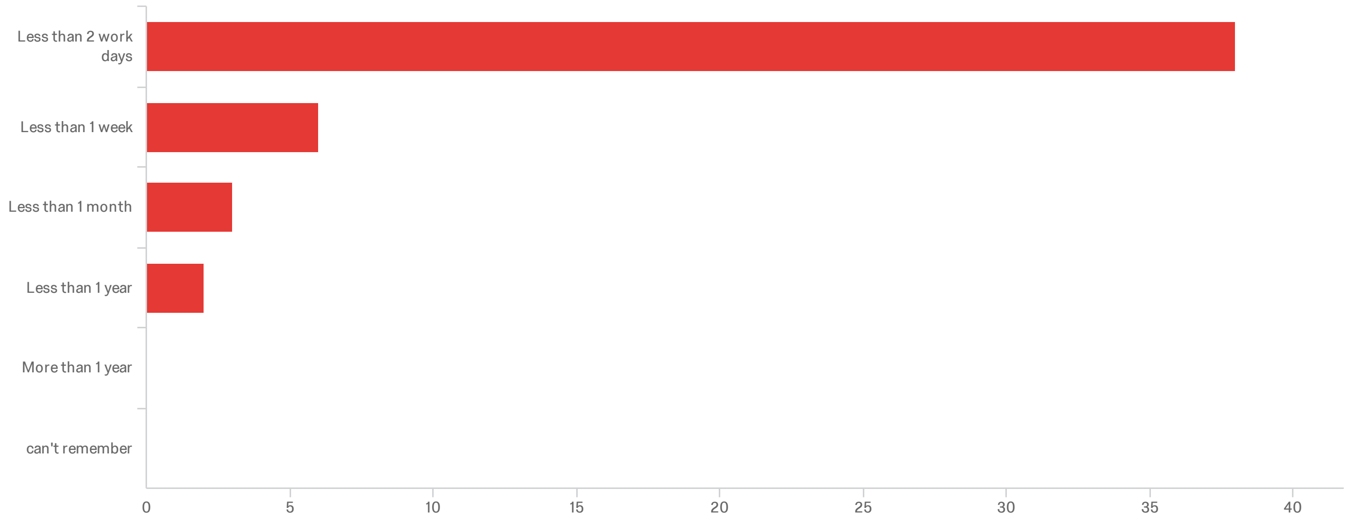
Test

CodeShovel

March 5, 2019 12:09 PM MST

Q1.1 - Q1.1 How recently did you last use source code history of any kind?



#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	Q1.1 How recently did you last use source code history of any kind?	1.00	4.00	1.37	0.77	0.60	49

#	Field	Choice Count	
1	Less than 2 work days	77.55%	38
2	Less than 1 week	12.24%	6
3	Less than 1 month	6.12%	3
4	Less than 1 year	4.08%	2
5	More than 1 year	0.00%	0
6	can't remember	0.00%	0
			49

Showing rows 1 - 7 of 7

Q1.2 - Q1.2 Please describe this most recent activity. How did you use source code history? What were you looking for? Did you find it? Did the tools you used support you in this investigation or could they be improved?

Q1.2 Please describe this most recent activity. How did you use source code...
Figure out who did certain changes
I opened a project I have not been working on for some time, so I used the commit-diff feature to see where I left the project, what changes I have made and where to keep on working.
Looked for changes. Tools that have been used, were useful.
shell, code changes, yes, yes
Used local individual file history to restore a previous iteration of my own code and used line annotations to identify when certain lines were changed last. Regarding the file history, I used only the most fitting version because there were so many revisions that finding the right one was too time consuming without knowing the approximate time where that version was actually in use.
who wrote the code initially / who made the last change and what did he change
I Iteratively fix bugs in code driving a data processing pipeline. I try out a few things, then commit them to the repo when they are tested.
Using git to manage multiple branches that depend on one another. Each branch corresponds to a separate feature, and each feature is submitted for code review independently. I was ensuring that each branch contained all the changes it needed to be atomic. This is possible in git but requires advanced knowledge of git -- I wouldn't say it's a first-class use case.
Reviewed a pull-request, merged a pull-request, changed to branches to see other people's code. I was looking to see that a bug fix was implemented. The tools I used were Git command line, to switch branches, and Bit Bucket online, where line changes were listed in red and green. The Bit Bucket tool could certainly be improved. Unlike Github, it is difficult to click into full files and the entire code history. I suppose that the Git command line could begin to use plugins, if they do not already do, to implement interesting features.
I searched in some diffs between two git commits for reasons why something in the code broke.
Tagging the repo for a release. Pushing the tag to origin. Used git CLI.
I was looking for the origin of a merge conflict. I was able to find the origin by using standard Git commands like git diff, git show and git ls-files -s.
I did some changes to the source code but needed to look at the previous state to resolve some bugs I introduced. I found the problem. I used the GitHub website to look at the history.
git log, git diff, git show to find reasons for recent changes
I had to do some version updates of dependencies that were done in another project too. So I had a look at the history to check which versions did we use in the other project. The tool (Bitbucket) I used supported me in this investigation.
checked the last commits to identify was was changed recently - and yes found it

Q1.2 Please describe this most recent activity. How did you use source code...

Figure out history of changes. Continuous Integration tool was not doing a build anymore. No change was made to source code. So I could concentrate on supicous configuration. Tool support was okay for me.

Used git to make and push some changes to the repo. Also, browsed for a change made by specific commit in the history (for 2 month old commit), i.e., looked at the "git diff" based on the SHA of the commit. Git was good enough for what I wanted and I did get the work done.

I reviewed recent changes. I used source code history to identify the changes, verify they were correct, looking for a specific change that might have introduced an issue. I confirmed that the problem was not recently introduced. The tools were quite good at helping me find those changes.

I was looking to merge branches into the main branch. (There were several disparate branches because each individual team member was working in isolation.) The worst aspect was that all of the deleted garbage files cluttered the history, so I had to filter that out. I used a lot of git diff.

Looked through a diff in a pull request

I searched for an old version of a file. I used the bitbucket for that. I found it and the bitbucket interface was great help.

I was merging few code changes that I implemented few days ago with my current working branch. I use git and I'm completely satisfied with it so far.

I was using github to go check who had been contributing to a enterprise project before and who I could contact for dev support.

I was reviewing a pull request in GitHub. It was a small PR and I had no issues with it. But reviewing PR is usually a tedious and boring task, because this may involve reviewing many dozens of files with unrelated changes (especially at the beginning of a project, you may need to refactor a few things while implementing a functionality). You could group them under commits, but when you review a PR it's easier to review the whole changes instead of commit by commit.

I used “git log” and “git show” to check what I modified. I was ensuring I didn’t commit the wrong thing. I was also trying to have a overview what I have done. Just many commands involved. Better to have an easier ones.

Had to look if the stage branch contained all hotfixes from production, and how to name the next stage release. Used git on the commandline for merges, but SourceTree too look at the graph (tube map).

Looking for which commit a block of code was inserted in. I did find it eventually, but the tools aren't great, since I couldn't find a way to track lineage of a particular line of code, just the last commit which modified it (or my git/hg-fu aren't good enough!)

I was using it to identify when a colleague made a change, and also looking to see how commits were made. I was easily able to find it. I was using git-x. I also used history to identify what was changed previously and why. I found it using an annotation addon in intellij.

pull/commit/push cycle in git and the equivalent in hg and looking through history to identify what changed recently (since my previous commits) and also who is associated with some changes that I noticed.

I was checking the diffs in my branch against the committed code. A normal diff on the terminal was sufficient.

- Check when a change was made - Read code from a pervious version Of course I found what I was looking for. I can't thing of anything to improve.

I was looking for an outdated implementation of a functionality we aimed to re-introduce (in a better way)

Mainly for the basic tasks pulling, committing and pushing. I did not search for something specific. It's mostly a problem of myself that in addition to a certain improvement or bug fix, I put irrelevant enhancements into commits instead of committing the irrelevant ones separately. So I like the Bitbucket function of commenting on certain code parts of a commit and highlighting a particular spot that was responsible for a bug.

Looked for an old git commit using IntelliJ. Found it because of good commit messages. IntelliJ helped excellently.

Q1.2 Please describe this most recent activity. How did you use source code...

Checking for which ticket the changes in a file have been made. Yes I found it.

git

1. Embedded Eclipse plugin for GIT. 2. Recover changes overwritten by a team member by accident. 3. Yes. 4. Yes, Yes, visualization could be better, had to walk through all files line by line to compare changes.

Analyzed code changes over a period of time to find a bug/introduction of the bug. Tool support was sufficient (git/bitbucket/IDE)

I wanted to understand how the solution to a certain problem was implemented. I did find what I was looking for. The tool (Stash) was perfectly adequate in displaying the added and the changed lines of code.

Evolving of software architecture. Understanding what steps a certain component took to get to the shape/position it was in. I used GitHub and Tower to navigate the repository and the module's diffs. Navigating the file history and searching through the history of near-by modules by navigating the repository at different positions in time. No, the architectural evolution and reasoning of the modules under consideration was not reconstructible through observing the repository at different positions in time.

I often study the history of my source code to understand how and where other people worked since my last commit/push. I do not need any particular advanced tool to do that, since I am usually looking at small modifications that can be easily managed by the built-in tools of git/svn.

- reviewing a pull request - with the user interface of Atlassian bitbucket - compare the changes to the previous version of the file - yes - yes

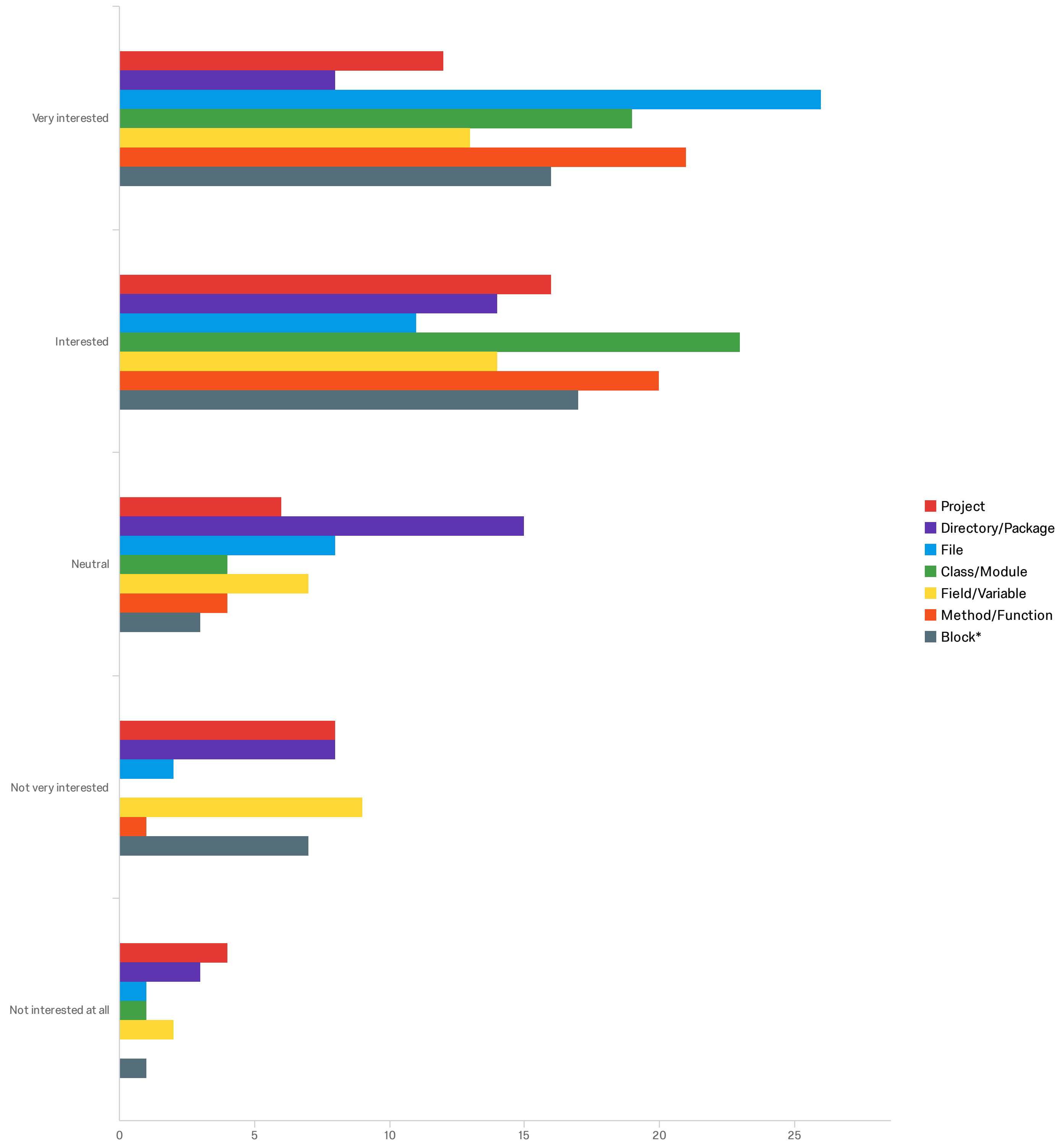
I was trying to revert a previously deleted feature, which wasn't tagged correctly. Unfortunately it took some time to find the right revision. In PHPStorm there is only a history view without further search functionality.

