Inspection, Review, Walkthrough

CSE 3311 & 5324, Christoph Csallner
University of Texas at Arlington (UTA)

Shari Lawrence Pfleeger and Joanne M. Atlee: Software Engineering: Theory and Practice. 4th edition, Prentice Hall, 2009. pages 413 ff.

[MC] Steve McConnell: Software Project Survival Guide, Microsoft Press, 1998.

[RF] = Richard E. Fairly, Managing and Leading Software Projects, Wiley, 2009

[CJ] = Capers Jones: Applied software measurement: Assuring productivity and quality, 2nd edition, McGraw-Hill, 1997

[Masterminds] = Federico Biancuzzi and Shane Warden: Masterminds of Programming: Conversations with the Creators of Major Programming Languages, O'Reilly Media, 2009

Power of Reviews (aka Inspections)

- Purpose: Find problems early
  - Before they develop into major headaches
- Comparison of defect removal methods
  - CJ, Figure 5.3, page 373: Need set of techniques

	Requirement defects	Design defects	Code defects	Document defects	Performance defects
Review/ Inspections	**Fair**	**Excellent**	**Excellent**	**Good**	**Fair**
Prototypes	Good	Fair	Fair	n/a	Good
Testing	Poor	Poor	Good	Fair	Excellent
Correctness proofs	Poor	Poor	Good	Fair	Poor

Other Motivation: Tom Love

- Co-Creator of Objective-C (used in iPhone)
- Q: "**How would you train a software engineer?**"
- Tom Love:
  - "I would start off by putting them in a testing group and teach them how to test code and how to read code.
  - **The software business is one of the few places we teach people to write before we teach them to read.** That's really a mistake.
  - It's nothing like picking up a really awful piece of code and trying to figure it out. It turns out to be very instructive.
  - I would also encourage them to become familiar with existing software products that are well designed and well architected so that they are gaining that experience from the inside as contrasted from the outside."

[Masterminds, page 250, reformatted]

Fagan Inspection

- Introduced by Michael Fagan at IBM
  - Classic paper: “Design and code inspections to reduce errors in program development”, IBM Systems Journal, 15(3): 182—211, 1976
- Relatively formal
  - Training, preparation, meeting with fixed roles, follow-up
- Can apply to any project document:
  - Code, design, specification, etc.
- Now also called “Software Inspection”
- Textbook on software inspection:
  - “Software Inspection” by Tom Gilb & Dorothy Graham, Addison-Wesley, 1993 [GG]

Software Inspection

- Original intent:
  - **Find (but not fix) errors in design and code**
  - By checking against upstream documents
  - Example: Check design against requirements

- Not participating: Manager, customer, user rep.
  - Less politics, more technical without manager
  - People should not feel evaluated by inspection
  - Manager participates if manager also has a technical role (more likely in small projects)

Role of Project Manager

- Provide inspection training, if needed
- Include enough time for inspection activities in schedule

- Review inspection results

- Adapt and improve review & development process
  - In response to trends seen in review reports
  - E.g.: Improve checklists

Software Inspection Steps [RF]

- Short planning meeting
  - Assign roles, schedule time / place of inspection meeting
  - Optional: Author-led document walkthrough

- **Reviewers individually study documents**
  - **Each spends 2-3 hours**
  - **Core activity**
  - Use checklist, record bugs

- Main inspection meeting & follow-up

Inspection Meeting (<2 hours): Roles

- Moderator: Chairs / coaches inspection meeting
  - Responsible that meeting is productive
  - Technical person from unrelated project
  - Should get training for this role
- Reader
  - Paraphrases document in inspection meeting
- Recorder:
  - Record bugs discovered in inspection meeting
- Reviewers

- Author: Answers questions in inspection meeting

Inspection Meeting (1)

- Moderator asks for preparation times
  - Excuse unprepared people or reschedule meeting

- Reader presents document
  - Summarizes section N
  - Moderator asks if comments on section N
  - Participants contribute comments from their log
  - If multiple comments, moderator ensures that participants take turns after each comment
  - For each comment, moderator leads discussion if comment should be accepted
  - If comment accepted, recorder adds comment to log

Inspection Meeting (2) – KEY SLIDE

- Record for each accepted defect:
  - **Finder** → for follow-up questions, not for performance
  - **Severity**
    - **Major** = ***Will cause major problems if not fixed now***
    - **Minor** = ***Must be fixed, but ok to fix later***
    - **Open issue** = ***Needs further investigation***
  - One of: **Missing / wrong / extra**
  - **Defect type,** e.g.: Design defect, maintainability
  - **Description – of the defect –** not how to fix it

- No place for low impact “findings”, e.g.:
  - Typo, missing comma

Inspection Meeting (3)

- Everyone contributes bugs they found individually
  - Perhaps find additional bugs together

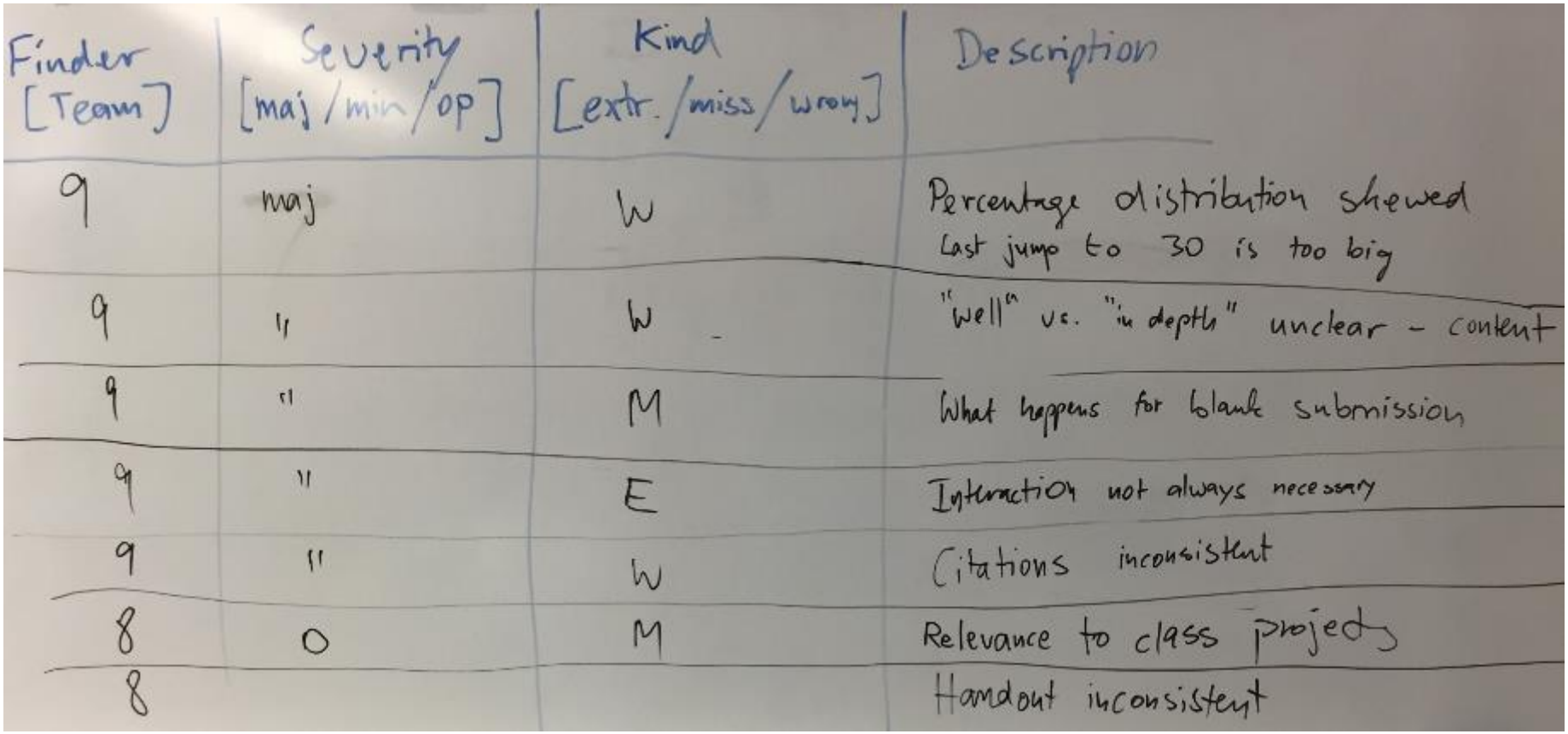
- After going through document:
  - Moderator reviews log with team to ensure team agrees with bugs and understand bug descriptions (i.e., author)

