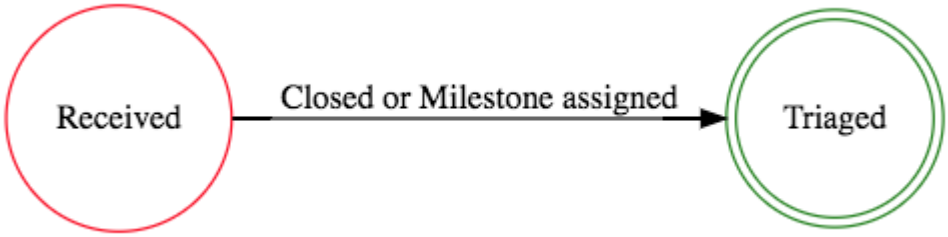
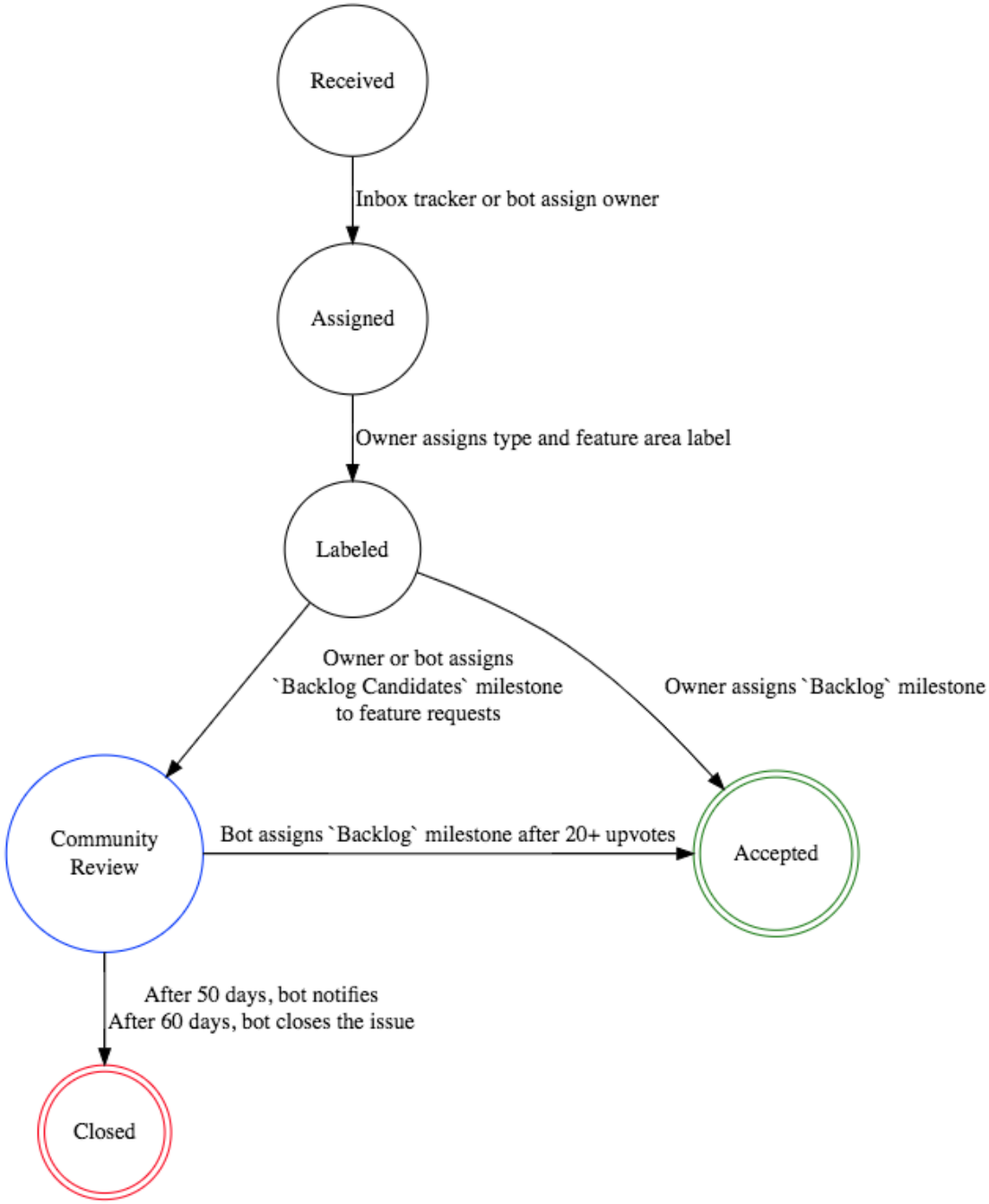
Triaging an issue is a multi-step process that is collaboratively performed by the inbox tracker, the feature area owners, and our issue bot. Triaging an issue usually takes around one business day but may take longer, for example, when the feature area owner is not around. Goal of triaging is to provide you with a clear understanding of what will happen to your issue. For example, after your feature request was triaged you know whether we plan to tackle the issue or whether we'll wait to hear what the broader community thinks about this request.

