How to build a SaaS side-hustle that actually makes money

> **For Programmers and Hackers who can build software independently.**

About Me

Hello, my name is Quinston Pimenta. I am an Engineer based out of Pune, India. I build internet products for a living. I've been writing code for over a decade and what differentiates my journey from other Engineers is the fact that I pay attention and consider the business side of the equation just as much as writing the software. I own a piece of all the products I build and all of them make enough money to be profitable on their own.

Preface

This is not a technical programming book but a write-up on how you—as a Programmer, Tinkerer or Hacker—can go about building a SaaS Product on your own and the general principles (guidelines) that you need to adhere to. The aim of this book is to be super dense and anti-verbose. It is a culmination of all of the concepts that I've learned over the past 8 years. I will be updating it periodically while maintaining a changelog as we 'optimize' to the most condensed version. Let us begin.

What is a side hustle?

I know a bunch of people who would be turned off when they see someone using the term 'Side-Hustle'. The problem is - I can't seem to find another term that describes what I'm about to describe, better. A side hustle is a something you do to supplement the income from your full-time job. That's the simplest way I can put it.

There are situations where you, as a programmer, run a side hustle purely for creative fulfilment and not for cold hard cash - and that's fine too. Some projects which are super interesting to build, might not have the same potential for commercial success. In this book though, we are not focusing on non-commercial projects - but on products that can actually bank you some cash.

Disclaimer: Building a SaaS product that actually makes money is really really hard. Painfully hard. So, don't expect any quick fixes or immediate results. This is a gruelingly painful task to actually achieve. There is a reason why cash-cow SaaS companies are so rare.

The bright side is that, if you actually make it to the other side, you build equity apart from just the free cash flow you generate every month. Equity is a commodity that can be traded in for resources. You can actually sell it to

someone who would pay up-front for it's ownership. You get yourself a payday and the buyer gets to keep your side hustle. This happens all the time (sometimes even if the product makes no revenue). There are entire websites that exist on the Internet where you can find buyers. So, you can do a bunch of cool things if you actually get this right.

Another Disclaimer: Do not quit your job to follow a SaaS side-hustle. I cannot stress this enough. In my opinion, you should build the SaaS product side-hustle in tandem to your full-time job. If you have wealthy parents who can pay for everything, that's great—you're privileged but for the rest of us, do not quit your full-time job. This is to supplement your income, not replace it.

Expectation Mind-set

Now, as programmers, we're used to being paid up-front per hour based on the amount of work we do. This inculcates a sense of security and entitlement. You have the voice in your head saying, "Hey, I wrote code for 20 hours straight - I MUST GET PAID FOR THIS!". Although this might be a reasonable assessment, that's not how entrepreneurship works in general.

You get paid for the value you provide. Period.

All inhibitions about not getting paid in proportion to the amount of work put into development should be thrown out, especially in the early stages. We all know that product development is incredibly expensive - which is why this book is geared towards programmers, so that cost of development is compensated with time.

Context Switching

If you're working a full-time job, you'll need to find a way to switch thinking patterns when working on the side hustle and as we all know, context switching is expensive. One way I've found to alleviate this issue is by **not**

working on multiple products per day. This might be difficult as everyone's schedule is different and a lot of programmers are required to come into work everyday - but this is the most effective method I've come across.

Sleep resets contexts when you wake up in the morning, you can focus on one thing until you go back to sleep. Next day you wake up, you can focus on another product. There is zero overlap. This is, of course, the best case scenario. In a lot of cases, you're going to need to focus on tasks which are queued up for different products. Unavoidable scenarios like immediate customer service is well, unavoidable. I personally dislike any sort of scheduling or deadlines - as I like to take my sweet time building what I'm working on. So, experiment around and find your sweet-spot. There are no wrong answers.

Frustrations

To put it lightly, you're going to be frustrated a lot and you're going to get angry a lot. There are a lot of reasons for this apart from just your stack overflowing - from the cuts you have to give to hosting companies, not getting the right domain name to simply realizing that no one wants what you've been building for the last six months. There are no solutions for this and I'm not a qualified professional to help you with your feelings - all I can do is offer you solace in the fact that it sucks. Trust me, you're not the only one who feels that way. I have perpetual nightmares about what-if-this-or-that-happens. Again I'm not a qualified professional, but what I've found to work in my case is to just trust myself and trust the decisions I'm making. You can't worry about things you can't control.

You don't need a Business Plan

You only need a business plan if you're building a conventional product. If you're building something that is completely untested in the market, you don't need a business plan. Mainly because, you have no idea what the metrics really

are. It's pointless to stick to a string of random numbers you threw up on the board because someone told you that you need to have a business plan.

Do not put too much of your ego/identity in the Product

You are not your product. Internalize this. You are not your product. You might have built the product but it is a disconnected entity on its own. You are responsible for it to work, but you're not worth less as a human being because it failed to function. The feedback customers give you is about the product—not about you as a person. In fact, I encourage you to bathe in the disdain of your customers. Revel in their annoyance. That is the only way you can fix what is broken.

If you're too scared to ask the customers why your product sucks, you're not going to get anywhere. If you get cold feet because someone left a scathing review, how are you going to turn the ship around? Every battle is won before it's fought. Pragmatism over Idealism.

Work Environment and Habit Formation

You might not believe me, but good habits and good work environments go hand-in-hand. To create a good habit, you have to create an environment where those habits can be executed with minimum friction while doing the opposite to get rid of a bad habit. That's how the work environment plays a part in your efficiency. If your work environment is not optimized for you to be at your best, the amount of willpower that you will need to exert will be massive.

Willpower is a muscle, and if you work it too much, it will need time to rejuvenate. The solution is to design your work environment such that it reduces friction and maximizes willpower. I use a Mechanical Keyboard as I like how it clicks and feels under my fingertips. The reason I buy expensive equipment is that it helps me do more. 95% percent of all SaaS products can

be built using a single laptop. You don't need expensive hardware, but using expensive hardware will help you maintain your sanity and get you to your destination faster and more efficiently.

Let's jump into talking about how to find what to build.

How do you know what to build?

The common notion that the first idea you get will be incredibly successful is as flawed as time is old. In reality, you have to cast a wide net and try to figure out which idea you can execute on your own that customers will want while also having a sustainable revenue model.

In Blitzscaling, Reid Hoffman talks about three items, without which your SaaS will fail.

