

Code Reviews



• Why review code?

- High-quality program on first try?
- Struggle to compile/test a defect-prone first draft?

From *A Discipline for Software Engineering* by Watts Humphrey

Defect Removal Efficiency

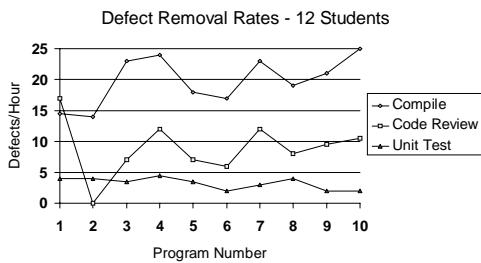
• Unit test

- 2-4 defects/hour
- Rarely more than 50% found

• Code reviews

- 10 defects/hour
- Experienced reviewers - 70% found

PSP Defect Removal



Review Versus Testing

- **After unit test, defect removal much more expensive**
 - 10-40 hours to find/fix each defect
 - Inspections: less than 1 hour/defect
- **Cleaner entry ⇒ cleaner exit**
 - For each development phase

Review Versus Testing

- **Testing**
 - Start with problem, find defect
 - Devise fix, implement, and test
- **Reviews**
 - See defect
 - Devise fix, implement, review



Testing



- **Detect unusual result**
- **Figure out what happened**
- **Find where it was in program**
- **Figure out what defect could cause behavior**
 - Search for unplanned/unexpected

Reviews & Inspections

- You follow your own logic
- When you find defect
 - You know exactly where you are
- You know program
 - What it should do and did not do
 - Better able to devise complete fix

Review Entry Criteria

- Program design
- Program listing
- Code review checklist
- Standards and documents
 - Coding, defect type, logging forms

Review Guidelines

- No more than 200 LOC/hour?
 - Efficiency falls off at higher rates
- Checklists
- When to review
 - Before or after compile?

Checklist Example

- **Code covers all the design**
- **Includes: verify complete**
- **Initialization - data objects/args**
 - At program initiation
 - At start of every loop
 - At function entry



Checklist Example

- **Calls - check formats**
 - Pointers and parameters
 - Use of '&' - "address of" operator
- **Names - spelling and use**
 - Consistent, in declared scope
 - Member reference syntax ('.')

Checklist Example

- **Strings (if C-style)**
 - Identified by pointers
 - Terminated in NUL character
- **Pointers**
 - Initialized NULL
 - Deleted iff dynamically allocated

Checklist Example

- **Output format**
 - Line stepping & spacing are proper
- **Compound statements**
 - “{ }” are proper and matched
- **Logic operators**
 - Verify ==, =, | | , parentheses

Checklist Example

- **Line-by-line check**
 - Instruction syntax
 - Proper punctuation
- **Coding standard followed?**
- **Files**
 - Properly declared, opened, closed

Compile Before Review?

- **Better for some defect types**
 - Compiler highly effective
 - About 90% of syntax/naming defects
- **Review effectiveness varies**
 - Miss 20-50% of syntax defects?
- **Find missed defects in test**
 - Including syntax

Review Before Compile? (1)

- **Compiler misses 9% syntax**
- **Finding defects in review**
 - Saves compile time
 - Makes it more predictable
- **Review more efficient (vs test)**
- **Compiler/unit test leaves 4%**

Review Before Compile? (2)

- **If reviewing compiled code**
 - Few syntax problems left
 - Review less rewarding, less effective
- **If reviewing for syntax errors**
 - Won't save time by compiling first
 - Will save time by reviewing first

Review Before/After?

- **Humphrey says before**
 - Based on PSP student data
 - Hard to motivate review of compiled
- **Keep your own data, decide**
- **Review goal**
 - No defects later found in test!