I search for a property in a configuration file hosted in a Git repository to understand how and why it was changed. I found the commit in which the property was changed. I used ' git blame' and 'git diff' to trace back the changes to the property.

Pull requests to review code using Bitbucket. Works fine most of the time, but sometimes changes (especially refactoring/reformating, but even adding of new methods) get displayed in a confusing way. Diffs of specific files (mostly changes of the last few hours) while developing using IDE. (Mostly to restore old states after realizing "I should have committed that".)

Contents of a file contradicted what has been documented elsewhere. Wanted to identify who changed the file at which point and whether there were any regressions (e.g. due to merges). (result: the discrepancy was intended) Used the Bitbucket Server interface to navigate first the history of a file, looking at various diffs. Color-highlighting of the changes definitely helped. Sometimes the context of what other changes where done in the commit were a bit sparse.

Q1.3 - Q1.3 In terms of source code granularity, how interested are you in gathering information on source code history at the following levels?



#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count

#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	Project	1.00	5.00	2.48	1.28	1.64	46
2	Directory/Package	1.00	5.00	2.67	1.12	1.26	48
3	File	1.00	5.00	1.77	1.00	1.01	48
4	Class/Module	1.00	5.00	1.74	0.78	0.62	47
5	Field/Variable	1.00	5.00	2.40	1.22	1.48	45
6	Method/Function	1.00	4.00	1.67	0.72	0.52	46
7	Block*	1.00	5.00	2.09	1.12	1.26	44

#	Field	Very interested		Interested		Neutral		Not very interested		Not interested at all		Total
1	Project	26.09%	12	34.78%	16	13.04%	6	17.39%	8	8.70%	4	46
2	Directory/Package	16.67%	8	29.17%	14	31.25%	15	16.67%	8	6.25%	3	48
3	File	54.17%	26	22.92%	11	16.67%	8	4.17%	2	2.08%	1	48
4	Class/Module	40.43%	19	48.94%	23	8.51%	4	0.00%	0	2.13%	1	47
5	Field/Variable	28.89%	13	31.11%	14	15.56%	7	20.00%	9	4.44%	2	45
6	Method/Function	45.65%	21	43.48%	20	8.70%	4	2.17%	1	0.00%	0	46
7	Block*	36.36%	16	38.64%	17	6.82%	3	15.91%	7	2.27%	1	44

Showing rows 1 - 7 of 7

Q1.4 - Q1.4 When you use code history, how far in the past do you usually examine?

How do you determine how far in the past you want to go?

Q1.4 When you use code history, how far in the past do you usually examine?...
if I see an interesting change i don’t care how far back it was introduced...
~ 1-2 Months. Really depends on the project (how many developers are working on it, commit style: micro vs major, developer skills: is an intern involved?)
as far as necessary, it depends
Usually a version number or release date can narrow down the time frame of relevant commits to reasonable levels. Depending on the number of possible elements to analyze, I usually adapt the code level and more precisely select individual files or even just parts of a file to look into.
i am interested in the history of a code snippet (who did it / what was the code before), it doesn't matter how old it is - what matters is the number of iterations where the functionality was changed
It's generally for a few reasons. - I either want to figure out why/when a bug was introduced (by bisecting). There's no hard limit on how far back this might go in the history. I pick a point in history that I know was a good spot, and the current buggy point, and search between the two. In larger teams, it's often due to bad merges, and so it's interesting to track persons or features that introduce a regression. - Often I just want to revert a recently committed change, generally in my local history (stuff I haven't pushed). I like staging things, and using "git diff" as a tool to keep track of where I am or what I'm doing at the bleeding edge. -- especially from one day to the other. I'll often try a few things, and rebase/squash into the local commit that fixes that stuff. - I go by concept/feature for my commits. So sometimes there is no right granularity to look at patches. A refactoring operation, for instance, is conceptually simple, but ends up touching everything often. I have a vague notion of how things used to work, and so to find a commit in the past, I'll often rely on both my own commit messages, and I'll go by file. For projects that have very few (large) files, I will have to go by function, or local spots, so I use "blame" tools to figure out which commit touched each line last. If I don't remember where I changed something, but I remember an identifer, or a comment I've typed, I'll use git grep and go from there.
Depends on the pace of code development. In a project that sees several commits daily I might look back as far as 50-80 commits. Also I regularly use the functionality of git blame, which can document changes to a file that occurred arbitrarily long ago, and I find this useful when I want to understand the context in which a particular line of code was written.
Within a few months. I usually check to see if something has been modified that is relevant to the current issue that I am working on. If it is current, I want to see other related changes to keep in mind. I don't want to destroy other people's changes, so I want to understand their business logic and see why they implemented some code.
Depends: On older projects there is no exact time frame, in current projects it's often from now until the last one of my commits (i.e. all commits someone else made).
1-3 commits.
Most of the time, I only need to go back a couple of days. However, if some functionality seems odd or obsolete while reviewing source code, I need to inspect commits that are many months old.
Usually the last few changes (most of the time only the last change). To go back I use the commit history.
difficult to answer as it depends on the priority / impact of the defect / feature and how crucial the understand of it is ...
Usually I examine the last few commits (maybe 3 to 5 last commits). I have a look at the git network and read the commit messages, to find the correct commit.

Q1.4 When you use code history, how far in the past do you usually examine?...

Only couple of Commits which is mostly a couple of days. Sometimes I've been looking for a feature that was/wasn't implemented some months ago.

Last 5 - 10 commits, not depending on time.

&gt; how far in the past do you usually examine? Usually 2-3 month, rarely anything more than a year (or all the way to the origin). &gt; How do you determine how far in the past you want to go? I usually stop when I find what I am looking for. It usually happens to be 2-3 month old. I'll go all the way to the origin until I find what I am looking for.

Usually, I search back one or two weeks. Occasionally I will search back farther than that. How far depends upon the goal - if I am looking to find when an issue was introduced, I might search back years.

Not very far at all. I would keep it within the current sprint which is generally 2 weeks. Anything beyond that will be classified as a bug not a work in progress. This is because the branches are closed once things are merged into master and deployed as part of continuous deployments. This is very difficult to retrieve after the code is deployed. Unfortunately, we'd end up having to talk to the person who developed the feature rather than relying on code commits history. It also doesn't help that teams commit tiny changes 3-4 times a day, so it's a lot to sort through.

Depends on what I want to do. If I review a pull reuest, I only look at the latest changes. When I need to look up, how something was done in the past, I might even go back to the beginning of the project

Based on the current problem I try to solve, I go back in history until I find relevant code changes in this part of the code

I don't usually go more than a few weeks before in the history. Also it is very rare that I search commit history based on timestamp.

As long as I need

I don't have any fixed answer for this, it really depends a lot on the particular case. Sometimes I need to trace back the lifespan of a class until it was created (which might get tricky if it was renamed). Sometimes I need to check the very first commit of the project. I usually have to go to the file history and select the commit where the change I'm looking for might had happened.

Probably days at most a week if I want to ensure my work and have a overview. If using git blame, it can go very far.

Phew, as far as needed. Usually as far as the cause of some bug. Sometimes I use git bisect to find that. Sometimes I also use the history for "When did we launch v1" and that can be some years...

Usually a couple of weeks back to try and locate what caused a regression. Sometimes, have to go much further to find a changeset which is unaffected.

Time does not matter in distance I go back. The only thing that matters most is changes. I go back only 1-2 changes max.

Usually examine up to my last commit since that was the last time that I had a snapshot of the code in my head in a consistent form. If using branches then consider the entire branch, especially if am trying to figure out a merge.

I go as far as the last commit *I* made to the project, or from where I started working on the project, whichever is more recent. If this does not help in my search, I look at the commit messages to find something that seems relevant and then explore the diff.

As far as the change i'm looking for is in the past. I use tags and commit messages.

Most of the time I'm looking for relevant Jira issues in the first place, so that I know in what time frame the changes I'm searching for were made.

Until the last release tag.

Q1.4 When you use code history, how far in the past do you usually examine?...

Depends on what I'm looking for. Usually not further than 10 commits, because of different branches.

Depends, sometimes years if I'm looking for the reason for a code change. This depends on how old the project is.

till i find what i am looking for

Depends...usually I know who committed changes in the past days as I am constantly checking for updates in the SCM. When I'm uncertain about the history I have a look at the merge graph (GIT) and who of my teammembers contributed changes. Depending on the (hopefully good commit comment) I decide which commit to analyse, when searching for a specific change in history e.g. in a class file.

At least back to the oldest currently productive code

Usually only one step into the past. Rarely more. Only for very specific investigations one might have to step back a few times or look at a diff compared to a certain time point when an issue was believed to have been introduced.

Starting at a blame I get a rough overview of the age of certain parts of a module. I then try to triage down to issues I am looking into. Often line based visualisation is not really helpful for that. One has to jump back and forth in time between often unrelated changes/parts of history.

I am usually updating myself on what it has been done since I last worked on the project, so it highly depends on when I last worked on it. However, usually, I look at 3 to 6 days of work.

- usually around 1-4 weeks in the past - i move back in time by commits

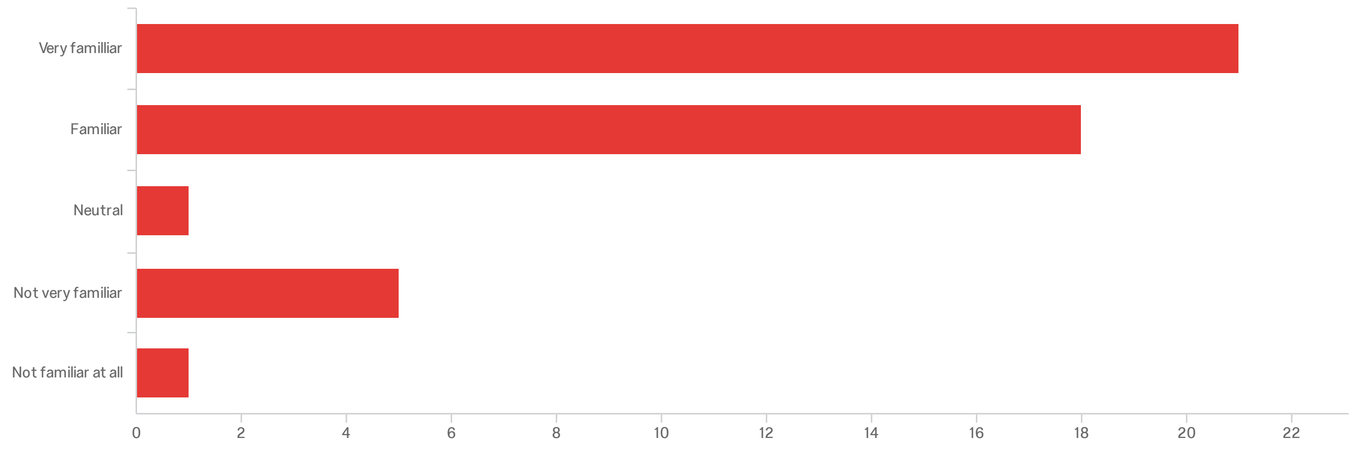
It'll be great to have the complete history available all the time.