From your perspective it's straightforward to understand whether or not your issue is triaged:



Our Triaging Flow

Below is the basic flow your issue goes through. At any step in the flow we may also close the issue or request more information. We omitted that from the flow chart to keep it as clean as possible.



The chart uses three major states. They are easily identifiable:

State	What your GitHub issue looks like
Closed	matches the query `is:closed`
Community Review	matches the query `milestone:"Backlog Candidates"`
Accepted	has any milestone except `Backlog Candidates`

In the rest of this document, we'll go into more detail about each of the activities of triaging and how we make decisions.

Closing Issues

We close issues for the following reasons:

Reason	Label
The issue is obsolete or already fixed.	
We didn't get the information we need within 7 days.	`needs more info`
It's a duplicate of another issue.	`*duplicate`
The feature request is best implemented by an extension.	`*extension-candidate`
What is described is the designed behavior.	`*as-designed`
The issue was caused by an extension.	`*caused-by-extension`
The issue is a developer question.	`*dev-question`
The issue is a user question.	`*question`
The issue is not related to the goals of our project and/or therefore unactionable	`*off-topic`
The issue does not contain any valid or useful information or was unintended.	`invalid`
Given the information we have we can't reproduce the issue.	`*not-reproducible`
The feature request is out of scope. (See below)	`*out-of-scope`
The issue is tracked in a repository we rely on.	`upstream-issue-linked`, `upstream`

We close issues with the help of a bot that responds to a particular comment such as `/duplicate of #1234` or to assigning a label with adding a pre-canned comment to the issue and closing the issue.

Requesting Information

If an issue misses information that we need to understand the issue, we assign the `needs more info` label. We usually add the `/needsMoreInfo` comment to the issue which lets the bot add the label as well as a comment with links to documentation. If the issue is a performance issue we ask you to follow this documentation.

The bot is monitoring all issues labeled `needs more info`. If we don't receive the needed information within 7 days the bot closes the issue.

Categorizing Issues

Each issue must have a **type** label. Most type labels are grey, some are yellow. Bugs are grey with a touch of red.

Type	Description

needs more info	not possible to assign a type label due to missing information; see above
bug	the implementation of a feature is not correct
feature-request	request for a new feature
under-discussion	not decided whether the issue is a bug or feature
debt	improve the implementation/architecture
*question	we should direct questions to StackOverflow
upstream, upstream-issue-linked	an issue used to track an issue in an upstream component
electron	an issue with electron
engineering	issues related to our engineering system or our processes
polish	a feature could be improved, but not necessarily a bug

We also use the following **type** labels, although they don't directly play a role in the triaging process as they are usually assigned to issues created by our team or by tooling.

