The Pragmatic Programmers

Debugging TypeScript Applications

Build Web Apps That Don't Break

BETA

Andrey Ozornin

edited by Kelly Talbot

This extract shows the online version of this title, and may contain features (such as hyperlinks and colors) that are not available in the print version.

For more information, or to purchase a paperback or ebook copy, please visit https://www.pragprog.com.

Prioritizing Bugs Using RICE

Priority is a function of reach, severity, and time. You want to tackle first the worst of the issues concerning the most people that is the fastest to solve, and now, when you're able to evaluate all three parameters, you want to combine them into a single metric, but how do you do it? Do you take first a small issue concerning a lot of people, or a severe one concerning only one? What if one of the issues is easier to resolve? The three metrics are measured in different units, and combining them is not straightforward.

Let's assume that you have two known bugs in an application with 1000 daily users.

1. Two users reported that they cannot register in your application because their full names are 52 characters long, and you have a validation limit set to 50.
2. All users of your application experience slow initial loading. They see a loading indicator for around five seconds.

First, kudos for having a thousand users and only two known bugs. Awesome result! Usually it's the other way around. You need to choose one to take first. This might seem like a trivial exercise, but when you have a hundred bugs for a team of five, and you need to balance it with working on new features, you better have a good methodology at hand.

One common way of prioritizing bugs is using the RICE score. It stands for Reach, Impact, Confidence, Effort, and lets you calculate a single numeric value for every task to sort your bug backlog by.

$$\frac{Reach \times Impact \times Confidence}{Effort}$$

Be aware that such formulas are canonical examples of what the Nobel laureate Paul Romer called "mathiness".[3] Carl T. Bergstrom and Jevin D. West gave many good examples of this in their book *Calling Bullshit*. These expressions "create the impression of rigor and accuracy," which we expect of anything that looks mathematical, but on a closer look we find out that it involves a good share of guesswork, operations on barely related units, and an arbitrarily chosen multiplicator, so don't try treating the RICE score the same way as measurements.

Nevertheless, use them without a doubt in situations where reaching an agreement and moving on is preferable to finding the best possible solution

3. https://paulromer.net/mathiness/

at the cost of time spent on discussions. These situations are very common in product development. All models are wrong, but some are useful. If you get a score of 100 on one item and 5 on another, you can confidently prioritize the former over the latter.

Reach

Estimate the absolute number of users who encounter this bug. If the bug is reproduced only in Safari, you can prorate this number by the share of Safari users (around 15-20% overall as I'm writing this, and can vary depending on your audience profile). If the bug is in a part of the site that you know only a quarter of users bother visiting, divide it by 4. If it's only for users using a particular locale, divide accordingly.

Keep in mind that often people do not report bugs. Use your best judgment to approximate how many people can be possibly affected. How many users of your app do you think have full names over 50 characters long?

For our case, we assume it's 2 people for bug #1 and 1,000 for bug #2.

Impact

You can rate severity, or how bad the bug is, on a scale from 0.25 to 4.

In our example, not being able to log in is just about as bad as it can be. Only a serious data loss would be worse. It would be fair to rate it at 2 or even 4. By contrast, waiting for five seconds to load is noticeable but causes at worst a minor disturbance. 0.25 or 0.5 would be a good impact estimate, depending on your performance requirements.

Confidence

Confidence is the rating of how sure you are that the approximations of effort and reach are correct. Don't calculate too hard, and choose between 100 if you're fully confident, 80 if you are aware of some unknowns, or 50 if there are significant unknowns. If you believe 50 is too much, you shouldn't be picking this until you resolve some of the most significant unknowns.

Effort

RICE can be seen as a cost-benefit analysis, where reach, impact, and confidence combined measure the benefits of resolving the bug, and effort is the single score representing the costs, or how much developer time you think it will take to fix the problem. The original RICE model developed in Intercom used developer-month as a unit of effort with anything less than a month

rated as 0.5, but if you find yourself in a faster-paced project and lucky to have some granularity of tasks, using month as a unit does not make much sense, and trying to estimate a project large enough to fill several months will likely fail without proper decomposition anyway. In practice, it is easier to not introduce new entities into the process and use whatever measure you already have in use, be it weeks, days, hours, or story points.

If the supposed fix is a matter of changing a constant in the validation schema, and you have an idea where this constant is, you can get the minimal possible estimate: five minutes in a day, depending on the company processes. If it's a matter of changing database schema in order to support longer values, that's the whole other story. In our example, we assume that it's a constant.

Prioritizing Beyond Methodologies

Whatever you do, don't be a dogmatic follower. Be smart, use reason. You'll inevitably be breaking some of the time limits when finishing the task will be a right thing to do; you will end up using some combination of estimation, no estimation and timeboxing, depending on circumstances.

Business needs will impact your priorities too, and you'll have to evaluate that on a case-by-case basis. Imagine that the company logo is missing in the application. Users might not care at all, if they can use the app and solve their needs, but the business you own or work for needs branding to be visible, you'll have to take that into account.

Techniques and methodologies are tools, and you are free to use or not use them to get things done.