

COMS W4156 Advanced Software Engineering (ASE)

October 13, 2022

Agenda

1. Working in Teams
2. CheckStyle demo
3. Pair Programming



Software Engineers always work in Teams

People who write run-once programs are
programmers not software engineers

Teams always have at least three
members: you, the past you, and the
future you



You / Past You / Future You

Motivate...

version control - It worked last week, why isn't it working now?

coding style - Why did I name this function 'FixThisLater' two years ago?

code documentation - What does 'FixThisLater' do?

task board - Where is the sticky note with my to-do list?

....even if your team never grows



Multi-Member Teams

There are usually more members, all of which join the team at some point (not necessarily beginning) and may leave at some point (not necessarily end)

A team of 4-5 software engineers often has >5 other members (some shared with other teams), such as tech lead, engineering manager, product manager, tech writer, customer support and/or devops liaison, UI/UX designer and/or researcher (when applicable), possibly a separate testing team, with several layers of management above each

Some teams also have student interns or trainees



Multi-Member Engineering Teams



Mandate...

version control - My code worked last week, now it doesn't, who changed it?

coding style - Why did Jing, who left two years ago to work for Kookle, name this function 'FixThisLater'?

code documentation - What does 'FixThisLater' do?

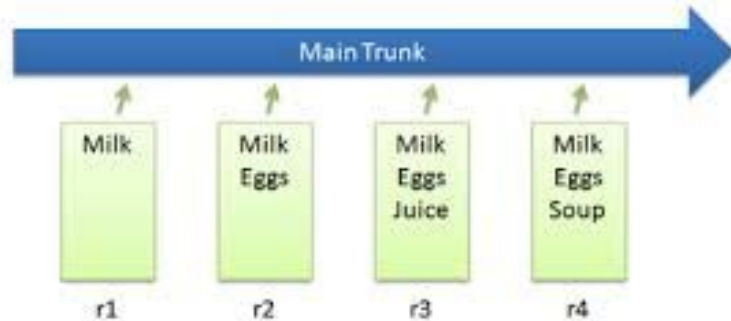
task board - What am I supposed to be working on?

