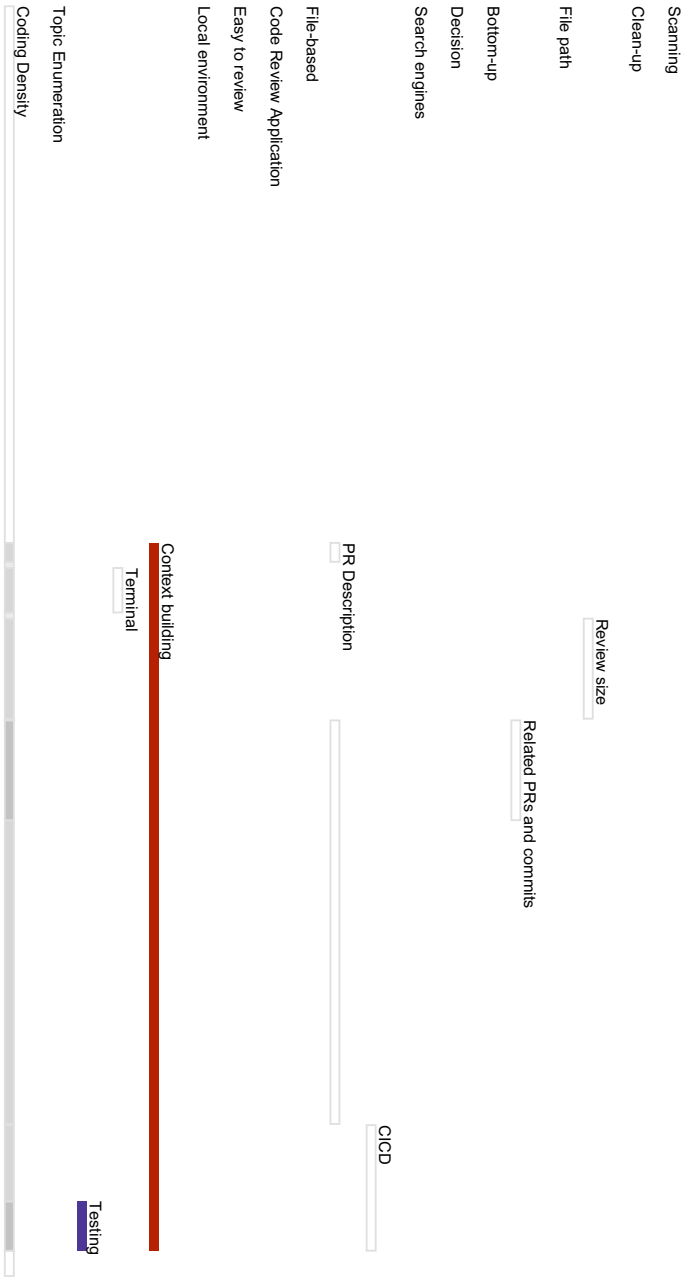
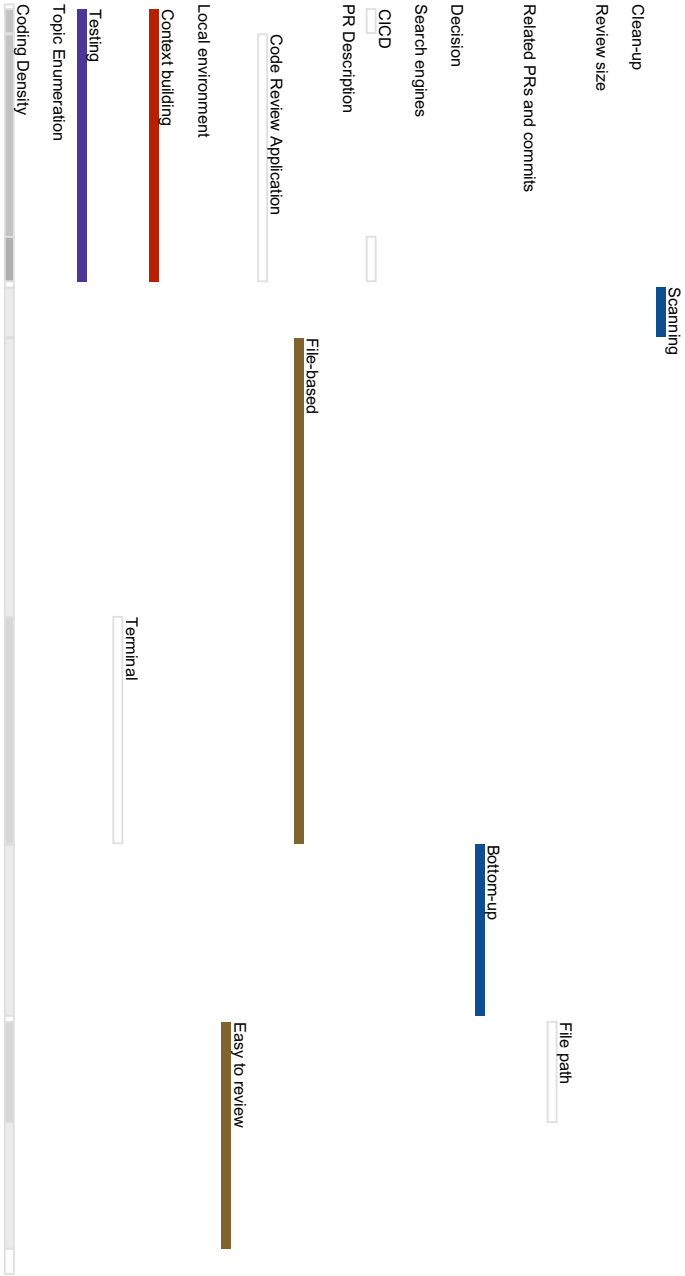
REVIEW TYPE AND GOAL

