


Flutter project proposed new triage processes for 2023

February 2023 - Ian 'Hixie' Hickson - Flutter Team - flutter.dev/go/triage-2023-rfc

This proposal has been put into place.

Introduction	2
Confusion about priority labels	2
Confusion around triage	3
Low quality bug reports	3
Terminology	3
Background	4
Triage processes	4
Priority hygiene	5
Assignee hygiene	5
Priority labels	5
P0-P2	6
Flakes	6
P3	7
P4-P6	7
Proposal	9
Priorities - this section has been deployed! 	9
Adjusting the scale	9
Merging P0, P1, and P2	9
Merging P5 and P6	10
Mechanics of changing the priority scale	10
Automations related to priorities	10
P1 (old P3)	10
Missing priority labels	10
Stale issues and candidates for closure	10
Complete proposal for priorities	10
Triage changes	11
Team labels	11
Naming	12
Assignments	12
Process for triage	12
Incoming issues	12
P0 and assigned issues	12
Additional labels	13
Example	13
First-level triage	13
Critical triage	13

One-time migration 🕒	14
Automation relating to triage	15
Maintaining invariants	15
Staleness checking	15
Thumbs-up	16
Additional automations	16
Flaky tests	16
Cleaning up the labels	16
Labels to rename, merge, and document	16
Obsolete labels	19
Heuristics for bugs to keep open vs bugs to close	21
Deployment Plan	21
Automations	22
Pseudo-code for listener-based bot	23
Pseudo-code for polling-based bot	24
Necessary APIs	25
old notes	25
Discussion	26
Team labels	26
Changing our priority label spectrum	26
Renaming labels	26
Automated closures	26
VSCode	26
Kubernetes	26
Other discussions	27

Introduction

This proposal relates to the issues in the flutter/flutter issue database. The Flutter project has some other bug databases, notably the one in the flutter/website repository, as do some contributors (e.g. Google has its own bug database for Flutter issues raised by Google teams). Dart also has its own issue databases. These are all considered out of scope for now.

There are several areas to be considered, as described in this section.

Confusion about priority labels

It's not always clear who should be adding priority labels.

Our priority labels are "off by two" from the industry norm.

It's not clear exactly what P3 and P4 mean in practice.

It's not clear exactly how to distinguish between P4 and P5, and P5 and P6.

We are inconsistent about whether priorities say how long until a fix is expected, how likely a fix

is expected, or how important we think an issue is.

Many issues, including many from recent years, are lacking priority labels entirely (>2400, ~21%). We use our priority labels to indicate to our users the relative importance of issues. This therefore represents a failing in our promise to communicate transparently, which we would like to address.

There is confusion and unhappiness around our policy of automatically filing flake reports as P1.

Confusion around triage

Some subteams use the priority label as an indication that they have looked at the issue. If someone else adds the priority label, or if the issue is transferred between teams, the new team may never see the issue.

Some issues are assigned to multiple teams. It's not always clear which team has responsibility for an issue.

It's not obvious which labels are "team" labels vs which are metadata.

It's not clear who is responsible for championing "customer: crowd" and "customer: product" bugs, i.e. bugs that do not directly affect "top tier" customers but do affect a large number of customers or have an impact for the product to deliver on key promises.

Low quality bug reports

There is a concern that we have many open issues that are stale, obsolete, duplicate, or unactionable.

Stale and obsolete bugs are those that just don't apply any more. Maybe the relevant code has gone away, or the bug was fixed in an unrelated effort.

Unactionable bugs in particular are those that describe issues that nobody working on the project is able to reproduce, such that there is literally nothing useful that anyone on the project can do to make progress on the bug. They can also be issues that are so vague that no particular action can really be taken to resolve it.

Keeping these bugs open is not valuable and artificially bloats our issue database, making searches harder, costing valuable time during triage, and generally making planning more difficult. For these reasons, we hope to be able to clean our issue database this year to increase the average quality of the bugs we have open, closing these low-quality bugs that we are never going to usefully pay any attention to.

Terminology

In this document, the terms "bug", "bug report", and "issue" are all synonymous and refer to an entry in the flutter/flutter GitHub issues database.

The term "team" refers to a group of contributors that either owns a part of the codebase in the Flutter project, and/or that triages issues in the Flutter project, as described on the [Project](#)

[Teams](#) page of our wiki.

Background

Triage processes

Our relevant documentation is split between the [Triage](#) wiki page and the [Issue Hygiene](#) wiki page.

Currently our practices vary on a team-by-team basis, as follows (issue counts are very approximate and are presented only for comparative purposes, as they are continually changing):

Team	Labels	Triage process	Open Issues	Missing P labels
Engine	engine -platform-web -a: desktop	P labels indicate triage progress	2129	18 (<1%)
Design Languages (Material and Cupertino)	"f: material design" "f: cupertino"	Look at all new issues	1716 287	1235 (~72%) 202 (~70%)
Framework	framework -f: material design -f: cupertino -platform-web	P labels indicate triage progress	2412	42 (<2%)
Tool	tool	P labels and "has reproducible steps" indicate triage progress; P3+ must be assigned	1766	356 (~20%)
Web	platform-web -tool	P labels, assignee, waiting labels indicate triage progress	837	46 (~5%)
Android	platform-android -framework	P labels indicate triage progress	1030	4 (<1%)
iOS	platform-ios	Look at all new issues in real time (subscribing using Google-internal tool)	1118	200 (~18%)
Ecosystem	plugin package	P labels indicate triage progress P labels indicate triage progress	1507 380	3 (<0.2%) 2 (<1%)
Desktop	"a: desktop"	P labels indicate triage progress	617	5 (<1%)
Infrastructure	"team: infra"	P labels indicate triage progress ¹	427	245 (~57%)
Go Router	p: go_router	Project inclusion indicates triage	113	1 (<1%)

¹ As documented, but the high proportion of unprioritized bugs suggests otherwise.

Package		progress		
Release team	team: release	"passed secondary triage" and priority levels indicate triage progress (implicitly)	37	22 (~60%)
Google Testing	team: google testing	"passed secondary triage" and priority levels indicate triage progress (implicitly)	5	4 (~80%)
Codelabs	d: codelabs	No documented process	16	8 (~50%)
News Toolkit	news_toolkit	Looks at all issues and assigns priorities	4	0 (0%)

This leaves about [356 issues](#) not in any team's triage bucket, [105 of which](#) are marked as having passed first triage. The following labels seem to sometimes be used as if they are team labels, though they are not mentioned in the triage docs as having triage processes associated:

- a: triage improvements (mostly about updating the issue templates)
- d: devtools (these are either issues that should be moved to flutter/devtools, or issues that affect devtools but are not on the devtools team's backlog)
- d: intellij (these are either issues that should be moved to flutter/flutter-intellij, or issues that affect that IDE plugin but are not on the relevant team's backlog)
- d: conductor (no longer in use)
- TPgM (no longer in use)

We also have some variety in terms of whether teams use GitHub project boards for planning.