Type	Description
iteration-plan	a plan tracking the work for an iteration
iteration-plan-draft	a draft of an iteration plan
plan-item	an issue to organize and structure planned work
testplan-item	an issue describing how to test a feature
endgame-plan	a plan tracking the endgame of an iteration
perf-profile	a issue representing a performance profile

Assigning Feature Areas

Each issue must have a **feature area** label. Feature area labels are dark blue. See the list of feature area labels here.

Assigning a Milestone

In addition to milestones representing our iterations and releases such as `November 2019`, we have three milestones with special meaning:

- Issues assigned to `Backlog`: Our team is in favor of implementing the feature request/fix the issue. The issue is not yet assigned to a concrete iteration. If and when a Backlog item is scheduled for a concrete iteration depends on how well the issue aligns with our **Roadmap**. We review and update our roadmap at least once every 12 months. The Backlog helps us shaping our Roadmap but it is not the only source of input. Therefore, some Backlog items will be closed as **out-of-scope** once it becomes clear that they do not align with our Roadmap.
- Issues assigned to `On Deck`: Our team wants to implement the feature/fix the issue. The issue is on the short list to be assigned to a concrete iteration. Note: `On Deck` is used sparsely. More commonly, issues go from `Backlog` directly to concrete iterations.

- Issues assigned to `Backlog Candidates` : Our team does not intend to implement the feature request/fix the issue but wants the community to weigh in before we make our final decision. See also Managing Feature Requests.

Important Issues

- We assign the `important` label to issues that
  - result in data loss
  - a breakage of extension
  - critical security, performance issues
  - UI issue that makes a feature unusable

Asking for Help

We label ** `Backlog` **issues, particularly feature requests, that we encourage the community to take up with `help-wanted` . If issues are suitable for beginners we may add the `good-first-issue` label and we add code pointers that help beginners to get started with a PR.

Sometime, we get issues that we can't or don't have the time to reproduce due to the complexity or time requirements of the setup but that we indeed suspect to be issues. We label those issues with `investigation-wanted` . What we are looking for is help in reproducing and analyzing the issue. In the best of all worlds we receive a PR from you. :smiley:

Please note, we only accept PRs for issues that are accepted, i.e. have a milestone assigned that is not `Backlog Candidates` .

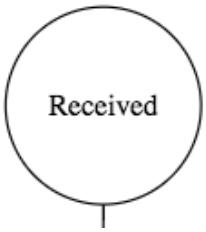
Type-specific Characteristics / Triaging

