Extracted from:

Program Management for Open Source Projects

How to Guide Your Community-Driven, Open Source Project

The Pragmatic Bookshelf
Raleigh, North Carolina

The Pragmatic Programmers

Program Management for Open Source Projects

How to Guide Your Community-Driven, Open Source Project

Ben Cotton

Edited by Michael Swaine

Program Management for Open Source Projects

How to Guide Your Community-Driven, Open Source Project

Ben Cotton

The Pragmatic Bookshelf

Raleigh, North Carolina

The team that produced this book includes:

CEO: Dave Rankin
COO: Janet Furlow
Managing Editor: Tammy Coron
Development Editor: Michael Swaine
Copy Editor: Corina Lebegioara
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics
Founders: Andy Hunt and Dave Thomas

ISBN-13: 978-1-68050-924-3
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—July 2022

Triage Bugs

Problem: Nobody is sure which bugs are the important ones. And the problem gets complicated by the presence of duplicate and invalid reports.

You have to do something with a bug once it is opened. To do something with it, you have to make some decisions. How important is it? How easy is the fix? Who has the time and skills to work on it?

In medicine, *triage* is the practice of grouping patients based on the extent of their injuries and their likely mortality. People with minor injuries can wait. People who are almost certain to die aren't treated so that time and resources are focused on those who are critically wounded but could survive. Similarly (except far less emotionally and ethically fraught), bugs that are of trivial impact (for example, a tyop (sic) in a book that doesn't affect the meaning) can wait. A bug that is impossible to fix without shutting everything else down for six months while you rewrite the entire project in a new language won't get much effort. So bug triage involves assessing the impact field described in the previous section in order to set the priority. But there's more to it than that.

Start the process by assessing some basic information: is this actually a bug? Is it a duplicate of an existing report? From there, you can start collecting the information that will allow you to make decisions on priority.

Answer Questions

Problem: The bug tracker is full of invalid reports and duplicates. You're losing security bugs in the pile.

When triaging bugs, you're asking yourself a few questions. How you handle the bug report depends on the answers.

- *Is this a bug?* Not all issues in your bug tracker are bug reports. Users or other contributors will often file feature requests in the bug tracker as well. This is acceptable—even desirable—in most cases, but you'll still want to make sure you set the category appropriately. While you may accept bug fixes in passing, you may want to limit new features submissions to "trusted" developers or at least require additional planning work to discourage hasty contributions. If it's neither a bug nor a feature request, it might be a question or a support request. Mark it as such or—if your project uses a separate tool—direct the reporter to the right venue. Keep in mind that questions and support requests often highlight documentation bugs, so open a bug against your docs if that seems appropriate.

Lastly, the report may be an unactionable rant or unrelated spam. You can close those without feeling bad.

- *Is it a duplicate?* If you're getting a feeling of *déjà vu*, the bug report may be a duplicate. Ideally, the reporter checked for existing issues that match theirs before submitting a new report. People don't always do that. But even if they did, they may have just missed it. If you spend a lot of time in the bug tracker, you can recognize duplicates when they come in and mark them as such. You'll probably miss some (see the sidebar at the end of this list) and that's okay. The more duplicates you catch, the more developer time you'll save.

- *Is it a security bug?* One of the foundational tenets of the free software movement is that users should control their computing. Security flaws violate that by giving someone else control of the computer in some way. These things happen, but you'll want to make sure you clearly mark them so that someone can fix them quickly. You want security bugs to stand out to developers.

- *Is it filed in the right place?* Users don't always know where the problem lies; they know how they experience it. The larger and more complex your project is, the more likely it is that the bug report will be misfiled initially. A Fedora kernel maintainer once told me that the kernel component seems to be the starting point for any bugs that happen during the boot process because there are so many moving parts and it's hard for most people to tell where the actual failure happens. The bug might not even be in your software to begin with—it could be in an upstream project. In that case, you might close your report and direct the user upstream or act as a bug concierge and file the report for them. Regardless, part of bug triage is to attempt to get the report into the right place using your deeper knowledge of the project. Of course, the bug report may still get passed around a few times before it gets fixed.

- *Is it reproducible?* You don't necessarily need to reproduce every bug yourself, but you should make sure that the report contains the information needed to reproduce it. If it doesn't, ask the reporter to provide what's missing. After a few weeks pass with no reply, you can choose to close the bug.

How Many Duplicates Do You Have?

Bear in mind that the number of bug reports marked as duplicate isn't the same concept as the number of duplicate bug reports. In theory, a project with a small number of bugs will have relatively few duplicates because it's very easy to check before filing a new report. As the number of bugs increases, you may expect the number of duplicates to increase proportionally, especially as you get into very large numbers because it becomes more difficult to find duplicates.