...particularly as team grows or churns

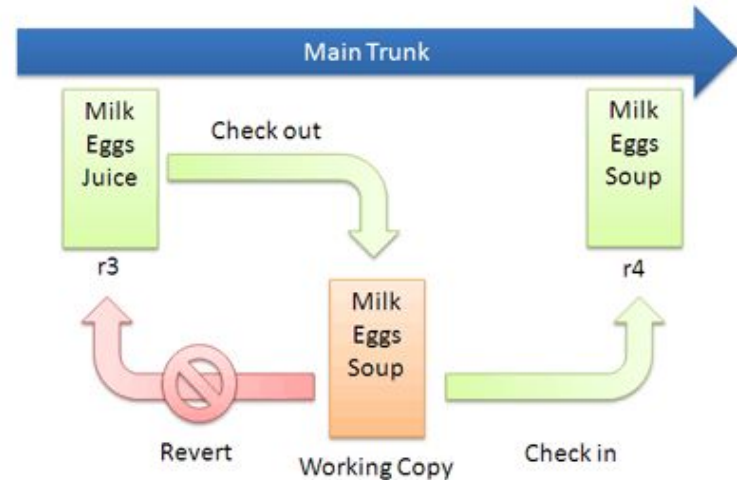
Version Control

All (or nearly all) of you already understand basic version control

Basic Checkins



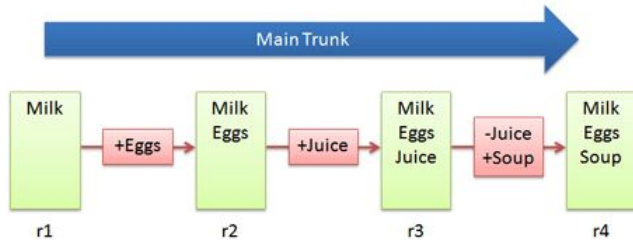
Checkout and Edit



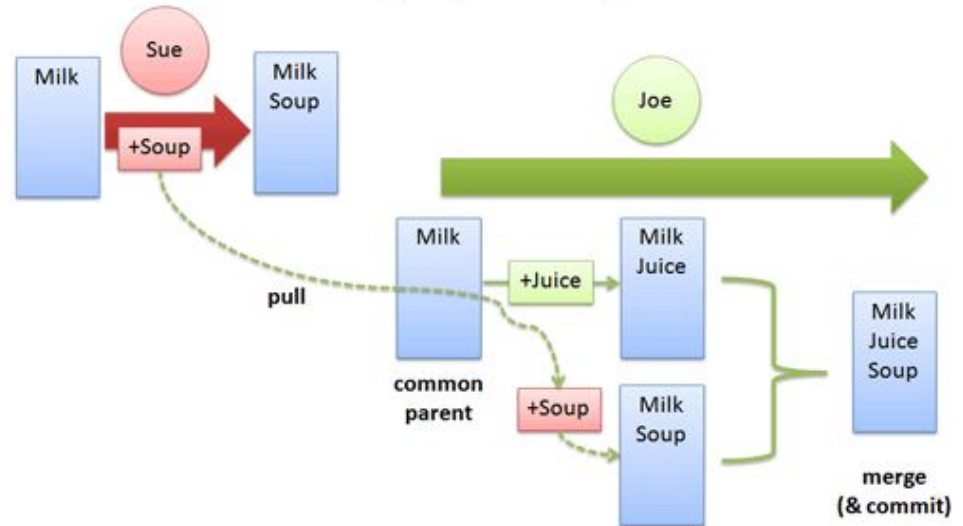
Version Control

But maybe not how merging changes works

Basic Diffs



Merging Changes



Example Auto-Merge

```
int main(){
    char* ptr;
    FILE* data;
    char buf[100];
    ....some code....
    data = fopen("data.txt", "r");
    if(fgets(buf,100,data) == NULL)
        return 1;
    printf("Read: %s", buf);
    fclose(data);
    data = fopen("data.txt", "w");
    fwrite("juice\n", 6, 1, data);
    fclose(data);
    return 0;
}
```

```
int main(){
    char* ptr;
    FILE* data;
    char buf[100];
    ....some code...
    data = fopen("data.txt", "r");
    if(fgets(buf,100,data) == NULL)
        return 1;
    printf("Read: %s", buf);
    fclose(data);
    data = fopen("data.txt", "w");
    fwrite("soup\n", 7, 1, data);
    fclose(data);
    return 0;
}
```

Example Probably Needs Manual Resolution

```
/* original */
void display(char* str){
    write(1, str, strlen(str));
}
int main(){
    char* msg_ok = "Ok!\n";
    char* user = calloc(256, 1);
    char* pass = calloc(256, 1);
    strncpy(user, "default", 8);
    strncpy(pass, "default", 8);
    ...some code...
    write(2, msg_ok, strlen(msg_ok));
    display(user);
    return 0;
}
```

```
/* concurrent change */
void display(char* str){
    write(1, str, strlen(str));
}
int main(){
    char* msg_ok = "Ok!\n";
    char* user = calloc(256, 1);
    char* pass = calloc(256, 1);
    strncpy(user, "default", 8);
    strncpy(pass, "default", 8);
    .... some code...
    read(0, user, 255);
    read(0, pass, 255);
    write(2, msg_ok, strlen(msg_ok));
    display(user);
    return 0;
}
```

```
/* concurrent change */
void display(char* str){
    write(1, str, strlen(str));
}
int main(){
    char* msg_ok = "Ok!\n";
    char* user = calloc(256, 1);
    char* passwd = calloc(256, 1);
    strncpy(user, "default", 8);
    strncpy(passwd, "default", 8);
    ...some code...
    read(0, passwd, 255);
    write(2, msg_ok, strlen(msg_ok));
    display(user);
    return 0;
}
```