Priority hygiene

As shown in the table above, some teams have been regularly prioritizing issues, while some others have not. In practice it seems that prioritizing issues is a long task, but doable.

Assignee hygiene

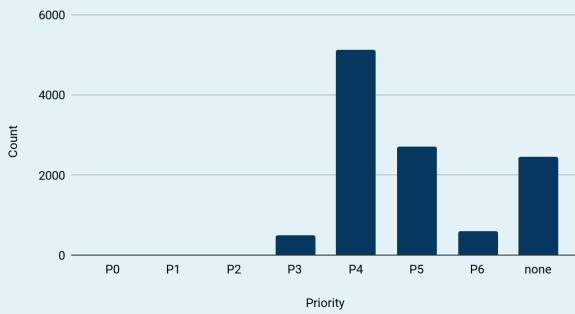
We have 896 issues currently assigned. Hundreds of them [have not been updated in literally years](#). We have bugs labeled "assigned for triage" that have [not been touched for months](#).

While there are some exceptions (issues that people filed for their own purposes), some automation to automatically unassign issues that have not been touched for several months is probably worth considering.

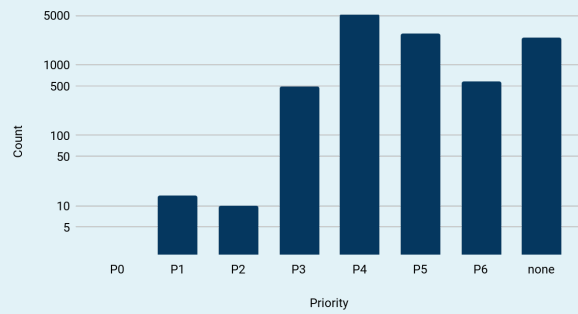
Priority labels

Our distribution of priorities is currently as follows (linear scale on the left, log scale on the right; this is just a snapshot, of course, these will change over time):

flutter/flutter issues by priority

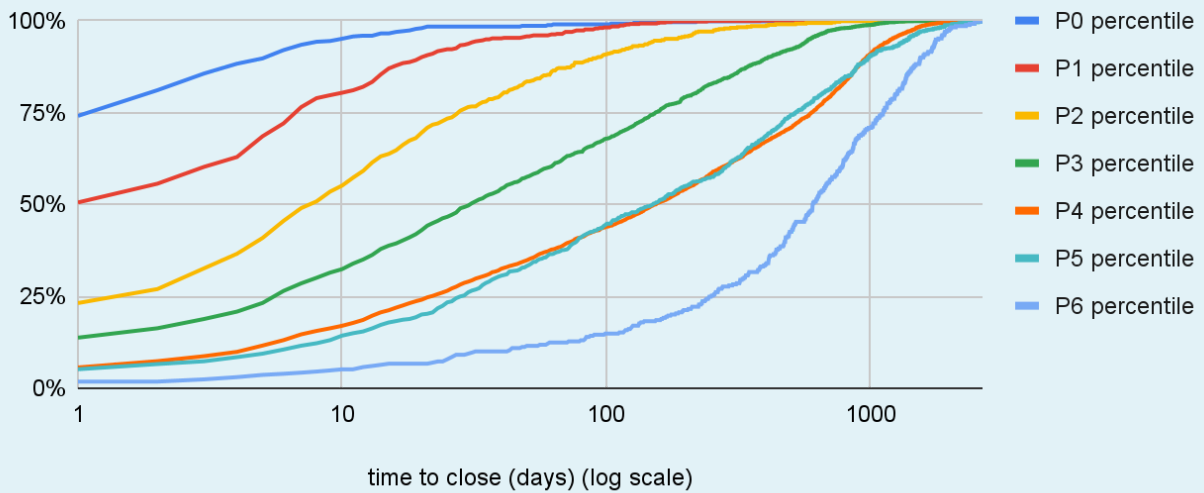


flutter/flutter issues by priority (log scale)



Our time to closing bugs is as follows (this is a cumulative percentile histogram with a logarithmic time axis; for each priority, given an amount of time, the given percentage of issues at that priority level are closed in less time than that):

Time to close issues by priority (cumulative percentile histogram)



P0-P2

(These levels correspond roughly to P0 in most projects. We split them into three levels because they have different expectations around update frequency.)

We expect very few P0-P2 issues; these have tight expectations and are tracked aggressively to avoid issues falling through the cracks. P0-P2 represent critical issues either for the health of the project or for high-profile Flutter customers (e.g. those that are being supported by a contributing team; the canonical example being Google product teams that are supported by Google's Flutter engineers).

The current situation around P0-P2 is relatively under control. We have very few open bugs at these levels, and we address them promptly. There's a clear distinction between the levels, too, as seen on the time-to-close graph above.

Flakes

We automatically file some issues as P1 (or P2) when we detect flakiness. Historically this has had some utility but more recently the impression from the team is that the majority of bugs

being filed for flakiness have a significantly higher cost to resolve than the benefit that resolving them would have (e.g. creating a replacement for chromedriver).

P3

(This level corresponds roughly to P1 in most projects.)

P3s are supposed to represent "high-priority issues that are at the top of the work list". In practice we have [numerous bugs](#) that are labeled P3 and have not been updated for years.

There are probably several reasons for this:

- the issue might have been resolved,
- the issue might have dropped through the cracks (e.g. the assignee left the team and we never reassigned it),
- the issue might not really be as high-priority as it was first assumed,
- there might be too many high-priority issues in that area.

For the first two represent a process failure for which getting eyes on the issue again will likely result in a quick resolution. The third similarly would be resolved by the issue being re-triaged and given a lower priority.

The fourth is a difficult situation, but the reality is that if all the issues are high priority, then none of them are, so really it's probably the same as the third option.

