

Your First MVP and Next Steps

Making the most out of the stepping
stone that an MVP is



What's inside

Introduction	3
How to build a value-adding MVP	4
Researching users and conducting user tests	10
Keeping the development process efficient	14
Preparing for a big launch	16



Introduction

Every application has one purpose: to **help the users achieve a concrete goal** in a seamless and pleasant way. That definition is as general as it gets, but there's a point to this vagueness.

And this point is: usually, this is all you know when you start developing or expanding your software. You know you want to help the users. You may even know the exact activities the users should perform in your application. But **do users really need this?** And what do *seamless* and *pleasant* mean in this context? These questions are more challenging to answer.

Here's the thing: you can assume that the users DO need this feature, and what a good user experience should look like. But **assumptions most often backfire**. And betting real money on them is not something many people would want to do.

So there are three questions here:

- If you're building from scratch, how do you make **sure** you **deliver value**?
- If you already have an **MVP**, what are the best options for **moving forward**?
- How do I **minimize the risk** of ill investment?

Maybe surprisingly, the answers to all of these questions are similar.

How to build a value-adding MVP

In Eric Ries' book, *Lean Startup*, he perfectly describes creating a value-adding MVP. If you want to get to know the theory, we strongly recommend reading the book. Here, instead of giving you the *Lean Startup* summary, we'll share our practical conclusions based on over a decade of developing software.

Let's start with the most important insight:

MVP's goal is to test a hypothesis. However, when the development gets going, it's often the case that ideas for features accumulate.

When building software, there is a strong temptation to write down and plan every feature that comes to mind. This accumulation is quite a natural process but planning to chase after numerous ideas poses a couple of risks:

- The market can change, and the ideas may stop being viable;
- The funding can deplete before you get to something that the users will value;
- It may turn out that the ideas didn't pinpoint the users' needs.

There are three ways to go about these risks here.



The best approach: remember that you're experimenting

In this approach, you're doing things by the book (and a very concrete one: Lean Startup). This means that you overcome the natural tendency to plan ahead, and focus on building a small app that will allow you to **test** things like:

- **Do users need what you thought they needed?** Using Dropbox's example: instead of building a cloud file-storage service, they started with a landing page that described what Dropbox would do and placed an email subscription form there. This allowed them to test the idea cheaply and quickly.
- If the users need something slightly different from what you thought they needed, **what exactly is it?**
- **What is the most efficient way to deliver what users need?** How should the product behave for the best user experience?

You see how users react, and based on that you adjust. In short, MVP in this scenario is one iteration of the build-measure-learn cycle.

● Example of “the best approach”: Lean UX methodology

Lean UX methodology gives you a set of battle-tested tools to implement the approach mentioned above. The core of **Lean UX** is this: **everything is a hypothesis until you test it.**

To implement this core philosophy, Lean UX uses an analogous process to the aforementioned build-measure-learn cycle called: **the think-make-check cycle.**

Here are some pointers on how you could implement these phases.

● Step one: think phase

In this phase, your team **searches for possible areas of improvement.** The ideas for those should come from user feedback, user research, competition analysis, or product use observations. There are a lot of tools to obtain these insights, some of them being: Google Analytics, Hotjar, or direct interviews.

It's good to **have a structure** for this process. Hotjar, in [their blog post about Lean UX](#), has described a good example of such structure used in the Doodle company. And the way they structured the *think phase* was:

- Collect all the ideas for improvement;
- Compare these ideas to the overall company objective and

see if they match;

- Then look at Objectives and Key Results (OKRs) that sprout out of the overall company objective and categorize each idea based on these OKRs;
- Objectives describe things like “convert more users after a trial period”;
- Each objective has a couple of key results like “get users to create a poll during the trial period”;
- Assign ideas to specific key results.

This is just an example of how the *think phase* can be structured — it's always good to experiment and adjust.

● Step two: make phase

The make phase is the part of work most commonly associated with software development. Here, the production team tackles the ideas that have **the most potential in the core product's refinement**. In short: **a new feature gets developed**.

