B9Lab

The Definitive Guide to Web3 Developer Adoption

Executive summary

The capacity for blockchain organisations to drive developers to adopt their technology and build lasting products and careers is critical to the future of web3. If you are a project founder or active in any developer-facing role, then you will recognise the importance of getting skilled devs engaged with your technology and retaining their commitment. The central challenge is in identifying developers with the skills you need, and forming adoption strategies that account for varying degrees of familiarity with your tech and with web3 as a whole.

B9Lab have led the way in web3 developer adoption since 2015, both independently and in partnership with some of blockchain's most significant entities, including Ethereum, Hyperledger, Interchain, R3, and Tezos, as well as numerous developer-enabled projects. We have designed, created, and delivered self-paced and instructor-led learning systems, and have innovated highly effective methodologies for developer adoption.

We have distilled this knowledge into a modular framework called the **DevJAM stack**: a comprehensive system covering **Dev**eloper **J**ourney, **A**doption, and **M**anagement. It looks at learning portals and materials, documentation, and support tools; effective DevRel, and community and ecosystem management; the value of establishing academy platforms to generate sustainable growth for both your technology and the web3 environment; and when in an organisation's growth these strategies are best implemented.

Developers enter web3 with such different motivations, interests, and skill sets that any single label is a misleading illusion. They include pioneering innovators; entrepreneurial project leads; behind-the-scenes system administrators; and the numerous career coders needed to satisfy the increasing demand for skilled talent as web3 ecosystems scale up to compete or integrate with the structures of web2. Imagining that there is a one-size-fits-all solution to this challenge is clearly a mistake, but it is a mistake regularly made.

In this paper we outline strategies and tools for the effective practice of developer adoption derived from our many years of insight. They require adoption pathways focused on the needs of different types of developer with different degrees of web3 exposure, taking them to a place you and they both want to reach. By coordinating these journeys effectively, you can help any dev to embrace your technology, and reinforce their commitment to using it long-term.

If you want to know more, read on, or reach out to us directly at B9Lab.com.

About the authors

Elias Haase

Elias has spent the past 15 years planning, building, and running digital products. His background is in Artificial Intelligence. Since the early days of bitcoin, he has been following the progress of blockchain tech and gathered vast experience with smart contracts and trustless computation.

Elias is one of the Founders of B9lab and focuses on strategy, ecosystem outreach, and sales.

Lucy Meiland

Lucy has been client-facing for over 25 years and has a strong track record in marketing, sales, and alternative finance.

She has run multiple teams at small and large firms and ran her own company for eight years before joining B9Lab.

Currently, Lucy heads up Partnerships, where she finds the best ways to leverage B9Lab's knowledge and reach to help partners meet their goals.

About B9Lab

B9Lab is a global blockchain consultancy. We've been designing and building developer adoption strategies for leading web3 organisations since 2015. Through our onboarding and learning journeys, we've minted thousands of developers. Our team covers the whole spectrum of developer adoption strategy and management. We're developers, business strategists, marketers, and blockchain educators.

We have worked with a wide variety of clients and partners from different industries over the past eight years, including ecosystem stewards, institutions, and developer-enabled appchains.

Find out more at B9Lab.com.

Table of Contents

Introduction

Many organisations in the web3 space are developer-enabled. That means their tech is not aimed at consumers but at intermediaries who develop applications for end-users. This is the case for simple financial transactions, which need wallets for end users, as well as more complex solutions using smart contracts, decentralised storage, or other web3 technologies.

Naturally, developer-enabled projects rely on having developers adopt their technology: they are central to the project's growth and longevity. **Developers are the client.**

If your project is developer-enabled, the success of your project is *entirely* based on your ability to attract and retain developers. Effective developer adoption is central to your future growth, and must be a holistic activity shared across your entire project. If you don't follow a fully integrated approach, you risk underperformance and failure. But do it well and you will see sustainable growth in value and activity, meaningful applications, and loyal developers.

Developer adoption is not DevRel

Developer adoption is often not recognised as a cohesive approach, but instead is chopped up and devolved across marketing, sales, development, and DevRel teams. When decision makers hear "developer adoption", they think *developer relations* – and to them this just means sending a couple of people with merch to conferences, maybe setting up a stall, maybe giving a talk. If they think they're being really devoted, they have a DevRel engineer who is also responsible for documentation, support, and many other things that should require a whole team to do well.

This is not developer adoption.

Developer adoption starts with basic questions about the business model; it extends across target group analysis, metrics, documentation and content marketing, and developer-focused promotion and PR; and it includes enabling targeted grant programs, events, education, and developer support.

Our objective

This document is intended to help you navigate the world of developer adoption and understand what is needed to plan an effective strategy. We will explore the challenge of achieving effective developer adoption by directing your focus to four specific considerations:

- **The Adoption Journey**

  The steps needed to coordinate diverse developer-specific learning pathways that onboard targeted devs to your project based on their past experience and future needs.

- **The Developer Landscape**

  An analysis of several distinct "types" of developer, their unique needs and objectives, and how differing degrees of web3 familiarity will influence their adoption journeys.

- **The DevJAM Stack**

  A deep-dive into our developer adoption system, which examines broad strategies and approaches to learning materials and documentation, support tools and portal design, establishing a training academy focused on your technology, and approaches to thinking about DevRel, promotions, events, and incentives.

- **Applying DevJAM**

  A breakdown of where web3 organisations stand during their early, growth, and mature stages of development.

We believe that the true flourishing of web3 will be contingent on more than just the disruption of legacy technologies and the points of view that gave rise to them. It will require providing the inclusivity and support that can underlie a stable, nurturing, forward-looking developer culture.

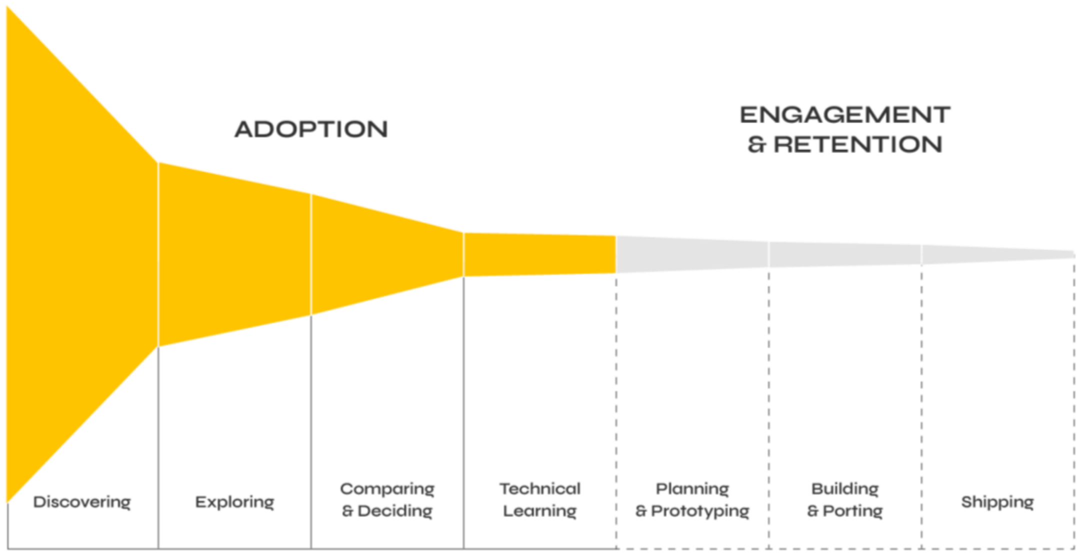
So let's dive in.

The Adoption Journey – guiding your developers towards building

"Adoption journey" is shorthand for creating a path for developers that leads to the destination you want them to reach. If you're at the early stage of your project's lifecycle, you could be looking for developers who are inspired to try creating something on your testnet. Later on, you might want to populate the teams building with your tech.

This isn't a manipulation of developers to get them to do what you want; quite the opposite. You can be very clear about what your goals are. Trying to discern the destination is difficult for new developers in web3; having it clearly spelled out is a very refreshing change. Allowing developers to get on board with the journey ensures your goals are entirely aligned. As the authority in the tech, you are plotting the journey to make it easy for you both to achieve a shared goal.

The full map of adoption can encompass many journeys, with many destinations. Each journey represents an individual developer's path. This conglomeration of distinct individual journeys is the developer funnel. You can have great fun creating complex maps of interacting journeys that allow for multiple decisions along the way and result in various outcomes – each of which may even be part of multiple funnels! However, even if you do this, each developer's journey will ultimately include a few common core elements:



This paper covers the adoption portion of the developer funnel. The greyed-out area is when the building starts, and we will cover that in a later paper. We're focusing on adoption here, which is about discovering, exploring, comparing and deciding, and technical learning. This diagram crudely represents the volume of individuals – you'll never convert everybody discovering web3, and probably wouldn't want to. Some will self-select out, freeing you to help those who want in. Let's take a look at each phase in more detail.

Discovering

In this phase you're reaching out to your target developers where they are. This is the purview of your promotions, marketing, events, and PR teams, with backup from your technical content teams and DevRels.

Many projects target only one type of developer: the *experienced web3 pioneer developer*, who is certainly a blockchain expert and may also know something about your stack – what we call an **emerging insider**. Superficially, this makes sense: it's easy to find emerging insiders, they're probably already on your discord or following you on Twitter; they're a receptive audience, and it's stimulating for you because they already know the basics. Blockchain experts can also be easily targeted: there's no need to do more than a bit of sponsorship, some YouTube interviews, paying a PR firm to get you some nice profiles on Cointelegraph, etc.

However, because this kind of targeting is easy, everyone does it. The cacophony of projects screaming for the attention of experts and insiders is overwhelming. It's hard to differentiate your message over the noise. We've all been in groups on Telegram, so we know what that's like.

In fact, a much richer vein of future talent are the **curious outsiders**, people interested in web3 but who have yet to start their learning journey. They are actively looking to discover blockchain technology, they just need to be guided towards your part of it. They may face a slightly longer journey to adoption, but they are many – and they can bring with them a wide breadth and depth of experience in general, which is beneficial. The structured journey for a curious outsider will naturally be longer than the structured journey for an insider. However, both of these are significantly shorter than an *unstructured* journey for insiders, which many of them face today.

Curious outsiders could be anywhere, so reaching out to them may seem difficult – but don't forget: *they are curious!* Segmenting by developer type is crucial here, as is focusing on your business plan. For example, if you're at the early stage of building a decentralised cloud solution, your obvious targets are developers already working in the web2 cloud space. They know the drawbacks of web2 and as curious outsiders will be interested in your solution. Since your project is in its early stages, pioneer developer types are ideal targets, so you should devise messaging that targets that subset of the larger group. Suddenly, it's not so difficult.

