Collaborative Code Construction: Code Reviews and Pair Programming

CPSC 315 – Programming Studio

adapted from John Keyser's 315 slides

Collaborative Construction

- Working on code development in close cooperation with others
- Idea
  - Developers don't notice their own errors very easily
  - Others won't have the same blind spots
  - Thus, errors are caught more easily by other people
- Takes place during the *construction* process

Benefits of Collaborative Construction

- Can be much more effective at finding errors than testing alone
  - 35% errors found through testing through low-volume Beta level
  - 55-60% errors found by design/code inspection
- Finds errors earlier in the process
  - Reduces time and cost of fixing them
- Provides mentoring opportunity
  - Junior programmers learn from more senior programmers

More Benefits

- Creates collaborative ownership
  - No single "owner" of code
  - People can leave team more easily, since others have seen code
  - Wider pool of people to draw from when fixing later errors in code

Some Types of Collaborative Construction

- Formal inspections
- Walkthroughs
- Code reading
- Pair programming

Code Reviews

- Method shown to be extremely effective in finding errors
  - ratio of time spent in review vs. later testing and error correction ranges from 1:20 to 1:100
  - Reduced defect correction from 40% of budget to 20%
  - Maintenance costs of inspected code is 10% of non-inspected code
  - Changes done with review: 95% correct vs. 20% without
  - Reviews cut errors by anywhere from 20% to 80%
  - Several others (examples from *Code Complete*)

Reviews vs. Testing

- Finds different types of problems than testing
  - Unclear error messages
  - Bad commenting
  - Hard-coded variable names
  - Repeated code patterns
- Only high-volume beta testing (and prototyping) find more errors than formal inspections
- Inspections typically take 10-15% of budget, but usually reduce overall project cost

Formal Inspection Characteristics

- Focus on *detection*, not correction
- Reviewers prepare ahead of time and arrive with a list of what they've discovered
  - Don't meet unless everyone is prepared
- Distinct roles assigned to participants
  - Stick to these roles during review
- Data is collected and fed into future reviews
  - Checklists focus reviewers' attention on common past problems

Roles during Inspection

- Moderator
- Author
- Reviewer(s)
- Scribe
- Management

- 3 people min
- ~6 people max

Roles during Inspection

- Moderator
- Author
- Reviewer(s)
- Scribe
- Management

- 3 people min
- ~6 people max

- Keeps review moving
  - Not too fast or slow
- Technically competent
- Handles all meeting details
  - distributing design/code
  - distributing checklist
  - Setting up room
  - Report and followup

Roles during Inspection

- Moderator
- Author
- Reviewer(s)
- Scribe
- Management

- 3 people min
- ~6 people max

- Plays minor role
  - Design/Code should speak for itself
- Should explain parts that aren't clear
  - But this alone can be a problem
  - Explain why things that seem to be errors aren't
- Might present overview

Roles during Inspection

- Moderator
- Author
- Reviewer(s)
- Scribe
- Management

- 3 people min
- ~6 people max

- Interest in code but not an author
- Find errors during preparation
- Find more errors during meeting

Roles during Inspection

- Moderator
- Author
- Reviewer(s)
- Scribe
- Management

- 3 people min
- ~6 people max

- Records errors found and action assigned or planned
- Should not be moderator or author

Roles during Inspection

- Moderator
- Author
- Reviewer(s)
- Scribe
- Management

- 3 people min
- ~6 people max

- Usually should not be involved
  - Changes from technical to political meeting
- Might need to see results of meeting

Stages of Inspection – Planning

- Author gives code/design to moderator
- Moderator then:
  - chooses reviewers
  - ensures code is appropriate for review
    - e.g. line numbers printed
  - distributes code and checklist
  - sets meeting time

Stages of Inspection – Overview

- If reviewers aren't familiar with code at all, can have overview
- Author gives a brief description of technical requirements for code
- Separate from review meeting
- Can have negative consequences
  - Groupthink
  - Minimize points that should be more important