Release Tags

When you submit an iteration or present a demo, **tag** the revisions that contributed to that milestone so you can return to it later



git docubot

File Edit View Help

Local uncommitted changes, not checked in to index

- 1.1.2 master -> remotes/origin/master Package
- Note license in readme
- Add explicit MIT license
- 1.1.1 master -> remotes/origin/master
- Treat mailto: href as unvalidated external links (fixes #)
- 1.1.0 master -> remotes/origin/master
- Force proper ordering of files (in case the file system dc)
- 1.1.0 Add option to skip pages based on meta i
- 1.1.0 Do not use line numbers for markdown syntax
- 1.0.1 master -> remotes/origin/master
- Fix bug with ampersands in filenames linked from TOC
- 0.9.0 Add test files (reused) with special characters
- Update release notes
- Fix TOC lookup to be based on text of element rather th
- Adjust fragile specs to account for new feature that finds
- Add sure-to-fail TOC entry for specs
- Additional require_relative changes
- Fix CHM writer to work from pure Ruby outside of the bir
- Remove rubygems require; add more helpful error mes
- Read every file as UTF-8
- Use require_relative; spec to reproduce bug with mixed
- 0.7.1 Update stats
- Add files needed to create the directories in the git repo
- Do not make TOC entries for directories inside _stat
- 0.7.0 master -> remotes/origin/master
- Fix broken link in markdown.
- Fix bug creating CHM files for folders with ampersands
- 0.6.1 updating version
- logfile option
- Update version number for new release
- Adding a -n command line option for skipping auto-prev
- 0.5 Bump version number for release
- support %20 in links
- Ignore pretty much anything that starts with underscore.
- solo octothorpe may be okay for link target
- allow relative output paths
- ignore my local dev setup
- Minor readme cleanups
- Minor readme cleanups
- Remove outmoded question-mark metasection looking
- Tweak stats slightly
- Save a snapshot of stats for posterity
- Useless stat counter
- Adding missed specs for CHM writer
- Properly fix CHM writer to work with specified default top
- Revertin host % in L4BP navigation for rhmc

SHA1 ID: 5577b4bd2412bc9033b7d9581c836942b79b7bc2

next prev commit containing: Exact All fields

Search

Diff Old version New version Lines of context: 3

Author: Gavin Kistner <gavin@phrogz.net> 2012-07-10

Committer: Gavin Kistner <gavin@phrogz.net> 2012-07-10

Tag: 1.0.1

Parents: 28a91b7779a0b12c6a201b32c5f4aaf6438907

Child: 28a91b7779a0b12c6a201b32c5f4aaf6438907

Branches: master, remotes/origin/master

Follows: 1.0.0

Precedes: 1.0.2

Fix formatting typo in CHANGES

CHANGES.md

index: 3f0c561 c64784c 100644

Releases · Phrogz/docubot

GitHub, Inc. [US] https://github.com/Phrogz/docubot/releases

Releases / Tags

Latest release

1.0.1

Phrogz released this 24 minutes ago · 8 commits to master

- Fix problems with Table of Contents linking to files/folders

Source code (zip) Source code (tar.gz)

0.7.0

Phrogz released this 25 minutes ago · 23 commits to master

- Switched to using kramdown instead of BlueCloth for Ma

Source code (zip) Source code (tar.gz)

1.1.2

Phrogz released this 26 minutes ago · 0 commits to master

- Treat mailto: href as unvalidated external links (fixes i
- Added explicit MIT License.

Source code (zip) Source code (tar.gz)

1.1.1

Phrogz released this 6 months ago · 4 commits to master

Coding Style

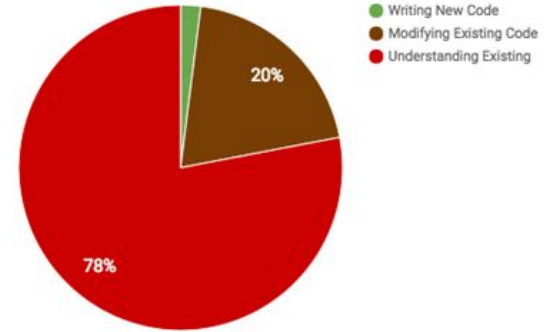
All (or nearly all) of you learned some standard coding style when you learned to program in xxx