In the Discovering phase, targets respond to educational content marketing, so aim at answering the questions they are asking without too much insider jargon. At the top of the funnel (TOFU), devs are curious to know how they can learn more about blockchain, about the applications of your tech and its various use cases. Highlighting non-technical incentives works well too. They want basic information to assess if it's worth continuing on the journey, or if they should be exploring other options; even if your stack is the perfect choice for them, the wrong messaging at this point can easily drive them away and into the arms of your competition.

Exploring

When targets begin exploring, you can get to work informing them in detail. You have already captured their interest by clearly explaining the basics and the benefits of your project, now you start answering more complex questions. It is here that your developers will start interacting with people in your space. Expect them to join some sort of forum or community group (likely that's on Discord and Telegram) and maybe start asking questions through your website chat, or even emailing you directly or going to a meetup. You may also find them posting questions on Reddit.

We have analysed many client Discords that are open to developer questions. In almost every case, around 65% of interactions are one-time interactions (ie, a single question is asked and answered, with no further interaction in the next three months). Or worse, a question is asked *with no answer*. This isn't just the problem of their community team, this is also because of their website and generally available materials; they're not addressing target groups correctly, or are funnelling them in the wrong direction. Devs (and non-devs) are going to these interactive spaces for answers to discovery questions, which results in a very messy journey; and often, for the wrong reasons, a very short one.

Correctly funnelled, the exploring phase is where your community team can shine. Most teams have no problem answering basic technical questions about their product. They also have little problem directing technical questions up the chain if they can't be answered immediately. Where most projects fail is in understanding the reasons behind non-technical questions from technical people, and in stimulating further interactions. Your community team should be able to triage developer types onto distinct journeys. Helping your community team identify your target devs based on the questions they ask is an important part of your developer adoption strategy.

Here is a list of non-technical questions that developers may be asking in the exploratory phase. Your website, content marketing, community team, promotions, and DevRels should be able to use them to identify each developer's type and direct them to the next step on their journey (we will be discussing these developer types in the next section, **the Developer Landscape**).

- Are there a lot of jobs available and what sort of jobs are they?
  – Career dev
- If I build something with it, is there a ready market for my product?
  – Entrepreneur dev
- Are there developers available for me to hire if I migrate my project to this tech?
  – Enterprise adopter
- Is it easy for me to learn how to use this?
  – Career dev, entrepreneur dev, enterprise adopter
- Is it easy to get my team onboarded?
  – Enterprise adopter

- What are the existing use cases for this tech?
  – Entrepreneur dev, enterprise adopter
- Can my project benefit from the other projects using this tech?
  – Entrepreneur dev, enterprise adopter
- Are there examples of successful applications of this tech by others like me?
  – Enterprise dev
- Can I get a grant or raise funding to dedicate myself to my project?
  – Pioneer dev, entrepreneur dev
- Is there support if I run into trouble?
  – All devs!
- How much will this cost in time, effort, and money?
  – All devs!

There are probably hundreds of possible questions, but these key ones are a good start.

Managing your target devs through the exploring phase doesn't mean resorting to the dark arts! It is simply a case of anticipating each developer's pain point and helping them deal with it.

Comparing & Deciding

Your tech does not exist in a vacuum. Devs aren't just assessing your tech, they're comparing it with tech from other sources. There is a very important point here: it doesn't matter whether *you* consider something as comparable, it only matters what *they* see as comparable. At this phase of their journey, they are not as informed as you – but if you dismiss their notions of comparable tech you're not helping them choose your solution, you're alienating them.

This is a problem we've seen hundreds of times from projects that get annoyed with being compared to what they see as "inferior" tech. Frankly, this is *their* mistake, not the dev's. They haven't delineated the functionality of their tech in a way that speaks effectively to their targets.

When it gets to this phase, for better or worse you have little sway over a potential dev's actions: now it's entirely down to them. If you have built effective adoption journeys, target devs will be able to properly compare and make informed decisions on which tech to use. You've delineated your tech in their minds, and addressed the fundamental non-technical questions they're asking.

Developers will happily adopt tech that isn't a perfect fit, *if* their non-technical questions are well answered – that clarity is simply more valuable to them at this stage. They know they will need to solve tech problems; that's their passion, and it's rewarding. Their passion is not sifting through outdated tutorials or repeating the same questions on a dead Discord. Imagining such roadblocks on their path to success will be more than enough to drive them elsewhere.

Technical Learning

At last we reach the middle of the funnel (MOFU) in the adoption journey. For an adopter to get to the Technical Learning phase, they have decided your tech is the one for them. And that's great! However, they still have a way to go – if they feel friction here they will still abandon you, because it is only now that they really start to spend their most valuable asset: *time*.

Technical Learning is the culmination of your outreach efforts to your target devs, and their stepping-stone towards becoming Stack Insiders; it is also a deep and complex topic in its own right. We will cover all the main aspects in the sections *Learning Materials and Documentation*, *Support Tools and Portal*, and *Academy* when we get to **the DevJAM Stack**.

The Developer Landscape – targeting the right developer at the right time

Developers are not a homogenous blob of talent that materialises at the moment you need them. At different times in the evolution of your project you will need to attract and nurture different types of developers. Many decision makers have subconscious biases around what kind of developer they want to attract, largely informed by their own journey.

If you only have one takeaway from this paper, make it this: **your project is unlikely to need more developers exactly like you**. It needs different types of developers, and they are attracted by different things. Targeting the right type of developers at the right stage of your project is the key to your success. What you start with, you can't scale with.

The Pioneer Developer

The most over-targeted developers are the early-stage **pioneer developers**. These are risk-takers with advanced tech skills and some team-management abilities. They often form the backbone of a new project. Most of the founders of large projects and ecosystems are pioneer developers themselves: they had the vision and ability to build the tech and be the first movers to build *on* the tech. These are the devs you want working with your testnet, sandbox, or beta.

Pioneer developers are mostly not deterred by weak or absent documentation, buggy code, and unclear lines of communication, because they see these as signs of a project where they have first-mover advantage. But even though pioneer developers can deal with imperfect onboarding journeys, their experience and success can still be optimised to the benefit of the ecosystem.

The Entrepreneur Developer

There is also a much larger group of late-stage **entrepreneur developers** that are often overlooked, or at best mistargeted as pioneer devs. This group consists of the entrepreneurs who build *using* the tech. They flex their entrepreneurship through the ideas they have, and they are led by their ideas rather than the potential of your tech specifically. They are looking for an existing solution that they can work with to realise their projects, and have a wider set of non-technical questions they also need answered.

Entrepreneur devs are made happy when all their non-technical questions are answered well. Satisfying this concern is more valuable to them in their discovering and exploring stages than identifying the *perfect* technological match for their plan. Entrepreneur devs are highly sensitive to friction and will shift to adopt another tech that will cause them less, even if its functionality does not fit as well. Ecosystems are won and lost through targeting this developer type.

However, hiring both pioneer devs or entrepreneur devs to populate your production teams can be a very costly and time consuming mistake. If your core development team consists solely of pioneer and entrepreneur devs, you will be paying high wages to attract them in the first place –

enough to make them neglect their entrepreneurial drive. This inevitably leads to high turnover as their repressed impulses kick back in, they neglect your project, and leave to start their own.

Pioneer developers and entrepreneur developers see big pictures and get things moving, but to implement their visions and *keep* things moving you need devs with a different kind of skill set: we call them *career developers*.

The Career Developer

In contrast to the over-targeting of early-stage pioneer developers, the career developer is often forgotten and undervalued. However, career devs form the foundation of any ecosystem that can keep pace with growth demands. They populate the teams that build the tech, and help the ventures founded by pioneer developers and entrepreneur developers to scale. They are also much more risk-averse: their aim is to land a rewarding role with a financial upside; in return they supercharge the growth of both projects and ecosystems, and can often evolve into loyal open-source contributors.

Career devs need to know they aren't wasting their most valuable resource when learning your tech: *their time*. Career devs get paid for their time and their skills, so they want to see a quick path to enhancing their skill set and proving to employers what they offer. This means they need access to up-to-date training materials without fluff or deadwood, access to support when they get stuck, and some form of certification to validate their efforts and signal their value. They also need to see a thriving ecosystem where they can find work. Skills are nothing without jobs.

Neglecting the career developer type is pandemic to web3. If you only court entrepreneurs and pioneers, you may find yourself with a lot of very clever ideas and no-one at all to execute them.

The Sysadmin

Early-stage web3 ecosystems massively benefit from early-adopting sysadmins experimenting with running nodes and validators. As an ecosystem matures, these early sysadmins often start validator or service businesses. For these reasons, it is also important to keep growing the sysadmin community to support the growth of other businesses in the ecosystem.

As a general rule, sysadmins are under-targeted across most of tech, but particularly in web3. This is surprising, as web3 is evolving rapidly and hitting many obstacles along the way. Web3 projects are notorious for unreliable release pipelines, pressurised deployments, and panicked maintenance. A focus on retraining expert web2 sysadmins would benefit the entire industry. More importantly, projects that are well-staffed with sysadmins at this stage in web3's growth will provide them with a significant advantage. This may be most evident when it comes to issues of enterprise adoption, where predictability and reliability are seen as key signifiers of longevity.

The Enterprise Adopter

While this isn't so much a developer type as it is a *client* type, enterprise adopters may well be central to the growth of any developer-enabled project. These adopters are mostly looked after by your growth team; however, there is still a specific developer-focused adoption journey that needs to take place to help decision makers choose your tech.

Enterprise adopters are either looking to migrate their existing web3 project to your tech, or they are looking to incorporate web3 tech into their web2 project. Their unifying feature is that they already have an existing project, plus the funding and infrastructure to support it. If you are already addressing entrepreneurial, career, and sysadmin developers using targeted strategies, they will find it easy and attractive to adopt your tech, as their needs will span all three groups.

Clearly, the key personas you will be addressing are the CTO, architect, and other technical decision makers, but you also need to address supplementary questions that are non-technical. You need to demonstrate that there are developers available to hire who are already proficient in your tech. You need to show them that the career developers they already employ will be able to skill up quickly and efficiently. You also need to show you can provide support if their devs get stuck, and that your materials are fit for purpose, up-to-date, and relevant to their use cases.