Stages of Inspection – Preparation

- Reviewers work alone to scrutinize for errors
  - Checklist can guide examination
- Depending on code, review rate varies
  - 125 to 500 lines per hour
- Reviewers can have varied “roles”
  - be assigned “perspective”
    - e.g. evaluate from user’s view, or from designer’s view
  - evaluate different scenarios
    - e.g. describe what code does, or whether requirement is met
  - read code/design in certain order/way
    - e.g. top-down, or bottom-up

Stages of Inspection – Inspection Meeting

- A reviewer chosen to paraphrase design or read code
  - Explain all logic choices in program
- Moderator keeps things moving/focused
- Scribe records errors when found
  - Record type and severity
- Don’t discuss solutions!
  - Only focus is on *identifying* problems
  - Sometimes don’t even discuss if it actually is an error – if it seems like one, it is one
- No more than 1 per day, about a 2 hour limit

Stages of Inspection – “Third Hour” meeting

- Depending on interest/stake of reviewers, possibly hold a *separate* followup meeting
  - Immediately after inspection meeting
- Focus here is to discuss possible solutions

Stages of Inspection – Inspection Report

- Moderator produces report shortly after meeting
  - List of defects, types, and severity
- Use this report to update checklist to be used in future inspections
  - List main types of errors commonly found
  - No more than 1 page total length
- Collect data on time spent and number of errors
  - Helps evaluate how well things work, justify effort

Stages of Inspection – Rework

- Moderator assigns defects to someone to repair
  - Usually the author

Stages of Inspection – Follow-Up

- Moderator verifies that work assigned was carried out.
- Depending on number and severity of errors, could take different forms:
  - Just check with author that they were fixed
  - Have reviewers check over the fixes
  - Start cycle over again

Adjusting Inspections Over Time

- Organizations will have characteristics of code unique to them
  - Density of code determines how fast reviewers and inspection meeting can go (application tends to be faster than system code/design)
  - Checklists highlight common problems
- Measure effect of any changes
  - Evaluate whether they actually improved process

Inspections and Human Egos

- Point is to improve code
  - Not debate alternative implementations
  - Not discuss who is wrong/right
  - Moderator needs to control discussion
- Author needs to be able to take criticism of code
  - May have things mentioned that aren't "really" errors
  - Don't debate and defend work during review
- Reviewers need to realize the code is not "theirs"
  - Up to author (or someone else) to determine fix

Walkthroughs

- Alternative to formal code inspection
- Vague term, many interpretations
  - Less formal than inspections, though
- Usually hosted and moderated by author
- Chance for senior and junior programmers to mix
- Like inspection:
  - Preparation required
  - Focus on technical issues
  - Goal is detection, not correction
  - No management

Walkthrough Evaluation

- In best cases, can match formal code inspections in quality
- In worst cases, can lower productivity, eating more time than saved
- Can work well for large groups
- Can work well when bringing in “outsiders”

Code Reading

- Alternative to inspections and walkthroughs
- Author gives out code to two or more reviewers
- They read independently
- Meeting held for everyone
  - Reviewers present what they’ve found, but don’t do a code walkthrough

Code Reading Evaluation

- Most errors tend to be found in individual review
  - Reduces effort and overhead of managing group dynamics at inspection meeting
  - Maximizes productive effort per person – time not wasted in meetings where others are speaking
- Works well for geographically distributed reviewers

Pair Programming

- Basic idea: One person codes with another looking over the shoulder.
- Person at keyboard writes code
- Second person is *active* participant
  - Watch for errors
  - Think strategically about code
    - What's next?
    - Is code meeting overall goal/design?
    - How to test this code

Successful Pair Programming

- Standardize coding style
- Don't force pairs for easy tasks
- Rotate pairs and work assignments frequently
- Use "good" matches
  - Avoid personality conflicts
  - Avoid *major* differences in speed/experience
- Set up good work environment
- At least one pair member should be experienced

Evaluating Pair Programming

- Seems to achieve quality level similar to formal inspection
- Tends to decrease development time
  - Code written faster, fewer errors
- Tends to be higher quality code
  - Holds up better during crunch time – fewer shortcuts taken that come back to haunt
- All the traditional collaborative benefits