POSITIONING FREE OPEN SOURCE SOFTWARE

How maintainers of open source projects can use positioning strategically to accelerate their project's growth, build a dedicated community, and build exceptional software.

EMILY OMIER

Emily Omier is a positioning consultant who helps founders of open source companies use positioning strategy to differentiate themselves in the marketplace, streamline their product and project roadmaps, and increase the quantity and quality of leads, ultimately leading to both improved revenue growth and community growth. She also hosts The Business of Open Source, a podcast about building open source companies.

Table of Contents

Introduction

Most users of open source software have experienced the frustration of encountering a project's website or ReadMe and not being able to understand what the project does; Most people who use open source software have had at least one experience in which they spent time reading the docs and setting up the software only to discover they'd misunderstood the project's core goals.

When that happens, the project has a positioning issue. As a maintainer, your goal should be for people who would benefit from the project to find it as quickly as possible, and for those who would not benefit from the project to understand that it's not a good fit as quickly as possible. When you do this, you're both able to accelerate growth in the number of project downloads and users, but also able to build a community of people who are trying to accomplish the same things — and reduce the number of distractions from poor-fit community members.

In this eBook, you'll learn how to apply product positioning strategy to an open source context, from why positioning is particularly important for open source projects and how to get the information you need to position your project to the process you should go through as you position the project and what you should do once you've established a strong project positioning.

If you're invested in the sustainable success of an open source project, this eBook is for you.

WHO IS THIS EBOOK FOR?

This book is for anyone who cares about the continued, sustainable growth and success of an open source project. It is for maintainers who are willing to think strategically about their project, not only in a technical sense but also in terms of long-term community, sustainability, and growth prospects. If you are not interested in thinking about the non-technical aspects of building, growing, and sustaining an open source project, you won't get much value out of this book.

Even within that audience, there are different reasons you might be invested in the long-term success of an open source project. Here are some of the specific personas I'm writing this book for.

Founders of open source startups. As the founder of an open source startup, positioning your open source project well is a core part of your business strategy, because it's a critical part of how you position your company. Your open source project also has to be positioned

in a way that makes sense given your commercial offering, as the two have to be positioned relative to each other but distinctly.

This eBook is focused only on the open source component of an open source company's positioning challenge, but that is usually the first positioning challenge an open source founder encounters and is critical to building a strong community and long-term success for the company.

Founders of open source startups often find that positioning problems aren't just a business challenge — they can be personally frustrating and embarrassing. Being unable to explain what a project does in a way that people can understand — feeling stumped when asked to explain why your project is better than a well-known alternative — these are both embarrassing and bad for adoption and commercial prospects.

Open source program office leaders. Open source program offices are tasked with helping their company set success metrics for the open source projects they create as well as helping teams reach those success metrics. Good positioning is a part of making any project successful, assuming that 'success' means 'people using it and liking it.'

Individual maintainers, particularly those interested in future monetization. Individual maintainers are always struggling with limited time and endless to-do lists, whether they are developing the project as part of their job or entirely in their free time. Evangelizing the project is probably not at the top of many maintainers' urgent to-dos. However, especially if you, the maintainer, hope/plan to monetize the project at some point in the future, getting the project's positioning as strong as possible is the best way to get the most growth out of your limited evangelism efforts, as well as the best way to set the project up for successful monetization in the future. Better positioning will also help clarify what can be removed from the project roadmap — in other words, what things you can eliminate from your to-do list without harming your project.

Like startup founders, maintainers also often find poor positioning personally embarrassing and frustrating.

The bottom line is that this book is for anyone who cares about the long-term success and growth of an open source project. It will be especially valuable for those who are personally invested in making the open source project a success, with widespread adoption.

What is Positioning?

Positioning is all about controlling the assumptions that people make about your project and narrowing down the audience of potential users as much as possible. The goal is to find a way to describe your project that is specific, differentiated, and immediately understandable to people in your target audience (who could, to reiterate, be your users, your potential contributors, or another group you've decided you want to reach).

Most projects could be useful in a variety of use cases, and the key to good positioning is to find the use case and target audience where your project will deliver the maximum value and focus on that use case exclusively. The goal is to identify the problem your project can solve dramatically better than any other option, and then ensure that you're making it as easy as possible for people who have that problem to find your project and recognize it as the solution.

SETTING THE CONTEXT AND CONTROLLING ASSUMPTIONS

Nothing exists in a vacuum, whether it's a consumer product you find on the shelf in a big box store or an open source project you hear about from a friend. Everything is surrounded by rich contextual clues that give you an idea of how you should be thinking about the product or the project. These contextual clues can be used to your advantage if you're aware of them and consciously choose them. Even if you don't pay attention to the context around your project and the assumptions that context leads people to make, it's still there — but possibly leading people to make assumptions about your project that don't show it in the best light possible.

At its most basic, positioning is about controlling the context your project is put in so that the assumptions people make when they encounter the project are in line with what you want them to assume.

For example, when you say that a certain project is cloud native, it can cause people to assume that it's only useful for people running microservices. Julien Pivotto, one of the maintainers of Prometheus, talked about this issue in the podcast we did together, saying that when Prometheus is described as a cloud native monitoring tool it leads people to assume not just that it *can* monitor Kubernetes clusters, but that it is *only* good for monitoring Kubernetes clusters. This isn't true — Prometheus can be used to monitor anything and is used to monitor data centers and wind turbines, not just Kubernetes clusters.

Sometimes the context that your project is in can't be changed. For Prometheus, the fact that it's hosted by the Cloud Native Computing Foundation means it will probably forever be associated with cloud native technology. But most open source maintainers have a lot

of tools at their disposal for controlling the context around their project and influencing the knee-jerk assumptions that people in the ecosystem make about their project. If you go to the Prometheus website, for example, you don't see it being called a 'cloud native monitoring,' precisely as a way to position away from cloud native use cases only.

YOUR "MARKET" CATEGORY

The most important way to signal what assumptions people should make about your project is in how you define your market category. Market categories are short descriptions of your project, such as:

- The distributed SQL database for cloud native applications
- Open source content management system

The key to a good market category is that it has to be specific, it has to be unique and it has to make sense to your target audience. It also should be short — under about 8 words is a good guideline. Your market category is the answer to the question, "What is XYZ project?" As such, it should be a noun phrase, with some adjectives added in. A market category that is all adjectives doesn't work.

Specific

The most common mistake maintainers make in identifying their market category is not being specific enough. Think about the projects that call themselves 'software supply chain security' — this is so vague that it fails to give potential users enough information to know if the project will solve the problem at hand. It also sounds like BS. A good market category description has to make it immediately clear what specific problem the project solves.

Unique

Your market category should also be unique. You're defining which slice of the market you want to own completely, so you should make sure there are no other projects that describe themselves the same way you do. These criteria are easy to test for: If any other projects are describing themselves with the same words, you need to find a new market category. Walk around KubeCon and you'll probably see ten booths calling themselves "Cloud Native Security." For those companies or projects, their positioning is a failure.

