CSE 403
Lecture 10

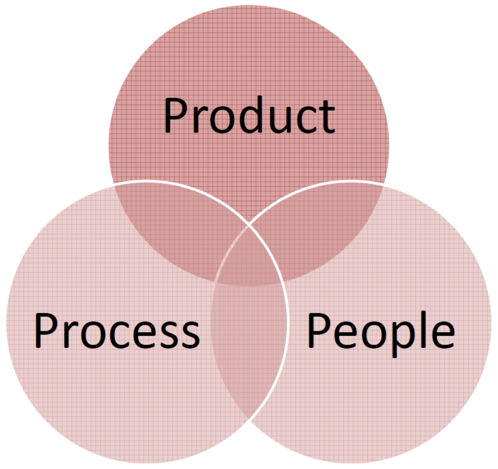
Code Reviews

slides created by Marty Stepp
http://www.cs.washington.edu/403/

Software QA

It is difficult to assure the quality of a software product.

- What are we assuring?
  - Building the right system? Building the system right?
    - (validation versus verification)
  - Presence of good properties / absence of bad properties?
  - Identifying errors? Confidence in the absence of errors?
  - Robust? Safe? Secure? Available? Reliable?
  - Understandable? Modifiable? Cost-effective? Usable? …
- Why are we assuring it?
  - Business/Economic reasons, Legal, Ethical, Professional, Social, Personal satisfaction, ...
- How do we assure it?
  - How do we know we have assured it?
    - Depends on "it"
    - Depends on meaning of "assurance"



Large code challenges

- How to ensure…
  - Maintainable code?
  - DRY code?
  - Readable code?
  - Bug-free code?

- Average defect detection rate for various testing
  - Unit testing: 25%
  - Function testing: 35%
  - Integration testing:45%

  - How can this be improved?

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: writing articles for a newspaper
  - What is the effectiveness of…
    - Spell-check/grammar check?
    - Author editing own article?
    - Others editing others' articles?

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 previously agreed upon set of quality standards.
  - Feedback leads to refactoring, followed by a 2nd code review.
  - Eventually reviewer approves code.

- *when:* 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

Why do code reviews?

- > 1 person has seen every piece of code
  - Prospect of someone reviewing your code raises quality threshold.
- Forces code authors to articulate their decisions
- Hands-on learning experience for rookies without hurting code quality
  - Pairing them up with experienced developers
- Team members involved in different parts of the system
  - Reduces redundancy, enhances overall understanding
- Author and reviewer both accountable for committing code

What *not* to use code review for

- Assessment of individuals for
  - Promotion
  - Pay
  - Ranking
  - Personal attack
  - Etc.

- Focus should be on the code, not the person
  - http://www.codinghorror.com/blog/2006/05/the-ten-commandments-of-egoless-programming.html

- Management may not be permitted at reviews for this reason
  - Not all companies adhere to this philosophy