Governs how and when to use comments and whitespace (indentation, blank lines), proper naming of variables and functions (e.g., camelCase, snake_case), code grouping and organization, file/folder structure

Compliance with a team's coding style helps other developers understand your code and vice versa, even helps your future self understand your past self's code

See Google's [many style guides](#)

How Software Engineers Spend Time



There are two types of people:

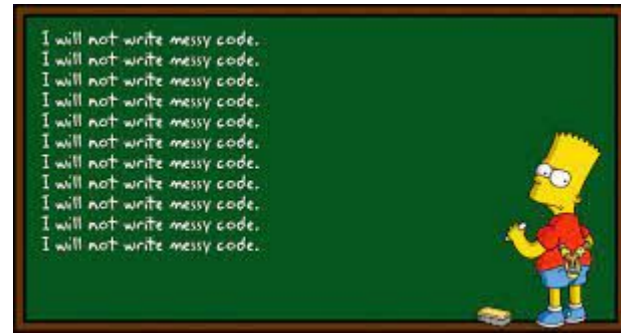
```
if (Condition) {  
    Statement  
    /* ....  
    */  
}
```

```
if (Condition)  
{  
    Statement  
    /* ....  
    */  
}
```

What is Wrong with this Code?

```
switch(value) {  
    case 1:  
        doSomething();  
  
    case 2:  
        doSomethingElse();  
        break;  
  
    default:  
        doDefaultThing();  
}
```

Style Checkers



Most languages have automated tools that detect deviations from a prescribed coding style, e.g., [checkstyle](#)

The auto-merge feature of git and other version control systems depend on style compliance to avoid superficial, format-related merge conflicts - so should always run style checking and fix any problems *before* committing code changes

Running the style checker (and doing various other things) prior to each commit can be automated for github with a [git pre-commit hook](#)

Code Documentation



See Google's [Documentation Best Practices](#) and [Engineering Practices](#)
- *"Documentation is the story of your code"*

A non-trivial method (function, procedure, subroutine, etc.) has *inline* comments that explain “why” the code is written this way or “why” it works this way - intended audience is future developers who modify

Methods have a header (Javadoc, docstring, etc.) that explains “what” the method does and “how” to use it - intended audience is future developers who use or modify

Every behavior documented in a method header has corresponding test cases to verify that behavior, and the test cases themselves are documented with what arguments the method takes, what it returns, any “gotchas” or restrictions, and what exceptions it can throw or errors it can return

Code Documentation



Classes (modules, files, etc.) document the class as a whole, overview what the class does and show examples of how it might be used (both simple and advanced, if applicable) - intended audience is future developers who use or modify

Folders (directories) have a [README.md](#) file that states What is this directory intended to hold? Which files should the developer look at first? Are some files an API? Who maintains this directory and where can I learn more?

There are even “prose-linters” like [Vale](#) that check some documentation content - different from spelling/grammar checkers, because they know they are checking code documentation (and can deal with code syntax in the midst of prose)

Task Boards

Documents who is working on what this iteration




















Tasks boards show to-do, in-progress, etc. columns of tickets - each with title, priority, who assigned to, tags, description, project its part of, and links to other tickets, revisions and/or releases

Tickets typically written and assigned by team leader or product owner

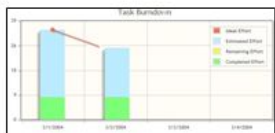
[JIRA](#) and [Trello](#) probably best-known systems, there's free versions for small teams

Github has built-in “issues” and “projects”

