EFFECTIVE CODE REVIEW

Code review is a key part of the software engineering process. I'm going to be talking to you today about making code reviews as effective as possible.

JUSTIN SPAHR-SUMMERS

@JSPAHRSUMMERS

Let me introduce myself. My name is Justin Spahr-Summers, a.k.a., @jspahrsummers on basically every platform.

I've worked at smaller companies and larger ones, and been a key contributor to several successful open source projects in the Cocoa community, including Carthage, ReactiveCocoa, the Squirrel update framework, and the Mantle model framework for Objective-C.

In total, I've done thousands of code reviews—possibly even *tens* of thousands—in both commercial and open source contexts.

CODE REVIEW PUTS THE ENGINEERING INTO SOFTWARE ENGINEERING

Why this talk?

Code review is *fundamental* to our discipline. Without code review (and without other key fundamentals like testing, or CS theory), we're just *programming*—not engineering. We elevate our craft, and hold each other to a higher standard, through peer reviews.

POOR QUALITY **REVIEWS...**

- WASTE EVERYONE'S TIME TODAY
- WASTE EVERYONE'S TIME IN THE FUTURE
- PROVIDE A FALSE SENSE OF SECURITY

Many (I daresay *most*) teams are already doing some kind of code review. But if it's not *effective*, what's the point?

If your reviews aren't preventing technical debt, aren't improving the end result, and don't provoke good conversation... it's just burning everyone's time for little benefit.

GREAT REVIEWS...

- MINIMIZE TECHNICAL DEBT
- IMPROVE THE ARCHITECTURE
- SHARE DOMAIN KNOWLEDGE
- PROVIDE TEACHING OPPORTUNITIES

Submitting a pull request should be the *start* of a conversation, not the end of one. Use code reviews to achieve all of these goals, and really dig deep into the essence of what you're trying to achieve. The result will benefit everyone!

I intentionally do not say that "great reviews prevent bugs." I think bugs should be *primarily* prevented through type systems, tests, etc., and not rely upon humans to catch before merging. But of course, some reviews will indeed catch some number of bugs.

FIRST PRINCIPLES

Before deciding on or describing any process, I like to identify the first principles. Process should follow principles, not the other way around.

For code review, these are...

GOOD CODE REVIEW
STARTS FROM THE SAME PERSPECTIVE AS
WRITING GOOD CODE

IF WE WERE TO WRITE THE BEST VERSION OF THIS, WHAT WOULD IT BE?

A great code reviewer is a great engineer, because being able to dig in and understanding someone else's code, as well as how to *improve* it, are required all the time in writing code too!

Note that it doesn't work the other way—not all great engineers are automatically great reviewers! It's an additional skill, and requires commitment to become good at.

UNBLOCK OTHERS

INCREASE YOUR TEAM'S PRODUCTIVITY

One of the most impactful things you can do is unblock your colleagues. PRs sitting open and unreviewed = lower productivity for everyone.

I recommend establishing a maximum turnaround time for code reviews (e.g., 24 hours during the working week). If you're having trouble context switching, or bouncing back and forth between reviews and your own coding, establish a few recurring review times as a habit—like before starting your day, after lunch, at the end of your day

CRITIQUE THE CODE (NOT THE PERSON)

You may have heard the expression, "you are not your code." I prefer this reversal of it.

The code that someone submits should not reflect upon them *as a person*. Code reviews should never make it personal. It should be about the problem we're trying to solve, and whether this particular version of the change is the best way to do it.

DON'T ASSUME IT'S OBVIOUS

CODE CHANGES AND FEEDBACK BOTH NEED EXPLANATION

Many, many important things get omitted in communication because someone assumed, "this is obvious, I don't need to say it."

Always state your assumptions, and overcommunicate. Code authors, explain what you're doing and why! Code reviewers, explain the feedback you give, and why you believe it might help!

THE PROCESS

TODO: Better linking before this slide?

Okay. Let me share the process that I use—the one I've honed over thousands of code reviews.

START AT THE HIGHEST LEVEL, THEN DIVE DEEPER...