Actual Studies

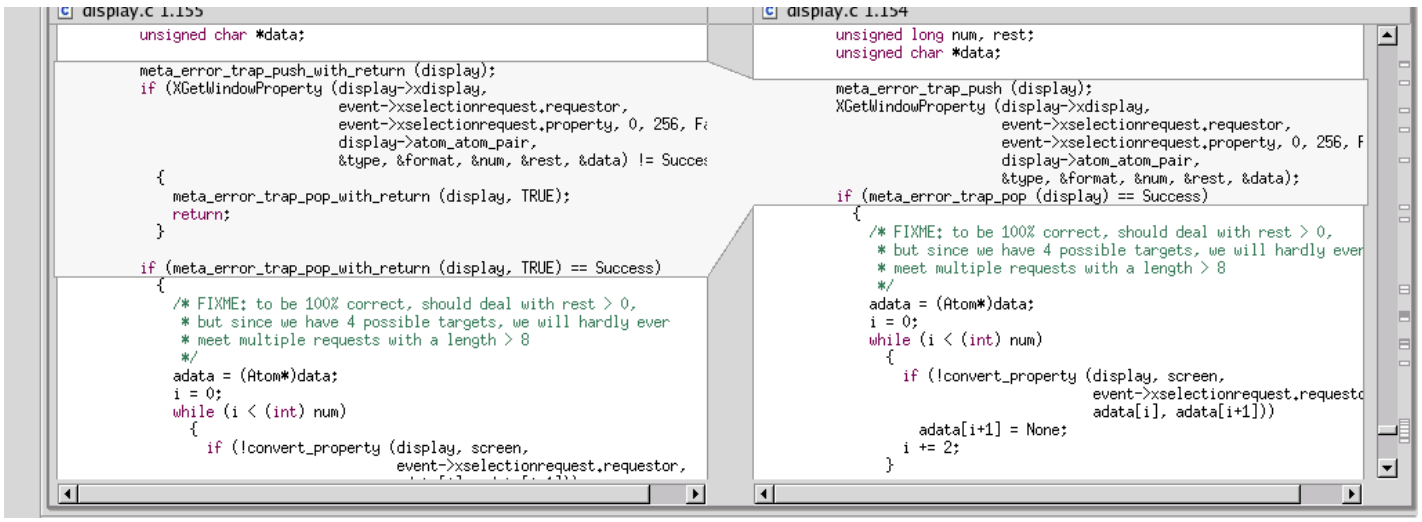
- Average defect detection rates
  - Unit testing: 25%
  - Function testing: 35%
  - Integration testing:45%
  - **Design and code inspections: 55% and 60%.**

- 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.**
    - IBM's Orbit project: 500,000 lines, 11 levels of inspections. Delivered early with only 1 % of the errors that would normally be expected.
    - After AT&T introduced reviews, study with > 200 people reported +14% productivity, -90% defects.
      - (From Steve McConnell's *Code Complete*)

Code Reviews in Industry

- Code reviews are a *very* common industry practice.

- Made easier by advanced tools that:
  - integrate with configuration management systems
  - highlight changes (i.e., diff function)
  - allow traversing back into history
  - E.g.: Eclipse, Git/SVN tools



display.c 1.155

```
        unsigned char *data;

        meta_error_trap_push_with_return (display);
        if (XGetWindowProperty (display->xdisplay,
                                event->xselectionrequest.requestor,
                                event->xselectionrequest.property, 0, 256, Fa
                                display->atom_atom_pair,
                                &type, &format, &num, &rest, &data) != Succes
          {
            meta_error_trap_pop_with_return (display, TRUE);
            return;
          }

        if (meta_error_trap_pop_with_return (display, TRUE) == Success)
          {
            /* FIXME: to be 100% correct, should deal with rest > 0,
             * but since we have 4 possible targets, we will hardly ever
             * meet multiple requests with a length > 8
             */
            adata = (Atom*)data;
            i = 0;
            while (i < (int) num)
              {
                if (!convert_property (display, screen,
                                       event->xselectionrequest.requestor,
```

display.c 1.154

```
        unsigned long num, rest;
        unsigned char *data;

        meta_error_trap_push (display);
        XGetWindowProperty (display->xdisplay,
                            event->xselectionrequest.requestor,
                            event->xselectionrequest.property, 0, 256, F
                            display->atom_atom_pair,
                            &type, &format, &num, &rest, &data);
        if (meta_error_trap_pop (display) == Success)
          {
            /* FIXME: to be 100% correct, should deal with rest > 0,
             * but since we have 4 possible targets, we will hardly ever
             * meet multiple requests with a length > 8
             */
            adata = (Atom*)data;
            i = 0;
            while (i < (int) num)
              {
                if (!convert_property (display, screen,
                                       event->xselectionrequest.requesto
                                       adata[i], adata[i+1]))
                  adata[i+1] = None;
                i += 2;
              }
```

Variations

- **inspection**: 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
    - possibly focusing on flaws that have been seen previously
    - possibly focusing on high-risk areas such as security
    - possibly focusing on coding standards
      - automated tools (type checkers, lint, etc.) may be better for this
  - specific expected outcomes (e.g. report, list of defects)

- **walkthrough**: informal discussion of code between author and a single reviewer; can be useful to “play computer” by tracing execution manually away from the machine

- **code reading**: Reviewers look at code by themselves (possibly with no actual meeting)

