ANDREW LAVERS

Are your interns reviewing code?

CONFOO MONTREAL 2020

FEB 26, 2020

zendesk

What do you look for in a job?

Salary? Interesting projects? Location? People & culture?

What do you look for in a job?

Salary? Interesting projects? Location? **People & culture?**

There’s more to code review than reviewing code

ANDREW LAVERS

About me

(*in other words:* why should anyone care about anything I have to say)

- in 15 years I've worked on software as:
  - intern
  - team lead
  - senior developer
  - tech lead
- both large & small companies (smallest: **16** people, largest: **350,000**)
- every team had a different code review culture
- some worked better than others

2005

matrox

LOCKHEED MARTIN

IBM

Microsoft

radialpoint

smooch

2020

zendesk

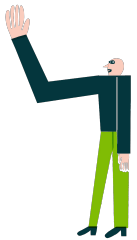
WE'RE HIRING

Now, About *you*

What is your team's code review culture?

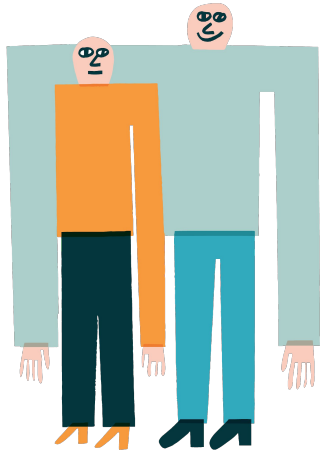
0) No code review

You're either a one-person team, or a team who trust each other so much they don't bother looking at each other's code.



1) Ivory tower

Code review is the solemn duty of the ***Council of Senior Reviewers***. Beware their wrath.



2) Neglect

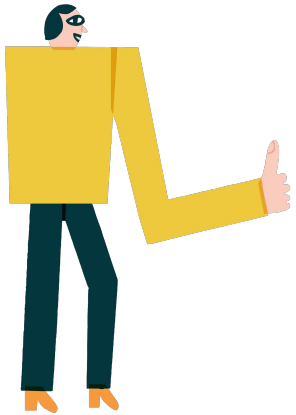
It takes 30 days before anyone looks at your pull requests.

3) Battleground

Code review is an arena where devs take the opportunity to shit on new ideas and show off their own brilliance (aka insecurity)

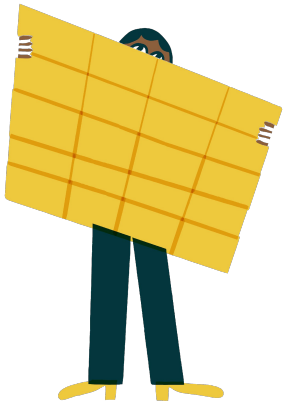
4) Rubber stamp

You posted a reminder about your open PR in Slack and got two approvals ✔️30 seconds later. It's magic!



5) Bike shed

Code review is where you labor over every detail. Instead of designing the power plant, you spend all of your time deciding what color to paint bike shed out front.



WHICH ONE MOST RESEMBLES YOUR OWN TEAM?

Is there a better way?

0) No code review

You're either a one person team, or a team who trust each other so much they don't bother looking at each other's code.

1) Ivory tower

Code review is the solemn duty of the ***Council of Senior Reviewers***. Beware their wrath.

2) Neglect

It takes 30 days before anyone looks at your pull requests.

3) Battleground

Code review is an arena where devs take the opportunity to shit on new ideas and show off their own brilliance
(aka *insecurity*)

4) Rubber stamp

I posted a reminder about my open PR in Slack and I got two ✔️30 seconds later. It's magic!

5) Bike shed

Code review is where you labor over every detail. Instead of designing the power plant, you spend all of your time deciding what color to paint bike shed out front.

zendesk

what are the outcomes of a healthy code review process?

6) Healthy

A code review process that actually works.

The purpose of code review

zendesk

Detect faults

Learn the codebase

Learn how to write code

Detect faults?

Obviously

But maybe it's not so obvious…

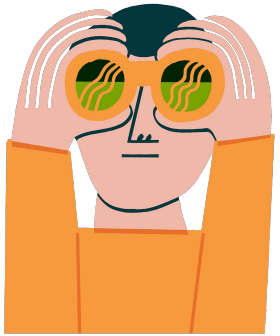
Does your team rely on code review prevent bugs from getting shipped?

Learn the codebase

Code is a jungle: it tends to change over time. Reviewing those changes regularly keeps your mental waypoints up to date.