The screenshot displays a Jira Task Board for the 'Teams in Space' project. The interface includes a top navigation bar with 'Your work', 'Projects', 'Filters', 'Dashboards', 'People', 'Plans', 'Apps', and a 'Create' button. A search bar is located on the right. The left sidebar shows the project's navigation menu, including 'Scrum: Teams in S...', 'Roadmap', 'Backlog', 'Active sprints', 'Reports', 'Issues', 'Components', 'Releases', 'Project pages', 'Add item', and 'Project settings'. The main area is titled 'Board' and features a 'Quick Filters' dropdown. The board is organized into four columns: 'TO DO 5', 'IN PROGRESS 5', 'CODE REVIEW 2', and 'DONE 8'. Each column contains a list of tasks with their titles, priorities, assignees, and due dates. For example, in the 'TO DO' column, tasks include 'Engage Jupiter Express for outer solar system travel' (assigned to TIS-25) and 'Create 90 day plans for all departments in the Mars Office' (assigned to TIS-12). The 'IN PROGRESS' column shows tasks like 'Requesting available flights is now taking > 5 seconds' (assigned to TIS-8) and 'Engage Saturn Shuttle Lines for group tours' (assigned to TIS-15). The 'CODE REVIEW' column has tasks such as 'Register with the Mars Ministry of Revenue' (assigned to TIS-11) and 'Draft network plan for Mars Office' (assigned to TIS-15). The 'DONE' column lists completed tasks like 'Homepage footer uses an inline style - should use a class' (assigned to TIS-68) and 'Engage JetShuttle SpaceWays for travel' (assigned to TIS-23).

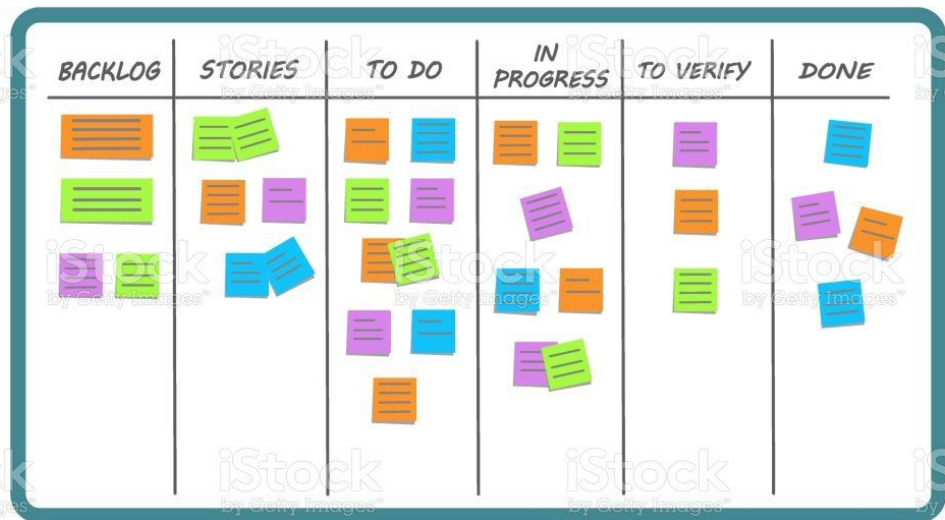
User Stories this Sprint	Tasks To Do	Coding	Test	Done
Urgent Tasks				
	 			
	  			
		 		 

Impediment



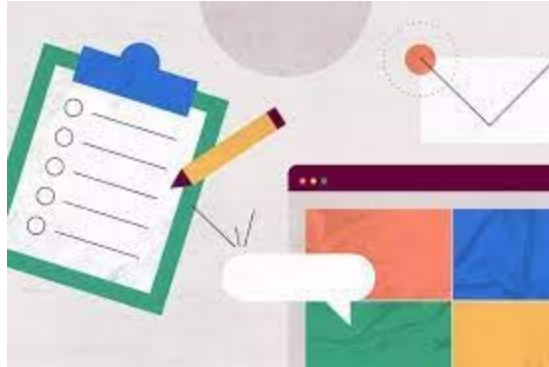
Burndown Chart

'Done' Conditions



Agenda

1. Working in Teams
2. CheckStyle demo
3. Pair Programming



CheckStyle demo: Ira Ceka

[demo presentation](#)