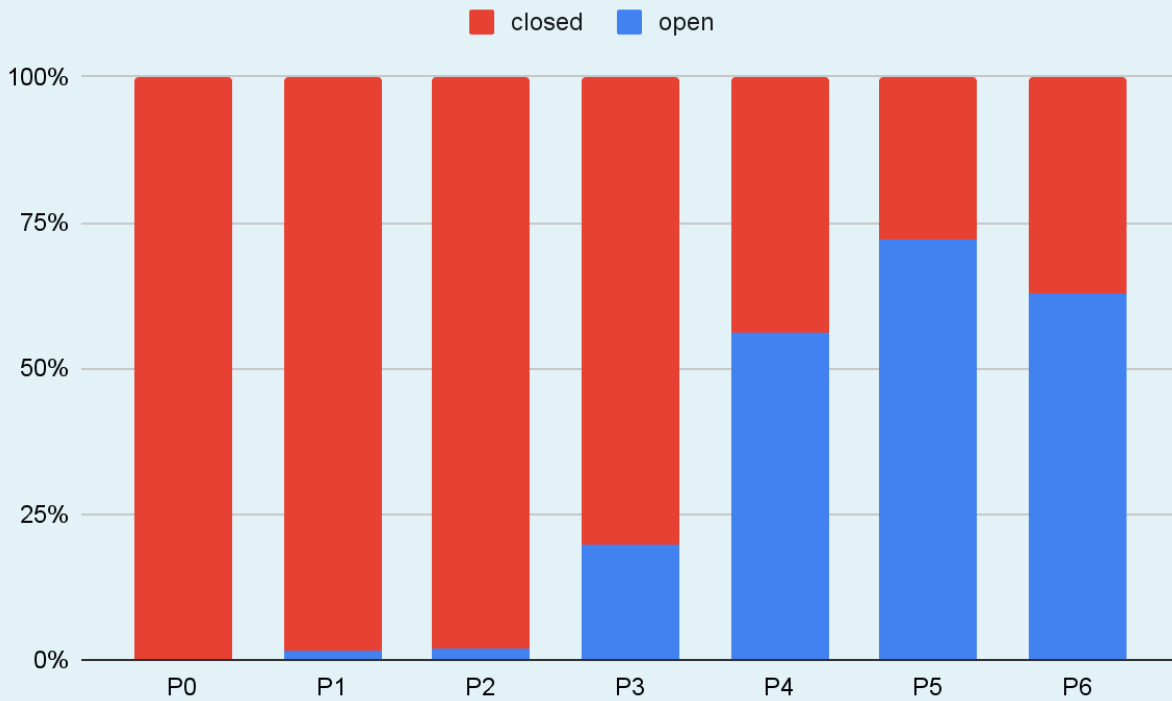
All of which is to say that it probably makes sense to have automation around handling P3 issues.

P4-P6

(These priority levels correspond roughly to P2-P4 in most projects.)

How useful the distinctions between P4, P5, and P6 are is up for debate. In general it seems useful to have some indication to users about whether an issue is considered a normal bug (P4) or something less likely to be fixed (P5, P6). Whether separating P5 and P6 into separate buckets is useful is unclear.

We do have data that can help us understand the distinction between these levels. First, the fraction of P5s that get closed is roughly the same as the fraction of P6s that get closed, and a lot less than the fraction of P4s that get closed. This is not actually obvious in the data:



...because the data makes it look like we actually close P6s *more* than P5s, but it turns out that there was a single event (Stuart closing a bunch of plugin-related issues after labeling them all P6) that visibly affects the data, and when you filter those out the number of closed P6s drops to more or less match the fraction of closed P5s. This suggests that P5s and P6s are about as likely to get fixed.

The cumulative percentile histogram graph from earlier tells a different story, though: P4s and P5s have about the same curve, distinctly different from all the other levels. This suggests that those P5s that we *do* close get treated just like P4s as far as priorities go.

There's also the earlier graph showing that overall we have way fewer P6s than P5s (and fewer of those than P4s).

Proposal

The wiki will be updated to describe the process described herein. Any links in the wiki, notably on the Triage page but also anywhere else, would be updated accordingly. Tooling will be created as described below.

There are several broad changes being proposed here: a simplification of our prioritization schema, a unification and normalization of our triage processes, a label cleanup, and a stricter policy concerning what is considered a valid bug.

Key proposals are highlighted.

The proposal is being deployed. Decisions are annotated as follows:

- ✅ - done!
- ▶️ - actively being deployed right now!
- 🕒 - ready but not yet deployed
- ⏸️ - not yet done but still planned
- 🚫 - no longer planned

Priorities - this section has been deployed! ✅

Adjusting the scale

One of our problems is that what we call "P4" is what most teams call "P2", and so people frequently misunderstand what we mean when we've labeled an issue with a priority level (and similarly for all the other levels). The obvious solution is to shift all the labels over by two.

Merging P0, P1, and P2

The naive way to shift the 7 priority levels over by two would result in what we call P0 today being called P-2. The problem is that "P-2" looks an awful lot like "P2", and I think that would be very confusing.

One solution is to find better names for P-2 and P-1. Ideally these names would be clearly ordered, short (2 or 3 characters), and unambiguously more important than the others.

Ideas so far:

- P000, P00, P0, P1, P2, P3
- P0A, P0B, P0C, P1, P2, P3
- P0++, P0+, P0, P1, P2, P3
- P0!!, P0!, P0, P1, P2, P3
- P 🚨, P 🔥, P0, P1, P2, P3

If we reduce the number of flaky test issues that end up in these buckets, the number of actual issues at these levels becomes low enough that having just one priority is probably fine, however. **Therefore, we shall merge our three top levels into one.** ✅

Given that we only ping people for updates once a week anyway, and given that P0 and P1 issues were not getting more frequent updates, typically, despite the general policy around those, this actually brings us more in line with our real behavior.

Merging P5 and P6

To reduce the confusion around P5, we merge that label with P6 and remove P6. ✓

Mechanics of changing the priority scale

The obvious way to implement the solution to our "off-by-two" issue is renaming the issues. ✓ Previously, GitHub's bugs made that questionable ([see below](#)), but recent fixes from GitHub make this viable again. (Actually, nothing has been fixed, but we have a way to get them to reindex our issue database so it's not a permanent mess.)

Alternatively, we could manually rename labels using a bot to scan through all the issues. This would take days (given GitHub's API rate limits), during which time our bugs would be in a very inconsistent state, and during which time anybody trying to search for bugs by priority, or set a bug's priority, would likely be unable to do it correctly, but we could mitigate some of these issues by transferring issues one label at a time. First, create PX-2, and cycle through all bugs with PX and remove PX and add PX-2. Then, delete PX, and restart with the next pair (PX-1 and PX+1), etc. With a stickied issue saying what is happening and which labels to stay away from, we could probably do this without too much of a disaster.

