
courses.cs.washington.edu/courses/cse403/16wi/

Code Reviews

Emina Torlak
enerima@cs.washington.edu
Outline

• What is code review?
• Kinds of code review
• Example
code reviews: what and why
Assuring software quality is hard ...
Assuring software quality is hard ...

• What are we assuring?
  • Building the right system
  • Building the system right
    • correct, secure, reliable, available
    • usable, cost effective, maintainable
Assuring software quality is hard ...

• What are we assuring?
  • Building the right system
  • Building the system right
    • correct, secure, reliable, available
    • usable, cost effective, maintainable

• Why are we assuring it?
  • Business, legal, ethical, social reasons
Assuring software quality is hard ...

- What are we assuring?
  - Building the right system
  - Building the system right
    - correct, secure, reliable, available
    - usable, cost effective, maintainable

- Why are we assuring it?
  - Business, legal, ethical, social reasons

- How do we assure it?
Assuring software quality is hard ...

• What are we assuring?
  • Building the right system
  • Building the system right
    • correct, secure, reliable, available
    • usable, cost effective, maintainable

• Why are we assuring it?
  • Business, legal, ethical, social reasons

• How do we assure it?

• How do we know we have assured it?
Challenges of building large systems

• How to ensure maintainable, DRY, readable, bug-free code?
• Average defect detection rate for various testing approaches
  • Unit testing: 25%
  • Function testing: 35%
  • Integration testing: 45%
• How can we do better?
Code reviews
Code reviews

• **Code review**: A constructive review of a fellow developer’s code. A required sign-off from another team member before a developer is permitted to check in changes or new code.
Code reviews

- **Code review**: A constructive review of a fellow developer’s code. A required sign-off from another team member before a developer is permitted to check in changes or new code.

- **Analogy**: when writing articles for a newspaper, what is the effectiveness of …
  - spell-check/grammar check?
  - author editing own article?
  - others editing others’ articles?
Code reviews: mechanics
Code reviews: mechanics

• **Who**: original developer and reviewer, sometimes together in person, sometimes offline.
Code reviews: mechanics

- **Who**: original developer and reviewer, sometimes together in person, sometimes offline.
- **What**: reviewer gives suggestions for improvement on a logical and/or structural level, to conform to a common set of quality standards.
  - Feedback leads to refactoring.
  - Reviewer eventually approves code.
Code reviews: mechanics

- **Who**: original developer and reviewer, sometimes together in person, sometimes offline.

- **What**: reviewer gives suggestions for improvement on a logical and/or structural level, to conform to a common set of quality standards.
  - Feedback leads to refactoring.
  - Reviewer eventually approves code.

- **When**: code author has finished a coherent system change that is otherwise ready for checkin
  - Change shouldn't be too large or too small.
  - Before committing the code to the repository or incorporating it into the new build.
Code reviews: why do them?
Code reviews: why do them?

• Improved code quality
  • Prospect of someone reviewing your code raises quality threshold.
  • Forces code authors to articulate their decisions.
Code reviews: why do them?

• Improved code quality
  • Prospect of someone reviewing your code raises quality threshold.
  • Forces code authors to articulate their decisions.

• Hands-on learning experience from peers
  • Direct feedback leads to better algorithms, tests, design patterns.
**Code reviews: why do them?**

- Improved code quality
  - Prospect of someone reviewing your code raises quality threshold.
  - Forces code authors to articulate their decisions.

- Hands-on learning experience from peers
  - Direct feedback leads to better algorithms, tests, design patterns.

- Better understanding of complex code bases
  - Reviewing others’ code enhances overall understanding of the system, reduces redundancy.
Code reviews: studies
Code reviews: studies

- Average defect detection rates
  - Unit testing: 25%
  - Function testing: 35%
  - Integration testing: 45%
  - Design and code inspections: 55% and 60%.
Code reviews: studies

• Average defect detection rates
  • Unit testing: 25%
  • Function testing: 35%
  • Integration testing: 45%
  • Design and code inspections: 55% and 60%.

• 11 programs developed by the same group of people
  • First 5 without reviews: average 4.5 errors per 100 lines of code
  • Next 6 with reviews: average 0.82 errors per 100 lines of code
  • Errors reduced by > 80 percent.
Code reviews: who does them?
Code reviews: who does them?

• **Everyone**: a common industry practice.
Code reviews: who does them?

- **Everyone**: a common industry practice.
- Made easier by advanced tools that
  - integrate with version control
  - highlight changes (i.e., diff function)
  - e.g., github pull requests
kinds of code reviews
Common types of code review

• Tool-assisted reviews
• Formal inspections
• Walkthroughs
• Pair programming
Tool-assisted code reviews
Tool-assisted code reviews

• Most common form of code review
  • Authors and reviewers use software tools designed for peer code review.
  • The tool gathers files, displays diffs and comments, enforces reviews.
Tool-assisted code reviews

• Most common form of code review
  • Authors and reviewers use software tools designed for peer code review.
  • The tool gathers files, displays diffs and comments, enforces reviews.

• Advantages
  • Lightweight, integrated into the workflow.
Tool-assisted code reviews

• Most common form of code review
  • Authors and reviewers use software tools designed for peer code review.
  • The tool gathers files, displays diffs and comments, enforces reviews.