I gave the example of an 'open source content management system' above, but open source projects need to be careful about putting open source front and center. First of all, it's often not unique — the competitive alternatives are usually other open source projects. Second of all, the fact that a project is open source is often obvious from other clues, like the fact that you're reading a ReadMe on GitHub. Third, 'open source' is a feature, not a value/outcome. There are different reasons people might want an open source software over a proprietary one: Transparency, flexibility, and low cost. Make sure you're using the limited number of words you have for your market category to highlight the things that are critically important for your audience, that they cannot get from other contextual clues.

Understandable

Even though your market category needs to be unique, it also has to make sense. Another common mistake with market categories is to describe your project in a way that's so unique, that no one understands it.

The way to get around this is to combine concepts and categories that people already understand, but that no one else is combining. Another way is to take a category that everyone already understands, and then apply it to a very specific target market in a way that no one else does.

NARROWING DOWN YOUR TARGET USER CHARACTERISTICS

The second critical part of positioning relates to having a clear understanding of who your target users are. The most common mistake here is to define your target user too broadly. For example:

- JavaScript developers
- Kubernetes engineers
- SREs

Those are job titles, and they are way too broad to meaningfully narrow down your target users. The key to getting it right at this stage is to stop thinking about who *can* use your project, and instead focus on only those people who will get the maximum benefit from the project.

Another common mistake is to get overly focused on job titles or industry verticals. In some

cases, those can be important — but they often are not. Characteristics of people in your target market might be about technical constraints they operate under, other technology they use, compliance frameworks they have to meet, business constraints or business objectives, or even personal priorities or values. You won't know exactly what those constraints are just by a person's job title.

Values are more likely to be important in defining your target user when we're talking about an open source project than they are when discussing a commercial product, because choosing to use an open source project is often an individual decision, even when it's being used professionally.

For example, Ethyca, the maker of Fides, a project that helps developers get more visibility and control over the privacy implications of the software they create, determined that for the core open source project especially, one of the key characteristics of a target user is that he or she is personally paranoid about privacy. It's a personal value that a person holds, and there's no way to connect it to a job title or industry vertical. Which is fine.

Another important characteristic is outcomes. What outcome is your ideal user expecting? What outcomes is he or she responsible for delivering in his or her organization? These are much more specific than job titles, and much more relevant as you position your project.

The idea is that you want to get so specific about the exact person who'll get the most out of the project that when someone who is in your target market encounters the project, they will feel like '*This was made specifically for me.*' That won't happen unless you've focused on a group of people with very specific needs, values, desires, and goals.

Lastly, a mistake to avoid when you're thinking about the characteristics of your target market is placing too much attention on irrelevant characteristics. A user's age, for example, is rarely going to be relevant to how much value they get from an open source project. Experience level, however, might be. Often, having experience with a certain problem or a certain technology in the past is a very good indicator of how much a user is likely to understand and appreciate the value of a project quickly.

Many marketing books are written for marketers of consumer products, so when they talk about segmenting markets, they think about personal characteristics like someone's disposable income, age, marital status, gender, how often they eat in restaurants, etc. None of those characteristics are relevant for open source projects and they don't belong in your list of user characteristics.

YOUR POINT OF VIEW

Your project has to stand for something. As you've created the project, you've made a series of decisions based on opinions you hold — including technical opinions, values, and opinions about the ecosystem your project is part of. In some cases, those decisions are not particularly interesting — anyone else would probably make the same ones. But every project has some controversial opinions baked in, whether it's the technical decisions or the philosophical approach that led you to create the project in the first place.

Let's take the example of Snyk. One of the core opinions behind Snyk is that developers need to have both more agency and more responsibility for the security of the code they write. Not everyone is going to agree with that — which is ok. You don't want everyone to agree with your point of view — that would defeat the purpose.

Your point of view is an important part of positioning because your goal as the maintainer of an open source project is to build a community of *like-minded* people. You need to be clear about which opinions are core to your project so you can make it clear to potential community members what those opinions are — and they can decide if they agree or not. Because of the nature of open source, you need people who agree both with your philosophical approach as well as your technical approach, especially if you aim to get more code contributors. Someone who fundamentally thinks your technical architecture is wrong is not going to be a good fit as a code contributor. Someone who holds a fundamentally different worldview (related to your project) is also going to be a poor fit and will become a distraction.

In the example of fides, the privacy engineering platform, one of the baked-in opinions is that protecting user privacy is valuable in and of itself, not just because regulations require it. Someone who views privacy as a box to check to meet compliance frameworks can certainly get value out of the project, but they would not be an ideal fit, particularly not as a code contributor.

YOUR UNIQUE VALUE PROPOSITION(S)

So, what exactly is unique about your project? Why would someone choose it over the other options?

And we're not talking about features or underlying technology. We're talking about values, or why those features matter.

Your unique values are the pillars of your positioning that hold up everything else. You don't need more than one, but shouldn't have more than five at the absolute maximum — and better to have no more than three.

A value proposition is related to the outcome someone gets from using your project. For example:

- Reduce errors
- Get to market faster
- Tighten your security posture

You should also have a clear idea of how your unique values relate to the features or attributes of your project and should record them in your positioning canvas. However, it's the unique values that you want to stress when you talk about your project. Think of the features and attributes as part of how you prove that you provide the outcomes that you provide. When someone asks 'How do you do {value}?' You want to be able to tell them the exact features or attributes that contribute to the outcome you're promising. But lead with the outcome, not with the feature.

Free Open Source Software and Positioning

The focus of this book is on positioning for free open source software — free as in freedom and free as in beer. For this eBook, we'll treat all free open source projects the same, whether they are run by a single maintainer in his or her garage as a side project, maintained by a large corporation and spearheaded by an Open source Program Office (OSPO), or are part of an open source startup that has a commercial product related to the project. When it comes to positioning, the approach you would take as a maintainer is the same in all of the above scenarios. While the latter two scenarios are likely to have more resources available than the single maintainer working nights on a project, open source projects as a whole tend to be more resource-constrained than commercial projects, so many of the principles still hold. Importantly, the challenges of positioning open source projects — a general lack of transparency into user identity, user behavior, and user motivations — are constant regardless of the project's financial resources.

POSITIONING FOSS VERSUS POSITIONING COMMERCIAL PROJECTS

One of the biggest challenges in positioning open source projects is that you, as the maintainer, just don't have your finger on the pulse of your user base in the same way a commercial product's sales team/customer support team would — particularly if the commercial product is sold in sale-led, enterprise-style motions.

Sales teams are going to understand things like what problem the customers want the product to solve, who the true stakeholders are, what alternatives are being considered, and what does and does not matter for the customer. Customer support teams are going to understand how customers actually see the product after they've had a chance to use it, and how their experience differs from the expectations set during the sales process. Customer support teams generally know exactly what users hate about the product, and also what they love.

Even a commercial product that is entirely self-serve, however, will usually provide much more telemetry and information about user behavior than an open source project will give maintainers. Even talking about telemetry is taboo in the open source world, and while it can be done ethically (if you are transparent about exactly what telemetry is built into the project and why you've done so), even projects with some built-in telemetry will have less of it than a self-service SaaS product would. This remains true if we're talking about free tiers of commercial products.