CONTEXT
- upgrading package dependency manager
- already has a local version of the branch
- They describe about the programming stack of their environment, programming language used in the front and backend, Git, dependecy manager used.- The reviewer is now not in the team responsible for the focus of the PR but historically the reviewer was responsible and the author is now responsible for that part but there is not that many people who have been deeply familiarised with it
- they had a hard time upgrading the same package manager before, it took them several weeks but they got a lot of benefit from it as every developer is much faster in their work - the package fuels their local environment. It sped up things, it created some opinionated ways of doing things in the repository and helped them a lot.
- The next upgrade was quite OK and for the next versions, the reviewer doesn't know yet for this one.

TRANSCRIPT
- Reads the title of the PR
- Goes to local terminal, checks out to the local branch of the PR again to be sure.
- The reviewer returns to the Github PR view. They look at the number of lines added and deleted in the PR and describes that even though the change is big, it is not application code and only tooling and additional stuff which is keep tracked in the repository.
- Reads the PR description The author says that the PR is blocked on some other library but the reviewer will check it anyway even though they are still waiting for another upgrade and give some feedback so that the author can change some stuff anyway. They can wait for that PR.
- They describe their PR conventions. In the PR, they always have to track issue with a link from linear (issue tracking system) that is automatically connected to the PR. Then, they have a self-check checklist as a part of the PR description where the author ensures they have done several things as requested in the checklist, for example, linked the issue with issue tracking system, assigned designer or security engineer if needed, added testing, tracking and left the code base in a better shape (better code comments, code style) than they found it. It is not super important, it is rather a check to self-reflect whether they for example left the code in a better state, but they might sometimes fight a bit when people claim they did something in the self-check but then they didn't really or it is subjective.
- Then, in the Conversation tab they also have the CI integration information - in their team it's specially type check, semgrep for consistency, linter for conventions on how o write code, docker etc.
- In this PR there are 2 checks failing. The reviewer always checks whether it is something related to the change, something expected or



