Open Source Development

Justin Hutchins - jushutch@umich.edu

26 April, 2021

Flutter

Overview

Flutter is a free, open-source SDK (software development kit) for developing applications that can be compiled natively for iOS, Android, and the web, all from the same source code. It was released by Google in 2018 and has undergone constant development by the open source community. The documentation for the project lives at https://flutter.dev/ and the codebase exists on GitHub.

Context

The motivating idea behind Flutter is to make it easier and quicker for developers across skill levels to build applications for all major platforms at the same time. The main users are software developers, from multi-billion dollar tech companies such as Google, eBay, and Capital One, to small startups and personal projects for people with intermediate programming experience. Since Google is such a prominent figure in the technology and software engineering industry, they've set out to create their own public development stack, which so far includes the dart programming language, the Flutter SDK, and the Firebase platform, which all integrate together seamlessly. There are many other technology stacks for developers, but nothing that competes with Flutter directly. Flutter is the easiest way to develop for all platforms at once and has the potential to become a primary tool for app development, especially since it's backed by Google and has an active open-source community.

Governance

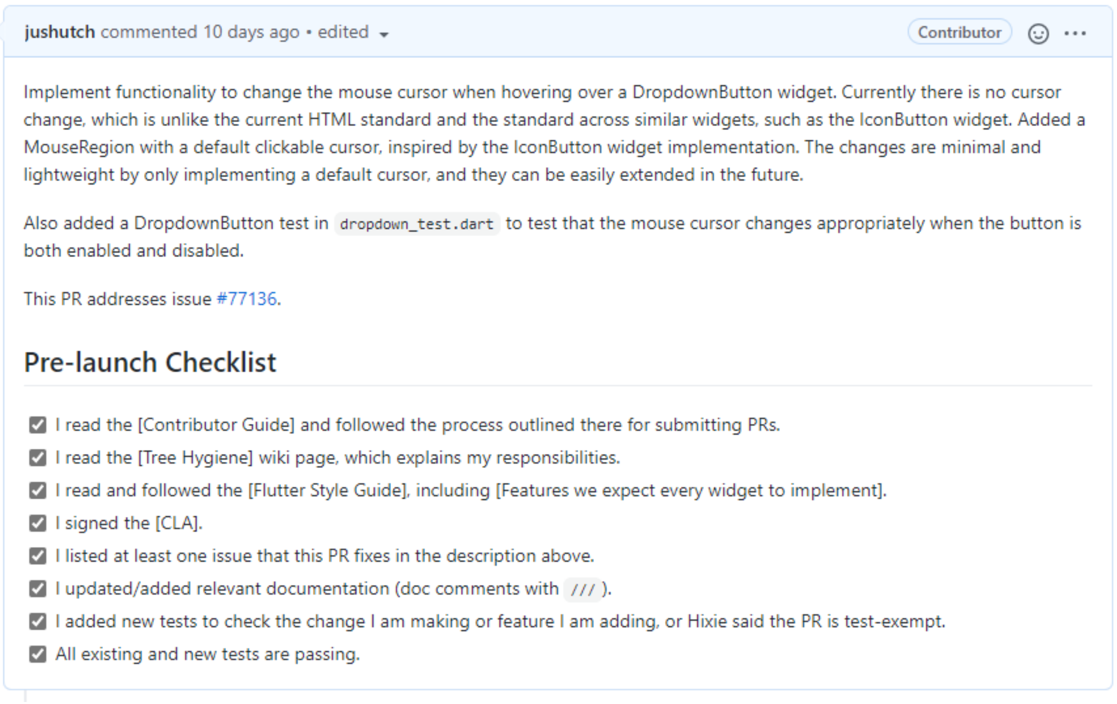
The project is officially operated by Google, meaning that the lead developers and maintainers are employed by Google. Any contributor can be given commit privileges if they demonstrate a long history of high quality contributions and meet certain requirements outlined in the documentation on contributing. Currently, GitHub shows that there are over 800 users with commit privileges, but it's obvious that a large majority of these users are inactive. In reality, it appears that the main team consists of roughly 10 people who manage pull requests, issue triage, current build status, and various administration.

Discord is the only forum of communication for the project and includes multiple channels for different areas of the Flutter project. For example, there's a channel with updates about the current build status, channels for each of the current platforms (iOS, web, and Android), and general help channels for users and developers, along with many more. The Discord is fairly inactive however, with most discussion coming from users who have questions about building personal applications and the lead contributors discussing issues with each other. However, the Discord is also the intended place for a contributor to request a code review for a pull request if it hasn't already been reviewed within 2 weeks. As such, there are frequent posts from contributors asking for help with issues and pull requests. All of this communication is professional but informal, with no strict guidelines about who can post or what can be posted.