Automations related to priorities

P1 (old P3)

To help P1 (P3 in the old scale) maintain its meaning as "high-priority", we install some automation that removes the "triaged-team" label when a P1 issue has not been touched by any team member in over 5 months². ✓ It's the responsibility of the team to verify that the priority is appropriate, and if so, to adequately schedule its work in the near-term.

Missing priority labels

We also add some automation to remove the "triaged-team" label of issues that don't have any priority labels ✓ (but in a rate-limited way to avoid overwhelming teams and drowning out new issues).

Stale issues and candidates for closure

Eventually we may find heuristics that indicate issues that are good candidates for being closed (e.g. if we deprecate a feature, then issues related to bugs in that feature are likely no longer relevant). If we do have such heuristics, the automation should re-request triage for these issues, in a strongly rate-limited fashion.

Complete proposal for priorities

The new definitions are as follows. Notable new text is highlighted.

² This may be too long. We can adjust this over time.

Priority	Definition
P0	<p>The P0 label indicates that the issue is one of the following:</p> <ul style="list-style-type: none"> • a build break, regression, or failure in an existing feature that prevents us from shipping the current build. • an important item of technical debt that we want to fix promptly because it is impacting team velocity. • an issue blocking, or about to block, a top-tier customer. (See [above](https://github.com/flutter/flutter/wiki/Issue-hygiene#customers) under "customers" for a definition of "top-tier customer".) <p>There are generally less than twenty-five P0 bugs (one GitHub search results page). If you find yourself assigning a P0 label to an issue, please be sure that there's a positive handoff between filing and a prospective owner for the issue. Issues at this level should be resolved in a matter of weeks and should have weekly updates on GitHub. ✓</p>
P1	<p>The P1 label indicates high-priority issues that are at the top of the work list. This is the highest priority level a bug can have if it isn't affecting a top-tier customer or breaking the build. Bugs marked P1 are generally actively being worked on unless the assignee is dealing with a P0 bug (or another P1 bug).</p> <p>Issues at this level should be resolved in a matter of months and should have monthly updates on GitHub. ✓</p>
P2	<p>The P2 label indicates issues that we agree are important to work on, but not at the top of the work list. This is the default level for new issues. A bug at this priority level may not be fixed for a long time. Sometimes an issue at this level will first migrate to P1 before we work on them, but that is not required. ✓</p>
P3	<p>The P3 label indicates valid issues that we currently consider unimportant. We use "thumbs-up" on these issues as a signal when discussing whether to promote them to P2 or higher based on demand. ✓</p>

We also add a label "would require significant investment" to indicate that we would not accept a random PR to fix the issue, regardless of priority, unless it came with a significant commitment of resources to ensure ongoing maintenance. ✓

Triage changes

Team labels

We introduce a set of mutually-exclusive (by convention, similar to the priority labels) set of labels that unambiguously indicates who owns an issue. ✓ Each issue must have one of these labels; the lack of such a label indicates that the issue needs initial triage (unless it is assigned or labeled as needing additional triage).

For each such ownership label, there is a corresponding "triaged" label that indicates that the relevant team has handled the issue during triage ✓, and an "fyi" label that forces the issue into the team's triage pile even though it's not the team ✓.

That way, teams can unambiguously ask for another team's input, we have an unambiguous list of teams, and each bug has an unambiguous owner.

These changes still leave the area labels (e.g. "engine", "framework", etc) unaffected, so teams

who wish to observe other labels as part of their triage can do so.

Naming

The labels take the form of "team-*team*", "triaged-*team*", and "fyi-*team*". The teams are those that have an active triage process, namely:

- team-android
- team-codelabs
- team-design
- team-desktop
- team-ecosystem
- team-engine
- team-framework
- team-google-testing
- team-gorouter
- team-infra
- team-ios
- team-news
- team-release
- team-tool
- team-web

Assignments

A bug that is assigned is considered triaged, regardless of other signals. This allows people to file bugs to track work without incurring the cost of the triage process.

Process for triage

Incoming issues

To triage incoming issues, teams can either use tooling to watch labels asynchronously (notifying them whenever the "team-*foo*" or "fyi-*foo*" labels are added to an issue), or have a dedicated meeting in which all issues matching this query are examined (where "foo" is the team name):

```
is:issue is:open
label:team-foo,fyi-foo -label:triaged-foo
no:assignee -label:"will need additional triage"
```

i.e. open issues that have either the "team-*foo*" or "fyi-*foo*" label (the comma is an "or" in GitHub searches), but not the "triaged-*foo*" label, that are not assigned, and that are not labeled for handling at the critical issue triage meeting.

For each incoming bug, teams should consider whether the bug is worth keeping (see the heuristics below), should provide feedback to the reporter as appropriate, should give the issue a priority, scheduling work if appropriate, and then should add their team's "triaged-*foo*" label to the issue to mark it as triaged, taking it out of the team's queue.

P0 and assigned issues

On a weekly basis, teams should also look at all P0 issues in their area, ensuring they are assigned, and should look at all assigned bugs in their area (including P0s), to ensure they are making progress and have regular updates. P0 issues should have at least one update per week. Assigned bugs should have at least one update per month (if the update is "no update" then the issue should probably be unassigned). P0 issues that are not making progress should be escalated, P0 issues that are wrongly prioritized should have the priority labels changed, and assigned issues that are not being worked on should be unassigned.

Both of these tasks can be done asynchronously, or in a synchronous meeting, as desired by the team's members.

Additional labels

Teams may look at recently added or recently updated issues with labels related to their work, and see if any of them require their attention.

Teams can add the "will need additional triage" label to indicate that the issue should be examined during the weekly critical issue triage meeting. For example, if it is not clear if an issue is sufficiently actionable.

Teams may have additional team-specific triage steps. For example, looking at issues labeled as regressions specifically, or examining issues with many thumbs-up reactions.

Example

For example, front-line triage might add "team-engine" to an issue. The engine team would then add "triaged-engine" once they took a look at it and prioritized it appropriately. If the engine team wanted the framework team to look at the issue, they might add "fyi-framework" to the issue. The framework team would then find this issue in their triage, comment accordingly, and then add "triaged-framework" (at which point the bot would remove the two framework labels).

