What I Learned at Clubhouse

Every <hello@every.to>
To: <ben@joinladder.com>

Thu, Sep 1, 5:31 PM

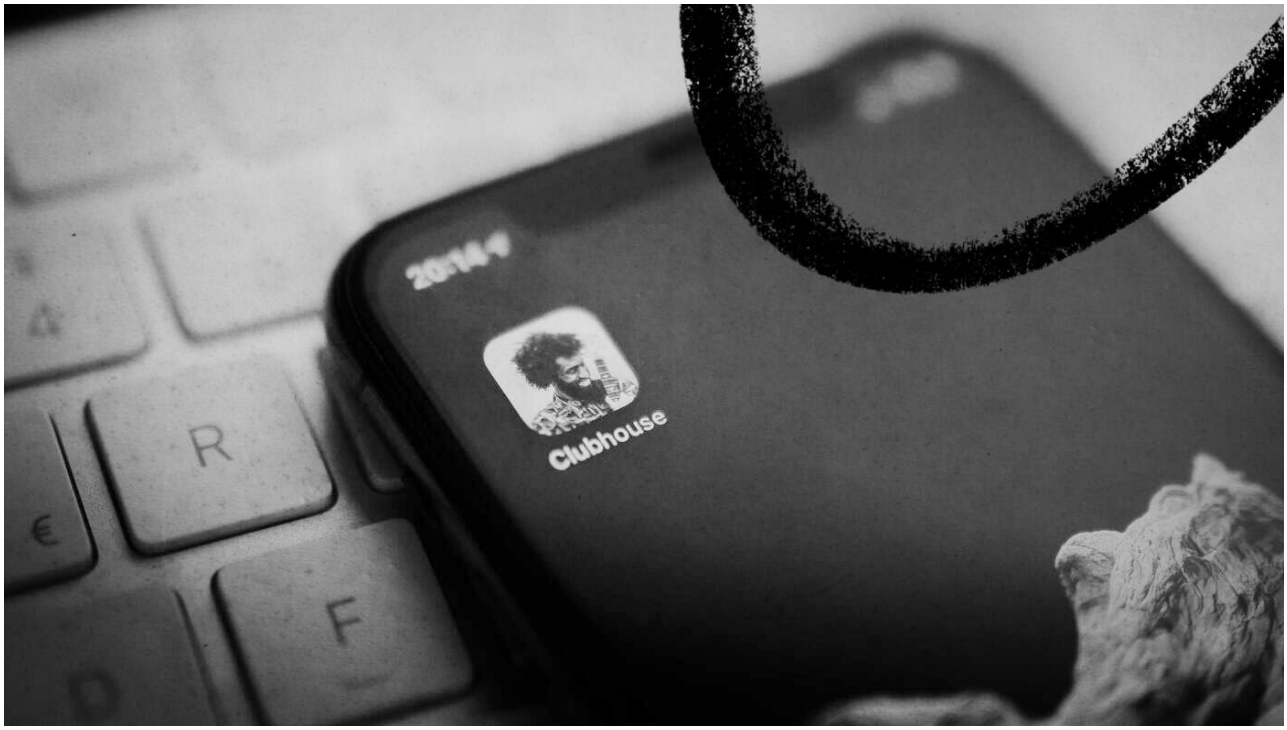
EVERY

This is Every's pick for essay of the day.

What I Learned at Clubhouse

25 lessons from an early employee on product, community, go-to-market, and more

by Anu Atluru



William Kraus / Unsplash

Anu Atluru is a startup builder, angel investor, and writer. She spent two years at Clubhouse as its first head of community, and in this article she shares everything she learned as the app grew from just 1,000 early users. We're

extremely proud to have her share her learnings on Every, and we hope you like it!

As an early-stage operator, angel investor, and former founder, I've learned a lot about building startups and have become much more opinionated on how to do it well. I spent the last two years building at Clubhouse, where my learning was supercharged by the caliber of our team, the rate of growth, and the very nature of building in a new product category.

When I started at the company, the app had ~1,000 users. Rooms didn't have titles and "clubs" didn't exist yet. The team was just the two founders. Clubhouse has been built in public ever since, and as an early employee and its first head of community, I got to contribute to nearly every part of the company and developed a public persona in the process. During my tenure, the team grew from three to more than 100, and went from serving hundreds to millions of users daily around the world.

In recent months I craved time to explore new ideas again as a founder, and decided to leave Clubhouse while staying on as an advisor. I've taken away many personal lessons and refined perspectives on how to build and scale products, communities, and teams at startups—and how to do so in public.