IN-CLASS INSPECTION EXERCISE

In-Class Inspection Exercise

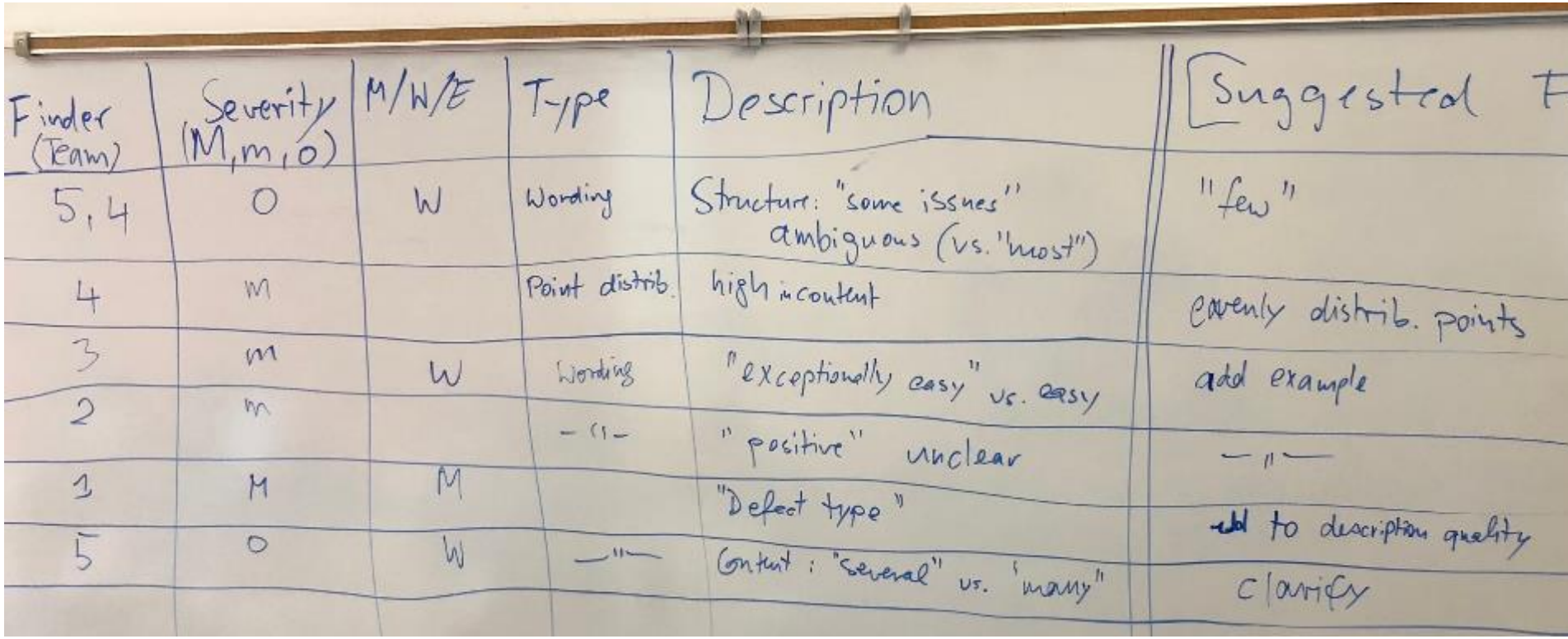
- Get together with your team. Two steps:
- (1) Individually review the handout (for goal of the reviewed document delivering max value). Focus:
  - **Major: Needs immediate fixing**
  - **Minor: Must be fixed, but ok to fix later**
  - **Open question**
- (2) Perform team inspection meeting:
  - Reader presents one box at a time
  - Reviewers describe their findings
  - Scribe keeps log of team findings
- Be prepared to present results