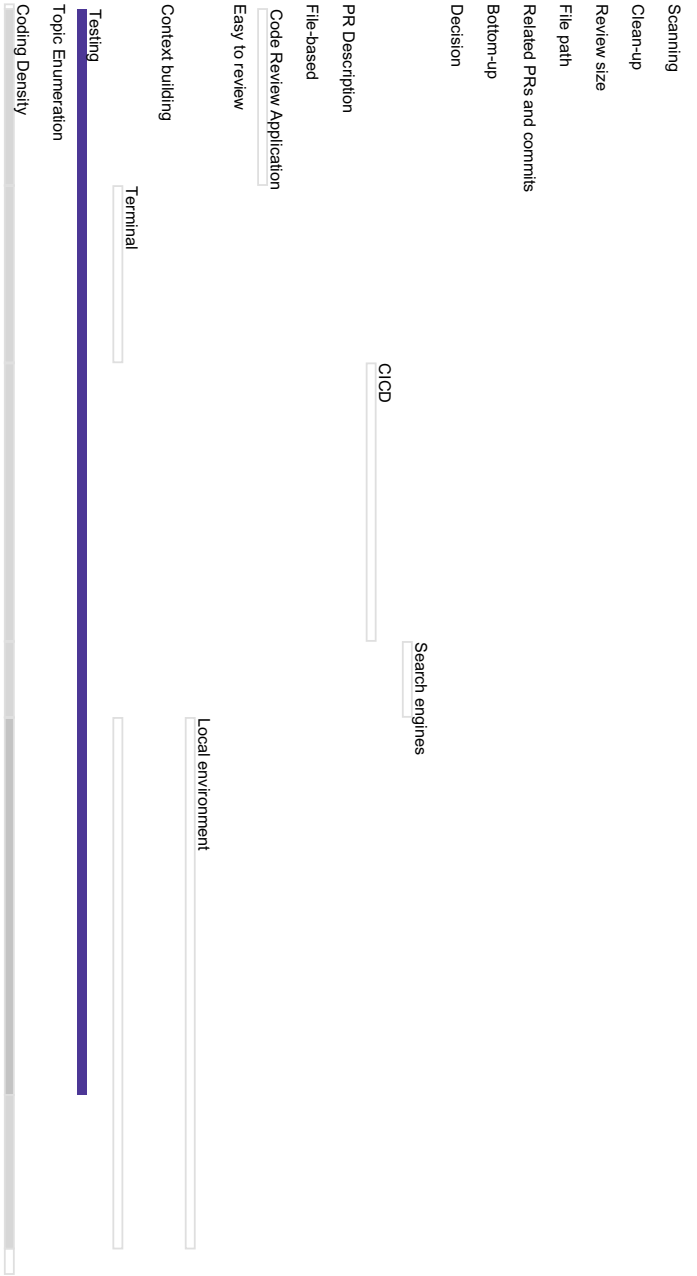
really broken before even looking at the code.
- The reviewer also always runs these review apps (front end or back end) if the whole web app runs. The app is normally renewed every day so because the PR is 5 days old, the reviewer uses a bot link in the Conversaiton to redeploy the review app so that when the reviewer is done with the review, the review app launched and the reviewer can click through it.
- The review app opens in a new Tab and is loading, so the reviewer goes back to the Conversation.
- Since the tests are not passing, maybe the review app will not launch, we have to see.
- Goes to Files tab, firstly scrolls through the whole change to see how many things are changed.
- Already sees that majority of the code change is in the autogenerated .lock file that locks an exact version of a dependency that will be used for the build and marks it as Viewed. Since they are upgrading the package manager handling the generation of these files, it is not really important to the change.
- Since the libraries can have direct dependencies or indirect dependencies then there could be some conflict in the versions of the used libraries in the autogenerated files. In such cases, it could be meaningful to review the auto-generated files. They can manuall update either the lock file or the versions in the main code base as well so that they don't make the build file bigger or prevent shadowing each other.
- Decides to look locally at the auto-generated file.
- Goes to the terminal and shows the change in the terminal window via Git diff.
- There are changes in checksums which is expected. There are also some formatting changes that are systematic and probably related to the new version. The reviewer could probably go through all the 17000 changes in the file, but it is not worth it as if something would be broken there, they would notice it sooner or later. If it wold be a security issue they have other tools to let them know about that.
- Goes back to the Files tab where the view is scrolled to the bottom to the last .lock file. Sees two hunks above that the author tried to update the library that does not support the upgrade yet. So the reviewer says the author probably tried upgrading and figured it doesn't work so the author will need to fix it.
- Next sees below that there is the upgrade of the package manager itself. Marks the file as viewed.
- Scrolls totally up and comments that there are some other parts of the package manager's ecosystem and the reviewer wants to have a look at that. The package manager's ecosystem files are all in the same directory which is always in the beginning of the file name.
- Sees that there are probably some files that were deleted as they are probably discontinued in the new version. Marks the deleted files as viewed without loading the diff.
- One of the deleted files is part of a module responsible to launch the library locally which is just changing from the old version to the new