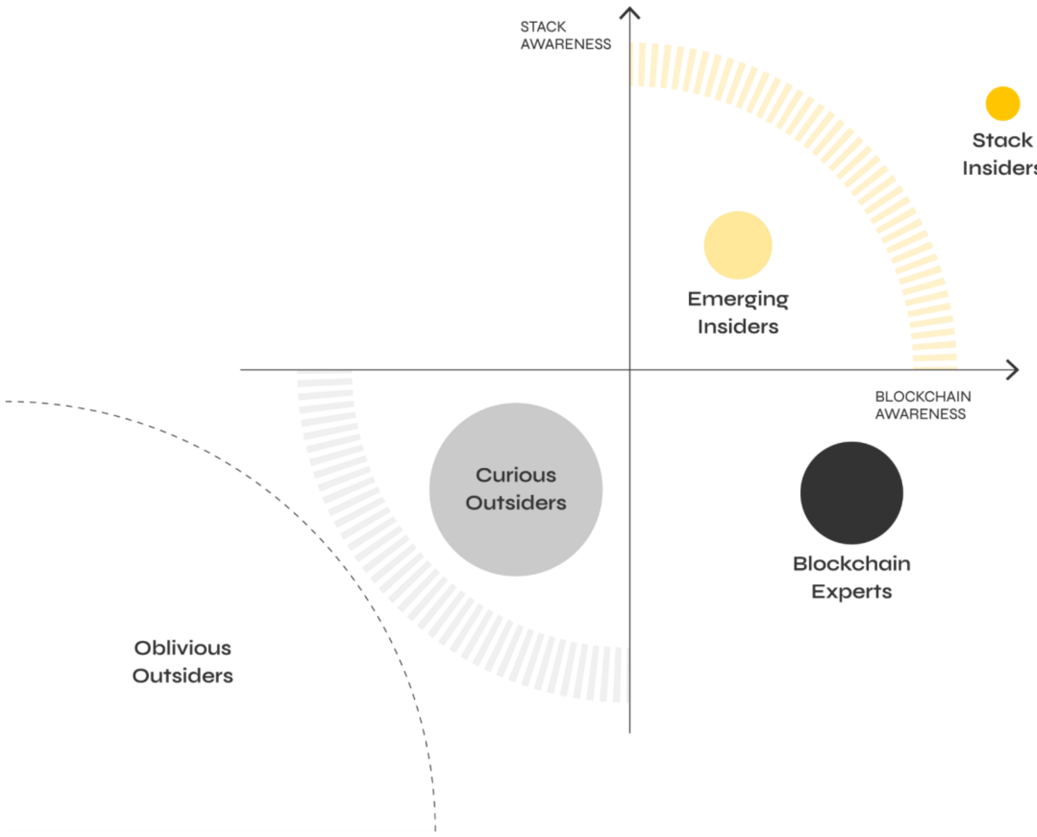
For this target group you should provide non-technical content that enables the CTO and other technical decision makers to champion your tech solution to their non-technical counterparts. These non-technical decision makers are just as important, and they also have a journey.

Targeting experience and awareness

The talent gap in web3 is a hot topic. Dev scarcity naturally increases salary costs, recruitment costs, and churn. This is all a natural consequence of projects prioritising web3 insiders for the majority of their communication and outreach efforts. DevRel and marketing teams spend most of their time on Twitter spaces, Discord, and arranging hackathons in ways that only target the people already in web3. These can be useful channels, but they don't consider outsider devs who are curious about the tech, nor raise awareness that a career in blockchain is even viable.

Of course, you should be supporting and nurturing devs who are aware of or already using your tech. They may play an active role in your community and come to be major ambassadors for your stack, which is great. But you can't scale if you focus a disproportionate amount of time and resources on this one group. You have to pay attention to devs outside that circle who, with the right encouragement and developer adoption strategy, could become valuable insiders too.

All of the developer personas we've defined – *pioneer devs, entrepreneur devs, career devs, sysadmins,* and *enterprise adopters* – can also be categorised by their **web3 experience** and **awareness of your tech**. We segment them into five groups based on these two factors:



1) Your **Stack Insiders** are in the top right quadrant. They have experience building with blockchain technology and they have high awareness of your ecosystem or tech. They've already bought into what you do.

2) In the bottom right quadrant are the devs who have extensive blockchain experience but no or little awareness of your tech specifically. These are the **Blockchain Experts**.

3) Next are the **Emerging Insiders**. These are devs who have slightly less experience building with web3 technology, or are not as familiar with your tech and ecosystem as the other insiders. They are keen to learn more and contribute to the ecosystem.

4) The bottom left quadrant is where it gets interesting. Here we first see the **Curious Outsiders**, web2 devs with little or no blockchain experience but who want to explore this technology further. They may have no idea you exist, but they're highly motivated.

5) Lastly come the **Oblivious Outsiders**: web2 devs who don't even realise a career in blockchain is a possibility, and likely have never heard of you at all. As you can see, they are the biggest segment by far – so big that there's no way to visualise them completely!

To give an idea of the magnitude of difference between the groups, the total blockchain expert group (of which your stack insiders are a subset) has been variously estimated at from 23,000 to 120,000 devs. Electric Capital's 2022 report erred on the low end, as they used *making regular open-source contributions* as a metric. A range of high-end figures can be found in blogs online; take from that what you will.

However, even these high estimates pale into insignificance next to the number of professional developers in the world. When you factor in those working in web2, most surveys and articles (by variously reputable sources) assume a global dev population approaching thirty *million*.

At most, existing blockchain experts represent just 0.4% of the total addressable market.

It may seem too obvious to mention, but you want as many active, loyal, and qualified devs using your tech as possible. Neglecting 99.6% of possible adopters is a strange way to achieve that, but it is apparently a very common strategy of web3 recruitment.

A sustainable adoption strategy focuses on *nurturing* curious outsiders, and builds journeys for them to become stack insiders, devotees and evangelists of your tech. An ambitious adoption strategy devotes resources to *creating* curious outsiders as well. This is how you scale.

Basic principles of interaction

We will say it again: *Developer adoption isn't DevRel*. Nevertheless, relationships themselves are vital, and the decisions you make regarding your interactions with devs can have a profound impact on the longevity of your project. This is just as true if you go to great lengths to formulate a developer adoption strategy as it is if you made no effort at all.

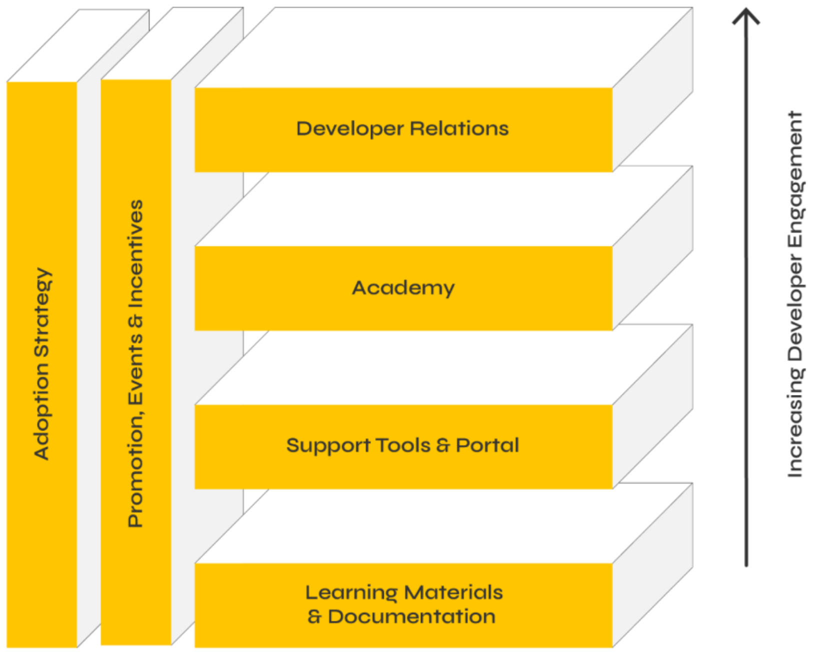
Devs are your lifeblood and time is their most valuable asset. Treat it accordingly. Make it easy for them to understand whether your solution works for their project. Make it easy for them to learn your stack. Provide troubleshooting support. Help pioneer and entrepreneur devs with grants and paths to funding, and help career devs with job placement support and certification.

Devs are also allergic to marketing speak and fluffy language. Honest straight talk goes a very long way. Identify their pain points and the questions they need answered, and reply directly and clearly. Expect to repeat yourself multiple times, because developers will access your tech in different ways and at different stages of their knowledge acquisition. The worst thing you can do to a learning developer is dismiss their pain points or the questions to which they need answers.

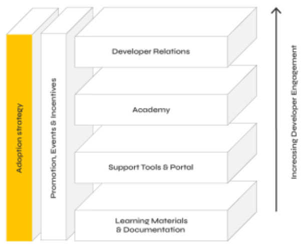
The DevJAM stack – all the elements of great adoption

After eight years of building adoption journeys, targeting all of the above developer types and at various stages of a project's development, we've distilled our knowledge into the **DevJAM stack** – for **Dev**eloper **J**ourney, **A**doption, and **M**anagement. It's a modular framework for growing the skills of individual developers and growing engaged developer communities around your tech.

Each developer journey is unique, but you can establish clear routes that developers will want to follow. Successful developer adoption is contingent on accessibility and ease of comprehension, key factors you can control. The DevJAM framework allows you to direct each new developer's journey and get various parts of your organisation working together to drive developer adoption.


Adoption Strategy
Promotion, Events & Incentives
Developer Relations
Academy
Support Tools & Portal
Learning Materials & Documentation
Increasing Developer Engagement

Adoption strategy



Your adoption strategy is the translation of your overall business goals into your developer goals. Your adoption strategy defines all the other parts of the stack, and it should set out the measurements that will help you iterate and define success. The keys to a good adoption strategy are not all that different from good strategic planning as a whole:

1) **Adoption strategy is not an event** – it is an ongoing process that needs refining monthly, quarterly, and annually to check you are meeting targets, that your assumptions still stand, and whether the changing environment requires you to change goals. Without this attitude, an adoption strategy can be left on the metaphorical shelf and is never truly implemented, measured, or iterated.

