

(Mostly) Painless Code Reviews

Presenting:

Frederic Boulanger

President, Macadamian Technologies, Inc.

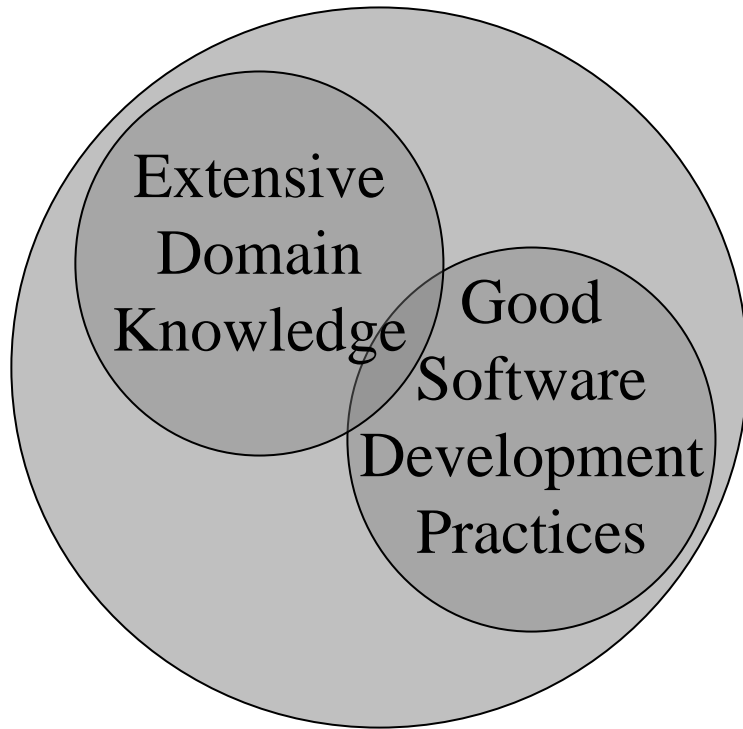
Overview

- The ROI of Code Reviews
- If Fagan-style review meetings are so good for you then...
- A less painful method of code review
- Single Committer 101
- Streamlining Peer Reviews with tools

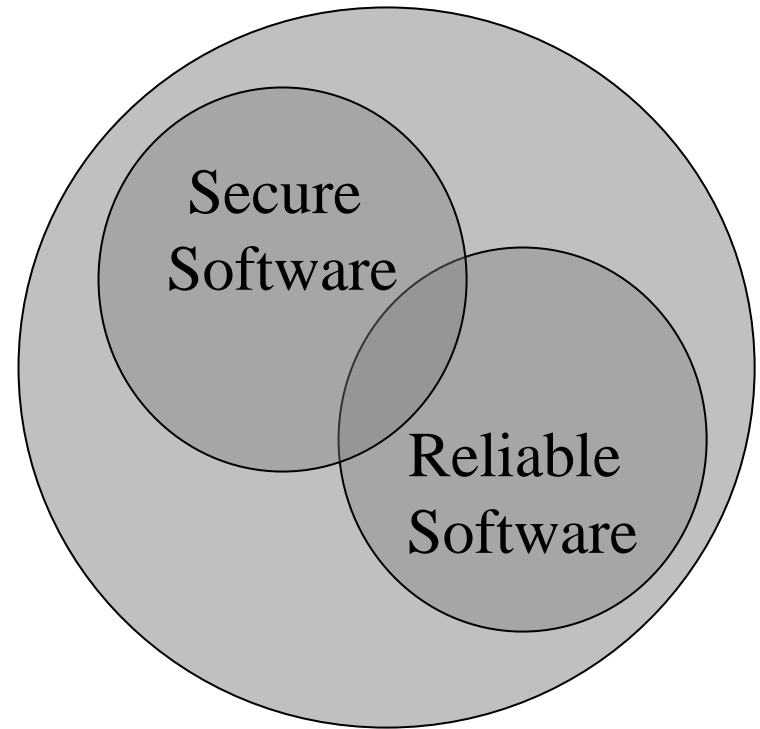
ROI of Code Reviews

- Catch more defects earlier in the cycle
- Cost of fixing bugs later in the cycle
 - Non-incremental cost of discovering a defect at each phase (inspection > milestone > field)
- The Bus Number
- Security

Quality



Quality Product



Quality Software

Code Review Meetings (eat your Broccoli)

- How many companies have a policy of group code reviews?
- How many don't know why?
- If code review meetings are so good for us,
 - Why are so few of us doing them?
 - Why do they fall off in crunch?