version. So the reviewer marks the deleted old version related file as viewed without loading the diff and also marks the new file as Viewed without loading the diff.
- Then, there is a file containing some logic. The reviewer considers it important but skips it saying to get back to it later.
- Sees bunch of plugins (eslint linter in this case) which specify what versions of them to use with the new library and some of them contain the opinionated ways of specifying configurations for how to handle those cases.
- Scrolls through the plugin files to see if there is something controversial.
- Usually sees vastly what files can be marked as viewed that the reviewer can forget about and make the number of files to be reviewed lower. Then, the reviewer can focus on what is important.
- There is some new setting that the reviewer could probably check but doesn't feel like it togather with changes on the plugins that got removed so the file looks ok. Marks it as Viewed.
- At the end, sees some files related to the library that needs to be upgraded first so lets them be.
- Now the reviewer knows that there are 3 things to check (7 files left) - the configuration of the CI, plugins related to the skipped files and then the dependency that needs an upgrade
- The first important file is related to configuration of their CI integration (circleci) system. It is related to caching update. They check the current dependencies so it makes sense to update that cache. It works with the checksums as we saw them earlier so it needs to be updated. Then, there is just a comment update in the file. Looks good. Marks as Viewed.
- Then, scrolls down to the plugins. Sees that some plugins were removed (pointing at the removed files). Points out that one new library (or plugin) is added. Looks through the newly added library and its logic. The reviewer does not understand it so checks it out.
- Tries to Google for the upgrade as the author might have just copied the part from some blogpost on the migration topic. Check the release notes of package manager. He tries to use the Find to locate "migration" related content. The first link did not contain anything the reviewer was looking for. The second link was related to the old version so it was irrelevant. The reviewer gives up and accepts the changes.
- Goes back to the Files tab. Says the piece of change is probably a recommended way of plugging something that is not supported in the current version.
- Says that the library is a bit nuanced thing to use as it is replacing something that it is also partially using and then there are various different ways and work arounds to make unsupported things work so the reviewer assumes that the 2 files they are currently looking at are some wacky way to make things work.
- Moves on but does not mark the files as viewed.
- The next file is (like a linter) taking care that the code is formatted properly (and people do not fight over it) and the change ensures that it loads well. Looks fine. Marks as Viewed.

- The next file is just changed mods. The implementation is a bit more permissive than the reviewer would expect but they think it makes no difference. Marks as Viewed.
- In the next files recognizes a file related to their formatting tool and now it needs to be explicitly linked. Probably there used to be a built-in way to do it and now they have this verbose way of linking that tool.
- Comes back to the wacky file (unsupported api). The reviewer again googles the library field. They copy the name from the code and pastes it in the browser to search for how it is used in the context of the library. The reviewer find a GitHub link and understands that most of it is linking some libraries in this verbose way and the plugin needs another plugin to customise how it is used. The reviewer is still confuses on the naming part and misses the rationale for the presence of a last line.
- Scrolls back up to the place where it was first used and says that doesn't understand why did the author call it in the way they did.
- The reviewer doesn't think any of it changes matters as long as it works when they run it locally.
- Adds a comment asking that they don't understand why the data are being loaded where they are and why are they loading what they are loading. The reviewer wants to wait what the author has to say. This review is not going to be just accepted and merged as it is waiting for the other library upgrade so the reviewer is going to be just commenting at the end to just start a discussion. So this is like a preliminary review.
- Marks the file as Viewed and moves on.
- The next file is a package json for one library version. Interestingly, the library version is downgraded to a lower one in there.
- Adds a comment asking for a reason of that downgrade. Assumes that it is because that version did not work with the migration but wants to let the author to give more details.
- The next file is OK as it is just removal of path but the content seems to be the same. The reviewer also the inline comments is specified in a better way, so it is OK. Marks as Viewed.
- The next file (plugin related) also OK. Marks as Viewed.
- The reviewer is done with the plugin related files.
- Now there are the files related to the end-to-end tests that are not supported in this nx (repo manager) world.
- The reviewer apperciate a change. There's a nice change where they used to manually insert an externally available command in the command line but now the package handles the library directly. The reviewer says it's cool, that they like it. Marks as Viewed. Probably it is still not working as it is the library that needs an upgrade.
- Then there is a config which marks as Viewed and says to see whether it works.
- All the files are reviewed now.
- Opens the tab in the browser with the review app (deepnote) to see whether it is up and running. The app is running.
- Shows to the researcher their official website.
- Returns to the review app - it shows the web app of their product with a specific PR applied. The reviewer sees that the change doesn't break