Code Reviews at Google

- "All code that gets submitted needs to be reviewed by at least one other person, and either the code writer or the reviewer needs to have readability in that language."
- "Most people use Mondrian to do code reviews, and obviously, we spend a good chunk of our time reviewing code."
  - – Amanda Camp, Software Engineer, Google

Code Reviews at Yelp

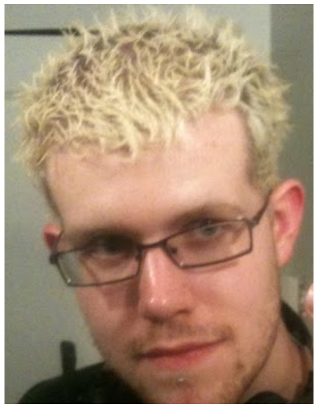
- "At Yelp we use review-board. An engineer works on a branch and commits the code to their own branch. The reviewer then goes through the diff, adds inline comments on review board and sends them back."
- "The reviews are meant to be a dialogue, so typically comment threads result from the feedback. Once the reviewer's questions and concerns are all addressed they'll click "Ship It!" and the author will merge it with the main branch for deployment the same day."

  – Alan Fineberg, Software Engineer, Yelp

Code Reviews at WotC

- "At Wizards we use Perforce for SCM. I work with stuff that manages rules and content, so we try to commit changes at the granularity of one bug at a time or one card at a time. Our team is small enough that you can designate one other person on team as a code reviewer."
- "Usually you look at code sometime that week, but it depends on priority. It's impossible to write sufficient test harnesses for the bulk of our game code, so code reviews are absolutely critical."

  – Jake Englund, Software Engineer, MtGO



Code Reviews at Facebook

- "At Facebook, we have an internally-developed web-based tool to aid the code review process. Once an engineer has prepared a change, she submits it to this tool, which will notify the person or people she has asked to review the change, along with others that may be interested in the change -- such as people who have worked on a function that got changed. At this point, the reviewers can make comments, ask questions, request changes, or accept the changes. If changes are requested, the submitter must submit a new version of the change to be reviewed. All versions submitted are retained, so reviewers can compare the change to the original, or just changes from the last version they reviewed. Once a change has been submitted, the engineer can merge her change into the main source tree for deployment to the site during the next weekly push, or earlier if the change warrants quicker release."

  – Ryan McElroy, Software Engineer, Facebook

Code Reviews in CSE 403

- code reviews required for each code phase (Alpha, Beta, V1)
  - Idea: 3 per week, one review per developer
  - Not required to review before checkin
- Need to document code review in some fashion
  - Email thread, checklist, tools, etc.
- Exact structure up to group: something between walkthrough and inspection
- Don't review "trivial" code but "trivial" threshold is probably lower than you think
  - example: exception tests added, good

Counter-argument 1

- *"I don't like doing a code review of every commit. It takes too long. I want to make lots of commits and it slows me down."*

- Responses?
  - Maybe you are making too many commits.
  - Commit coherent, complete changes to the code base, not incomplete work or tiny changes.
  - If something was quick to write, or a small change, it will also be quick to code review.
  - If you check in buggy code, you'll lose more time than it would have taken to code review it and (hopefully) find the bug.

Counter-argument 2

- *"We tried doing code reviews, but it was not useful. We never find any bugs or errors; the code is always approved."*

- Responses?
  - Maybe the reviewer should be more thorough. Almost all reviews of non-trivial code uncover at least *some* issues to look into.
  - Sometimes the point is not to find flaws and fix them, but also to make sure that another developer is familiar with that code.
  - The fact that the original coder knew it would be reviewed increases the chance that they wrote better code to start with.

Code Review Example

- What changes (if any) would you suggest?

```
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 Code 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.

Improved Code

