Participant A

Experiment

1. A: Topology
    a. Manually found the commit in the history that introduced these lines and navigated to their browser to visit KAFKA-7523 and visited KIP-401.
    b. Manually looked through the history with a combination of using IntelliJ's "Show History for Selection" feature to find the newly added `addSink` overloaded methods. Navigated to their browser to visit KAFKA-4936 and visited KIP-303.
2. B: StreamsBuilder (with plugin)
    a. Toggled on commit highlighting. Began looking through the commits and their diffs from most recent to oldest commit, invoking the Jira issue fetch. Found KAFKA-12648 (908a6d2a) as the first important commit; was able to justify their answer without having to leave the IDE. Same process for the second important commit, continued to look through highlighted commits chronologically from most recent to oldest after this and found KAFKA-6958 (075b368d); able to justify their answer without having to leave the IDE.
    b. Immediately found the highlighted KAFKA-7027 (e09d6d79) commit in the history. Invoked the Jira issue fetch and visited the external link.

Interview

1. 4 years of experience (3 years non-professional, 11 months professional).
2. Occasionally or rarely looks at issues associated with commits and have familiarity with using Jira at work. Company does not have a convention of associating or explicitly linking issues with commits.
    a. "Basically there'll be a task saying, you know, implement or add this functionality. That will be the issue and then we'll go off, make a pull request, make a bunch of commits on our branch, pull request gets merged in, does not get linked to the issue, and then the pull request gets closed"
    b. "There's no tie-in from the issue to the commit history."
    c. "Whenever we do a pull request, we'll kind of mention the general nature of the issue. [...] We're not copy-pasting the title [of the issue] but we're pretty close so in that sense, if we see an issue and we think this issue made a difference we need to go back and check, then we'll go back through the pull requests and try to find one that kind of looks similar basically. It's not a good system."
    d. "I don't actually typically look at issues. The issues aren't generally helpful because a lot of the people on my team don't really use Jira. It's mostly my manager that uses Jira with the other managers. [The people on my team] get assigned stuff but they don't really interact with the issue at all. Some people are getting better at adding comments on the issue to say 'this is what I did' so generally I go through the commit history. If I am going through the history, then it's because something has happened in the code, like bug-fixing. I look at the file to see what's changed and when, like the change that a problem was tied to."

3. When looking through diffs or the code itself to see what change might be related to a problem, they consider added comments to sometimes be useful. For example, if there is a potential problem or a note addressing that something only works a certain way but is obscure, the author might add a comment explaining that. Checking to see if a variable was modified or removed or added, especially if a variable was removed. Focused on changes specific to the section of code they're looking at.
4. Appreciated the Jira integration within the IDE. Did not find highlighting as useful despite receiving the question set that might be more favorable for using the plugin. Didn't use diff metadata at all.
    a. *"I thought that the Jira integration was very helpful because you don't have to go to Jira to look at all the stuff and deal with Jira. It's just in the IDE which is nice, it's quick and fast."*
    b. *"Highlighting commits is nice when you're looking at the history but generally I try to look at a specific line or section. If I was trying to get a sense of major changes in a file, like say there was an issue that came up in a file and I'm looking for where it could have gotten introduced, then that would be very useful because it gets rid of all the changes that don't matter. [...] That would trim down a lot of the stuff." Describes a scenario where a change occurs and several changes happen afterwards, then a developer tries to go back to find out where or when it could have happened using the highlighting.*
5. Wishes their team would actually tie their Git history to Jira more. Enjoyed the highlighting.
    a. *"I liked the highlighting. It's nice to not look at so many commits. [...] Looking at the history without the highlighting is kind of overwhelming because it's just like 'oh my god, this is a flood of commits, what are we going to do?' but when you get rid of the less important ones, it's like 'okay, this is so much more manageable.' "*

Participant B

This participant thoroughly read the diffs for each commit they looked at, often jumping between previously seen commits and new commits to establish a relation between them. They also read the diffs in detail before settling on which commits to propose in their answers.

