutions to real problems on actual schedule with available resources

Image source: http://commons.wikimedia.org/wiki/File:La_Brea_Tar_Pits.jpg

Software Engineering

Realities

- a concerted effort should be made to understand the problem before a software solution is developed
- design becomes a pivotal activity
- software should exhibit high quality
- software should be maintainable

Figure: Unplanned vs. disciplined approaches to software

Image source: http://commons.wikimedia.org/wiki/File:Spaghetti_all'_arrabbiata.jpg
Image source: http://commons.wikimedia.org/wiki/File:Mexican_lasagne_with_parsley,_April_2011.jpg

Definitions of Software Engineering

Definition (Software Engineering)

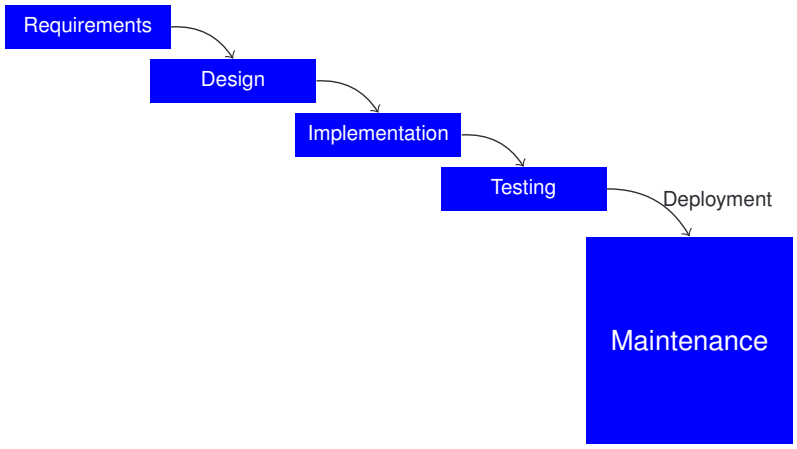
[Software engineering is] the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines. (Bauer, 1969; emphasis added)

Definition (Software Engineering)

1. The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
2. The study of approaches as in 1.

(IEEE, 1993; emphasis added)

Maintenance accounts for about 70–90% of the total lifecycle budget of a software project [Pigoski, 1996; Seacord et al., 2003]



Outline

Outline

Git

CUEBALL: This is Git. It tracks collaborative work on projects through a beautiful distributed graph theory tree model.

PONYTAIL: Cool. How do we use it?

CUEBALL: No idea. Just memorize these shell commands and type them to sync up. If you get errors, save your work elsewhere, delete the project, and download a fresh copy.

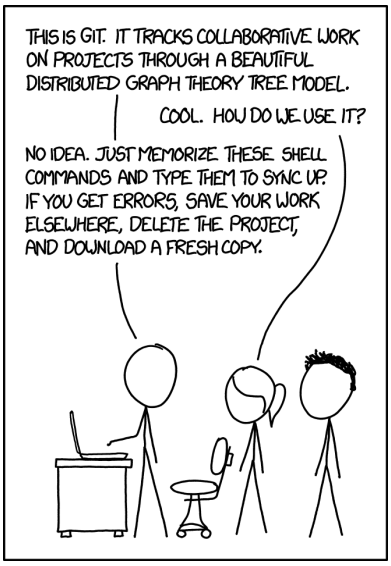


Image: https://xkcd.com/1597/

Common Commands

Action	Command
Create a new repository	`git init`
Track files	`git add`
View changes	`git status`
Undo changes	`git checkout`
Commit changes	`git commit`
Show history	`git log`
Pull changes from another repository	`git pull`
Push changes to another repository	`git push`
Merge changes	`git merge`

Branching

So it turns out that branching is really helpful when working with Phabricator...

