Guidelines for triage

Triaging issues

The process of triaging bugs is to first go through these bug lists and make sure they have all been processed as described below:

1. Process the bugs with none of the classification labels.
2. Process the backlog of bugs with no priority.
3. Finally, one should look at the least recently updated bugs of any kind and see if they are still relevant, applying new labels, updating the subject line if necessary, providing an update if necessary, etc.

General

First, look at the bug report, and try to understand what the described problem is. Edit the original comment to remove boilerplate that the bug reporter didn't remove. Edit the original comment to add backticks (```) around blocks of stack traces, code, the output of shell scripts like `flutter doctor`, etc. Ensure that the title is a meaningful summary of the issue. These changes make the bug much easier to manage.

If their report is **unclear**, doesn't give sufficient steps to reproduce, or is otherwise lacking in sufficient detail for us to act on it, add a polite comment asking for additional information, add the `waiting for customer response` label, then skip the remaining steps.

If the bug is **still unclear** -- we have previously asked for more detail, and the bug reporter has had a chance to provide additional feedback, but has not been able to do so in a way that makes the bug actionable -- either apologize for us not being able to fix it and then close the bug, or add the `waiting for customer response` label, depending on your confidence that the reporter will be able to eventually provide sufficient detail. Then, skip the remaining steps. It is fine to be aggressive in closing bugs where the issue is not clear, because we have plenty of other issues where the bug *is* clear and there's really no value to us in having low-quality bugs open in our bug database.

If the issue describes something that you know for a fact has been **fixed** since the bug report was filed, add a cheerful comment saying so, close the issue, and skip the remaining steps.

If the bug is **clear enough** for us to act on it, continue with the following steps. To reach this step, the bug should be actionable, with clear steps to reproduce the problem. We have enough bugs filed that we will not run out of work any time soon; therefore, it is reasonable to be quite aggressive in establishing if a bug is sufficiently clear.

Artifacts

Ideally every issue would have a sample app that demonstrated the problem.

Performance bugs should have timeline traces.

Crashes should have crash logs with a Flutter version so that the flutter-symbolizer-bot can do its work (see also [[Crashes]]).

Unactionable bugs with unusual symptoms

As discussed above, if a filed issue is unactionable due to vagueness or a lack of steps to reproduce, it should be closed, because we're never going to get to it if we don't know what the problem is given that we have many, *many* other bugs that we *can* make progress on today.

In the specific case of a bug with unclear steps to reproduce but very specific symptoms, we like to leave the issue open so that other people having the same problem can congregate together and maybe together we can figure out the underlying cause. This only applies to issues that have very specific symptoms like a specific and unusual crash signature, a specific and unusual error message, or other unusual and recognizable symptoms, and where some effort was made on the part of the bug reporter to determine the cause (even if that effort was ultimately futile).

Duplicates

If you recognize that this bug is a duplicate of an existing bug, add a reference to that bug in a comment, then close the bug. Skip the remaining steps. As you triage more and more bugs you will become more and more familiar with the existing bugs and so you will get better and better at marking duplicates in this way.

When closing the duplicate bug, the github issue tracker does not copy the list of people being notified on the closed bug into the original bug. This can matter, especially when asking on the original bug for things like reproduction steps. Consider cc'ing the author of the duplicate issue into the original issue, especially if we're still trying to determine reproduction steps for the issue.

Requests for help (documentation issues)

If the bug report is a question, then it probably belongs in Stack Overflow or on our #help channel or some other forum for getting help. However, if it appears that the reporter did try to read our documentation to find the answer, and failed, or, if you look in our documentation and find it is inadequate here, then please consider it a documentation bug (and update the summary accordingly).

If you are confident our official documentation (on flutter.dev or api.flutter.dev) fully answers their question, then provide a link to the relevant page and close the issue, being very polite and asking them to reopen if the documentation is not sufficiently clear for them.

Labels

General rule: The more labels an issue has, the better! *See also: List of labels*

Some labels are used to track the flow of issues from the time they're filed until they're assigned to a specific team for execution. You should use these to track the state of an issue through your first-level triage process. These are:

- `in triage`: You are presently looking at an issue and trying to determine what other labels you should give it.
- `assigned for triage`: The issue is assigned to a domain expert for further triage.
- `has reproducible steps`: The issue has a reproducible case or test, Flutter doctor output, and usable stack traces if appropriate. It is actionable in the sense that it can be routed to a domain team for action.
- `needs repro info`: We need more reproduction steps in order to be able to act on this issue.
- `will need additional triage`: Assign this if you don't know how to route it to a team.

Label the issue based on what area of the project the bug relates to:

- If it's a bug with the underlying engine, add `engine` and optionally one of the bugs with the "e:" prefix.
- If it's a bug with the Flutter framework, add `framework` and optionally one of the bugs with the "f:" prefix, especially:
  - If it's specific to Material widgets, include the `f: material` label.
  - If it's specific to iOS widgets, include the `f: cupertino` label.
- If it's a bug with plugins, add `plugin` and optionally one of the labels with the "p:" prefix.
- If it's a bug with packages, add `package` and optionally one of the labels with the "p:" prefix.
- If it's a bug with the "flutter" tool, add `tool` and optionally one of the labels with the "t:" prefix.