Experiment

1. B: StreamsBuilder
    a. Jumped through several commits in history, mostly chronologically, while reading the diffs for each commit. Checked many commits to look for potential candidates before proposing answers. They also looked through commits prefixed with "MINOR," not only KAFKA prefixed commits. They chose KAFKA-10379 as the first important commit and KAFKA-12648 as the second most important commit.
    b. Narrowed down the set of commits that changed the specified lines, examined them, and found the commit that actually introduced the overload with KAFKA-7027. Mostly got an idea of why the overload was introduced based on the Javadoc comments inserted as part of the commit so they didn't actually need to leave the IDE for this, even though there was no descriptive commit message. They also further explored the commit with KAFKA-12647 (908a6d2a) found from the first question to support their answer; this commit does have a commit message.
2. A: Topology (with plugin)
    a. Used IntelliJ's "Show History for Selection" feature to locate KAFKA-7523 and fetched Jira issue within IDE. They visited the external link to read the Jira issue. They also cross-referenced the changes in the diff with the Jira issue reporter's problem description/rationale for the change.
    b. Used chronology to locate the commit that introduced a new overloaded `addSink` method(s). Found KAFKA-4936 and fetched the Jira issue in the IDE.

Interview

1. 3 years of experience (0.15 years non-professional, 2.75 years professional).
2. Often looks at commits and issues. Has experience working in a project where each commit explicitly references an issue. The commits themselves are linked to work items. Mainly looking for the description in the issue, i.e. the person's explanation for what they were doing in a change.
    a. "When I look at the commits, if it's a questionable change, then I would look at the issue." They define "questionable" as something that is not that well explained. "When you find a problem in the code that leads to the bug, you wonder if the bug has always been there or was put there by something else."
    b. "The commit message is usually not sufficient for the more precise reasons behind the changes."
3. Mainly uses `git blame` feature in daily work but has problems with the blame showing you most recent changes that could have just moved the line in some negligible way. Mainly comments and adjusting of spacing is not that interesting for them.
    a. *"There are clearly some obvious, negligible changes particularly to comments, spacing, things like that that are kind of not useful. Especially when annotating because it shows*

you the most recent change which might be negligible, when you're more interested in when that line came about."

   b. *"If you add a parameter to the end of a method, you have to change the previous line, depending on where you add your commas. If you have a closing parenthesis on the last line then you must modify that line as well but it's not very interesting for that line because it will override the commit that says that line was added."*

4. Would have preferred to have been able to use the plugin to answer the StreamsBuilder questions or tasks *that ask more generally to find specific commits that do something overall to a class.*
   a. *"I see the use of the plugin but at the same time, the questions asked I feel the plugin didn't help that much. For example, one of the questions in the StreamsBuilder class was to look for commits that improve things, so I feel like if I had to redo that task, then using the highlighting would have been useful. Since I used it on looking at specific lines and areas of code, I didn't really have to look through the general history as much since there were fewer commits affecting those specific lines. For the questions I answered, I didn't feel like they were that useful but I do see the use of them."*
   b. *"In general day-to-day usage, it would seem interesting to use because I looked at the commits that it [Intelligent History] darkens and they do not seem to be useful for this file [Topology] at all. If I were just looking at some sort of change that modified the behaviour of a class, then this seems to nicely filter what is important and what isn't."*
   c. *"When I think to what I have to do at work, often you don't know what kind of modification you're looking for, especially when automated tests break or something, you're going to be looking for something that is very recent and if you're on a popular file, there might be a lot of commits. But with this, you'd be able to easily filter out things that actually made differences rather than negligible changes."*
5. The revision diff metadata is language-specific, the participant mentioned they would like to see the regular expressions used be crowd-sourced or to allow users to create and supply their own rule sets for filtering commits in different programming languages. There might also be ways of flagging specific commits to try to generate something from them.
   a. *"Facilitating the opening of the Jira issue [in the IDE] definitely helps."*

Participant C

This participant examined very few commits as they mostly took advantage of trying to deduce rationale or intent from the code itself. Their answers were partially incomplete or not to full satisfaction, though they did demonstrate effort in relating what they saw in the code changes to answering the questions.