1. *A killer product.*
2. *A clear and sizeable market.*
3. *A robust distribution channel.*

Problems from your Perspective

To give you a personal example of how most of the products I work on came about, I'd like to tell you about my YouTube Channel and TimeBolt. I was a second-year engineering student in Pune, India when we were being taught the Dijkstra's Algorithm. I was unhappy with the way the code for the algorithm was being taught and could not bear to see the pain it inflicted upon the other students (I'm dramatic). So, I made a code explanation video and uploaded it to YouTube. This snowballed into me making 100+ videos on DS&A (a lot of which were after I graduated).

I have a stuttering issue - so I usually can't speak in straight sentences. My slow speech became a massive issue when it came to editing the videos, because of the constant spaces between each sentence I spoke. I was also learning about Data Signal Processing at the same time coincidentally, hence I knew a little bit about how to process audio signals. I sat down one day and started writing the script for a program that would edit my videos and remove these spaces automatically. All the tools I needed were already available to me, coincidentally and I finished writing the program - and I didn't use any AI. I just wrote a pure mechanical algorithm.

The issue was that I didn't know how to package this up for distribution - nor did I have any concept of entrepreneurship at the time. I just used the script for my own videos for the next year. When I got out of college and started my first company, I realized that I could actually monetize this algorithm. So, I wrote the script again - but this time in a format that could be distributed and sold as a product. In its current form, the algorithm lives inside TimeBolt.

The point is, you can find problems in your own life, or in the life of someone you know, that you have already identified solutions for and can monetize those. If you have identified a problem, there is a very high probability that someone else also has the same problem—this is the lowest hanging fruit that you could go after.

The other way to go about it is to check out forums on large platforms like Premiere Pro, Apple etc. where people usually leave complaints about what they don't like about the current software. These complaints are opportunities in disguise. They are indicators of success that need to be optimized. If a customer is spending time complaining about a product, the assessment is not that they don't want the product, it is that a few changes in the product might bring the customer over.

If the customer didn't want the product, they would never complain about it!

The issue with fetching ideas from the forum is that you might not be interested or passionate about solving those problems. This might not seem

like a big deal, you could be thinking, "Oh, I'll just write this software and sell it." Well, even if you get to the point of writing and selling the software, you're going to have to do customer servicing and if you don't enjoy the product you built, you would probably not know a lot about the field in general - and that would lead to the customer not trusting you because of your lack of knowledge. That is an instant disqualification.

But if you find something from the forums that you're actually passionate about, well, I'd say run with it. Your interest in the product coupled with the problem being widely accepted (as a problem) gives you a certain amount of validation - that this is something worth going after.

Read a lot of Blogs and watch a lot of YouTube Videos

Culture can be monetized. Programmers and rappers are pretty similar in that sense. They both capitalize on culture. And today, culture is found online - in blogs and vlogs. Very few people actually read books, they watch YouTube videos and click-bait off to spicy blogs. The videos and blogs I'd like you to watch and read are about technology, content creation, sales, and marketing. Most of the ideas I've had came from just watching a lot of YouTube videos. Not every video is filled with gold, but once in a while, someone makes an off-hand comment that gets stuck in your head.

Creativity needs Inspiration

Everything is built on top of something else. Name a technology, and I'll name a precursor. Everything has originated from something. The point I'm making is, you're not going to be able to come up with an idea out of thin air. It's next to impossible to have that sort of cognitive excellence. In most cases, your idea is going to form based on what you've previously perceived from an external source. Nothing is original, everything is inspired. Your brain needs input to process the output. No input will result in no output. Even Machine Learning Neural Networks need to be initialized with random values. Creativity is expensive.

Feature-sets and Wide-nets

The first SaaS 'Minimum Viable Product' that you build is most likely not going to hit. That brings up the question of whether you should be building the MVP in the first place. There are some products that you simply cannot build MVPs for as their use cases are too complex - but those are not the kind of products we're trying to build as a side-hustle anyway. By definition, you should be choosing an idea that you can build an MVP for by yourself.

The feature set for such a product will need to be **very narrow** for you to be able to do so on your own. You definitely won't be building something like TikTok or Instagram. You'd be building something like The Levelator or a plugin for a larger platform like Salesforce or Premiere Pro.

Think: How do I build something that does one thing really really well?

In addition to this, you should build a few MVPs (cast a wide net) - and see which one performs the best in the market. The metrics you can use for gauging interest are page views, downloads, customer complaints (my favorite), etc. and the holy grail of all metrics - revenue. If a few people purchase your MVP, that's an instant winner (yes, you charge money for your MVP on Day One). If your MVP fails at all these metrics, its *no bueno* (not good). In a lot of cases, customers will email you (or you will email the customer) and you will ask them, "Why doesn't this work for you? And what can we do to make sure it does?". If this scenario comes up—there is hope and you're on the right path. In cases where there is just dead silence and zero activity, it's time to try something else.

Target Audience

The target audience is probably the most important item-set in this entire exercise. Who is your customer? Who are you building the product for? You

need to imagine this person in your head. Make a mock-up of what they do for a living, what tools they use in their daily lives, and how they go about their day. How does your product fit into their workflow? Before you even write a single line of code, you need to know exactly who will potentially buy from you and why they will buy from you. You don't have to build a 'Customer Persona', but a general idea of where and how you'll reach your target customer is important.

If you're building a photo editor app, your customer is probably a teenage girl who posts photos on Instagram—you have to think, okay, does she have the funds to pay for this product? Answer incredibly simple questions like these by imagining yourself in their shoes. A lot of times your answers might be biased by your experiences but that's okay. You can rely on actual data when you start testing and optimizing your stance.

Incrementally Better vs Exponentially Better

In Zero to One, Peter Theil talks about two kinds of new products -

1. *Products that do the job incrementally better than it's competitors.*
2. *Products that blow the competition out of the water by doing something exponentially better.*

If you're building a SaaS product as a side-hustle, and you build something that does an incrementally better job than a competitor, I'm sorry to say—but the product isn't going to get anywhere (unless the service you provide is exponentially better and cheaper than the competition - which is still a stretch). You have to get creative here, think out of the box... be unconventional. It's hard - but not impossible.

You might be thinking, "Quinston, building something exponentially better than the competition is next to impossible!" Yes, that's true which is why you don't build a product that has existing competition. You're supposed to build something that has zero competition! Whatever you build is then automatically

exponentially better because no alternatives exist. This is not a programming skill, this is creativity.