When I looked at the duplicate percentages for Fedora Linux bug reports,[a] I saw a different picture. Once a component had enough bugs to rise out of the "noise," the percentage of duplicate bugs held fairly steady, as expected. But the components with the most bugs saw a *decrease* in the duplicate percentage.

The challenge users face when trying to find existing bug reports also applies to project contributors. When a project has several thousand bug reports, it's unlikely that anyone can look at a new report and say "ah yes, this is probably a duplicate of bug 12345." So don't worry about missing some duplicates. But be wary of making decisions based on reports marked duplicate when you're actually thinking of duplicate bugs.

a. https://communityblog.fedoraproject.org/exploring-our-bugs-part-1-the-basics/

Ideally, you answer all of the questions in this section when the bug is filed before any real debugging work begins. But partial triage is better than no triage, and the answers may change as the report is investigated further.

Create a Triage Process

Problem: There's no coordinated process for triaging bugs. Some get triaged right away. Some don't get triaged at all. Some get triaged twice.

Once the questions in the previous section are answered, the bug report is appropriately marked as triaged. Then you send it off to the developers to get fixed. But how did you answer those questions in the first place?

This is one of the most challenging aspects you'll face as a program manager in an open source project. It is labor-intensive work that doesn't appeal to most people, so it scales very poorly. Small projects probably don't have enough bugs to need a process. Large projects have too many bug reports to reliably triage each one. This section describes a process that can work for the middle-sized projects, but don't feel bad if you find you end up triaging bugs on an *ad hoc* basis.

First, you must assemble a triage team. Bug triage is difficult, often thankless work. Don't be surprised if volunteers are scarce. Not just any volunteers will do. Triagers need a moderate understanding of how the project works at a technical level. If the project has multiple components, they need to know what each one does and doesn't do so they can sort it into the right pile. But again, some triage is better than no triage. If you can get someone to do basic categorization and check that the report contains the required information, that's a big help. Over time, if they stick around, they'll learn how the pieces fit together.

Next, let's figure out how the team will interact. Although doing a live triage meeting is a great way to share knowledge, it's slow and not particularly accessible for a distributed volunteer team. Consider pairing newcomers with an experienced person for a brief period to learn the ropes, but after that, your triagers will work mostly independently. Since, unlike medical triage, they're not making life or death decisions, it's okay for them to be fast and wrong. But they should have a channel to communicate with each other to ask questions or get help on something that truly stumps them. Regular meetings to discuss patterns, ask questions, and ask for a second opinion provide the team with an opportunity to grow.

Of course, there has to be some way to get the new bug reports to the triage team, so let's look at a few options. The simplest way is to have new reports automatically assigned to the team. Triagers can pick bugs out of the team's queue and assign them to the right place (or leave them unassigned) once the bug report is ready. The downside is that if the triage team doesn't get to a bug report, it'll sit there and not get fixed. An alternative is to have a report that lists the new, untriaged bugs and have triagers work from that. This means if a developer starts working on a bug before the triage team gets to it, they do their own triage. That's fine, since the point is to help get the bugs fixed, not win a turf war. When someone triages the bug report, they apply a label (however your bug tracker implements this) to indicate that it's done.

Although the word triage comes from the French word for sorting, the "tri-age" spelling implies that it'll age you three times as fast. That's a reasonable interpretation. Do what you can to recognize and reward your triage team for the work they do. And expect to see a lot of turnover on the team.

Prioritize Bugs

Problem: No one is sure which bugs are the most important. Trivial bugs get fixed long before serious bugs.

The priority of a bug—in which order it gets fixed—is generally up to the person working on it. People are contributing on a volunteer basis; you can't dictate priorities to them. By and large, they'll work on the bugs that they think are important, and their own interests and abilities are factors in that assessment. But most developers understand that being in a community means taking the community's priorities into account. That means the community has some say in what bugs are most important. So let's develop a framework for evaluating a bug's priority from the community's perspective, not the individual's.

Ask Questions

Problem: Each contributor prioritizes differently. There's no consistency in priorities, which frustrates contributors and users alike.

As with triage, you ask a set of questions to set a bug's priority. But sometimes questions need technical investigation of the bug before they can be answered. And while basic triage mostly involves objective decisions about a bug report, you make subjective decisions for prioritization. Ask the following questions when you're evaluating a bug's priority:

- *Can physical harm result from this bug?* We often talk about the Internet like it's separate from "real life." This is wrong. The software your project produces can be physically harmful. For example, a poorly secured web application might leak the mailing address of a user. Software that controls hardware could cause the device to draw too much current and start a fire. Thankfully these cases are rare, but you must be aware.

- *Does this bug cause data loss or corruption?* Yes, backups are important. No, most people don't have a sufficiently robust backup scheme in place. A bug that causes data loss can be enough to scare users away forever. If it happens in a business environment, it could cost your user real money. If it happens at home, it could result in the loss of irreplaceable mementos—pictures from grandma's 90th birthday party or a video of a baby's first steps.