- A branch allows you to keep your work separate from ongoing development
- Changes should be developed in branches
  - Differential allows all the code in your branch to be reviewed at once

Branches and Code Review

Workflow

```
# starting on master branch (the default branch)
git branch name-of-feature-branch
git checkout name-of-feature-branch
# make changes to code
git add ...
git commit
arc diff

# return to master branch to start new work
git checkout master
git pull  # fetch latest changes
```


Project Management

Importance of Project Management

> I've visited dozens of commercial shops, both good and bad, and I've observed scores of data processing managers, again, both good and bad. Too often, I've watched in horror as these managers futilely struggled through nightmarish projects, squirmed under impossible deadlines, or delivered systems that outraged their users and went on to devour huge chunks of maintenance time. (Page-Jones, 1985)

Weak project management is disastrous for software projects

The Four Ps

People	The most important element of a successful project
Product	The software to be built
Process	The set of framework activities and software engineering tasks to get the job done
Project	All work required to make the product a reality

The order is not arbitrary!

People

IEEE study indicates that people are the most critical part of a successful software project

Product

Objectives and scope dictate cost estimates, risks, breakdown of tasks, and manageable schedule

- Understanding objectives and scope allow the exploration of alternatives

Objectives The overall goals for the product without considering how to achieve those goals

Scope The primary behaviors that characterizes the product and quantitative bounds on its characteristics

Process

The software process provides the framework for a software development plan

- Most tasks are adapted to the project—not a “one size fits all” approach!
- Some activities apply to all projects—e.g., quality assurance and configuration management

Project

Complexity of software project dictates the amount of project management

Study of 250 large software projects between 1998 and 2004 [Jones, 2004] revealed

- $\approx$ 25 achieved schedule, cost, and quality objectives
- $\approx$ 50 had delays or overruns <35%
- $\approx$ 175 had major delays or overruns or terminated without completion

Outline

People

IEEE study indicates that people are the most critical part of a successful software project

- Too often people are taken for granted—managers' actions do not match their words

Image source: Scott Adams, *Dilbert*, 22 September 1995. Online: http://dilbert.com/strips/comic/1995-09-22/

Stakeholders

Customers Specify the requirements for the software to be engineered

End-users Interact with the software once it is released

Senior managers Define the business issues that significantly influence the project

Project (technical) managers Plan, motivate, and organize the practitioners who do software work

Practitioners Provide the technical skills to engineer the product

(partial list)

Outline

Team Leader

People skills are critical—technical skills are insufficient

Important Habits of Effective Managers

- Even-keeled bosses
- Helping people puzzling through problems
- Interest in employees' lives and careers

Michael Schneider, "Google Employees Weighed In on What Makes a Highly Effective Manager (Technical Expertise Came in Last)," *Inc.*, 20 June 2017 (Online: https://goo.gl/nQ3GG6; accessed 18 July 2017)

Team Leader (continued)

Responsibilities

- Concentrate on problem,
- Manage flow of ideas, and
- Ensure that quality counts and will not be compromised

Team leader exercises "loose" control, stepping in only when necessary

- Allow people to follow their instincts
- Reward achievement and not exclude risk
- Remain under control in high-stress situations

MOI Model [Weinberg, 1986]

Model for technical leadership

Motivation Ability to encourage (by "pushing" or "pulling") people to produce to their best ability

Organization Ability to adapt processes (or invent new ones) to enable the initial concept to be translated into a final product

Innovation Ability to encourage people to create within the bounds established for a particular product

MOI Model (continued)

How to be a failure:

- Kill the motivation: Make people feel unappreciated, do everything for them, and discourage anything that they enjoy doing for its own sake
- Foster chaos: Encourage competition to prevent cooperation, do not provide the necessary resources, and suppress information of value
- Suppress the flow of ideas: Don't listen when you can criticize, punish those who offer suggestions, and prevent people from working together

Outline

Software Team

Not every group is a team, and not every team is effective. ∼ Glenn Parker

Putting a bunch of people to work on the same problem doesn't make them a team—as the sloppy performance in all-star games should teach us. [Weinberg, 1971]

Structure

Software teams are typically dictated by the company's organizational structure, but. . .