Experiment

1. A: Topology
    a. Used IntelliJ's "Show History for Selection" feature to find KAFKA-7523 and visited the Jira issue in the browser. Visited KIP-401 to obtain the answer to how users of the Kafka API benefit from the change.
    b. Same as above, the participant visited KAFKA-4936 and KIP-303.
2. B: StreamsBuilder (with plugin)
    a. Scanned through highlighted commits and invoked Jira issue fetching until they found suitable issues.
    b. Answered based on looking at the code, no commits from the history examined. The answer was somewhat close since they used the comment associated with the method to explain their answer but it was obviously missing concrete intent which they would have gotten from examining the Jira issue or the commit that introduced the method.

Interview

1. 6 years of experience (3 years non-professional, 3 years professional).
2. Often looks at commits and issues. Seeks information from a GUI for navigating history and finds owners assigned to features when trying to obtain rationale information.
    a. "For stuff that I don't know why they're added or if there's a change that I want to get more context in, I look at the history and `git blame` to know when it was last changed, what the change reason was, and if it fixed any bugs or why it was added basically."
3. Whitespace, formatting, renaming, and import changes are not important. What is important are functions and the changes to their bodies.
    a. Considers changes made to documentation as important, only second to function/method changes: "Documentation is still important to know why it's [the code] changed but also when you explain why, a lot of it is done in the commit message versus the documentation. Those [documentation] might be edited but they might not actually reflect the reason for the changes."
4. Found the Jira view helpful and with the ability to shortcut opening the Jira issue link from the IDE. On whether the plugin made a difference to their experience, they recognized the different nature between the two sets of different questions, pointing out that the first set was about looking at particular lines or methods to ask why they had changes. The second set was about identifying some useful commits so the plugin helps to eliminate trivial commits and made it more helpful to navigate the history.
    a. "Having an integrated view for seeing the Jira issues is helpful because you don't have to navigate between multiple windows and you do have tabs alongside it. Seeing the description is quite important because that is the summary of the entire reason for those issues and the changes."

   b. *Rather than using Intelligent History on an entire file and specifically on how Intelligent History could be more useful for tasks related to specific lines of code: "If you want to look at the full context of that function, [Intelligent History] could still be helpful because looking at the history, there might be renames or other changes within a few specified lines. That would be where the plugin would be potentially more useful."*
   c. *"In terms of metadata, I don't think comment counts matter. They might show you how active an issue is but the priority should be the main indicator for that. Same thing with people involved, unless maybe if it gives you a point of contact like listing the LDAPs [email alias] for instance of people involved."*
5. Having a link to associated design documents could potentially be helpful. Descriptions and highlighting are helpful, especially when providing the reasons for why certain things are not highlighted.
   a. *"To get more information [to make their judgment on the plugin], you would need more questions [tasks] around looking at the history of a specific change."*

Participant D

This participant often read the commits or searched through the commits by reading the commit messages before closely examining the diffs. Often backtracked to perform comparisons between different commits to get a sense of what changed between pairs of commits.

Experiment

1. B: StreamsBuilder
    a. Performed a sequential search of commits while also reading the commit messages before examining the diffs.
    b. Same strategy, using "Show History for Selection," tried to get a sense of when the overloaded method was initially introduced as part of a file but realized the first commit shown in the window was not what introduced the exact method but most likely just touched the area of code. Scanned the commits for comparison again before settling on the correct commit for the answer.
2. A: Topology (with plugin)
    a. Tried to skim through commits by reading the commit messages, did not select the commits directly but did a visual filter by reading the commit subject lines and using the highlighter feature from the plugin. Read the commit message and said aloud "That seems related" before clicking on a commit. Only paid attention to highlighted commits while ignoring faded commits.
    b. Their strategy was to perform a search starting from the initial commit that introduced some `addSink` overloads to try to find the commit that introduced more overloads later, using "Ctrl + F" to look for the addition of `addSink` methods. Only paid attention to highlighted commits.