The process of choosing the most promising ideas to be turned into features is also something you should consider. Doodle, for example, was determining the impact, certainty, and effort for developing each idea, and based on this, they chose which ones they want to work on.

● Step three: check phase

Without this phase, there's no actual improvement of the product.

Here, you **validate** whether the **idea** you chose and the way you delivered it, actually **make a difference**. And what kind of difference. If the users like the change, you keep it. If not, learn why and **leverage this knowledge**.

There are many ways to structure the check phase, some of them being:

- **A/B tests** — you put up two versions of a new feature and see which performs better. You learn a lot about what works and what doesn't. A/B tests require a lot of traffic.
- **Guerilla tests** — it's not always possible to get a lot of traffic, especially in the initial phases of product development. Guerilla tests are much easier to conduct and give you quicker results, at the cost of certainty. You get a smaller test audience, like a focus group, and test how they respond to a new feature. You extrapolate the results.

● Summary of the best approach

In a nutshell, with the *best approach*, you use as few assumptions as possible, build an MVP, test everything with the users, and then adjust. Rinse and repeat.

● Second-best approach: develop the crucial features and test them

If, for some reason, you cannot approach the project as an experiment and you need to deliver a fixed idea, it's best to **identify** just a couple of features (or even one) that are at the core of the idea. In other words: **a feature without which the app couldn't even exist.**

Then, you deliver this small and sometimes rough around the edges app to the first users and see how they react. You learn what they like, what else they need, and what needs adjusting.

● Third-best approach: prioritize 20%, deliver, test

Some products are supposed to compete in **an established market.** In this scenario, the **users are used to a certain way of doing things,** and going against the current can be ill-advised. So naturally, there is already **a set list of things** a product should do and how it should work to ensure both **differentiation and familiarity.**

The best way here is to take the list of **features and prioritize 20%** of them. This will help structurize the development process and will still give **some room for feedback from users.** Because after they test the first version of the product, you can still adjust the planned features based on the users' feedback.

Researching users and conducting user tests

Before you start do anything else, you need to **validate the business model** you want to employ. There are things you can do without user research, like:

- Write down goals, assumptions, and hypotheses
- Assess the market size and share
- Research search volume and related terms

It's also important to define a **Product Vision** that's both ambitious and broad but, at the same time, will allow you to keep the product development (and user feedback interpretation) focused around a concrete goal. Product Vision, in a nutshell, **is a clearly defined motivation behind the efforts to deliver the product**, so it's a good practice to make it inspiring. For example, Apple's product vision is: "Bringing the best user experience to the customers through innovative hardware, software, and services."

But after you do all that, it's time to start reaching out to the target users.

In the initial phases of user tests and research, it's crucial to **keep**

things minimal. It would be easy to spend large budgets on surveys and research companies but this is rarely necessary. Sometimes, just a couple of people suffice. Think of the target users you want to get, reach out to them, and ask them for feedback. You can find them in community groups, or if that's not an option — you can always ask your friends to test-run the MVP.

You'll get some suggestions for improvements for sure. And here's the first trap to watch out for: **you should not implement everything the users suggest.** The goal here is to see if your MVP satisfies the business need and if there's anything to improve in the user's journey toward fulfilling that need.

A priority matrix with axes "importance" and "frequency" will be helpful here. After you place all the findings on such a matrix, the next step toward refining your product should be easy to find.

This doesn't mean that you should throw the majority of the user feedback through a window. It means that some things can simply wait. To use an example from Eric Ries' *Lean startup*: after Kodak Express gathered initial user feedback for their new product, they saw that they already had some suggestions in their development backlog. It meant they they were on point with the product's idea, and they should prioritize what next steps will refine the core of it.

User feedback is not a one-off exercise. The best practice is to do it continuously (it's a practice known as continuous discovery). You should run user tests after each major release, and use the feedback

to structure the next phase of development. To not get sidetracked during these cycles, you should always compare the user feedback to your core value proposition. **The things you should develop next would complete two requirements:**

- They come out of the user feedback
- They are the ones that will help refine the core of your product the most

Even if the user feedback tells you to pivot, a core value proposition should always spearhead your development efforts.