A free tier or self-service software will usually gather information about users' identity and user behavior, giving the company the ability to understand how many of the people who sign up actually use the product, how frequently they use how, and where in the onboarding process they might be getting stuck. There's also an expectation that when you've signed up for a self-serve software product, including one that is free, you are part of the company's sales funnel. Users and customers will expect a certain amount of nurturing and upselling. It is possible to overdo it and alienate your users with a pushy sales process, but the difference is that with an open source community tolerance for any sales communication is zero.

So, for open source maintainers looking to understand their users? You pretty much have nothing unless you specifically go out looking for it — and the only way to look for it is the time-consuming, 'old-fashioned' technique of reaching out to people and talking to them. You can also look at comments on your Slack/Discord/Community area, but that can be misleading, because only a certain subset of your users will ever interact with your community, and you have no way of knowing if those who do are more or less excited about the project than your average user. You will still need to initiate conversations: Your community is a great place to find users to talk to, but you have to proactively reach out to them and start conversations because just looking at comments they've made will not be enough.

You have to get that information, though, because to position your project well, you need to understand how your most enthusiastic users think about the project, discover your project, and interact with your project. And in an open source context, you'll almost always have to proactively reach out to users to get that information.

The goal of positioning an open source project

Yet another complicating factor when positioning an open source project is that you have to start with your goal — and that is less straightforward than the goal for a commercial product.

When you position a commercial product, the end goal is clear: Increase profits. One of the obvious questions to ask during the positioning process is "Which customers are the most profitable?"

Open source projects do not all have the same goals. Before even beginning the positioning process, it's important to understand what end goal you're positioning for. Is it:

- To get more total downloads
- To get more active users using the project in production

- To get more contributors
- To drive adoption of a commercial offering
- To increase profits at your commercial open source company
- To get someone to hire you
- To find people to hire

This adds a step to the positioning process, one that too many open source projects neglect when they start thinking not just about positioning but about evangelism and growth in general.

Positioning for a technical audience

Open source projects also often have the 'selling shoes to shoemakers' problem — something they have in common with many developer tools. Unlike your average B2B product for lawyers or accountants, any project or product that is aimed at a developer market has to strike the right balance of value-based and technical information. When you're positioning a project for people who could — or at least think they could — build it themselves, you encounter positioning, messaging, and communication issues that don't come up when the audience would never dream of building something similar themselves.

This doesn't apply to all open source projects — there are certainly open source projects whose primary users are non-technical. For those types of open source projects, this isn't as much of a concern.

However, many open source projects have to strike this balance as they position their project. If the maintainers' primary goal is to attract more contributors, they will always have to grapple with the 'shoes for shoemakers' problem, regardless of how technical the average user is. You will always have to convince people that they should join your project rather than going out and building their own.

This does not mean you should lead with feature-based messaging. No matter how technical your audience is, you need to lead with the outcomes someone can expect to get from using your project, in your positioning and all your communication. At the same time, you should be prepared to connect the dots between those outcomes and the specific features and technical specifications that make those outcomes possible.

POOR POSITIONING IN THE OSS ECOSYSTEM

Unfortunately, poor positioning is extremely common in the larger open source ecosystem, even though open source norms force maintainers to state what their project does as succinctly as possible at the beginning of the ReadMe.

Even though most maintainers know they need to write a ReadMe, they often aren't very good at doing so. The result is that most first-time visitors who land on the ReadMe (or on the website) don't get a clear understanding of what the project does, who it is for, or which situations it is or is not ideal for.

This is bad for everyone. For users, it means that they waste time digging deeper into the docs or actually trying out the software without having a clear idea of whether or not it will solve their use case. Multiplied by the millions of open source projects out there and that a potential user would evaluate multiple projects for any given need, that means open source users end up wasting a lot of time testing out projects that will never meet their needs.

It also makes redundant projects more likely. If it's hard to find the project that meets your needs, you'll be more likely to start a competing project — simply because you didn't know an existing one already meets your needs. This leads to duplicated work. If there are too many projects that are so similar that there's no clear differentiation between them, it also dilutes the user base and the potential community, leaving all the projects weaker.

Poor positioning hurts open source users. When READMEs and homepages don't make sense or unintentionally misrepresent what a project does, users end up wasting time wading through the documentation and/or downloading and setting up projects that ultimately don't meet their needs. Or they end up using a project that sort of meets their needs but that isn't ideal for their use case but fail to discover the project that would have been a perfect fit.

Poor positioning hurts the open source ecosystem. When users waste time evaluating projects and have to deal with misleading or confusing information about the projects, it erodes trust in open source software in general. Especially in a business context, users need to find software that will meet their needs as quickly as possible. They need that software to work as promised. Poor positioning that makes it hard to discover the best project for the job undermines engineers' faith in the entire open source ecosystem.

Poor positioning is bad for maintainers. The closest parallels between open source projects and commercial projects are often drawn between maintainers and companies

who create a project. It should be obvious that poor positioning hurts maintainers who want their projects to grow.

- Poor positioning makes it harder for users to discover the project.
- Poor positioning can attract the wrong users, who try to pull the project in a direction away from what it really does best.
- Poor positioning usually translates into less focus, so the maintainer lacks a clear idea of what to focus on and what to de-prioritize.
- Poor positioning is embarrassing. It means a maintainer feels uncomfortable explaining what the project does.
- Poor positioning means wasted time and wasted resources. Open source projects never have enough of either. Tightening your positioning will let you get more growth with fewer resources.

Maintainers feel the pain from poor positioning most acutely. They are also the ones who have the most power to improve positioning. No single person is going to fix the entire ecosystem's positioning problem, but as a maintainer, you can fix your own project's positioning. This eBook will teach you how.

GOOD POSITIONING IS BETTER FOR THE ENTIRE OSS ECOSYSTEM

If every open source project had focused, immediately understandable positioning that was communicated on the ReadMe, it would be good for the entire ecosystem, not just for individual projects' maintainers. It would:

- Make it easier for users to find the projects they want
- Prevent projects from drifting away from their focus and trying to do too many things
- Reduce the number of redundant projects
- Improve the cohesiveness of open source communities
- Improve projects' quality
- Improve the reputation of open source in the engineering world

So how do you know if your project's positioning is off? Read on.

Symptoms of Poor Positioning

How do you know that you have a problem with positioning? First of all, it's helpful to recognize that positioning isn't something to be constantly optimized and adjusted or something that you do more of to get better results. If you're not seeing signs that your positioning is broken, don't fix it.

On the other hand, it is important to recognize signs that there might be a problem with your positioning. Here are some symptoms of a positioning problem that you shouldn't ignore.

PEOPLE YOU TRUST SAY YOUR POSITIONING IS A PROBLEM

Before launching into a discussion of how you can recognize positioning problems, let's start with a cold truth: Other people will notice your positioning problem before you do. It's up to you to listen to them.

Who might this be? It will usually be someone who understands your project and the ecosystem you operate in, but who also has enough distance from the project to avoid drinking the same Kool-Aid as the maintainers.

Some people who might be likely to see positioning problems before you do are active contributors who are not maintainers, colleagues you've told about the project and who've interacted with it, and anyone outside of the maintainers who've helped with documentation or creating a website.

If you're commercializing an open source project, more people might spot positioning problems before you. Investors frequently see positioning issues before founders, as do PR experts.