Subset of Results (9/20/17): Reviewed Rubric for Optional Tool Presentation



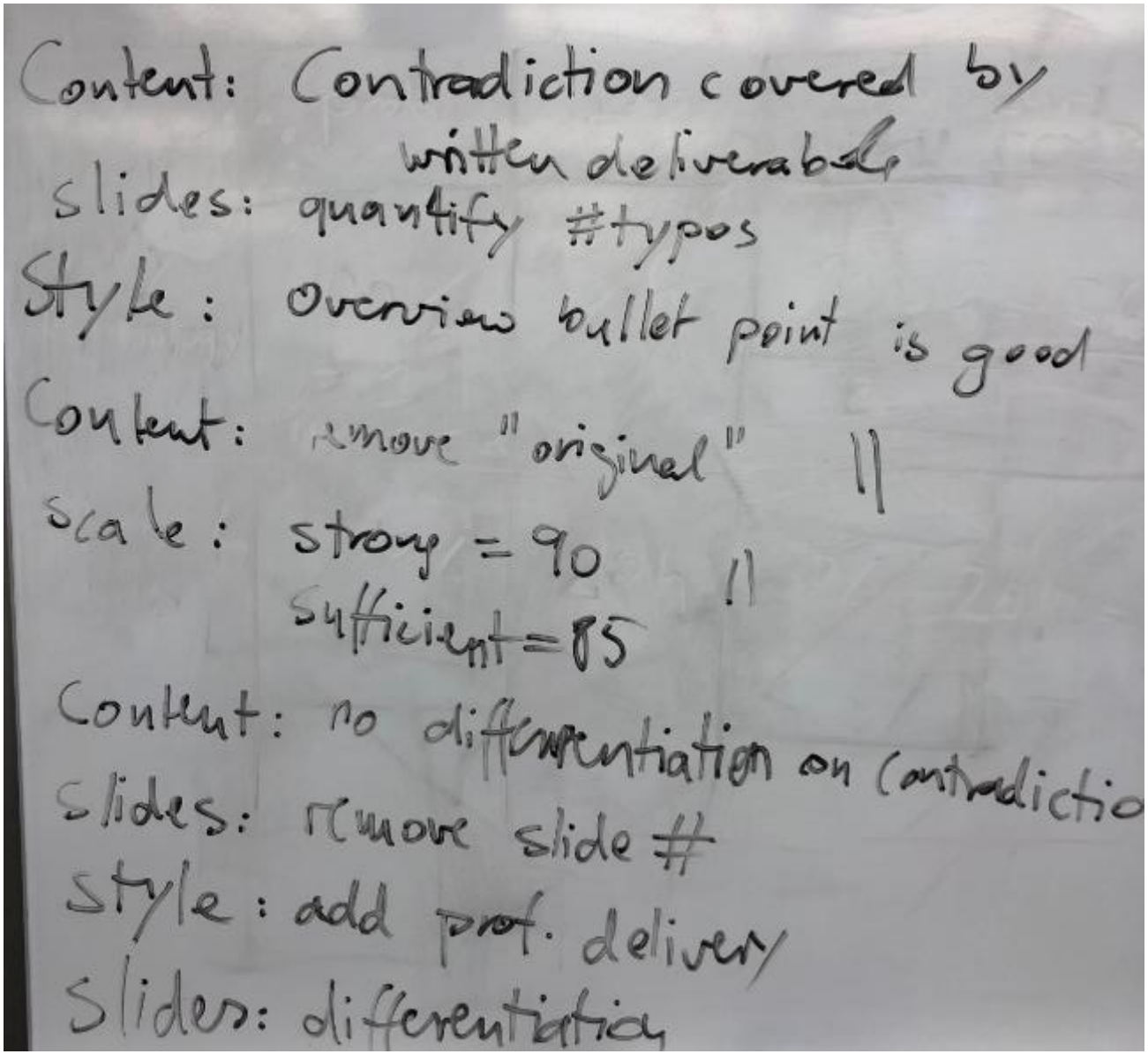
Finder [Team]	Severity [maj / min / op]	Kind [extr. / miss / wrong]	Description
9	maj	W	Percentage distribution shewed Last jump to 30 is too big
9	"	W	"well" vs. "in depth" unclear - content
9	"	M	What happens for blank submission
9	"	E	Interaction not always necessary
9	"	W	Citations inconsistent
8	O	M	Relevance to class projects
8			Handout inconsistent

Subset of Results (2/7/18): Reviewed Rubric for Project Reviews



Finder (Team)	Severity (M, m, o)	M/W/E	Type	Description	Suggested F
5, 4	o	W	Wording	Structure: "some issues" ambiguous (vs. "most")	"few"
4	m		Point distrib.	high in content	evenly distrib. points
3	m	W	Wording	"exceptionally easy" vs. easy	add example
2	m		-"-	"positive" unclear	-"-
1	M	M		"Defect type"	add to description quality
5	o	W	-"-	Content: "several" vs. "many"	clarify

Results (9/17/18): Reviewed Rubric for Project Presentations



Results (1/28/19): Reviewed Rubric for Inception Written Deliverables

Severity	M/W/E	Description
Minor	M	Php → Python ("new / exciting features")
& Open	M	~~deferring~~ justification for project in addition to difference from competitors
Major	M	php → python, competitor: what if there are none?
Minor	E	70 vs. 17 ~~on~~ sound similar on competitors
"	E	Features: "excellent new / exciting" redundant w/ competitors
"	M	▲ exceed/good is very subjective (risks)
D	M	excellent vs good new features
Minor	M	missing: testing
A	W	"exceed expectat" should be "meet exapertation"
Minor	M	risks: php → py: dependence on mentor
"	M	risk types: not clear what are parts of risks
"	M	doc style: what is scope of deliverable: how is it defined?
"	M	doc: UML or commenting
"	W	risks: unrealistic vs "no risks"
D	M	compr. competitors: put side by side?

Inspection Report

- After defects corrected

- Author and moderator prepare review package
  - Cover sheet
  - Inspection defect list
  - Individual preparation logs
  - Inspection summary report

After Inspection Meeting

- Directly after meeting, author may ask for "3rd hour"
  - Informal post-meeting to help author on how to fix bugs
  - **Only if author wants this advice**

- After fixing at least major bugs author and moderator meet
  - Make sure bugs fixed
  - Prepare summary report
  - Moderator creates action items for remaining issues

Inspections Appear Heavyweight

- Several people prepare individually + meeting
- **Empirical data show that inspections are very good use of resources**
- Collect data to convince yourself and management
  - Measure how much time spent on inspection & testing
  - Measure how many bugs found by inspection & testing
- If inspections are not effective for you, you are likely doing them wrong (for your particular environment)
  - Collect data, tweak your inspection process, until you see good results
- Expect inspections to take pressure off subsequent Q&A activities such as testing, as many bugs found

Code Inspection

- Fagan inspection applied to code

- Effectiveness backed by empirical studies
  - Fagan 1976 paper
  - A. Frank Ackerman, Lynne S. Buchwald, and Frank H. Lewski: Software inspections: An effective verification process, IEEE Software, 6(3): 31—36, 1989: http://doi.ieeecomputersociety.org/10.1109/52.28121
  - Capers Jones: Applied software measurement: Assuring productivity and quality, McGraw-Hill, 1991: http://portal.acm.org/citation.cfm?id=109758

Inspection Pitfalls [GG Chapter 11]

- Concentrating on trivial issues: Typos, etc.
  - Takes time away from major / minor / open issues
- **Making / taking bugs personal**
  - Inspections typically focus on high-value assets → Problem found in your code = you work on important stuff
- Inspecting anything regardless of initial quality
  - Fix document first if it is clearly in bad shape
- **Authors and reviewers not using checklists**
  - Prevents the kinds of problems that are on the checklist

Inspection Pitfalls (2)

- **Not publicizing benefits if costs are publicized**
  - Need to collect metrics on which bugs found
- Inspecting without author's agreement
  - More effective if everyone wants to find & fix bugs
- Reviewers pretending to understand the document
  - Volunteering to "not understand the document" is hard
  - Moderator should get such problems out of reviewers

- **Author trying to "resolve" a reviewer confusion by in-meeting explanations**
  - If reviewer did not understand document, then a new team member or maintainer will not understand it either

MC, page 192 ff. ← Microsoft eco system

EXAMPLE OF MODERN INSPECTION: REVIEW OF DETAILED DESIGN

Review of Detailed Design

- **Design review can cut project cost in half**
  - Find & remove missing and unneeded features

- Start when author of design of a component thinks that the design of this component is complete

- Assemble reviewers (review team)
  - Different people find different issues
  - More reviewers → Find more defects
  - Additional cost often does not pay off beyond 3 reviewers
  - **Good size in practice: 2 or 3 people other than author**

Core of Review

- *Each reviewer reviews design individually*
  - Most productive part of this review process

- **Do this part even if you do not have time for other parts**
  - In this case: Each reviewer reports directly to author

Review Meeting

- Discuss findings of individual reviews
- How much to review per meeting?
  - Depends, initially start with one class per meeting (object-oriented programming class)

- Review each detailed design component for:
  - Correctness: Will design work as intended?
  - Completeness: Will design work for all intended purposes?
  - Understandability: Can others understand the design easily?