. . . the following factors must be considered when selecting a software project team structure:

- the difficulty of the problem to be solved
- the size of the resultant program(s) (e.g., lines of code)
- the time that the team will stay together (team lifetime)
- the degree to which the problem can be modularized
- the required quality and reliability of the system to be built
- the rigidity of the delivery date
- the degree of communication required for the project

Organizational Paradigms [Constantine, 1993]

Closed Structures a team along a traditional hierarchy of authority

Random Structures a team loosely and depends on individual initiative of the team members

Open Emphasizes collaboration and consensus-based decision making but with traditional team roles

- Attempts to maintain some of the control of the *closed paradigm* without precluding the innovation of the *random paradigm*

Synchronous Relies on the natural compartmentalization of a problem so team members work on pieces of the problem with little active communication

Chief Programmer Team [Baker, 1972]

Early organizational paradigm similar to *closed paradigm*

Senior Engineer Plans, coordinates, and reviews all technical activities of team

Technical Staff Conduct analysis and development activities

Backup Engineer Supports the senior engineer

Specialists Includes (e.g.) language expert and database designer

Support Staff Includes (e.g.) technical writers and administrative assistants

Software Librarian Maintains versions of software

Comparison of Organizational Paradigms

	Amount of Structure	
	High	Loose
Project size	Large	Small
Certainty	High	Low
Repetition	Similar to past projects	New technology
Accountability	Formal	Informal
Contributors	Specialized	Generalized

Table: Recommended organizational structure based on project characteristics. Adapted from Pfleeger and Atlee [Pfleeger and Atlee, 2006].

Desired Qualities of Team

A high-performance team should

- trust each other,
- have distribution of skills appropriate for problem, and
- exclude mavericks to maintain cohesion

But note:

> If the people on the project are good enough, they can use almost any process and accomplish their assignment.

(Cockburn and Highsmith, 2001)

Avoid Team "Toxicity"

- A frenzied work atmosphere in which team members waste energy and lose focus on the objectives of the work to be performed.
- High frustration caused by personal, business, or technological factors that cause friction among team members.
- Fragmented or poorly coordinated procedures or a poorly defined or improperly chosen process model that becomes a roadblock to accomplishment.
- Unclear definition of roles resulting in a lack of accountability and resultant finger-pointing.
- Continuous and repeated exposure to failure that leads to a loss of confidence and a lowering of morale.

Preventing Team "Toxicity"

Toxin	Prevention
Frenzied work atmosphere	Major objectives not modified unless absolutely necessary
Frustration and friction	Delegate as much responsibility as possible to team
Fragmented process	Allow team to select process model
Unclear roles	Team responsibile for accountability
Repeated failure	Team-based techniques for feedback and problem solving

Agile Teams

"Agile" methodologies are an oft-suggested antidote to problems

- Team members must have trust in one another.
  - Note: Trust refers to individuals' intentions (i.e., they want the project to succeed) and does not eliminate objective criticism and verification of work products
- The distribution of skills must be appropriate to the problem.
- Mavericks may have to be excluded from the team, if team cohesiveness is to be maintained.
- Team is "self-organizing"
  - An adaptive team structure that uses elements of Constantine's paradigms [Constantine, 1993]
  - Significant autonomy

Team Coordination and Communication

Effective communication is critical to coordinate people

Question

How many intercommunication paths are there for...

- 2 people? 1 communication path
- 3 people? 3 communication paths
- 4 people? 6 communication paths
- 5 people? 10 communication paths
- n people? $n(n-1)/2$ communication paths

Outline

What about Meetings?

A meeting is a place where you keep the minutes and lose the hours.

Most common complaints about meetings:

- The purpose of the meeting is unclear
- Meeting participants are not prepared
- Key people are absent or late
- The conversation veers off track
- Meeting participants don't discuss issues—they dominate the conversation, argue, or do not participate
- No follow-through on decisions

Catherine Dressler, "We've Got to Stop Meeting Like This," *Washington Post*, 31 December 1995. Online: https://goo.gl/LMpd4J (Accessed: 1 August 2016)

Productive Meetings

How to ensure that a meeting is productive:

- Identify participants, duration, and desired outcomes
- Have a written agenda and distribute it in advance
- Have a *facilitator* keep the meeting on track
- Record decisions and track the resulting actions
- Minimize the number of meetings and number of required participants

Catherine Dressler, "We've Got to Stop Meeting Like This," *Washington Post*, 31 December 1995. Online: https://goo.gl/LMpd4J (Accessed: 1 August 2016)

Outline

Scope

Context How does the software to be built fit into a larger system, product, or business context and what constraints are imposed as a result of the context?