The bottom line, though, is that if someone with expertise in your field says that there might be a problem with your positioning, you should pay attention to that feedback. It's a strong signal that there is a problem.

PEOPLE IN YOUR TARGET AUDIENCE DON'T UNDERSTAND QUICKLY

If you're talking to people who are in your general target audience and they don't understand what your project does or why they should care within five minutes, you have a positioning problem.

When positioning works correctly, people in your target audience should be able to understand what outcome your project provides, whether or not the project is appropriate for their use case, and what pain points it solves before they scroll down on the project's homepage. If you're having a conversation with someone, you should be able to describe the project in one (short) sentence that gets the person you're talking with to nod in understanding.

→ Note: Some people talk about the 'mom test.' This only works if your mom is in your target audience! If your mom does not meet the characteristics of your target user, whether or not she understands what your project is about does not matter. You want to *avoid* using language that is so general that it makes sense to a non-specialized audience because it will end up sounding like BS to people who are in your audience.

ODD/OFF-TOPIC QUESTIONS IN COMMUNITY SPACES

Positioning is largely about people understanding your project more quickly, you need to look for signs that people aren't getting it. Often, people won't get it even after spending a lot of time with your project. That is a sign of a positioning problem.

The surest sign that people don't understand what your project does is questions popping up in your forums or other community spaces that seem strange.

We might like to say that there are no stupid questions, and that's *true for the asker*. But the questions you get are a strong signal about what people do or do not understand. Sometimes getting 'dumb' questions is just a sign that you need to improve your message. But if you get questions that seem off-topic, you should pay attention. This includes:

- Questions about functionality, when it seems like it should be obvious that you wouldn't have the functionality in question
- Requests for features that seem out of line with the goals of the project
- Questions that make it seem like the asker fundamentally misunderstands the project

The key thing to think about is the disconnect — if there's a disconnect between what you think people should be discussing and what they are discussing, you have a positioning problem. It might be that you are miscommunicating about your project and that the people in your community misunderstand the project's value. But it is just as likely that *you*

misunderstand your project's value, and need to take the hint from the community that the project's true genius is something you hadn't expected entirely.

But right now, we're talking primarily about symptoms of poor positioning, so the key is that if there seems to be a disconnect between how people are talking about the project in your community spaces and the way you would expect them to be talking, that's a sign that your positioning needs to be examined.

BEING COMPARED TO PROJECTS/ALTERNATIVES THAT AREN'T REAL COMPETITORS

Another sure sign that there's a positioning problem is that your users are comparing your project to alternatives that you don't see as competitors. This most frequently happens with other projects or technology that you see as complementary and might even integrate with, but users (or others, like the press) ask questions that imply that the two options are competitive. If this happens, you have a positioning problem.

STAGNANT GROWTH

If your project's growth seems to have stagnated, there could be many reasons. But it's always worth investigating whether or not positioning could be the problem. Especially if you had rapid growth at the very beginning, when you had higher-touch interactions with each new user, and then have seen that growth stagnate as you rely more on word-of-mouth and your website to spread the word.

Assuming your project is as awesome as you think it is, stagnated growth is a sign that *something* is wrong.

HIGH BOUNCE RATES

If you find that a lot of people visit your documentation but don't download the project, it might be a positioning problem. An even better signal for this is high churn rates, but unfortunately, it's often hard to tell with open source projects what percentage of the people who download the project go on to use it regularly.

It's hard to see from just bounce rates on your home page whether or not there's a positioning issue. Because good positioning will attract some people, but should also repel those who aren't a good fit. So from just raw bounce numbers on a home page, it's hard to tell if you're repelling the right people or the wrong people. But if instead you look at how many people who've dug deep into your documentation end up downloading the project, you can get a sense of whether or not people are leaving even after spending a significant amount of time to understand what your project does.

DISCONNECT IN GROWTH BETWEEN OPEN SOURCE AND COMMERCIAL OFFERINGS

If you're commercializing an open source project, you should expect to see relatively steady growth in both your open source and commercial offering(s), especially if you are investing in both. If it feels like one is growing much slower than the other, that could be a sign of a positioning problem.

Relatedly, many open source startups worry that their open source project is cannibalizing their commercial offering. If you worry about that, it means you're not clear on how your open source and commercial offerings are positioned differently, with different target markets and different value propositions.

Improving your Project's Positioning

If you've seen some signs that your project's positioning could use improvement, what do you do? What follows are some concrete steps to follow to improve your project's positioning.

But first, some tips:

Do this exercise with your co-maintainers, if you have them. You'll get a better result if you get multiple perspectives, and you'll need all maintainers to buy into the result if you want to take action based on the new positioning.

Solicit feedback from your community. You don't want too many cooks in the kitchen, especially if you're doing this exercise during a synchronous meeting, which is what I'd recommend. However, getting community feedback on each step of the process is a very good idea, especially because transparency is one of the pillars of successful open source communities. It can be as simple as asking people in your Slack or Discord to contribute their thoughts on each step.

WARNING: RE-POSITIONING A PROJECT IS EMOTIONAL

Getting positioning right involves making some hard choices. Changing positioning means changing the project's identity. Before you work on improving your project's positioning, you have to make sure you (and your co-maintainers, if you have them) are prepared to let go of your previous assumptions about what your project is and what you're building.

If the most important thing to you, as a maintainer, is to continue to develop your project in the same direction as you imagined when you first started the project and to continue building exactly the project you want rather than what your users want, then there's no point in working on positioning. If growth is really important to you, you will have to let go of your previous ideas about what your project is to find the best way to position it for growth.

In my experience, at least half of the positioning challenge for open source maintainers is not related to the technical/intellectual challenges of determining the right positioning for the project. It's the need to let go of all your previous ideas about who your project is for and what use cases it was best suited to. It can also mean downgrading, in the short term, the size of your potential audience.

You're probably not used to making emotionally fraught decisions about your open source project. But it's counter-productive to pretend that positioning decisions aren't emotional. Positioning is about the project's core identity — and by extension, it can be a part of the

maintainer's identity. If you've been creating a project with a particular use case in mind, as the maintainer you get attached to being the person who solves that use case. Pivoting to another use case requires letting go of that previous identity and adopting a new one.

Improving positioning can also be emotionally challenging because of the focus it requires. You'll need to carve out an audience that will feel uncomfortably small, and focus entirely on that audience — which means ignoring a lot of potential users who would get some benefit from the project but not *as much* benefit as those in your ideal audience. This is scary for maintainers, and it's even scarier if you're running an open source startup, where financial success depends on open source growth. It feels limiting and counter-intuitive because it will feel like you're sabotaging your growth. You're not — even the narrowest definition of your audience is much larger than you realize. But being very specific about both the problem you solve and the people you solve it for does mean that you're saying you do not help everyone.

You shouldn't ignore the emotional component of positioning. Whenever I work with companies, I always ensure beforehand that all the founders are emotionally prepared to let go of the current identity of the project and company. If any of the maintainers or founders are too attached to the current positioning — or too afraid of narrowing in on a specific niche — working on positioning is ultimately going to be a waste of time. And while you can bring someone in to help you overcome the intellectual challenges to improving your positioning, there's no shortcut to overcoming the emotional hurdles on your own.