- *Does this bug allow unauthorized access?* Bugs that allow someone to access—or worse, modify—systems or data they're not supposed to are bad. Even if no physical harm results, it could lead to a loss of data. A system may be used to send a harassing, fraudulent, or otherwise impermissible message. It could incur financial costs, or cause harm to someone's reputation.

- *Is accessibility reduced by the bug?* Software is only useful to the degree that people can use it. If a bug breaks accessibility features, that makes it less useful. This might include screen reader functionality or the ability to resize text. If a GUI or website can't be used without a mouse, that's an accessibility bug, too.

- *Is the default configuration affected?* Most people won't deviate far from the default configuration. It represents your project's opinionated view on how the software should work. So if there's a bug in the default, that's more important than a bug in a custom configuration, all other things being equal. If nothing else, a bug in the default configuration is likely to affect a much larger portion of the user base.
- *Do you not have a reasonable work-around?* If there's a way to avoid the bug, or at least to mitigate the harm, then it might go further down the priority list. "Reasonable" is doing a lot of work here. The work-around has to be something that a typical user can do easily and without significant loss of functionality.
- *Does the bug affect all platforms?* In an ideal world, it doesn't matter which hardware and operating system are in use. A bug is a bug. But in the real world, not all platforms are the same. If the bug only occurs on Windows 3.1 running on an i286 processor, that's probably at the bottom of the pile. The important platforms vary from project to project. For Internet of Things projects, ARM processors are probably important, but mainframes are not.
- *Does the bug impact any key downstreams you have?* If users often or primarily get to your project's software by way of a downstream, this is a factor to consider. A bug that prevents a popular downstream project from working should move up the stack in most cases.
- *Is the bug embarrassing to your project's reputation?* Some bugs don't cause any particular harm, but they still make you look bad. Maybe you misspelled your project in the loading screen. Perhaps the insulting error message you put in as a placeholder didn't get removed before release. It could be related to functionality, too. If it takes 30 seconds for a WiFi connection to be available, that's functional but it's also annoying. They say that there's no such thing as bad publicity, but most of us would rather not have people saying bad things about us or our work.

One question remains that lends itself to a more objective decision (or at least a binary response). *Is this something the software must always (or never) do?* In other words, will you delay a release if you find a bug like this. We call this a "release blocker," which we'll cover in Set Release Criteria, on page ?. For now, let's assume that a bug report in your bug tracker is for a version that has already been released, so it's too late to be a blocker.

Rate the Priority

Problem: The priority of a bug isn't clearly and consistently communicated.

In the previous section, we asked a lot of questions, and all of them have answers that are more spectrum than binary. You're probably tempted right now to come up with a rating system and give each bug a score. Don't. Once you get beyond a few bugs per developer, it's less "a stack" and more "assorted piles." Remember that these decisions have a large amount of subjectivity, so attaching an exact value is lying to yourself. It's not even a particularly useful lie, as it'll cost a lot of your time to develop and refine the scoring system. Even then, a developer looking for a bug to fix might not fix the highest-scoring bug because they may not have the knowledge or skills to tackle that particular one.

So how do you put the answers to those questions into practice? Let's say a bug's priority falls on a low/medium/high scale. The questions are roughly in order of descending importance. The sooner you answer "yes" to a question, the higher the bug's priority. The more "yes" answers a bug gets, the higher its priority. So you might say that any bug that gets a "yes" to the first four questions previously discussed is a high priority, a "yes" to the remainder is medium, and all "no" answers is low. But if a medium priority bug has three or more "yes" answers, it's also a high-priority bug.

Another option is to have the priority be a binary state instead of a low/medium/high scale. With a binary state approach, a bug is either prioritized or not. A community member (whether user or contributor) nominates a bug and then a group evaluates it and makes a decision. The group can be fixed or variable (for example, "whoever shows up to the meeting this week"), but you should make sure to notify the assignee and the person who nominated it so they can weigh in on it. If you accept a bug as prioritized, that signals to the assignee that they should work on this one first. This can be particularly helpful if your project has a corporate sponsor or at least corporate contributors. You can use prioritized bugs to say "this is where your support is most valuable to the community at the moment."

You can also choose a mixed approach. You can have low/medium/high priority set by a developer or the triage team, but also have an extra tier of highest priority that requires an approval process. However you do it, you need to be aware of the proportion of high-priority bugs. If everything is the top priority, nothing is.

And of course, document the prioritization guidelines in your project's contributor documentation so that everyone can find them. If you have a triage team, and they're relatively skilled, they can set the priority at triage time. (Of course, with the understanding that it may be adjusted by whichever developer ends up working on it.) If you don't have a triage team, each developer can apply the agreed-upon prioritization guidelines as they work on a bug report.