Don't do exactly what customers tells you to do

The customer does not know your product inside out. You do. Just because a customer (or potential customer) told you that X should be like Y. Always take that with a grain of salt. In most cases, you don't have the infrastructure built out to track real-time customer interactions with your product. So, don't be afraid to make the change suggested by the customer if you think it will help, but at the same time be prepared to roll back the change if someone else complains.

Remember: You're building the product for a lot of people, a feature that's positive for one customer might actually be a negative for another customer. At the end of the day, trust your instincts (but only if you're strapped for data).

The 80-20 Principle

The 80-20 Principle is highly relevant in this scenario. In Product Development, it's a common notion that 20% of the product gives you 80% of the results. My whole thesis revolves around building just 20% and outsourcing the rest 80% to other SaaS products. You outsource the surrounding infrastructure but build only the core feature. For example, using Firebase User Authentication instead of building your own. It might not seem like much, but building the surrounding infrastructure takes a massive toll on your brain - simply because it's not a core product feature but if it breaks, well - you're f*ed. One more reason to outsource infrastructure is also that you're not superhuman and you have limited time on your hands.

Most infrastructure SaaS products are free to use up to a certain limit and pay-as-you-go after that. The architecture you design will obviously decide

what infrastructure products you use but the core philosophy remains the same—

Outsource everything that is not essential to the product.

Business Attribute Considerations of the Operation

I'd like to talk about a few concepts that are very crucial when it comes to thinking about the business. The first two concepts are...

1. Linear vs Non-Linear Returns.
2. Potential to Scale.

Let's take an example of a Product Company vs Service Company to illustrate these points.

A Product Company is a company that owns the Product (it necessarily does not make it). A company that makes products for other people exchanging money is a Service Company. A very important distinction. A product is highly replicable as you don't need to change major aspects of it for every customer. Hence, linear input of effort into a product has the potential to yield exponential returns - simply based on low (or non-existent) replication costs. Ex. 1x input effort can have x^100 returns.

The same is not true for Service Companies. Services have to be fine-tuned (or built from scratch) for every single client. Executing separate projects for each client is operationally heavy and is a linear exchange of resources. A Service Company is paid based on the time spent creating the Project - and time is not an infinite resource. The returns expected from a Service Company are linearly tied to the input effort. The expectation is mostly a multiplier. Ex. 1x input effort can have 3x returns.

Scale is only achievable if replication costs are low - which is true for a Product Company but not so much for a Service Company. Massive value is always created at scale.

Massive returns also come with massive risks. It's always less risky to run/work for a Service Company vs a Product Company.

If you want a stab at exponential results - build a Product Company that has high margins and low replication costs, and if you want to have a certain amount of security (less risk) - build a Service Company.

The point is—your SaaS product should function with minimum (or zero) interaction from your end. Operations of the product should be independent of you—unless there is an error in how it works and a customer raises a query—if you want to also have the time to go to your main job. Or else, the side hustle will consume you and you won't even make a significant profit. Everything from sign-up to payment to exit emails should be automated.

Competition

You might have read on the Internet that you should do a 'Competitive Analysis', and you should if you're building a conventional product like an e-commerce store or a photo editing clone app but if you are building something where a competing product does not exist—how exactly are you going to do a competitive analysis? And if I had anything to say about it, do not build a SaaS product that already has existing competition—unless you can build something that is ten times better and ten times cheaper. Always remember, creativity is expensive.

This reminds me of Peter Theil's famous saying, "Competition is for losers." All I'm going to say is, don't pay too much attention to your competition—but hyper-focus on your own customers. Improve your product to fit the needs of your target customer and keep finding more of the same. You are not going to be able to build a fully fleshed-out product like Google Sheets because you

simply don't have the time and resources. This brings us back to the core philosophy—build something that does one thing exceptionally well.

Do you have the skills to build this?

Building an entire product from scratch can be incredibly daunting even if you're an advanced programmer. There is just something about building the entire pipeline from the front-end to the back-end that gives you the chills (I'll have to decide the color of the button and the backend code to execute it when it's pressed?!).

I'll give you an example of how I approach the skills problems in my own products. Let's talk about Farsight X. The core feature set of Farsight X is a 360 Guided Tour. The reason I was able to build it, was because I knew how to use Blender! Using Blender from when I was a sixteen gave me an understanding of how the Computer 3D Space worked, and then all I needed was a tool that would help me manipulate it (three.js). It did take more than a year to fully build out Farsight X though, even with all that knowledge, as three.js has a learning curve.

Do not fixate on the technology before you decide which product you're going to build. This usually involves learning whole new programming frameworks as you're building the product. On the positive side, using the technology you're building with immediately after learning it—will reinforce the knowledge you've learned significantly. I dislike learning a framework and never using it in production. That's almost as good as never learning the technology in the first place! So, don't be scared. It's okay to make mistakes. Dive in!

Writing horrible code is a pre-requisite to writing *spectacular* code.

Product Development Learnings

This section is an outline of all of the product development learnings I've personally had in my journey. The whole point of this book is to be as concise and to-the-point as possible and that is exactly what I've tried to do here.

When do you launch? As early as possible.

As early as possible. Even if your code is broken and doesn't work, list it for sale and give people a refund if they don't like it - while you keep making it better. No, it's not going to ruin your reputation as an Engineer. If anything, it will save you a lot of time and help you realize whether you should be spending all this time building the product further. Reid Hoffman in Blitzscaling says, "Launch a product that embarrasses you." Thankfully, embarrassment is not the worst-case scenario. The worst-case scenario is that no one notices.

Lean into Validated Learning

In the book, The Lean Start-up, Eric Ries talks about the concepts of validated learning. Validated Learning is the assessment generated from running the **Build-Measure-Learn** cycle.

Yes, in that order. You go about building the product i.e., the MVP in our case, measuring whether customers use it or not and make assessments based on the data collected. With the assessments, you decide whether you have to keep the product as it is, improve it or end it. This can also be applied to individual features inside the product.

You can never predict before hand what the market would react like, and people who pretend to know the future should be ignored. No one knows the future. All you can do is get the product out there, see how people behave and make assessments based on the data.