GET CLEAR ON YOUR DEFINITION OF SUCCESS

The first step is to understand exactly what you want to achieve with your open source project. A shocking number of maintainers don't put much thought into what success will look like for their project — and this goes for maintainers of all kinds, from hobbyists to employees at big companies and even to founders of open source startups. It's important to be specific here. You might say you want your open source project to grow. But what does that mean?

Do you want your open source project to have more downloads? More users? More stars? More community engagement?

More importantly, why did you create this project in the first place? Why do you continue to maintain it?

Because before you start working through positioning your project, you need to know what success will look like.

The profile of people who will become active community members is different from the profile of people who will be very satisfied users but not necessarily contribute to the community. Which is more important to you?

If you do not know what success looks like, your chances of reaching success are much lower. Start the positioning process by defining what growth means to you, how you want your open source project to contribute to your business or your life, and what metrics you would use to determine that your project's growth trajectory is on the right track.

TALK WITH USERS

Hopefully, you're already talking with users regularly. If you aren't, you should start your positioning exercise by making sure you talk with several users.

These users shouldn't be randomly selected. They should be the type of users you want more of. For example:

- If you want more users who work at a particular type of company, make sure you talk to users who fit that profile
- If your goal is to increase the number of people who are involved in the community, make sure to talk to your most active community members

When you talk to users, you need to come in with a very open mind and not let your preconceptions about how your project should be used influence the discussion. Your goal is to uncover information about:

- What led the user to start looking for a project like yours?
- What outcome were they hoping to get from using your project?
- What were they doing before they started using your project to get that outcome?
- What alternatives did they consider?
- What did they expect from your project before they started using it?
- How closely did those expectations match the reality of using the project? What were the similarities and differences?
- What surprises did they encounter when first using the project?
- How do they use the project now?

- When do they use the project?
- Are there other people on the team who also use the project? Who are they? Why do they use the project as well?
- How would they describe the project to a colleague?

During the conversations, pay special attention to anything that users describe as an aha moment or a magical moment.

As you're talking with users, it's important to simultaneously pay attention to the ideas the users convey as well as the language they use. What they are saying as well as how they are saying it. You don't want to cut and paste your users' language into your positioning and messaging, but you *should* use it as the starting point for determining how you should talk about your project.

You'll be using the input you get from users as, well, input, in the positioning process. It is an important signal but also needs to be combined with your vision for the project and your experience with the project.

IDENTIFY ALL THE COMPETITIVE ALTERNATIVES

Once you've talked with users, you need to make a list of all the competitive alternatives to using your project. The most common mistake people make when thinking about their competitive alternatives is to only consider other software solutions.

In most cases, there will be some open source software that is at least somewhat competitive with your project, as well as some proprietary software. But there will also be non-software strategies for getting the same (or similar) outcomes, and you need to make sure you list those. Often these involve doing something manually, hiring another engineer, paying (or risking) a fine, or essentially crossing your fingers that nothing goes wrong.

For example, if your project helps evaluate compliance with open source licenses, one legitimate competitive alternative for a user is to cross their fingers that they aren't violating any licenses, and that if they are, no one finds out. This sounds cynical, but it is how users think and how businesses operate. In this example, if the cost, financial or in time, of getting the project up and running is greater than the hassle of dealing with a discovered license violation, no one will ever use the project.

IDENTIFY THE PAIN THAT ALL COMPETITIVE ALTERNATIVES CAUSE

Why do those competitive alternatives suck? What makes your users search for an alternative?

When you do this step, think of all of the pain points that your users experience when they use the competitive alternatives. Again, you want to think about personal pain points and organizational pain points. Some pain points will be felt acutely by an individual, whereas some harm an organization.

Individual-level pain points might include:

- Frustration (about something related to your project)
- Worry (about bugs, about security or compliance risks, about looking bad in front of colleagues/a boss)
- Overwork/burnout
- Ethical concerns

Those pain points can also be expressed as organization-wide pain points and might be expressed as:

- Slowing down the development process, slowing down the process of getting a product to market, inability for the engineering team to meet deadlines or predict when a product would be finished
- Risk of financial penalties from legal and compliance violations
- Loss of reputation in the marketplace after security incident
- Loss of customer trust after an outage
- Employees burning out and leaving, leading to high turnover in the engineering department and slowing down product delivery

For free open source software, you will need to pay attention to both categories of pain points, but especially to individual-level pain points. Free open source software's adoption pattern is usually bottoms-up, with an individual discovering and adopting the software before bringing it to wider use in the organization. The pain points the ideal user experiences as an individual should be the primary pain points the project highlights.

What if your project doesn't solve all the pain points you list?

You should focus on the pain points your competitive alternatives cause that your project does solve. Focus on the ones it solves now. However, for a pain point to be worth highlighting you don't have to completely eliminate the pain, you just have to make it better.

For example, let's say a process takes two weeks to complete without your project and two days to complete with your project. Two days is still annoying, but it's better than weeks.

LIST OUT YOUR PROJECT'S UNIQUE ATTRIBUTES

Once you've figured out what pain points the competitive alternatives cause, it's time to switch gears and focus your attention on your project. The question you want to ask yourself here is "What is unique about this project compared to the competitive alternatives?"

In this part of the exercise, you want to think broadly about your project's attributes. There are two 'gotchas' to avoid.

First of all, you want to think of attributes in general, not just project features. Attributes can include the reputation of the maintainers, the language the project is written in, the project's longevity, or the geographic location of the maintainers — to name just a few attributes that do not count as features.

On the other hand, you also want to make sure that you're focusing on things that are *unique* to your project and not shared by any competitive alternatives.

For example, if *all* of the competing alternatives are open source, the fact that your project is an open source project rather than a commercial product doesn't belong here and isn't something you'll want to stress when talking about how amazing the project is.

Features versus attributes

Features are a subset of a project's attributes. When you're considering your positioning, you want to think about all of a project's attributes, not just its features.

Attributes include specific features, but also include things like:

- Made in Europe
- Maintainers are highly respected in the field/have written books
- Written in Python

This is a brainstorming process — expect to make a long list of your project's unique attributes.

MAP THOSE UNIQUE ATTRIBUTES TO UNIQUE VALUES

At this stage of the process, you need to take attributes — features and other attributes of your project — and map them to values, i.e., why someone cares.

Most open source maintainers vastly overestimate how easy it is for someone to make the jump from attribute to value themselves. Especially since most open source projects are aimed at a technical audience, maintainers automatically assume that their highly technical audience can take a list of technical features and easily translate that mentally into why that technical feature matters.

No matter how technical your audience is, you need to make sure you're isolating your unique value, not your unique features.

EXAMPLE of what a unique value is, and how it's related to attributes:

Attribute	Benefit	Value
Written in Python	Data engineers can customize the project and contribute back to it themselves, without having to interact with software engineers, because they already use Python	Improves the productivity of the data engineering team and allows them to iterate more quickly
Open source	You or your security team can verify for yourself that the code does what we say it does	You maintain control over your security posture and data privacy.