2) **Adoption strategy must be aligned with a project's overall strategy, priorities, and goals**. Without this approach, plans lack useful strategic thinking. This means you need a cohesive business strategy, and if or when this changes it must be communicated and reflected in the adoption strategy. The business strategy must include the following:

    a) **An external review** that assesses your industry, the competition, regulation, and market trends. This should give you an idea of the external forces that can affect adoption, and allows you to see flags that may force you to change strategy (eg *Will any upcoming legislation dictate which countries you can hire talent from?*).

    b) **An internal review** of your organisation (typically a SWOT analysis), so that the adoption strategy can focus on rectifying the weaknesses and complementing the strengths of the project.

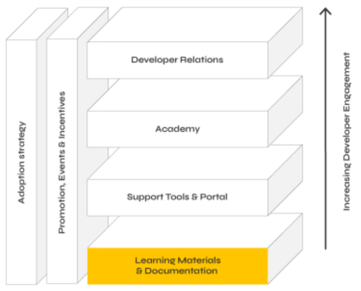
    c) **Project goals and priorities**. Your adoption strategy cannot align with either of these factors if they're not clearly identified and understood.

3) **Adoption strategy defines functional objectives and key initiatives for the period** (normally a year). For example, if your testnet is planning to launch in six months, *how many projects and individuals would you like to be experimenting with it, and by when? What does successful experimentation look like? Where can you find the developers to participate? How many must you target to successfully interest one developer? What do they need to know to be able to experiment? What will incentivise them to experiment on your testnet?* By answering those questions, you've defined the end goal – what success looks like – and begun to plan the journey to get your developers to that end goal. Now your teams must work together to flesh out their respective parts of the strategy: factors like marketing, growth, events, DevRel, website, PR, development, and education.

4) **Adoption strategy determines budgets, financing, and staffing needs**. Surprisingly, many projects still don't budget or plan their staffing needs when it comes to adoption strategy. Often this is because they split adoption strategy across disparate departments, as discussed previously. But equally, many projects neglect budgeting and planning for almost all functions that aren't core development. Most founders have a development background, so they know that discipline very intimately and may prize that over all other functions. This is a very common mistake. Failing to plan, budget, and staff developer adoption is like building a stadium and expecting players and fans to magically appear.

5) **Adoption strategy identifies and tracks success metrics monthly and quarterly**. Tracking progress on all strategic goals and objectives on a regular basis is key to ensuring a plan is being implemented well, and allows you to iterate or change course if needed. The discipline to make progress and regularly report on success metrics ensures accountability and follow-through. It's advisable to assign a person to collecting, tracking, and reporting progress on the strategic plan. A quarterly review should include a status report on adoption strategy implementation through KPIs or OKRs.

Learning Materials & Documentation


Adoption strategy
Promotion, Events & Incentives
Developer Relations
Academy
Support Tools & Portal
Learning Materials & Documentation
Increasing Developer Engagement

Providing clear, up-to-date educational content is the bedrock of ecosystem growth. Building foundational learning materials and documentation is a rare and discrete skill, yet it is a common practice to assign this work to busy core devs or juniors in training. However, in this situation your writers are either too busy or too inexperienced to be effective and often see the writing of materials as a chore, or at least a non-core activity. By making this choice, you stifle the learning and innovation of your next generation of developers – the key to your growth – before they even come in the door.

As you build your learning materials and documentation, you need to take into account three types of learners and what they are using them for:

- **Curious outsiders** need an entry point that takes them from not understanding your stack at all to being able to deploy production applications.
- **Emerging insiders** know the basics or more, and are looking to improve their skills and learn certain aspects of your stack.
- **Stack insiders** are looking for how a very specific part of your stack works to solve a defined problem they have.

Your learning materials and documentation should be designed to stand alone without the support of an expert as a guide.

Self-paced learning

The most effective way to serve the needs of the above three learning types is to create a single cumulative learning path that provides materials from entry level all the way through to advanced concepts and production deployment, ideally alongside an example project that provides devs with a progressive demonstration of techniques and solutions. Self-paced learning can be later supported with an instructor-led offering; see the **Academy** section for a detailed discussion.

Information is divided into individual steps within a didactical structure, grouped by concept into small, digestible units that can be addressed by the learners in a modular and flexible way.

This approach means that devs can always validate their progress through an established learning system, or as they gain experience can choose to jump to a specific topic, check out the cumulative code of the previous sections, and get started. By offering these resources within a coherent shared learning environment, the potential for today's beginners to steadily evolve into tomorrow's problem solvers is baked in from the outset.

Passive developer support

Your own documentation, knowledge base, and learning materials, plus content on third-party platforms like Stack Overflow that focuses on your tech, represent your passive support offering. These are usually the first port of call for a developer trying to *troubleshoot*, but they are not a guided learning journey. The more comprehensive your passive support offering, the less your active support will be burdened with repetitive and resource-draining questions.

Documentation

Many organisations confuse documentation and learning material. Your documentation should be a reference lookup that explains how specific functions or API calls work, for example. It's not a tutorial or guided learning path, it is a trusted reference for when a developer needs to find out how to correctly use a specific thing.

Knowledge base

As you answer support questions and are collecting best practices within your wider team, you should establish a place where this information can be collected, edited, and shared publicly.

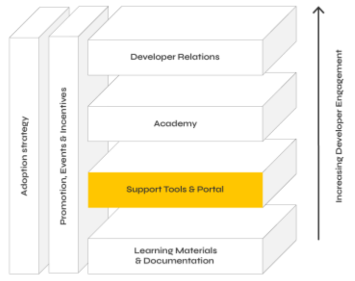
Learning material

Your learning materials are outside of the holistic, cumulative learning journeys described above; you need a space for smaller pieces of content that keep pace with the bleeding edge of the development of your codebase. Think of tutorials and other technical explainers as an incubator for your holistic course. A tutorial space allows engineers to quickly explain how something works. This can then over time be integrated into the holistic course.

Third-party platforms

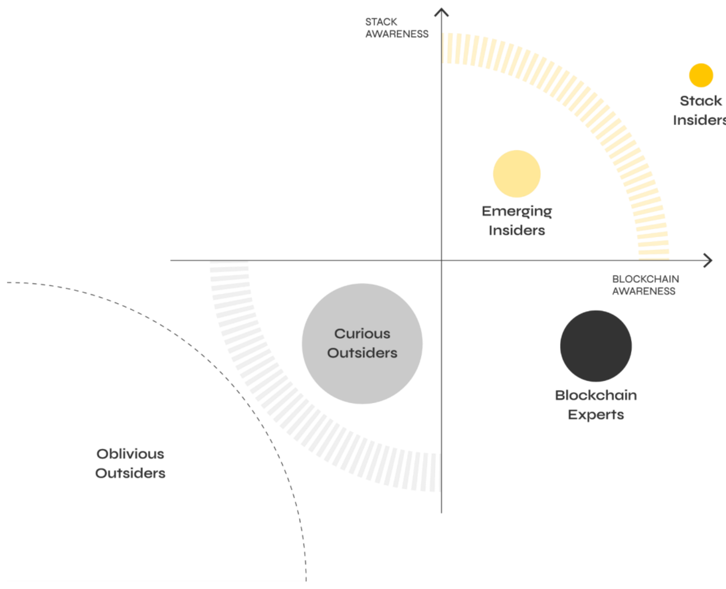
This is one of the most undervalued and overlooked tools in your toolbox. When developers get stuck, they google their issue. Stack Overflow and other developer communities usually top the search results, so use their expensive SEO to your advantage: dedicate part of your DevRel engineer's time to trawling Stack Overflow, Reddit, and other communities and then provide direct support wherever it is being looked for. This is doubly effective, in that it provides active support to a developer in need while also adding to an easily discoverable knowledge base. It also reflects positively on your engagement with your community of developers: no bad thing.

Support Tools & Portal


Adoption strategy
Promotion, Events & Incentives
Developer Relations
Academy
Support Tools & Portal
Learning Materials & Documentation
Increasing Developer Engagement

A learning portal is an easy-to-access, verifiable source of truth in a potentially confusing universe. It provides a place to house content for self-directed learners so they can brush up on valuable skills, and is the first port of call if troubleshooting. It is the best way to serve devs at all stages of experience. New devs invest time in your tech, and rewarding them with support tools that allow for fast, accessible responses is proven to strengthen their satisfaction and commitment.

Your learning portal is not the same as your documentation portal. Many projects utilise a combined documentation and learning portal, and deliver far less effective learning journeys because of it. When your aim is to onboard new talent, a clear learning path is essential. If you have a documentation portal too, that's great; but it should be as a separate reference tool only.



Who is your portal supporting, and how?

It's also important to consider the distinct needs of curious outsiders, emerging insiders, and stack insiders.

Curious outsiders

It's true, there's lots of entry level information freely available across the internet, and it will remain valid for most protocols and projects. Superficially, there seems little point in recreating it on your learning portal. However, it is worth considering who is providing this information, and why. Material created by your competitors will naturally direct potential developers towards their tech, not yours. It doesn't matter if their tech isn't as good, or if it does slightly different things – devs learning through such materials are much less likely to discover yours once they're onboarded by your competitor.

Even if onboarding paths have been created by well-meaning and entirely impartial community members, relying on them gives you little control over that part of your funnel, nor the quality and longevity of their information. You are also training your community to believe that true, verified information might come from different brands and people, which can become an attack vector.

You don't necessarily need to recreate all these materials, but you do need to at least curate a path that allows a dev to transition from having minimal blockchain knowledge to being a fully fledged, devoted member of your team. That can involve sponsoring content, commissioning it from existing creators, or even engaging in some sneaky guerilla tactics! And, of course, it *can* mean creating content for yourself. As you and your ecosystem grow, content aimed at beginners should grow with you to widen the intake of your developer funnel. Neglecting to offer entry level information cuts your access to new developer talent drastically as they will be actively funnelled elsewhere.

After you capture a beginner dev's attention through in-bound marketing and event outreach, you need somewhere to direct them. Paths are very standardised at this stage, you should not need the troubleshooting skills of your DevRel team to support an initial learning journey; they will just need to point curious outsiders in the right direction. However, if you don't have beginner materials on your portal, the task of helping new devs get up to speed will fall on your DevRel team through your support channels. Alternatively, curious outsiders will look beyond your ecosystem for what they need, and you will have wasted your effort regarding in-bound marketing and event outreach.

Catering to the needs of those unfamiliar with web3 may seem like an unasked-for burden, but there are real benefits to bringing in blockchain beginners. When you train someone with excellent web2 experience but little knowledge of web3, they can typically demand a much lower starting salary than someone with existing web3 experience but perhaps fewer skills that are applicable to your project. You also capture them at a potentially very confusing time in their learning journey, instantly solving one of their largest problems. The relief of well-structured, easily accessible, and reliable materials at this stage will fast track their loyalty to your project and your brand.