Information objectives What customer-visible data objects are produced as output from the software? What data objects are required for input?

Function and performance What function does the software perform to transform input data into output? Are any special performance characteristics to be addressed?

Software project scope must be unambiguous and understandable at the management and technical levels

Problem Decomposition

- Sometimes called partitioning or problem elaboration
- Once scope is defined, the problem. . .
  - is decomposed into constituent functions,
  - is decomposed into user-visible data objects, or
  - is decomposed into a set of problem classes
- Decomposition process continues until all functions or problem classes have been defined

Outline

Process

The software project should dictate the process model

- The inverse should not be true!

Outline

Project

Projects get into trouble when. . .

- Software people don't understand their customer's needs
- The product scope is poorly defined
- Changes are managed poorly
- The chosen technology changes
- Business needs change (or are ill-defined)
- Deadlines are unrealistic
- Users are resistant
- Sponsorship is lost (or was never properly obtained)
- The project team lacks people with appropriate skills
- Managers (and practitioners) avoid best practices and lessons learned

90-90 Rule

Jaded software engineering viewpoint. . .

Rule of Credibility [Bentley, 1985]

The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time. (Tom Cargill)

How to Avoid Common Problems?

- Set realistic objectives and expectations
- Give team autonomy and authority
  - Management should stay out of team's way!
- Emphasize quality and minimize personnel turnover
- Track progress
- Make smart decisions
  - Buy vs. build
  - Avoid obvious risks
- Postmortem analysis
  - Learn how to improve
  - Formally document the lessons learned

How to Avoid Common Problems?

What can you do now?

- Avoid projects with high personnel turnover
- Track progress, *especially your own*
- Conduct "retrospectives" of your own work
  - What could you have done better?

Critical Practices

- Formal risk management
- Empirical cost and schedule estimation
- Metrics-based project management
- Earned value tracking
- Track defects against quality targets
- People-aware project management

Outline

Summary

Project management is an essential part of software engineering

- People are the pivotal element in (successful) software projects

W^{5}HH Principle [Boehm, 1996]

How to define key project characteristics

- Why is the system being developed?
- What will be done?
- When will it be accomplished?
- Who is responsible?
- Where are they organizationally located?
- How will the job be done technically and managerially?
- How much of each resource (e.g., people, software, tools, database) is needed?

Applies regardless of project size or complexity

Review Techniques

Reviews

> . . . there is no particular reason why your friend and colleague cannot also be your sternest critic. [Weinberg, 1971]

What Are Reviews?

- A meeting conducted by technical people for technical people
- A technical assessment of a work product created during the software engineering process
- A software quality assurance mechanism
- A training ground

What Reviews Are Not

- A project summary or progress assessment
- A meeting intended solely to impart information
- A mechanism for political or personal reprisal!

Lexicon

Definition (Fault)

"A physical defect or flaw within a hardware or software component" [Rogers, 2009]

Definition (Error)

"The manifestation of a fault: a deviation from accuracy or correctness in state" [Rogers, 2009]

Definition (Failure)

An externally-visible (e.g., user-visible) event representing a deviation from the system's required behavior [Pfleeger and Atlee, 2006; Rogers, 2009]

What about bugs?

The term "bug" suggests that faults originate from external sources

Faults originate from. . .

- Missing or unachievable requirements
- Incorrect specification, design, or implementation

Image source: https://commons.wikimedia.org/wiki/File:Buggie_new.png

Defect Amplification

A defect amplification model (IBM, 1981) can be used to illustrate the generation and detection of errors during the design and code generation actions of a software process.