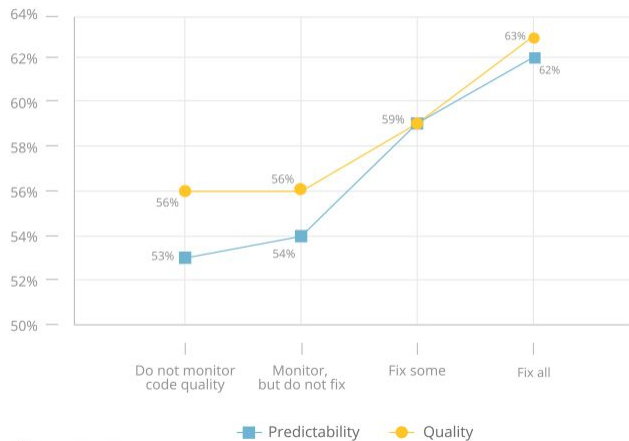
Read the Docs

[CheckStyle](#) is for Java

[SonarLint](#) is for C++ (and for Java and many other languages)

Also see [SonarQube + SonarLint](#)

The effects of fixing code quality issues on predictability and quality



Agenda

1. Working in Teams
2. CheckStyle demo
3. Pair Programming

Multi-Member Engineering Teams

Enable...

[pair programming](#)!



Ought to be called “pair software engineering”,
but “pair programming” sounds better



Used for many software engineering activities, not just programming

Some companies use [pair-programming interviews](#) between job candidate and evaluator (current employee) even if their employees do not pair-program routinely

Ideal Pair Programming

Two people sit side by side at same computer or use remote desktop sharing

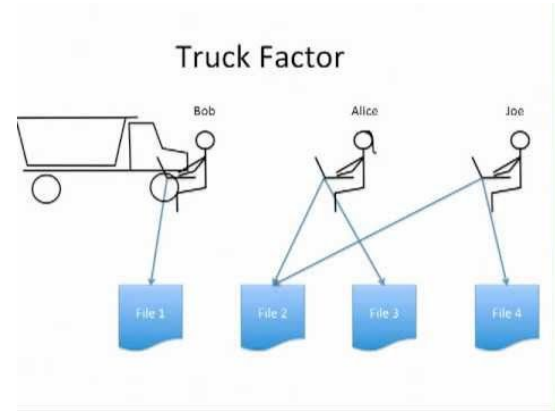
- Take turns “driving” (typing) vs. “navigating” (continuous review of code or whatever working on)
- Switch roles at frequent intervals (eg, [25 minutes](#))
- The pair does **not** divide up the work, they do everything **together**
- If necessary to work separately on something, eg, when one partner is absent, they review together



Benefits of Pair Programming

Reduce risk through collective code ownership and maximize “truck factor” = number of engineers that would have to disappear before project would be in serious jeopardy

- Shared responsibility to complete tasks on time
- Stay focused and on task
- Less likely to read email, surf web, etc.
- Less likely to be interrupted
- Partners expect “best practices” from each other
- Two people can solve problems that one couldn’t do alone and/or produce better solutions
- Pool knowledge resources, improve skills



Pair Programming

Instant code review happens on the fly during pair programming, where the navigator reviews while the driver writes/edits



Pair programming styles



Unstructured

In unstructured pair programming, the developers can trade off who takes the lead, and should discuss decisions about the code.



Driver/Navigator

In the driver/navigator approach to pair programming, one developer sets the architectural or strategic direction, and the other implements these decisions as code.



Ping-pong

Ping-pong pair programming shifts rapidly back-and-forth between the two developers, like a game of ping pong, where the software is the ball.



Code Review

Style checkers and static analysis bug finders cannot find all bad code (static analysis bug finders discussed next week)

In “[code review](#)”, one or more *humans* read and analyze the code

Like automated static analysis, code review can potentially consider *all* inputs whereas testing can only consider *some* inputs - often a very small subset of all inputs.