Understandability

- **Design complex ➔ More coding errors**
- Make design as simple as possible
  - Simple design pays off even beyond initial development
- Average program maintained by 10 generations of maintainers
- Maintainers spend more time on understanding program than on making changes

Traceability to Requirements

- Determine if design
  - Addresses all requirements of specification
  - Addresses additional (superfluous) requirements

- Missing requirements
  - Want to find out now
  - Finding out later will be more expensive, requires rework

- Design of unneeded functionality: Feature creep
  - Increases project scope, risk, cost, schedule
  - May look small now, but will require more coding, testing, debugging, documentation, support

Unneeded Requirements

- Red flag: Developer says “it would be cool if we would do X”
  - For example, X being the latest language / library / process
  - What is the impact of X?
  - Will X make the design more complex?

- **If X increases cost by C and schedule by S, would customer by willing to pay C + S to get X?**
  - In many cases: Customer does not notice if we add or remove X

Importance of Clear Project Vision

- Clear vision provides guidance for design review
  - Recall project plan vision statement

- What are the project objectives?
  - To produce the software cheaply?
  - To make the software reliable, adaptable, portable?

- Reviewers can point out problems if project has a clear vision and reviewers know the objectives

Review As Training

- What if developer Joe gets hit by a milk truck?
  - Will it derail the entire project?
- What if developer Joe is hard to get along with?
- Is there anyone who understands what Joe has been working on?
- Review spreads knowledge to reviewers

EXAMPLE OF MODERN LIGHT-WEIGHT INSPECTION: CODE REVIEW

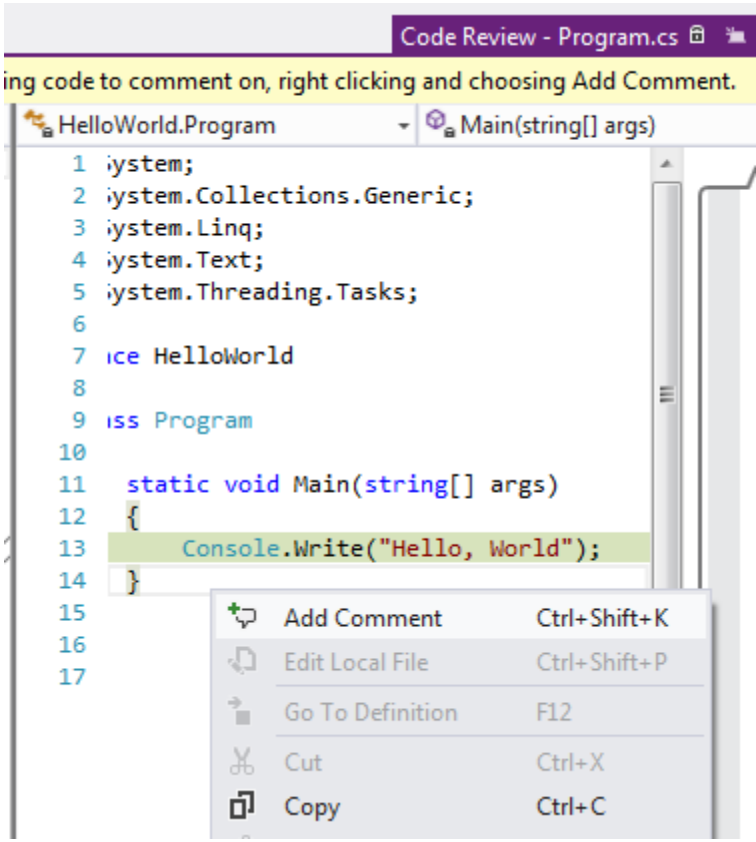
Code Review

- Less formal than code inspection
  - Often all online, no meetings

- **Very common in industry**

Code Review Tools: Microsoft

- Microsoft CodeFlow
  - Used internally by 40k Microsoft developers
  - Collaborative tool
  - Reviewers can annotate code
  - Allows developer to submit code change to reviewers