Beginner-level information is central to capturing and retaining developer interest. If you aren't doing the work to provide it to those who need it most, someone else is likely reaping the benefits.

Emerging insiders

Your emerging insiders want to skill up on a specific aspect of your tech. For sure, they could dig about in the documentation, search through a bunch of repos, and hopefully get the answers they're looking for, but that's very inefficient – particularly if you have multiple devs on the same journey.

Emerging insiders are looking for one of two things:

1. **A way to specialise, to continue their learning by focusing on a specific role or aspect of your tech.** This is an easy request to fill. If you have a series of certifications, their journey is very clear; if not, the most effective way to promote specialisation content is to make sure onward journeys are specifically signposted from beginner-level learning materials. Also, use developer-focused content marketing and promotions, events, and active outreach.

2. **A quick way to get updated on anything that has happened recently in your tech that's going to affect their role or plans.** Address them using tutorials, quick update videos, and dynamically evolving content. This needs to be supported by your DevRels, with a strong element of outreach through content marketing and targeted updates. Create dedicated email lists for your various personas. Letting devs self-select for updates and content saves you much more energy than a scattergun approach (and it really cleans up your Twitter).

Stack insiders (and blockchain experts)

A big reason developers abandon adoption journeys is a lack of troubleshooting support. Bugs and a lack of information are to be expected with early-stage, fast-moving codebases, and are often positively embraced by pioneer developers. But often if a dev has to spend too much time troubleshooting an issue, they are likely to give up on your solution. Entrepreneur devs and Career devs are particularly sensitive to this kind of friction in their learning journey. They are not as dedicated to your tech as the early-stage adopters, and will assess their journey with you by comparison to those provided by competitors, and with their own expectations of service.

Providing access to quick and easily accessible basic support through a learning portal is essential, and should be clearly signposted across all interactions with devs. Showing devs that their questions can be easily answered without having to engage with support teams will be much appreciated by learners (and by your support teams!).

Support tools

Providing the infrastructure for support is essential. It is common practice to provide support through Discord. Discord has realised this is one of its use cases and so it is making significant improvements in this direction. The structure of your Discord channels can also facilitate support and they should clearly signpost which are for general chat and which are for technical support.

You will also find that, without adequate signposting pre-enrollment, your developer channels get silted up with coin noise and spam. If you are going to use Discord for developer support, you need to direct general crypto chatter to another venue, say Telegram.

Using bots

An answer bot in Discord can be useful, but only as far as your teams are prepared to dedicate time to updating the bot's auto-response bank and iterating on existing answers. A badly maintained bot is worse than no bot at all.

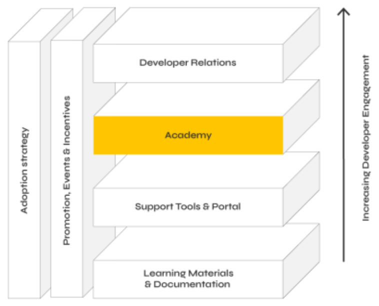
We always recommend implementing a formal ticketing system for support. This helps question triage, lets you track activity types, and gives you valuable information about your community and devs. It allows you to track success and acts as a verifiable bank of questions and concerns coming directly from your community. You can use it to decide which content to prioritise for updates, and see when demand for knowledge indicates new content needs to be created.

Tracking tools

Without data, it is difficult to see whether you have holes in your support and where you can improve. Having interactive dashboards that track a variety of metrics over varying timeframes is essential. The basics should include:

- Number of questions
- Number of questions answered
- Number of questions not answered
- Average time to answer questions
- Sentiment analysis
- Average number of support tickets
- Questions on specific topics
- Individual level engagement
- Times of day, week, and month that questions are asked

Academy – supported learning


Adoption strategy
Promotion, Events & Incentives
Developer Relations
Academy
Support Tools & Portal
Learning Materials & Documentation
Increasing Developer Engagement

Having an academy to train your target developers is the prime way to acquire accredited talent at scale, and one of the lowest-cost methods of doing so. It also allows you to curate the developer types coming in and pick other skills that they will bring to your ecosystem. When used correctly, an academy builds a loyal community around your technology and helps to create a strong brand presence.

If you want to attract career developers, offering an academy is a very strong incentive. A clear, timeboxed learning program allows them to easily assess what they will need to do to become proficient. Simultaneously, the cohort structure creates a platform for devs to interact, which can lead to more innovative products and solutions and thus be a strong pull for entrepreneur devs.

The benefits of an academy are five-fold for the developer:

1. **Access to expert guidance:** Academies enable contact with experienced developers and mentors who can provide learners with targeted guidance and advice throughout the program. This is invaluable for developers who are looking to upskill or cross-skill.
2. **Structured learning paths:** Academies provide a tailored curriculum that covers the latest upgrades and industry best practices. This structured learning path helps developers quickly learn and apply the skills they need to succeed in the industry.
3. **Network expansion:** Developer academies also unlock access to a network of other devs, allowing learners to collaborate and benefit from each other's experiences. This can be invaluable for developers looking to expand their network and build relationships.
4. **Job opportunities**: By completing an academy program, devs demonstrate to potential employers that they have the skills and knowledge to excel at their job. This can open up more job opportunities for developers and make them more competitive.
5. **Problem-solving skills:** Academies also offer developers the opportunity to learn how to think critically and solve real-world problems in a safe and guided environment. This is particularly valuable in web3.

To operate an effective academy you need four basic phases:

1. **Screening** – You should actively target your chosen developer types and invite them to complete a timed screening test that assesses their ability as coders. This test should not require *blockchain* experience, to make sure you do not alienate curious outsiders. Framing this activity either as a competition to "win" a place on the course or as a screening test to access free training are both approaches that have proven effective.

   The timing for this phase should not be longer than six weeks in order to maintain developer interest while maximising the number of applications. If you wish to further curate your developer intake, you can also have interview rounds and ask developers to submit additional information such as resumes, portfolios, etc.

2. **The course** – We have found that a self-paced learning plan within a clear timeframe works best. If participants need to balance study while maintaining existing full-time employment, you should manage their expectations to approximately 7-10 hours per week over a defined number of weeks. For the best graduation rates, participants should be guided through the content, and prompted to complete various milestones within a certain timeframe to remain part of (or even to try and beat) "the pack".

   Crucial to a successful academy is providing expert support. This should be delivered via a communication platform such as Slack or Discord, with clear signposting about when support is *not* provided (weekends, office hours, etc). Including live sessions with experts helps build a collegiate atmosphere, puts a human face to the tech, and allows the participants to project themselves into their future post-graduation. We have found that experts are asked almost as many non-technical questions as technical ones.

3. **Examination** – We recommend awarding certification to successful devs based on an examination, as achieving course *completion* is only a signifier of presence, not quality of understanding. At the end of the program, participants should complete and submit a standardised project that tests their knowledge and application of the course content.

   Consider awarding successful graduates an official certification. This concretely rewards devs while passively promoting your program, and allows them to signal their quality to network partners looking to hire. The final exam should be completed within a defined timeframe (usually within 2-4 weeks, depending on the complexity of the assignment).

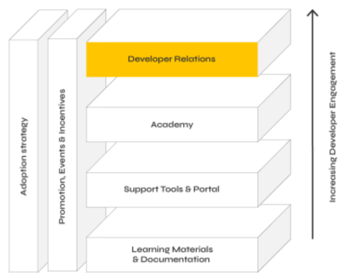
4. **Hiring** – As the course goes on, you should open up the cohort to ecosystem members looking to hire. This engages the participants and signals to ecosystem members and enterprise adopters that talent is available. Live AMAs and project presentations benefit both career devs and enterprise devs. We also recommend hosting a jobs board that collects relevant roles in one place for participants to see.

For an academy to be a truly effective recruitment and retention tool, we recommend as much interaction with developers as possible. Consider an opening ceremony and weekly "office hours" support sessions at times that work for your participants (accounting for global access). Weekly AMAs with subject matter experts are a real bonus for participants, and also create excellent content for your future cohorts and can be repurposed into promotional material.

Include quizzes and self-assessment tasks that stimulate interaction. You could also create a non-standardised project challenge that allows successful participants to work together on ideas for pitching to your grants or enterprise programs. We also find a closing ceremony with awards creates engagement and gives devs something to work towards and an incentive to finish.

Developer Relations

Active developer support


Adoption strategy
Promotion, Events & Incentives
Developer Relations
Academy
Support Tools & Portal
Learning Materials & Documentation
Increasing Developer Engagement

Developers evaluate new stacks or frameworks using different heuristics. One of the most important, as discussed, is the ability to solve problems quickly. A good way to do this is to talk to someone knowledgeable. Providing this kind of active support and stimulating intra-community support sends a strong signal to developers that you take them seriously.

Dedicated support

The easiest way for you to facilitate developer support is to dedicate team members to answering community questions in your Discord or other community space. Applying proven tech support practices is indispensable; inefficient support can get expensive fast. We always recommend ticketing and detailed support performance tracking.

Be sure not to casually assign this activity to junior developers on your team, or worse, rely on your senior developers to provide it. Doing so pulls their focus from their core roles, and as a result they are unable to keep response times that work for exploring and learning developers.

Community support

This is the most overestimated way to provide support. Many organisations hand responsibility for support and experience to their community on the easy assumption that “They can help each other”. Community support *is* a useful resource, but it still requires proper implementation.

Community support must be structured well. Find the most dedicated community members and make them ambassadors. Onboard them on your support system and leaderboard. Make it exciting for them to look behind the curtain and be part of the team (and pay for their time in some way – token payments can be a great way to incentivise ambassadors). But even with a strong team of ambassadors, you'll need a continuous campaign to recruit more: ambassadors have a shelf-life; you will need to replenish your stock. Your ambassador program is an ongoing concern, you can't just set it and forget it. You need a team to manage your community team!

Reactive FAQs versus directive FAQs

Everyone misses even obvious things, and FAQs are a ubiquitous element of online support. They may easily become a feature of many different stages of your overall design. However, it’s important to differentiate between *directive* FAQ and *reactive* FAQ banks.

It’s part of your team’s job to help devs out, so you likely already have some form of **reactive FAQ bank** for repetitive questions asked on a daily basis:

"What is [basic concept]?"

"Where is [thing that can be found on your website]?"

"When is [event that is clearly signposted]?"