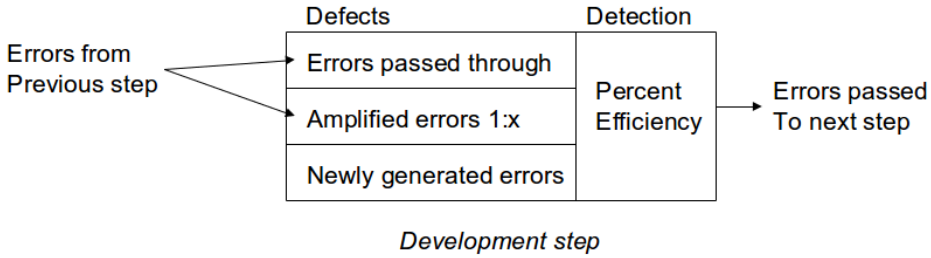


Image source: Pressman, R. (2010). *Software Engineering: A Practitioner's Approach*. McGraw-Hill, Inc., New York, NY, 7th edition

Defect Amplification (continued)

Example

Assume 10 errors exist in the preliminary design. In the detailed design, 6 of these errors will be passed through, 4 amplified by a factor of 1.5:1, and 25 additional errors introduced. In implementation and unit testing, 10 errors will be passed through, the remainder amplified by a factor of 3:1, and 25 additional errors introduced, but unit testing uncovers 20% of the errors. Assume that integration, validation, and system testing are each 50% successful in identifying errors. How many errors are released?

Defect Amplification (continued)

Example (continued)

Assume 10 errors exist in the preliminary design. In the detailed design, 6 of these errors will be passed through, 4 amplified by a factor of 1.5:1, and 25 additional errors introduced.

Defects	
Errors passed through	6
Amplified errors	4
Newly generated errors	25

$$\begin{aligned}\text{Errors} &= \text{Defects} - \text{Defects} \cdot \text{Detection efficiency} \\ &= \text{Defects} \cdot (1.0 - \text{Detection efficiency}/100) \\ &= (6 + 4 \cdot 1.5 + 25) \cdot (1.0 - 0.0) \qquad = 37\end{aligned}$$

Defect Amplification (continued)

Example (continued)

In implementation and unit testing, 10 errors will be passed through, the remainder amplified by a factor of 3:1, and 25 additional errors introduced, but unit testing uncovers 20% of the errors.

Defects	
Errors passed through	10
Amplified errors	27
Newly generated errors	25

$$\begin{aligned}\text{Errors} &= \text{Defects} - \text{Defects} \cdot \text{Detection efficiency} \\ &= \text{Defects} \cdot (1.0 - \text{Detection efficiency}/100) \\ &= (10 + 27 \cdot 3 + 25) \cdot (1.0 - 0.2) \qquad = 93\end{aligned}$$

Defect Amplification (continued)

Example (continued)

Assume 10 errors exist in the preliminary design. In the detailed design, 6 of these errors will be passed through, 4 amplified by a factor of 1.5:1, and 25 additional errors introduced. In implementation and unit testing, 10 errors will be passed through, the remainder amplified by a factor of 3:1, and 25 additional errors introduced, but unit testing uncovers 20% of the errors. Assume that integration, validation, and system testing are each 50% successful in identifying errors. How many errors are released?

	Defects				
Stage	Pass	Amp.	New	Found	Adv.
Preliminary design	—	—	10	0	10
Detailed design	6	4	25	0	37
Code / Unit test	10	27	25	23	93
Integration test	—	—	—	46	47
Validation test	—	—	—	23	24
System test	—	—	—	12	12

Defect Amplification (continued)

Example

Repeat the prior scenario but assume that reviews uncover 70%, 50%, and 60% of defects in the preliminary design, detailed design, and code / unit test phases. How many errors are released?