- If it's specific to writing desktop Windows, macOS, Linux, or Web apps with Flutter, add the `a: desktop` label.
- If it's related to a specific platform, add one of the "platform" labels. In particular, issues involving the Web backend should have the `platform-web` label.
- If it's related to our testing infrastructure (LUCI, Cocoon, devicelab, Cirrus, etc), add the `team: infra` label.

Bugs relating to the developer tools or Android Studio should be moved to the `flutter/devtools` repo, unless it looks like the first step is a change to the core parts of Flutter (in which case it should receive the `devtools` label as well as the pertinent labels for where the work should occur).

Bugs relating to the website should be moved to the `flutter/website` repo.

Bugs relating to the IDEs should be moved to the `flutter/flutter-intellij` repo.

Priority labels

Provide an initial priority for the issue. In general, the priorities you should consider are:

- `P4` for bona fide bugs in current functionality (including regressions)
- `P5` for feature requests.
- `P6` for feature requests we're unlikely to work on in the immediate future, according to our Roadmap.

If it looks like it's critical (something that stops further builds or is a major regression in core functionality affecting all users) consider the `P0` label. If you assign the `P0` label to a bug, you *must* reach out to a prospective owner to pass the baton for investigation. Try the `#hackers` channel on the Discord if you don't know someone who can help, or reach out to @Hixie. The `P0` bug is reserved for things that "stop the presses", as they also block further rolls to customers.

Like `P0`, we reserve `P1` and `P2` labels for things that should surface in our weekly critical triage meeting. In general, `P1` issues are bugs blocking top-tier customers from shipping immediately or breaks in core functionality affecting all users, and `P2` issues are bugs that block top-tier customers from shipping soon, or breaks in core functionality affecting most users.

`P3` issues are issues that have been triaged up from `P4` by team triage.

Additional labels

Once the main labels above are added, consider what additional labels could be added, in particular:

Add any of the applicable "severe: *" labels; typically only one will apply but sometimes `severe: regression` will apply in conjunction with one of the others.

Add any of the applicable "a: *" labels. There are many, it's worth browsing the list to get an idea of which ones might apply.

Issues requiring domain expertise

Sometimes it's not clear what the appropriate labels are for an issue, or you suspect it may be a duplicate but you lack enough domain knowledge to know for sure. If you know someone who has the expertise to triage the issue, you can assign the issue to them and add the label assigned for triage, so the person knows that they are not expected to fix it, but to triage it.

Additional comments

If you have something to say regarding the bug, for example if you happen to notice what the problem is, or if you have some insight based on having seen many other bugs over time, feel free to add a comment to that effect. Your experience is valuable and may help both the reporter and the rest of the Flutter team.

Triaging PRs

The process for triaging PRs is to look at this list:

1. the PRs with none of the classification labels.

When triaging PRs, it's important that incoming PRs:

- Contain tests for the functionality they're providing. If it's a bug fix, there should be tests to ensure that we don't regress the bug fix. If it's new functionality, the functionality should have tests as well.
- Code and tests should match our Style guide. Look carefully for spelling mistakes and violations of our coding style; invite the contributor to fix these.

For PRs, each PR should match one of the following categories:

- It can be in a specific repository (e.g. the website or engine repository) that has a narrow scope. These do not need labels. (Only the "flutter" repository is considered to not have a narrow scope currently.)
- It can have one of the following labels: `framework`, `f: material design`, `f: cupertino`, `tool`, `platform-web`, `a: desktop`, `team: infra`, `a: existing apps`.
- It can have the `will need additional triage` label. Use this if it's not clear what label it should have.

We should (but currently do not) regularly check that the following PRs are not being neglected, and have appropriate labels:

- The flutter/flutter PRs with none of the classification labels
- PRs in our esoteric repos
- All open PRs

Critical triage

During our weekly critical triage meeting, we check the following lists:

- P0: all bugs should be assigned, and progress should be happening actively. If no progress is happening and owner cannot work on it immediately (e.g. they're on vacation, they're busy with their day job, family reasons, etc), find a new owner.
- P1: all bugs should be assigned. There should be a recent (last few days) update on the issue. Blockers to addressing the issue should be identified. If no progress is happening and owner is not working on a P0 or other P1 but cannot work on it immediately, find a new owner.
- P2: all bugs should be assigned. There should be a recent (last couple of weeks) update on the issue. Blockers to addressing the issue should be identified. If no progress is happening and owner is not working on another P0-P2 but cannot work on this one immediately, find a new owner.

- Bugs flagged for additional triage: figure out what should be done with the bug, then remove the `will need additional triage` label.
- The stale PRs: examine the 25 least-recently updated PRs, if the least recently updated one was updated more than 2 months ago. Until early 2022, this ignores PRs with the `backlog` label, to allow teams time to catch up.
- The oldest PRs: examine the 25 oldest PRs regardless of when they were updated, to make sure that they are not falling into the trap of asking for updates every month, thus getting them marked as recently-updated.