By contrast, a **directive FAQ bank** helps identify the specific questions that each developer *type* asks. They not only answer a given question but also direct the questioner to the next set of information on their journey. Let's see an example of a query that is more dev-type specific:

Q: "Where is the grants program?"
[*Identifies the questioner as a* pioneer dev *or* enterprise dev.]

A: "You can find a link to our current grants program here."
[*Primary answer*.]

"We also have a list of projects in our ecosystem so you can see what has been funded before."
[*Shows what they will become a part of; that there are compatible projects already in your ecosystem; that grants are successful with you*.]

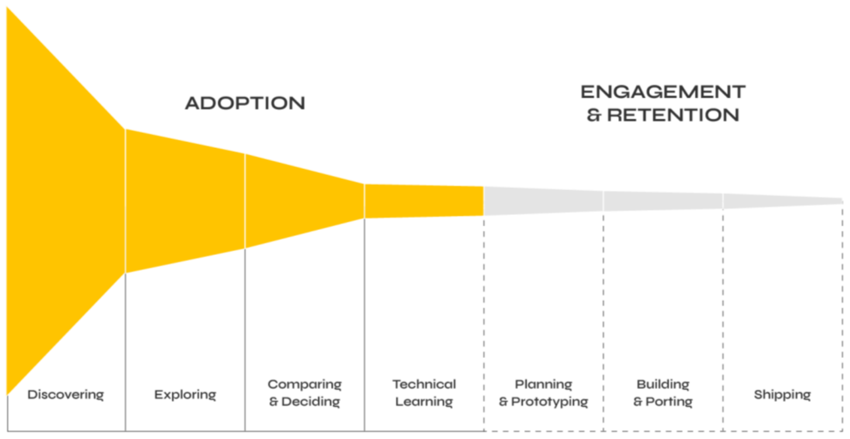
"Later on you might be interested in our XYZ program. You can read an outline here."
[*Directs them to your enterprise offering or other business help you provide*.]
[*Clearly indicates the next step after the grant program*.]

"We have an AMA with SpeakerName on XYZ which will likely cover ABC updates. I think it might be useful, would you like me to send you an invite?"
[*Shows your commitment to new devs; that your tech is moving; that you provide relevant information*.]
[**Important:** *initiates a second interaction and seeks commitment to an event*.]

That might seem like a lot of answers to a single question, but as you've been able to identify a target type from that question's context, you continue by showing them a path to success that directly relates to their likely needs and interests. Training your team to identify and satisfy target dev types is key to successful developer adoption, as is making them familiar with your adoption journeys and giving them the materials to help direct new devs to their next step.

Promotion, Events, & Incentives

Cultivating the next generation of expert developers means reaching out to where they are now and bringing them into your community and ecosystem. Undertaking promotion and events focused on upskilling devs is a consistently engaging message that attracts attention and reflects positively on you. Wielded effectively, the DevJAM stack creates the raw material for promotional teams to tailor, boosting your signal.



Promotions, events, and incentives are relevant to all parts of the developer adoption funnel and form the backbone of the engagement and retention portion of the developer journey. They are the way you tell your target developers why your project is important to them.

Promotion

When you target the right devs, and create the right messaging and positioning to do so, you make them aware and interested – and they come to you. Once you've captured their attention, promotion moves them down the funnel. It shows them the benefits of the project *for them*, the social proof to encourage them to engage; it tells them how to make the next step, and makes them want to. Promotion also nurtures developers in your funnel if you are not ready for them to move to the next stage; without it, you risk them dropping out and moving on to something else.

We use this catch-all term, "promotion", because it should be a consideration for all your developer-facing teams and require them to work together to create a seamless promotional experience. Promotion in developer adoption encompasses everything from marketing driven by educational content to PR messaging, email, and your promotional content across all social media platforms and your website. It's also a vital part of developers' experience with your community and DevRel teams.

Promotion ranges from the entirely non-technical – say a blog post announcing an event – to the intricacies of the benefits of your next upgrade. And here we highlight an important word: ***benefits***. Telling your insiders about the technicalities of your next upgrade is not promotion, it's *dispensing information*. Telling a *curious* audience about the *benefits* of your next upgrade is *promotion*.

Many projects confuse the two, and think that because they have conveyed information they have done promotion too. Even experienced marketing teams in web3 fall into this trap. That's not to say that information has no value, of course it does. But its value is limited to your insiders, and for your outreach to flourish it needs to extend far beyond that core group.

Promotion and your call-to-action

The acid test for the *information versus benefits* trap is identifying your **call-to-action** (CTA). Is there a clear next step you'd like your audience to take? CTAs are the indicators that show your audience the next step in their journey; they are absolutely essential to your developer adoption funnel. **If there is no CTA, you're not doing promotion**, because the aim of promotion in developer adoption is to capture *and direct* target devs through your funnel.

Best practices with CTAs:

- CTAs should create urgency, offer incentives, encourage participation, or do all three.
- CTAs should be targeted to your developer types and personalised if possible.
- You should adapt your CTAs to their context; for example, the CTA at the end of a blog post is going to be very different from a "sign up" button on a landing page.
- Each CTA should be concise and only reference one action. You can use different CTAs to allow your targets to funnel themselves, but if they're next to each other they should be about the same thing. *"Start now", "Start later",* etc.
- CTAs should always lead to the next step in the funnel, and align with the overall goal of developer adoption. A CTA that takes a dev into a side loop is not a useful CTA.
- Always experiment with your CTAs, both for different placements and strategies such as A/B testing, in order to determine where calls-to-action prove most effective.

Educational content marketing

In developer adoption, education-focused content marketing is the core of your strategy. That is not to say that you have to create entirely new pieces of content solely for promotion. Instead, you should be repurposing existing information to highlight the benefits of your tech for your audience, with a call to action telling them what their next step should be.

Look at your last 15 blogposts. There's a high likelihood that 90% are informational posts for your stack insiders. As an exercise, see if you can turn three of them into educational content marketing: think how they might become of interest to a curious outsider of any of the five developer types, and what you want their next step to be after they've read it.

For the record, the suggestion above will take you out of the journey we intended for you – to continue reading this document – meaning this is not an effective CTA for us! We should rewrite the CTA to say, *"When you've finished reading this document, look at your last 15 blogposts"*!

Clearly, any information in your education portal is fair game when it comes to repurposing, but there are many different ways to repurpose. The opening few paragraphs of an entry-level explainer can be targeted in an email nurture campaign, to lead devs to click through to the full content to know more; this in turn might lead to a recording of a relevant Twitter space; this in turn leads to a sign-up for an AMA; *all of a sudden your target is on your Discord*.

Repurposing snippets from an in-depth AMA for a less knowledgeable audience is also pretty simple. Then you push these snippets out via your socials along with a CTA to sign up for an entry-level event in the near future. Content originally targeting one group is now targeting two.

Curating the right content for a given target audience requires that you have their needs top of mind when you're evaluating your output. This allows you to create comfortable and easy journeys to adoption. As discussed previously, you've agreed on your shared goal; now your job is to make it as easy as possible to get devs to the destination. Educational content marketing is the simplest way to do that.

Nurturing interest

Many web3 projects severely neglect the nurturing aspect of promotion when reaching out to capture curious outsiders and get them into the funnel. It's common to leave promotional activities to the last minute, which doesn't allow campaigns to run for a sufficient period of time to raise awareness and gain the reach that will attract the curious outsider persona.

The failure of nurturing in web3 is particularly stark when doing organic promotion (social media campaigns, email campaigns, etc). It is not enough to send out three tweets, a blog post, and a press release one week before something is happening and expect the word to reach curious outsiders – it won't. Promotion isn't about moving fast and breaking things, it's about thoughtful journeys and giving your audience time to assimilate and join you.

Interestingly, web3 projects tend to give TOFU *paid promotion* more time (paid features, advertising, etc), possibly because they outsource it. However, this type of promotion is often incorrectly targeted, perhaps also because it is outsourced. If you're going to invest in paid promotion, use your best-performing organic promotion as the basis for your campaign, particularly those that are highly targeted and are benefit focused. If you give your agency that kind of content to work with, you will get far more for your money. Again, you'd be amazed how many teams don't do this.

Once devs are in your funnel, nurture campaigns are often neglected if you aren't yet ready to take them to the next stage. For example, while gauging interest in your new academy program, your teams ask for sign-ups so that they can let developers know when the academy launches – but then those devs may wait weeks, maybe months, before you get in touch again. Even worse is when your teams sign-up curious outsiders to receive information that is way outside of their skill level, developer type, or both. These approaches will turn devs off, either by moving their interest to something else or actively blocking you because you've sent the wrong type of engagement.

Instead, teams should create a nurture campaign that relates to the academy (which the dev is interested in), keeps the waiting devs warm, and even skills them up before the academy starts. Create a weekly or two-weekly initiative that communicates skill-relevant information, events, and other timely benefits you can iterate on, and hone them into a comprehensive onboarding experience that is curated through an automated email campaign. Your devs are now learning, staying engaged, and becoming part of your community, all while waiting to start your academy.

Metrics

Web3 suffers from a lack of metrics analysis and iteration when it comes to promotion. Teams are very focused on moving fast, shipping, and moving on to the next thing, and this attitude often bleeds into promotion. It forces your promotions team to focus on output rather than outcomes and doesn't give them the space to iterate. Promotion isn't development, and forcing your promotions team to treat it as such does your overall project no good.

Developer adoption is an evergreen process that takes developers from *somewhere near point A* to *somewhere near point B* over and over again, fine-tuning content and feeling out new tactics with each round of activity. Even though each dev will only experience a given interaction once, this is a continuous process for your promotions team. Making sure to track and analyse the metrics of your journeys improves the experience for the next dev.

Every planned (and unplanned) interaction with your target devs should be deeply understood. At its most basic you should be assessing the effectiveness of every email in a nurture campaign so you can make improvements. Sources for sign-ups to an event should be tracked so that your promotion can be optimised, as should the success of any devs attracted into starting a journey through your various channels; only then can you know what works.

Setting up and consistently analysing the metrics of your developer adoption journey is the most effective way to improve it over time. Treating your promotion as a collection of one-time "ship it and move on" activities is a serious mistake.

Events

Events are one of the few places you are able to directly drive engagement and facilitate interaction. Events can be anything from large-scale hackathons, local meetups, Twitter spaces and YouTube lives, through to intimate coding events with just a handful of devs. They represent a rare two-way interaction in promotional activity. Events allow devs to make actual human connections, with you and with each other. That can make a huge difference in their emotional and psychological journey while adopting your tech.

Building connections from the beginning increases a dev's confidence in themselves, and of their place in your tech. It also increases the perceived credibility of your project as a whole.