Landing a Pull Request

The process of actually contributing to the project is outlined in the documentation on contributing and the project wiki on GitHub. The document lists the following steps with links to more documentation in order to start contributing: read the code of conduct, read about the project values, set up your development environment, learn about tree hygiene, and read the style guide. After doing so, the actual process to land a pull request is a bit more complicated. In my experience, the first step is finding a suitable issue and leaving a comment to let other developers know that you're working on it. Then you set up your local environment according to the documentation and start developing locally, making sure to keep your code up to date with the current master branch, since commits are made frequently which might affect your work. Once you've made your changes, you'll need to run the existing test suite to verify that it's not a breaking change. This process is also well explained in the documentation, along with information on how to handle a breaking change. After passing the test suite and committing your changes locally, you can push your development branch and open a pull request.

Opening a pull request on the Flutter project automatically provides a template that outlines what information needs to be included in the pull request. For example, you must explain the change, link to the issue it resolves, mention the new tests that cover your changes, and complete a checklist of other miscellaneous tasks. After you've opened the pull request, a GitHub bot will add appropriate labels to the pull request and check that you've submitted a Contributor License Agreement. At the same time, the CI testing will start which is ran using GitHub Actions. The exact testing depends on the changes in the pull request, but consists of at least 50 tests, verifying that the changes don't break any existing tests on any supported platforms. The last step is getting a code review from a lead contributor, which can take up to two weeks. After making any required changes, the code reviewer will comment ”LGTM” (looks good to me), and either merge the branch if the tree is currently green or add the label ”waiting for tree to go green”, which the GitHub bot will use to merge the branch automatically when the tree is green. Once the changes are merged, the contributor is responsible for monitoring the Flutter dashboard to checks for regressions and revert the commit immediately. This unique case is detailed explicitly in the documentation on tree hygiene.



Implement functionality to change the mouse cursor when hovering over a DropdownButton widget. Currently there is no cursor change, which is unlike the current HTML standard and the standard across similar widgets, such as the IconButton widget. Added a MouseRegion with a default clickable cursor, inspired by the IconButton widget implementation. The changes are minimal and lightweight by only implementing a default cursor, and they can be easily extended in the future.

Also added a DropdownButton test in `dropdown_test.dart` to test that the mouse cursor changes appropriately when the button is both enabled and disabled.

This PR addresses issue #77136.

Pre-launch Checklist

- ✓ I read the [Contributor Guide] and followed the process outlined there for submitting PRs.
- ✓ I read the [Tree Hygiene] wiki page, which explains my responsibilities.
- ✓ I read and followed the [Flutter Style Guide], including [Features we expect every widget to implement].
- ✓ I signed the [CLA].
- ✓ I listed at least one issue that this PR fixes in the description above.
- ✓ I updated/added relevant documentation (doc comments with `///`).
- ✓ I added new tests to check the change I am making or feature I am adding, or Hixie said the PR is test-exempt.
- ✓ All existing and new tests are passing.

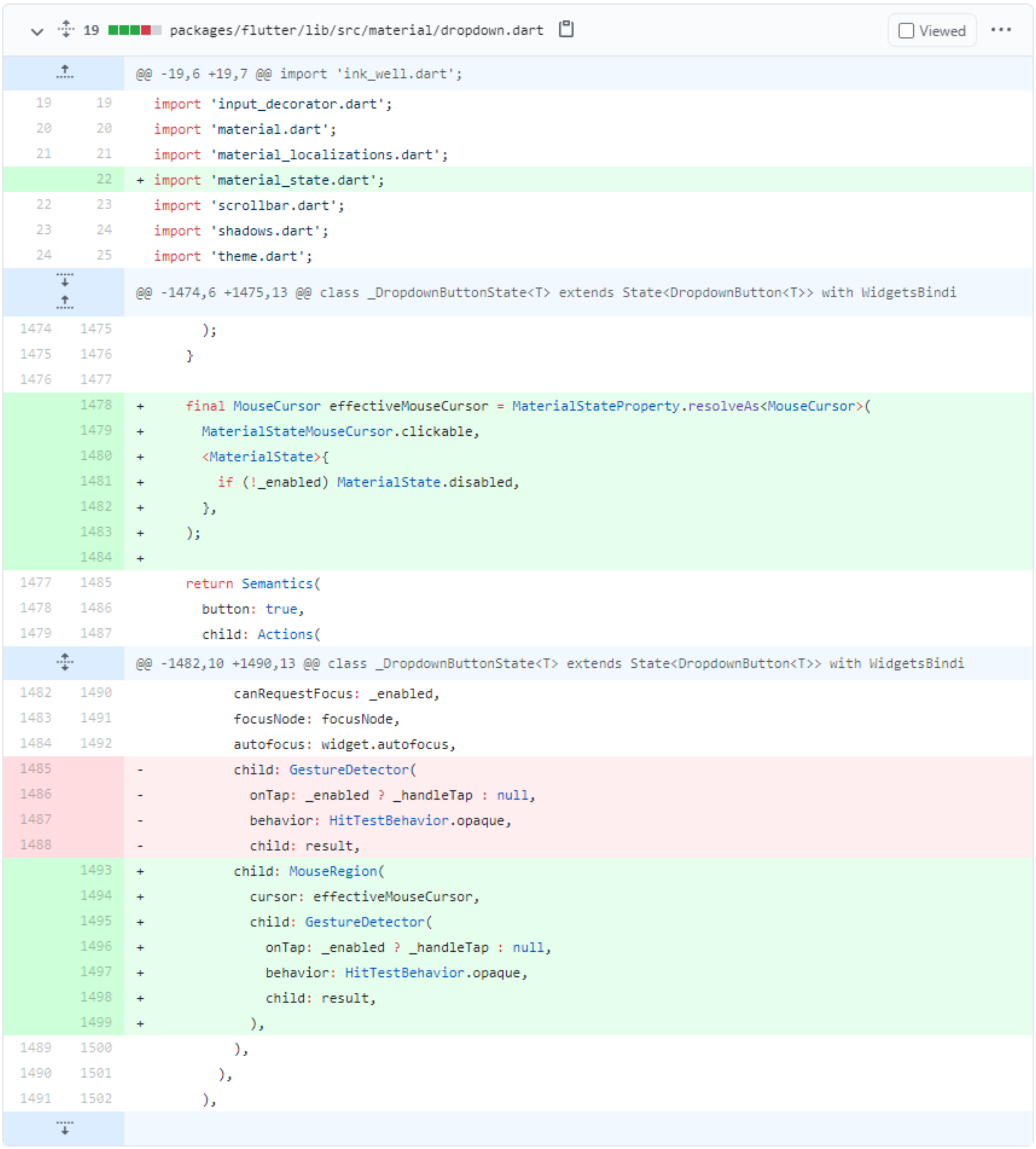
Figure 1: An example of the pull request template from Task 1