It's much easier to do this incrementally.

Waypoints speed up your ability to contribute new code and inspire intelligent refactorings



Learn how to write code

The only way to learn how to review code is to actually do it.

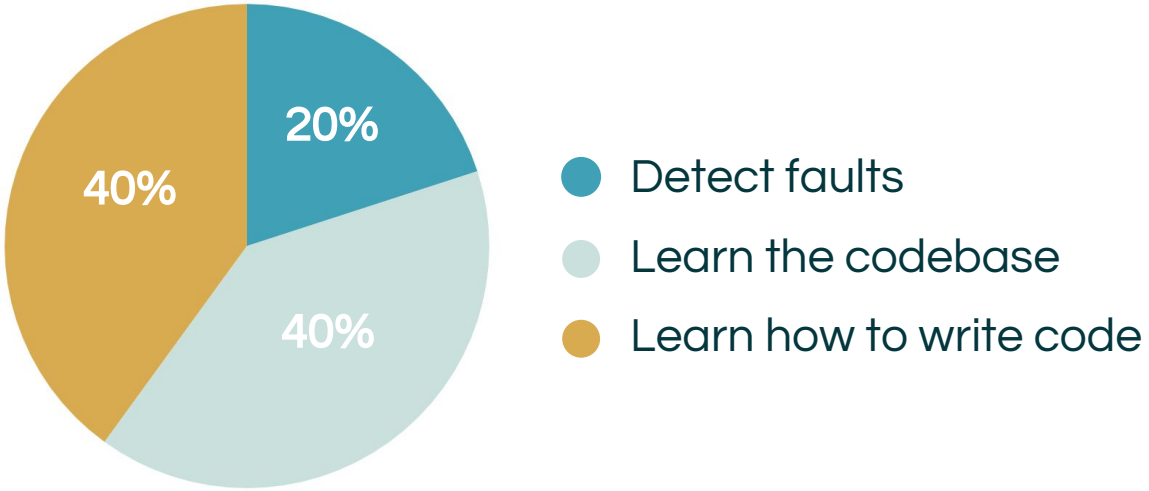
Your peers are your teachers, regardless of rank.

You'll see fresh ideas come from interns as often as senior devs.

Pro tip:

Are you a senior developer? Then you should *never* let your juniors review code. That way you can stunt their growth and forever stay on top 😈

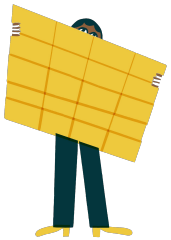
The most tangible benefits of code review



What are the outcomes of a healthy code review culture?

6) Healthy

A code review process that actually works.



First let's talk about some code review worst outcomes

1) Approval gates

You want to make sure every PR is reviewed by a **senior developer**
(with the best of intentions of course)

You've just created an approval gate

Yuck

Worse still, by putting a small group of senior devs on every critical path you've created a *delivery bottleneck*

1) Approval gates

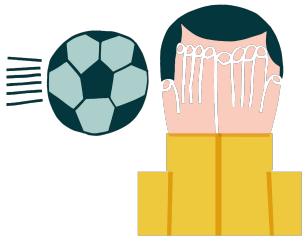
Mission critical changes deserve a close inspection

But if you require senior approval on every change you're making a serious tradeoff on delivery throughput, and to some extent, individual accountability

2) Dilution of accountability

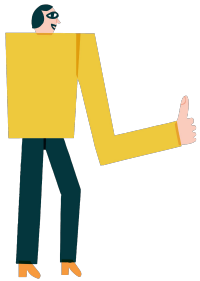
Has your team ever shipped a bug?

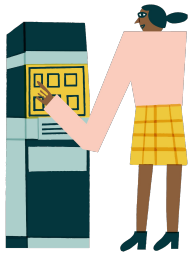
Have you ever been left wondering *“who approved this change”*?



2) Dilution of accountability

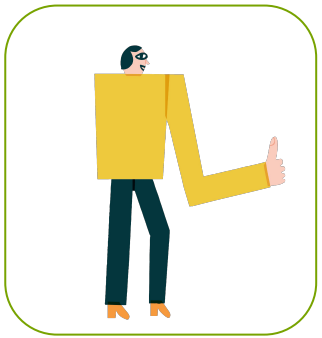
Who's accountable for the success of a pull request?



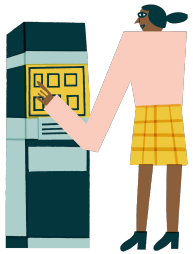


2) Dilution of accountability

Who's accountable for the success of a pull request?