Collection of data is hard, hence sometimes you'll need to make an instinct call. In these cases, you must trust that you're making the right call but, once

you have realized that the feature you just pushed actually made the product worse, immediately rollback. There is no shame in accepting defeat - and in this case, it's not defeat... it's just learning. Validated Learning.

Building a fresh, new product is like running a science experiment.

1. *You have a hypothesis.*
2. *You run an experiment to test the hypothesis.*
3. *You come to a conclusion.*
4. *REPEAT*

The experiment doesn't just have a single pass, you do this over and over again until the Laws of Diminishing Returns kick in. Continuous improvement (in the right direction), baby!

Optimize for X, not Y, Z or K

One of the biggest blunders you can make is optimizing the wrong problem statement. When you launch the product for the first time, early adopters will sign up for a very distinct reason. Find out what that reason is, because if you don't - you will not know what problem you need to optimize for. Product Development is an iterative process. You have to improve upon what you previously built but the direction you take while you do it is equally important. If you optimize for the wrong problem statement, the reasons customers signed up for - will disappear.

How do you actually optimize the product? Based on what the customers have told you and seeing their behavior, imagine a customer using the product - write down their flow, their journey. Their experience from landing on your page, to signing-up to actually using it and then exiting. Their experience is going to be filled with ups and downs. The first most important thing you need to do is to get rid of all the downs.

If you get rid of all the downs, everything that's left is up!

The downs are breaks in your product flow, where you lose customers. Imagine a bridge that links the customer from one phase of the product to the next. If there is massive friction moving from one bridge to another, there is a high likely hood that the person is not going to last very long.

Your objective to get the customer from point A to point B as fast as humanly possible and with the least amount of friction! Once you fix the broken bridges, you can then improve them. Make them stronger and more resilient.

How fast? AS FAST AS HUMANLY POSSIBLE.

At the initial stages, you are going to push out product iterations incredibly fast, sometimes multiple times a day! Mark Zuckerberg, at the start of Facebook, followed the phrase, "Move Fast and Break Things" but later on it became "Move Fast With Stable Infra". As funny as that sounds, it's a general progression over time towards more stable infrastructure. As the product matures, you will need to make lesser and lesser updates—it will reach peak optimization.

Product-Market-Fit

In a YC Podcast, Michael Seibel said that product-market-fit happens when money starts piling up in your account! You can read more about product-market-fit in detail about it on the Andreessen Horowitz Blog. According to Andreessen, “product/market fit means being in a good market with a product that can satisfy that market.”

I, personally, think about this from the perspective of how a Machine Learning Model 'fits' a Dataset. The Machine Learning Model is your product and the Dataset is your market. If you've learned even a little bit about Machine Learning, you'll get what I'm talking about. Your product should fit the market.

Not under-fit it, not overfit it - but fit the market perfectly. If you under-fit the market, your product will lack a bunch of features that will lead to customers leaving (leading to a higher churn rate). If you overfit the market, your product will become super niche and won't generate enough revenue.

The key is to listen to your customers, but at the same time not really listen to them. A single customer, like a single datapoint in a Dataset will want you to add a feature that may not be favourable for the rest of your user-base. If you fulfil this request, you're essentially overfitting the Dataset. In another situation, a lot of customer will ask for a feature which you don't think should be added to the product, this is a case of under-fitting the Dataset.

If a lot of customers are asking for a feature, there is a high chance that you should implement it. Personally, I use customer complaints as indicators to prioritize which features to build. To claim to know exactly what the customers want, is to be entitled and egoistic. Product Development has no room for that. I like to build the feature, test it's effectiveness, and then make a conclusion. Everything is a science experiment backed by data.

Value Hypothesis and Growth Hypothesis

The Value Hypothesis and Growth Hypothesis—which you can read about in more detail here—might sound very similar but they are very distinct assumptions of the same product. The Value Hypothesis answers the fundamental question of whether your product truly provides value to the customer. Value could mean a lot of things, and usefulness is a subset of value. The Growth Hypothesis is an assumption that you make regarding the strategy of growth that you would employ to scale the product to more customers, who will receive similar value.

The thing to pay attention to is that—both of these assumptions are, well, assumptions. They are hypotheses, which means they can be proven wrong with data. So, your Value Hypothesis and Growth Hypothesis can change over time as you continue to 'fit' the market. So, don't be hell-bent on having a

single one of each, you can have a bunch of them for your product and evaluate each one over time with more customers and more data.

Value Threshold

The value threshold is the positive amount of value (as stated in your value hypothesis) that you're providing to your target audience. Usually, when we try to create a SaaS product, we are essentially trying to automate a task that is tedious and time-consuming. Zapier did that with Lead Forms (among other things) and TimeBolt is trying to do that with video editing. The problem is that if the task you're trying to automate can be done with a few clicks in software that already exists after watching a YouTube video, that's not really a task worth automating. Simply because very few people will actually pay for it. The value threshold of your product must be significantly higher to create a dent in the concrete.

Revenue Vs Expenses

For a well designed SaaS product, the more it scales, the higher the expense - but it also makes a larger profit. Why? Because revenue grows disproportionately faster than expenses. You don't have to think about this at the very beginning but as the product starts generating revenue, think about which costs grow in tandem to scale and which costs grow linearly with scale. Ideally, your expenses should grow linearly, while revenues grow exponentially. Something like this.

How to visually design the product?

I am not a visual designer. I personally have zero skills in even deciding which color a button should be—but you know what! A lot of customers don't care about visual design as long as the product does what it is supposed to do. Visual Design is probably the last thing you need to worry about. All you have to do really is design the wireframe and build out the front-end code that

accomplishes the wireframe. Don't be afraid to use Bootstrap or MaterialUI. In fact, I would recommend using those frameworks and saving all of that time, rather than tinkering with custom UI code. Remember, your job is to validate the existence of the product, and if people would pay for the service rather than making it look slick on day one.

Once the product starts fitting the market, you can invest the money you made into hiring a professional to design a UI which you could then implement, but don't waste your time and resources doing all that initially.

If it takes too long, it's probably wrong.

In an Interview with the Everyday Astronaut, Elon Musk dropped a statement that blew my mind. "If a design is taking too long, the design is wrong." It's such a simple statement but has such strong connotations. This was a philosophy used to design literal spaceships, so I think most of our bases are covered here. Whenever I've encountered a computational problem that seemed impossible to solve, there has always been a simple solution hidden in plain sight. Find that simple solution.