1. **INTENT**
2. **DESIGN**
3. **BEHAVIOR**

1. INTENT

WHAT IS THE GOAL?

IS THE EXPLANATION CLEAR?

mplement `jest-haste-map` instead of `node-haste` #896

Closed **cpojer** wants to merge 10 commits into `facebook:master` from `cpojer:jest-haste-map`

Conversation 93 | Commits 10 | Checks 0 | Files changed 47

cpojer commented on Apr 15, 2016 • edited ▾ Collaborator

This is a new haste map implementation for Jest which is much more scalable than node-haste.

node-haste2 isn't well designed and not scalable for short-lived services like Jest. The startup time of node-haste1 vs. node-haste2 on www is almost the same, both between 6 and 8 seconds which is not acceptable for our engineers. This implementation is attempting to accomplish a much reduced and more scalable startup time. It also has reduced scope – the goal of this is to only build a haste map and provide a way to resolve a one-level deep dependency tree, which is all that Jest really needs from node-haste. `jest-haste-map` can serve as an ideal basis for rewriting node-haste2 (into node-haste3!).

With this, the cold start time (building the entire haste map) is now about 14 seconds on www (that's acceptable) but the incremental invocation is only 2 seconds (**@kyldvs** will love me). I haven't heavily micro-optimized the JavaScript in `packages/jest-haste-map/src/index.js` and there is a bunch of data copying with `HasteResolver.js` – I believe I can get close to 1 second with these optimizations once I'm done. One of the goals I have is to allow tacking on data (list of mocks) to the haste map and serialize the haste map and read that one in directly in the workers instead of keeping two caches.

https://github.com/facebook/jest/pull/896

mplement `jest-haste-map` instead of `node-haste` #896

Closed **cpojer** wants to merge 10 commits into `facebook:master` from `cpojer:jest-haste-map`

Conversation 93 | Commits 10 | Checks 0 | Files changed 47

cpojer commented on Apr 15, 2016 • edited ▾ — Collaborator

This is a new haste map implementation for Jest which is much more scalable than node-haste.

node-haste2 isn't well designed and not scalable for short-lived services like Jest. The startup time of node-haste1 vs. node-haste2 on www is almost the same, both between 6 and 8 seconds which is not acceptable for our engineers. This implementation is attempting to accomplish a much reduced and more scalable startup time. It also has reduced scope – the goal of this is to only build a haste map and provide a way to resolve a one-level deep dependency tree, which is all that Jest really needs from node-haste. `jest-haste-map` can serve as an ideal basis for rewriting node-haste2 (into node-haste3!).

With this, the cold start time (building the entire haste map) is now about 14 seconds on www (that's acceptable) but the incremental invocation is only 2 seconds (**@kyldvs** will love me). I haven't heavily micro-optimized the JavaScript in `packages/jest-haste-map/src/index.js` and there is a bunch of data copying with `HasteResolver.js` – I believe I can get close to 1 second with these optimizations once I'm done. One of the goals I have is to allow tacking on data (list of mocks) to the haste map and serialize the haste map and read that one in directly in the workers instead of keeping two caches.

https://github.com/facebook/jest/pull/896

mplement `jest-haste-map` instead of `node-haste` #896

Closed **cpojer** wants to merge 10 commits into `facebook:master` from `cpojer:jest-haste-map`

Conversation 93 | Commits 10 | Checks 0 | Files changed 47

cpojer commented on Apr 15, 2016 • edited ▾ — Collaborator

This is a new haste map implementation for Jest which is much more scalable than node-haste.

node-haste2 isn't well designed and not scalable for short-lived services like Jest. The startup time of node-haste1 vs. node-haste2 on www is almost the same, both between 6 and 8 seconds which is not acceptable for our engineers. This implementation is attempting to accomplish a much reduced and more scalable startup time. It also has reduced scope – the goal of this is to only build a haste map and provide a way to resolve a one-level deep dependency tree, which is all that Jest really needs from node-haste. `jest-haste-map` can serve as an ideal basis for rewriting node-haste2 (into node-haste3!).