Code review can find problems that dynamic testing and static analyzers cannot, such as code that passes style checking and other static analyses, and passes all test cases, but is still [unreadable](#)

Humans reviewers can also spot application-specific bugs as well as generic bugs that are beyond the state of the art in static analysis

Training



Pair programming helps train junior developers -
over-the-shoulder synchronous code review by senior developer

Lightweight Code Review



Many organizations require code to be reviewed by a peer, a senior developer, or a designated reader prior to committing to the shared code repository

Lightweight forms of code review involve one or two readers besides the coder

May be synchronous (reviewer sits with coder immediately after coder “done”) or asynchronous (separately)

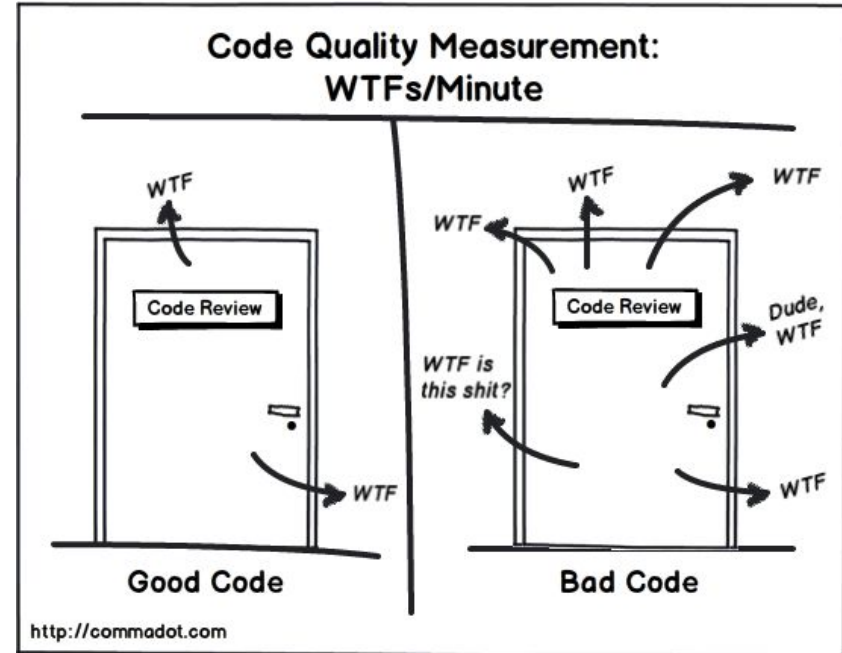
Aside from pair programming and junior developer training, there are [major problems with synchronous reviews](#), so asynchronous generally considered better - submit prospective changes (pre-commit) for review and do something else

Formal Code Review

Formal code review (sometimes called [Fagan inspection](#)) is heavyweight - involves an elaborate *team* review process

Mandated for safety-critical software, or any [software that must work right the first time and every time](#)

Rarely used for conventional business and consumer software other than for training purposes



Pair Programming and Code Review for Your Projects

We strongly recommend (but do not require) pair programming for your team projects. This is why four-member teams are preferred to five-member teams, since triplet programming does not work very well - but five member teams can trade off pairs



We strongly recommend (but do not require) lightweight code review for your team projects. Please ask some other member of your team to read your code (or your pair's code) before committing to the main trunk in your github repository, even if it passes all tests and style checking

Upcoming Assignments

[First Iteration](#) due October 24

[First Iteration Demo](#) due October 31

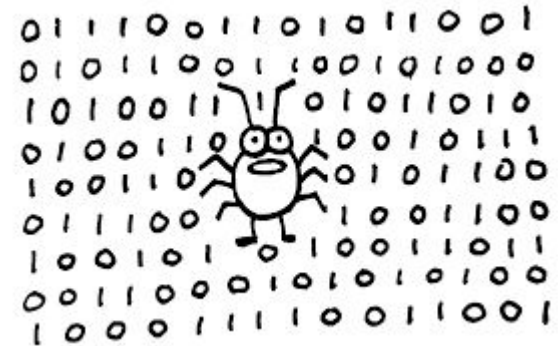
- You can keep coding and testing between first iteration submission and first iteration demo, but [tag](#) both separately! Also see [github releases](#)



Next Class

multi-client API Testing demo

Finding Bugs: beyond testing



Ask Me Anything