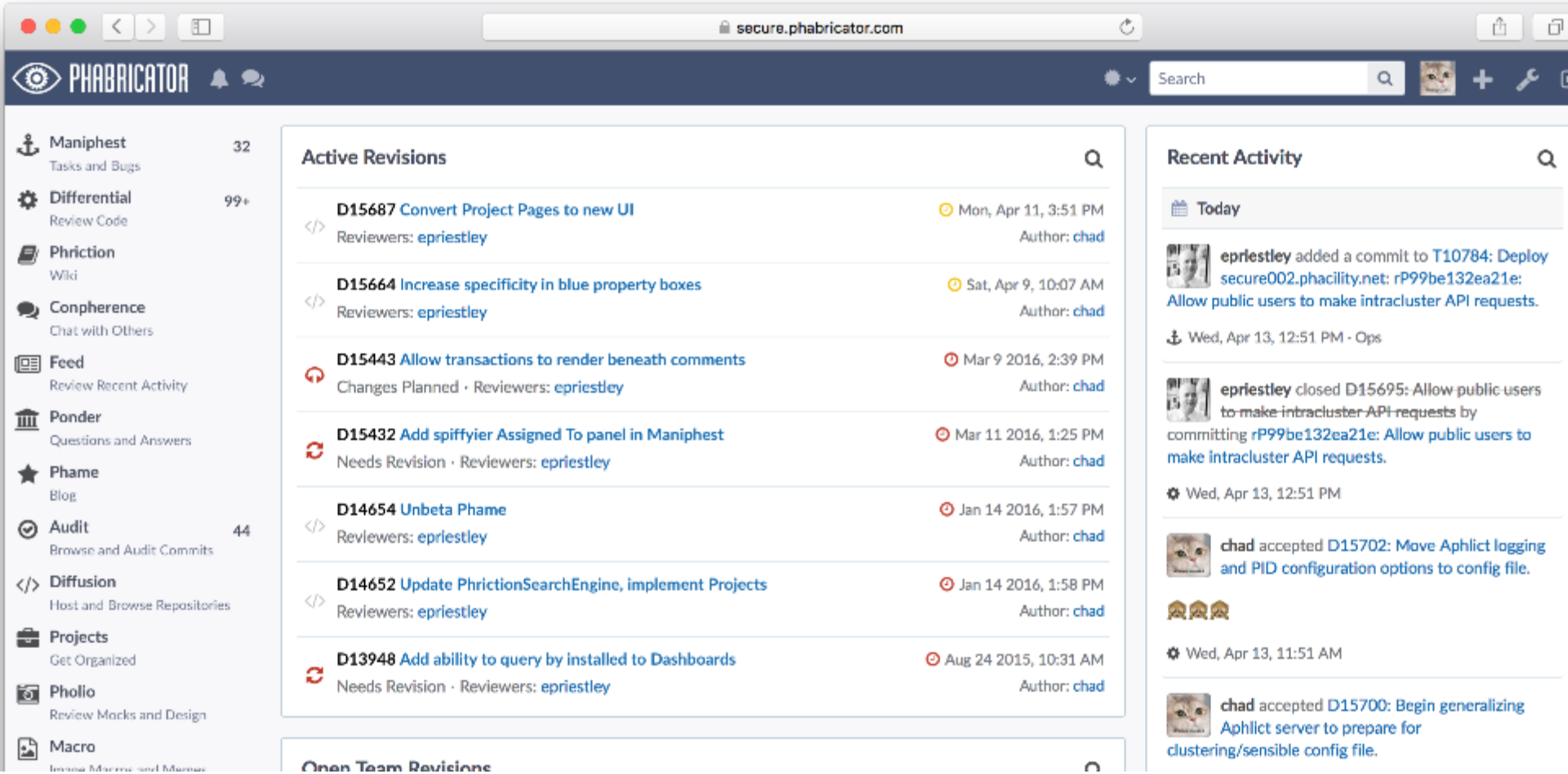


Now integrated in Visual Studio

- https://www.visualstudio.com/en-us/docs/tfvc/get-code-reviewed-vs

Code Review Tools: Facebook

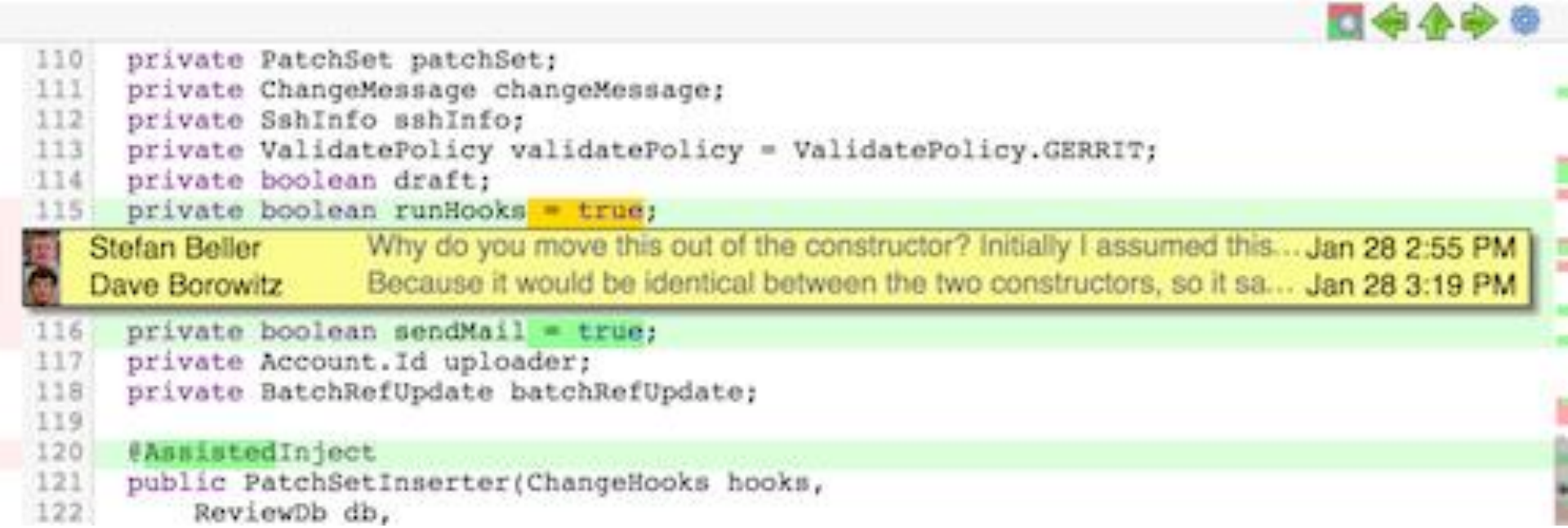
- Phabricator (open source) → Phorge
  - https://www.phacility.com/ → https://we.phorge.it/



[https://www.phacility.com/]

Code Review Tools: Google

- Gerrit (open source)
  - https://www.gerritcodereview.com/



  - Used by several high-profile projects
    - Android: https://android-review.googlesource.com
    - Chromium: https://chromium-review.googlesource.com
    - Eclipse: https://git.eclipse.org/r

[https://www.gerritcodereview.com/]

More Review Tools

- Review Board (open source):
  - http://www.reviewboard.org/
  - Also review text files, images, etc.

[https://www.reviewboard.org/]

Pre- vs. Post-Commit Review

- Pre-commit
  - Code reviewed before author commits code to repository
  - If review results in changes, only final version committed
  - Does not expose others to intermediate version

- Post-commit
  - Author commits code changes, then submits changes for review
  - Initial commit faster than in pre-commit model
  - If commit requires changes, then the changes are committed subsequently
  - Usually not done when committing to main code branch

Jason Cohen: 11 proven practices for more effective, efficient peer code review, 2011:

http://www.ibm.com/developerworks/rational/library/11-proven-practices-for-peer-review/ (link no longer works)

CISCO 2011 STUDY

Background

- Possible **conflict of interest**
  - Study reports on using SmartBear product
  - Study conducted and reported by SmartBear developers

- Some insights are still interesting

- Large study on code review
  - Cisco team developing MeetingPlace product
  - 50 developers on 3 continents:
    In Bangalore, Budapest, and San José
  - 2500 code reviews over 10 months

Code Review Process in Case Study

- Review process enforced by tools:
- Code must be reviewed before it can be checked in
  - Also common in other large companies
- In-person meetings for reviews not allowed
- Tools collected review metrics

Best Practices (1)