Marathon code reviews

- Schedule constraints usually mean marathon sessions
- Marathon review sessions mean:
 - Defects are caught later
 - Some are missed due to fatigue and boredom
 - Redundant work
 - Not everyone is speaking freely, while some are speaking too freely

Less painful reviews

- Peer reviews:
 - Catch more bugs earlier in the cycle
 - Help junior developers learn from their more experienced peers
 - Help developers avoid common mistakes
 - Communication—everyone is aware of everyone else's status
 - Produce code that is easier to maintain

Tips - implementing reviews

- Teams are uncomfortable with change - this might take a while
- Have the team create the review list
 - That way, they are documenting the unofficial standards they are already using, not new standards down from on high
- Code review works best when integrated into the cycle as a step that can't be skipped (more later)

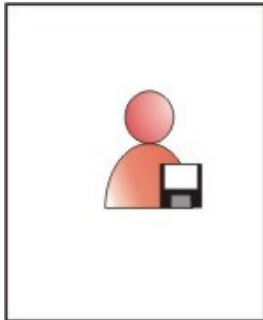
Best practices

- Don't always use the same reviewer
- Find a champion
 - Every team has an influential developer you need to get onside
- Proof = pudding
 - Team will need to see that it actually improves code quality
- Review small chunks
- Review only clean code

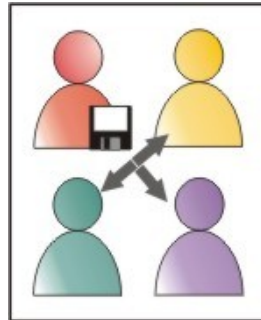
Single committer 101

- Confession:
 - Not entirely our idea
- Adapted from open-source
 - Our team was heavily involved in an open-source project
 - Level of organization and high code quality was a shock
 - Adapted for use in commercial/IT dev

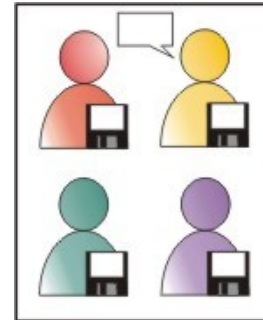
How it works



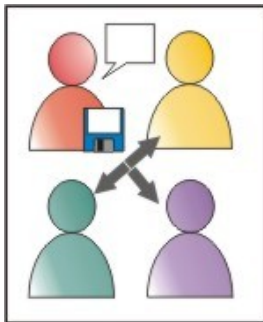
Developer writes code



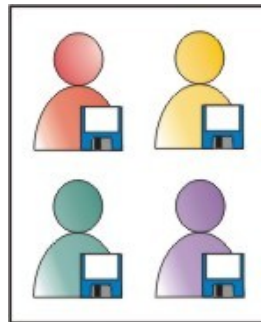
Code (diff) sent to team on mailing list



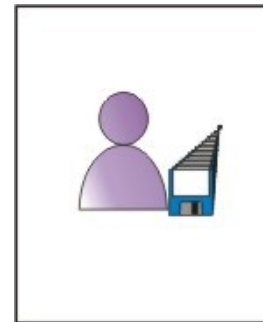
Team comments on quality, defects



Developer resolves issues and sends code back to list



Team members comment



If no issues, committer adds code to SCCS

Our ingredients

- Egoless programming
- Feedback loop
- Teach someone how to fish ...
- Keenness of the individuals

Comparison of time spent

<ul style="list-style-type: none"> •Preparation for meeting •Inspection meeting 	<ul style="list-style-type: none"> •1hr x 3 reviewers •1hr x 4 attendees 	<ul style="list-style-type: none"> •3.00 •4.00 <u>7.00p/h</u>
<ul style="list-style-type: none"> •Review of code •Reviewer-Developer Com •Review of fixes 	<ul style="list-style-type: none"> •1hr x 1 reviewer •.25hr x 2 •.25hr x 1 reviewer 	<ul style="list-style-type: none"> •1.00 •.50 •.25 <u>1.75p/h</u>
<ul style="list-style-type: none"> •Review of code •Communication •Review of fixes 	<ul style="list-style-type: none"> •1hr x 3 reviewers •.25hr x 2 people x 3 •.25hr x 3 	<ul style="list-style-type: none"> •3.00 •1.50 •.75 <u>5.25p/h</u>

