Code Review

- Introduction
- How to Conduct Code Review
- Practical Tips
- Tool Support
- Summary

What is it?

- A systematic examination of source code to ensure sufficient code quality
  - Correctness: Try to detect faults that may exist in the code
  - Maintainability: Try to make the code easier to understand and maintain

Why?

- Help to find and fix bugs early
  - Two brains are better than one brain!
- Help to improve code structure
- Enforce coding standards
- Spread knowledge among team members
  - Good training opportunities for new hires
  - What if the original author leaves?
- Developers know their code will be reviewed, so they will work harder.

When and how often

- Not too soon, not too late
- Typically after unit testing has been done, and after basic features have been tested
- Weekly, or after each major feature

UTA

Philosophy

- A forum to discuss and learn from everyone
- Not an opportunity to criticize people
- Not to demonstrate who is a better programmer

UTA

Potential Misuses

- A waste of time and effort, if not performed effectively
- Harsh reviews may destroy a less experienced developer
- May create social problems if ego and/or politics are involved

Lightweight vs Formal Review

- Lightweight review: over-the-shoulder, email pass-around, and tool-assisted review
- Formal review: a well-defined process, physical meetings, prepared participants, documented results

Fagan Inspection (1)

- Planning
  - Preparation of materials
  - Arranging of participants
  - Arranging of meeting place
- Overview
  - Group education of participants on the materials
  - Assignment of roles
- Preparation
  - The participants review the item to be inspected and supporting materials
  - The participants prepare their roles

UTA

Fagan Inspection (2)

- Inspection meeting
  - Actual finding of defects and opportunities for refactoring
- Rework
  - Resolve the comments made the review
- Follow-up
  - Verification that all the comments are addressed

UTA

A Simplified Process

- Preparation
  - Establish the review group (the programmer, two reviewers, a recorder, and a leader)
  - Make the materials available
  - Come prepared
- Review
  - The leader opens with a short discussion (goals and rules)
  - The programmer explains the code (what it is supposed to accomplish, what requirements it contributes to, and what documentation it affects)
  - Each participants raises questions, comments, and suggests
  - The programmer responds (explain the logic, and problems, and choices
- Follow up

UTA

Who

- Leader: technical authority, experienced, supportive and warm personality
- Recorder: keep a written record
- Reader: summarize the code segments, could be the author
- In general, participants should have a balanced mix
  - An architect, a peer of the contributor, someone in the middle, new hires
- People should not be there: non-technical people, system testers, and managers

UTA

What to look for (1)

- Logic errors: programming mistakes, incorrect assumptions, misunderstanding of requirements
- Adherence to coding standards
- Use of common code modules
- Robustness – adequate error handling

UTA

What to look for (2)

- Readability: meaningful names, easy-to-understand code structure
- Bad smells: opportunities for refactoring
- Tests: make sure unit tests are provided
- Comments: adequate comments must be provided, especially for logic that is more involved

UTA

Tips - Statistics

- Size: 200 ~ 400 lines of uncommented code
- Review time <= 1 hour
- Inspection rate <= 300 LOC/hour
- Expected defect rates around 15 per hour
- # of reviewers: 3 to 7

UTA

Tips - Management

- Code reviews cannot be optional
- But it can be selective
  - Critical and/or complex code, code that is written by less experienced people, e.g., new hires
- Require separate code reviews for different aspects
  - Security, memory management, and performance

UTA

Tips - Reviewers

- Critique the code, not the person
- Ask questions rather than make statements
- Point out good things, not only weaknesses
- Remember that there is often more than one way to approach a solution
- Respect, be constructive

Tips - Developers

- Remember that the code isn't you
- Try to maintain coding standards
- Create a checklist of the things that the code reviews tend to focus
- Respect, and be receptive

Dont

- Should not use it for performance measurement
- Avoid emotions, personal attacks, and defensiveness
- Avoid ego and politics
- No code changes after the review copy is distributed

UTA

The Seven Deadly Sins

- Participants don't understand the review process
- Reviewers critique the producer, not the product
- Reviews are not planned, and reviewers are not prepared
- Review meetings drift into problem-solving.
- The wrong people participate.
- Reviewers focus on style, not substance.

UTA

Tool Support

- Tools that try to automate the workflow
  - Rietveld (Google), Review Board (reviewboard.org), Code Striker (Sourceforge), Java Code Reviewer (Sourceforge), Code Collaborator (SmartBear), and many others
- Tools that try to automate the actual inspection
  - Checkstyle: check compliance with coding standards
  - Splint: check C programs for security vulnerabilities
  - BLAST: a software model checker for C programs
  - And many others

Summary

- One of the most effective ways to improve code quality
- It is the code that is being reviewed, not the developer.
- A good opportunity for knowledge sharing and team building.
- Code review should be an integral part of the development process.