Follow-up Interview

1. 4.5 years of experience (2 years non-professional, 2.5 years professional).
2. Situationally or sometimes looks at commits and the issues associated with them.
    a. Investigation and introduction of bugs: *"If I'm trying to add or extend a feature or correct a bug, generally speaking I look at the commit history to see any associated tickets and find where a bug arose from."*
    b. Explained that the issues they look at for work give a summarized version of why certain functions exist and the logic put into designing functions, which are also kept in code documentation and relational diagrams (Doxygen for C++).
    c. Uses Jira for their current job and has worked in a project where every commit is associated with a subtask (under a primary task). When merging back into the main branch of their project, the merge is associated with the primary task.
    d. Often defers to coworkers when trying to find the answer to why code has been written a certain way; their work is more mathematically and physics-oriented.
    e. On the conditions of their last job: *"Oftentimes, someone who wrote the code several months ago may not be presently working at the company."*

f. On cases where they would look at the issue linked to a commit: “*If I take a cursory look at the code and I don’t understand it or the logic behind it, then I will look at the Jira ticket.”*

3. Major things they look for in diffs or on what stands out as important changes in a commit: changes to typing and accessor modifiers like variable types of function signatures, logic changes like altering of the fundamental form of a function. Unnecessary is import changes, typos, documentation changes, naming changes, injections (specific to a framework, e.g. Spring).
4. On plugin.
   a. *“It [Intelligent History] allowed for me to skim through the necessary commits a lot quicker and it addresses a lot of the issues I have with looking at trivial commits.”*
   b. *“It [the highlighting] visually reduces the amount of clutter I would have to look at.”*
   c. *I noticed this participant often preferred to view the Jira issue in the browser rather than read the fetched information within the IDE. The participant mentioned that they prefer having the full view of the issue even if viewing it in the browser is an extra step: “At the present moment, the tooling does not seem to include any linked or associated tickets, merges, etc. and when I’m looking at a Jira ticket, I look at everything associated with the ticket to try to get a much broader context.”* Doesn’t have the additional information that they personally look for.
5. Would prefer more information to be presented in the Jira summarization view. Lack of ability to see Jira issue comments. Would prefer a separate tab to see comments, e.g. if the ticket assignee made two comments then it would be nice to see them.

Participant E

This participant would verbally read the commit message subjects aloud rather than selecting every commit for closer inspection. Their search approach was more thoughtful based on scanning the commit message subjects rather than diving in deep for every commit. They mentioned that in general, they do not really read the description or discussion involved in issues and they tend to use them more to look at the set of files changed in a pull request linked to an issue to get a big picture because the issues themselves can be difficult to read.

Experiment

1. A: Topology
    a. Found two candidate commits based on reading the commit message subjects. Found the correct commit quite fast and easily since the commit's message included a term used in the code segment. Didn't need to select any commits to check more closely to find this commit.
    b. Read the Javadoc documentation accompanying some of the `addSink` overloads to try to get an idea based on the comments. Then tried to look through the commit history to see certain keywords used in the documentation and to try to find out which commits added them. Strategic approach in wanting to find the most general `addSink` method since other overloads were variations. Afterwards, they decided maybe the better approach would be through trial and error. I recommended a strategy of using "Show History for Selection" on an overloaded `addSink` added after the initial commit in the file.
2. B: StreamsBuilder (with plugin)
    a. Performed their search based on reading the commit message subjects, only selecting some of them for further examination. Only used the Jira issue fetcher action on the commit candidates they were sure about which lead to less usage of the plugin than anticipated. When looking for their second commit, there was one candidate (d98ec333) that turned out to be just a refactor. For their second commit, they also chose a commit that was part of a multi-commit change (KAFKA-6761) so they also went to examine the related commits before settling on this answer.
    b. Used "Show History for Selection" to narrow down the commits.