Product

Lesson #1: Earn the right to build the next thing

Commit to following through on what you've already planned or launched before you chase the next creative, experimental, or shiny idea. "Earn the right to build" and "have respect for sequencing" are principles I've internalized.

At the earliest stages, it can mean doing diligence on user needs and testing with no-code MVPs before coding anything. While scoping, it can mean being disciplined about what *needs* to be in a version 1.0 of a feature. Post-launch, it can mean rigorously ensuring feature quality and evaluating feature success before thinking about what's next. And so on.

At a meta-level, it's valuable to set up a consistent product development process that considers strategic fit, prioritization, scope, launch, evaluation, and iteration.

Lesson #2: Strive for a great UX and a good-enough UI

Simply put, how things *work* matters more than how they *look*. This is especially true in the early stages of a product when you have supportive early adopters.

A great UX facilitates your product's core interactions — it's clear, easy, and simple. A bad UX makes it hard for users to do the thing you want them to do (or even know you want them to do it). Bad UX can turn users off a good product idea.

On the other hand, a "good-enough" UI is … good enough, especially since it can be more subjective. For example, color and font schema can influence perception of a product's purpose and audience (e.g., target age and interests). But a differentiated product with strong utility and great UX should overcome this.

Some evidence that this holds true: bug reports and product feedback are almost always about UX (or new feature requests), and the ones that seem like they're about UI usually overlap with usability issues.

Lesson #3: Product judgment is more skill than talent

Good product judgment is hard to define because it's empirical; you're only proven "right" after the fact and over time.

It's also a skill more than it is pure talent; you have to practice to get good and stay good at it. Arguably the most important way to invest your time is to deeply understand how users:

- Experience *your* product (so much so that you can even anticipate how various target user segments would react to changes in your product)
- Experience *other* products (know the evolving socio-cultural trends and understand the products that may compete to serve similar needs)

Product thinkers say that some combination of expertise, empathy, and creativity underlie good product judgment; deeply understanding the *product* and the *market* directly through users will help with all three. This doesn't have to mean building a UX research function early. The best results can come from doing it yourself (see lessons #6 and #22) and drawing on the collective wisdom of a team.

Lesson #4: Beware of complexity creep

The first version of your product is usually simple. As you keep building, the product becomes more robust in quality, utility, and experience. But there's a tipping point after which "robust" can sneakily switch to "complicated."

Some signs your product might be too complicated: you've launched a bunch of features and haven't killed any of them. You're running out of literal surface area to put new features. Users aren't clear about what to do with the product.

Complexity creep is a human problem. When we make things, we get attached. We don't want to discard people's work. We may not rely enough on data to make decisions. Or, we just aren't clear on the one or two *core* interactions that users need and value, so it's hard to choose what stays and what goes.

Anticipate this, then build in early checkpoints. Define success and failure *before* a launch. Build a solid process and track the right metrics to help make objective, decisive decisions—and don't forget to create a framework to sunset features that don't make the cut.

Community

Lesson #5: A 'community' is really a collection of many communities

If you have two users, you probably have two user personas. Even a small group of users can have a vastly differing set of experiences, perspectives, goals, and needs for your team's attention and your product. Hence, groups of users form and represent many distinct communities as a network grows.

User segmentation isn't an academic exercise; done right, it's a strategic and operational one. Use qualitative insights, data, and reference users to help define personas. At a strategic level, match value propositions to user segments. Choose to prioritize one user segment and therefore *not* another. At an operational level, classify users into specific segments and support them accordingly. Then build systems and allocate resources to do it well at scale.

Lesson #6: Talk to users (even if you don't think you need to)

User research shouldn't be a one-off thing, and it should be something *every* team does. You should talk to users when you're trying to figure out what product to build, when you're ready to launch, or even when users reject it. Talk to them not just about the product but also about their motivations and needs from a community perspective. Bonus points if you can engage users in a way that's native to your product.

There are many traps, both logical and emotional, that can get in the way. Steve Jobs's philosophy is one school of thought that can be misapplied ––*"Some people say give the customers what they want, but that's not my approach. Our job is to figure out what they're going to want before they do."* Even if this is true, talking to users is still valuable. Coming up with solutions without understanding the problems users face and the root of their requests can set your team up for failure.