Task 1: DropdownButton Widget

Pull request: https://github.com/flutter/flutter/pull/80567

This task required adding functionality to change the cursor when hovering over the DropdownButton widget. The current HTML standard is that the cursor changes from the basic cursor to the clickable cursor when hovering over any button. In the original issue, it was mentioned that this functionality must have been overlooked for the DropdownButton widget, and that it should be implemented to follow the HTML standards.

To complete this task, I had to implement the requested functionality and also add a new test to verify this behavior. Changing the cursor on hover was as easy as adding a MouseRegion widget around the button and setting the clickable cursor as the default. This solution was inspired by other similar widgets such as the IconButton and TextField widgets, but was a minimal implementation to provide more extensibility in the future. As such, I decided not to make the cursor a public property and to only change the cursor states when the button was enabled or disabled.



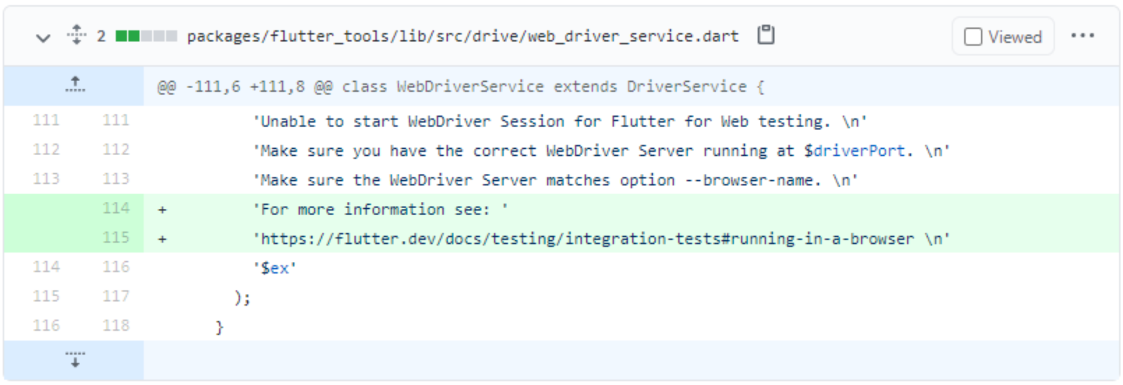
```diff
19 packages/flutter/lib/src/material/dropdown.dart
@@ -19,6 +19,7 @@ import 'ink_well.dart';
 import 'input_decorator.dart';
 import 'material.dart';
 import 'material_localizations.dart';
+import 'material_state.dart';
 import 'scrollbar.dart';
 import 'shadows.dart';
 import 'theme.dart';
@@ -1474,6 +1475,13 @@ class _DropdownButtonState<T> extends State<DropdownButton<T>> with WidgetsBindi
       );
     }

+    final MouseCursor effectiveMouseCursor = MaterialStateProperty.resolveAs<MouseCursor>(
+      MaterialStateMouseCursor.clickable,
+      <MaterialState>{
+        if (!_enabled) MaterialState.disabled,
+      },
+    );
+
     return Semantics(
       button: true,
       child: Actions(
@@ -1482,10 +1490,13 @@ class _DropdownButtonState<T> extends State<DropdownButton<T>> with WidgetsBindi
           canRequestFocus: _enabled,
           focusNode: focusNode,
           autofocus: widget.autofocus,
-          child: GestureDetector(
-            onTap: _enabled ? _handleTap : null,
-            behavior: HitTestBehavior.opaque,
-            child: result,
+          child: MouseRegion(
+            cursor: effectiveMouseCursor,
+            child: GestureDetector(
+              onTap: _enabled ? _handleTap : null,
+              behavior: HitTestBehavior.opaque,
+              child: result,
+            ),
           ),
         ),
       ),
```