Interview

1. 13 years of experience (7 non-professional, 6 professional).
2. Often looked at commits and issues associated with them in the past. Not as much currently, qualified as rarely. When they started working at a larger company on a large-scale system, they did look at issues more frequently. Has also had experience working on a project where there was a convention of linking each commit to an issue from an issue tracking system.
    a. *"Usually you want to look at the set of files that were changed together because you might be only looking at a small glimpse of a larger change and you want to get (1) the full picture and (2) especially for testing, because if I'm doing a small change in this file, I want to look at the commit history to see all the other files that tested this thing, so I'm sure that I'm gonna test all the test cases."*

   b. On being asked why he would go further to look at the issue related to the commit: "*With the issue part, you might have a pull request link and then from the pull request, you can get to the files [that were changed together]. I wouldn't bother that much with looking at the issue for other things like discussion, other than the description and perhaps the last two comments, I wouldn't look much further than that."*
   c. On why he doesn't look further than the description, etc: *"There is a lot of discussion on the design, status updates, and planning and all of that, so it's kind of hard to read a full bug [report] and get to understand all of that, especially in distributed systems … so instead of looking at the comments, looking at the files in the pull request is kind of like [saying] 'this is what we settled on' so it's easier to look there and if you don't understand, you can go back to the comments and backtrack."*
   d. On finding code rationale information: *"There are a lot of assumptions about code conventions and style, which once you are familiar with the project, you get a sense of why it [the code] was written like that. It is harder for decisions like why is there so much overload here or why didn't they do something different and so on, but it's hard to think about this one [the answer to this question]." and "In one of the best jobs that I had, we were always involved in the design discussion, so usually I would always know the why for that one [job]. And the one [job] before that, it was more structured, like for each change, you would have a design document that was written before saying what new feature you added and so on. But this was like back in the waterfall days."*
3. Ignores any changes made to test files at first. Doesn't bother with changes that affected a large number of files; would look at discussion or issues first. Looks for files which contain the largest additions or removals.
4. Positive thoughts about the highlight and Jira actions.
   a. "I think that the biggest benefit was especially removing minor changes. It's really handy, it filters a bunch of, not garbage, but noise in the commit history. With just this single feature, it's already worth using the plugin."
   b. "I noticed I didn't use [(F2) Show Diff Metadata] feature that much. I think it's mostly because of having one extra click." The participant suggested maybe making this popup a hover instead but still wouldn't be sure how much use they would get out of it.
   c. "Even if it [the content] was not formatted, it [the Jira action] helps you contextualize the code change in a commit really quick."
5. Specifically addressing the utility or suggestions for improvement to the (F2) *Show Diff Metadata* feature, they would like a summary of things like methods modified, methods removed, or methods added, to let the developer quickly decide where to navigate to in the file. This was more of a suggestion for summarizing structural changes made to a file. A way to make diffs themselves easier to read and navigate.

Participant F

This participant made a lot of use out using the annotation and blame features in IntelliJ. They also alternated between scanning the commit messages and brute force checking every commit, particularly for question set B.

Experiment

1. B: StreamsBuilder
    a. This participant mentioned that they usually do read the commit message subject/title so this was their strategy. They immediately found their first commit by doing this. However, for their second commit, they took a while clicking on every single commit as they switched their strategy to looking at the diffs. Afterwards, they switched back to glimpsing the messages again for a meaningful change. In the end, they ended up looking at the present source code along with the "annotate" and "blame"-like features in IntelliJ to look at the commits that most recently modified lines in the present state of the class. They ended up jumping back to a commit they actually already saw before.
    b. The participant located the commit based on the commit message subject, given that the commit explicitly states that it added an "overloaded build method."
2. A: Topology (with plugin)
    a. Used `git blame` feature.
    b. Used `git blame` feature.