	Defects				
Stage	Pass	Amp.	New	Found	Adv.
Preliminary design	—	—	10	7	3
Detailed design	2	1	25	14	15
Code / Unit test	5	10	25	36	24
Integration test	—	—	—	24	12
Validation test	—	—	—	12	6
System test	—	—	—	6	3

Effort With and Without Reviews

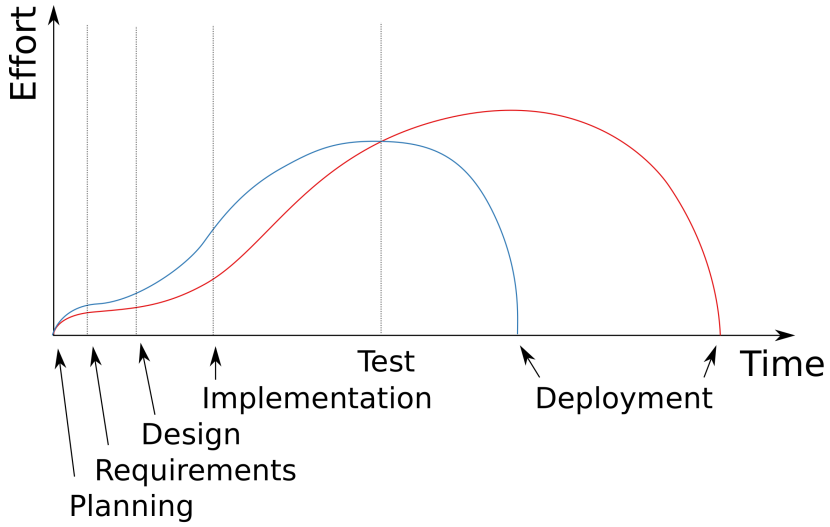


Figure: Effort without reviews (red) and with reviews (blue). Image adapted from [Pressman, 2010].

Informal Reviews

- Informal reviews include. . .
  - a simple desk check of a software engineering work product with a colleague
  - a casual meeting (involving more than 2 people) for the purpose of reviewing a work product, or
  - the review-oriented aspects of pair programming
- Pair programming encourages continuous review as a work product (design or code) is created
  - The benefit is immediate discovery of errors and better work product quality as a consequence

Formal Technical Reviews

- The objectives of a formal technical review are
  - to uncover errors in function, logic, or implementation for any representation of the software,
  - to verify that the software under review meets its requirements,
  - to ensure that the software has been represented according to predefined standards,
  - to achieve software that is developed in a uniform manner, and
  - to make projects more manageable.
- Types
  - Walkthrough
  - Inspection

Walkthrough

Developer presents code and documentation to the review team in an informal setting

- Developer leads and controls discussion

Inspection [Fagan, 1976]

Review team checks code and documentation against a prepared list of concerns

Steps

1. Overview of code and description of goals for the inspection
2. Individual study of code and documentation to note faults
3. Discussion of individuals' findings
   - Some findings may be "false positives"

The Review Meeting

- Between three and five people (typically) should be involved in the review.
- Advance preparation should occur but should require no more than two hours of work for each person.
- The duration of the review meeting should be less than two hours.
- Focus is on a work product (e.g., a portion of a requirements model, a detailed component design, source code for a component)

Conducting the Review

- Review the product, not the producer
- Set an agenda and maintain it
- Limit debate and rebuttal
- Enumerate problem areas, but don't attempt to solve every problem noted
- Take written notes
- Limit the number of participants and insist upon advance preparation
- Develop a checklist for each product that is likely to be reviewed
- Allocate resources and schedule time for reviews
- Conduct meaningful training for all reviewers
- Review your early reviews

Review Options Matrix

	IPR	WT	IN
Trained leader	✗	✓	✓
Agenda	?	✓	✓
Preparation	?	✓	✓
Producer presents	?	✓	✗
"Reader" presents	✗	✗	✓
Recorder	?	✓	✓
Checklists	✗	✗	✓
Errors categorized	✗	✗	✓
Issues list created	✗	✓	✓
Sign-off	✗	✓	✓

Legend

IPR	Informal Peer Review
WT	Walkthrough
IN	Inspection

Sample-Driven Reviews

Attempt to quantify those work products that are primary targets for formal technical reviews. To accomplish this. . .