With this, the cold start time (building the entire haste map) is now about 14 seconds on www (that's acceptable) but the incremental invocation is only 2 seconds (**@kyldvs** will love me). I haven't heavily micro-optimized the JavaScript in `packages/jest-haste-map/src/index.js` and there is a bunch of data copying with `HasteResolver.js` – I believe I can get close to 1 second with these optimizations once I'm done. One of the goals I have is to allow tacking on data (list of mocks) to the haste map and serialize the haste map and read that one in directly in the workers instead of keeping two caches.

https://github.com/facebook/jest/pull/896

DOES IT SUCCEED?

HOW DO YOU KNOW?

Is it tested? How will you know if it regresses?

nitial version of jest-worker #4497

Merged cpojer merged 8 commits into `facebook:master` from `mjesun:jest-parallel` on Oct 4, 2017

Conversation 85 | Commits 8 | Checks 0 | Files changed 14

mjesun commented on Sep 17, 2017 • edited ▾ Contributor

This PR introduces a new module, `jest-worker`, intended to allow heavy task parallelization over multiple workers.

The module has a few advantages over the currently one used both in `jest` and `metro-bundler`:

- 100% `flow`-ified.
- 100% test coverage on it, all statements, methods and branches.
- Slightly faster than the currenly used one.
- Natively provides a `Promise` based interface, which allow us to avoid the extra wrapping layer in order to be used with `async`/`await.
- It only has one single dependency (`merge-stream`), which we could also remove.

https://github.com/facebook/jest/pull/4497

Performance test

It can be run by doing `node --expose-gc test.js` under `__performance_tests__`. Note that the percentage improvement shown (~ 10%) applies to 10,000 calls, meaning the performance improvement per single call is negligible. The test implements a `Promise` wrapper over the current implementation, so we can equivalently test both implementations as we use them in real scenarios.

```
------------------------------------------------------------------
jest-worker: { globalTime: 738, processingTime: 707 }
worker-farm: { globalTime: 885, processingTime: 866 }
------------------------------------------------------------------
jest-worker: { globalTime: 738, processingTime: 718 }
worker-farm: { globalTime: 865, processingTime: 849 }
------------------------------------------------------------------
jest-worker: { globalTime: 708, processingTime: 685 }
```

https://github.com/facebook/jest/pull/4497

Coverage

File ▲		Statements		Branches		Functions		Lines	
child.js		100%	27/27	100%	20/20	100%	4/4	100%	27/27
index.js		100%	67/67	100%	29/29	100%	11/11	100%	67/67
types.js		100%	5/5	100%	0/0	100%	0/0	100%	5/5
worker.js		100%	44/44	100%	13/13	100%	8/8	100%	44/44

https://github.com/facebook/jest/pull/4497

A GOOD SUMMARY SHOULD INCLUDE...

- BACKGROUND CONTEXT
- WHAT THE BUG OR FEATURE IS
- WHAT THE CHANGE ACHIEVES
- HOW THE CHANGE IS TESTED
- KNOWN LIMITATIONS OR ANYTHING STILL MISSING
- REQUEST CHANGES IF THE SUMMARY IS INCOMPLETE!

Allow creating an alias for repositories #12000

Merged sergiou87 merged 8 commits into `development` from `repository-alias` 25 days ago