Figure 2: Changes made to the DropdownButton widget

Task 2: Error Message

Pull request: https://github.com/flutter/flutter/pull/81107

This task involved improving an error message in the Flutter CLI tool that was caused when running integration tests without the proper WebDriver configuration. The original issue was raised because the error message didn't mention the ChromeDriver, which was a common reason for the error. However, after reading through the documentation and understanding that ChromeDriver is the WebDriver for Google Chrome, and that each browser requires it's own WebDriver to run integration tests, I realized that the error message was intentionally generic to cover all of these cases. As a result, I decided that adding a link to the relevant documentation would be the best solution, since it would help direct users to a more detailed explanation and provide troubleshooting help. This error message pattern is used elsewhere in the code and maintains the simple and generic spirit of the original message.



packages/flutter_tools/lib/src/drive/web_driver_service.dart

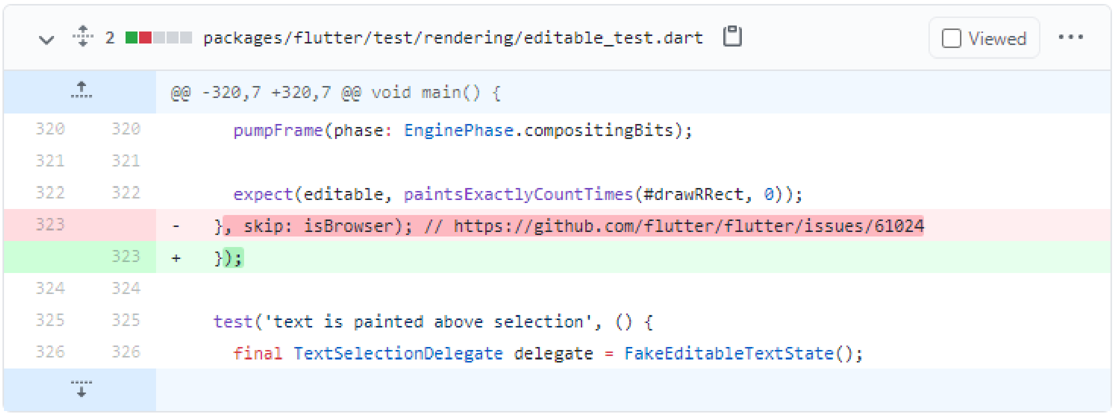
```
@@ -111,6 +111,8 @@ class WebDriverService extends DriverService {
          'Unable to start WebDriver Session for Flutter for Web testing. \n'
          'Make sure you have the correct WebDriver Server running at $driverPort. \n'
          'Make sure the WebDriver Server matches option --browser-name. \n'
+         'For more information see: '
+         'https://flutter.dev/docs/testing/integration-tests#running-in-a-browser \n'
          '$ex'
        );
      }
```

Figure 3: The changes made to complete Task 2

Task 3: Broken Test

Pull request: https://github.com/flutter/flutter/pull/80761

This issue was related to a broken test for the RenderEditable widget. The original test was failing and flagged by the lead developers to be marked as skipped until it was fixed. The original issue included the error message that was being raised, which mentioned a problem with painting a widget. In general, the error was vague and the issue had no clear direction, which led me to believe that it might be more time intensive than the other tasks. However, I was able to deduce that the test had actually been inadvertently fixed in a previous commit that was unrelated to the original issue. Upon this realization, resolving this issue was as simple as removing the code to skip the test.



packages/flutter/test/rendering/editable_test.dart

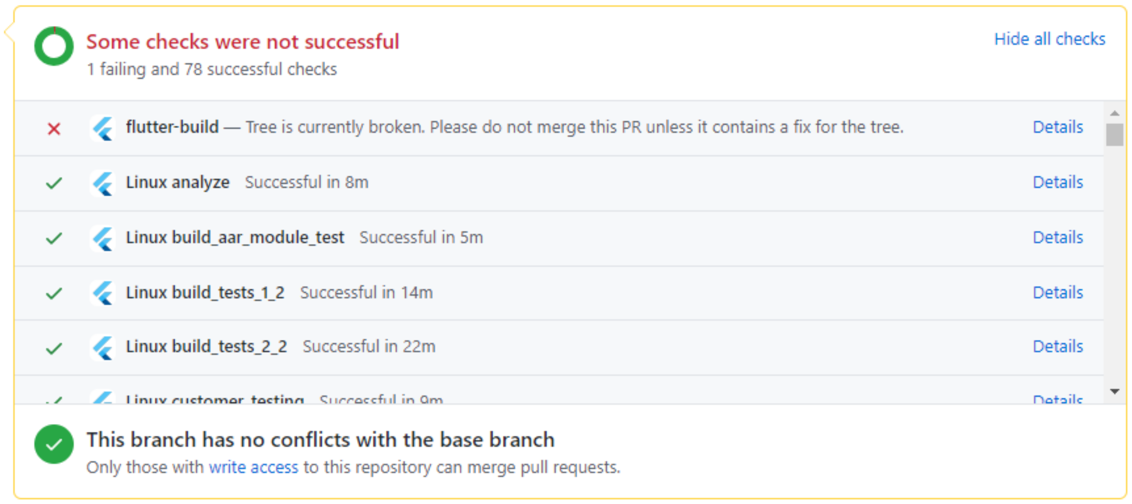
```
@@ -320,7 +320,7 @@ void main() {
      pumpFrame(phase: EnginePhase.compositingBits);

      expect(editable, paintsExactlyCountTimes(#drawRRect, 0));
-   }, skip: isBrowser); // https://github.com/flutter/flutter/issues/61024
+   });

    test('text is painted above selection', () {
      final TextSelectionDelegate delegate = FakeEditableTextState();
```