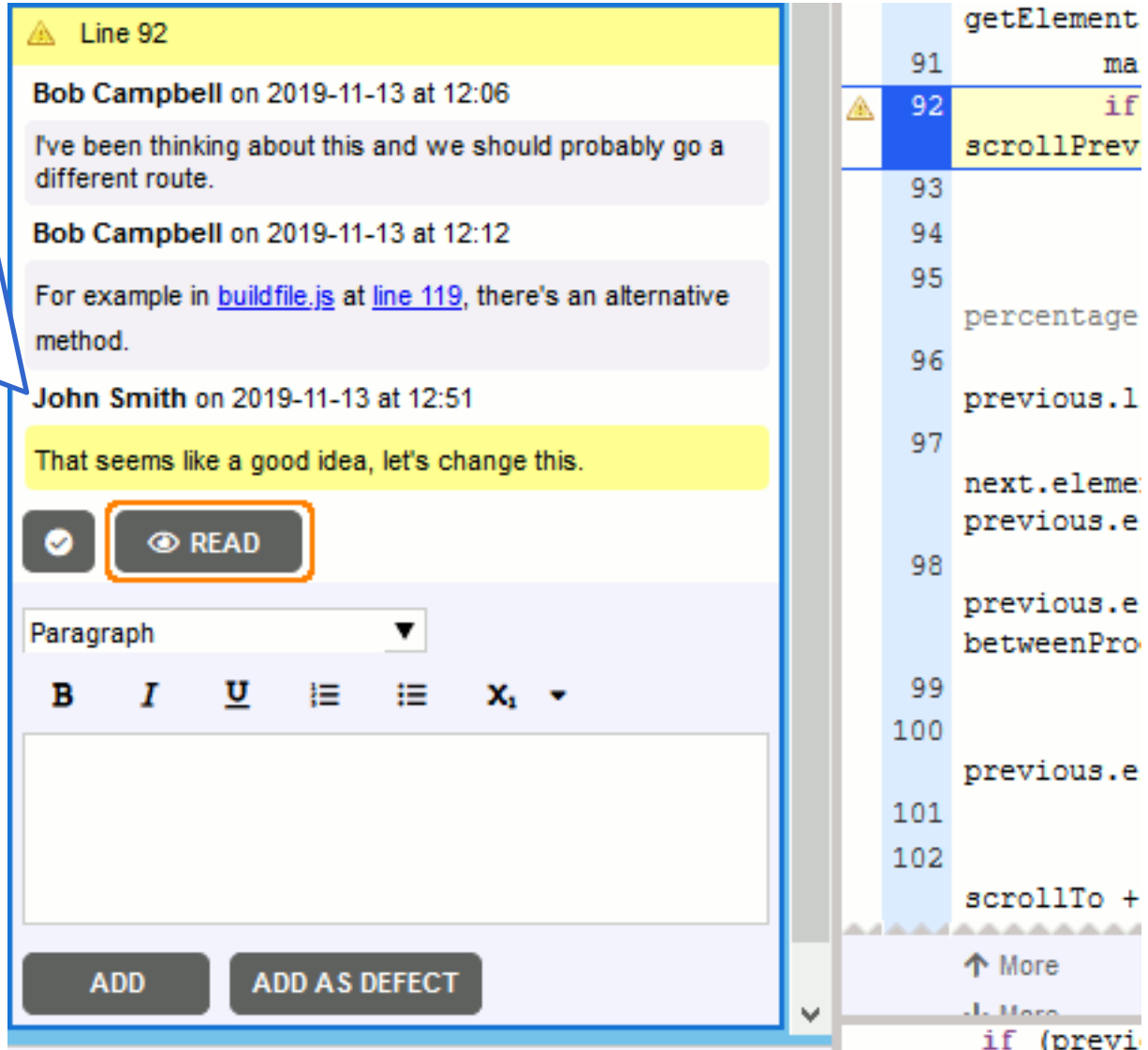
- Do not review longer than 60-90 minutes at a time
  - Concentration drops off after 60 minutes

- **Annotated code has fewer defects**
  - Annotation guides reviewer through code (changes)
  - Annotation explains and defends commit to reviewer
  - Annotation ≠ code comment
  - **Developer spends extra effort to think through changes again and write annotations**
  - Developer finds bugs while preparing annotations

Annotating Code Before Review

- https://support.smartbear.com/collaborator/docs/working-with/review-participate.html#annotating-review

Current product lets all review participants annotate ***before*** review.
But makes most sense for author.
Reviewers can comment during review



Best Practices (2)

- **Make sure that bugs found by reviews are fixed**
  - Otherwise the review goes to waste
  - Add found bugs in bug tracking system

- Maintain a positive review culture
  - Management should treat found defect as positive
  - Much better to find defect now than finding it later
  - Do not use it to assign blame, instead:
    Emphasize that it is a team effort to find bugs
  - Learning, growing, communication
  - Negative attitude can be poisonous

Best Practices (3)

- Make it clear that collected metrics will not be used for annual performance evaluation, raises, etc.
- Do not use metrics to single out developers
  - Does X take less time than me to finish a review?
  - Else: Developers game metrics

- **Most difficult parts of problem often handled by most experienced developers**
  - This code is often reviewed most thoroughly and has the highest number of bugs found
  - Number of bugs found correlates with feature complexity and importance

Best Practices (4)

- Some positive effects remain even if not all code is reviewed before it is checked in
  - **Probability of a code review motivates developer to double-check own code before checking code in**
  - But motivation disappears once review probability becomes too small

CHECKLISTS

Motivation for Checklists

- Encode best practices
- Reminder for both authors and reviewers
  - Author can make sure major items are included
- Reviewer will usually get document that satisfies checklist
  - Reviewer can focus on other issues

Checklists Overview

- Does the design satisfy the requirements?
  - Detect bugs = design deviates from requirements
  - Detect design that is internally inconsistent
- Do the test cases satisfy the requirements?
  - Do the test cases test the requirements?
  - Are important test cases missing?
- Does the code satisfy the design?
  - Detect bugs = code deviates from design or requirements
  - Detect inconsistencies between code and comments
  - Detect complex code that is left uncommented

Example Checklists

- Following are example checklists from popular books

- **These are not “the best” or only checklists**
- Several items will not apply to your project

- Each organization should develop its own checklists
  - Start with existing checklists
  - Keep refining checklists for your environment

Example checklists from:
"Applied Software Project Management" by Andrew Stellman & Jennifer Greene, O'Reilly, 2006

EXAMPLE CHECKLIST FOR PROJECT PLAN

Example Checklist: Project Plan

- Statement of Work (SOW)
  - Does project plan include statement of work?
  - Does SOW contain all features that will be developed?
  - Are all work products represented?
  - If estimates are known, are they included?

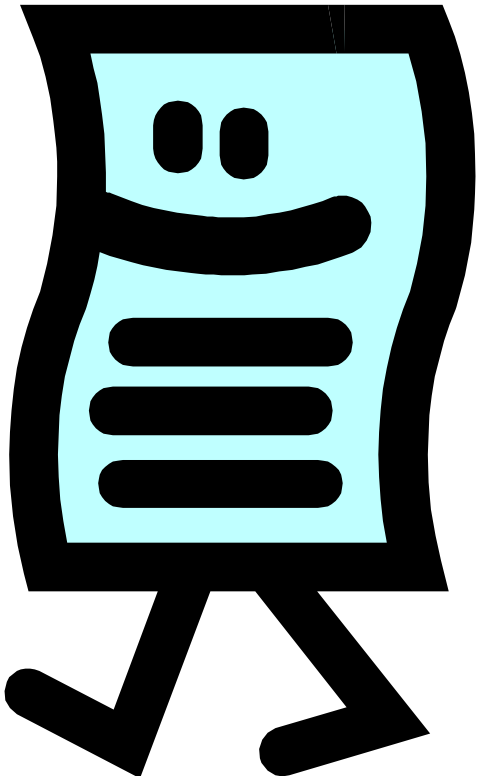
- Resources
  - Does the project plan include a list of resources?
  - Does the resource list contain all available resources?
  - Are any of these resources known to be assigned to another project at the same time?
  - Are known resource unavailability times included?