Conversation 12 | Commits 8 | Checks 6 | Files changed 13

sergiou87 commented 26 days ago — Member

This is part of #7856

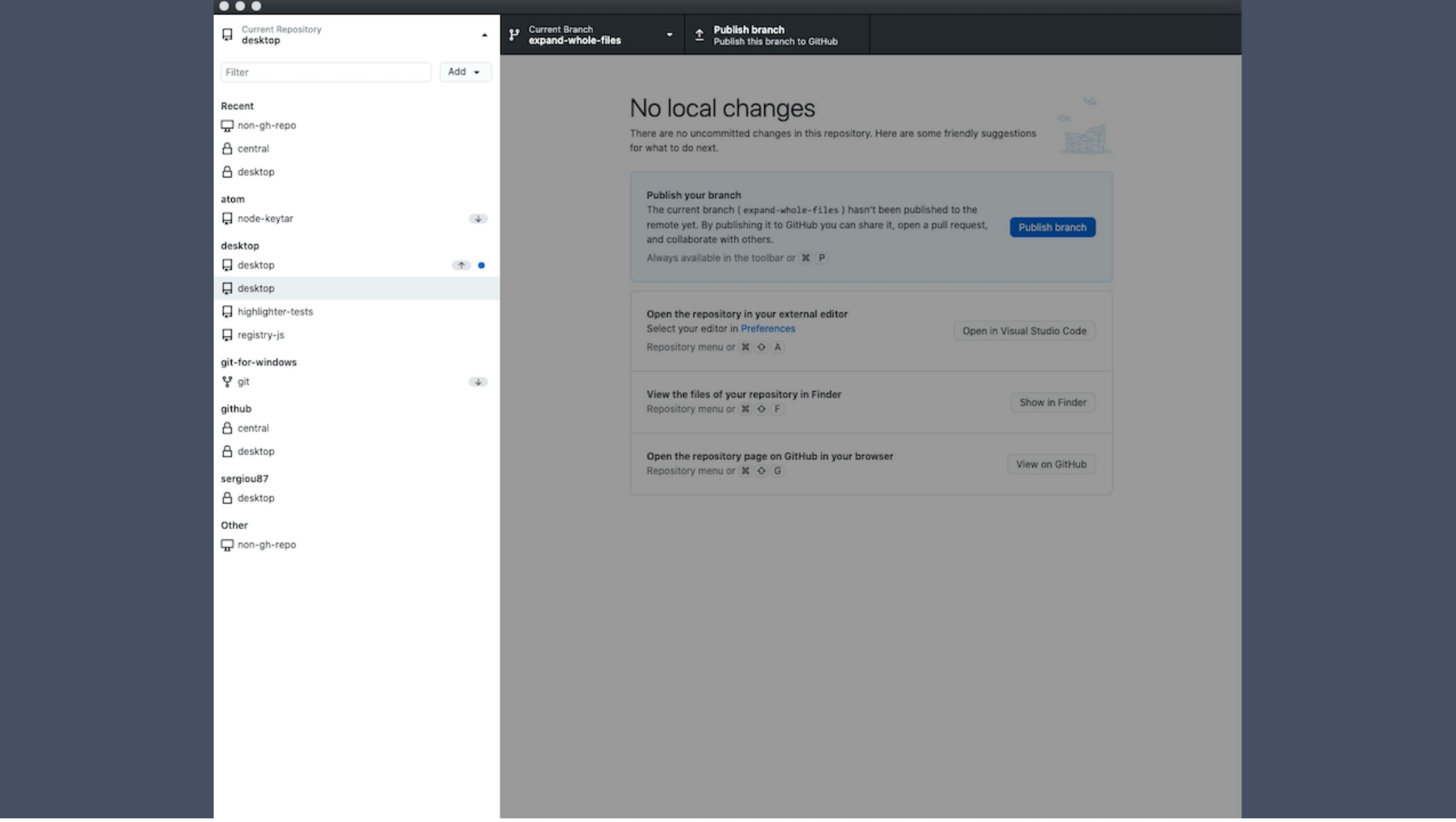
Description

This PR adds initial support to create alias for repositories:

- New context menu actions to create, change or remove the alias of a repository from the repository list. Any character is valid.
- Filtering repositories now takes the alias into account.
- The original repository name is still visible by hovering over it, in the tooltip.

Screenshots

https://github.com/desktop/desktop/pull/12000



https://github.com/desktop/desktop/pull/12000

ARE THE CHANGES APPROPRIATE?

Do they align with the explanation in the summary? Are major architectural shifts explained? Do other changes depend on this one? Even incomplete PRs should meet a high quality bar.

2. DESIGN

ASK YOURSELF:
HOW WOULD YOU DO IT?

