Today

- Code Reviews

Homework

- HW8 out
- Quiz next Tues
**Terminology: Metrics**

**LOC:** lines of code (issues?)

**Defect rate:** defects found / hour

**Defect density:** defect found / LOC

**Inspection rate:** LOC / hour
Code Review

A variant of “software inspection” (heavyweight/formal)

• code review typically less formal

How they work (high level):

1. Either meet to decide on review goals or just part of dev process

2. Design, code, documentation, etc. is reviewed
   • Review done by someone other than the developer
   • Either with author or “out of band” (more common)

3. Fixes made, reviewed again, repeat until committed into codebase

Q: What kind of “testing” is a code review?

1. White box

2. Independent verification

3. Sometimes not viewed as a testing task itself

Q: Most companies do code reviews. Why?

1. Find issues (defects/faults) before testing (saves time/money)

2. “Knowledge” share among team

3. Mentor new hires
Example data on inspections & defect rates

Raytheon

1. Reduced rework from 41% to 20% of cost
2. Reduce cost to fix integration errors by 80%

Space shuttle software:

1. $1 if bug found in inspection
2. $13 if bug found in system test
3. $92 if bug found after delivery

IBM

1. 1 hour of inspection saved 20 hours in testing (if caught)
2. 1 hour of inspection saved 82 hours after release

HP

1. System use found 0.21 defects/hour
2. Black box found 0.28 defects/hour
3. White box found 0.32 defects/hour
4. Inspection found 1.06 defects/hour

Various (C. Jones)

1. Design/code inspections remove 50-70% of defects
2. Testing removes 35%
**Problem:** Formal code inspections are expensive ...

Often have many “phases”:

- plan, introduce, read, inspect, rework, verify, complete

And 4 or more participants:

- author, moderator, reviewer, reader, observer

Average inspection, e.g., shown to take **9 hours per 200 LOC**

**Good news:** (Lightweight) code reviews also work ...

- Lightweight techniques also work at finding defects (Cisco)
- Take significantly less time
- Some “best practices” can help

Today, code review largely expected as industry standard
Main approaches to code reviews

Walkthrough (... over-the-shoulder, peer desk check)

- author walks reviewer(s) through code
- reviewer(s) asks questions, makes suggestions

Pass around

- author gives code to reviewer
- reviewer gives feedback later

Ad hoc

- informal feedback from a team member
- Pair programming
- Debated whether really a form of review ... why?
What can be reviewed? ... Just about anything!

- Requirements
- Design
- APIs
- Code
- Test cases (e.g., associated unit tests)
- Code comments and documentation
- User guides, installation instructions

For each, can also review for different things

- defects, clarity/quality, efficiency, security, ...
Some Code-Review “Best Practices”

1. Formalize code review in development process
   - all committed code is reviewed first
   - use tools (e.g., automate basic checks first, for communication)

2. Define checklists and expectations
   - what aspects should/will be reviewed
   - preconditions for review (e.g., test coverage, passing tests, etc)

3. Don’t waste time reviewing hard to understand code
   - Just ask for it to be clarified ...
   - Healthy code is easy to quickly read and understand
   - Allows you to focus on issues with the logic, approach, design, etc.
   - For example (from Google Testing Blog):

     ```python
     def is_okay(n):
         f = False
         for i in range(2, n):
             if n % i == 0:
                 f = True
         return not f
     ```

   - Q: What would you do as a reviewer?
– After a couple of seconds looking at this add comment saying:
– “It’s hard for me to understand this piece of code”
– Or better, “Please use more descriptive names here”

• You might get this back:

```python
def is_prime(n):
    for divisor in range(2, n//2):
        if n % divisor == 0:
            return False
    return True
```

• Q: What would you do now as a reviewer?
  – Note the function actually changed (e.g., 2 to n/2)
  – Now much easier to understand, and see issues
  – e.g., strange behavior with 0 and 1
  – Might tell author to replace this with a built-in function

4. Verify defects are actually fixed
• always document issues/defects found
• again, encode in process
5. **Have the right mindset / attitude**
   - finding defects is good! ... author & reviewer improved product together
   - correcting bad habits and learning new techniques
   - don’t punish someone if defects found (also beware rewards)

6. **Always do some review ...**
   - even if don’t have time to review everything
   - “Ego Effect”: you’ll write better code if you know it will be reviewed
Exercise 1: Checklists ...

- Braninstorm an initial checklist for your project.
- What should reviewers look for when reviewing code?
- How will this change how you write code (if at all)?

Some Examples:

- Style guide conformance
- Identifier naming
- Use of libraries (use existing library unless good reason not to!)
- Interfaces, APIs
- Readibility (easy to understand!)
- Reusability (and self contained)
- Dependencies clearly identified (and minimized)
- Security
- Design issues
- Comments, documentation
- Unit tests, code coverage, testability

Exercises 2–5: More on code review and version control