```
/** 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;
    }
    ...
```

Improved Code 2

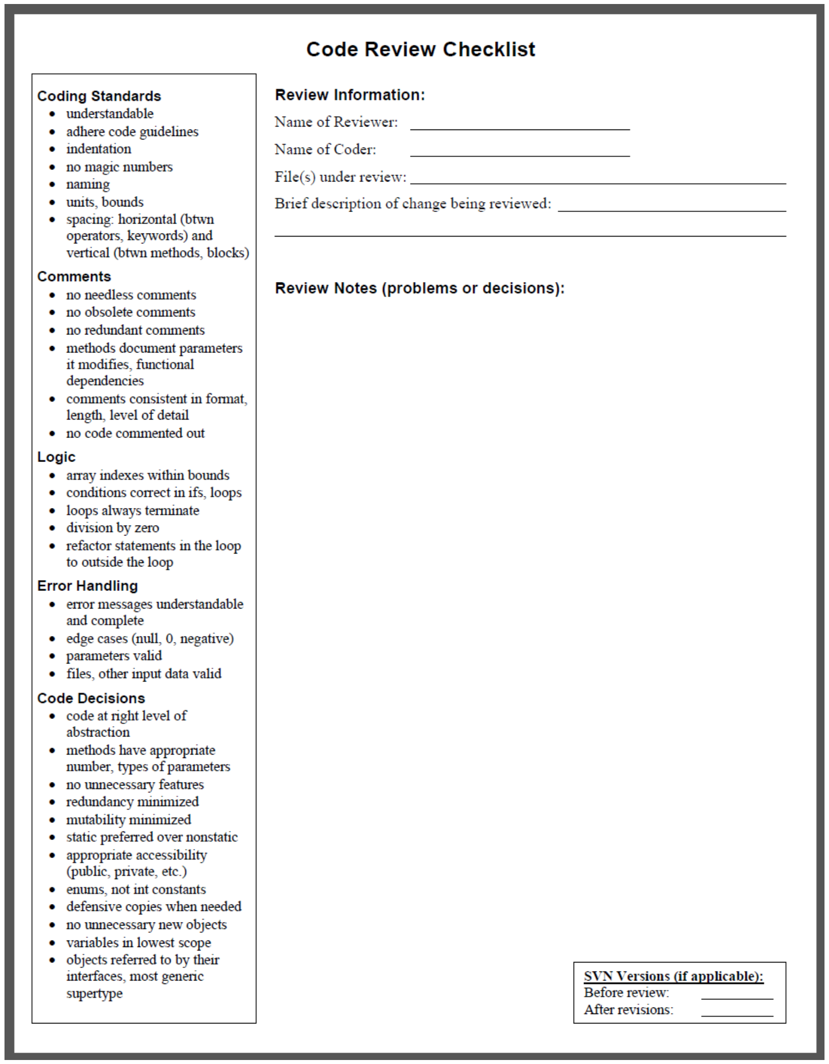
```
        ...
        /** Return true if this is a premium account. */
        public boolean isPremium() {
            return accountType == Type.PREMIUM ||
                   accountType == Type.PREMIUM_PLUS;
        }

        /** 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;
        }
    }
```

Code Review Checklist

- Useful to divide checks into categories, such as:
  - Coding Style Standards
  - Comments
  - Logic
  - Error Handling
  - Coding Decisions

- Some tools (e.g. GitHub) allow you to comment directly on the code in the commit.



Code Review Checklist

Coding Standards
- understandable
- adhere code guidelines
- indentation
- no magic numbers
- naming
- units, bounds
- spacing: horizontal (btwn operators, keywords) and vertical (btwn methods, blocks)

Comments
- no needless comments
- no obsolete comments
- no redundant comments
- methods document parameters it modifies, functional dependencies
- comments consistent in format, length, level of detail
- no code commented out

Logic
- array indexes within bounds
- conditions correct in ifs, loops
- loops always terminate
- division by zero
- refactor statements in the loop to outside the loop

Error Handling
- error messages understandable and complete
- edge cases (null, 0, negative)
- parameters valid
- files, other input data valid

Code Decisions
- code at right level of abstraction
- methods have appropriate number, types of parameters
- no unnecessary features
- redundancy minimized
- mutability minimized
- static preferred over nonstatic
- appropriate accessibility (public, private, etc.)
- enums, not int constants
- defensive copies when needed
- no unnecessary new objects
- variables in lowest scope
- objects referred to by their interfaces, most generic supertype

Review Information:

Name of Reviewer: ____________________

Name of Coder: ____________________

File(s) under review: ____________________

Brief description of change being reviewed: ____________________

____

Review Notes (problems or decisions):

SVN Versions (if applicable):
Before review: ________
After revisions: ________