Think Workflow, not Tips and Tricks

Build your product around workflows, which you design for your customers to use. Don't tell the customer that there is an obscure tip or trick that they can use. Think about how to get the customer from point A to point B with the shortest path possible, while maintaining all of their sanity. You don't want a hundred dialog boxes flying around. The flow must be singularly directed. One workflow to achieve a single result. Imagine writing a function, you're not going to write one that returns three separate values and does three different things. Similarly, imagine the product like a cohesive combination of functions that the customer can call. Every possible output value has a specific workflow. Don't confuse the customer. Confusion increases the churn rate.

Perfectionism is your Enemy

You're never going to build a perfect product, and honestly, why would you want to? The 80-20 Principle clearly states that 20% of effort gives you 80% of the results. Why would you want to put in the rest of the 80% effort to get 20% of the results? That's perfectionism. And that's how you overfit the market. Avoid it. You need a product that's good enough and gets the job done—at least at the start. When you have millions of dollars in funding, sure, blow all of it on getting the last 20% but you have no business of doing that when you're trying to get your initial monthly revenue up. In most cases, your product is probably failing because the value proposition isn't strong enough, and not because you have fewer features. But yes, whatever your product says that it does, it should do that one thing exceptionally well.

Don't be afraid to shut it down

If it's not working, and you know it's not working. Shut it down. There is no point in dragging something out that has zero prospects. In fact, it's a blessing. Now, you can take all of the attention that you were focussing on a product that is failing, to something that has new hope. Give the failing product a nice burial, and move on to greener pastures. You are not your product. Do not tie your self-worth to it. Let it die in peace. I can't tell you when it's the right time to shut it down. You will know in your gut—when it's time to kill your baby.

How to find the right pricing model?

Pricing is going to be a very crucial part of your whole operation. Incorrect pricing is death-blow to great products (as I have experienced first hand). It sometimes happens that customers love using your product, but they simply don't have the budget to pay for it - and you can't afford to reduce your costs because of backend expenses. This situation is one that should be absolutely avoided.

The Value Cycle

A lot of times, even though we can charge more for a product, we tend to keep the prices low (or give it out for free). We fear that if the prices are perceived to be too high, users will shy away. There is a concept discussed in Zero to One by Peter Theil, that you can use to justify raising your prices. Your product generates value, you can think of this value in the form of units.

Scenario 1

Let's say your product generates 100 units of value. For generating 100 units of value, your cost is 100 units of value. The value you captured in this case, is 0 units (100 - 100). If you generated 0 units of value for yourself, what are you going to invest in improving the product?

Scenario 2

Let's say your product generates 100 units of value. For generating 100 units of value, your cost is 50 units of value. The value you captured in this case, is 50 units (100 - 50). You can now put away some of the money and invest some in the product. You can improve the product so that you can generate more value for the customer, and in turn, capture more value and then repeat the process.

The value cycle has to output a positive captured value for your operation to get anywhere. If you don't capture value for yourself, you're actually killing the product. I once witnessed one of the craziest things ever on Product Hunt with a product called Sheety. It's a product that lets you turn your Google Sheet into an API instantly. I was going through the comments section of this product and users were literally asking the creator where they could send money because they were worried the creator would abandon the project if it didn't make any money (as it was free to use)! I found that absolutely fascinating. It convinced me that if your product truly provides value, no price is too high. Capture as much value as you possibly can.

The product is worth what the customers are willing to pay.

So, don't be afraid to charge the amount that you feel the product is worth to the customer. You can always tweak it as the market responds.

Pricing Models

There are essentially three pricing models that you can go for depending on the kind of product you're building. They are,

1. *One-time Purchase*.
2. *Fixed-rate Subscription*.
3. *Metered Rate*.

Although I've mentioned these pricing models distinctly, feel free to combine them in various permutations to make the best one for you. At TimeBolt, we have a combination of Fixed-Rate Subscriptions and if the customer chooses, a Lifetime License. At Farsight X, we have a Pre-paid Metered Rate. There is no one shoe fits all pricing model - it totally depends on what you're building.

I like to choose the pricing model based on the graph of my backend expenses. If my costs go up significantly based on the customer's usage of the product, I'd charge a metered rate. If my prices are changing ever so slightly by usage, I'd charge a fixed-rate subscription. And if it's a product that doesn't have any backend cost, the one-time purchase seems perfect. Always try to pass on the benefit, that you receive back on to the customers.

Customers are not stupid, they can see through you. If you do good by them, they will do good by you. Imagine yourself in their shoes and make the call. The common notion that customers should be swindled to make money, will ruin your product in the modern era. One scathing review about how poorly you handled a customer's complaint can end you. That single review will haunt you for life. A review about your product sucking can be survived, but if you

mistreat a customer—hell will be wrath upon you. Offer a customer an immediate refund if anything goes wrong (no questions asked), or if you feel like there is a threat. It will be your saving grace.

Getting the first sale.

Congratulations! If you've made it till here, I'm going to assume you've built an MVP that you'd like to start testing out. The question is where do we start? We must do a mindset shift here. We have to stop thinking like engineers and start thinking like marketing/business professionals. I know it feels like going over to the dark side, but that's the game we're playing!

Creating Marketing Collaterals

Before listing your product on the internet, you need to get a few bases covered. You need marketing collaterals. Now, if you wanted to spend a bunch of cash, you'd hire a designer and get all of these made for you but I'm going to assume we're on a budget here. The next best thing is to make them yourself (apart from maybe the logo, which you can get made for $5 on Fiverr).

When I was figuring out how to structure the content that I'd need... I looked through a bunch of books about copywriting and found one that covered all the bases in one framework! It's from the book "Building a Story Brand: Clarify Your Message So Customers Will" written by Donald Miller. The framework is called BrandScript, and this is what I use to structure my content for SaaS Products. I can't tell you how much I recommend reading the book—especially for programmers who have zero inklings about marketing.

The BrandScript reads like a plot-line from a movie where the customer is the main character and you're the guide - and that's exactly the way you want your customer's experience to be like. Everything is part of the product. From the customer coming onto the landing page to signing up, to payment to

accomplishing the task... it's all a storyline. A script. A BrandScript. This tool gives you the structure to write your content. You can use this structure and then go about building your landing page and all the corresponding material necessary.

The framework is as given below - all rights for this framework are reserved by Donald Miller.

A Character...

WHAT DO THEY WANT?