When to implement events

Depending on the phase of the developer adoption journey you are focusing on, there are different advantages to be gained by thinking clearly about what you and your target devs currently most need to achieve. Let's look at some examples.

Top of funnel events

Events planned for TOFU allow your target devs to ask the questions they want answers to, but *you* also gain an opportunity to gather information! TOFU events also facilitate objection handling: they let your devs raise points and objections, so you can address them head-on. These events allow devs who might not be quite right for your stack to self-select out early, which means you can focus entirely on those devs who are right for you right now.

TOFU events should occur monthly, and be branded as introductory events. If you want to consistently attract new talent, you need to undertake these events religiously. They don’t need to be carbon copies of the same event: one month you do things online, the next it might be an in-person meetup within a targeted geography, and so on. However, the aim of these events is always the same: to engage developers at TOFU and triage them.

Promoting these events is straightforward. They form the main focus for the early stages of your email nurture campaign, and other ways you might be targeting curious outsiders. You wouldn't promote them monthly on your social media – that is not a targeted enough medium.

Admittedly, snippets from events can provide useful TOFU promotion on your socials, but they should direct to an email sign-up or some other type of content that leads to the event sign-up. Spamming your socials for the event specifically will bring in lower and lower returns from your target audience, and will also serve to alienate your core bought-in audience.

Middle of funnel events

Events that target your active learners, those in the "technical learning" phase of developer adoption, are invaluable. An academy is itself a large MOFU event, and will prove much more successful at turning curious outsiders into stack insiders than a hackathon. But there is lots of space for smaller MOFU events as well, run in conjunction with your academy, or as standalone events if you aren’t at the stage where an academy makes sense for you.

With a lot of MOFU events the value is not just the event itself but the content that the event creates: material you can repackage across many platforms, not just for short-term promotion but as extra content for your learning portal and for your long-term nurturing campaigns.

With MOFU events, greater frequency with smaller turnout is ideal: not only does this give you a valuable stream of content, it also works especially well for the attending devs. They each have more time to ask questions and interact with your representative, giving a sense of connection and value. The smaller the event, the more interaction and participation, and therefore buy-in.

Your portal gives you a lot of data on areas that you should create events around. Analysing that, and the questions asked on your Discord, will show you how face-to-face engagement will bring rewards. AMAs, live coding events, or structured presentations based on portal content should always be recorded – firstly to enrich your portal, and secondly to act as promotional material for catching the attention and interest of your TOFU devs.

Presentations and AMAs with teams from projects already using your tech are very useful as MOFU events. They allow target devs to interact with potential employers and expose them to new ideas and ways of thinking about your tech. They represent perks and promotions for the projects you showcase, and can work across the social media platforms of all involved when properly packaged.

Bottom of funnel events

You're probably already doing a few BOFU events for stack insiders further down your funnel. Even BOFU events should be promotional as well as informational. They can be repurposed as teasers for your MOFU audience, making sure you highlight the benefits. We'll cover effective BOFU events in our later paper on effective engagement and retention for your builders.

Local meetups

As part of developer adoption, local meetups are great TOFU and MOFU events. They are ideal for community building, and for engaging curious outsiders. If things are going well in your ecosystem, you will hopefully have a few knowledgeable community members present that can triage and direct your target devs to the right places in their journey.

However, the prime advantage of meetups is tribe building, helping devs feel they belong. This is a very important part of their emotional journey. Let's not forget that even though we are distributed and global, having local like-minded people is a strong driver of adoption.

Hackathons

Clearly, hackathons have promotional use for your project, and it's a boost to the ego to be a judge at the hackathon or keynote speaker at your own dev conference. It's hard to pass up the opportunity for a flattering photo with a flesh-coloured head mic in front of a huge screen with your name on it. If you're *really* lucky, it will be a gently swirling blue background with quite a bit of black, and you'll look serious but also dynamic. We're all human, and that feels good.

Hackathons are flashy. They're a big concrete thing in your diary, and everyone in your team gets energised as *Hackathon Day* approaches. It's buzzy, it's different from your nine-to-five, and it costs a lot of money. It's a signal to your competitors that you're successful. It speaks well to stack insiders, and may even act as a pull to late-stage emerging insiders.

However, as part of a developer adoption journey, hackathons alone are a waste of time.

If your main aim is to drive developer adoption, then you should be aware that these events give a very low return on investment, *at any stage of growth*. They can provide a degree of success, but only if they're part of a defined developer journey with clear entry points and a variety of end goals, with harvestable metrics to measure successful outcomes.

Traditional hackathons and other large developer conferences don't answer the questions that make new developers adopt your tech. You need to consciously include these answers before, during, and afterwards. Pre- and post-event activity is far more valuable to adoption than the hackathon itself. The real work is done months in advance, and months afterwards. The exciting main event is just a shiny, expensive bauble to attract attention. With snacks.

Making hackathons pay

If you are determined to go down the hackathon route and developer adoption is a desired outcome, then there are ways to help make that happen. It won't happen on its own.

Before the event, you should have in place a structured learning path for your target devs, one that gives them enough time to learn in advance while maintaining a full-time job – that means launching educational content months beforehand. Curate an exclusive virtual space to foster a collegiate atmosphere and a variety of spaces to provide support. To demonstrate your tech is thriving, populate a jobs board or portal, have teams give project presentations and AMAs, and invite venture capital firms to present their best practices.

You should also have a parallel track for enterprise adopters that makes the most of the effort you're outlying and specifically addresses enterprise issues, and the issues of devs working in those enterprises. Sending their team to a hackathon is a nice shiny bauble for them too. These are only a few ideas, barely scratching the surface of all the things you need to do pre-event!

After the event, you should build on that collegiate atmosphere in every way you can: *create an alumni group; actively follow up with participants individually; do a second round of project presentations, raise awareness of your grants program, and provide introductions to VC programs; juice up your jobs board, and make sure that any tech that is tangentially associated with yours is represented – even create a dedicated team to provide business support; revisit your learning materials and update them where needed; ensure there are a variety of paths open to devs for further learning and development*… the to-do list is long.

There are many more options for pre- and post-event activities – that could fill an entire paper in itself. The real takeaway is that, if you must do all that pre- and post-work, it's logical to make them part of your everyday developer journey, while expensive hackathons become merely optional. These events are *marketing* more than they are a value driver or path to developer adoption. Worse, they could be your promotion and events team saying *"See? We're doing a thing, we're being productive!"* – which means you're not incentivizing them correctly.

Ask your marketing team what they would spend that money on. If you give them a budget and you have your developer funnels in place, generally they won't say "a hackathon". They won't *need* a hackathon to generate buzz, because you are already doing everything that captures the interest of the target developers you want to attract. You have achieved that goal already, and your marketing team can instead spend their budget more wisely.

A hackathon might be a glamorous hook to hang your adoption strategy from for a week, but effective developer adoption should be the foundation of your project, not decoration for it.

Incentives

Incentives deserve special attention, because a lot of the incentives for developers reside much further down the developer funnel and outside of the adoption purview. However, they need to be consciously promoted to developers right from the beginning of their journey, to attract their attention and retain their interest throughout the process.

For career devs, a clear incentive for adoption is demonstrating there are well-paying roles with other benefits, ready for them to fill. You may have an automated jobs board that consolidates roles in your ecosystem. You should also create content that highlights developer success stories: *Dev X from Place Y gets new role in Company Z, with interesting anecdote attached*. Case studies of successful projects work well for attracting career devs (as somewhere to find work) and entrepreneur devs (as a success story they can emulate).

Bug bounties and other rewards for self-directed troubleshooting are interesting to devs at the beginning of their journey, highlighting the possibility of reaping rewards relatively quickly after learning. Resurfacing and promoting these rewards right at the end of their technical learning phase can incentivise devs to keep going as they hit the harder challenges.

Hackathons *can* act as an incentive for pioneer devs at the very start of a project's lifecycle. Pioneer devs generally like the intensity of a hackathon, and if you have massive funding it's a good way to splash some cash for other reasons with the added benefit that you will attract pioneer devs. Still, we'll never recommend them for growth and mature developer adoption!

Grants programs

The correct curation of a grants program can be an amazing stimulus to developer adoption across all stages of your growth, but in particular in your early stages.

You know your tech, and you know your ecosystem. One of the most powerful ways to make sure that strategic projects get built and pioneer or entrepreneur devs enter your ecosystem is to create well-structured incentives through targeted grants. When it comes to developer adoption, grants act as a strong signal that committing to study provides a path to reward.

As the expert on your ecosystem, you are perfectly placed to create detailed specs of the projects and infrastructure your ecosystem needs to be successful. Grant programs are simply a way to find collaborators to fulfil your needs and, to a lesser extent, to fund innovative ideas coming from the community.

- **Myth**: If you create an open RFP-based grant program, creative developers will rush to submit proposals you hadn't even considered. New developers will be incentivised to join the ecosystem and submit a proposal.
- **Reality**: You know your ecosystem best. You know best what business cases are most valuable. You know what's needed strategically. New developers must spend months orienting themselves before being able to think creatively about your tech.

Migration grants

This is your most important tool early-stage when targeting pioneer developers, and to a lesser extent enterprise adopters. Migration grants are designed to lower the perceived risk for teams on other platforms investigating a move to your platform. Target the projects you want to migrate and pay them to investigate and build proof-of-concepts.

Project grants

Early-stage growth is also driven by support for clearly specified applications and infrastructure. If you want a DEX on your platform, spec it out and ask for teams to apply to build it. This wins both ways: applicant teams don't have to speculate about the viability of an idea, or whether there's interest in it; and you only need to assess whether the team applying has the ability to deliver on the specification.

Infrastructure grants

You might need people to run and experiment with nodes, validators, retrieval nodes, archive nodes, and more. Specify what you need and dispense grants to interested sysadmins.

RFP grants

Open proposals can still have a place in your grants program, just not as the most important element. Once you have an established grant process and you see your developer community maturing, it can make sense to invite open proposals.

DevJAM in real life

As is no doubt clear, the extent of DevJAM is extremely wide-ranging. The impact it can have on the health and future of projects, technologies, and ecosystems is similarly profound. Of course, B9Lab can help you with every stage of the DevJAM stack – but before you reach out to us, why not take a look below to learn the best time in your evolution to activate which bits of the stack?

Now *that's* a CTA!

Applying DevJAM – what to do and when

For organisations as much as individual devs, strategising for adoption is not a one-size-fits-all proposition. Depending on the scale of your activities, your needs and your scope for realising returns on investment can vary significantly. That is what we will address in this final section.