Example Project Plan Checklist (2)

- Project Schedule
  - Does the project plan include a schedule?
  - Are any tasks missing or incorrect?
  - Does each task have a predecessor?
  - Are resources allocated for each task?
  - Is there a better way to allocate resources to tasks?
  - Does the schedule contain periodic reviews?

Example Project Plan Checklist (3)

- Risk Plan
  - Does the project plan include a risk plan?
  - Does the risk plan miss important risks?
  - Is each risk prioritized correctly?
  - **Is each risk impact estimate realistic?**
  - **Does each risk have a mitigation plan?**



"Code Complete" by Steve McConnell,
2nd edition, Microsoft Press, 2004

EXAMPLE CHECKLIST FOR REQUIREMENTS

Functional Requirements

- **Are all system inputs specified?**
  - Source, accuracy, value range, frequency
- **Are all system outputs specified?**
  - Destination, value range, frequency, format
- **Are all external interfaces specified?**
  - Hardware, software, communication
- Are all use cases specified?
  - Including data used in and produced by each task

Quality Requirements

- Is response time for each operation specified?
  - From the user's perspective
- Are internal performance goals specified?
  - Processing times, system throughput
- Is the security level specified?
- Is the reliability level specified?
  - Consequences of software failure, error detection & recovery
- Are minimum required resources specified?
- Is the maintainability specified?
  - Adapt to changes in environment

Requirements Quality (1/2)

- Are requirements in user's language?
  - **Does user agree?**
- Do any requirements conflict with each other?
  - Are trade-offs specified?
- Does any requirement require more (less) detail?

Requirements Quality (2/2)

- Can different team understand requirements?
  - **Do such other developers agree?**
- Is every item relevant to problem and solution?
  - Can each item be traced back to problem?
- **Is it easy to test each requirement?**
- Are likely changes (& change likelihood) captured?

Requirements Completeness

- Is missing information noted?
- **Will the product be accepted if it implements all listed requirements?**
- Can you implement all requirements?

“Code Complete” by Steve McConnell, 2nd ed., Microsoft Press, 2004

EXAMPLE CHECKLIST FOR ARCHITECTURE

Architecture Specifics (1)

- Is the overall program organization clear?
  - Good overview, good justification
- Are the major components well defined?
  - Responsibilities, interfaces to other components
- Are all tasks of requirements covered by a reasonable amount of components?
- Are the most critical modules (classes) described and justified?
- **Are the data structures described and justified?**
  - Including databases
- Are important business rules described and justified?

Architecture Specifics (2)

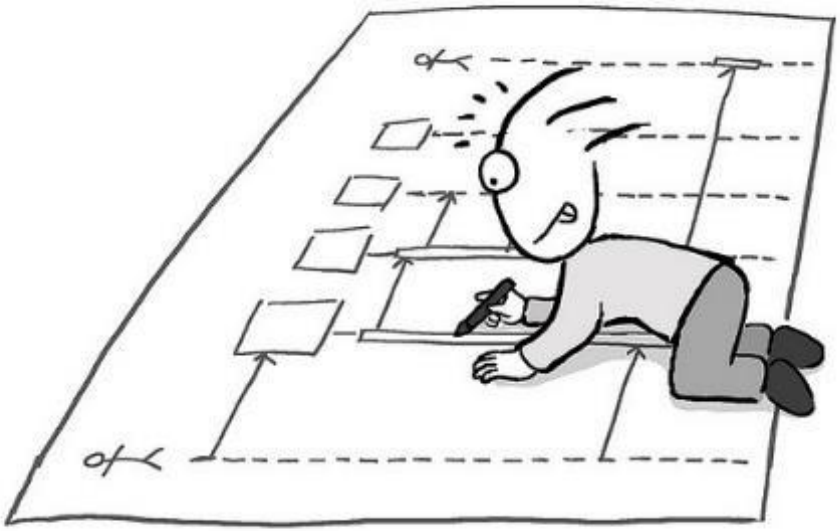
- Is the UI design strategy described?
  - Is the UI design decoupled from the rest of the system so UI changes won't affect rest of system?
- Is I/O handling described and justified?
- **Are scarce resources and their management described?**
  - Threads, network bandwidth, battery power, etc.
- **Are security requirements described?**
- Does each module have a space and speed budget?
- **Is a scalability strategy described?**

Architecture Specifics (3)

- Is interoperability addressed?
- Is the internationalization strategy described?
- Is there a consistent error-handling strategy?
- Is there a fault-tolerance strategy?
- Has it been verified that all parts are technically feasible?
- **Are buy-vs-build decisions described?**
- Is there a strategy for incorporating external code?
- **Does the architecture accommodate likely change?**

General Architecture Quality

- Are all major decisions justified?
- **Are the developers who will implement the system comfortable with the architecture?**