Another potential trap is viewing your power users as a "vocal minority," hence not the users you're building for nor a source of valuable feedback. If you're distinguishing between users you're building for and those you're not, make sure you *really* understand the difference (see lesson #5). Then do the work to find and talk to your target users. I'm confident there are both thoughtful advocates and critics of every feature, program, or strategy you might be considering.

Lesson #7: Create value through user-to-user interactions

A true community is defined by the interactions, relationships, and value exchanged among its members. Hence, building a community alongside your product shouldn't focus just on team-to-user interactions, but also on user-to-user interactions. Figure out where team needs ––product insight, user education, user growth––align or don't align with the needs and interests of engaged users. Use this to help identify and test new engagement initiatives.

Importantly, don't just tell, but show users how you want them to engage. To that end, collaboration between community and product can be valuable. For example, for user education, demonstrate the feature in use via the product (e.g., onboarding flow) and the community (user-led onboarding sessions). If you want users to create, engage existing creators to show what a good process and output looks like. Your users and supporters can augment your team's impact.

Lesson #8: Trust and safety is a continual effort, not an end point

Managing user trust, safety, and privacy has to be a conscious and consistent effort; it's never too early to start and it's never fully "done." The first thing to do is anticipate what challenges your product uniquely presents in this domain. Then, figure out how to tackle challenges with the product, process, and team.

Create an initial set of rules––read: community guidelines––for how you want users to engage with your product, with each other, and with your team. Get input from your existing users. Avoid legalese up front so it's a quick exercise. This also helps give clarity to your team and guidance for decisions that arise.

Taking a stance on trust and safety early is especially important if you want to build an open network. Figure out which rules you want to centralize versus leave up to users. Anything centralized sets a network-wide standard where your team may be the arbiter; anything decentralized can be taken as permissible and may feel out of your control.

Go-to-market and growth

Lesson #9: Your friends are false positives until proven otherwise

Often the earliest users of your product are your friends, family, colleagues—people you personally onboarded and who have a vested interest in your success.

If you can't get your day 0 supporters to use and like the product (assuming they're in the target market), that's probably an accurate, and negative, signal. Even if they do like it and act like an "ideal" user would (e.g., sharing the product with their friends), you should still consider it a false positive. It's valuable feedback but isn't an objective indicator of quality or growth potential.

Extend out one more degree of separation—if you start to see friends of friends invite *their* friends repeatedly, *that's* a stronger type of signal. You're finally seeing intent that doesn't rely on your direct relationships and influence.

Lesson #10: Product-GTM fit comes before product-market fit

Product-market fit is an end state; a go-to-market (GTM) strategy, however, is an input you control. And without product-GTM fit, you can't *create* product-market fit; you can only stumble upon it (and who wants to depend on luck?).

One example of product-GTM fit is in choosing the right user to serve. If you're building a group-based product, figure out if there's a group "leader" archetype and if that's the user you should explicitly target and build for. In other words, make sure the user you're courting is right for the product you're building, and vice versa.

Another example is in picking an effective seeding strategy. If you're building a closed or default private network, you can pick many distinct cohorts to test with since they're effectively walled off from each other. If you're building an open or default public network, the first cohort matters more—seed it deliberately in character, size, and pace. This group will set early user behavior and culture, influence direction of growth, and give feedback that shapes the product.

Lesson #11: Start building a growth engine before you need it

You don't get to choose your growth rate. You can influence it or slow-drip it (e.g., by gating early access), but the

input-output relationship isn't linear, and it takes time to figure out the formula. Whether you onboard early users one by one or are fortunate enough to have viral growth, you *will* have to build a growth engine.

Start understanding how growth works for your product early so that when, not if, the growth rate shifts––up or down––you know what your levers are and can quickly move into execution mode. Treat the growth engine as a set of "loops" (circuits that compound) rather than "funnels" (one-way channels that don't compound).

Lesson #12: Invest in your community-led growth flywheel

Traditional product-led growth (PLG) means building a great self-serve product that can basically acquire and retain users itself. This is a direct product-to-user sale and a "best product wins" approach. Community-led growth (CLG) engages the community in the process, enabling a "best recommendation wins" approach.

PLG may work best for products that are utility-based and can provide value on day 1 (e.g., without a network of users), or once a product is socially or culturally validated. CLG stands out when the product's primary purpose isn't utility (e.g., it's entertainment, status, or new social connections) and people may have lots of options. For these types of products, a vibrant, vocal community can make a difference.

For instance, even a small group of early users that are community organizers or "network nodes" can champion a product and informally acquire, onboard, and help retain users. The tradeoff with CLG is that it's operational, and therefore slower and likely harder to find great economies of scale. But validated community-led approaches can also be built into the product in creative ways.