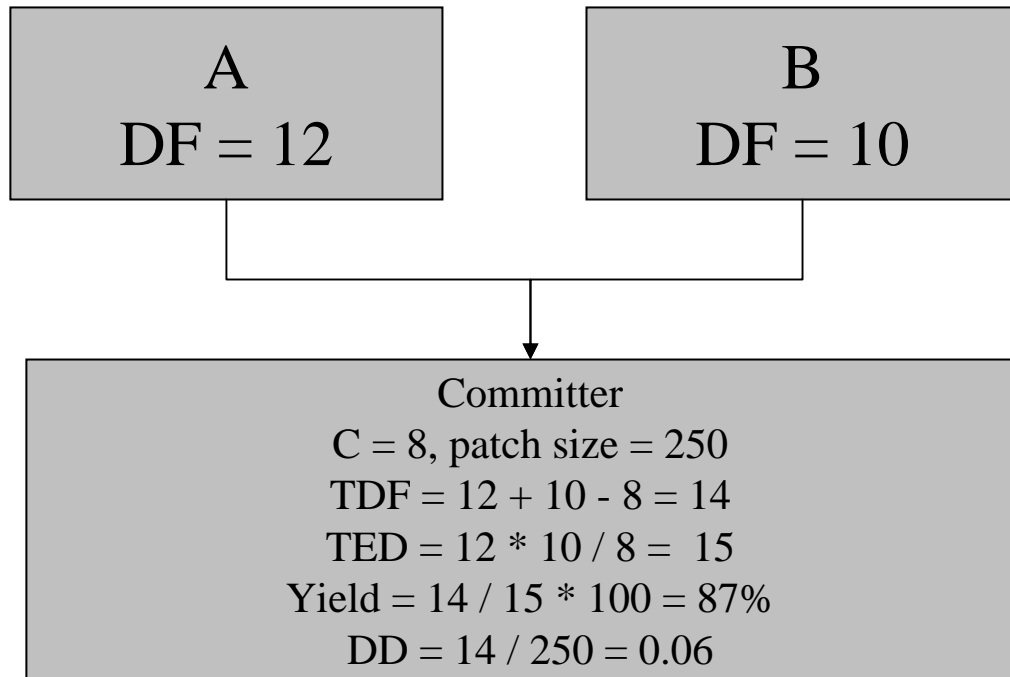
Benefits of Single Committer

- Scales from small teams to large distributed projects
- Everyone in the loop
 - Mailing lists/ Discussion groups
 - Increase your bus number
- Everything in source tree always reviewed
 - Source code control \neq backup system

The next step, metrics

- **Total Defects Found = $A + B - C$** , where A and B are the number found by reviewer A and B respectively and C is the number found by both A and B.
- **Total Estimated Defects = AB/C**
- **Yield = Total Defects Found / Total Estimated Defects * 100%**
- **Defect Density = Total Defects Found / Size**

Example



- DF = Defect Found
- TDF = Total Defect Found
- TED = Total Estimated Defect
- DD = Defect Density

Re-review?

- The committer compiling on data point from both review sheet, call for a second review based on:
 - Yield too low
 - Majors found / Total found too low
 - Unusual defect distribution
 - High defect density
- Project dependant, but must be known

Using tools

- There are a variety of tools
 - See reference
- Tool
 - imposes some process
 - Is the process good for your organization
 - Standardize and automate data gathering
 - Less loss data an data entry
 - Corporate standard

How about Open-Source?

- Debatable, but...
- Definitely closed source != greater quality
- Theory is that more eyes on a piece of code, the more secure/quality
- We have to learn from OSS, but no blank check to the code/product

Security: Things You know

- Our people
 - Genuinely want to write good code
 - Functional Specification compliance != secure system
- Result in vulnerabilities
 - Architecture/Design
 - Implementation
 - Operations

Things you know they know

- They will
 - Strike at will
 - Play dirty
 - Choose weakest point
- Outsiders will try to compromise your application
 - Input/environment
 - Design axioms/metaphor
 - Prediction

What are you looking for in a security review?

- it will depend on your technology choices and the systems being developed
 - Managed vs. non-managed code
 - Socket programming vs. web apps
- Onus is on you to identify known weaknesses of your technology
- Train the *whole* team on weaknesses

Developing a security check list

- Part of your overall code review checklist
- Checklist helps organize reviews (rather than leaving it up to experience/whim of reviewer)
- Let the team create the checklist
 - More likely to buy into the process
 - Education

Checklist primer

- Hardcoded stuff!
- Failure
- Buffer overruns (string, integer)
- Application Privileges
- Resource Access Control List
- Encryption

Checklist cont'

- Important Data
 - Do you need to store it all?
 - It's so hard to keep a secret
- All Input
 - User, environment, internal -> never trust it
 - It must be validated before consumption
 - Even output!

Checklist cont'

- Database input
- Web Specific
- Internationalization
- Socket
- RPC, ActiveX, WebServices, DCOM
- Minimize attack surface

The Security Review

- Small chunks leads to better result
- A few questions to have in mind
 - Elevated privileges?
 - Listening to network interface?
 - C/C++
 - Sensitive data
 - Reusable module
 - Ranking in the threat assessment
- Pay extra attention to pointers
- Data is not to be trusted
- Review those old bugs
 - When you have the data handy

Resources

- Code Review Checklist:
 - <http://www.macadamian.com/codereview.htm>
- MSDN security checklist:
 - <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/THCMCh21.asp>
- Single committer development primer
 - <http://www.macadamian.com/column/singleCommitter.html>
- Download the CodeReview Add-In for VS .NET
 - <http://www.macadamian.com/products/codereview/download.html>
- Smart Bear Software
 - <http://www.codehistorian.com/>

Reference for Defect estimation

- Let the number of defects found by one reviewer be the tagged population (A).
- Assume an even likelihood of finding all defects (even distribution)
- Count the number of defects found by the second reviewer (B)
- Count the number of defects found by the second reviewer that were also found by the first (C the common defects).
- Calculate the proportion of common defects in the second reviewers defects ($T=C/B$).
- We assume that T is representative of the proportion of common defects in the total estimated number of defects (TED), so
 - $T * TED=A$, or for our purposes $TED = A/T = (A*B)/C$.

Q&A

- Questions?