It's worth stopping here for a moment and splitting the testing into two groups: the actual *user testing*, and *usability testing*. User testing, as mentioned, allows you to verify what exactly the users need and if you're on point with your product, and what should be done from the get-go. Usability testing shows you if and how you need to adjust how users accomplish goals inside your product — these can be run at later stages, and can be done with minimal effort (5-7 participants are sometimes enough). But both tests are essential for developing a quality product and should be done continuously.



What to watch out for

As outlined above, one of the most common traps, is chasing the temptation to develop as many features as possible. Of course, you should value the users' feedback, but it's best for you to prioritize the next development steps and ensure they match your core value pro-

position (which rarely means “extending it”).

Another common pitfall is starting the quantitative tests too early. During the MVP phase, most probably you won't have enough users to run reliable quantitative tests, so you should stick to the qualitative user and usability testing.

So if we were to give one sentence on the whole topic, it would be:

User tests are not for extending an MVP but for validating it.

GRZEGORZ HAJDUKIEWICZ
CHIEF DELIVERY OFFICER AT MONTERAIL



Keeping the development process efficient

There are four core elements to keeping the development efficient:

- Users' feedback
- Continuity
- Elasticity
- Proximity of the development team to the users

We highlighted the importance of the user-generated feedback in the previous subchapters. This is the key variable to consider when planning next steps in product development. Without listening to this feedback, we risk building an isolated vision that won't be shared or used by anyone.

Continuity means that you **repeat the build-measure-learn cycle indefinitely**. As long as your product is used, there will be things to improve. Instead of thinking in phases, it's better to view the **product development as a process**. Use the feedback you gather from the users to build your backlog from which you prioritize a couple of features to develop, and when you test them, a direction for the next couple of steps appears.

From the business perspective, continuity means that product deve-

lopment works best with a monthly budget, not a fixed one.

Elasticity means that assumptions and vision are fleeting. It may be that the users tell you that a pivot in strategy is necessary.

Proximity of the development team to the users is also important. When the entire production team has a lot of insight into user behavior, feedback, and experience, they can challenge the status quo, look for new solutions, and take initiative. They will fuel product development discussions and bring in a unique perspective. Most of all — if they see that they do something that matters to people, they will be that much more engaged.

Preparing for a big launch

The reality is that **very few digital products** nowadays have a *big launch*. Sometimes releasing an application to a mobile store could be considered as such. Still, in practice, this is just the next step in development, where the user base starts growing.

So, in line with the philosophy of previous chapters — we advise avoiding **thinking about big launches**. Product development is an iterative, continuous process, and any big milestones, jumps, or launches from the production team's perspective are a risk because it means some things are untested, unvalidated, and unconfirmed.

A *big launch* could be a milestone for the marketing team, but there's also one caveat. Do you really need a big marketing launch? Maybe it's better to start with a soft launch, test usability, performance, and reception, and then iteratively increase the marketing reach?

You may ask: if everything in product development is continuous and iterative, and big milestones and launches should be avoided, **how do I know if we're making progress?**

The answer is: create **well-designed KPIs** that will help you measure the health of your product. Good KPIs are much more than down-

loads, visits, or conversion rates — they depend on the situation and your position inside your niche. Finding the right KPIs can take a couple of iterations, but it's an effort that pays off manyfold.

In a nutshell, we believe that instead of *big launches*, the best way to develop a product is to have iterations that grow your user base and refine your product. This way, no team will be overwhelmed with sudden jumps in workload, and by that extent — the possibility of human errors will be reduced to a minimum.

We're Monterail

A full-service software agency

We hope you'll find this short booklet helpful and informative.

If you need help with defining KPIs for your product, a reliable development partner, or someone to help you refine next steps, we're here for you. Just scan the QR code below and leave us a message, we'll get back to you before two working days pass.



www.monterail.com