First-level triage

Prioritisation is removed from the responsibilities of first-level triage (this is reflecting the current reality and not a change in actual processes).

The incoming queue for first-level triage is defined as the list of bugs that do not have an appropriate team label, are not assigned, and do not have the "will need additional triage" label. The first-level triage team should add whatever labels are appropriate to an issue, including one (and only one) team-level label.

When an issue cannot be fully triaged by the first-level triage team, the label "will need additional triage" is applied. It is then examined during the weekly critical issue triage meeting.

Critical triage

Each week, the critical triage meeting will examine all P0 issues to ensure they are assigned, are making progress, and have had an update in the last week.

The same meeting will then examine bugs that are labeled "will need additional triage", applying whatever judgements are being requested.

Finally, this meeting will examine the least-recently-updated PRs across the organization to ensure they have not been forgotten.

One-time migration 🕒

To aid this process, we make a few one-time changes.

First, we add a team label to all the issues that are currently being handled by a secondary triage team, as follows:

Issues with labels...	...are assigned the label for the team...
engine -platform-web -"a: desktop"	team-engine
"f: material design"	team-design
"f: cupertino"	
framework -"f: material design" -"f: cupertino" -platform-web	team-framework
tool	team-tool
platform-web -tool	team-web
platform-android -framework	team-android
platform-ios	team-ios
plugin	team-ecosystem (by renaming "ecosystem" label <input checked="" type="checkbox"/>)
package	
"a: desktop"	team-desktop
"team: infra"	team-infra (by renaming the "team: infra" label <input checked="" type="checkbox"/>)
p: go_router	team-gorouter
"team: release"	team-release (by renaming the "team: release" label <input checked="" type="checkbox"/>)
"team: google testing"	team-google-testing (by renaming the "team: google testing" label <input checked="" type="checkbox"/>)
d: codelabs	team-codelabs (by renaming <input checked="" type="checkbox"/>)
news_toolkit	team-news (by renaming the "news_toolkit" label <input checked="" type="checkbox"/>)

Next, we add "triaged-team" labels to account for the old processes used by the teams, as

follows:

Team label	Action
team-engine	Add triaged-engine to issues with priorities.
team-design	Add triaged-design to all issues since that predate the last triage meeting.
team-framework	Add triaged-framework to issues with priorities.
team-tool	Add triaged-tool to issues with priorities.
team-web	Add triaged-web to issues with priorities.
team-android	Add triaged-android to issues with priorities.
team-ios	Add triaged-ios to all issues.
team-ecosystem	Add triaged-ecosystem to issues with priorities.
team-desktop	Add triaged-desktop to issues with priorities.
team-infra	Add triaged-infra to issues with priorities.
team-gorouter	Add triaged-gorouter to issues in the project.
team-release	Add triaged-release to issues that have a priority or the "passed secondary triage" label.
team-google-testing	Add triaged-release to issues that have a priority or the "passed secondary triage" label.
team-codelabs	None of the issues.
team-news	Add triaged-news to all issues.

Automation relating to triage

Maintaining invariants

Issues that have more than one ownership label should have them all removed, sending the issue back to primary triage. ✓

Issues that have one of the "fyi" labels and the corresponding "triaged" label have both removed automatically. ✓

We occasionally create an issue that has all the team labels on it, with instructions to just mark it as triaged. After a week, we check to see if the issue has been triaged by all the teams. This will let us catch cases where we have decommissioned teams but forgotten to update the list of labels. ✓

Staleness checking

We install some automation that ensures that any issue that has not had a comment from the assignee in over 5 months is automatically unassigned and has its "triaged-team" labels removed to cause it to be re-examined during regular triage, unless it was filed by the person to whom it is assigned and that person has commit access. ✓

Thumbs-up

We would like to add some automation to remove the "triaged-team" label of issues that have recently changed their number of thumbs-up by an order of magnitude since the last time the issue was triaged. ✓

Additional automations

We could consider other automations, too. For example, if an issue is in the top-20 by thumbs up, and it has not received an update from a team member in more than 6 months, remove the "triaged-team" labels to trigger a re-evaluation. (This could be done by getting the list of top-20 issues and comparing it to what we know about those issues' last update dates.)

Similarly, if we add a label to indicate issues that are tracking design docs, the triage labels could be removed after a few months to get the team to decide if they should be closed. (This is similar to just doing that for all issues that haven't been touched for a few months, but presumably we would have a shorter timeline for design doc issues.)

Flaky tests

We change the convention around automatically filing flake issues as follows:



- The one or two issues for tests with the highest rate of flakiness each week are marked P0 by the bot when the issue is filed. Other issues do not get a priority label. (There are various ways this could get implemented, e.g. each week the threshold for considering a flake to be P0 is set to be 99% of the rate of the most flaky test the previous week, or we could have a hard-coded threshold that is manually adjusted over time.)
- The flakiness bot never retroactively adjusts the priority on any issue, though it still keeps posting updates to open issues to report the current rate of flakiness.
- Issues are filed assigned to the test owner, as today. (This means these issues do not appear in any triage meeting other than critical triage, as discussed later.)
- When a flaky test's flakiness rate goes to zero, a new label, "r: no longer flaky", is added to the corresponding issue, and the bot stops updating the issue until the rate increases above zero again. An issue that has that label and has not been updated for two weeks is automatically closed.

























Cleaning up the labels

As part of this, we audit the labels and remove largely-unused labels, labels we have not recently used, and labels we use regularly but that have no documented purpose. For example, "passed first triage" currently gets a lot of usage but is not documented. The "team" label, similarly, ends up being used as a catch-all with no real purpose.