Long-term triage

Occasionally, when time allows, the following are worth taking a look at also:

- PRs needing additional triage.
- Flakes: Check a few of the oldest flakes: are they still a problem (or is the test enabled and working now?). Is anyone looking at them?
- Most requested features: none of the top 20ish bugs are new, and they've all been considered before and have appropriate priorities.
- Most requested bugs: nothing surprising in the list; consider adding `customer: crowd` `P2` labels.
- Oldest bugs; see if they can be closed or updated.

In addition, a bot takes care of these, but it's good to check on them:

- Bugs pending feedback: close bugs with outstanding requests after 4 weeks, remove the label for bugs with answers.

Area-focused triage

We intend for each area of the product to go through the following triage regularly:

- Look at open PRs and review them.
- Look at open bugs and determine what needs to be worked on.

It is recommended to do these in separate dedicated meetings. For teams with multiple areas of focus (e.g. design languages), it's recommended that each area of focus have its own meeting.

Within each meeting, the links below provide the order in which issues should be looked at, to best align with the team's unique needs.

Teams can use a variety of tools to prioritize bugs.

- Teams should use the priority labels `P0` … `P6`.
- Another tool is the Projects page on GitHub. Feel free to create new projects as desired, and use it as you see fit.
- If you like using milestones, we can create new milestones. Contact @Hixie if that's interesting to you.
- The priority labels assigned to an issue indicate that the issue has been triaged by the relevant team.

Engine

Issue Triage

> *:warning: This triage happens on Mondays. The contents here are mirrored from the Google-internal document go/flutter-engine-triage-playbook and may be stale. Do **NOT** make changes to the instructions here as they will be overwritten. :warning:*

Agenda

- All issues in severity classes P0, P1 and P2 need weekly review.
- Fix priorities of ***regressions*** found in the releases still in the CP window (currently 2.8 and 2.9 and review each week).
- Assign priorities to unprioritized issues that have been tagged as regressions.
- Assign priorities to unprioritized issues that have been tagged as flakes.
- Assign priorities to unprioritized issues that have been tagged as fatal crashes and crashes.
- Assign priorities to unprioritized issues that have been tagged as performance issues.
- Catch unprioritized issues not caught by other dragnets to assign priorities.
- Shed a tear for all the other issues.
- Pitch your favorite issues.

Issue Quality Checklist

- Correct labels.
- If you had to remove the engine label, stop and move on to the next issue.
- If you had to remove a label that makes the bug no longer qualify for triage, stop and move onto the next issue.
    - For example, if you were looking for unprioritized regressions but the issue is not a regression, remove the label and move on.
- Hide low quality comments.
- If an assignee is not necessary or cannot be found, find someone on the team that can be CC-ed if you think it's necessary. CC folks sparingly. If you CC folks too much they will understandably stop responding. Don't CC the same person more than a few times per triage run.

Tips

- Restrict time spent on each issue. You don't have to fix the issue!
    - 30 seconds for P0, P1, P2 and 10~15 seconds for the others seems to work.
- If the triage process cannot be completed within the allotted time, the process has failed. Perform items that could not be completed offline. Seek help.

Pull Request Triage

> *:warning: This triage happens on Thursdays. The contents here are mirrored from the Google-internal go/flutter-engine-pr-triage-playbook and may be stale. Do **NOT** make changes to the instructions here as they will be overwritten. :warning:*

Objective

Make sure Flutter Engine Pull-Requests on GitHub are getting adequate attention, especially ones from external (non-Googler) contributors.

Agenda

- After completing each step, fill out the relevant entry in the worksheet so progress can be tracked.
- Take a look at (the relatively few) buildroot pull requests.
- Attempt landing all pull requests that have been approved.
    - Ensure all presubmits pass.
        - If there is a presubmit flake, file a report and re-run presubmits.
    - Ensure there are no conflicts.
        - If there are conflicts, ask the author to rebase.
        - If a previous request to rebase has not been addressed, close the PR as stale.

    - If the author is a Googler, ping them with a request to land it.
    - If the author is a non-Googler, add the “waiting for tree to go green” tag for the bot to land the PR.
- Address un-approved non-draft pull requests that have been least recently updated.
    - If the PR is related to the web engine, add the platform-web tag and move on.
    - If the author has indicated that the PR is a work in progress, add the “work in progress” tag and move on. Do **not** mark the PR as a draft as that will disable presubmits.
    - If a reviewer has requested changes that havent been addressed, ping the author for an update.
        - If a preview ping for an update has not been addressed, close the PR as stale.
    - If a reviewer has requested changes that have been addressed, ping the reviewer for an approval.
    - If there are no reviewers, assign a reviewer familiar with the relevant subsystem.
- Address draft pull-requests that have been least recently updated.
    - If the pull request has no recent updates, ping the author if any progress is likely.
        - If a previous ping was unanswered, close the PR as stale.

Design languages

Material Design

- Material Design PRs
- P0 bugs
- P1 issues
- P2 issues
- Flakes
- Regressions
- Crash bugs
- Consider issues with the annoyance and quality labels
- Popular issues (look at the top 10ish)
- Glance at the Recent issues (to see if any unexpected trends show up)
- If you get this far, triage all the other Material issues