What do your customers want as it relates to your product or service?

→

...has a Problem...

VILLAIN

Is there a root cause of your customers' problems? Can you personify this root cause as a villain? What is the villain in your customer's story?

→

EXTERNAL

What is a problem your customers deal with as it relates to your product or service?

→

INTERNAL

How is this villain making your customers feel?

→

PHILOSOPHICAL

Why is it "just plain wrong" for your customers to be burdened by this problem?

→

...and Meets a Guide...

EMPATHY

What brief statement can you make that expresses empathy and understanding?

→

AUTHORITY

How can you demonstrate competency in solving your customer's problem?

→

...Who Gives them a Plan...

PROCESS

Are there 3 or 4 steps your customers can take that would lead them to a sale or explain how they would use your product after the sale?

→

AGREEMENT

List the agreements you can make with your customers to alleviate their fears of doing business with you.

→

...and Calls them to Action...

DIRECT

What is your direct call to action?

→

TRANSITIONAL

What transitional calls to action will you use to on-ramp customers?

→

...that Helps them Avoid Failure...

List the negative consequences your customers will experience if they don't use your product or service.

→

...and Ends in Success.

List the positive changes your customers will experience if they use your product or service.

→

Character Transformation

From

How was your customer feeling about themselves before they used your product or service?

→

To

Who will your customer become after they use your product or service? What is their aspirational identity?

→

The Name

I don't really have a lot of things to say about the name, apart from the fact that you must keep it clean. No sexual innuendos. No swear words. No weird references. A super short domain name would be incredibly expensive to purchase but you can combine a few words to create a unique new word that probably isn't taken. TimeBolt started out as Silence Trimmer. For purchasing the domain name, I personally use Google Domains and use Netlify DNS to route DNS entries. It's a massive plus if you can reserve social handles for Twitter, Instagram, etc. You don't need to post anything there necessarily, just create an account with the handle to reserve it.

The Logo

If you're a designer and you can make this one yourself, go ahead and get it done. If that's not the case, you have to find someone to do it for you. The best place is Fiverr. You can get a logo made for $5. It's an absolute no-brainer. Make sure you get a PNG file with a transparent background. You're going to need that when you list your product on Product Hunt.

Components of the Landing Page

The landing page, in our context, is the page that customers are going to land on when they open up your website. Traditionally, a landing page is different than the homepage of the website, and is directed to specifically from an ongoing marketing campaign - but we don't have time to create 10 different landing pages, nor are we going run any marketing campaigns. So, for us, the homepage is the landing page.

Before we get into the components of this page, we must talk about something called the Grunt Test which is again, developed by Donald Miller.

The Grunt Test states that the customer should be able to answer the questions in less than 5 seconds after looking at any piece of collateral.

1. *What do you offer?*
2. *How will it make my life better?*
3. *What do I need to do to buy it?*

The key point to note here is that your customer should know exactly what you're selling within 5 seconds of landing on the page. It should be absolutely unambiguous. Confusion will lead to the customer dropping out (lowering conversion rates). You have to be straight to the point. Avoid any sort of flair that would hamper understanding or readability. Your bet is that the customer will sign up based on your value hypothesis - not the flair of your writing or how beautiful your page is. This is an important distinction. I'm not saying that it's okay to have an ugly page, what I'm saying that you should have an effective page.

Another minor point that I think should be noted here is the usage of Fonts. Do not use fonts that are super hard to read. If the customer has to squint to get any sort of legibility, they won't even be able to understand what your value hypothesis is. Use fonts that are commonly used on the internet and which are rated high for legibility. Avoid any sort of cursive fonts (unless they are part of the logo).

The components of the landing page are -

1. *Header (and Sub-header)*
2. *Call to Action*
3. *Demo video*
4. *The Pitch*
5. *Feature list*
6. *Pricing*
7. *Reviews*
8. *Legal*

Of course, there are a lot of other items that you can add on top of this, but these are the core elements of the landing page. If you have written out the BrandScript for your product, filling out the content for the landing page is not going to be very hard. You might get stuck in figuring out how to layout the text on the page - usually, as a programmer, you have a UI Designer that does this job for you, but in our case, we don't have access to one.

What I have found works best is by going to Dribbble and just searching for layouts that look legible and clean. Once I find one that I really like, I take that as inspiration and build out something similar with my own twist (it's awesome if you credit the inspiration to the original designer somewhere in your website - doesn't have to be on the front page).

The Header: The header usually should not be something mechanical. It should inspire some deep emotion in your customer. Like the header of this book for example, "How to make a SaaS Product that actually makes money". It gives you some sort of emotional feedback and has a sense of future accomplishment. An aspiration. Hope. That's the kind of emotion your header should incite.

The Sub-header: The sub-header on the other hand, should be incredibly specific. It should tell the customer exactly what problem the product solves and how they can take advantage of it. Very concise and absolute.

Call to Action: The whole reason the landing page even exists is for the customer to click through the Call To Action. The Call To Action is usually a button that takes the user to the next stage of the process. This next step can be either downloading a free version of your app or creating an account on the website. Usually, I recommend placing the Call To Action button on a fixed header in the top left corner of the page but it depends.

The Demo Video: The Demo Video can be daunting to build because a lot of times we're conscious of recording our own voices. As I said, I stutter and so speaking into a microphone is like a life or death situation. The best way to get over this is by writing robust script. Your demo video should not be more than one and a half minutes. All you have to do is explain what the product does and show the product doing that. You'll accomplish this by recording your screen. If you want to take a look at a free video editing tool, iMovie works on Mac and you can get DaVinci Resolve for free to do some more advanced tasks.

The Pitch: The pitch is a one-line condensed version of the BrandScript.

Feature list: The feature list is a list of sequential value propositions that the customer can achieve with the product. As we discussed earlier, we're building a product that does one thing really really well—so it's okay if this list contains only one item. You can mention other auxiliary items like compatibility etc. to fill up the gaps. Don't overthink this. Do not mention your competitors if you

have them, how much better you are (the last thing you need is a checklist!). You want to distance yourself from the competition as much as possible.

Pricing: Plainly show your pricing. Use Dribbble as inspiration about different ways of displaying the costs. For metered rates, you can have a GUI that helps the user calculate potential spends based on usage. For fixed-rate subscriptions and one-time purchases, you can simply list the subscription costs. Again, don't overthink this. You are not expected to get everything right on the first go.