Figure 4: The changes made to complete Task 3

Quality Assurance

Quality assurance is a huge priority for a project as large and well-known as Flutter. The fact that it's supported by Google is a testament to the level of quality that is expected, and also explains why the QA requirements they set forth work so well. There were multiple forms of QA that were required by the project: every pull request must include tests that covered the changes made, unless there was a special exemption granted by one of the lead developers at Google, every pull request must pass CI testing, which consisted of at least 50 unique checks, every pull request must undergo a code review by a lead developer, and the merged changes must pass all post-merge tests that are ran and monitored on the Flutter dashboard. All of these checks were mandatory and outlined thoroughly in the contributing documentation and project wiki.

The test requirement was enforced by a GitHub bot for the Flutter project that comments on a pull request without tests, letting the contributor know that they must have tests or seek a test exemption. The CI tests were all required to pass before the branch could be merged, with one of the checks testing that the tree was currently green, i.e. that the current master branch was passing all tests on the Flutter dashboard. The final requirement was much harder to enforce, since it performs QA after the commits have already been merged. However, the rule is that any commit that breaks the tree must be reverted as soon as possible, no questions asked. The documentation on contributing includes a page on tree hygiene that outlines the responsibilities of the contributor in this scenario, but ultimately there is no way to enforce them, meaning that the lead developers often have to step in to resolve these issues.


Some checks were not successful
1 failing and 78 successful checks
Hide all checks

✗	**flutter-build** — Tree is currently broken. Please do not merge this PR unless it contains a fix for the tree.	Details
✓	**Linux analyze** Successful in 8m	Details
✓	**Linux build_aar_module_test** Successful in 5m	Details
✓	**Linux build_tests_1_2** Successful in 14m	Details
✓	**Linux build_tests_2_2** Successful in 22m	Details

This branch has no conflicts with the base branch
Only those with write access to this repository can merge pull requests.

Figure 5: Example of CI testing using GitHub Actions for pull requests

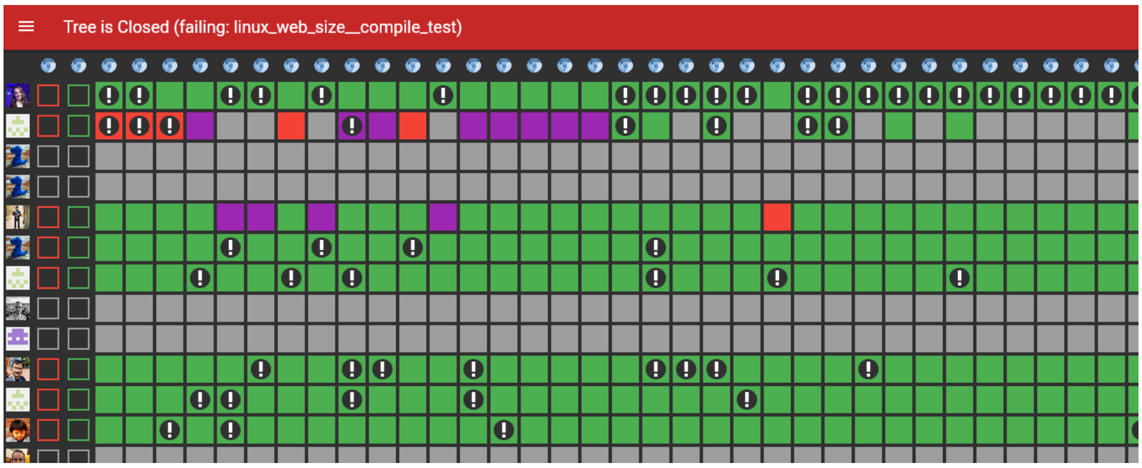
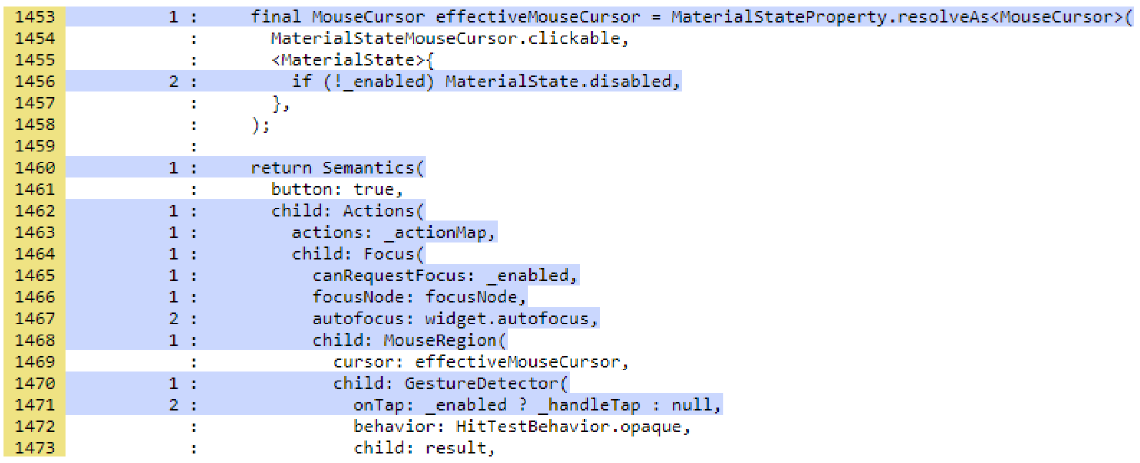