Strategy and operations

Lesson #13: Strategy is *not* doing everything

Strategy is equally about defining what you want to do *and* what you don't want to do. A good strategy can be written down in one page. It's actionable and measurable. It forces you to tackle distinct (or diverging) priorities in series instead of attempting them in parallel. It drives team alignment, hiring, and resource prioritization. A good strategy is communicated transparently and reiterated constantly inside the company, and the essense of it is shared with users, too.

Most teams start with a strategy. But once a product launches, there's no guarantee it'll be used how you expect or want it to be. When you start to see multiple use cases and user segments form, it's exciting, but it risks muddying your strategy––especially if one user segment demands a change in your product roadmap. In times like this, you are forced to decide which directions fit your strategy or are a distraction—because doing *everything* isn't a strategy.

Lesson #14: Expand scope in good times, narrow it in the bad

Your team's work should never be stagnant, but know when to push and when to pull back. When things are going well (e.g., the product is strong, growth is good, team is operating well, morale is high), expand the scope of work and experiment more. If things aren't great, revisit priorities and narrow scope in one or more ways. This is helpful at both a macro and micro level—for a company, team, or individual.

Lesson #15: Every team needs a strong operator

Operations is not just a function; it's also a skill. In the early days of a startup, you're one cross-functional team. One strong operator is enough. But as startups grow, you might set up independent teams for each function, and "operations" can get siloed. Or you might set up several distinct cross-functional teams that are each responsible for a business goal.

Regardless, make sure you still have a strong operator on every team—whether someone native to that team or by embedding a member of the operations team. Part of operating well is breaking down big challenges into actionable plans, getting team alignment, and relentlessly executing on them. Make sure someone on *every* team has the skills, agency, and support to play this valuable role.

Lesson #16: Know when and when *not* to measure ROI

You can think about ROI in terms of money or time––either your own, your team's, or even the company's. Consider this upfront before committing resources, but also when things have been done a certain way for a long time; there's usually room to prioritize and focus further. Sometimes this can mean building operating models to help understand the financial impact of operational decisions, and at other times it's a simple 80/20 exercise based on gut and team experience.

But sometimes ROI isn't the best measure—often for sub-scale, experimental, or other creative initiatives. Resist the urge to believe that things without a clear cost-benefit equation *today* aren't valuable. Instead, outline the objectives clearly and create a process to measure inputs vs. outputs and impact expected vs. delivered.

Building and managing teams

Lesson #17: Focus on skills, not functions

For early-stage and/or high-growth startups, traditional organizational functions are too rigidly defined. I've started to think in terms of *skills* instead.

There are only four hard skills you might need to do any job––data, design, engineering, and operations. Every "job" requires a unique weighted average of these four skills. Being a product manager can be a heavily operational role. Being a marketer can be a data-oriented role. "Functions" tend to focus on the application of these skills with a defined objective and in a defined context. Functions don't always translate across organizations, but skills usually do—so consider both when hiring.

With this framework as a starting point, you can better break down any role and understand skills required and the nature of day-to-day work. It can help hire more effectively, better structure and integrate teams, and help employees develop.

Lesson #18: Teams shouldn't be 'upstream' or 'downstream'

There are three top-level domains in most organizations—make, sell, and support. In a consumer tech startup, for example, the product, design, and engineering teams might constitute the "make" domain. The "front of the house" teams may "sell" (e.g., marketing), and the "back of the house" teams may "support" (e.g., customer success). Don't think of these domains as upstream or downstream of each other––think of them as collaborative.

It's easy to forget to build strong yet flexible touch points between domains, especially as you scale. This is also true within sub-teams and sub-sub-teams. Make sure to build in timely cross-team communication, collaboration, and shared decision-making.

Lesson #19: In hypergrowth, hire startup-stage chameleons

Hypergrowth feels like you're running a huge company with a tiny team, and like it happened overnight. Hiring yourself out of the imbalance is challenging. First, you should be reasonably confident that you have product-market fit so new hires will match the strategic direction. This is, of course, more critical for specialized and senior roles. Timing is also important because growing the team will paradoxically slow you down as you onboard, ramp up, and add layers.

New leadership hires should have three types of expertise:

- Building processes and systems to scale
- Hiring and managing teams
- Applying industry experience, ideally multiple reps at tackling similar challenges end-to-end

Bring on people who have this expertise *and* are extremely curious, use the product, want to talk to users, can run fast and lean, execute as an IC or manage people, and work cross-functionally. Yes, it's a lot to ask. You want people who *can* build 0 to 1 and *have* built the 1 to n—i.e., they can play early- or growth-stage roles.