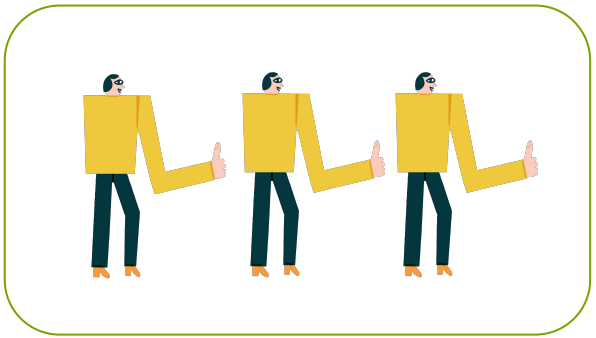


The approver?

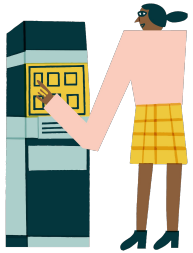


2) Dilution of accountability

Who's accountable for the success of a pull request?

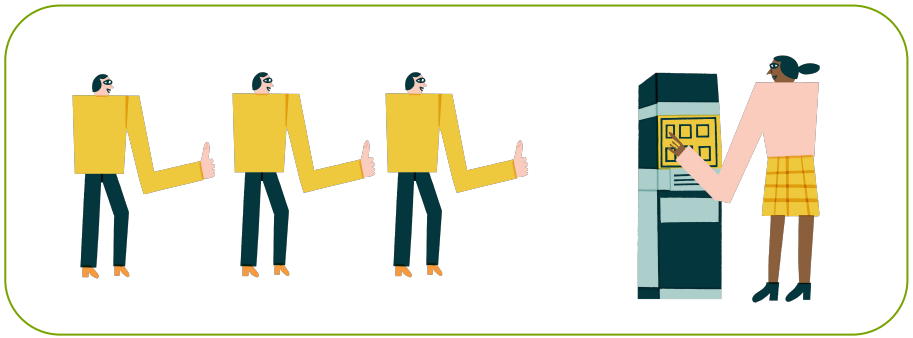


Suppose there are multiple approvers?



2) Dilution of accountability

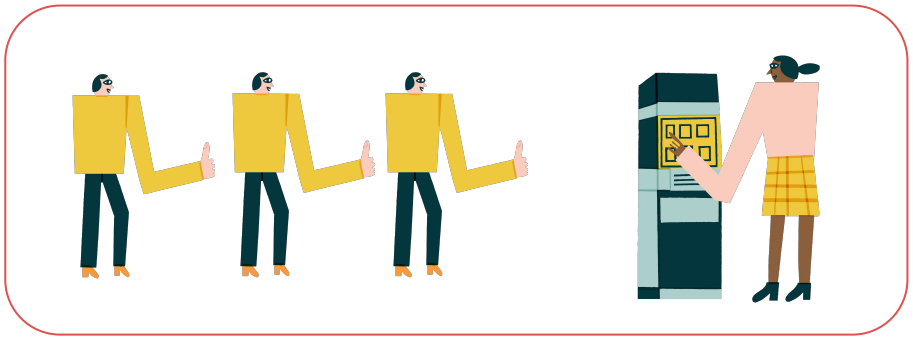
Who's accountable for the success of a pull request?



Do they all share accountability?

2) Dilution of accountability

Who's accountable for the success of a pull request?



Shared accountability = no accountability

2) Dilution of accountability

Suppose the approver is accountable to the success of the change

They'll be very opinionated (as they should be)

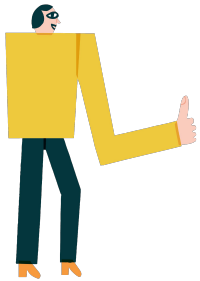
In every comment they make they'll expect to have it their way

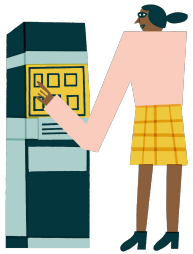
At some point you might as well have the approver implement the change themselves

Which would make them the author...

2) Dilution of accountability

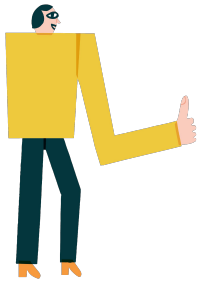
So who’s ultimately accountable for the success of a pull request?