Reviews: Reviews help you build credibility and trust. Customers are compelled to try products that have a lot of social proof associated with them. Let's be honest here. You're not going to get 100s of reviews on the first day. It takes a very long time to accumulate that sort of social capital. The way you can get your first few reviews is by giving your product to people you shortlist from your target audience for free and ask them to write an unbiased review ('ask' is the operative word - do not coerce them). The review has to be unbiased. Don't do this for more than 3 reviews. If you have to keep doing this to get reviews, your product probably doesn't have the value you think it does. In most cases, you can just email an existing customer and politely ask them to leave a review. Customers are usually enthusiastic about reviewing a product—if they really like it.

Legal: The Legal section isn't really a section but a bunch of links to your Privacy Policy and Terms of Use. You don't really have to look into this at the start, but it's highly advisable to have a Privacy Policy and Terms of Use. There are a lot of free generators that will create these documents for you. The only thing I'd like to mention here is to make sure your product has an age limit of above 13 years old. You don't want kids using your product as you're not allowed to store information about children on your servers in a lot of countries. This has to be mentioned in your Privacy Policy and Terms of Use.

Where do we get our first slew of customers from?

I'm not a marketing expert but I'll tell you exactly what I did for Silence Trimmer. Technically, I didn't even set-up a landing page, I finished building the MVP and created a Gumroad listing with a flat pricing model. Then I bought the domain, silencetrimmer.com (which is still active), and created a listing on Product Hunt. You can still read the original pitch I made, it was not very polished, but it is what it is. The listing on Product Hunt was actually appreciated by a lot of people and got about 95 likes in 5 hours. It was featured for that day (no, it was not #1 or #2) on the homepage. That gave me a lot of confidence that this was something people wanted. It sold about 30 licenses in the first 2 months, not a massive amount but enough to convince me that this was a problem worth solving.

Now, there are also products that I built that did not get likes on Product Hunt. Does this mean that those products suck? Of course not. A very certain kind of person uses Product Hunt. Sometimes you're not going to find your target audience there, so you'll have to go where your Target Audience is. The best place to find these individuals are User Forums! A single successful post on the right user forum can give you a slew of consistent hits.

Now, to be good at this, you're going to have to let go of all your inhibitions. I know as programmers, we have a sense of elitism, where we think we're better than everyone else, especially people who do sales but sales are just as important as writing code. Without sales, writing code is pointless and without code, there is nothing to sell. It is a symbiotic relationship.

Apart from this, Twitter is also a great place for garnering awareness, but don't spam about your product all day long. Instagram, on the other hand, works only for certain kinds of products like Quick Photo and Quick Video Editors. I've never promoted a product on Instagram—so I can't really vouch for it. Use the GaryVee Model for Social Media—provide value without the expectation of immediate return. You will be rewarded for helping as many people as possible.

Email your (potential) customers

Email is not dead. It's scary how effective emails can be. You must get the email of every user before they use your product (even if it's a free download). If you don't, you're essentially blind. When you get a new email id, you should email them from your **personal email account** about how they like the product and that you're looking for feedback on what you can improve on. The keyword here is 'personal'. If you send them an email from sales@myproduct.com, there is no connection or humanity in that email. It's easier to ignore.

But an email from quinston@myproduct.com can create waves, and make the customer feel welcome and trusting. You have to do this to gather feedback. Customers are not going to email you first, you're going to have to go out of your way and reach out to them. Don't worry, you don't have to write every single email yourself. Personally, I use an automation tool called Zapier to send out these emails. Zapier lets me automate sending the email directly through my personal email - which means if the customer responds to it, the email comes directly to my Inbox. Obviously, this is a horrible strategy at scale, but when your product is at a nascent stage, you have to reach out to every customer.

You can automate sending emails at multiple points in time, send the first email 2 hours after sign-up, send the second email 2 days after sign-up, and another email, if the customer signed up for a free trial, and then cancelled. Keep testing the flow and keep making optimizations as you go along. Apart from this, newsletters are a must!

Distribution. Distribution. Distribution.

Before we get into this topic, I'd want you to develop a sense of appreciation about how important distribution is. Sales solve all problems. Sales and Product are two sides on the same coin—they have a symbiotic relationship. So, how do you go about growing a SaaS product that you just built (and sold a few copies of)?

Find someone that can help you

Yes, the best way to do this—is to find someone who works in that space. A marketer who understands the knowhow of how to grow a product. I believe that people should keep doing what they're good at. As programmers, we're so used to doing everything ourselves that we think this should also be done by us. NO! Find someone who can help you with distribution and cut a deal with them. Again, you're not superhuman. You can't do everything. This is more of a reminder to me, as it is for you.

The issue is, marketers are very picky about which products they take up—and if you start working with someone who is not passionate about the product, you might as well say good bye to the whole operation. On the other hand, you must work with someone who you think is generally a nice person, reasonable and responsible but also firm and ambitious. You should also have all of these qualities. Always think win-win. Always think about what is fair for you, and fair for them. Working with a partner on a product, is as good as getting married, you're both responsible for each other. Always remember that.

So, how do you convince someone to work with you?

1. *They should have an inherent belief in the product.*
2. *The traction so far (as little as it may be) should be an indicator that this product is worth pushing.*

These are two undeniable data-points that would convince not only your potential partner, but also yourself - that this product deserves to see the light of day. At the same time, continuous improvement should not be abandoned. You have not achieved product market-fit, you're not even close. The more customers that come in, the more of an opportunity you have to create a better fitting product.

If you ever get a bad feeling about working with the other person, don't go ahead with it. That's your gut telling you something is off. If you get a bad

vibe, call it off—unless there is mountainous evidence to suggest that you're wrong to feel that way.

Power of Word of Mouth

In the 'Further Reading' section, I've mentioned a book called 'Contagious: Why Things Catch On'. This book has convinced me (along with my own experience) that no 'type-of-marketing' is better than 'Word of Mouth' marketing. When someone you trust, tells you about a product—as opposed to an ad on Instagram—you develop significantly more trust in that product. When someone mentions your product on Twitter while telling their followers how awesome your product is... that's 'Word of Mouth' marketing. Think of all then instances in your own life when a loved one has told you about a product like Clothes, Groceries, Books, etc. and you actually went out and bought it. Happens more often than you think, doesn't it?

