Coko

How the 'itch-to-scratch model' can solve our UX woes

Every good work of software starts by scratching a developer's personal itch. Originally published on Opensource.com

Published: Apr 20, 2017
Author(s): Adam Hyde
Illustrations: Henrik van Leeuwen
PDF Design: Harshna Haswani
Rendered from HTML using Paged.js

Open source is a developer-centric solutions model, which, in a nutshell, could be described as building communities of developers to solve problems.

In its most simplistic form, the model has two stages. First, a developer has a problem, which they can fix with some new code, and they make a start on it. Second, if they then make their solution available to other developers it can develop into a full blown thriving open source community. When it works it is a fantastic process to behold and it this simple model that has changed the history of computing.

But outside of developers solving developer problems, open source hasn't done so well. This is pretty well discussed in open source circles, often framed in questions like: "Why are the user interfaces so sucky?" Framing the issue in terms of bad UI is common but it is perhaps a shallow treatment of the issue. 'User facing' open source projects don't just fail at UI, they fail to meet user needs at a much deeper level.

When looking around at the available open source solutions that are primarily aimed at users, have you ever asked yourself why open source seems to perform so poorly?It's not just the UX of your favorite desktop app that feels clunky, but why isn't open source winning in the desktop space in general?

Where is that open source desktop app everyone wants? What about web space for that matter?

Where are the open source equivalents that kill the proprietary opposition? Why isn’t open source wiping out the competition in every user sector from text editors to CRM, from webmail to social media?

There are few examples.

I would argue Unity desktop outstrips the competition, and that Mattermost and GitLab are better than Slack and GitHub, but I'm not in the majority on that.

Outside of a small handful of examples, I can’t see any open source products killing the proprietary competition when it comes to fulfilling user needs. Why has open source done so well in producing developer-facing solutions and web and internet infrastructure, but has done so poorly in the user-facing world?

I think open source can be dominant in social networks, text editors, CRMs, webmail, and more. In fact, I think the open source model has some essential characteristics that lend itself to beating the proprietary competition right across the board. But not without first understanding why it is currently not doing this.

To understand the problem, as a first step towards solving it, we must look deep inside the way we create solutions in open source communities. We need to do this critically and be prepared to ask ourselves some difficult questions.

The itch-to-scratch model "Every good work of software starts by scratching a developer's personal itch." -- Eric Raymond Eric Raymond suggested that open source and free software has a particular kind of solutions model he called the itch-to-scratch model. Basically, the idea is that open source projects start because someone, somewhere, sees a problem (the itch) and they start programming their way to a solution (the scratch).

That makes perfect sense and it is a very succinct way to describe what actually happens. Eric Raymond wrote more about it in The Cathedral and the Bazaar, which is considered a foundational work when it comes to understanding open source. As Eric Raymond states:

> "Every good work of software starts by scratching a developer's personal itch." -- Eric Raymond

The itch-to-scratch model has two very important characteristics which have played extremely well for open source:

1. The fact that it is your itch is essential for developing a good solution. If you have a problem and you know it very well then you are very well paced to start solving that problem. You are much better placed as an 'insider' to solve the problem than an 'outsider' that has a shallow grasp of the issue. Because it is your itch you are both motivated to solve it and, more importantly, you will have unique insights into how to solve that problem.
2. Others with a similar problem are essential for building community. If your itch is a common itch, and you know others that need your partial or complete solution, then you are likely to see a growth in contributions and/or adoption. That's basically the secret sauce to the growth of open source projects. There is more to successful open source communities of course, like governance, volunteer management, understanding intrinsic and extrinsic motivations, and so on, but the heart of the matter is that if you know others with the same itch, open source is, if nothing else, an excellent model for magnifying the scratch effect. It is also not without precedent, as anyone that reads the works of John Abele or Everett Rogers knows. In the outside world, this process is known as 'diffusion'.

So, the itch-to-scratch model, as a model for open source communities, really succeeds because you know your itch, and, secondly, you know others that share the same problem and, consequently, want to share, or participate in creating your solution. So far it sounds thoroughly unhygienic!

In real life

In the real world, we see this in action. Perhaps the cultural genesis of this is Linux. Linus Torvalds has a problem, he wants a free operating system (the itch). He makes a go of it (the scratch) and posts an email to the world asking if anyone else is interested in solving this problem with him. And, we know the rest of the story: one big community-driven scratch that has changed the history of computing forever. This is the itch-to-scratch model in action in a prototypical manner.

That is pretty much how every successful open source project from day one has evolved. Developers solving technical problems and gradually attracting other developers to work with them to solve or adopt it. At the heart of this model are the developers. The people with both the problem and the solution. I call this model the developer-centric solutions model. It is, pretty much, the model for every open source project that exists.

So, we know that this model is powerful. We have seen what it can do. But how many times have we stopped to ask ourselves what it can't do or what it doesn't do well? How often have we asked ourselves where it is failing and why? I don't think we have this conversation enough.

Let's start to consider these questions by first being clear about where the developer-centric model works. Well, it works well when solving problems developers have, which are almost by definition exclusively or at least primarily, technical problems. This is why the internet is basically run on top of an open source layer from BIND to OpenSSH, and why you can't get far in the web hosting or container world without open source. In fact, you can't really develop software of any kind without using open source libraries of one sort or another. The developer-centric model wins, hands down when solving these kinds of problems.

However, while the developer-centric model has proven to be amazingly good at solving technical problems, history has demonstrated that it hasn't proven to be very good at solving user-facing problems. If it were good at doing this we would expect to see open source crushing proprietary applications in the user space. But we don't see this very often.

Could it be that this model is not good at solving these kinds of problems?

Here is where I believe the problem lies. The developer-centric model works for solving technical problems because, as Eric Raymond pointed out, the people with the problem are at the heart of creating the solution. The developers know their itch.

When developers are solving technical problems, problems that they have, understand well and want to fix, it works extremely well. And when developers are

solving user problems, problems that are not their own, where they do not understand the issue intimately and the required solution is not primarily a technical issue, it does not work very well at all.

This is because in these cases we have broken the first requirement of the itch-to-scratch model: know your problem. The developers do not 'know' the problem intimately the way the users do because it is after all, not their problem. They are trying to solve the famous SEP, Someone Else's Problem. Consequently, they do not have the same in-depth experiential knowledge and understanding of the quirks and nuances of the problem that the users do. It is a qualitative, experiential, difference but it is an important one.

Consequently, we get software solutions that reflect the developers understanding of the problem and reflects a developer's approach to the solution; one that typically does not match the users understanding of the problem or their needs.

In open source today, developers are the dominant solutions providers. When they build solutions for themselves it is an almost perfect model. But when it comes to building solutions for a user base whose needs the developer doesn't intimately understand (because they are not the user), then the model fails.

How do we solve this?

I think the answer is clear. We stick to the itch-to-scratch model.

It is a good method for solving problems. It is the model that has enabled open source to squash the proprietary competition in the technical world. We should stick to it, but we need to be consistent in how we apply it. We need to keep the people with the problem, those with the itch, the people that understand the problem intimately, at the heart of the problem-solving effort.

It is not enough to add 'UX people' or 'user forums' to an otherwise developer-centric project culture to solve user needs. That approach is an interim patch and is only going to go so far. We need to go back to core principles of the itch-to-scratch model and bring the people with the problem (the users) into the heart of the solutions model.

To do that we need to begin asking ourselves what a user-centric solutions model might look like for open source.

What does it look like to you?