Interview

1. 6 years of experience (4 non-professional, 2 professional).
2. Often looks at commits and issues in their actual work. Uses Jira in their current work and looks at the issues daily for tracing bugs and finding out who wrote the code that introduced a bug. Also when implementing features, particularly gave an example where someone else implemented a similar feature and they were following the code as a high-level example of how to write their own implementation for something similar.
3. The motivation behind how the code is written is the most important information to them in a Jira issue. They would first look at the commit message and then the Jira ticket. A diff should show the different parts that are actually meaningful. A diff could be improved if it only showed what was important.
    a. "For example, like your plugin's diff metadata feature, it shows what kind of diffs there are. It would be even better if the diff is showing only what is important, for example documentation, if given the option to hide those diffs, it would be awesome. Or if you could hide the introduction of a library, it would make [examining diffs] easier."
4. Experience with the plugin:
    a. "It helps me ignore irrelevant commits."
    b. *"I would prefer to just open the Jira link instead of having the Jira issue information displayed in the side [of the IDE]."* They mention that they're just used to seeing the Jira as-is in the browser/website. They currently see no issue with context switching between the IDE and browser.

c. Features of the plugin are too tedious to access, i.e. the diff metadata and Jira issue fetching require extra clicks. It would be better if this information could be shown more immediately on issue select.

5. No further comment.

Participant G

This participant spent time manually selecting and inspecting many commits to gain a broad sense for changes in the commit history. They often backtracked to previously seen commits as well.

Experiment

1. A: Topology
   a. Looked through many of the commits in the history, presumably trying to locate the commit based on manual search or trial and error. Eventually, I showed them the "Show History for Selection" feature to try to narrow down the commits.
   b. Found the commit that introduced the overloads again based on the same strategy as above, sequentially going through commits until they found the commit that later introduced more overloads. Visited the Jira issue to get the rationale information to answer the question.
2. B: StreamsBuilder (with plugin)
   a. Looked through the highlighted commits. They did initially select a commit that I accepted from another participant but that participant was able to explain that it was part of a multi-commit change and identified the improvement specifically. For this participant, I had them move on to select another commit if they didn't feel confident in a candidate commit. This participant was more liberal with invoking the Jira issue fetching action.
   b. Used "Show History for Selection" with some guidance. Explored the commits shown in this tool window. The participant invoked the Jira issue plugin and visited the commit in the browser due to their preference for the native presentation.

Interview

1. 2 years of experience (2 non-professional, 0 professional).
2. Has had experience working issues and commits, would qualify as sometimes.
   a. On what makes them go to look further at the issue associated with a commit: *"Usually I would need some idea of the motivation behind the change. Sometimes a change makes things a little harder to understand from what the API documentation states so I would look at the change to see if that decrease in readability was worth improving some aspect of the code base."*
3. States that documentation changes can still be useful: *"In cases where a commit changes the behaviour of a function, typically that sometimes involves changes to the documentation, and while it's really easy to just ignore the documentation change, it is helpful to filter out documentation changes for other functions that rely on a modified function."*
4. Liked the Jira issue hyperlinking the most.
   a. *"I think the major quality of life improvement was hyperlinking to the Kafka issue itself. The summary in the IDE was helpful but presentation of code fragments in Jira issues poses an issue. The lack of presentation for code segments makes it confusing to read in the IDE but it's still a much more improved experience than having to manually type the Jira issue ID itself [in the browser]."*
5. No other comments or feedback.

Participant H

This participant used a combination of sequentially examining commits in the history and using IntelliJ's "Show History for Selection" feature to narrow down some commits for finding information.