• Advantages
  • Lightweight, integrated into the workflow.

• Disadvantages
  • Hard to ensure review quality and promptness.
Tool-assisted code reviews

NO NEED TO DOUBLE CHECK THIS CHANGE LIST, IF SOME PROBLEMS REMAIN THE REVIEWER WILL CATCH THEM.

NO NEED TO LOOK AT THIS CHANGE LIST TOO CLOSELY, I’M SURE THE AUTHOR KNOWS WHAT HE’S DOING.
Formal inspections
Formal inspections

• A more formalized code review with
  • roles (moderator, author, reviewer, scribe, etc.)
  • several reviewers looking at the same piece of code
  • a specific **checklist** of kinds of flaws to look for
    • flaws that have been seen previously
    • high-risk areas such as security
Formal inspections

• A more formalized code review with
  • roles (moderator, author, reviewer, scribe, etc.)
  • several reviewers looking at the same piece of code
  • a specific checklist of kinds of flaws to look for
    • flaws that have been seen previously
    • high-risk areas such as security

• Advantages
  • High review quality with specific expected outcomes
    (e.g. report, list of defects)
Formal inspections

• A more formalized code review with
  • roles (moderator, author, reviewer, scribe, etc.)
  • several reviewers looking at the same piece of code
  • a specific checklist of kinds of flaws to look for
    • flaws that have been seen previously
    • high-risk areas such as security

• Advantages
  • High review quality with specific expected outcomes
    (e.g. report, list of defects)

• Disadvantages
  • Heavyweight, time-consuming, expensive
Walkthroughs

THAT LINE OF CODE GIVES ME GAS
Walkthroughs

- An informal discussion of code between author and a single reviewer.
  - The author walks the reviewer through a set of code changes.
Walkthroughs

• An informal discussion of code between author and a single reviewer.
  • The author walks the reviewer through a set of code changes.

• Advantages
  • Simplicity in execution: anyone can do it, any time.
  • In-person interaction, learning, and sharing.
Walkthroughs

• An informal discussion of code between author and a single reviewer.
  • The author walks the reviewer through a set of code changes.

• Advantages
  • Simplicity in execution: anyone can do it, any time.
  • In-person interaction, learning, and sharing.

• Disadvantages
  • Not an enforceable process, no record of the review.
  • Easy for the author to unintentionally miss a change.
  • Reviewers rarely verify that defects were fixed.
Pair programming
Pair programming

• Two developers writing code at a single workstation with
  • only one typing
  • continuous free-form discussion and review
Pair programming

• Two developers writing code at a single workstation with
  • only one typing
  • continuous free-form discussion and review

• Advantages
  • Deep reviews, instant and continuous feedback.
  • Learning, sharing, team-building.
Pair programming

• Two developers writing code at a single workstation with
  • only one typing
  • continuous free-form discussion and review

• Advantages
  • Deep reviews, instant and continuous feedback.
  • Learning, sharing, team-building.

• Disadvantages
  • Some developers don’t like it.
  • No record of the review process.
  • Time consuming.
a code review example
What changes, if any, would you suggest?

```java
public class Account {
    double principal, rate; int daysActive, accountType;
    public static final int STANDARD = 0, BUDGET = 1,
    PREMIUM = 2, PREMIUM_PLUS = 3;
}

public static double calculateFee(Account[] accounts)
{
    double totalFee = 0.0;
    Account account;
    for (int i = 0; i < accounts.length; i++) {
        account = accounts[i];
        if (account.accountType == Account.PREMIUM ||
            account.accountType == Account.PREMIUM_PLUS)
            totalFee += .0125 * ( // 1.25% broker's fee
                account.principal * Math.pow(account.rate,
                (account.daysActive / 365.25))
                - account.principal); // interest-principal
    }
    return totalFee;
}
```
Possible changes

• Comment.
• Make fields private.
• Replace magic values (e.g. 365.25) with constants.
• Use an enum for account types.
• Use consistent whitespace, line breaks, etc.
/** An individual account. Also see CorporateAccount. */
public class Account {
    /** The varieties of account our bank offers. */
    public enum Type {STANDARD, BUDGET, PREMIUM, PREMIUM_PLUS}

    /** The portion of the interest that goes to the broker. */
    public static final double BROKER_FEE_PERCENT = 0.0125;

    private Type type;
    private double principal;

    /** The yearly, compounded rate (at 365.25 days per year). */
    private double rate;

    /** Days since last interest payout. */
    private int daysActive;

    ...
/** Compute interest on this account. */
public double interest() {
    double years = daysActive / 365.25;
    double compoundInterest = principal * Math.pow(rate, years);
    return compoundInterest - principal;
}

/** Return true if this is a premium account. */
public boolean isPremium() {
    return accountType == Type.PREMIUM ||
            accountType == Type.PREMIUM_PLUS;
}

/** Return the sum of broker fees for all given accounts. */
public static double calculateFee(Account[] accounts) {
    double totalFee = 0.0;
    for (Account account : accounts) {
        if (account.isPremium()) {
            totalFee += BROKER_FEE_PERCENT * account.interest();
        }
    }
    return totalFee;
}
Summary

- Code reviews improve
  - code quality
  - teamwork
  - knowledge and skills

- Kinds of code review
  - tool-assisted
  - formal inspections
  - walkthroughs
  - pair programming