It depends: On a project I'm actively working on I usually look at the code history for the past week. When using an open source project and depending on the change I'm interested in there are times where I have to look at the commit history years ago.

Mostly between a few hours to about two weeks. But for some cases (e.g. to understand why and how specific parts became the way they are today) it can expand to several years.

This largely depends of the goal. On larger projects with multiple developers, I most often start at the last commit I did myself (if any are available) trying to recreate the context in my mind and then go from there forward in time.

Q2.1 - Q2.1 Does this scenario sound familiar to you (i.e. have you encountered this in the past)?



#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	Q2.1 Does this scenario sound familiar to you (i.e. have you encountered this in the past)?	1.00	5.00	1.85	1.04	1.09	46

#	Field	Choice Count	
1	Very familliar	45.65%	21
2	Familiar	39.13%	18
3	Neutral	2.17%	1
4	Not very familiar	10.87%	5
5	Not familiar at all	2.17%	1
			46

Showing rows 1 - 6 of 6

Q2.2 - Q2.2 Please describe very briefly how you would approach this problem. What kinds of questions would you like to answer? What tools or approaches would you use to answer them?

Q2.2 Please describe very briefly how you would approach this problem. What...
I always use the intelliJ "inspect tool" to quickly navigate through the files
First, have a look at any linked issues to find problem or feature descriptions, then consult documentation or the pull request creator to get more information. If too unsure, then at least test it locally or on an existing test system whether the intended functionality works as intended. At the end of the day, the responsibility of verifying the correct implementation is on the programmer side, not on the reviewer's.
first i ask for an informal description from the author. second i ask for a unit test / integration test. normally after understanding the test, one understands the code what was changed (ideally: test failed before the change)
The purpose of a patch is often not visible in the code alone. It It is my opinion that good comments (or PR messages) should describe the "why". The "what" and the "how" are essentially the code itself. I would want to know: - it is a fix, or a feature improvement - is it worth merging in (PRs can introduce other bugs) - is it related to something that's been repeatedly breaking. (is it a regression). looking at the code history for the affected region might reveal some important things. - is the patch tested properly. does it cover all cases. - are the limitations of the improvement/fix documented. is it a silver bullet. - is it linked to a particular issue. does the code submitted refer to that issue.
Tools I've used for PR code review, mainly Github and Bitbucket, usually don't make it easy to browse parts of the code that weren't modified in the PR. Yet, if the changes are to a part of the code with which I am unfamiliar, this is necessary to understand the full context of the changes. For this reason I will usually pull the feature branch in question to my local copy of the repository, and use my IDE to browse the changes, and the context in which they occur, at my leisure. This usually involves switching between e.g. Github, which highlights the changes and allows me to make comments on them, and the IDE, for semantic browsing of the code.
I would like better explanations of the bug/feature that is fixed or implemented. Sometimes context from a comment helps monumentally. Tying a code change to a ticket in Github, BitBucket, RT, etc would be beneificial in the code-viewer, so that context is always available and it does not have to be tracked down.
The obvious questions are why was this change made and what is it doing? I would check who made the commit and get in contact (in our case comment in Bitbucket on the specific pull request). Or look for a test for this function and check the test if this clarifies the reason and function behind the change.
Look at the whole file. Look at corresponding ticket. Ask commiter for intention of change.
To solve this problem, I would view the diffs of the relevant commits and try to run the relevant function in the original and the modified form and compare their results.
I would look at the code and follow its execution path (might need to look at other modules/classes). I would add a comment on parts that I don't know so that the person that created the pull request could answer my questions. The questions will mostly be general, like: Why does it need to be changed? What other parts of the code will be affected by the change? Are all boundary conditions satisfied? Will it break something? Tools and approaches: GitHub website for reviews and comments, IDE to look through the code, IRC to ask questions to the contributor.
pairing with the developer and find it out :-)
1. Checkout the complete code 2. Try to understand the general problem that the code should solve 3. Run & Debug the code, to see what happens at that specific part where the changes being made Bitbucket, IntelliJ IDEA, Tools that I need to run the code.

If I don't understand the code that I should review, I assume to contact the developer to get all needed background information. In addition I would check if there is any ticket related to the changes that provides more details.

I would analyze and use an editor to write down my comments. If code is very very complicated I will print out on paper and use a marker.

Usually there is a bug report (e.g., JIRA) for a small pull requests (PR) or a separate documentation for the bigger PR (e.g., feature). I prefer to get the higher level picture by reading docs, followed by the actual code.

I would read the code and reason about what it is doing. Generally I don't need anything more than an editor to read such code.

In a code review, I would be familiar with the code because only teammates with expert domain knowledge should review and give +1 to a PR. I would have to ping the person and ask what the change is for, but the PR will be linked to a JIRA task that I should review before bothering them. PRs are assigned within a small team and we were generally working on top of each other's code and very aware of any buggy code or upcoming enhancements.

Comment on the pull request. Either comment on the code directely or regularly on the pr itself

If I come to this kind of problem, this is mostly a sign of bad code quality. I will then ask the pull request creator what this should do and think about possible refactoring actions and comment them in the pull request

I never had to look for the past changes made to a code to understand it. But I had used git blame command to look for who was the last person changed some lines of code.

I would read through the code and try out the branch locally, in addition to running the tests and follow the flow of the code. Github and editor.

The one that triggered this PR should add a description with any information relevant to the reviewers (i.e. why, how). Besides that, the tool could help by grouping the changes by topic (not by commit, which may not contain final changes).

Read the commit messages first, then follow the thought, examine codes. Since commits are likely to be small, it should not take too long to figure out each commits, then the entire pull request.

Ask my IDE to show all callers of that function in my IDE, and then also look at the diff in this PR...

I'd really like much more context around the diffs. The other thing I'd like is subcommits to a larger commit; for example, assume a change that modifies a function and several call sites. I'd like to be able to group all the actual functional changes in a single subcommit, and then group together all the other call site changes and the like. Individual commits don't really work well because they break bisection.

I would use two tools, and issue tracker and github diff The issue tracker would tell me what were the design decisions that lead to the change. Github diff tells me the previous change with the current code.

Look at the comment in the pull request -- who made it, and what issue it is referencing. If the pull request is huge (sigh) then consider the tests first, then iterate with the person in the comments section to figure out what is what and why it is so large. Usually relying on the person who made the pull request is my go to strategy -- they have to be accountable for the pull request and in their interest to get it merged, so I expect them to volunteer answers quickly and to the point.

First, I do a diff to understand the relevant sections that I need to review. I then see if the change seems proper. To inspect a code fragment that I can't reason about, I use cscope to find how it is being used. I trace the calls to that function and try to reason about its usage. If the change does not make sense to me and seems sketchy, I make notes on the PR and ask for clarification.

Everything is in the code. Of course I know what it is doing.

Asking the Pull Request author? :D

Q2.2 Please describe very briefly how you would approach this problem. What...

Phew ... good question. I always ask the pull request creator directly. Of course you can comment with the already existing tools code parts and ask why was it done the way it was done. But typing takes time and during working hours you do not want to write big explanations. So my priority approach would be the Hipchat Call.

The PR should be linked to an issue which creates enough semantic context most of the time. If you then still don't understand the code it's either because the code is bad or you do not know the context well enough. In that case you would need to read the codebase.

I'd either talk to the developer directly or comment the file in the pull request.

read the code read comments aks committing person test the code

To understand "why" the code was changed, the easiest way is to ask the developer committed the change and ask for the reason. To answer the question "what" was changed in the code I would use a diff tool to compare and analyze the changes.

What was changed? Did the behaviour change (was a test modified to match the new behaviour)?

Investigate what the whole repository or relevant sub-systems are meant to do, then determine how this PR fits into that picture. Consult the PR description or associated design docs as to what the PR is supposed to achieve and then determine whether the code changes seem reasonable in that context.

Instead of understanding the evolution of the module/change-set I tend to ask questions or perform a pair review. Reconstructing the knowledge from plain diffs and the PR's change request is often to cumbersome. I often wish for more Architecture Decision Records I can read to understand why certain architecture decisions (APIs etc) arrived at the stage they're in.

If the change in the code is not self-explanatory, I would expect to find a clar and explanatory message in the commit.

Find the bug/feature ticket in which the problem is described and try to understand the business case. Questions are very specific to the problem. - is there a mockup (if ui related task)

In general the pull request should contain all aspects of what has changed and how the changes should behave. Also it must contain a detailed setup to build an exact test environment.

I would discuss the pull request either directly with the developer who opened the pull request or in a code review meeting in order to clarify the functionality of the code and establish a common understanding of how to use pull request and their descriptions to improve the overall code review process. Some question would be: Who did this change and when? Was it a single change or did it evolve over time? Why was the change made? Is the change related to an issue/bug or is it an improvement? In order to answer these questions my first step would be to look at the code history and then look at the tools involved in the development process (requirements documents, isse tracking system, ...).

Inspect the diff, read the comments and Javadoc, checkout the branch and browse it in the IDE. In more important/complex cases or projects I'm very new to, talk to the developer or ask questions as comments. Questions can be general like what the responsibility of each class is or target specific changes and choices made while implementing them.

Hopefully there is some textual description of the goal that has been achieved with the code changes. From that functionality I would work backwards to see what the different changes in themselves actually achieve and what their role is. Questions I would try to answer: - Why were the changes made? - Any sideeffects? - Does it harmonize with the general code-architecture of the project? Toolwise I would do most of this in the web UI of the code repository (Github, Bitbucket, ...). For larger changes I also run that version locally and step through the different functions using a debugger.

Q2.3 - Q2.3 Using source code history, how would you find changes to this method only? Please describe briefly.

Q2.3 Using source code history, how would you find changes to this method o...
Annotate
Text search
Use multi line history for the method block to identify commits with changes in this area. If that is not enough, expand to file history or check the identified commits globally to see interconnections between files.
git blame
That's hard with line-based history search. I find it painful enough that I don't even bother. I'll narrow by file. Methods can move between files, but they generally don't span multiple files. Then, I would probably word grep for commits containing strings in that function -- if I know that an identifier changed, for instance. Short functions are easy. Long functions are harder. I will often fallback to a line-based blame output. I'll have commits that touched each line in a file (git/svn/hg blame) last, and work from there. Most often I'm interested in only the latest commit to have touched something -- if that commit was a whitespace fix (e.g. change a line ending), I'd have to recursively look back from that point. If I needed anything fancier, I would need a program to produce snapshots in time, and then track where the function is with an index (like cscope). But that's language specific. It is an interesting problem. I think searching by method has its uses, but the syntactic scope of where a change was made is not always relevant, or not always possible to track, so I feel the tools haven't pushed that feature forward too much. In Javascript for instance, you have these closures all over the place, and they are often anonymous -- so you can't fully rely on names, really to narrow down to only what you want to find. I feel that most SCMs throw the towel and give you generic "region-aware" history, rather than an informed history search based on program structure. It is interesting though, that when you are staging commits, the tools will often pull out the name of the function/class in which a hunk is to be applied, even if that line is outside the hunk. So there is value in extracting context. Gitk will give you the same thing when looking at the history (it will print the name of the object/class/function before each diff hunk). I'm not sure you can search on it though! -- or at least I don't remember doing this. You could ultimately look through the history with a visual tool, and let gitk extract the function name for you, but that's a lot of clicking.
I would use git blame to show changes to this file. This would work as long as this method hasn't been moved from one file to another. Github provides a decent UI around this, where they show the list of commits that have made changes to a file. However, this requires clicking on each commit separately, and navigating to the method in question within the list of changes brought about by that commit.
I would typically check to see if the file has been changed on a commit, and then git diff those changes to see if the method is included.
I would use the git history of the containing file in IntelliJ.
Looking at the history of the file where the method is placed in.
I would use options to limit the diff output of git log.
I would also look at the places where the method is used to see if the change breaks something. Also unit tests come in handy here.
git log -L :&lt;funcname&gt;:&lt;file&gt;
1. Checkout the branch with the changes (if there is an extra branch for that changes) 2. git diff from branch to develop or git diff between the commit with the changes and the commit before 3. Have a look at this method
I would use revision history and click through revisions.