Lesson #20: Communicate with optimism about the future but realism about the past

Optimism is valuable, but it's only forward-looking. When looking at the past, it's important to reflect with a heavy dose of realism. Team retros are a great tool. Which hypotheses, decisions, outcomes were good? Which weren't? Why?

It's also critical to *communicate* what's going well and what's not. Teams are smart and they'll know the reality regardless, whether it's because they're close to the data, talk to their teammates, spend time with users, or just use the product a lot themselves. Trust and morale aren't based just on optimism or success; they're built up by a consistent sense of transparency, alignment, and having a unified plan of action.

Lesson #21: Company culture is built in DMs and 1:1s

It matters far less what you say in all-hands meetings or post in a #general slack channel than what happens the rest of the time. Behaviors demonstrate culture, and most behaviors—good and bad—happen behind the scenes, in the slack DMs and the 1:1 meetings that are far more frequent than any public gestures.

Keep this in mind when hiring, too. The first 10-20 employees have an outsized impact on the culture and direction of a company, directly and indirectly. The people they hire, the team culture they set, and how they engage at all levels and in all channels will be hugely influential, so evaluate cultural fit holistically.

Working at a startup

Lesson #22: Be a power user of your own product

If talking to your users is important, being a user of your own product is just as critical. (Luckily in consumer tech you're almost always a potential user.)

There are lots of reasons to do this:

- It builds empathy for your users.
- It builds empathy for your team.
- It helps clarify user feedback and context-poor data.
- It can reveal new product ideas and, as a result,
- It improves your product judgment.

Pro tip: know your own usage patterns and those of your team. Find the power users across your team and draw on them for insights. Keep track of how usage changes, with or without product changes, and understand why. Product and user-facing teams in particular need to be immersed to keep learning.

Lesson #23: Unblock yourself and everyone else

Whenever possible, don't rely on or wait for other people to remove an obstacle. Tangible obstacles are things like skills, information, access, or time. Intangible obstacles are often related to communication or a psychological barrier like fear or lack of alignment.

Be proactive about recognizing and removing your own obstacles, communicating them transparently (see lesson #24), and making it easy for others to help you when necessary. Take it one step further and try to help unblock others proactively.

Lesson #24: Overcommunicate, don't overpromise

If I had to give a single piece of advice to anyone working in a fast-paced, high-ambiguity, cross-functional team environment, it's this—*overcommunicate.* This applies to the macro and micro, to every level, and on every day. Examples:

- Be responsive to requests and set context even if you won't be able to fulfill them quickly or possibly at all (read: *don't* overpromise).
- Keep people updated on the status of your work. Share weekly progress updates in an easily digestible form. Link to a central place where people can dive deeper into relevant work or catch up on progress at any time.
- Don't ask for permission––ask for forgiveness, but give a heads-up. Share your thought process and plan of action and leave room for debate or disagreement. You'll usually just get confirmation to go ahead and execute and avoid unwelcome surprises.
- Give constructive feedback to everyone (not just people who report to you) and ask for it in return. Do this whether the feedback is good and bad, and at regular intervals.

- Set boundaries to respect your personal health, relationships, and goals. It goes without saying these should be reasonable; then communicate them!

Lesson #25: Know which lane you want to be in

As I pondered leaving Clubhouse in recent months, I talked with mentors and friends in the company and externally. They all echoed a version of the same questions.

"What do you want to spend your days doing?"

"Which direction do you want to take your career?"

"What's important to you short-term vs. long-term?"

A common framework for startup career paths emerged––***founder, executive,*** or ***investor***. There's no hierarchy because they are fundamentally different lanes, but it's important to know which lane you're in now and which you want to be in. Some people stay in one lane; others zig and zag between them.

I've also found it clarifying to think about *energy*. People often ask how energized you are about your role; I like to ask how energized you are about the idea of doing anything else. *Relative* energy is just as impactful as absolute energy.

Once you know what you want, make sure you're:

- Building the relevant skills.
- Being direct about your goals.
- Tracking your own progress.
- Regularly zooming out to think big-picture.
- Finding a community of like-minded people.

My two cents: no matter what you want to do, do it with intention and energy.

This post originally appeared on Anu Atluru's Substack. You can also find her at her personal website here.

What did you think of this post?

Amazing Good Meh Bad