CPSC 491

Lecture 14:
Testing Intro + Code Reviews

Notes:
- HW 9 posted (not “assigned” until Tues)
- Exam 1 on Tuesday

Exam 1

Basics:
- 5 Multipart Questions
- Worth 10% of final grade
- Closed book, notes, etc.

Topics:
- Class lectures, quizzes, and readings
- Projects and process models
- Scrum techniques (process, backlogs, meetings, ...)
- Requirements (types, properties)
- Estimation and planning
Terminology: Verification vs Validation

**Verification**: Are you building it correctly (to spec)?

**Validation**: Are you building the correct thing (user)?

**Independent**: Testing by a “third party”

---

Terminology: Back Box vs White Box Testing

**White Box**: Test with knowledge of the code
- Tests based on how system design/coded (e.g., unit tests)

**Black Box**: Test without knowledge of the code
- Tests based on input/output (e.g., system tests)
- How user interacts with or “sees” the system

**Gray Box**: Mix of both
- Testing connections to subsystems
- Performance testing another example
**Terminology: Manifestations**

*Mistake*: incorrect human action (select bad loop condition)

*Fault*: the manifestation of the mistake (coded condition)

*Failure*: the result of the fault (wrong value calculated)

*Error*: expected vs unexpected (off by 20%)

*Bug*: usually same as fault … sometimes all of these!

*Defect*: usually same as fault

**Terminology: Metrics**

*LOC*: lines of code (issues?)

*Defect rate*: defects found / hour

*Defect density*: defect found / LOC

*Inspection rate*: LOC / hour
Before diving into testing ...

**Code reviews** (as a kind of testing)

- A variant of “software inspection” (heavyweight/formal)
- review typically less formal

How they work (high level):

- Optional meeting to decide on review goals
- Design, code, documentation, etc. is reviewed
- Review done by someone other than the developer
- Either with author or “out of band”

Questions ...

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

- White box
- Independent verification
- (Not often considered a type of testing though)

Q: Most companies do code reviews. Why?

- Find defects (bugs) before testing (saves time/money)
- “Knowledge” share among team
- Mentor new hires
Lost of data on inspections & defect rates

Raytheon
- Reduced rework from 41% to 20% of cost
- Reduce cost to fix integration errors by 80%

Space shuttle software:
- $1 if bug found in inspection
- $13 if bug found in system test
- $92 if bug found after delivery

IBM
- 1 hour of inspection saved 20 hours in testing
- 1 hour of inspection saved 82 hours after release

HP
- System use found 0.21 defects/hour
- Black box found 0.28 defects/hour
- White box found 0.32 defects/hour
- Inspection found 1.06 defects/hour

Various (C. Jones)
- Design/code inspections remove 50-70% of defects
- Testing removes 35%
But formal inspections are expensive

Often have 7 phases:
- plan, introduce, read, inspect, rework, verify, complete

And 4 or more participants:
- author, moderator, reviewer, reader, observer

Average inspection shown to take *9 hours per 200 LOC*

---

Good news: Lightweight reviews also work!

Lightweight techniques also work at finding defects (Cisco)

Take 20% (or less) time

Some “best practices” can help
What can be reviewed ...

Just about anything

- Requirements
- Design
- APIs
- Code
- Test cases
- Code comments and documentation
- User guides, installation instructions
- ...

For each, can also review for different things
- defects, clarity/quality, efficiency, security, ...

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?
**Exercise: Code walkthrough ...**

Select roles: *author, reviewer, and observer*

- observer just for the exercise (not usually needed)
1. Author walks reviewer through code
2. Reviewer looks for issues/changes (write down)
3. Agree on changes needed (write down)
4. Observer analyzes process & improvements

*What to review in the code*: documentation (comments), coding style, code design, potential defects

---

**Some Best Practices**

1. Review fewer than 200-400 LOC/Review (6-12 screens)
   - ability to find defects diminishes beyond this
   - for code (other artifacts may be different)

2. Aim for an *inspection rate* of 300-500 LOC/hour
   - how long should a review be? … less than an hour

3. Authors should go over code before review
   - like proof-reading
Some Best Practices

4. **Checklists** improve results for authors & reviewers
   - list of what things should be reviewed
   - class defs, imports, function signatures, pre/post conditions, loop invariants, api calls, ...
   - useful as a developer as well

5. Verify defects are **actually fixed**
   - always document defects/issues found
   - reviewer checks back to ensure fixes made
   - tools can help with this

Exercise

Q: Create an initial **checklist** for your code reviews
   - What should reviewers be looking for?
   - How will this change how you write code (if at all)?

Q: How will your group keep track of issues?
   - How to capture the issues
   - And whether they have been resolved
Some Best Practices

6. Have the right attitude
   - finding defects means author & reviewer worked as a team to improve the product
   - correcting bad habits and learning new techniques
   - don’t punish someone if many defects found

7. Do some review, even if don’t have time to review all
   - “Ego Effect”: devs will write better code if they know it will (even partially) be reviewed