Q2.3 Using source code history, how would you find changes to this method o...

Run 'git blame' on the file, get the 'git commit SHA' for the function, and 'git show commit_sha'. Most functions tend to be part of the single git commit. Or only 2-3 commits (modified the source code), in which case I'd see all commits one-by-one. I have not encountered the case where a function was modified by more than 2-3 commits, say 10, but if that's the case, I would probably look at the git history of the lines I am interested in. I am very unlikely to see all 10 git commits.

This can be more challenging, especially if the code has moved across files. I would need to establish the provenance of the code, then look at changes to the specific files. I would need to then read the code and understand how it changed.

This occurs in circumstances where the person responsible for this method has left (this is bad practice, eg. silo'ing). Then we would dig into the code history for this one method by looking at changes to the file itself. One method would not exist in isolation because they typically should be no longer than a screen's length. So generally the entire file and dependent files are also investigated. Things are trickier when the code doesn't link up in git, for example in microservices the API calls are difficult to connect and trace in git. For this we had a custom tool that linked calls (similar to Google's Dapper but not as nice) where we could trace the history of success/failures. From those metrics we dig into the code history to find a correlating change.

Have a look at the ticket connected to the PR. If done correctely, it should describe what should be done. However, if the ticket is done sloppy, I probably would have to contact the author or maybe the product owner

I would check the diff of the file and scroll to the method

As I said, I never had to do this. But if I had to do this, I probably would check the log of that particular file along with the changes made to it. git log -p command for example.

I use GitLens which makes it easy to find the exact commit and, hopefully, read a good summary on it.

You would need to check the "Blame" option for the file where the method is and check the commit that generated the last change. Then do the same for every deeper step you would like to go. You could guess the reason why it was changed with the commit message. In any case, it's not trivial.

use git blame and see which lines come from which commits.

Look at the file history for the last X commits.

blame-&gt;move to revision-&gt;blame recursively to rebuild lineage

I would search for changes to `this file` using gitx.

git log an grep for method if the norm is to mention this mention (unlikely); otherwise git blame (just looked up usage again in a blog since I use it so infrequently). I would figure out a person who last changed it and talk to the person if I can -- getting to the person is critical for me since I don't want to waste time making up a story about a change sequence that isn't true in reality (intent is important and is missing entirely from the source code repos).

I use git log -L to review a particular method name. Of course, this relies on the function name and path remaining constant.

Use the annotate feature in IntelliJ and klick on the commits in the method.

I'd use the bitbucket history of the file the method is in or the history of my IDE to see the changes commit by commit.

If the method is still in the same file, I would look at the diff of the file. But this approach is not really leading the way. The method could have moved from one file to another or the diff of the file is just too confusing because several places were adjusted at the same time in the course.

Don't know how, but would be great to be able to do it easily! I would probably look at commit messages and issues that I would expect to have impacts on said method.

Q2.3 Using source code history, how would you find changes to this method o...

Don't know, I'd probably check the file history.

using a diff tool

Aswing for a diff of the file the method is assumed to be included. Changes will be shown, if method is there. If method was (re)moved from file, I'll have a problem...

(graphical) diff tool

Use `git blame` to see the most recent changes to the relevant lines of code within that method and step through to the most recent change listed there.

Reading the blame of the method and jumping back and forth in history of the method. To the question blow: linting, automatic formatting and tests at different levels would ensure that the change is valid.

Unfortunately, it is not easy to directly and explicitly search changes in methods or variables or classes. I would search the diffs of the related files.

compare the file diff to the latest commit and find the changes for this function.

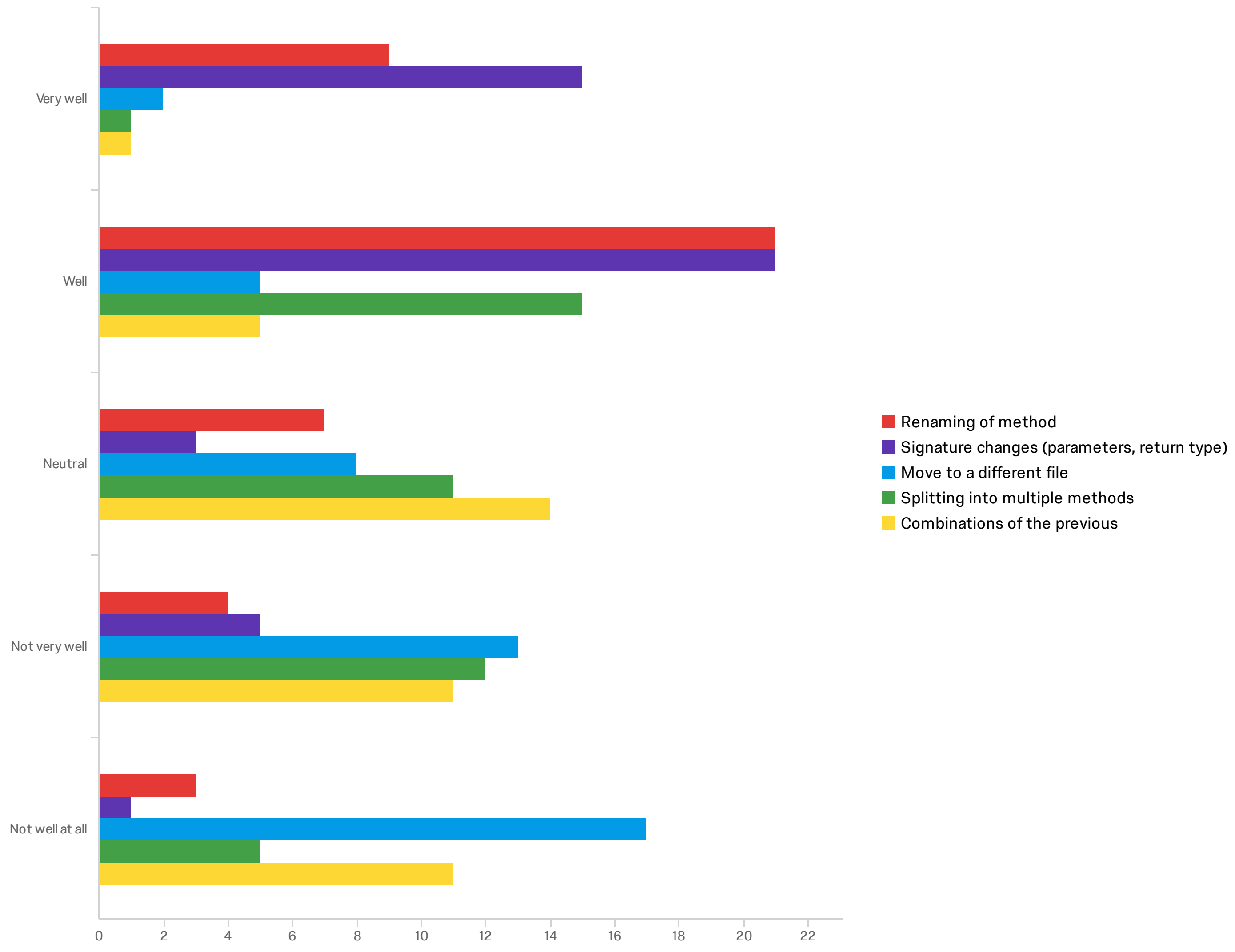
That´s a good question. Actually i would navigate to the file and open up the file git-history. Then i have to browse different versions to investigate the changes. It´s not the funniest job but it´s possible.

I would try different approaches: * First I would look at the source code history of the file that contains the method in order to understand the changes * If the first action does not result in a better understanding I would search for the method name in the descriptions of source code history

In IDE "show history" for the file, searching for the commit changing this method (by the latest change date of the specific lines), then using the commit message that hopefully includes a ticket ID.

I would look at the history for the lines of the method, for example using IntelliJ's "Show history for selection"

Q2.4 - Q2.4 How well would your strategy cope with more complex structural changes, e.g. method renaming, moving of a method, refactoring?

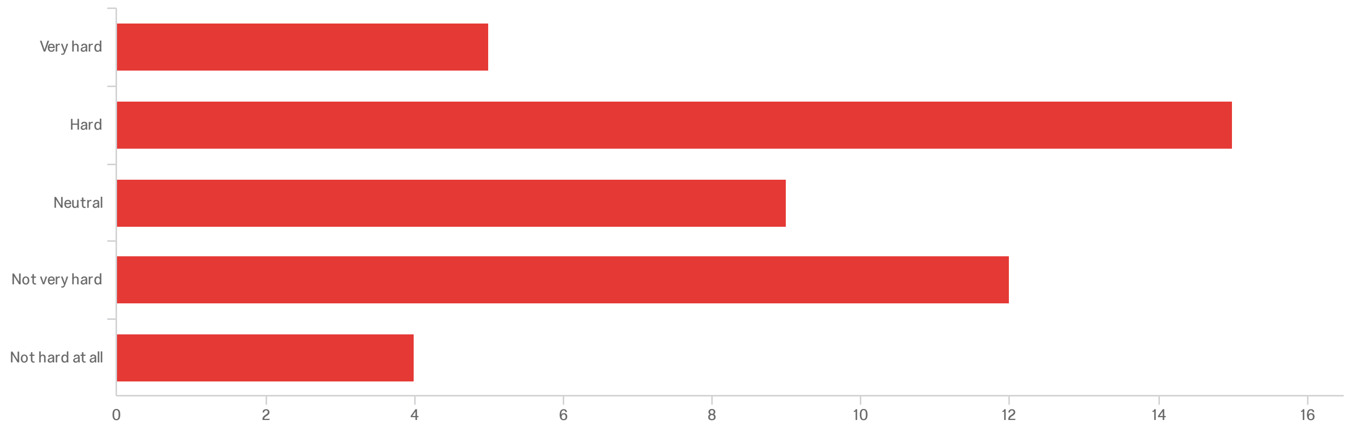


#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	Renaming of method	1.00	5.00	2.34	1.11	1.22	44
2	Signature changes (parameters, return type)	1.00	5.00	2.02	1.02	1.04	45
3	Move to a different file	1.00	5.00	3.84	1.17	1.38	45
4	Splitting into multiple methods	1.00	5.00	3.11	1.07	1.15	44
5	Combinations of the previous	1.00	5.00	3.62	1.07	1.14	42

#	Field	Very well		Well		Neutral		Not very well		Not well at all		Total
1	Renaming of method	20.45%	9	47.73%	21	15.91%	7	9.09%	4	6.82%	3	44
2	Signature changes (parameters, return type)	33.33%	15	46.67%	21	6.67%	3	11.11%	5	2.22%	1	45
3	Move to a different file	4.44%	2	11.11%	5	17.78%	8	28.89%	13	37.78%	17	45
4	Splitting into multiple methods	2.27%	1	34.09%	15	25.00%	11	27.27%	12	11.36%	5	44
5	Combinations of the previous	2.38%	1	11.90%	5	33.33%	14	26.19%	11	26.19%	11	42

Showing rows 1 - 5 of 5

Q2.5 - Q2.5 Using current tooling support, how hard is it generally to trace changes to a specific method?



