CS314 Software Engineering
Peer Reviews

Dave Matthews
Peer Reviews

• Casual inspections
  – Over the shoulder
  – Tool assisted (GitHub)
  – Email pass around
  – Pair Programming
• Formal inspections
  – Roles
  – Checklists
  – Preparations
  – Meetings
  – Logs
• Automated inspections

Peer Reviews—What?

• Requirements
• Architecture
• Design
• Code
• Documentation
• Tests
Peer Review – When?

- The earlier problems are found, the less expensive they are to fix.
- Peer reviews before extensive testing allows you to quickly get rid of many defects, rather than removing them one at a time.

Peer Reviews – Why?

- Peer reviews can find multiple defects at a time.
- Peer reviews can be done on incomplete systems.
- Peer reviews can consider broader quality aspects, such as non-functional requirements.
- Peer reviews can find defects difficult to detect by testing.
- Peer reviews can save you a lot of rework / technical debt.
Formal Inspections

- Process to systematically examine project artifacts (documents or code) to identify as many problems as possible.
  - Careful line by line review.
  - Use a checklist of common programming errors, local standards, and practices.
  - Multiple people with different points of view.
  - Review independently to find problems, meet to review and report findings.

Formal Inspection Principles

- Advance preparation – identify problems before meeting
- Only inspect things that are ready
- Re-inspect when changes are made
- No blame – work together, next time it will be yours
- Keep the discussions open – don’t involve managers
- Avoid discussing how to fix the problems
- Avoid discussing style issues
- Do not rush (200 lines code / hour, 10 pages of text)
- Avoid extended sessions (<2 hours at a time)
- Keep inspection logs to track quality of processes
Formal Inspection Example Roles

- **End user**
  - User experience, useful messages, input validation, responsiveness, ...
- **Maintainer**
  - Descriptive names, consistent exception handling, small classes/methods, good comments, ...
- **Tester**
  - Documentation, well defined interfaces, ...
- **Moderator**
  - Makes the process efficient and effective at finding problems, ensures only logging occurs – no fixing.

### Code Review Checklist - Java

1. **Specifications / Design**
   - Is the functionality described in the specification fully implemented by the code?
   - Is there any excess functionality not described in the specifications?

2. **Initialization and Declarations**
   - Are all local and global variables initialized before use?
   - Are variables and class members of the correct type and appropriate modes?
   - Are variables declared in the proper scope?
   - Is a constructor called when a new object is desired?
   - Are all used import statements included?
   - Variable names are spelled correctly and consistently?
   - Make sure that primitive data types are not set to null or empty
   - Is a static keyword used correctly?

3. **Method Calls**
   - Are parameters presented in the correct value?
   - Are parameters of the correct type for the method being called?
   - In the output method being called, does it use a different method with a similar name?
   - Are all arguments passed to a method logical?
   - When calling a method that has a return value, do we test the return value properly?

4. **Arrays**
   - Are there any off-by-one errors in array indexing?
   - Can array indexes even go negative?
   - Is a constructor called when a new array is desired?
   - Are array declaration synchronized correctly?
   - Are the row and column being indexed in the right order for a 2D array?

5. **Objects Comparison**
   - Are all objects (including strings) compared with "equals" and not "=="?

6. **Output Format**
   - Are there any spelling or grammatical errors in displayed output?
   - Is the output formatted correctly in terms of line stepping and spacing?

7. **Comparison, Comparisons and Assignments**
   - Do all comments end with a semicolon?
   - Check order of comparisons/evaluation, operator precedence and parenthesisizing
   - Can the denominator of a division ever be zero?
   - In integer arithmetic, especially division, are never used inappropriately, causing unexpected casts/rounding?
   - Check such conditions to test the proper exception handling operators are used.
   - If the test is an error-check, can the error condition actually be legitimate in some cases?
   - Does the code rely on any implicit type conversions?

8. **Exceptions**
   - Are all errors exceptions caught?
   - Is the appropriate action taken for each catch block?

9. **Flow of Control**
   - Is a switch statement every case terminated by break or return?
   - Do all switch statements have a default branch?
   - Check that each statement has a “return” or “throw”
   - Are all loops correctly formed, with the appropriate initialization, increment and termination guards?
   - Are all possible paths through a loop properly covered by if statements?
   - Do logical expressions evaluate in the correct true or false value?
   - Do boolean functions return the correct value?

10. **Files**
    - Are all files properly declared and opened?
    - Are all files closed properly, even in the case of an error?
    - Are I/O conditions initialized and handled correctly?
    - Are all file exceptions caught?
Inspection Assignment/Quiz on Wed

- **Monday (planning)**
  - Choose some code (200+ lines) in your repo to review
  - Start team/inspection1.md in your repo - checklist, inspection roles
- **Tuesday (preparation)**
  - Perform your individual reviews of the code using a checklist
  - Submit your preparation on Canvas (individual assignment).
- **Wednesday (in class - attendance required)**
  - Review the problems found, identify defects
  - Add issues to GitHub for all defects found
  - Complete the log in team/inspection1.md in repo