The book that I mentioned talks about a lot of frameworks on how to generate 'Word of Mouth' but in my experience, there are three factors that will help.

1. *Instant Aspirational/Emotional Relatablity.*
2. *Game-changing Product.*
3. *Good-will among customers.*

People will not share your product if it does not give them a sense of aspirational future accomplishment. There needs to be an emotional connection that the customers can latch on to simply based on what it claims to do. Don't confuse this with Virtue Signalling. Again, think about the title of this book, it rouses in you a sense of aspirational future accomplisment. That's why you wanted to read this book. The product has to be good—that's just an axiom. Also, the third point is really moot but no one will want to work with you if you suck at customer servicing. The whole reason you build a product is to help as many people as possible! I mean, if you fail at that, what's the point of anything?

The Affiliate Program

Whoever came up with Affiliate Marketing is a f*ing genius. Traditionally, you have to put in money up-front to run a marketing campaign and get customers. With Affiliate Marketing, you can reach out to content creators on the internet who have a target audience that coincides perfectly with the target audience for your product, and without offering any money up-front, give them a cut of all the sales that they generate promoting your product. I mean, that is genius. Yeah, sure, it might cost a little bit more over time—but if you don't have the budget to reach out to such a massive audience on your own, it's a no-brainer. Also, the audiences that watch these content creators are fiercely loyal, and they trust the creator. So, that particular promotion will have way more weight than any other ad that you run. Use it to your advantage.

The issue is, for the content creator to actually agree to such a deal, they have to like your product. Oh, yes. They are not going to push a product that they don't believe in, as an affiliate and if they do, there is something fishy going on.

What next?

You have a product that has continuous and predictable growth in revenue every month. You're making some money. Throughout all of the previous stages of development, you were just a writer of code who built a product and got distribution set up. But once the product starts reaching monthly revenue enough to rival your main income source, things start getting a little serious. At this point, you have to make a decision—either you continue working your main job and keep the side hustle as it is or you jump-ship and make your side-hustle your main hustle. This can include anything from registering an actual company, bootstrapping it, or raising VC-funds to scale further.

The factors that it depends on are.

1. *Does your product have a market big enough to have a continuous steady stream of customers (and excite VCs)?*
2. *Do you really want to risk it all by taking the plunge (or do you even have a choice)?*

The VC Game

Now, all this while, I've written about how you should ignore your competitors and focus on your customers but once your product starts maturing, competitors will start popping out of the woodwork—trying to claw at your customers. Their products will be very similar (if not exact replicas) of what you're building. In Blitzscaling, Reid Hoffman gives an example of how Airbnb had to raise tonnes of cash because of a competitor that built an exact replica of what they built—and this turned out to be a threat to their whole operation. Unfortunately, this happens more often than not.

What I'm trying to say is, if your product has a large enough market that it makes sense for big companies and VCs to pour money into, you don't have a lot of time before a competitor pulls up. VCs will either invest in your product or your competitor's product. VCs usually stay away from niche products as the return on invested capital might not be as lucrative. For example, investors might not put money into a Unity Asset that you created, but they might fund a TikTok Clone. If you build a product that has a large potential market, you're at risk if you don't move fast.

This basically means that if you don't keep growing fast, the revenue you're making at the moment might vanish. It's a hard pill to swallow. All that hard work, sleepless nights and frustration that you dealt with to reach this point, and then you read that it can all be taken away in one fell swoop. I'm not gonna paint a rosy picture but it's genuinely hard to create a sustainable, revenue-generating profitable product and the battle isn't won when you start getting consistent revenue, it's only getting started!

Just sell it off

The other play that you can make is just selling the whole thing off to a potential buyer—and get a nice payday. Companies would rather buy the product off of you than invest time and resources into building it themselves. The reason they do this is that it alleviates a lot of risk for them. Your product is already in the market and it has steady customers that use it. It's much easier for them to buy the infrastructure from you, and try to scale it up. The acquisition is actually cheaper for companies than you might imagine. A lot of costs for building products in larger companies are hidden away.

The Transition from Engineer to Entrepreneur

Once you reach a point where you transition from an Engineer to an Entrepreneur, at least in my case, it's very very hard to go back. It is truly a shift in lifestyle and mindset. Both of these things change simultaneously. Owning your product has a very different connotation to just working on one. When you own a product, it becomes part of you. You obsess over it. It's your gift to the world. You start to feel like it's your moral responsibility to create epic products to help as many people as possible, and this in turn becomes a vicious positive cycle.

I've had employees and I've not had employees, but I never stopped writing code. There is just something about writing a piece of text and then running it in a console that gets me every-time. The beauty and simplicity of telling the computer what to do, and witnessing the computer do it in front of you is mesmerising.

Final Thoughts

That's actually all that I wanted to write about. I wanted this piece of content to be incredibly dense and anti-verbose while at the same time acting as a platform from where you, as a programmer, can jump off. Most of what I've

written about is my own experience of going from a broke college kid to being able to support my family with SaaS product revenue. I know how crazy that sounds.

I'm not going to pretend I haven't done a bunch of services to support myself along the way. The first company we started is a Service Company where we eventually built a product that helped us charge more for services. I'm sure you could find my UpWork, Freelancer, and Truelancer accounts without even trying. Products will give scale that services can never unless they have low-replication costs and high margins. That does not mean doing services is a bad thing. Sometimes a secure and safe service company can do wonders for your mental health and sometimes it's just a painful exercise in perpetuity.

All in all, thank you for reading all the way until here. I hope reading this book gave you an insight into the risks, rewards, and steps into building a SaaS product. Follow me on the social links below to let me know you appreciated the book!

If you enjoyed reading this book, you can show your appreciation by purchasing it on Gumroad.

Thank You!

How to get in touch with me?

- Find me on Twitter @graphoarty (https://twitter.com/graphoarty).
- Subscribe to my YouTube Channel (https://youtube.com/quinstonpimenta).
- Send me an email on *mail@quinston.com*.

Have a good one!

Further Reading

These are the top 10 books I highly recommend.

1. Zero To One
2. The Lean Start Up
3. Blitzscaling
4. The Innovators Dilemma
5. Contagious: Why Things Catch On
6. HBR's 10 Must Reads on Strategy
7. The 80/20 Principle: The Secret to Achieving More with Less
8. Flow: The Psychology of Optimal Experience
9. Atomic Habits
10. Crushing It!