- Inspect a fraction a_i of each software work product, i. Record the number of faults, f_i found within a_i.
- Develop a gross estimate of the number of faults within work product i by multiplying f_i by $1/a_i$.
- Sort the work products in descending order according to the gross estimate of the number of faults in each.
- Focus available review resources on those work products that have the highest estimated number of faults.

Are reviews successful at identifying faults?

- 67% of system's faults detected via inspections prior to unit testing [Fagan, 1976]
- Inspections remove as many as 85% of faults [Jones, 1996]
  - No single technique (e.g., testing) identifies more than 60% of faults
- Faults found during discovery activities [Jones, 1991]

Activity	Faults per KLOC
Requirements review	2.5
Design review	5.0
Code inspection	10.0
Integration test	3.0
Acceptance test	2.0

How to know when reviews are complete?

If past history indicates that. . .

- the average defect density for a requirements model is 0.6 errors per page, and a new requirement model is 32 pages long,
- a rough estimate suggests that your software team will find about 19 or 20 errors during the review of the document.
- If you find only 6 errors, you've done an extremely good job in developing the requirements model or your review approach was not thorough enough.

For Next Time. . .

Further Information I

Does Anybody Listen to You? [Matsudaira, 2017]

How do you step up from mere contributor to real change-maker?

Highly Effective Managers

- Adam Bryant, "Google's Quest to Build a Better Boss," *The New York Times*, 12 March 2011 (Online: https://goo.gl/tG7k4T; accessed 18 July 2017)
- Henry Blodget, "8 Habits of Highly Effective Google Managers," *Business Insider*, 20 March 2011 (Online: https://goo.gl/qmbz1p; accessed 18 July 2017)
- Michael Schneider, "Google Employees Weighed In on What Makes a Highly Effective Manager (Technical Expertise Came in Last)," *Inc.*, 20 June 2017 (Online: https://goo.gl/nQ3GG6; accessed 18 July 2017)

Reading

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

"Don't waste time on Code Reviews"[1]

[1] http://swreflections.blogspot.fr/2014/08/dont-waste-time-on-code-reviews.html or http://goo.gl/8CyCSK

Project Overview

The primary intent of the class project is hands-on experience applying the principles taught in this course. Project development is ongoing with teams contributing to the open source code across individual class offerings *as a way to experience the challenges of contributing to existing code bases*.

Plagiarism Detection

This project will create a toolkit for source code plagiarism detection. The toolkit will eventually be used to screen source code submissions for possible plagiarism.

CADET: Course Assessment using Data Exploration of Text

Student feedback provides insight into course quality, but few tools support longitudinal analysis across multiple semesters. This project aims to create a toolkit for such analysis to aid the Engineering for Professionals (EP) program

Objectives

The essential objective is to mature the existing implementations, either by adding new features or refactoring and documenting existing code. Specific objectives are described in the project descriptions posted on Blackboard, but only high-level requirements are stated explicitly, as teams are expected to elicit and negotiate requirements with stakeholders as with other software engineering projects.

Project

Use the W^{5}HH principle to define the key characteristics of the project. Be as detailed as practical—the result should indicate considerable thought from a project management perspective.

W^{5}HH Principle [Boehm, 1996]

- Why is the system being developed?
- What will be done?
- When will it be accomplished?
- Who is responsible?
- Where are they organizationally located?
- How will the job be done technically and managerially?
- How much of each resource (e.g., people, software, tools, database) is needed?

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.

References IV

[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.

[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.

References V

[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.

[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).

References VI

[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.

[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.

References VII

[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.

[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.

References VIII

[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.

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

References IX

[Jones, 2004] Jones, C. (2004). Software Project Management Practices: Failure Versus Success. *CrossTalk: The Journal of Defense Software Engineering*, 17(10):5–9.

[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 X

[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 XI

[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 XII

[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.

[Rogers, 2009] Rogers, P. (2009). Software Fault Tolerance. *Ada User Journal*, 30(2):125.

References XIII

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

[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 XIV

[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

EP Engineering for Professionals. 85

KLOC thousands of lines of code. 79