#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	Q2.5 Using current tooling support, how hard is it generally to trace changes to a specific method?	1.00	5.00	2.89	1.18	1.39	45

#	Field	Choice Count	
1	Very hard	11.11%	5
2	Hard	33.33%	15
3	Neutral	20.00%	9
4	Not very hard	26.67%	12
5	Not hard at all	8.89%	4
			45

Showing rows 1 - 6 of 6

Q2.6 - Q2.6 Given your answer to the previous question (Q2.5), what makes this hard or easy?

Q2.6 Given your answer to the previous question (Q2.5), what makes this har...
If you use a good IDE with Git Plugins it isn't too hard
Accessing history for a specific part of a file works pretty well. Problems only arise once the changes go beyond the scope of the originally identified section either through involvement of different files or major structural changes. This makes partial file history basically useless if what you are looking for is very old and there were multiple other changes to the file you are working with.
See previous answer. I can only surmise that it is not easy because correctly defining the scope of a method automatically without false positives and false negatives is a hard problem. Each language has its own grammar, there can be syntax errors which would throw off counting scopes, or even preprocessing mish-mashes of various files. Some methods/functions don't have names. The golden standard seems to be to pick the line in the source file which "looks" like some sort of header. (I've now realized at this point that we may not be on the same page as to what would be considered a method).
Most often, methods don't get moved across files, in which case it is easy. In the cases that they do, it becomes hard as the forms a discontinuity in the source code history.
There is not a specific way to track this information with a Git tool. You have to track the file and mentally investigate changes in the file.
Currently using git and this has file based granularity.
History is on file level makes it hard.
Most of the time, Git repositories are used. Git does not track changes but uses snapshots.
It might affect parts of the code that are not obvious to spot
git log -L ....
git or bitbucket show me all changes that being made. So it should be very easy to trace changes.
Refactoring is hard to track. Without refactoring it's ok.
Function can be split into multiple functions or moved to another file. These cases make it hard for 'git blame' based history browsing.
It's either easy (the method hasn't undergone complex changes) or challenging (the method has undergone complex changes). When it is challenging, it is VERY difficult.
It's very difficult in distributed systems because IntelliJ and other IDEs aren't great at remote debugging. Combine this with microservice architecture and the problem explodes. Calls occur between different repositories and the github search doesn't link these well. However, with good naming practices and logging behaviour, it can be made more intuitive.
This depends on whether the PR is connected to a ticket. If so, it also depends on how detailed the ticket is written
The diff in bitbucket and github are pretty good.

Q2.6 Given your answer to the previous question (Q2.5), what makes this har...

Certainly git is not designed to do this, but usually developers don't move around methods or functions much i guess. So file level history is good enough. At least as far as I'm concerned

GitLens shows it all

Last change is relatively easy: "Blame" and find the file within the commit. The older the change is, the harder it gets. If the method has change file or name, it becomes more complicated. In general, every additional change you have to trace back, more chances you have to get lost on the way.

git blame does not show the blames being replaced by subsequent commits. And if the method got moved, or modified too much, git blame will get lost

When the method moves elsewhere you have to change to that other place...

Most of the tooling I'm aware of works on file and line granularity, but mostly ignores function boundaries in changeset metadata

It is hard using gitx. It is more easy generally using annotation (git history in intellij). But if the method is moved then much more difficult.

source code repo tools don't have lang semantics, so somehow attempting to capture 'method' in git command line options is just asking for a bad day.

The change information at sub-file level is difficult to identify and track. Like the previous questions identified, method changes like renaming and splitting are difficult to track.

Just go through the history of that file and look at the method. Don't know why it should be hard

Well, it's not that hard since changes are clearly marked in the history of my IDE if you one commit after another.

Commits refer to file changes without any further reference. One would have to tag a piece of code, be it a method, class, or block somehow during the commit, so that any associated changes could be identified across files and independently of other changes in a commit.

You just can't do it AFAIK :D

Depends like described in Q2.4. if method is not renamed, moved to another file than it should be possible to trace changes.

Sophisticated history support in SCM/SCM integration in IDE (we use git)

It's not very hard because it's possible through decent IDE support to step through different git commits but could definitely be made easier.

Not knowing the semantics. Not being able to visualize the change of e.g. a method renaming or split in a graph.

The is no direct or explicit abstraction to method/class levels.

Hard: - most of the time a PR/Commit contains more then a single change to the method - no clear way of seeing the impact of a specific change if not explicitly grouped in a commit - developer needs discipline to commit often and in terms of logical changes Easy: - if a commit contains only the renaming or signature changing of a function it should be easy to trace back the changes

It´s just time consuming but not that hard. I´d prefer a better overview with a timeline with specific filters and stuff. Not a single side-by-side view.

The tool I use the most is Git and as far as I know you can only narrow down the search results by specifying a path. Classes, methods, properties are not supported.

Q2.6 Given your answer to the previous question (Q2.5), what makes this har...

The more stable a method is from external perspective (keeping its signature and general purpose) the easier inspecting its history becomes.

Q3.1 - Q3.1 In the above version history, how would you identify the commits in which the method of interest has changed? Please describe your strategy briefly.

Q3.1 In the above version history, how would you identify the commits in wh...
Use selective history for the lines in question, which reduces the number of commits to those with changes in these. Obviously, this has the downside that once the method moves around more than a few lines, the history is losing track of it because we are only referencing line numbers. Then, the commit log might give an indication of where the method came from. Continue the search there...
git blame
the comment links to an upstream issue. I would read that first. I will state upfront that I don't like the github interface. I would create a temp branch, set to the revision against which the pull request is made. Then I'd run git blame on the file. And spot the revisions affecting lines of the method of interest. Then I would inspect these commits. If the function was refactored from a different file, i'll reach a point where the function is added to CommonUtils.java. Then I'd have to git grep to find where it came from (hopefully the removal of the initial function is in the same commit as the addition). If CommonUtils.java was called something else before, I'd have to find the corresponding delete operation.
Click through each commit and search for the method in question.
I would take a look at the commit messages and any time-periods where I know people were working on related features where this method is important. I would specifically look for the development of that method, where it would have had to have been changed, by finding a relevant git commit message.
You could diff the first and the latest version and then check the lines of method for commit hashes in which they got changed, then go to these commits and check if there are earlier annotations on these lines and so on.
Use google and found command: "git log --follow -p -- file"
I would run git log -p -- src/main/java/com/puppycrawl/tools/checkstyle/utils/CommonUtils.java and search for all chunks affecting CommonUtils.hasWhitespaceBefore.
Above the code field you can select changes from all or certain commits.
If "git log -L ..." is no option here I would possible do samples of different dates looking at the original file and comparing it (as the method is quite short)
I would check the commit messages of the above version history to find the correct commits.
I would try to figure out of the comments if the method was concerned. If I wan't to be sure I would have to check every commit.
I would use 'git blame' to get the last commit/change on the lines I am interested. I would not try to find 'function level' changes from the entire commit history to CommonUtils.java. As you pointed out, 47 revisions are too many to go through.
Usually I would use binary search to walk through the change set. The descriptions could be useful in eliminating unlikely changes.
I scrolled down to the bottom of the commits and found the commons-3 commit which looked interested but was actually not directly related. I clicked on that diff and looked for the same file and saw that it was only libs and constants changed. I then scrolled through all the irrelevant minors and version bump commits up towards the top. I found the most recent related commit which shows improvements to the whitespace method. I searched for the file CommonUtils.java (using the browser search for text) because the smaller changes just to the method call were arranged at the top, making it difficult to find what I was looking for. Ultimately, I'm looking for just the CommonUtils.java file and there's too much garbage noise because it's such a commonly used file.

Q3.1 In the above version history, how would you identify the commits in wh...

Looked at the commit messages and randomly clicked through commits. Not very easy

I would click through each version and check if there is a change

git log -p And then search for the function name

I have no idea how I would do this.

In the PR, click "View" file, click "Blame", search for "hasWhitespaceBefore", check commit next to the method. For older commits, click "View blame prior to this change". Only commits older than 29th August 2015 are related to this method. But the method comes from another class (Utils.java), that would be more difficult to trace.

Open each one of them and manually inspect. ;-(

It's annoying, but looking at each commit if stuff changed in that method.

Recursively looking up blame on method lines to see when they were last modified

I would first use a better tool to look at history of the file. Like gitx, then I would just scroll through all the changes to the file if I had no idea why the change was made.

Oh boy. Well, if the git interface allowed me to expand all the diff snippets then I would expand and search for method name. As is, I would have to click through each one (doesn't seem like this method name appears in any of the comments, plus could miss some this way), so for completeness I would expand each diff snippet and search using the browser page text search).

Basically, I would dig through the history to find the method in each commit. I don't know if there is any tool that allows me to track the function across commits.

Go through every commit and check the method

I don't knooow! Click every single commit? Damn... you got me with this one!

Click on each commit and search with Ctrl+F for method name.

Look at the commit messages and diffs for each.

I'm trying to figure out by looking at the commit description if the change affects the method I'm interested in and check the diff if so.

split the time to equals parts, compare with revision 24, then revision 12 or 36

Having a look at the commit comments to get a knowledge of what is the change committed is all about. In Git there is an option to search in comments or search for logges changes in a specific file. If there is a hint in the comment, I would have a deeper look into the specifit commit. If not, one would probably have to have a deeper look into every of the 47 commits to investigate...

1. commit message 2. linked issues 3. diff tool

Look at each commit diff to determine if the method was changed.

Not at all.

Q3.1 In the above version history, how would you identify the commits in wh...

I sincerely have no idea. I usually don't find myself in such complex situations. Personally, I would need to search commit-by-commit. However, as I said, I usually don't have to do this, so I never explored more efficient solutions.

open the different commits and check manually

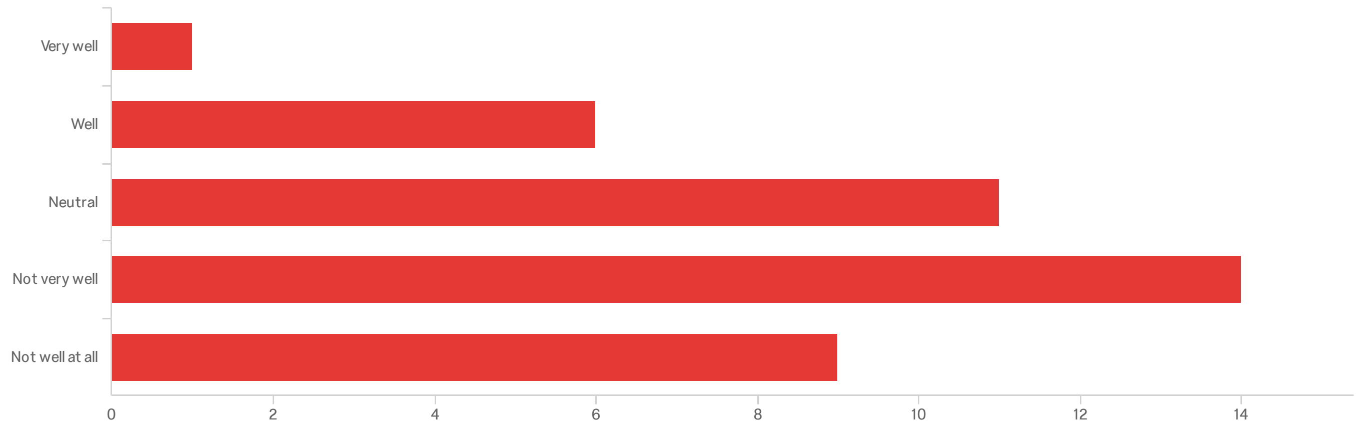
I´ll have to pick some previous versions to hopefully find commits related to this single method.