http://www.flickr.com/photos/joone/3050331298

"Code Complete" by Steve McConnell, 2nd edition, Microsoft Press, 2004

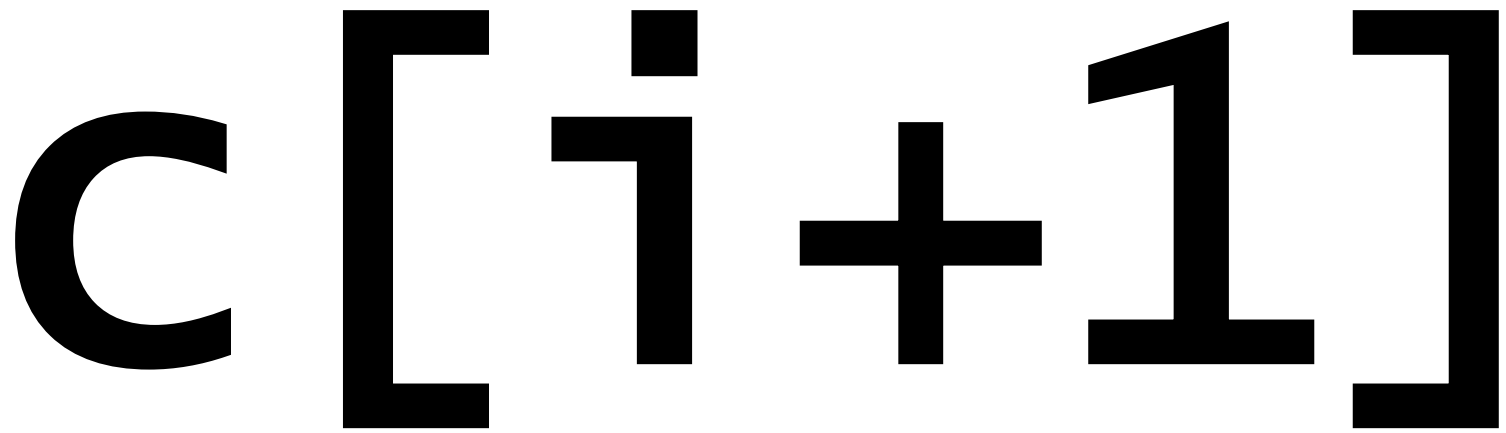
EXAMPLE CHECKLIST FOR DESIGN

Design Practices

- **Have you iterated on the design or just used the first attempt?**
- Have you designed several system decompositions?
- **Have you designed top-down and bottom-up?**
  - Decompose: Iteratively refine until coding obvious
  - Compose: Iteratively group responsibilities into units
- **Have you resolved big risks via small prototypes?**
- Have others reviewed your design?
- Does design make coding obvious?

Design Goals

- **Have you captured the design in writing?**
- Does the design address issues in the architecture?
- Is the design broken into layers?
- Are you happy with the system decomposition?
- Is communication between components minimized?
- Are components reusable?
- Will it be easy to maintain the program?
- Is the design minimal?
- **Is the design mainstream and does it avoid obscure design ideas?**
- **Does the design avoid complexity?**



"Applied Software Project Management" by Andrew Stellman & Jennifer Greene, O'Reilly, 2006

EXAMPLE CHECKLIST FOR CODE

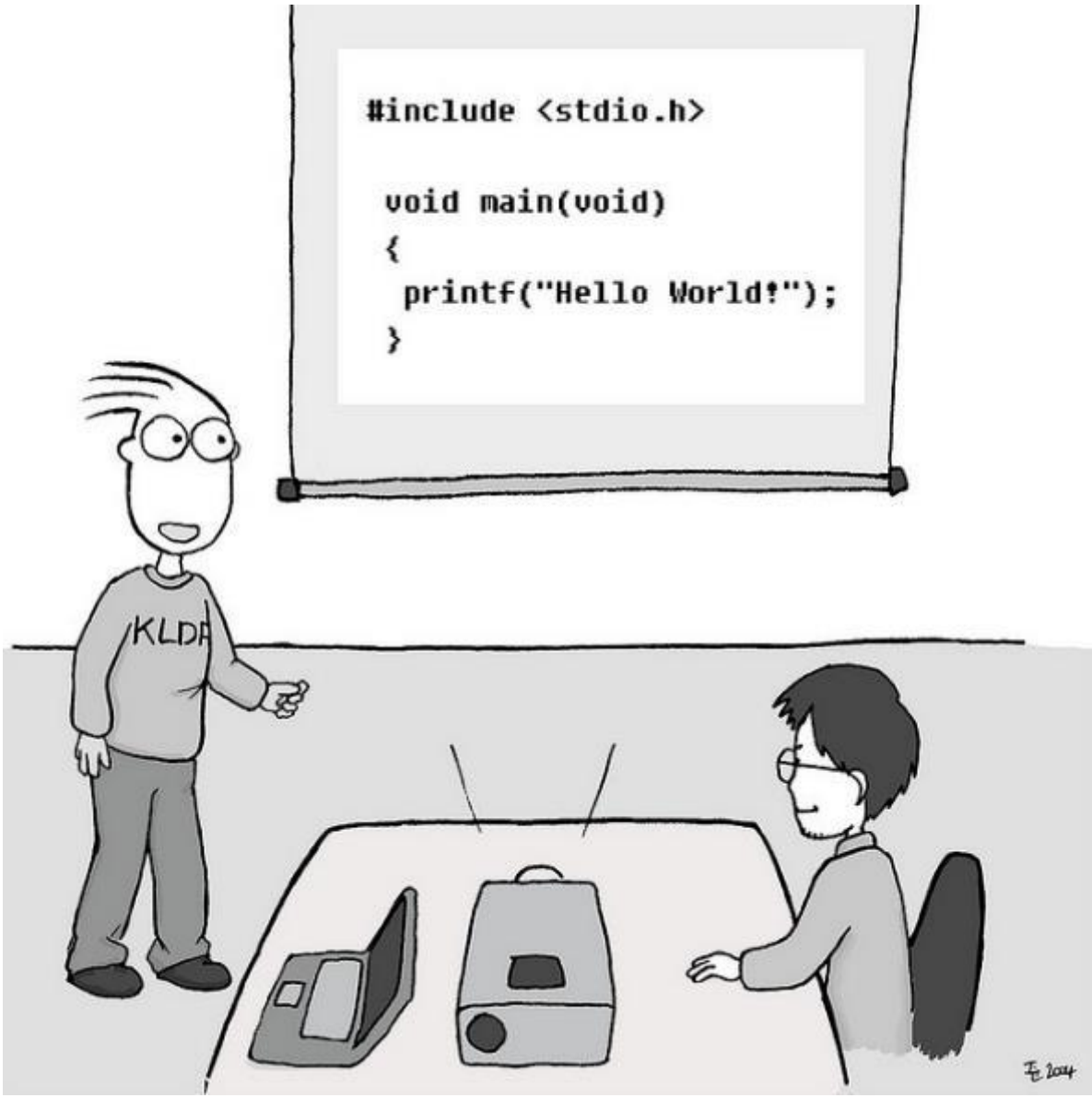
Example Code Checklist (1)

- Clarity
  - **Is the code easy to understand?**
  - Is the code obfuscated unnecessarily?
  - Can the code be changed to make it clearer?
- Maintainability
  - Can others maintain this code?
  - Is the code well commented and documented?
- Accuracy
  - Does the code do what it should do?
- Reliability and Robustness
  - Will the code handle abnormal input gracefully?

Example Code Checklist (2)

- Security
  - Is the code vulnerable to security violations?
- Scalability
  - Where are the bottlenecks of the implementation?
- Reusability
  - Can another application reuse this code?
- Efficiency
  - Does the code use resources efficiently?

WALK-THROUGH



http://www.flickr.com/photos/joone/3049457959/

Motivation for Walkthrough

- Communicate & review technical issues with possibly large group of people
- Relatively informal
  - Reviewers do not prepare individually (different from inspection and review)
- Find faults, not fix them

Roles

- Project team
  - Present document (e.g., code)
  - **Often find bugs in own document while presenting it**
  - Explain how document details fit into project
  - Lead discussion

- Review team
  - Ask questions
  - Make comments

DELIVERABLES

Deliverables of Review

- Potential problems found in
  - Project plan
  - Requirements (if available)
  - Architecture (if available)
  - Design (if available)
  - Code
  - Test cases (if available)

- **Send the list of your findings to the team that owns the project**
  - cc TA