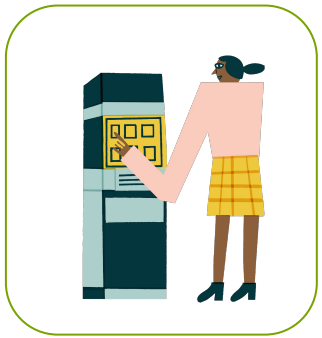




2) Dilution of accountability

So who's ultimately accountable for the success of a pull request?

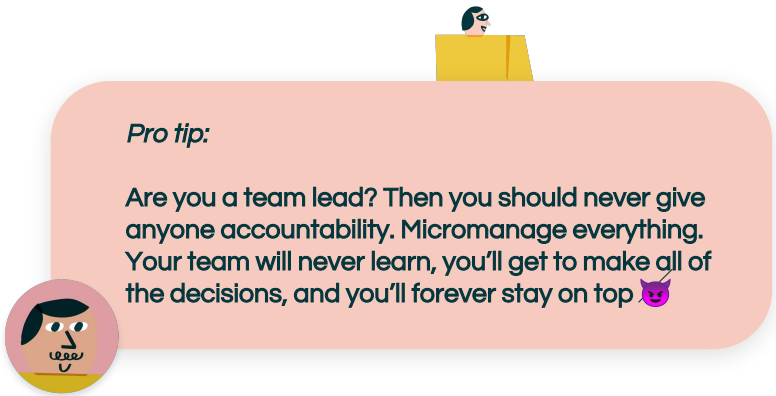


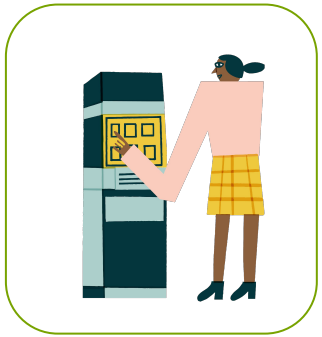


The author

2) Dilution of accountability

So who's ultimately accountable for the success of a pull request?
Approvers play a crucial role, but you must make it clear who is individually accountable





The author

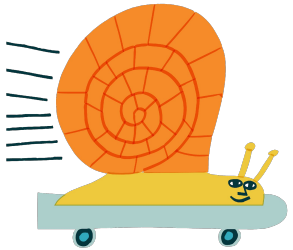
3) Wasting time

Using code review to argue about code style and whitespace

Example: JavaScript developers: you for or against semicolons?

You're both wrong

Automate these debates into oblivion with linters and formatters



4) Erosion of trust

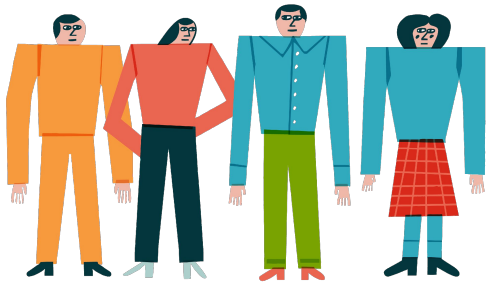
As an author, does having your mistakes pointed out in front of everyone feel like ridicule?

As a reviewer, has a comment of yours ever gone misunderstood? Or ignored?

Code review is a forum for disagreement. Discomfort is healthy. Resentment however is *disastrous*