Looking at the version history I would try to identify changes related to the 'hasWhitespaceBefore' method by looking at the commit messages or issues. If this does not lead to a positive result I would probably write a simple script in order to step through the source code history and generate a diff of the affected file and only print out the relevant parts for method 'hasWhitespaceBefore'.

On web interface: Click through history? Search for "whitespace"/"blanks" in commit messages? Or using IDE, as described before.

I would first check if the function existed already in the very first commit. If not attempt some divide and conquer mechanism to see when it was introduced. From there I would go forward through all commits.

Q3.2 - Q3.2 How well do existing tools support identifying these changes?

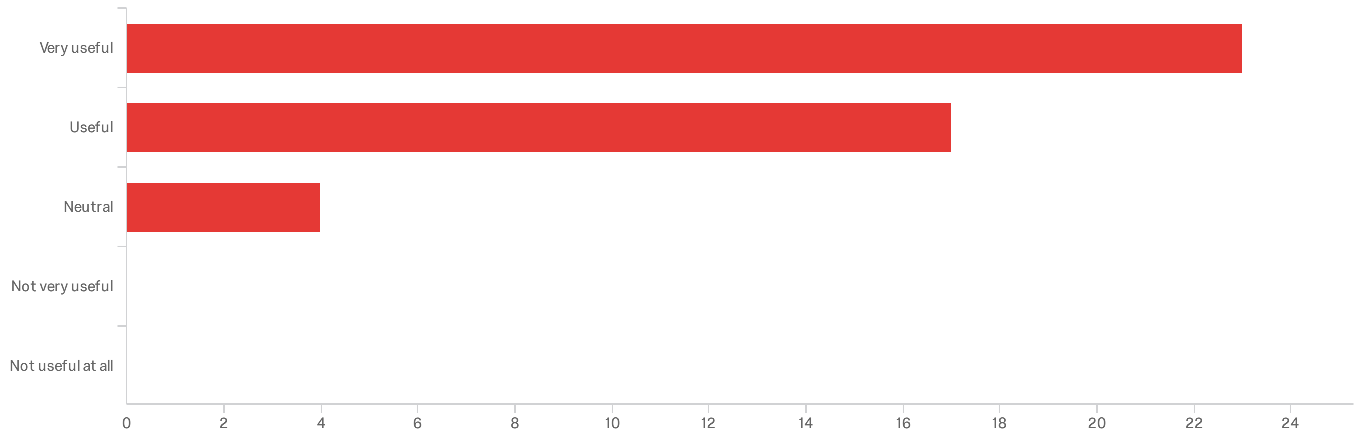


#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	Q3.2 How well do existing tools support identifying these changes?	1.00	5.00	3.59	1.06	1.12	41

#	Field	Choice Count	
1	Very well	2.44%	1
2	Well	14.63%	6
3	Neutral	26.83%	11
4	Not very well	34.15%	14
5	Not well at all	21.95%	9
			41

Showing rows 1 - 6 of 6

Q3.3 - Q3.3 How useful would it be to have support for a more semantic history in this scenario (e.g. history for this method or class only)?

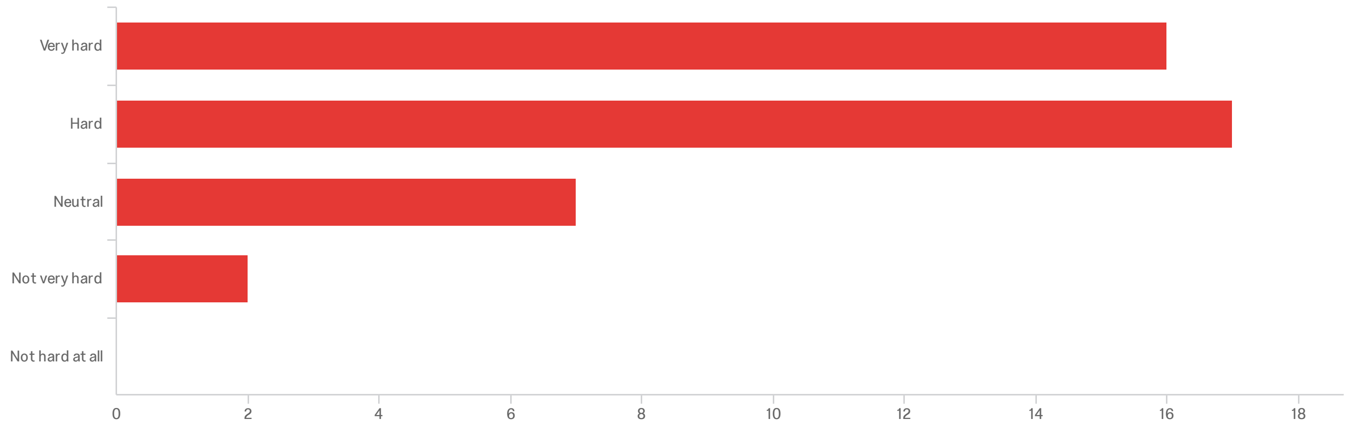


#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	Q3.3 How useful would it be to have support for a more semantic history in this scenario (e.g. history for this method or class only)?	1.00	3.00	1.57	0.65	0.43	44

#	Field	Choice Count	
1	Very useful	52.27%	23
2	Useful	38.64%	17
3	Neutral	9.09%	4
4	Not very useful	0.00%	0
5	Not useful at all	0.00%	0
			44

Showing rows 1 - 6 of 6

Q3.4 - Q3.4 How hard would it be to find the first commit for the given method and whether the method was really created then or if it was moved there from somewhere else (e.g. through a file renaming, or through a refactoring)?

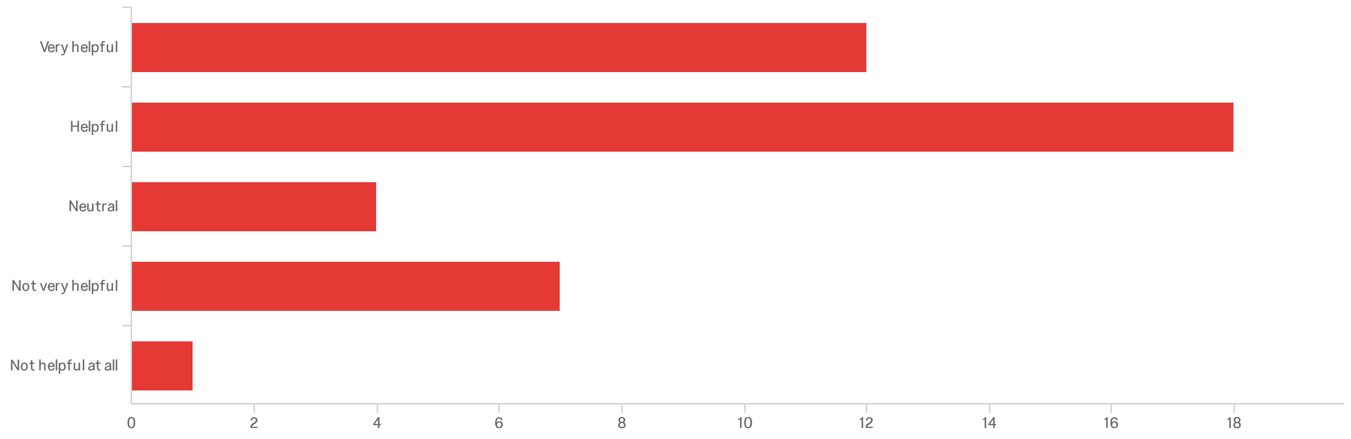


#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	Q3.4 How hard would it be to find the first commit for the given method and whether the method was really created then or if it was moved there from somewhere else (e.g. through a file renaming, or through a refactoring)?	1.00	4.00	1.88	0.85	0.72	42

#	Field	Choice Count	
1	Very hard	38.10%	16
2	Hard	40.48%	17
3	Neutral	16.67%	7
4	Not very hard	4.76%	2
5	Not hard at all	0.00%	0
			42

Showing rows 1 - 6 of 6

Q4.1 - Q4.1 Consider again the described situation of being faced with a pull request for a change of a method. How helpful would you consider the information above for getting a better understanding of the method and its history?

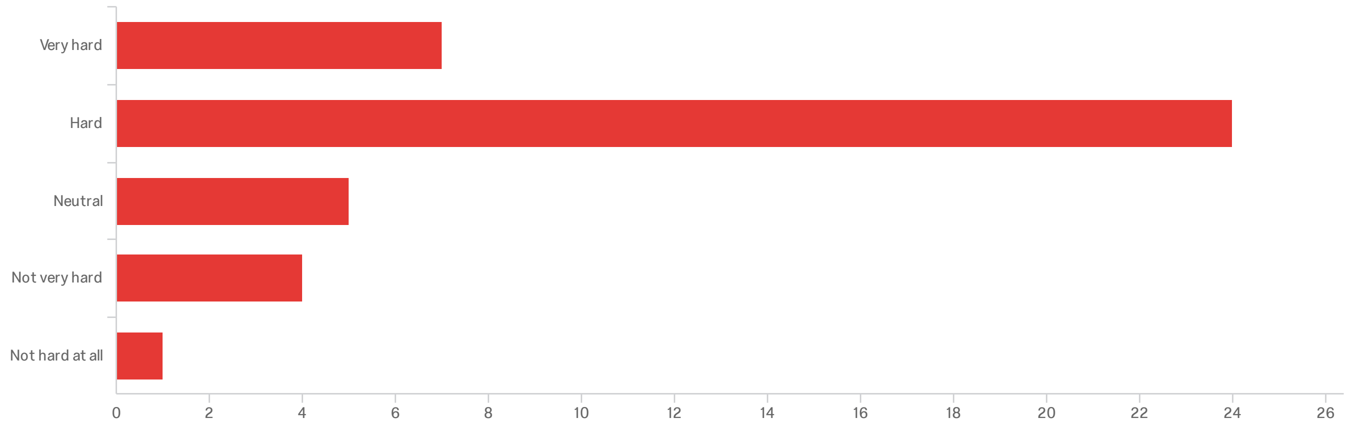


#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	Q4.1 Consider again the described situation of being faced with a pull request for a change of a method. How helpful would you consider the information above for getting a better understanding of the method and its history?	1.00	5.00	2.21	1.10	1.22	42

#	Field	Choice Count	
1	Very helpful	28.57%	12
2	Helpful	42.86%	18
3	Neutral	9.52%	4
4	Not very helpful	16.67%	7
5	Not helpful at all	2.38%	1
			42

Showing rows 1 - 6 of 6

Q4.2 - Q4.2 How hard would you consider retrieving information on the history of a method with the above level of detail?