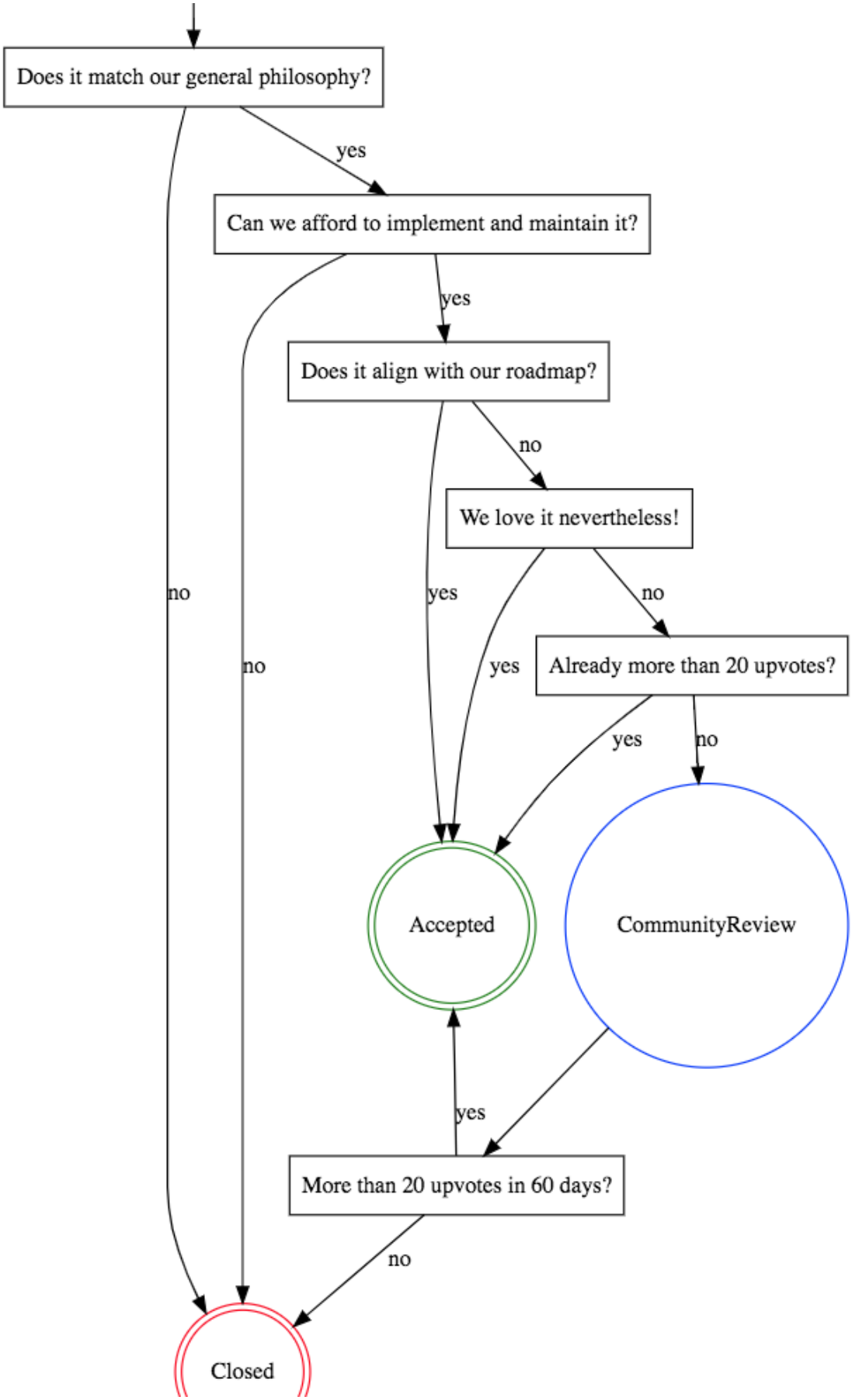
Managing Feature Requests

Feature requests like all issues are a means of communication between us and our community as well as among the members of the community. Thus, in principle, we could keep all feature requests open no matter what will happen to the feature they describe. Unfortunately, this makes it hard for you to understand what has realistic chances to ever make it into the repository. We therefore close feature requests we cannot address with `out-of-scope` while we assign feature requests we don't plan to address but want to give you time to weigh in to the `Backlog Candidates` milestone.

If you are the author of a feature request you might not like that we close or don't plan to address your request. It might even feel like a slap in your face. We understand that. All of us have been there—in this project or others we have contributed to. So, be assured, we love all of your input. Don't take personal offense when we close or assign the `Backlog Candidates` milestone to your issue :peace_symbol:. If you feel your feature request deserves to stay open, improve your use case and ping us or gather more up-votes.

This is our decision making tree. More details below.




Does it match our general philosophy?
yes
Can we afford to implement and maintain it?
yes
Does it align with our roadmap?
no
We love it nevertheless!
yes
no
Already more than 20 upvotes?
yes
no
yes
no
yes
no
Accepted
CommunityReview
yes
More than 20 upvotes in 60 days?
no
Closed

To put the diagram in words:

1. Does the proposal match with our general product direction? For example, VS Code is a light-weight extensible text editor and as such we are not interested in turning it into a platform to implement a web browser.

If the answer is `no` we close the issue as `out-of-scope`.

2. Can our team afford to implement the feature? I.e. are the direct and the opportunity costs to implement the functionality and maintain it going forward reasonable compared to the size of our team?

If the answer is `no` we close the issue as `out-of-scope`.