the app which is good. It would be strange if that change would break only some specific parts. It is possible but there are other automated ways for checking that. The reviewer goes to check some other contents as well anyway - login works, opens separately a free plan function and a project - it seems like it's loading and the content is there so it looks like it's working.
- The review app passed and also the change looks OK.
- Goes back to the GitHub Files tab.
- The reviewer did not expect it to break anything, but it's better to check as it is a tooling update and it can affect anything.
- Goes to the local terminal to check the change locally.
- Remembers that to upgrade the package manager you just type the name and it should work. It starts the update to the correct version.
- It takes some time as we are installing everything in a new system.
- Says they could have done it earlier and decides to review in the meantime why the CI is not passing. Which tests are failing.
- Follows the link for details of the not passing test to the circleci app.
- Two tests failed because of some time outs. Both tests that failed have a label FLAKY (describes the FLAKY tests) so decides to re-run them. It is not the fault of the author so the reviewer just expects to run the tests and see that they are fixed.
- The tests passed.
- Moves on to the next failing integration test which is what the author referred to in the PR description - they are waiting for another library upgrade.
- Switches to search in a new tab for the library name and the upgraded package manager version.
- Looks through the results but says "Whatever".
- Goes to the terminal as the last thing remaining is to check the migration in the local environment. The reviewer reminds their own review checklist that the typecheck and tests has to work locally and if opens some file here, it should also work.
- The installation finished. The reviewer can see that all the packages were literally changed.
- Checks by the gittstatus command whether some files were generated that were not committed which is not the case.
- Makes sure his local environment is working.
- Tries to run the landing page as the first thing. The run failed as the reviewer does not have the latest version of node. Updates the version.
- Reruns the landing page. It might take a minute but it looks like it's working so there is no apparent chaos. Then, he will check it on their local dev environment also.
- Once the local run will be finished, the reviewer will add their review.
- Starts preparing the Review message. Says "Good job!" Lists that it looks like most of the things keep working well and what methods the reviewer tried to validate it (review app, local installation). States that the reviewer put some comments on issues that are not super clear. Then asks whether they are just waiting for the library upgrade then and whether they should create an issue for the library to ask whether they



plan to address the integration issue.
- Goes back to the terminal to see whether the local run has finished which it did. Checks in the dev landing page that it works and looks well. Signs in locally and it seems to be working.
- Adds in the comment that the reviewer tried the local dev also in the review message.
- Realises that wants to try another thing - to check the typecheck is working.
- Goes to the terminal and runs the typecheck command.
- Needs to use the correct ode again saying it's his fault that that is not a default in his environment.
- Says they're sure that they can merge it once the library upgrade is done.
- Submits the review as Comment.
- Sees that there is a conflict now as someone probably merged something which changes the .lock file but that is an easily fixed thing.
- Leaves the typecheck running saying that if it fails they will let the author know in another comment.