It simply doesn't work without trust

Your code review culture is your team culture



Maslow's hierarchy of **individual** needs

Abraham Maslow *A Theory of Human Motivation* 1943

Lencioni's hierarchy of **team** needs

Patrick Lencioni *The Five Dysfunctions of a Team* 2002

code review *requires*
constructive conflict
without trust it's a turf war

Lencioni's hierarchy of **team** needs

Patrick Lencioni *The Five Dysfunctions of a Team* 2002

👉 the reviewer and the author don't have to agree, but they do have to *align*

Lencioni's hierarchy of **team** needs

Patrick Lencioni *The Five Dysfunctions of a Team* 2002

👉 the author has final say

👉

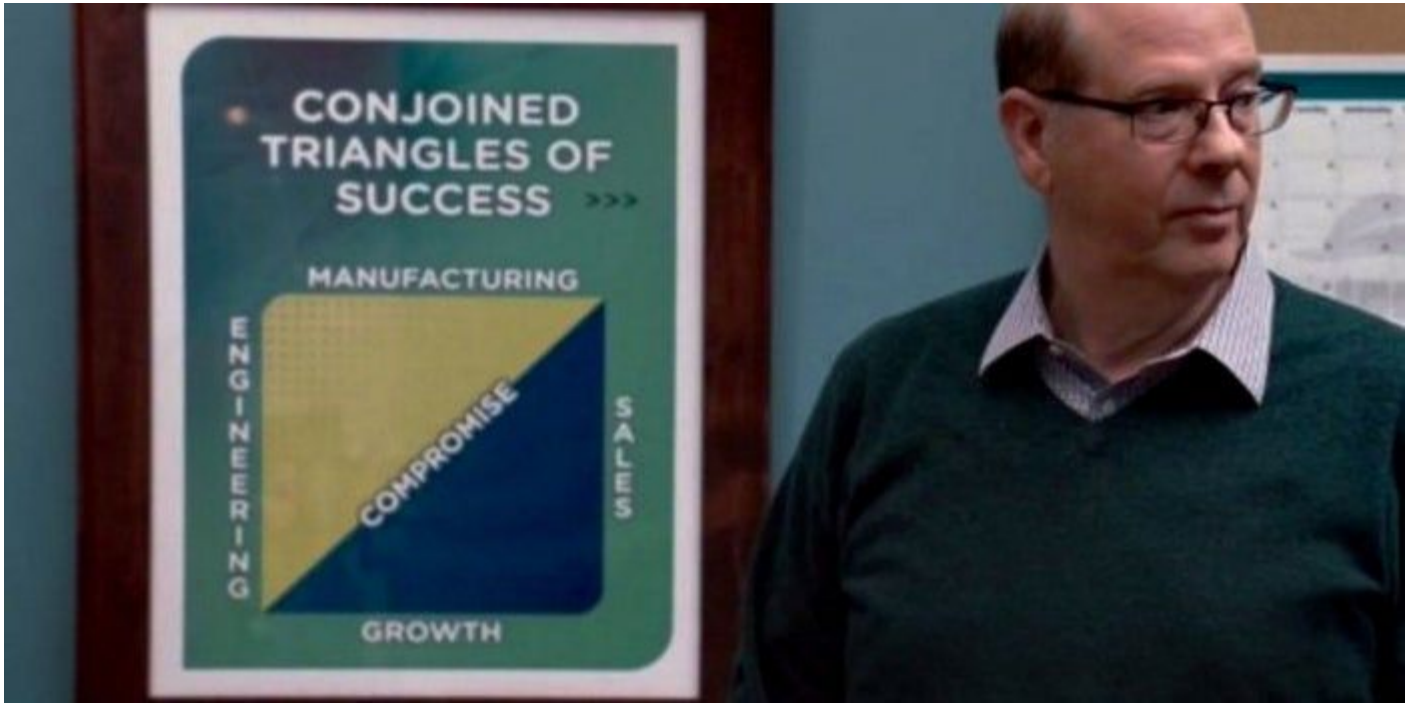
Lencioni's hierarchy of **team** needs

Patrick Lencioni *The Five Dysfunctions of a Team* 2002

👉 author's accountability doesn't end when the pull request is merged

Lencioni's hierarchy of **team** needs

Patrick Lencioni *The Five Dysfunctions of a Team* 2002

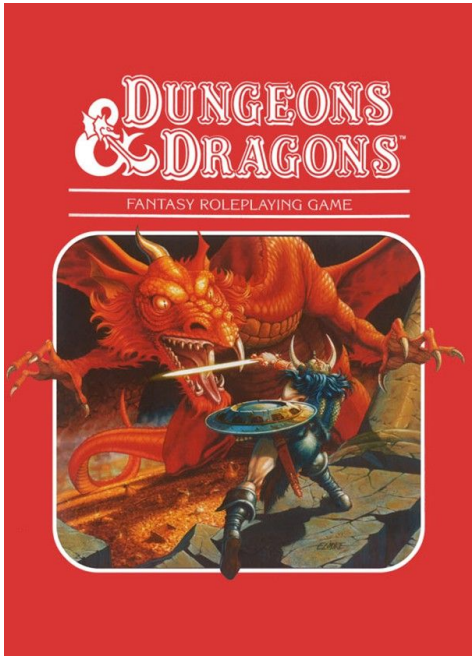

CONJOINED TRIANGLES OF SUCCESS >>>
MANUFACTURING
ENGINEERING
COMPROMISE
SALES
GROWTH

Here are some review guidelines from my team

Rule 0:

The DM can override published game rules for any reason

Meaning: **rules aren't an excuse to stop thinking**



Zendesk Montreal Code Review Guidelines:

Submitting a PR:

1. Always review a PR diff before asking others to take a look. The diff view GitHub gives you before you submit a PR gives you a nice opportunity to do this. As a rule of thumb, I'll catch and fix things in my own code at this step about 75% of the time. (stray TODO comments, residual code, etc)
2. Add roughly 2-5 reviewers to your PRs by adding them as reviewers. Use your best judgement to add relevant reviewers based on who else is working on similar features and who has expertise in a certain area. If you're not sure, simply pick reviewers at random.
3. **Always** include an explanation of what you did in your PR description. For example, a list of what's included, and if it's useful, any links to Jira cards (formerly Trello) or design specifications. You don't have to write an essay, but your description should provide the reviewer a map guide through your changes. If someone new to the team were to look at your merged PR 1 year later, your brief PR description should give them a good idea of why the change was necessary.
4. Now that Atlassian is being used, use the issue key from the Jira issues and include it somewhere, either in the pull request title, or in the commit message, more info can be found here: https://confluence.atlassian.com/jirasoftwarecloud/referencing-issues-in-your-development-work-777002789.html
5. If no one has looked at your code in 24 hours, post a link in Slack. If it's urgent, post it immediately.
6. Address reviewer's comments promptly. You should move on to work on something else while your code is reviewed, but don't let more than a day go by without responding to questions or addressing a suggestion.
7. Keep your code diffs clean and consumable. Don't keep tacking on changes to the same PR. If you need to, merge early, or create a new branch off of your main feature branch and show your code diff in a separate PR. Sometimes it's OK to merge first and address review comments in a follow-up PR.
8. Keep your PRs small. The rule of thumb is if you've worked on a branch for 3 days, you should find a sensible cut-off point and open a PR, or follow the feature branch model. Make it clear in your PR description that there's more to come.
9. Seek approvals before merging. For sizable changes you should seek 2 approvals (👍). For small config & style changes, 1 is sufficient. If you have a very good reason (eg. an urgent fix is needed and no one else is around) it's OK to merge a small change first and ask colleagues to review the merged PR afterwards.
10. Disagreement is OK (and expected). As the author, you can choose not to accept a reviewer's suggestion, but you must explain why. Try to reach a consensus.

Zendesk Montreal Code Review Guidelines:

1. Review your own PR diff before asking others to take a look.
2. Add roughly 2-5 reviewers to your PRs by adding them as reviewers. Use your best judgement to add relevant reviewers based on who else is working on similar features and who has expertise in a certain area. If you're not sure, simply pick reviewers at random.
3. **Always** include an explanation of what you did in your PR description. For example, a list of what's included, and if it's useful, any links to Jira cards (formerly Trello) or design specifications. You don't have to write an essay, but your description should provide the reviewer a map guide through your changes. If someone new to the team were to look at your merged PR 1 year later, your brief PR description should give them a good idea of why the change was necessary.
4. Now that Atlassian is being used, use the issue key from the Jira issues and include it somewhere, either in the pull request title, or in the commit message, more info can be found here: https://confluence.atlassian.com/jirasoftwarecloud/referencing-issues-in-your-development-work-777002789.html
5. If no one has looked at your code in 24 hours, post a link in Slack. If it's urgent, post it immediately.
6. Address reviewer's comments promptly. You should move on to work on something else while your code is reviewed, but don't let more than a day go by without responding to questions or addressing a suggestion.
7. Keep your code diffs clean and consumable. Don't keep tacking on changes to the same PR. If you need to, merge early, or create a new branch off of your main feature branch and show your code diff in a separate PR. Sometimes it's OK to merge first and address review comments in a follow-up PR.
8. Keep your PRs small. The rule of thumb is if you've worked on a branch for 3 days, you should find a sensible cut-off point and open a PR, or follow the feature branch model. Make it clear in your PR description that there's more to come.
9. Seek approvals before merging. For sizable changes you should seek 2 approvals (👍). For small config & style changes, 1 is sufficient. If you have a very good reason (eg. an urgent fix is needed and no one else is around) it's OK to merge a small change first and ask colleagues to review the merged PR afterwards.
10. Disagreement is OK (and expected). As the author, you can choose not to accept a reviewer's suggestion, but you must explain why. Try to reach a consensus.