Labels to rename, merge, and document

We will rename and otherwise fix up some labels to bring more consistency to the names:

Current name	Change
 platform-bigsur	Remove emoji for consistency with other platform-* labels. ✓
 platform-catalina	
arch: m1	Remove. ✓
a: multi window	Add description. ✓
a: state restoration	Add description. ✓
ask: dart	This is used only a few times per year. Merge it with "dependency: dart". ✓
blocked	Seems to have been created relatively recently, but is not documented in the wiki. Update the wiki's discussion of blockers (e.g. on the Triage page) to mention it. ✓
blog	Created recently and only rarely used, probably because it's not documented anywhere. Add documentation for (to onboarding materials, on the wiki, etc) about how to get something into release notes / release blogs, then remove the label. ✓
branch info	Remove. ✓
created via performance template	This label's description is out of date (the link is 404). Update the description. ✓ Rename to "from: performance template" for consistency. ✓ Update template. ✓
d: cookbook	Remove in favor of flutter/website. ✓
d: website - content	
d: website - infra	
dart:io	Rename to "dependency: dart:io". ✓
debugging_web	Name doesn't follow any current conventions. Label has no description or documentation. Seems to be redundant with "a: debugging" and "platform-web". Add those labels then delete this one. ✓
device doctor	Description is vague. No documentation. Rename to "infra: device doctor", improve description. ✓
device-lab	No description or documentation. Add description and rename to "infra: device lab". ✓

documentation	There's about 282 open issues with this label and none of the other d:* documentation labels. We should apply the "d: api docs" label to all these issues  and then remove this label.
easy fix	Fix all of them  then remove the label.  We have another label for good new contributor bugs.
ecosystem	Rename to team-ecosystem. 
entertaining new contributor project	Merge with "good first contribution" 
flutter.js	Rename to "e: flutter.js". 
gallery	The gallery has its own issue database. Issues specifically with the gallery should be moved there (looks like there aren't any). Rename this label to "customer: gallery". 
good first contribution	Add to wiki (it's already in our onboarding documentation). 
hc-only	Rename to "platform-views: hc". 
impeller	Rename to "e: impeller". 
infra auto flakes	Rename to "infra: auto flake bot" and fix description to start with a capital letter and be clearer. 
infra metric	Rename to "infra: metrics" and fix description to start with a capital letter and be clearer. 
integration_test	Rename to "f: integration_test" 
local engine development	Rename to "team: local engine development". 
p: animations	Add description. 
p: cross_file	Add description. 
parity	Rename to "c: parity". 
pigeon	Rename to "p: pigeon". 
plugin	Add "package" label to all issues with the "plugin" label, then delete "plugin". 
postmortem	Redundant with "from: postmortem". Add that  , then remove. 
sanitizer	Rename to "from: sanitizer". 
several labels starting with "severe: "	Rename to start with "c: " instead of "severe: "; they represent categories rather than severities. 
team: release	Add description. 

team	Bugs labeled "team" that are about technical debt get labeled "c: tech-debt". 🛑 The "team" label is then removed. 🛑 [blocked on reindexing, right now we can't usefully search for issues]
team: security	Rename to "infra: security". ✅
tech-debt	Rename to "c: tech-debt". ✅
tlhc-only	Rename to "platform-views: tlhc". ✅
vd-only	Rename to "platform-views: vd". ✅
web-host-platform-specific	Add existing "c: parity" label with which it is redundant, ✅ then remove this one. ✅

Obsolete labels

In addition, we outright delete these labels:

- "passed first triage" 🛑 and "passed secondary triage" 🛑 are deleted.
- "d: conductor" ✅ and "TPgM" ✅ are deleted.
- "enhancement" ✅ and "transferred" ✅ are deleted.
- "macos-metal" is deleted. ✅
- "d: website - content" is deleted in favor of flutter/website's issue database. ✅
- The following labels that have never been used and are not documented anywhere are deleted:
 - autoroller: dryrun (description is "dryrun test by engine to flutter auto-roller") ✅
 - created via support template (description is "Issue was created via github.com/flutter/flutter/blob/master/.github/ISSUE_TEMPLATE/SUPPORT.md" but that path is 404) ✅
- The following customer labels haven't been used in any issues that have been updated in the last 12 months and have no open issues, they are deleted: ✅
 - customer: aptree ✅
 - customer: boing ✅
 - customer: cradle ✅
 - customer: fashion ✅
 - customer: fire ✅
 - customer: gather ✅
 - customer: grand (eap) ✅
 - customer: green ✅
 - customer: headline ✅
 - customer: heart ✅
 - customer: homestead ✅
 - customer: indigo ✅
 - customer: latfin ✅
 - customer: look ✅
 - customer: marketplace ✅
 - customer: merch (g3) ✅

- customer: moon ✓
- customer: pachyderm ✓
- customer: paper ✓
- customer: payouts ✓
- customer: placard ✓
- customer: price ✓
- customer: public-app ✓
- customer: puzzle ✓
- customer: raven ✓
- customer: slides ✓
- customer: stories ✓
- customer: truckable ✓
- customer: truth (g3) ✓
- customer: udacity ✓
- customer: wellbeing ✓
- customer: windy ✓
- customer: workplace ✓
- customer: zebra ✓
- The following non-customer labels that haven't been used in any issues that have been updated in the last 12 months are deleted. Unless otherwise noted, there are no open issues with these labels.
 - dependency: firefox (**one issue, also labeled "browser: firefox"**) ❌
 - found in release: 2.7 ❌
 - jira (**one open issue, redundant with platform-fuchsia**) ✓
 - p: android_alarm_manager
 - p: android_intent
 - p: battery
 - p: cloud_functions
 - p: connectivity
 - p: device_info
 - p: firebase_admob
 - p: firebase_analytics
 - p: firebase_core
 - p: firebase_crashlytics
 - p: firebase_database
 - p: firebase_dynamic_links
 - p: firebase_messaging
 - p: firebase_ml_vision
 - p: firebase_performance
 - p: firebase_remote_config
 - p: firebase_storage
 - p: location_background
 - p: package_info
 - p: sensors
 - p: sentry
 - p: share
 - p: wifi_info_flutter 🛑
 - t: shrink ❌

- team: bimodal benchmark ✓
- The following labels have only rarely been used and aren't documented anywhere are deleted:
 - CQ+1 (description is "Pull request is ready for tryjobs", 29 PRs, only one in the last year, none in the last 3 months) ✓
 - autoroller: commit (description is "commits by engine to flutter auto-roller", lots of PRs in the past, but only 4 in the last year and none in the last 3 months) ✓
 - needs tests (apparently used by automation on other repos, but flutter/flutter doesn't seem to use it) ✓

Heuristics for bugs to keep open vs bugs to close

Part of the new triage process will be being more aggressive about closing bugs (both for front-line triage and for secondary triage and re-triage of old issues).

As part of this we will document guidelines ✓ on what issues to close. This section provides a start for these guidelines.