Experiment

1. B: StreamsBuilder
    a. Chronological approach to checking the diff and commit message for each commit. When looking for their second most important commit, they ended up leaving the IDE twice to look at Jira because one issue had no information in its Jira description.
    b. Used IntelliJ's "Show History for Selection" feature and examined the commits chronologically until they found the KAFKA-7027.
2. A: Topology (with plugin)
    a. Used IntelliJ's "Show History for Selection" to find KAFKA-7523 and KIP-401.
    b. Not used to using the features of Intelligent History and also seemed to have a preference for viewing the Jira issue fully rendered in the browser?

Interview

1. 3.5 years of experience (2 years non-professional, 1.5 years professional).
2. Doesn't often look at commits and issues, would qualify as rarely. Their current work does not have anything as structured as linking issues from Jira to commits.
    a. On how they would normally obtain answers to code rationale questions, they would try to seek out the person who wrote the code or ask around to try to find someone who might know. About 65% of the time, the person can usually answer the question or provide an idea on why the code was written a certain way. The other percentage is due to the person forgetting what they were thinking when writing the code.
    b. Commit messages in their workplace are mostly patches to an element not behaving correctly and is not very detailed. They would describe their current practice/commit history as quite lacking in content and information on code rationale.
3. Often used `git blame` in the past but it's hard to use this without much substance in commit messages. Comment changes in diffs rarely help them so they tend to ignore them.
    a. *"I used to use `git blame` to try and see what the changes are [...] but it's hard for me to understand that without commit messages so that's kind of like a last resort when nobody is there to ask."*
4. Using the plugin made a difference to their second task.
    a. *"I think what definitely helped the most was definitely being able to highlight important changes, so basically the pencil icon was amazing for speeding up what I had to search for. Especially, in finding that sink issue, if I didn't have the pencil, I think I would have gone through all the commits backwards and just try to see what the diffs but it removed a bunch of the ones that it [Intelligent History] deemed not useful and it ended up being correct so I just skipped over a bunch."*
    b. *"I also found the Jira feature pretty neat as well, the fact that it opens it really quickly in the same IDE kinda made the context switching slightly less awful. Plus it links the Jira so it's always an addition, never an impedance on being able to develop stuff."*

   c. *"I don't think I used the summary much but I guess it was implicitly used in the highlighting so I guess I'll still show appreciation for that even though I didn't necessarily use it."*
5. No issues with the tool but they thought they confused themselves a bit while using it, considering it an end-user issue rather than a tool. Could not think of any case in their experience where using Intelligent History was causing friction for them.

Participant I

Less commits explored compared to other participants overall. In the first task, they did a sequential search on the commits by examining each commit based on date and message to narrow down the appropriate commits. In the second task, they made use of the highlighter action and used it along with reading the commit message subjects to try to identify important or potentially meaningful commits. Overall they made use of reading the commit message subjects and in the end, the participant was most partial to the Jira integration.

Experiment

1. A: Topology
    a. Immediately shown the "Show History for Selection" feature. Read the Jira issue description and visited the referenced KIP to obtain the rationale information.
    b. Sequentially looked through each commit and its diff after the initial commit that introduced `addSink` methods to find the commit that later introduced more `addSink` methods. Search based on skimming the diffs of each commit. Found the correct commit through this approach. Made use of looking at the associated Jira issue.
2. B: StreamsBuilder (with plugin)
    a. Found the commits quite fast from using the highlighter action and presumably, skimming the commit message subjects to try to identify potentially meaningful commits.
    b. Used the "Show History for Selection" feature.

Interview

1. 4 years of experience (1 non-professional, 3 professional).
2. Sometimes looks at commits and issues. Typically uses issues to try to understand why a method was introduced or when they want to add something to a class but is not sure how it would work in the scope of the class.
    a. *"Sometimes they [issues] are really helpful or sometimes they are not detailed enough."* They also mentioned sometimes issues are missing from commits, such as being forgotten or lost in their issue tracking system.
    b. Typically reads the description in an issue to try to figure out why someone has written code a certain way.
    c. Sometimes changes are made by people who don't have the full picture of what they added code to: "That person actually never really worked on that part of the software and only added this one method so the person doesn't have the overall picture, so it was not enough information given."
3. What makes a commit important?
    a. *"Ideally I want to see from the title of the commit what methods or part of the files it touched so I can quickly go through and see if actually the part I'm looking at is in the commit history. I also want to know if a commit is just a hotfix or refactor or introduction of a new feature to get an idea of what's behind the commit."*
    b. On what kinds of diff changes they would prefer to not see: *"Ideally a filter or highlighter where something I'm looking at should be more than 1 or 2 lines or something that can filter out direct and indirect changes. One of the most basic indirect*