Zendesk Montreal Code Review Guidelines:

Submitting a PR:

1. Always review a PR diff before asking others to take a look. The diff view GitHub gives you before you submit a PR gives you a nice opportunity to do this. As a rule of thumb, I'll catch and fix things in my own code at this step about 75% of the time. (stray TODO comments, residual code, etc)
2. Add roughly 2-5 reviewers to your PRs by adding them as reviewers. Use your best judgement to add relevant reviewers based on who else is working on similar features and who has expertise in a certain area. If you're not sure, simply pick reviewers at random.
3. **Always** include an explanation of what you did in your PR description. For example, a list of what's included, and if it's useful, any links to Jira cards (formerly Trello) or design specifications. You don't have to write an essay, but your description should provide the reviewer a map guide through your changes. If someone new to the team were to look at your merged PR 1 year later, your brief PR description should give them a good idea of why the change was necessary.
4. Now that Atlassian is being used, use the issue key from the Jira issues and include it somewhere, either in the pull request title, or in the commit message, more info can be found here:
   https://confluence.atlassian.com/jirasoftwarecloud/referencing-issues-in-your-development-work-777002789.html

6. Address reviewer's comments promptly.

   responding to questions or addressing a suggestion.
7. Keep your code diffs clean and consumable. Don't keep tacking on changes to the same PR. If you need to, merge early, or create a new branch off of your main feature branch and show your code diff in a separate PR. Sometimes it's OK to merge first and address review comments in a follow-up PR.
8. Keep your PRs small. The rule of thumb is if you've worked on a branch for 3 days, you should find a sensible cut-off point and open a PR, or follow the feature branch model. Make it clear in your PR description that there's more to come.
9. Seek approvals before merging. For sizable changes you should seek 2 approvals (👍). For small config & style changes, 1 is sufficient. If you have a very good reason (eg. an urgent fix is needed and no one else is around) it's OK to merge a small change first and ask colleagues to review the merged PR afterwards.
10. Disagreement is OK (and expected). As the author, you can choose not to accept a reviewer's suggestion, but you must explain why. Try to reach a consensus.

Zendesk Montreal Code Review Guidelines:

Submitting a PR:

1. Always review a PR diff before asking others to take a look. The diff view GitHub gives you before you submit a PR gives you a nice opportunity to do this. As a rule of thumb, I'll catch and fix things in my own code at this step about 75% of the time. (stray TODO comments, residual code, etc)
2. Add roughly 2-5 reviewers to your PRs by adding them as reviewers. Use your best judgement to add relevant reviewers based on who else is working on similar features and who has expertise in a certain area. If you're not sure, simply pick reviewers at random.
3. **Always** include an explanation of what you did in your PR description. For example, a list of what's included, and if it's useful, any links to Jira cards (formerly Trello) or design specifications. You don't have to write an essay, but your description should provide the reviewer a map guide through your changes. If someone new to the team were to look at your merged PR 1 year later, your brief PR description should give them a good idea of why the change was necessary.
4. Now that Atlassian is being used, use the issue key from the Jira issues and include it somewhere, either in the pull request title, or in the commit message, more info can be found here: https://confluence.atlassian.com/jirasoftwarecloud/referencing-issues-in-your-development-work-777002789.html
5. If no one has looked at your code in 24 hours, post a link in Slack. If it's urgent, post it immediately.
6. Address reviewer's comments promptly. You should move on to work on something else while your code is reviewed, but don't let more than a day go by without

7. Keep your code diffs clean and consumable.

8. Keep your PRs small. The rule of thumb is if you've worked on a branch for 3 days, you should find a sensible cut-off point and open a PR, or follow the feature branch model. Make it clear in your PR description that there's more to come.
9. Seek approvals before merging. For sizable changes you should seek 2 approvals (👍). For small config & style changes, 1 is sufficient. If you have a very good reason (eg. an urgent fix is needed and no one else is around) it's OK to merge a small change first and ask colleagues to review the merged PR afterwards.
10. Disagreement is OK (and expected). As the author, you can choose not to accept a reviewer's suggestion, but you must explain why. Try to reach a consensus.

Zendesk Montreal Code Review Guidelines:

Submitting a PR:

1. Always review a PR diff before asking others to take a look. The diff view GitHub gives you before you submit a PR gives you a nice opportunity to do this. As a rule of thumb, I'll catch and fix things in my own code at this step about 75% of the time. (stray TODO comments, residual code, etc)
2. Add roughly 2-5 reviewers to your PRs by adding them as reviewers. Use your best judgement to add relevant reviewers based on who else is working on similar features and who has expertise in a certain area. If you're not sure, simply pick reviewers at random.
3. **Always** include an explanation of what you did in your PR description. For example, a list of what's included, and if it's useful, any links to Jira cards (formerly Trello) or design specifications. You don't have to write an essay, but your description should provide the reviewer a map guide through your changes. If someone new to the team were to look at your merged PR 1 year later, your brief PR description should give them a good idea of why the change was necessary.
4. Now that Atlassian is being used, use the issue key from the Jira issues and include it somewhere, either in the pull request title, or in the commit message, more info can be found here: https://confluence.atlassian.com/jirasoftwarecloud/referencing-issues-in-your-development-work-777002789.html
5. If no one has looked at your code in 24 hours, post a link in Slack. If it's urgent, post it immediately.
6. Address reviewer's comments promptly. You should move on to work on something else while your code is reviewed, but don't let more than a day go by without responding to questions or addressing a suggestion.
7. Keep your code diffs clean and consumable. Don't keep tacking on changes to the same PR. If you need to, merge early, or create a new branch off of your main feature branch and show your code diff in a separate PR. Sometimes it's OK to merge first and address review comments in a follow-up PR.
8. Keep your PRs small. The rule of thumb is if you've worked on a branch for 3 days, you should find a sensible cut-off point and open a PR, or follow the feature branch model. Make it clear in your PR description that there's more to come.

10. Disagreement is OK (and expected). As the author, you can choose not to accept a reviewer's suggestion, but you must explain why. Try to reach a consensus.

Zendesk Montreal Code Review Guidelines:

Reviewing PRs:

1. **Everyone** reviews code. Especially new hires and interns.
2. The time you spend reviewing code will vary from day to day, but about 1 hour a day is normal.
3. Add a 👍 comment once you're done. In some cases may want to wait until you see how the author responds to or addresses your comments before you give the 👍, but it's also OK to say "Rest looks good" and trust the author to respond to your comments appropriately.
4. You might not have time to look at every PR you were @mentioned in, but you should do what you can.
5. Don't skip over code you don't understand. If you don't understand a block of code, ask the author how it works. One of two things will happen: you'll uncover a bad design -OR- you'll learn something new. Win-win.
6. Don't argue about white space and bracket style. That's what we use Prettier for.
7. It's not a bad idea to take a quick look at a PR even after it's been merged.

Zendesk Montreal Code Review Guidelines:

Reviewing PRs:

1. **Everyone** reviews code. Especially new hires and interns.
3. Add a 👍 comment once you're done. In some cases may want to wait until you see how the author responds to or addresses your comments before you give the 👍, but it's also OK to say "Rest looks good" and trust the author to respond to your comments appropriately.
4. You might not have time to look at every PR you were @mentioned in, but you should do what you can.
5. Don't skip over code you don't understand. If you don't understand a block of code, ask the author how it works. One of two things will happen: you'll uncover a bad design -OR- you'll learn something new. Win-win.
6. Don't argue about white space and bracket style. That's what we use Prettier for.
7. It's not a bad idea to take a quick look at a PR even after it's been merged.

Zendesk Montreal Code Review Guidelines:

Reviewing PRs:

1. **Everyone** reviews code. Especially new hires and interns.
2. The time you spend reviewing code will vary from day to day, but about 1 hour a day is normal.
3. Add a 👍 comment once you're done. In some cases may want to wait until you see how the author responds to or addresses your comments before you give the 👍, but it's also OK to say "Rest looks good" and trust the author to respond to your comments appropriately.

5. Don't skip over code you don't understand.

6. Don't argue about white space and bracket style. That's what we use Prettier for.

7. It's not a bad idea to take a quick look at a PR even after it's been merged.

Learn the codebase

Learn how to write code

Zendesk Montreal Code Review Guidelines:

Reviewing PRs:

1. **Everyone** reviews code. Especially new hires and interns.
2. The time you spend reviewing code will vary from day to day, but about 1 hour a day is normal.
3. Add a 👍 comment once you're done. In some cases may want to wait until you see how the author responds to or addresses your comments before you give the 👍, but it's also OK to say "Rest looks good" and trust the author to respond to your comments appropriately.
4. You might not have time to look at every PR you were @mentioned in, but you should do what you can.
5. Don't skip over code you don't understand. If you don't understand a block of code, ask the author how it works.
6. Don't argue about white space and bracket style. Automate that shit.
7. It's not a bad idea to take a quick look at a PR even after it's been merged.

1. Code review is a daily activity; it will rub off on your personality
2. Your code review culture is your team culture
3. You can’t grow talent & trust without accountability

Are your interns reviewing code?

Thank you

alavers

alavers

alavers@zendesk.com

Up next...

(Friday @ 1PM in ST-Laurent 1)