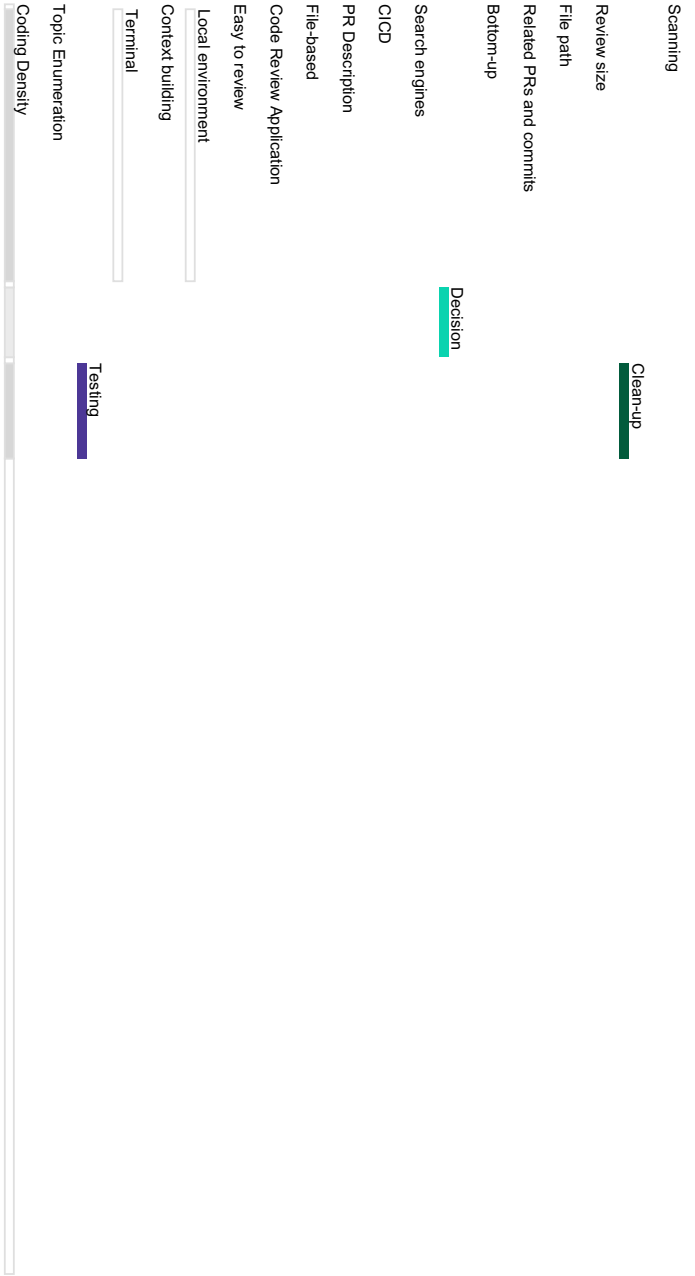
GENERAL STRATEGY
- Reads the title, prepares terminal for testing, gets rid of files that are unnecessary or easy to review, then goes topic by topic, uses several ways to implement the change and test how it effects the deployed web app, checks non-passing integration, tests locally.

NOTES
- The change seems like a very big thing but it is not application code but tooling code.
- Does not review in detail probably as long as stuff works locally.

INTERVIEW

I: Yes. So you were asking questions also in the send review phase, not just in the comments. What is a good thing to ask in the final review message and what is a good thing to ask in the code itself?

P9: Depends on the size, depends on the nature of the pull request. I don't know if there is a rule. I usually ask things related to the code when I'm just checking the code, something doesn't make sense, something looks insecure or slow, or I have questions, so I literally ask it there. And then I have all these extra things that I am checking. It's not like, you know, when I'm checking this website it's not like I'm checking this particular thing. I'm just checking the website in general, I'm just checking the local in general. So then the general things will land here because there is no line where I would attach that. But also like this depends like when, for example, if we, if we are designing like, for example, now in another pull request, we are designing this new website, right? So we have a different process for visual changes. Right? Well, we have a bunch of processes for the visual changes. But in



general in the code review if I am doing a final code review, I am not like necessarily checking - Oh, yeah, this copy or this color, I didn't like it. But if if I know that, it's again something that we previously agreed on, or if it's like literally broken. Again, I would like mentioned it here, like, Oh, yeah. By the way, this also got broken because of an accident, or something like that. So general things here and code specific things to wherever the code is changing that or touching that.

I: That he was mentioning the cypress issue also in the description of the pull request. Right? So, yeah. Okay.
Other thing that I wanted to ask is did you understand everything that was happening there? Or do you have some like level of: I don't need to understand it beyond that.

P9: Well, this is specific. This is very specific, [name] libraries, very complex in terms of configuration and plugins. And there is a lot of opinions inside the library on how things are done, and then, if something is not adhering today's opinion or standard, or specific standard of this library it's sometimes hacky to make it work. So these things here [points to the less understood piece of code], this one. I have a hunch why, it's like this, but I don't think it's the only way it should be. It's a little bit of a heck, probably to make this work, but it's probably heck that even the authors of the library like recommend. I think. I don't know. That's why I also asked these [points to the comment].
But I don't necessarily need to know in this case, because if this works this will not cause any security issue, any performance, issue anything. This is literally just like a little heck to make the library happy, the pluggins happy, the links happy. And yeah, that's just this package manager nuance. So so yeah, I don't understand anything beyond that. But like, if this was application code and I didn't understand anything, I would definitely either make sure that I understand or ask about it. Or there is another reviewer that can go a little bit deeper than me.