Figure 6: A small sample of the Flutter dashboard when the tree was closed

Besides the required QA, I performed multiple other QA activities to ensure that my contributions met quality standards, which included manual mutation testing, generating test coverage with lcov, using a static analyzer for style, and running the existing test suite to check for breaking changes. In Task 1, I wrote a unit test for the DropdownButton to test that the changes I made met the functional requirements of the issue. Using this test, I was able to manually mutate my implementation to verify the quality of the test and make sure I hadn't missed any edge cases. By performing this manual mutation testing, I actually realized that I wasn't testing behavior for when the button was disabled, and was able to improve my test to cover this case.

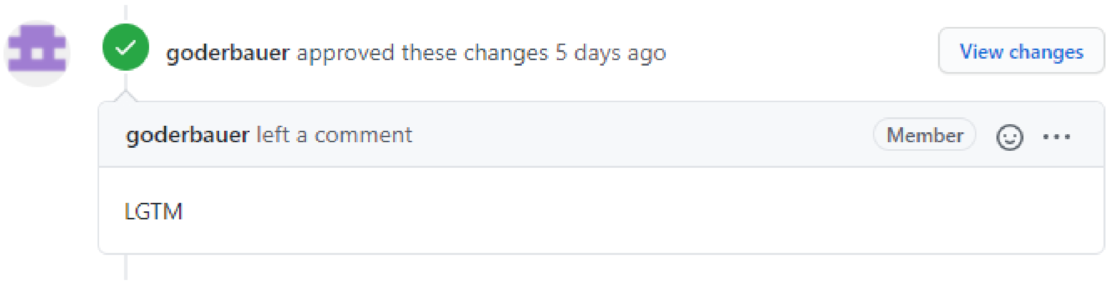
The Flutter project also includes test coverage functionality using lcov and a global lcov file containing the test coverage of the project, which is updated on every commit. Using this functionality, I was able to generate coverage for the test I added, and then viewed it using the web interface created by the genhtml tool. The Flutter tool also comes with inbuilt static analysis to enforce good code style, which was incredibly easy to use since it integrated with my IDE. Running the existing test suite was also straightforward using the Flutter CLI and only took 10 minutes at the most. This was a huge relief, since certain changes in Task 1 caused the suite to fail initially, but it didn't take long to resolve the issues and run the tests again.

I think that the combination of Flutter's required QA and my personal QA decisions were a good fit. Theoretically, a contributor could rely solely on Flutter's QA after submitting a pull request, and then make incremental changes until everything passed, but I felt more confident performing my own QA locally and opening a pull request when I was proud of the changes I had made. I also think that the QA activities I performed were perfect for the job: they were all relatively easy to perform and produced immediate results, compared to other activities such as dynamic analysis and more complicated static analysis tools.



```
  1 :     final MouseCursor effectiveMouseCursor = MaterialStateProperty.resolveAs<MouseCursor>(
    :       MaterialStateMouseCursor.clickable,
    :       <MaterialState>{
  2 :         if (!_enabled) MaterialState.disabled,
    :       },
    :     );
    :
  1 :     return Semantics(
    :       button: true,
  1 :       child: Actions(
  1 :         actions: _actionMap,
  1 :         child: Focus(
  1 :           canRequestFocus: _enabled,
  1 :           focusNode: focusNode,
  2 :           autofocus: widget.autofocus,
  1 :           child: MouseRegion(
    :             cursor: effectiveMouseCursor,
  1 :             child: GestureDetector(
  2 :               onTap: _enabled ? _handleTap : null,
    :               behavior: HitTestBehavior.opaque,
    :               child: result,
```