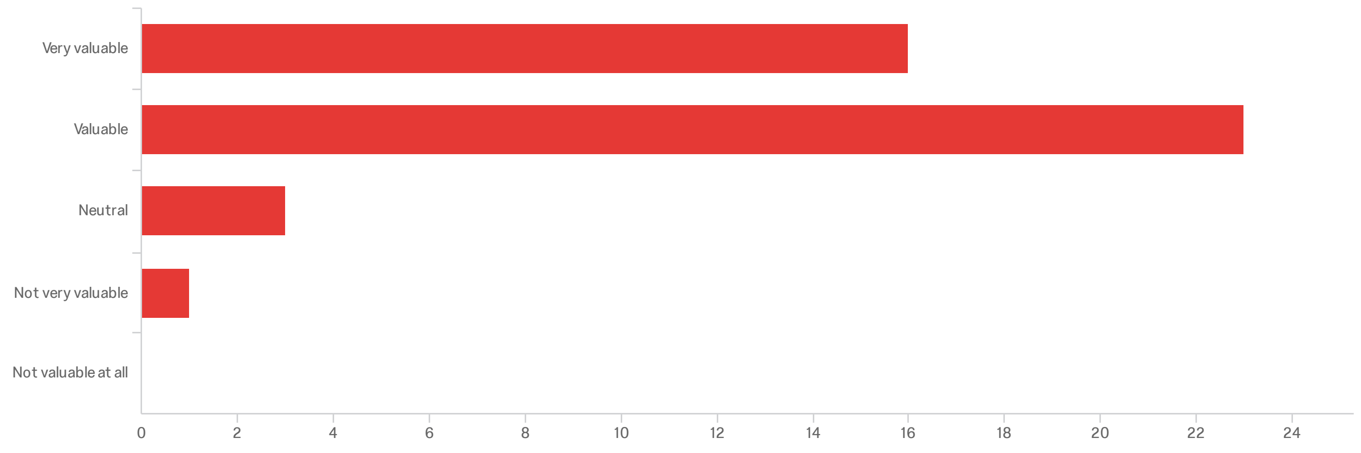


#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	Q4.2 How hard would you consider retrieving information on the history of a method with the above level of detail?	1.00	5.00	2.22	0.92	0.85	41

#	Field	Choice Count	
1	Very hard	17.07%	7
2	Hard	58.54%	24
3	Neutral	12.20%	5
4	Not very hard	9.76%	4
5	Not hard at all	2.44%	1
			41

Showing rows 1 - 6 of 6

Q4.3 - Q4.3 If a tool could generate information in the fashion of the above on any method or other code unit, how valuable would you consider this tool?



#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	Q4.3 If a tool could generate information in the fashion of the above on any method or other code unit, how valuable would you consider this tool?	1.00	4.00	1.74	0.69	0.47	43

#	Field	Choice Count	
1	Very valuable	37.21%	16
2	Valuable	53.49%	23
3	Neutral	6.98%	3
4	Not very valuable	2.33%	1
5	Not valuable at all	0.00%	0
			43

Showing rows 1 - 6 of 6

Q4.4 - Q4.4 What other information that is not in the descriptions above would you consider valuable?

Q4.4 What other information that is not in the descriptions above would you...
Information about whether content has only been moved from one class to another or gotten a real change. In the above example, I had to compare the method contents every time because it was marked as new content in the new files. For lines in that file, this may be true, but the method content itself didn't change at all.
I'm a bit torn here. On one hand I can appreciate the value that such a tool could provide. On the other hand, I don't think that in a real scenario i would want to trace such a small function (~10 LOC) back to the project's inception. It seems like more work than it's worth. If the question is "Should I merge this PR?", then I'm really trying to evaluate if the PR is a strict improvement over the previous commit. I'm not convinced going back to the roots of the project is necessary to make that decision. In other words, in what way does the function's provenance help assess whether the PR change is needed? There are other things that need to be done to evaluate the previous PR. For instance, there might be other calls to that function hasWhitespaceBefore that should also be refactored. There might be tests to worry about. I might be interested in finding commits that have modified both a test and the given function -- that might be a more compelling use case. Another thing to note is that the PR changes the semantics of what characters are allowed at the start of the line. isWhitespace() is more inclusive than just checking for ' '. Are callers of the modified functions aware that the semantics have changed?
It would be great if it was part of a git command line :) That is something that I would use to search for a method and its changes for sure.
Who made the changes and why (but this is available in the linked commits so not really missing).
Not just the information when and what changed but why the changes being made.
There is enough info here.
Short summary of all the places that are calling this method, so I could determine how important it is when scoping out changes.
I don't work on java a lot. Mostly C or C++. These languages don't have a restriction for giving same name file to the public class in the file. So I never had to face this problem.
I'd love to be able to specify some inputs and outputs to act as tests for a method. If it could show me how each version of the method would function for those; for instance, if I found a bug where it broke on a particular input, I could easily see if it were always broken or if some change added the bug.
I would like to have the issue number and design decisions that required the change.
The actual history of the method itself seems less relevant to me than what the method does right now (why/how would I need/use history; to blame people for bugs?). It seems that one useful bit of history is to find methods that were co-changed with this one. i.e., if there is another method that is a dependency or that somehow uses logic that is similar to this method, then knowing that they were changed together (or not!) would be useful. i.e., either to inform a change I would to make, or to learn about other parts of the code that are relevant to this method (e.g., if I'm learning about this method/this part of the code). Clearly knowing the people involved is key (for me), which you left out above. Knowing that Bob and Jane made all these changes would be super helpful (because I know Bob's on Slack and Jane left the company, so I know who to ask about the implementation or if I want to change the method -- code ownership comes into play).
If it is not an API, the most important information for me is that the method will still be used.
Maybe it would be interesting to see where it's (been) used (in the past).
The call reference to this method which in turn has been refracted just because of this method change

Q4.4 What other information that is not in the descriptions above would you...

Method usage

no answer

Associated test runs/suites and bug reports. Dependency updates which may be have trigger source file changes. Discussions of PR reviews going into more detail of reasonings.

- references / how often and where is the method used before and after the changes

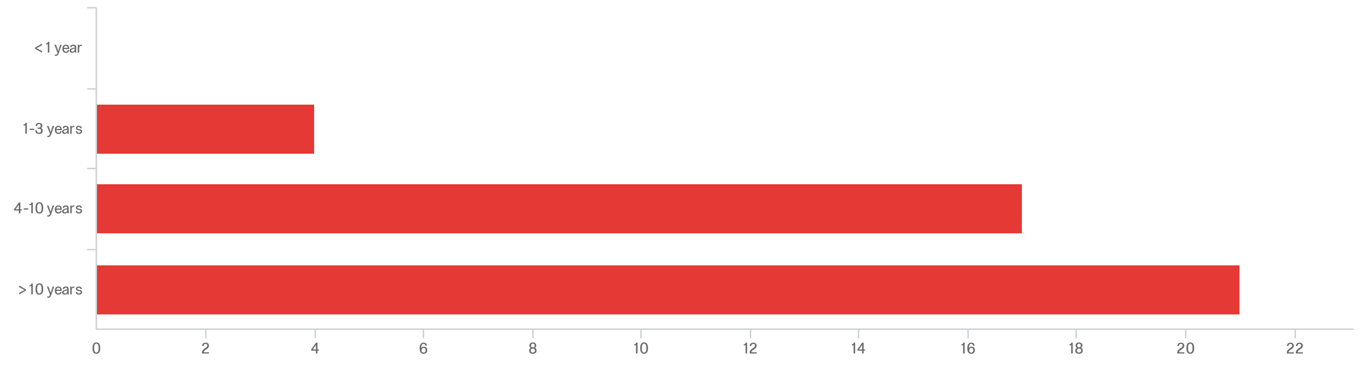
Having the information above would be a big improvement in the daily business.

The commit message associated with every commit. Normally the commit message should include a description why the change was made and also references to other tools (e.g. issue tracking system, ...).

Who worked on the method within which time spans. Associated issue ids. Changes of signature. History of Javadoc of this specific method.

If the programming language supports strong typing like this example, I think the signature (parameters and return type) could be retrieved and would help a lot.

Q5.1 - Q5.1 How many years have you been programming?

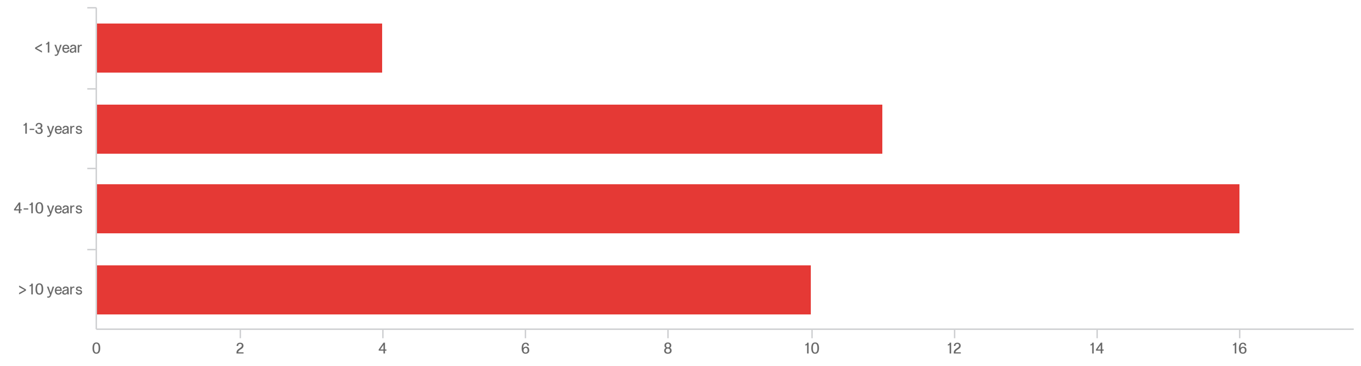


#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	Q5.1 How many years have you been programming?	2.00	4.00	3.40	0.66	0.43	42

#	Field	Choice Count	
1	< 1 year	0.00%	0
2	1-3 years	9.52%	4
3	4-10 years	40.48%	17
4	> 10 years	50.00%	21
			42

Showing rows 1 - 5 of 5

Q5.2 - Q5.2 How long have you been working as a professional software developer?

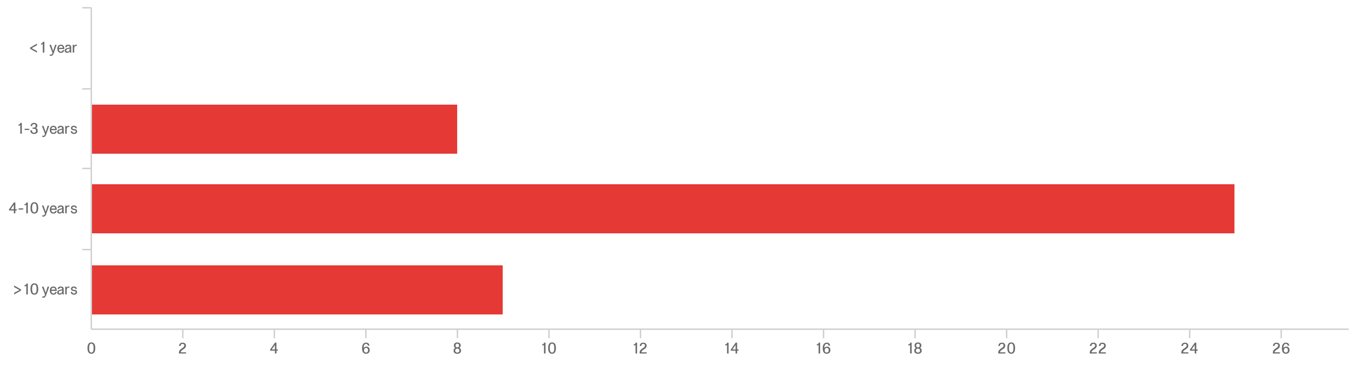


#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	Q5.2 How long have you been working as a professional software developer?	1.00	4.00	2.78	0.92	0.85	41

#	Field		Choice Count
1	< 1 year	9.76%	4
2	1-3 years	26.83%	11
3	4-10 years	39.02%	16
4	> 10 years	24.39%	10
			41

Showing rows 1 - 5 of 5

Q5.3 - Q5.3 How many years have you been using source code version control?