The most common pitfall here is that maintainers can ignore the competitive alternatives that are a version of manual process or using a general purpose software like Excel. If one of the competitive alternatives to using your project is using Excel to track something, the fact that your project is easy to learn is irrelevant — it probably isn't easier to learn than

Excel. In that example, chances are your project leads to more accurate data, is easier to keep updated, or something else. But it's not easier to learn than Excel.

Another important point is that some attributes can have different benefits and values in different contexts. This is why you need to explicitly understand why *your* users care about that particular attribute.

As you're working on mapping each attribute to a value, you'll notice that things start to get repetitive. Some values are going to come up over and over again. That's your jumping-off point for the next step, which is pulling out three or fewer value themes.

Open source is an attribute, not a value (and it might not be unique)

At this stage in the process, another major pitfall for open source maintainers is placing too much emphasis on the fact that the project is open source. This is a problem for two reasons.

1. **Open source is an attribute, not a value.** There are many things related to your project being open source that a user might value. Maybe your users are broke and they like the fact that your project is free. Maybe they are paranoid and they want to be able to inspect all of the code themselves. Maybe they are interested in longevity and need the confidence that they'll be able to use your software even if all the maintainers are hit by a bus. The point is, that open source means something different, value-wise, to different users. You need to figure out why your users care that your project is open source. You also need to determine *if* they care that your project is open source because they might not.
2. **Being open source might not be unique.** If all the primary competitive alternatives to your project are also open source, you should not include the fact that it is open source as one of your unique attributes or as a unique value. *Note that if, for example, your primary competitors have different licenses or legitimately approach open source differently, you can still consider this a unique attribute.

FIND THREE OR FEWER VALUE THEMES

Your project is probably going to have dozens of features and even more attributes. But you should focus on no more than three value themes.

What if you have more than three values?

Humans have a limited ability to remember things. Three is the magic number of items that most people have no problem both grasping immediately and remembering. In the likely event that you have more than three value themes that your project provides, you should choose the three most critical values. Here's what you should ask yourself.

Is this value differentiated?

If you're considering adding a particular value theme to your list of three, make sure that your primary competitive alternatives don't already provide that same value. What you don't want is to end up promising to do *the same thing* as your primary competitors.

Sometimes, maintainers will find that a competing project or product promises to provide a certain value, but doesn't. If that's the case, when you're articulating the value statement, you have to make it clear that you provide this value in a way that goes above and beyond what the competitive alternative does.

Which values are most critical?

Not all values are created equal. When you're looking to reduce the number of unique values, think about how important each one is to your audience. Also, think about what they were looking for when they initially discovered the project. Since positioning is the foundation of your evangelism, you need to highlight the value(s) that are core to the decision to search for, download, and invest time in setting up the project. If you're in doubt, ask your community. But hopefully, by this point, you've had enough user conversations to know what your users care about most. Prioritize those values.

** Just one value theme is okay, too. I've worked with companies who find that they only really deliver one unique value, and that freaks them out. As long as it's a value that people care about, there's no problem with delivering only one value. If you have only one value to deliver, it drastically simplifies your outreach, your project roadmap, and your project strategy.

FIGURE OUT WHO CARES THE MOST ABOUT THOSE VALUES

Once you've gotten your list of three or fewer differentiated values, the next step is to figure out who cares about those values more than anybody else.

This is an open source project, so you want to think primarily about your users as individuals, not as part of a larger organization. However, in many cases, your users' jobs will be an important characteristic that could make them care about the values and outcomes the project provides, so you also shouldn't ignore the characteristics of companies that care about the values you provide, either.

This is where you're listing out the characteristics of people who are going to get the most value out of your project.

Here are things you should think about.

Psycho-social characteristics. For some projects, there's a serious ethical or mindset component to the appeal. For example, someone might be paranoid about security because they work at a bank and their work involves protecting bank account numbers; someone also might be paranoid about protecting their personal Netflix password for reasons that are completely irrational to most people. Someone who is personally paranoid about security is not quite the same as someone obliged to worry about security because of where they work.

Job responsibilities. If your project is used by businesses, perhaps the most important characteristic of your users to consider is what their job responsibilities are. This is not a job title, it's a description of what outcomes they are expected to deliver and what responsibilities they have. Included in this is the type of application or workflow your ideal user would be working on.

Skills and experience. It's common for an open source project to be especially valuable to someone at a certain skill level (for example, projects that make an otherwise very difficult technology easier to use / harder to make mistakes at are generally more valuable for relatively inexperienced users). On the other hand, in some cases, people without enough experience simply don't know enough to understand how valuable your project is, and you might find an ideal user is someone who's tried and failed to build something like your project already.

Organizational constraints. Do ideal users have to meet certain regulations? Do they work for organizations that have no budget? Do they work in remote teams or in person? List out characteristics of the organization or company the ideal user might work for, but think broader than industry vertical. Industry vertical might matter, but so might things like growth rate, number of job openings, recent downtime or security incidents, geographic location / legal regime — the list is endless because there are infinite ways to slice apart the millions of organizations in the world, and the right way to do so depends on what your project does.

The key is you want to think back to your unique values, your unique attributes, and the outcomes your users expect from your project, and use those to make a list of the characteristics of people who will get more value out of your project than everyone else. It's important to be as specific as possible. It's also helpful, at this stage, to think of characteristics of users that would make them a better fit for a competing solution, in other words, people who you should be actively encouraging to *not* use your project.

PUT A LABEL ON IT

Now comes the time when you pull everything together and figure out what market category to put on your project — in other words, what label you're going to put on your project. This is the last step in the process, and what you're doing is deciding on your project's fundamental identity, using all the information that you've uncovered in the previous steps to inform your work.

Here are the general criteria you should think about as you define your market category:

- It should be a noun
- It should be as specific as possible
- It should be differentiated
- It should make your unique, differentiated value obvious
- It should be succinct. 8 words or less is a good guideline
- It should use vocabulary your target market already understands.

Let's go over each of these criteria individually.

Your market category should be a noun. If you strip everything else away from your market category, you will come up with something like 'platform,' 'manager,' 'layer,' or another noun. This isn't enough information, but grammatically speaking, you want your

market category to be a noun, and you want it to be the logical answer to the question, "What is your project?" For this part of the exercise, you want to focus on what it *is* not what the project does.

You will add adjectives and other descriptives to the noun at the core of your market category. This might look like:

- A privacy-first observability platform for safety-critical systems
- The petabyte-scale data catalog

Test your market category to make sure it fits these criteria by checking that it makes sense as an answer to the question, "What is your project?"

It should be specific. After all the work you've done to isolate how your project is unique, the last thing you want to do is to have a market category that fails to communicate that uniqueness. For example, "End-to-end security platform for cloud native applications" is too broad — it is also not believable, because I wouldn't believe that you can address every security issue a cloud native application could encounter in a single platform. Being specific is what makes your market category believable as well as actionable. It makes it easy for potential users to understand precisely the pain point the project solves, so they can quickly decide if it does or does not match what they need.

It should be differentiated. If your market category statement sounds exactly like your competitors, start over. A non-differentiated market category is often the problem that causes open source projects to start evaluating their positioning in the first place. It's important, though, that as you work through different potential market categories you make sure that it is differentiated from your primary competitors.