changes is a change to a method name or changes to parameters which results in a change elsewhere [where the method is called].”

4. For the plugin, they really liked the Jira issue action. Didn’t make any comment about the highlighting.
    a. *“I liked having a brief description of the issue and a hyperlink where I can click directly to access the Jira issue and get more information from there.”*
    b. *“I liked how I was able to see the priority [of the issue] because if something has a higher priority, it usually involves heavier changes.”*
    c. *“Without the plugin, it was a little more annoying and time-consuming opening up the browser, looking for all of the issues, whereas with the plugin, it was already there [in the IDE] and I can just take a quick look and see if it [the commit] might be relevant and I can take a deeper look if needed.”*
    d. Didn’t really need the issue metadata but thought it might be helpful for other tasks but for the tasks given, they did not need the information.

Participant J

This participant actually initially was reading the commit message subjects/titles or skimming them before going into inspecting them but very quickly stopped doing this and opted to do a "sequential search" instead by trial and error, visiting each commit sequentially, checking the diffs of each commit instead, and backtracking to previously examined commits. Before switching to this approach, they commented on the "misleading nature" of the commit messages and didn't find them to be accurate or descriptive of the changes made to the class.

Experiment

1. B: StreamsBuilder
    a. Initially did their search based on reading the commit messages but eventually turned to sequentially searching (trial and error) from the beginning of the commit history to the end of the history to try to look for their first important commit. Was reminded to visit the Jira issue to try to find the rationale information and was able to locate the first commit by visiting the Jira and referenced KIP. They re-examined commits they saw before as well (indicated by the graph for this question). Even though they were actually reading the messages in the beginning, it seemed like they decided to abandon this approach due to the large volume of commits.
    b. Used "Show History for Selection," narrowed down the commits, and visited the Jira issue.
2. A: Topology (with plugin)
    a. Used "Show History for Selection" and visited the Jira issue for the commit shown.
    b. Looked only at highlighted commits from the beginning of the commit history, similar approach to B(a). They commented that the highlighter did make a difference to their experience while they were working with the commit history for this task.

Interview

1. 6 years of experience (4 non-professional, 2 professional).
2. Rarely looks at commits and issues. They look at the commits more often than issues to know exactly what changes happened over the why. In their industry experience, they used Jira with labels for tagging each issue and each PR corresponding to an issue.
    a. *"Since each issue corresponded to a pull request, we usually put [rationale] information in the pull request. The issue was just to explain the problem and the solution was in the pull request."*
    b. *On what prompts them to look at the Jira issue: "When I'm trying to add something to the code or refactor something and I don't want to break anything. Once I see some code and I wonder why this happened, since the solution was in the pull request, I would usually go to the pull request."*
    c. *"I usually open up the `git blame` and see who historically edited [a certain part of] the code and that usually leads to a commit but then sometimes it's misleading because wiped or as part of a refactoring, they transferred or made a new class of old material, which causes their commit to show up first in `git blame`."*
3. Important in diffs:

   a. *"Usually from a pull request you would see if it [the pull request] was an added feature or a major improvement to the code."*
4. Experience with using the plugin:
   a. *"The benefit of the plugin is that information is all centralized in one tool and IDE and there is less navigation I have to do between the windows which is really helpful."*
   b. *"With the highlighting, it removed the obvious minor changes that I didn't really need to look at."*
5. No further comments.