Reasons to close:

- The issue makes multiple requests which could be addressed independently. Prefer GitHub projects instead of umbrella bugs or shopping lists of features. Encourage people to file separate bugs for each independent item.
- It's a feature request that we are extremely unlikely to ever address, and if we did address it we would not be part of the core SDK (e.g. it would be in a package).
- It reports a bug that is not widely reported and not catastrophic, and cannot be reproduced by anyone but the original reporter. Encourage the reporter to attempt to debug the issue themselves, potentially giving suggestions for places where they could instrument the code to find the issue, and invite them to join the Discord for help.
- It is tracking technical debt but the suggested improvements are marginal at best or would require significant research to be evaluated. Prefer having folks who work in the relevant part of the code make improvements based on their judgment.
- It is describing a *solution* rather than a *problem*. For example, it has no use cases, and the use cases are not obvious, or might have other solutions.

Reasons to keep an issue open:

- It's a well-described problem that we can reproduce reliably.
- It's a well-argued feature request with a solid use case and clear goal. (If it's something we're unlikely to ever do, it should be marked P3.)
- It is tracking technical debt that is clearly actionable.
- It is a request to add a customization to a material widget that fits cleanly into the existing material design library's ethos.

Deployment Plan

1. Using the following table:

Team code	Team name
-----------	-----------

android	Android platform team
codelabs	Codelabs team
design	Design Languages team
desktop	Desktop platforms team
ecosystem	Ecosystem team
engine	Engine team
framework	Framework team
google-testing	Google Testing team
gorouter	Go Router team
infra	Infrastructure team
ios	iOS platform team
news	News Toolkit team
release	Release team
tool	Flutter Tool team
web	Web platform team

...create labels with the following forms: ✓

Label name	Description	Color
team-\$code	Owned by \$name	
triaged-\$code	Triaged by \$name	
fyi-\$code	For the attention of \$name	

2. Run the script to migrate all issues to the new system. ✓
3. Do the adjustment that the script suggests for team-go_router. ✓
4. Remove "passed first triage" and "passed secondary triage" labels.
5. Update the triage instructions on the wiki, merging in the new section into the old, deleting old processes.
6. Address "team" and "easy fix" issues per section above.
7. Once any issues are fixed, deploy the bot on Firebase instead of running locally.

Automations

This is a long-lived Dart process running on a Linux box.

Question: where?

Question: how does it get TLS certificates?

Pseudo-code for listener-based bot

Data model:

- A set of all open issues, by issue number: ✓
 - last time it was touched by a contributor. ✓
 - last time it was touched by the assignee. ✓
 - current labels. ✓
 - last time it was locked (if it's locked). ✓
 - last time the assignee changed. ✓
- A list of pending issues to clean up. ✓
- The number of the current sanity-check issue. ✓

Continually, but at a slow rate, go through all the issues and update the data model. ✓ This handles situations where we miss notifications for whatever reason.

The premise here is that we can rely on a long-lived process that responds to GitHub webhooks, and supplement its logic with requests on the REST API. **Bold bits below are writes to GitHub.**

- Any time an issue changes labels, is opened, is closed, is locked or unlocked, or has assignee changes, update the data model. Forget issues when they are closed. ✓
- Any time an issue is touched by a team member, record the event time as the last team contact with the issue. ✓
- Any time an issue is updated by the person who is assigned to the issue, remember that time as the last update time for that issue. ✓
- When an issue has a "team-team" or "triaged-team" or "fyi-team" label added or removed, or any of the priority labels, or the "design doc" label, or an assignee change, or it is reopened, it gets added to the list of pending issues to clean up. (We can be pretty liberal about adding issues to this list in practice.) ✓
- If the sanity-check issue is closed, reset the current sanity-check issue. ✓

An issue that's in the list of pending issues to clean up which has not been touched for 45 minutes gets checked as follows:

- if it has more than one priority label, all but the highest one is **removed**. ✓
- if it has more than one "team-foo" label, they are **all removed**. ✓
- if it has any pairs of "triaged-foo" and "fyi-foo" labels, they are **both removed**. ✓
- if it has any "triaged-foo" labels without the corresponding "team-foo" label, the extra ones **get removed**. ✓
- If it has no priority (-label:P0,P1,P2,P3) and has a "triaged-team" label (label:triaged-a,triaged-b,...), **remove the triaged-team label**. ✓
- Issues that were locked then reopened should be **unlocked**. ✓

Periodically, enforce these invariants on issues that are not the sanity-check issue:

- Given an issue that is assigned but has not been touched by a team member for 5 months; and is not assigned to the person who filed it, or that person is not a team

member, or the issue has the "design doc" label; and has a "triaged-team" label: **remove the triaged label** if the number of issues with the corresponding "team-team" label but no "triaged-team" label is less than some threshold.

- Issues that have been locked for a few weeks should be **unlocked**. ✓
- Issues that have doubled in issue count since the last time they were triaged have the **triage label removed**.

Periodically:

- If there is no current sanity-check issue, **file an issue** that has all the team labels on it, with instructions to just mark it as triaged. ✓
- If the sanity-check issue is more than two weeks old, **mark it with the "needs additional triage" label**. ✓
- Look at the top-20 issues by thumbs-up, if any have not been updated by the team recently, then **remove the triaged-team labels**.

Pseudo-code for polling-based bot

Let *last update* be a stored time that can be updated (maybe a file in the script's repo):

Let *audit issue* be an issue number.

Continually, but pausing whenever GitHub rate limits are reached:

- update the local cache of issue metadata for flutter/flutter
- update the local cache of labels for flutter/flutter
- update the local cache of members of flutter-hackers
- for each pair of team * and fyi * labels, count of how many open issues have one of those labels but don't also have the triaged * label.
- for each issue:
 - if the issue is *audit issue*, skip it.
 - if the issue was last updated before *last update*, skip it.
 - if the issue was last updated within the last ten minutes, skip it.
 - if the issue has more than one priority label, remove all but the most important.
 - if it has any matching pairs of fyi * and triaged * labels, remove those pairs.
 - if it has more than one team * label, remove all of them and all triaged * labels, and add a comment saying that multiple team labels were found, why that's not ok, what should happen next, and pointing to the relevant policy.
 - if the issue has no priority label, but has a team label and a triaged label for that team, and has no fyi label, remove the triaged label unless the affected team has more than 100 issues in their backlog already, and add a comment saying that the issue is missing a priority and is being sent to triage as a result, and point to the relevant policy.
 - if the issue is marked P1 and has a triaged label:
 - get the time the P1 and triaged labels were added; if the more recent of those two is more than 5 months ago:
 - get all comments on the issue in the last 3 months
 - if none of those comments were by someone who has commit access, remove the triaged label and add a comment saying that P1s should be assigned and get monthly updates.