This doesn't necessarily mean that each of the values you communicate has to be unique. For many projects, the unique value might be that it eliminates a trade-off.

For example, let's say your target users have previously had to choose between latency and reliability. One of your primary competitors provides very low latency but isn't reliable. Another primary competitor is extremely reliable but has higher latency. Your project eliminates this trade-off. In this case, your market category should stress that you provide both values, and you shouldn't worry that your primary competitors provide those two values individually if you're the only one who can do both.

It should make your value obvious. You want everything in your market category statement to hit your audience over the head with your differentiated value.

It should be succinct. Your market category has to fit into a short sentence. This means you have to choose the words very carefully and make sure you're not including any words that don't communicate your differentiated value.

The need to be succinct is one of the reasons it's often not advisable to include 'open source' in your market category. The only reason you would include 'open source' as part of your core market category is if a) the fact that you are open source is incredibly important to your users and b) your primary competitors are all closed-source commercial products. Many projects compete against other projects. If that's your case, then being open source is not differentiated and you shouldn't be taking up real estate to talk about it in your market category.

It should use vocabulary your target market understands. Your market category statement does not have to make sense to everyone — it shouldn't. Because if you are specific enough, you will likely be using fairly technical language that will make sense only to people who are your target users.

However, you *do* need to make sure that your target users understand all of the terms you're using in your market category statement. The core goal when you declare your market category is to ensure that people understand what your project is, what it should be compared to, and what its unique value is *immediately*. This means you need to use vocabulary and terms that they will understand.

What's Next, After the Positioning

Positioning is a strategic exercise, and it won't do your project any good unless you take additional steps to translate your positioning strategy into tactics and then into actions.

WRITE IT DOWN

The first step, once you've figured out how to position your open source project, is to make sure to write it down. You should do this even if you are a sole maintainer and do not have anyone to share it with. The larger the maintainer team is, however, the more critical it becomes that there's a written source of truth about everything in your positioning. Even with just one person, it can be hard to spot if your positioning is inadvertently changing if you haven't taken the time to record it in writing.

The appendix has a positioning canvas to use as a template (or you can find it in Google Docs form here). If you've gone through the entire process, you should be able to complete the entire positioning canvas. After you've done this, make sure to save the positioning canvas somewhere that you can easily find and refer to. If you do have more than one maintainer, you should do the positioning exercise together, and every maintainer should have a copy of the positioning canvas and agree with everything it says.

If you have any frequent contributors or people who are very active in your community, it can also be useful to share the positioning canvas with them. This is a way to ensure your critical community members are on the same page about the project's value, the project vision, and the project's competitive alternatives.

UPDATE YOUR README/WEBSITE

Next, you need to update your ReadMe and, if you have one, your website.

Your ReadMe should clearly state your market category, in the section that is often called your "mission statement." The format usually goes something like "Project X is a {market category}." Underneath that, you should make sure you communicate very clearly who, precisely, your ideal users are and the unique value that your project provides.

In other words, you should get all the information that one might put on the homepage of a website into your ReadMe — even if you also have a website. If you don't have a website, your ReadMe will be the first thing many potential users see about your project, and your first opportunity to make an impression. But even if you do have a website, many visitors to your homepage won't bother to scroll down and will immediately click on your Docs or

GitHub icons. You still want those people to get the same message about what your project is, who it is for, and why it matters, and including that information as part of your ReadMe does precisely that.

UPDATE YOUR DOCS

Your ReadMe is not the only part of your docs you should update! Yes, you have technical, feature-focused how-tos in your docs, but docs are an important part of an open source project's marketing, just like a website is. As part of your positioning exercise, you should have identified the very specific situations in which your project absolutely excels. You need to make sure your docs clearly communicate what those situations are.

As part of the positioning exercise, you should also have mapped out what value each of your features delivers for users. This information belongs in your docs! Again, you want benefit-and-value information to be brief and not overshadow the 'how-to' nature of docs, but the information should also be there. Any section you have dedicated to a specific feature should start by clearly defining the outcome that a user can expect from using the feature, what situations the feature is and is not appropriate, and what value they'll get out of it.

EVALUATE YOUR PROJECT ROADMAP

Based on what you've learned from positioning your project, are there any things you can drop from your product roadmap? Or that you should give higher priority to?

You want to make sure that if you're spending your scarce development time on new functionality, it is delivering value that is differentiated and that reinforces the value that people are already getting from your project.

For example, if you've learned that users of your open source project tend to be extremely security conscious and choose your project because they believe it's the most secure option, you'd want to make sure that security is given the highest priority when you're developing new functionality. If there's ever a trade-off being speed or useability and security, you should make sure you prioritize security.

There are two product-related traps you can avoid with better positioning. One, the worst trap, is adding new functionality that detracts from your core differentiated value. For

example, if the thing your users like about your project is that it's simple, streamlined, and easy to use, each additional feature that clutters up the project could detract from that value.

Less bad, but still to be avoided, are features that just don't deliver value that your users care about. This results in wasted development resources and a longer wait for features that users *do* care about. It also ends up muddling your positioning by confusing users about what the goal of your project is.

EVALUATE YOUR EVANGELISM EFFORTS

Your project's positioning has major ramifications for how you are going to approach evangelism (you are doing evangelism, right? If you're serious enough about your project to spend time on positioning, you're serious enough to devote time to evangelism).

Most open source projects will evangelize their projects through social media, blog posts, webinars, and speaking engagements. You might have a website, and as the project gets larger you might also have user meetups. Revising and clarifying your positioning should cascade through all of these mediums.

First of all, you should make sure that everything that you're saying, whether it's on social media or in a blog post, is consistent with your positioning. Are you consistently describing your project as the market category you settled on during the positioning exercise, for example?

In addition to thinking about what you're talking about as part of evangelism efforts, think about where you are talking. You want to be where your ideal users already are. Go to the conferences they already go to; do guest appearances on the podcasts they already listen to. And these evangelism efforts are also ways to reinforce your positioning. If you want to be known as a security project, every time you appear on a podcast about security, or speak at a security conference, that reinforces the idea that your project is security-focused in the minds of your audience.

REVISIT PERIODICALLY

Positioning is not static. Plan to revisit your project's positioning periodically (though not continually).

Talk to users frequently. There are many reasons to stay in frequent contact with your users, and positioning is just one of them. However, you should dedicate time to having actual conversations with your users regularly, and one of the things you want to pay attention to during these conversations is how they use your project, how they perceive your project, and how well that perception matches your project's current positioning.

Pay attention to your competitors. New projects entering the ecosystem or suddenly gaining traction can be a sign that you need to reposition your project. If you suddenly find that your primary competitors are copying your positioning, for example, you have to find a way to alter yours that makes your unique value obvious.

Reevaluate even when nothing changes. Positioning should change, even if there are no big changes in the ecosystem you exist in. Your project will evolve, and technology will change, too. Reevaluating positioning doesn't have to be a time-consuming or expensive process, but merely a time when all of the maintainers review the positioning canvas you've established for the project and consider whether or not things have changed. If they have, it makes sense to start the process over again. As a general rule of thumb, if there are no major changes in the project capabilities or the surrounding ecosystem, it makes sense to reevaluate positioning about every 12 months or so.