Figure 7: Test coverage generated by lcov



goderbauer approved these changes 5 days ago

goderbauer left a comment

LGTM

Figure 8: Approval from a lead developer after a code review for Task 3

Plan Updates

Task	**Estimated**	**Actual**
Task 1: DropdownButton		
Understanding existing code	1:00	0:45
Research similar widgets	2:00	1:00
Writing tests	1:00	1:30
Check for breaking changes	1:00	1:00
Submit pull request	1:00	1:00
Total:	6:00	5:15
Task 2: Error Message		
Understand error	2:00	1:00
Research chromedriver	2:00	0:30
Improve error message	0:30	0:20
Discussion	1:00	0:20
Check for breaking changes	1:00	0:20
Submit pull request	1:00	0:30
Total:	7:30	3:00
Task 3: Broken Test		
Reproduce error	0:30	1:00
Understand test	2:00	1:00
Finding the bug	0:30	0:30
Fixing the bug	1:00	0:10
Verifying changes	0:30	0:30
Check for breaking changes	0:30	0:10
Submitting a pull request	1:00	0:30
Total:	7:00	3:50

Overall, the time estimate for Task 1 was mostly accurate, while the estimates for the other tasks deviated significantly, as illustrated in the above table. The biggest time differences for Task 1 were in researching similar widgets and writing tests. I was able to find multiple examples that were similar almost immediately, mostly because I was already familiar with the Flutter framework and similar widgets such as the IconButton and TextField widgets. Both had similar implementations for the cursor changing on hover, which made it easy to adapt the same idea and code to the DropdownButton widget. However, the testing framework ended up being much more confusing than I expected. I was able to write it based on similar tests for the other mentioned widgets, but the test was failing and I couldn't understand why. It ended up being a problem with the widget I was building in the test, where it was inadvertently disabled when it should have been enabled. Besides these two deviations, Task 1 was fairly linear and went as expected.

Task 2 ended up being significantly less time intensive than expected. I assumed there would be a lot of time spent understanding the error message, but the linked documentation in the original issue was clear, concise, and explained the issue well. It didn't take long before I realized that the ChromeDriver was not the only possible reason for this issue; it could be raised by a misconfigured WebDriver for any browser. Knowing this, I commented on the original issue with my findings and asked for opinions on how the error message could be improved. However, I did not receive a timely response and no discussion occurred beyond my comment, which meant that I overestimated the time spent on discussion. All of the other subtasks went quicker since I was getting more used to running the tests for breaking changes and opening pull requests.

Task 3 had the most interesting deviation; I planned on finding and fixing a bug in the broken test, but it turned out that the test had already been fixed in a previous commit. As a result, the test ran and passed, so I spent a lot of time trying to reproduce the bug, eventually having to reset to the commit that accompanied the GitHub issue. Upon closer inspection, there was only one following commit that affected the test, which I spent time understanding until I deduced that it had fixed the original issue. All I had to do was remove the code that skipped the test, which was extremely easy and quick.

Summary of Experience

Contributing

Flutter is the perfect example of an open source project: it's active, open to anybody, and easy to contribute to. Right from the beginning I was impressed by the wealth of documentation they had on contributing and how thorough their GitHub wiki was. They clearly outlined every step, from forking and cloning the repository to landing a pull request and monitoring regressions on the Flutter dashboard. I will admit that it was overwhelming at first, since I read through all of the documentation on contributing before I had even forked the repository, and that there seemed to be a lot of red tape to successfully landing a pull request. The most confusing and intimidating concept was the tree, which was detailed explicitly in a document called "Tree Hygiene". It sounded like there was a lot of responsibility involved in making changes and that the lead developers took all changes seriously. This is definitely something unique to large projects like Flutter with founders like Google; most of the smaller projects I considered contributing to seemed much more lax and informal, for better or worse. In general, it was hard to forget that I was working alongside professional software engineers at Google, which oftentimes made the process more stressful than it had to be.

However, actually working in the existing code base was incredibly enjoyable. It might have helped that I already had experience building apps with Flutter, but I found the code to be easy to read and well-maintained. Almost all code was accompanied by explanations and links to documentation, which was extremely helpful in understanding the code I was working on. Even though the entire Flutter project is large, it is well organized and easy to work on small pieces at a time. The Flutter CLI is also incredibly powerful, since it supports functionality for static analysis, testing with coverage, and building projects quickly. Overall, I never once felt impeded in development, and all of the provided resources and tools were invaluable to developing quickly and confidently.

There were a few things outside of the code that ended up being more challenging than expected. The first issue I had was being able to find open issues to work on. While there are currently over 5,000 open issues, they are claimed quickly and vary in difficulty. There is also a labelling system to help group together issues based on priority, platform, and other qualities, which seems like a good idea, except the number of possible labels seems to have grown out of control. This combination meant I had to spend a long time looking for issues that I was capable of resolving. However, this was the only main difficulty I had in contributing, which I consider to be a minor problem.