- if the issue is assigned and has a triaged label, and either the assignee does not match the reporter or the assignee does not have commit access:
 - get the time the assignee was assigned; if that was more than 5 months ago:
 - get all comments on the issue in the last 3 months
 - if none of those comments were from the assignee, remove the triaged label and add a comment saying that assigned issues should get monthly updates.
 - update *last update* to be the time of the issue's last update (before any changes triggered herein).
- If *audit issue* is a closed issue that was closed more than 3 months ago:
 - file a new issue, and let *audit issue* be that issue.
 - the new issue should have instructions to each team saying that all they should do is mark the bug as triaged.
 - the new issue should be given all the team labels.
- If *audit issue* was filed more than one week ago, add the "will need additional triage" label if it doesn't already have it.

Necessary APIs

- GitHub's REST API for fetching issues, reactions, and comments.
- GitHub's REST API for fetching a repo's labels.
- GitHub's REST API for fetching an organization's teams and team members.
- GitHub's REST API for removing labels
- GitHub's REST API for adding comments
- GitHub's REST API for the timeline (not currently available in package:github)

old notes

- Store data to disk then save that to git repo (upstreaming to github):
 - Should be able to use `git` directly to commit then push to the repo:
 - Maybe use a branch or other repo to store the data?
- Read data from disk:
 - Should be able to use `git` directly to do the opposite of the storing.
- Get count of issues from a GitHub search query for open issues with a label.
- Get list of issues from a GitHub search query for open issues with a specified sort order (ascending update time), with updates between two specified timestamps.
- Get a list of labels on an issue.
- Get list of labels matching a pattern.
- Remove specified labels from an issue.
- Add specified labels to an issue.
- Add a comment to an issue.
- Get a list of comments on an issue, specifically giving timestamps and authors.
- Get the time labels were added to an issue.
- Get the time that the assignee changed on an issue.
- Get the open/close state of an issue.
- Get the time an issue was closed.
- File an issue, setting labels and providing a summary and description.

Discussion

This section contains other ideas that were considered (and rejected) and discussions thereof.

Team labels

We could have the team labels be a set, meaning any issue can have one or more.

We could have two, horizontal vs vertical, mutually exclusive sets, e.g. framework/engine/tool/... and android/ios/web/....

We could have each team have a different label to indicate if they have triaged an issue.

Some issues won't fit a neat ownership model. For those, we should assign them to a specific person and consider bugs that are assigned to be exempt from the team ownership model.

Changing our priority label spectrum

Renaming labels

Historically, GitHub label renames have not been very reliable. However, more recently we have had luck having GitHub resolve the discrepancies.

Automated closures

VSCode

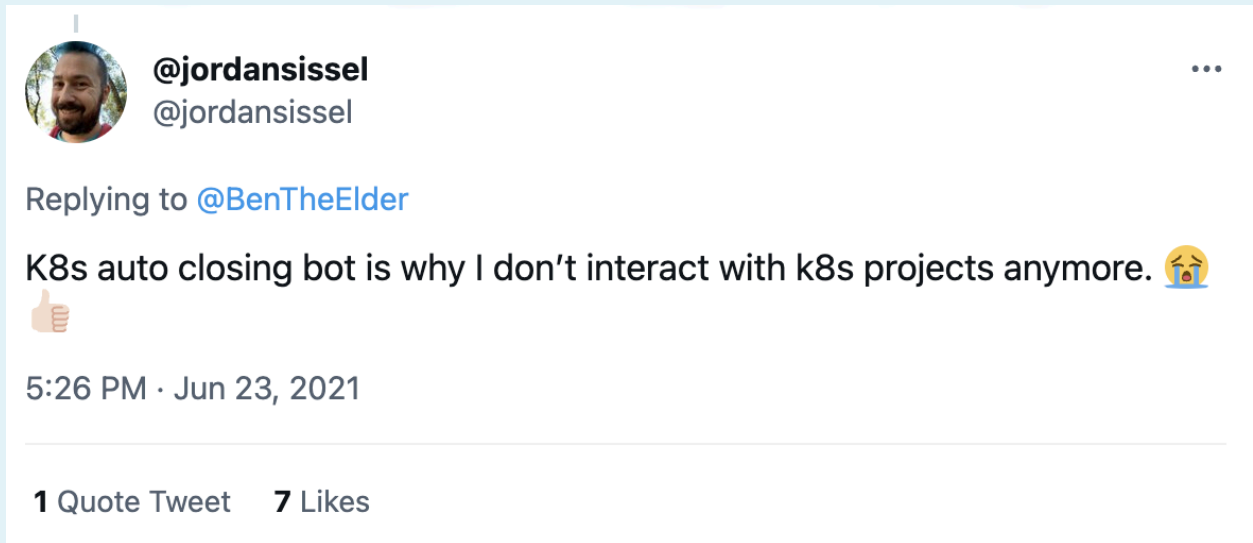
The microsoft/vscode repository [uses some automations](#) to automatically close issues. Specifically, new feature requests that don't get 10 votes within 60 days are closed.

In Flutter's lifetime, only 44 issues marked "new feature" ever got to 20 votes in 60 days or fewer, out of 6041 issues marked "new feature". Applying such a policy would close 2774 issues today, and would have closed more 2098 issues in the past that we nonetheless eventually closed of our own accord³, including 91 issues that eventually collected more than 100 votes each (including [wireless debugging of iOS devices](#), our second most-popular issue ever, second only to the [code push issue](#), which ironically would just barely not have been closed by this policy as it took 59 days to get to 20 votes despite being probably unfixable for non-technical reasons).

Kubernetes

Kubernetes is another project that has an auto-closing policy. Or rather, had an auto-closing policy; they [recently rescinded it](#) based on popular demand. It's interesting to look at the feedback around that discussion, for example, [this tweet](#) is one of the biggest concerns some have with such a policy:

³ The remainder are issues we closed in less than 60 days.



The new policy still uses automation, but instead of auto-closing stale issues, it sends them back to be retriaged.

Other discussions

- [Github Stale Bots: A False Economy](#) about Angular's practice of closing issues as stale.
- [GitHub stale bot considered harmful](#) (see also [Hacker News discussion](#)).
- [NutJS reversing their stance on auto-closing issues](#).
- [Gutenberg project proposal to autoclose issues](#) (and discussion); [actual decision](#).
- [NixOS project discussion about closing stale issues](#).
- [Gradle's explanation of why they auto-close stale issues](#).
- [Hacker News comment about a Mozilla bug that took 21 years to fix](#).