I: But I mean you tried to check whether that is some way that the package manager people are recommending right?

P9: I quickly trie but this package manager upgrade is very new. Let's check. It's it's very new. It's probably October 23. Okay, right? So it's like one week old. As a matter of fact yarn is not that popular anymore. Now, there is the new kid (pNPM) on the block that is even faster, efficient package manager. Right? So now this is popular. They are making fun of our library being already old, so we are already eyeing out on these. But we don't need to. We don't really want to change stuff if we don't really, really need. So we just want to upgrade this work. So maybe there are some things here but I don't know. Like, Oh, yeah, okay, so there is some update. But I think not a lot of people tried. So it may be something that this colleague of mine really, really found somewhere hidden inside the internet. Or it's just something he hecked around. I don't know. Let's see what he says.

Scanning
Clean-up
Review size
File path
Related PRs and commits
Bottom-up
Decision
Search engines
CICD
PR Description
File-based
Code Review Application
Easy to review
Local environment
Context building
Terminal
Testing
Topic Enumeration
Coding Density

I: So when when is it good to like? Try to find stuff yourself. And when you just ask the author, for example.

P9: So in this case is literally libraries and stuff that we use. So if quick Googling gives me something that I will know… But the owner, should make sure that it's clear why some changes are made. But again, like this is very, very specific. So maybe it's not that bad like. If it was not this bad application code I would not accept if he doesn't add comment, because next person that will look at this should definitely understand should not be googling around. But this one literally is just him and me who looks at this, if it works and not even me anymore. I looked at this, maybe one and a half years ago. So in general, the author should be responsible for making sure, like it's clear what they intend to do, or why is it not standard? Or why is it this or that way?

I: so just o summarize because you will have to go. Your process is basically to look through everything, reduce it down. Then go, let's say topic by topic, and check the logic of that, whether it holds and at the end you test everything. But you actually started some testing in the beginning, right?

P9: Yeah, yeah. So there is some automated testing again. If it's like application code, maybe very sensitive, or something with the data or billing or security I like test the hell out of it also in the code. Also in the like this application environment. So I really, really, really look, try to break it. But then, right? One reviewer is not enough to break everything in some, if it's something bigger. So then we have like internal testing session where multiple people don't look just at the code, but they test the whole functionality. But the reviewer also should be responsible in our case. If it's much bigger company, there might be some other processes. But in our case the reviewer should be responsible. At least I should be responsible today. Tried if it works. Not just look at the code because the code can look okay. But then something else is broken, or so the reviewers should be responsible also for the code also, for it works or not. If it's bigger, we can do something else about it. But yeah.

I: So what are the things that you would, for example, test before, and then you would test after. Does it have some logic? Or it's just about how fast it runs? Or

P9: No. Again, this depends on what it is. Since we are very multi tasking, multiple areas of expertise, every developer looks at front end and back end and security and design like a like random subset of all of the things that can be there. So depending on what is it and depending on that, I decide. Like, if it's just a visual stuff I don't need to look that much into the code but more like here [the web app], right? But we also have a designer for this. So usually I defer that to our designer in our team. For example, now we are redesigning the logos here, whether the

Scanning
Clean-up
Review size
File path
Related PRs and commits
Bottom-up
Decision
Search engines
CICD
PR Description
File-based
Code Review Application
Easy to review
Local environment
Context building
Terminal
Testing
Topic Enumeration
Coding Density

logos are nice and they work here, and also on Mobile. But then this is something that I would focus if it's all up to me. And in the code, it's just about some, some minor things. But if it's something related to billing. For example, another pull request, we are changing some versioning, for like prices that we offer for our machines. So in that case I literally spend more in the code, even think about the architecture of the solution in the code. Maybe we have some design doc that is describing why something works this sort of way, why some change. So sometimes we do like RFC request for change that is reviewed prior to the implementation. So I would already know what I should be expecting in the code if I was not reviewing, and I would check it out. And then, if we had some controversial thing in the code, I would definitely raise it right. I either need explanation or I would request change, because it's not either up to the RFC that we've agreed on, or it's like causing some future thing that can happen in 6 months, because some billing will something, because something something. But this is really really specific. So so yeah, so depending on the type of task. But there is more logic, business logic called back end or visual, or really just the tooling like this one - depending on that I kind of do different things.