Try to think through the bug or feature described, and mentally design your own solution for it.

^ If you didn't have this PR in front of you, how would you have done it? Does that highlight any gaps in the code you're reviewing?

REVIEW THE ARCHITECTURE

(THE COMPONENTS AND HOW THEY RELATE TO ONE ANOTHER)

- DO YOU UNDERSTAND IT WELL ENOUGH TO USE OR EXTEND?
  - ARE THE ARCHITECTURAL CHOICES JUSTIFIED?
  - WOULD EVERYONE ELSE BE HAPPY TO MAINTAIN THIS?

REVIEW THE API

(THE CONTRACT FOR USING EACH COMPONENT)

- IS THE API UNDERSTANDABLE WITHOUT THE PR?
- DOES THE DOCUMENTATION TEACH THE READER HOW TO USE IT?
- IS THE API CONVENTIONAL?

IS THE DESIGN GOOD?

YAGNI

YOU AREN'T GONNA NEED IT

SIMPLE VS. EASY

PIT OF SUCCESS

MAKE THE RIGHT THINGS EASY & THE WRONG THINGS POSSIBLE

SOLID PRINCIPLES

- **SINGLE-RESPONSIBILITY PRINCIPLE**
  EACH THING SHOULD HAVE ONLY ONE RESPONSIBILITY
- **OPEN-CLOSED PRINCIPLE**
  BEHAVIOR SHOULD BE EXTENSIBLE WITHOUT MODIFYING CODE
- **LISKOV SUBSTITUTION PRINCIPLE**
  TYPES SHOULD BE REPLACEABLE WITH SUBTYPES
- **INTERFACE SEGREGATION PRINCIPLE**
  MANY SPECIFIC INTERFACES ARE BETTER THAN ONE ÜBER-INTERFACE
- **DEPENDENCY INVERSION PRINCIPLE**
  DEPEND UPON ABSTRACTIONS, NOT CONCRETE IMPLEMENTATIONS

3. BEHAVIOR

REVIEW THE TESTS

- TESTS SHOULD BE DOCUMENTATION
- TESTS SHOULD PROTECT AGAINST REGRESSIONS
- TESTS SHOULD VALIDATE THE API CONTRACT
- TESTS NEED TO BE UNDERSTANDABLE
- ARE THERE MISSING TESTS?
- YOU CAN REQUEST CHANGES!

REVIEW THE IMPLEMENTATION

- THIS IS THE LEAST IMPORTANT PART TO REVIEW!
  - WOULD YOU BE ABLE TO DEBUG THIS CODE?
    - DRY: DON'T REPEAT YOURSELF
      - DON'T REINVENT THE WHEEL
    - DON'T IGNORE LINTERS AND WARNINGS
  - IF IT LOOKS CONVOLUTED, IT'S PROBABLY WRONG

Does the code achieve its goal in the simplest, most maintainable way possible?

BUT REMEMBER...

1. **INTENT**
2. **DESIGN**
3. **BEHAVIOR**

OTHER PROTIPS™

GIVE EFFECTIVE FEEDBACK

- ALWAYS REQUEST CHANGES OR ACCEPT
- BE PRAGMATIC FOR URGENT CHANGES
- IS EACH STAGE OF REVIEW IMPORTANT RIGHT NOW?
- SOLICIT SECOND OPINIONS
- PRIORITIZE YOUR FEEDBACK
- PROVIDE CONCRETE SUGGESTIONS

TELL A STORY WITH YOUR COMMITS

- EACH COMMIT SHOULD BUILD LOGICALLY UPON THE PREVIOUS
  - CLEAN UP AFTER YOURSELF:

```
git rebase -i
hg histedit
```

- STACK DEPENDENT CHANGES

QUESTIONS?

SLIDES AND NOTES ARE AVAILABLE AT:
GITHUB.COM/JSPAHRSUMMERS/EFFECTIVE-CODE-REVIEW

THANKS TO LIGHTRICKS AND BARAK YORESH FOR INVITING ME TO SPEAK!