Every developer-enabled project passes through broadly similar developmental phases with different needs and requirements in developer adoption. Let's look at three brackets.

Early phase – *from the garage to 50 developers*

In their early incarnations, projects are looking to launch a testnet, a sandbox, or have a beta stage in development. Here your goal is to get pioneer developers and a few entrepreneur and career devs to kick your MVP to see if it's fit for purpose. You will be planning to track success and failure, and use this experimentation to iterate and plan for your first rounds of funding.

In this phase you have the following characteristics:

- **No institutional infrastructure.** You are still a single organisation; you are yet to set up a foundation, or to have a dedicated department or organisation that runs your grants program, etc. You're agile and nimble, as there are many paths to success to pick from.
- **No third-party service providers**. You're still building and experimenting, and do most things in-house. You don't have much of an ecosystem, at best you have a community. This is highlighted by the lack of third parties interested in providing services to projects adopting your tech. There are no independent validators, no wallet creators, and head hunters aren't sniffing around for business because at this phase that would be overkill.
- **Little developer adoption structure.** You will need to implement developer adoption infrastructure to move your project to the next level, but currently your adoption is one-to-one. In the very near future you will need to create dedicated learning material to help your target developers get up to speed on your tech; take the time to document your one-to-one teaching, including common questions and mistakes, as that will form the base of later learning materials. Make sure to implement this approach across your whole team so you don't miss valuable detail. This knowledge base will be invaluable.
- **No structured education/certification infrastructure.** Certification and very structured learning isn't necessary and will have a very low ROI, in part because of the likely need for multiple revisions.
- **Focused on few regions**. By far the best strategy at this phase, not least to reduce the burden of language translations and the amount of support you will need to provide.

- **Not equipped for enterprise adoption**. Unless enterprise adoption is key to your early success, creating the infrastructure and journeys to onboard enterprise is burdensome.

Growth phase – *from 50 to 500 developers*

Developing projects have a testnet, sandbox, or beta launched. You have collected metrics and iterated based on them. You now have enough information and traction to move to mainnet or a product launch. At the early stage of this phase you need funding and infrastructure to support your launch, including a flow of developers to start deploying. Your adoption strategy needs to kick in now to get your 500 devs. During the later stages of this phase you need to be preparing for structured education.

- **Developing institutional infrastructure.** During this phase you will devolve some functions to other structures. You are setting up a foundation, a discrete grants program, perhaps splitting core development from other functions or dividing your open-source work from your for-profit arm. You are likely to have advisory boards, committees, and working groups. Equally, you are putting in place the team that will manage your enterprise adoption, be it a growth team or a builders' programme.

- **Third-party service providers**. During this phase you are becoming an ecosystem. A key sign is third-party providers recognising the value in teams outside of your purview and catering for them. Outsourced devs teams start to skill up on your tech because they are seeing demand from clients. Validators and wallets are being built without full funding from you. Independent YouTube channels and podcasts spring up discussing your tech.

- **Basic developer adoption infrastructure.** To cater for demand you need infrastructure to provide developer support, along with clear developer funnels and journeys for your target groups. Your documentation needs to be clean, cohesive, and on point. You have a developer relations team, a discrete knowledge base, and a set of learning materials. Your DevRel and community teams systematically leverage third-party platforms to stimulate engagement. Your marketing and events teams plan and enact the first version of your in-bound developer interest strategy, which will feed your growth from now on.

- **Basic structured education/certification infrastructure**. By the end of this phase you should have a learning portal that contains tutorials, tech explainers, and at least one curated learning journey to help developers learn the basics of your tech. This could be a modular course or something less structured, but it should have a clear beginning, middle, and end, with some form of self-assessment so devs can see their progress. You are piloting and possibly have already started a certification program, which is important for your career devs. Structured education reduces the burden on your DevRel team to provide basic support; they can then focus on creating more tutorials and explainers for your new releases and troubleshooting advanced issues, taking the burden from core developers who are too talented and expensive to be performing these tasks.

- **Deliberate cross-team collaboration.** When you were a smaller, leaner organisation cross-team collaboration came naturally, as it wasn't possible to do something impactful without getting the buy-in and collaboration from other teams. Not anymore! *Marketing creates initiatives that send the wrong type of enquiries onto the Discord and swamp your DevRel team. DevRel hosts and curates fascinating Twitter spaces, AMAs, and live streams without ever telling marketing or events. Your dev team decides on a new documentation system without talking to DevRel about what the devs they speak to are asking for.* Without coordination, you get chaos. It is now you need to consciously get full team buy-in on your holistic adoption strategy, and delineate the areas of ownership, areas of collaboration, processes for collaborating, and underline the jeopardy involved in not collaborating – otherwise you will be wasting time, money, and resources.
- **Developing global reach.** You have extended your strategy into a handful of regions that may include a second or third language. Having a team verify the quality of content and support in multiple languages is now key. Of course, you can outsource translation through community grants and by onboarding external support teams; but ensuring quality and monitoring outcomes and output will be your major effort at this time.
- **Looking to increase enterprise adoption.** As you are now serving the needs of your growing entrepreneur and career dev communities, you will be able to create learning support offerings and demonstrate your ecosystem's ability to provide what enterprise adopters need. You will be targeting CEO and CTO personas with technical and non-technical information from your growth, marketing, and education teams.

Mature phase – *from 500 to 5,000 developers, and beyond…*

Although this feels like a destination, this is really just the end of the beginning! Now comes the logistics of managing (and deliberately *not* managing) a disparate set of competing demands for attention, investment, development, and growth. Without an intentional, long-term strategy for developer adoption, you will come up against many obstacles in human capital management, culture, and project sustainability. This is especially complex when navigating from a relatively centralised structure to decentralised one, which as a web3 project is normally the goal. You are becoming the steward of a many-headed beast – picking which heads to control, if any, is vital.

- **Strong institutional infrastructure.** You have ironed out most of the kinks in your institutional infrastructure and it has bedded down to run relatively smoothly and without needing intervention. Unfortunately, the downside of becoming "the establishment" is dealing with the politics that inevitably comes with that. The larger you are, the wider the set of goals across individuals, teams, departments, and projects. Much of your senior teams' time is spent trying to align goals and settle disputes. As a web3 ecosystem, this is often done in an open forum, which can be tough.

- **Solid third-party service providers.** Your ecosystem is established and growing solidly. Third-party service providers are picking up the slack in areas you feel are non-core to your focus, even finding and creating pockets of innovation you hadn't anticipated. You are now increasingly attractive to scammers, an unpleasant side effect. This can reflect very badly on your ecosystem and deter developers from adopting your tech. It is also a heavy burden on your community managers, DevRel, marketing, and PR teams. Early in this phase you need to put policies in place, and even a team to help fight scammers.

- **Advanced developer adoption infrastructure**. Your portal now has advanced learning content in the form of a number of journeys addressing every type of developer you'd want in your ecosystem. You have the content to take a career dev with three years of web2 experience from having zero web3 knowledge to being eminently employable in your ecosystem across a range of roles, including sysadmin, QA, CTO/architect, non-technical roles, and enterprise adopters. These are all participants you need in your ecosystem, and each has a different learning journey that you can address and manage.

  As your ecosystem becomes more complex, you are curating submissions from ecosystem participants, linking to their content, and have processes to create consistent syntax, voice, and tone. Your portal needs to remain the primary source of up-to-date and verifiable truth, but its content doesn't need to be entirely created by your team. However, it does need to be curated by them, and that requires dedicated effort.

- **Structured education/certification infrastructure**. As you grow past 1,000 developers, it is in the interest of third-party providers to create gated, paid-for content. This can work in your favour, as you have created a certification program that third-party providers can prepare developers to take. Your certifications demand a high pass mark following an examination, and are complex and varied enough not to be gamed. They are the gold standard for certification in your tech. As steward of the ecosystem, it is imperative you only certify the best for multiple personas. You also offer tiers of certification, so that it is possible for less skilled developers to gain accreditation and start earning.

  Academies are an excellent addition to your learning offering in this phase. They let you manage the flow of highly educated, fit-for-purpose devs into your ecosystem. Having your own academy is one of the cheapest ways to orchestrate and direct ecosystem growth. They are signals of a flourishing ecosystem, and your dedication to sterwarding it – the career developer equivalent of the most successful hackathon you ever had.

- **Stewarding cross-team and cross-project collaboration**. The issue that almost all decentralised systems fall down on is admin. Actors in web3 continue to believe *there must be a tech solution* when, most of the time, it's humans we need. As the steward of your growing ecosystem, it's now your role to be the champion for cross-team and cross-project collaboration, and that means picking up the admin. Each working group and committee needs a moderator and people in charge of minuting, following up, creating project plans, and rallying the participants to do what they promised.

With every new piece of educational content produced for your portal, there needs to be onboarding, review, quality control, editing, testing, and publishing. You direct which content is needed, accept and reject submissions, manage funding pools, create grant initiatives, and rally interest at every step – all the while making sure the relevant teams are informed, from marketing, events, product development, DevRel, talent, and education to all the other working groups and committees across the ecosystem.

- **Global reach, with full documentation and learning journeys in multiple languages.** As you sail past the 1,000 developer mark, you are creating full education packages in a variety of languages. These support your growing international communities, for which you also provide community managers and DevRel. You are able to translate and update relatively quickly to keep pace. As you likely aspire to a decentralised structure, it's wise to kick off new language communities with a few native language stewards able to manage grants programs and community initiatives. They use best practices established in your base language, are part of your structure, and accountable to your team. There is also scope for them to become independent and steward their own ecosystem around your tech, a process which takes time and a lot of effort on your part to support.
- **Enterprise adoption infrastructure is in place**. Enterprise adoption is now a core pillar of your future growth, so naturally you now have a mature infrastructure to support it.

Conclusion

Constructing multiple developer journeys for a variety of developer types at different stages of their awareness and experience requires cross-team and cross-ecosystem coordination.

Developer adoption is a human experience that can be facilitated by technology, but ultimately it relies on the interactions between people. Using technology to remove common friction frees up people taking their journeys to share meaningful and impactful relationships. This brings value to both the developers doing the learning and your existing teams supporting them.

Whatever stage your project is at, positive interaction with new developers brings in new ideas and invigorates everyone involved. Good developer adoption makes these benefits possible.

If you’d like to have a chat about how B9Lab can help you accelerate developer adoption for your project, drop us a message and we can arrange a time to talk.

Index

B9Lab