Community

As mentioned previously, Flutter is primarily operated by software engineers at Google. On the bright side, it means that the project is in capable hands, and that the people in charge are motivated to keep the project active since it's part of their job. On the other hand, it means that there are incredibly high standards for contributing and a lot of red tape. One of the most notable problems was the lack of lead contributors. It seemed as if there were maybe 3 or 4 people in charge of handling issues, pull requests, test exemptions, build issues, and fielding questions. They were clearly spread thin, and I give them a lot of credit for being able to manage the project as well as they could.

Personally, I had a largely positive experience with these lead contributors. I was able to get in touch with Ian Hickinson (known as Hixie to Flutter contributors) in one of the developer channels on the Flutter Discord. I requested a review of my pull request for Task 1 and Hixie helped by mentioning two other lead contributors in a comment on the pull request. My pull request for Task 3 was reviewed by Michael Goderbauer, who left a helpful comment to rebase my changes onto the current master branch to resolve failing CI tests and ultimately approved the pull request. All of these interactions were professional and helpful, which gave me a positive impression of the project maintainers.

On the other hand, there were a few times were I felt that the community was either not active or just apathetic. This was mostly apparent for Task 2, when I left a comment on the issue to seek the opinions of other developers. After nobody responded to my comment, I decided to post it in the Flutter Discord and let people know that I was seeking opinions. Nobody responded to that post and I never received an opinion on the original issue. In general, it seemed that the Discord wasn't very active or friendly, and that the community as a whole was not collaborative. I might have been expecting too much for other developers to take an interest in such a small issue, but I though it would be easy enough for them to give an opinion. Besides this specific instance, I kept an eye on the Flutter Discord and noticed that a lot of people were ignored, and that the only people who engaged in conversation were the lead developers with each other. I'd like to believe that the problem is a result of so few lead contributors, but I also think that the atmosphere and tone of the community could be improved.

Changes

Overall, I think the Flutter project is a great example of an open-source community: it has strong leadership, great documentation, and significant quality assurance. All of these things are equally important for welcoming new contributors and promoting meaningful contributions. However, there are is a significant problem with how large the project is and the number of lead developers. The first symptom of this problem is the sheer number of open issues; if I were running a similar open source project, I wouldn't keep around issues older than a year and I would have higher quality standards for submitting an issue. In general, there seem to be a lot of duplicate issues, issues that don't have reproducible steps, or issues that are proposals with no follow up comments. I think the labelling system is partly to blame because it fails to categorize the issues effectively. There are too many labels, which makes it hard to find what labels are actually being used for what types of issues. Along with cleaning up the labels, I would also use more automation to filter and remove issues. For example, the Flutter project already uses GitHub bots to check and merge pull requests; I think it could be equally helpful for a bot to routinely remove stale or low quality issues.

Another symptom of this larger problem is an inactive community without enough lead developer support. As mentioned earlier, the Discord was full of unanswered questions and requests for code reviews. There was also a lack of discussion, with most of the talking done between lead developers, not with other contributors. This was extremely detrimental to the atmosphere of the Discord and the community as a whole because it felt like the lead developers didn't care about the other contributors. A similar problem was having to wait weeks before getting a pull request reviewed by a lead developer. I don't know if this is typical among projects of this size, but it felt a little ridiculous to me; once the code is reviewed the first time, it will often require changes and more discussion, meaning that resolving a single pull request could take months.

The main cause of this particular problem is the lack of lead developers who have the ability to perform code reviews. It seems like a large majority of the active leads are software engineers at Google, which gives me the impression that they don't accept enough lead contributors, or that the contributors they accept don't stay active in the project. I think the first problem could be alleviated by lowering the standards to become a lead contributor and by being more welcoming to new contributors. The second problem could also be a result of the inactive community making the experience less enjoyable. If I were in charge of a project such as Flutter, I would try not to alienate new contributors first and foremost, since those are the people that could ultimately become lead contributors and help share the workload.

Overall

Overall, no open-source project is perfect, but I think Flutter is a great example of what makes open-source projects great. I thoroughly enjoyed contributing to the project and I'm hoping to keep contributing in the future. I also thought it was a great opportunity to learn about practical software engineering and to see what industry level quality assurance looks like. It was interesting seeing how code reviews, mutation testing, CI testing, and test coverage were all used together to prevent breaking changes or low quality code, even for seemingly small changes. The high quality assurance standards set by Flutter seemed to work extremely well, considering it's ability to catch broken code and improve maintainability.

This experience was also educational in surprising ways; for example, I didn't expect to overestimate the time commitment of the tasks I chose. Usually software engineers tend to have the opposite problem, but in this case I was preparing for the worst-case scenario. This was especially clear for Task 3, since I expected to solve a time intensive bug, not spend most of my time trying to reproduce the problem before I realized it had already been fixed. The possibility that a task was easier than it appeared was a risk that I hadn't considered, and thankfully didn't end up being a problem in this instance. In general, the conditions of this assignment were unique in respect to the effort estimation, where in industry finishing a task early would be a beneficial outcome.