Conclusion

If you and your fellow maintainers are serious about building an open source project that grows steadily, attracts a community of active users and involved contributors, and is sustainable over the long term, you should take the time to intentionally position your project. Relative to the amount of time you're already spending creating a technically sound project, positioning your project intentionally won't take long.

Positioning is the foundation for effective project roadmaps, community building, and evangelism. Spending the time to refine your project's positioning and record it will make you more focused and make your project more valuable.

About me: I'm Emily Omier, a positioning consultant who works with open source startups to accelerate revenue and community growth with killer positioning. If you can fill out the positioning canvas below and are happy with the results, you're good to go! If you can't fill out the positioning canvas to your satisfaction, or you continue having positioning issues (like stagnant growth, no differentiation from competitors, long sales cycles) after completing it, reach out at emily@emilyomier.com and we'll see if I can help.

Positioning Canvas

HOW TO USE THIS POSITIONING CANVAS

Every company needs a positioning canvas that's written down, so you can share the positioning with your team and make sure your positioning doesn't shift without anyone noticing.

Plan on completing one positioning canvas per product, and an open source project counts in this case as a product. If you have just one product, you only need one canvas. If you have two or more products, it is also a good idea to have a positioning canvas for your company that provides overarching positioning that can inform anything that is company-wide, such as your company's homepage or an investor pitch deck.

The tagline and market category are the first things you want to appear when you're done, but if you're filling this out as an exercise, they should be the LAST thing you fill out, once you've done everything else (unless you are already certain of your market category). In addition to that guidance, here are some instructions for each section:

Market category: This should be the answer to the question "What is your project/product/company?" It should fundamentally be a noun. However, you'll want to throw in some adjectives at the beginning, and might end with 'for XYZ type of workload' or 'for XYZ type of organization.'

Point of view: What opinions related to your project or product do you have, as founders? The points of view you enumerate here should be relevant to your product or open source project, they should *not* be universally held, and it should be clear that if somebody agrees with your opinion, your product/project is the clear best choice for them to use. The best points of view are differentiated, as in they are not held by your primary competitors.

Buyer persona: This is only relevant for commercial products. If money exchanges hands, who signs off on the deal? Even if you have a product-led growth, bottoms-up GTM strategy, you need to know who is signing off on deals and what they care about. Remember that this is not just about job titles, though that might be one of the characteristics. Think more holistically about the characteristics of the buyer: What are they responsible for? How do they make themselves look good? What do they care about? For example, if you have a security product there might be a buyer who knows their head is on the line if there's a major breach.

User persona: Who is interacting with the product/project daily? Who are your users? You might have multiple user personas, but for this exercise focus on the person who will get maximum value out of the project/product *but does not have buying power.*

Champion persona (optional). In some cases, you'll want to include a 'champion persona,' if the characteristics of internal champions are different enough from people who just use the project happily. For example, you might find that champions have some kind of team leader role but not buying power. Or you might find that champions don't have different job titles or job responsibilities from users, but agree with your point of view more strongly than others do, or have some other characteristics that make them different from your average users.

Characteristics of companies/organizations in your target market: Certain companies will get more value out of your project/product than others. What characteristics do they share? Examples here might be that they're subject to a certain regulatory framework, they're in a certain industry, they have specific HR issues, they have a specific technical set-up... industry vertical could also be one of the characteristics, but should not be the only one. A great example of this would be 'any organization running in an air-gapped environment.' These are mostly government organizations, but the critical characteristic isn't that they're governments, it's that because it's a government organization, their environment is air-gapped and that creates certain technical challenges that others don't experience.

Competitive alternatives: What do people do to accomplish the outcome you deliver, instead of using your product/project? This can be technology competitors, i.e., other open or closed-source software, but it could also be:

- Cut corners
- Hire some more people
- Get woken up at 3 a.m. to fix things more frequently
- Pay a fine
- Take longer to deliver their product

In many cases, the alternative people are comparing you to is not another technology product, but some combination of a manual process or living with frustration and/or worry. Also, if you have both an open source project and a commercial product, they might be competitive alternatives to each other. That's OK, but you should be aware of it and note it.

Pain points: What pain do those alternatives cause? Focus on the pain your users/customers experience from using the competitive alternatives above.

Purchase/search triggers: If you can uncover the specific triggers that caused your users or customers to take action after presumably living with the pain points above for a certain amount of time, include them here. These can be personal to an individual or more organization-wide triggers. For example:

- We were refactoring the application and realized we had to solve this problem now
- Our competitor had a (breach/outage/compliance issue) that was very embarrassing for them, and we want to make sure that doesn't happen to us
- I was hired as the first platform engineer on a new team

The point here is, that your users and customers were simply living with the pain points above when something happened that made finding a solution float to the top of their list of priorities. You want to figure out what that event was. *Sometimes, the 'trigger' is that they heard about your project/product and decided to use it. If that comes up frequently, include that here too, and make sure to note how they heard about you.

Unique values/unique attributes/proof points: You should have at least one and no more than five primary values that you provide to your users or customers. Start by listing out the value you provide, then list out the attributes that contribute to each value point, and finally list out proof points that provide outside verification that you actually do what you promise to. Values could be:

- Make it easier for team members to collaborate
- Eliminate the trade-off between security and speed, allowing customers to get both
- Make it possible to run data-heavy processes locally on edge devices

You want to map those values to the features your project or product has that make it possible to deliver the value. In this section, you'll usually have one value statement, and then a long list of features in the column to the right of it. Then the final column is how you can *prove* that you really provide that value. This is often a demo, an analyst report, or a customer quote — the point is, it can't just be 'take our word for it.'

Company Name: **Project/product name:** **Tagline:**
Market Category: (between 2 and 8 words)
Your point of view: ▪ Bullet point 1 ▪ Bullet point 2 ▪ Bullet point 3
Buyer persona: ▪ Bullet point 1 ▪ Bullet point 2 ▪ Bullet point 3
Champion persona (optional): ▪ Bullet point 1 ▪ Bullet point 2 ▪ Bullet point 3
User persona: ▪ Bullet point 1 ▪ Bullet point 2 ▪ Bullet point 3
Characteristics of companies in target market: ▪ Bullet point 1 ▪ Bullet point 2 ▪ Bullet point 3
Characteristics of workloads/applications ideally suited to your solution: ▪ Bullet point 1 ▪ Bullet point 2 ▪ Bullet point 3

Competitive Alternatives: ▪ Bullet point 1 ▪ Bullet point 2 ▪ Bullet point 3
Pain points you solve: ▪ Bullet point 1 ▪ Bullet point 2 ▪ Bullet point 3
Purchase/search triggers: ▪ Bullet point 1 ▪ Bullet point 2 ▪ Bullet point 3

Unique Value	**Unique Attributes**	**Proof points**
Unique value 1	▪ Bullet point 1 ▪ Bullet point 2 ▪ Bullet point 3	
Unique value 2	▪ Bullet point 1 ▪ Bullet point 2 ▪ Bullet point 3	
Unique value 3	▪ Bullet point 1 ▪ Bullet point 2 ▪ Bullet point 3	