#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	Q5.3 How many years have you been using source code version control?	2.00	4.00	3.02	0.64	0.40	42

#	Field	Choice Count	
1	< 1 year	0.00%	0
2	1-3 years	19.05%	8
3	4-10 years	59.52%	25
4	> 10 years	21.43%	9
			42

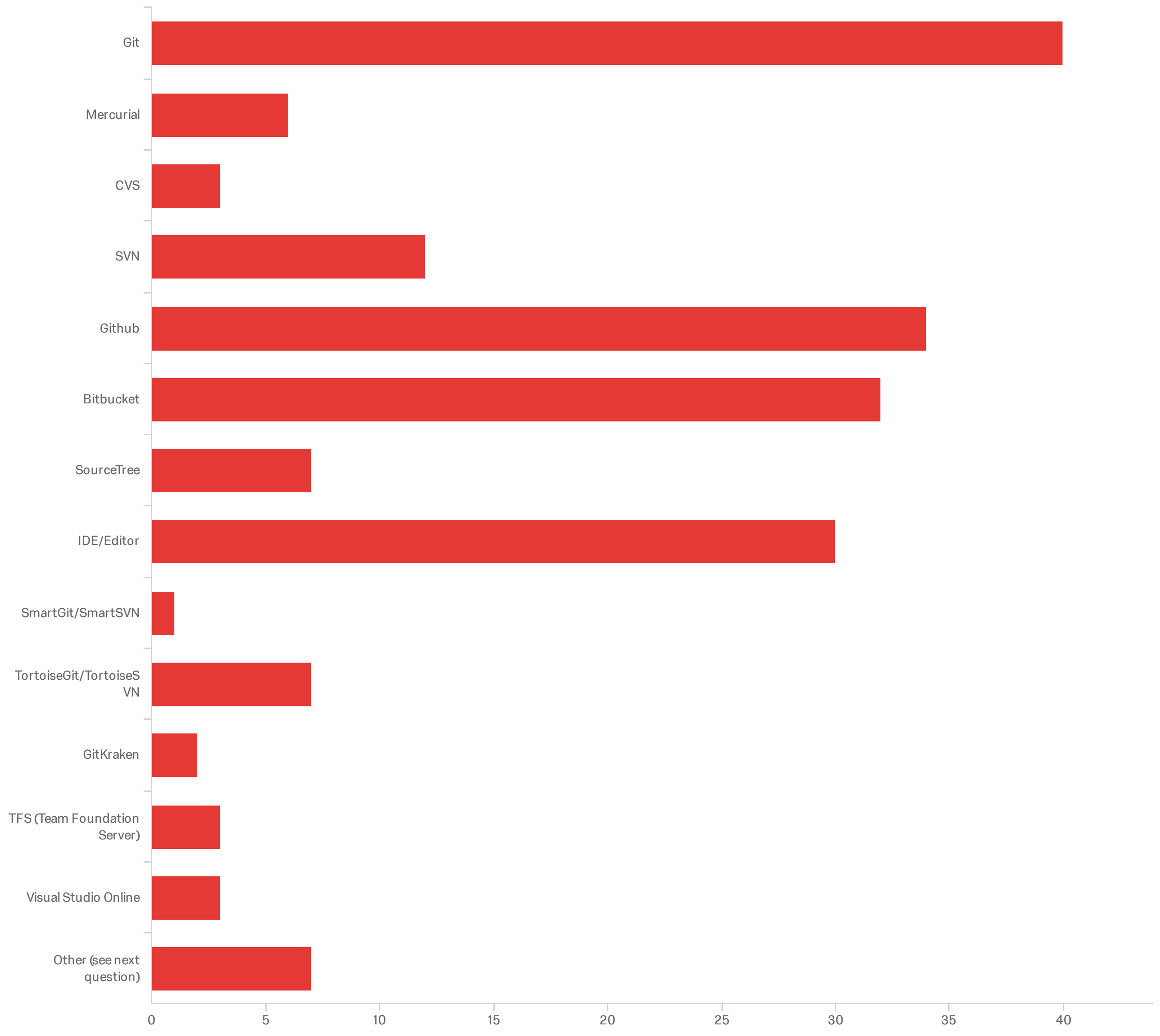
Showing rows 1 - 5 of 5

Q5.4 - Q5.4 What is your current job title?

Q5.4 What is your current job title?
software engineer
Postdoctoral fellow at UBC -- Data Science Institute
Software Developer
Backend Developer
IT Specialist
CTO
Software Developer
Graduate Student
Principal Consultant
Software Developer
IT Consultant/Fullstack Developer
Diplom-Informatiker
PhD student
PhD Student
Platform Engineer
Softaware Developer
Software Developer
Node.js Engineer
Software Engineer
Graduate student
Projektleiter Software & Consulting
PhD Student

Q5.4 What is your current job title?
Senior software product engineer
Assistant Professor
Grad Student
Software Developer
Software Developer / UI/UX Designer
Developer
Software Delevoper
Web Developer
Software Developer Web
Senior Software Developer
senior software consultant and architect
Senior Frontend Engineer
Permanent researcher
Professional Software Developer
Software developer
Software developer
Fullstack Developer

Q5.5 - Q5.5 What version control systems and tools do you use? Please select one or more options.



#	Field		Choice Count
1	Git	21.39%	40
2	Mercurial	3.21%	6
3	CVS	1.60%	3
4	SVN	6.42%	12
5	Github	18.18%	34

#	Field	Choice Count	
6	Bitbucket	17.11%	32
7	SourceTree	3.74%	7
8	IDE/Editor	16.04%	30
9	SmartGit/SmartSVN	0.53%	1
10	TortoiseGit/TortoiseSVN	3.74%	7
11	GitKraken	1.07%	2
12	TFS (Team Foundation Server)	1.60%	3
13	Visual Studio Online	1.60%	3
14	Other (see next question)	3.74%	7
			187

Showing rows 1 - 15 of 15

Q5.6 - Q5.6 If you selected IDE/Editor in the previous question, please specify what IDE/Editor (and if you know, what underlying version control system) you are using. If you selected "Other", please specify what other tools you use.

Q5.6 If you selected IDE/Editor in the previous question, please specify wh...
Qt+Git
I'm confused by the counterpart of this question. I use emacs, mostly. I've used Eclipse, and Sublime, and IDLE for python in the past. I use vim once in a while, but only to shoot myself in the foot or to insert a bunch of "^qq:" at the top of each file. I've only _heard_ of Visual Studio Online. Looks decent, i'd be keen to try it. What underlying version control system ? I'm not sure what that means. Do you mean under which SCM the IDE is developped (I don't know)? for most editors I use, I install plugins for most common SCMs. I have also used CVS, SourceForge, and SourceSafe in the past, but these tools have definitely lost their edge today. I feel ancient now. There's perforce that comes to mind, but it's not on the list.
- VScode uses git - the IDEA suite of IDEs from JetBrains provide an excellent feature for managing current source code changes called "changelists"
IntelliJ (with git)
- Atom (git) - vi - Eclipse (git) - IntelliJ (git)
Vim
Eclipse (git, svn), IntelliJ IDEA (git, svn)
IntelliJ IDEA
Using Eclipse, IntelliJ, Atom, and VS Code all along with git.
Eclipse with SVN plugin. Will change in the next weeks to Bitbucket.
VS Code (various including git) Visual Studio (TFS, Perforce, git)
IntelliJ, Eclipse Git but only through command line
VSCode
- VS Code - IntelliJ
Vim, ctags
VS code, GitLens
Intellij IDEA
Editor: sublime text

Q5.6 If you selected IDE/Editor in the previous question, please specify wh...

PyCharm, Atom

I use multiple IDE, Currently at work I am using Intellij, at home I use VS code.

Eclipse with all the version systems.

P4V and perforce.

IntelliJ

Php Storm

Jetbrains IntelliJ Visual Studio Code

IntelliJ, Visual Studio, Atom

PHP Storm & Visual Studio Code

IBM ClearCase , IBM RTC

IntelliJ

IntelliJ / Sublime (Underlying git)

Terminal and git with various helpers and configs.

IntellyJ IDEA and PHPStorm with Git

IntellijIDEA/Atom/Git

IntelliJ

Editor / IDE: - VS Code with GitLense - various IntelliJ products with both its "Local History" function and the Git integration Other: - The Git CLI itself (not sure if "git" included this)

Q5.7 - Q5.7 Do you have any final comments? Do you have any other ideas for tool support or systems to solve the general problems described in this survey and its scenarios? Is there anything else on your mind?

Q5.7 Do you have any final comments? Do you have any other ideas for tool s...
- informal / flowchart-like documentation is often more helpful than just the code for complex algorithms and structures. - reviewing code with unit/integration tests is easier than than just reviewing code. - we always should know who to ask: next to "git blame" sometimes I would like something like "how did something similar in this company"? (based on semantic similarity of code) often people solve the identical problems again, because they don't from each other. after the first commit a good tool could give a hint "are you looking for...?"
I do like the idea of being able to express rich queries on evolving code. I think that's a powerful concept, and one that could work across repos (or forks of a repo). I was thinking of use cases as I was filling in the survey, and I thought that it would be pretty cool to automatically merge patches that have been fixed in forked repos. If I find a bug in my own code, I could quickly determine whether it affects "descendent" repos, and vice versa. I could also look at a piece of code (a function , a class, a struct, etc.) , and compare it side by side with alternate implementations in other repos. However I think it's too big of a gun for inspecting the localized PR presented as a use case in this survey. I think a use case where a bug was reintroduced might be a more compelling example (it happens a lot!). There are other powerful use cases for a structured historical search during development (and less so patch approval). If I add a parameter to a function I would be interested in seeing what other parts of the code change along with that parameter. E.g.: "show me the code that changes when signature of method X changes". An obvious consequence of changing a signature is to change the callers of the modified function. A less obvious one is adding appropriate logging for the new parameter -- this is often forgotten, and added after the fact. I hope this is going to be a real tool! Would love to try it. I'm thinking even a prototype implementation of this as a wrapper script to the git commands that would be great to have at one's fingertips.
This was a long survey!
Being able to see who revised the method, and comment revisions would be nice.
I like the idea of being able to deep dive into a methods history, this would also be helpful for internal classes or fields that get moved from one file to another, but methods are definitely the best use case.
The described scenario is hard to solve with existing tools from my point of view. However I thought on my situations during the survey and I can't remember that I've needed such detailed information about one method over time. I'm pretty sure that I had these situations in the past but I think they are pretty rare. In other words: it is not something I miss in my daily work but of course it would be very helpful in these rare situations where method tracking is needed.
Merging the call tree with the historical view would be super useful. IDEs do this, for example when you rename a method, but the window is tiny and hard to read. It also only appears for the time when you're doing the refactoring and the historical view is difficult to retrieve.
I don't know method level history is very challenging considering the myriad of programming languages. I personally never had to encounter with the problems mentioned in the survey, may be because the tools that I use don't support this feature.
The history of a method helps to understand a pull request, but I usually don’t find the need to do so.
Thinking back, I dig a lot less in code version control systems than I thought. Generally it's to rollback or to find out who did what or at most figure out when something happened and what issues were involved. I'm not sure I would find rich method-level semantics that useful. But, rich meta data that is easier to get at would be helpful.
Use diffs to trace a method through history in a file. Use search to trace a method in different files.

Q5.7 Do you have any final comments? Do you have any other ideas for tool s...

Thank you, I realize now that I use VCS purely as an archive. Even with the pull request, the successful execution of the test is more important than the actual change.

No

none

I´d be great to generate a method-, class-, file-history/protocol in a human-readable way. Example: --- Method: ExampleClass::exampleMethodX - aka: DifferentClass::exampleMethodA, MovedIntoAnotherClass::exampleMethodB - created: 2017-06-03 by "john doe" - history: -- 2017-06-03 13:46: Moved to class DifferentClass (see diff) -- 2017-06-10 13:46: Return value has changed to Integer (see diff) -- 2017-06-12 13:46: Added new Argument "newArgument" of Type String (see diff)

Good idea with the semantic code history. But the example with "hasWhitespaceBefore()" differs from my usual code history challenges - in this case, I wouldn't consider the history shown very helpful.