3. Does the functionality described in the feature request have any reasonable chance to be implemented in the next 24 months? 24 months is longer than our roadmap which outlines the next 6-12 months. Thus, there is some crystal ball reading on our part, and we'll most likely keep more feature requests open than what we can accomplish in 24 months.

If the answer is `yes` we assign the issue to the **`Backlog`** milestone.

4. Do we think the feature request is bold and forward looking and would we like to see it be tackled at some point even if it's further out than 24 months? (Clearly, this is quite subjective.)

If the answer is `yes` we assign the issue to the **`Backlog`** milestone.

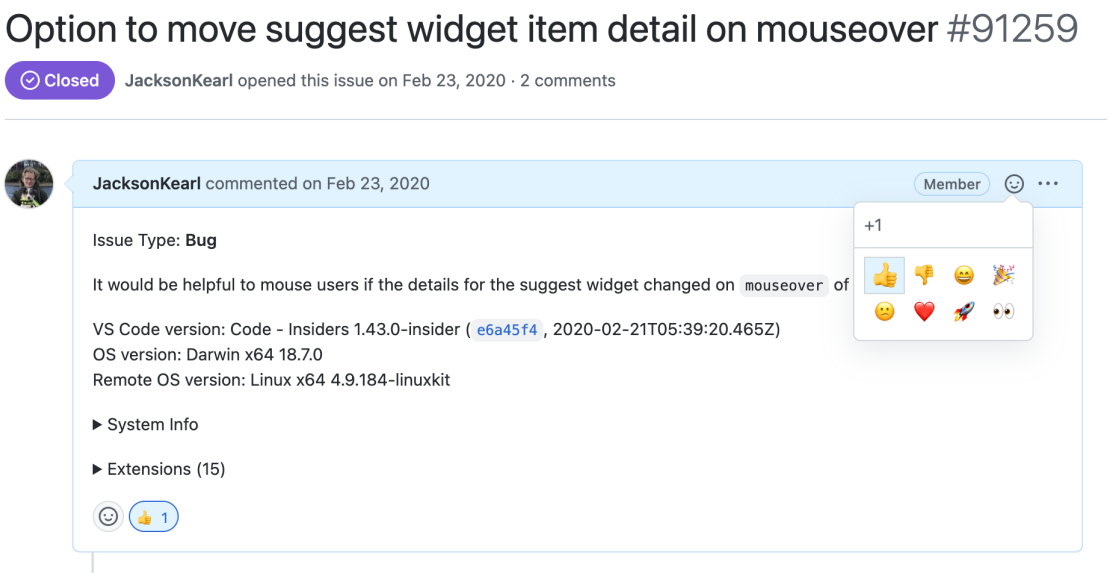
5. Has the community at large expressed interest in this functionality? I.e. has it gathered more than 20 up-votes.

If the answer is `yes` we assign the issue to the **`Backlog`** otherwise the **`Backlog Candidates`** milestone.

A bot monitors the issues assigned to **`Backlog Candidates`**. If a feature request surpasses the 20 up-votes, the bot removes the **`Backlog Candidates`** milestone and adds the **`Backlog`** milestone. If an issue is assigned to the **`Backlog Candidates`** milestone for more than 60 days, the bot will close the issue.

Up-voting a Feature Request

When we refer to "up-voting" a feature request, we specifically mean adding a GitHub `+1` /"👍" reaction to the issue description. In the GitHub UI this looks like so:



Won't fix Bugs

We close bugs as `wont-fix` if there is a negative cost-benefit balance. It's not that we don't care about users who are affected by an issue but, for example, if the fix is so complex that despite all of our tests we risk regressions for many users, fixing is not a reasonable choice. When we close a bug as `wont-fix` we'll make our case why we do so.

Upstream Issue

We label some issues as `upstream`. Upstream means that the issue is in a package or library that we consume and that we cannot fix independently. We close an `upstream` issue if we can establish a clear traceability link between the issue in our repository and an issue in the issue tracker of the package or library. In some cases this is not possible, for example, we might have identified an issue as a Chromium issue but Chromium does not accept the issue yet because the repro case is too complex.