From Idea to Product: The Application Development Process

Introduction

You should read this if you:

1. Are building a web and/or mobile based product
2. Have money (or have access to it)
3. Need a plan to ramp up development in order to churn out features

In this book you will learn:

1. Best practices for hiring individual developers and freelancers
2. Best practices for hiring development firms
3. Efficient methods for maximizing the long-term throughput of your development team
4. Risks you will encounter as you build your product

Context | Philosophies and Values

Everything in this book stems from three of my personal core values. These values drive everything we do at SmartLogic.

Stay Simple

Always select the simpler of two approaches. You can easily increase a system's complexity; however, the inverse is not true.

Always start as simple as possible. The benefits of simplicity cannot be overstated.

Be Efficient

Institute the minimal amount of process by which your operations can run smoothly and with minimal surprises.

Prefer deterministic, predictable and repeatable processes over flying by the seat of your pants.

Eliminate mundane tasks. Identify and employ techniques for maximizing the efficiency and throughput of your team.

State your Assumptions

When you assume, you make an ASS out of U and ME.

You may think your idea is genius, but what do actual customers think, and what do the data and numbers show? There’s no better way to find out than to ask and test.

Further, you may implicitly be assuming your coworkers understand things that you take for granted when they actually don’t.

Context | About the Author

I’m the President of SmartLogic, a consulting company that builds web and mobile-based products. Throughout the years we’ve learned that a simple, agile process is the best way to help startups and enterprises alike build web/mobile-based products that quickly adapt to the unpredictable needs of their clients and users.

I’m a lifelong engineer with BS/MSE degrees in Computer Science from Johns Hopkins University. I take an analytical approach to my work. I’m also very pragmatic.

Table of Contents

Let’s Talk About Your Product

Marketing 101: Just Build an Awesome Product

This is not a book about marketing but it is important to set the stage. Recall the four P’s:

1. **Product:** a tangible good or intangible service that satisfies a need for a given market.
2. If you are reading this I assume you are building a web and/or mobile based product, or a physical product with a substantial web/mobile based component to it.
3. **Placement:** the method by which the product is distributed to customers. In our world this is typically via the web (i.e. your URL), the App Store and Google Play.
4. **Promotion:** the methods by which potential customers find out about your product. E.g. Google Adwords, Facebook advertising, trade shows.
5. **Price:** the cost of your product. Common models include monthly subscriptions and free/ad-supported (e.g. Google Adsense).

My theory is as follows: build an awesome product and the rest will follow.

That is, if you build a product that satisfies a need in the market, distribute it in such a manner that it is easy to purchase, promote it in such a way that potential customers can find out about it, and price it in an affordable and reasonable fashion, then your product will sell like hotcakes and everything else will follow.

Application Development vis-à-vis the Scientific Method

Starting a web/mobile business is like a big science experiment.

First you **start with several facts** that you will leverage in order to bring value to one or more target markets. Example facts: people like to save money, there is an obesity epidemic in the USA, students graduate college with more debt than ever, baby boomers are retiring soon.

Then **generate a problem statement:** a statement of how your target market(s) is/are underserved by existing products and service offerings (if any). For example, cheap food options are unhealthy and there are no diet solutions that also focus on sticking to a tight budget.

Given a set of facts and a problem statement you can **form one or more theories** that will directly address your problem statement. Given your theories you will build a product and company to serve one or more target markets.

You then **formulate a number of hypotheses** in order to confirm or refute your theories. You test your hypotheses (i.e. run experiments) by shipping features to users. If the features are used by your customers then you build support for the respective hypothesis. If the features are not used then this disproves your hypothesis.

If your experiments are successful and support your hypotheses—in turn building support for your theories—then you are likely to achieve success, whatever that means to you:

1. Generating revenue; and/or
2. Making a marked improvement in the life of your end-users; and/or
3. Otherwise achieving a stated objective.

Example 1: Groupon / LivingSocial

Let's consider a well-known example—the generic daily deals site (Groupon, LivingSocial, etc.).

Facts (Assumptions / Postulates / Claims)

1. People like deals, and more generally, saving money.
2. Snail mail is a costly and dying medium.
3. Further, email is a much more effective and scalable medium by which to distribute deals.
4. Merchants offer deals to consumers to attract new (and re-engage existing) customers.

Problem Statement

Consumers and merchants alike are being underserved by the underwhelming service offerings of the existing coupons industry; the industry has not been brought into modern times. Existing coupon offerings suffer a branding issue because they're seen as uncool and inefficient (not worth the time). There exists significant opportunity in modernizing the industry.

Theories

Consumers will pay for deals that are distributed via modern mechanisms (email, apps, etc.).

Merchants will distribute deeply discounted deals via a third party company that is well equipped for distribution in the modern age.

Hypotheses (Core Features)

1. Users will sign up for email newsletters offering daily deal specials.

2. Users will download an iPhone app that will send them push notifications about daily deal specials.
3. Users will purchase a special and then spam their friends if offered the ability to receive the special for free if 3 friends purchase said special.
4. Merchants will offer 50% off deals to consumers and give a percentage of the deal revenue to Groupon/LivingSocial.
5. Merchants will use tools provided by Groupon/LivingSocial to track when a coupon has been redeemed.

Each of these hypotheses, or core features, can be broken down into a group of user stories that developers will build. We'll talk about user stories later.

Core Hypotheses Worksheet

This worksheet helps you formulate your core hypotheses, which in turn describe the first set of features your development team will build.

Product Name: *Working titles are fine*	______________________
Facts:	1. ______________________ ______________________ 2. ______________________ ______________________ 3. ______________________ ______________________
Problem Statement: *keep it short*	______________________ ______________________
Theories: *keep 'em short*	1. ______________________ ______________________ 2. ______________________ ______________________
Hypotheses: *keep 'em short*	1. ______________________ ______________________ 2. ______________________ ______________________ 3. ______________________ ______________________
Target Markets:	1. ______________________ ______________________ 2. ______________________

	____________________

If you’re able to complete this worksheet then you’re ahead of most “startups” out there. Of course, it’s not a substitute for a full-fledged business plan, but this is enough direction to get an experienced development team building your product. Keep in mind that hypotheses evolve and change as new information comes to light—don’t be afraid to go back and edit this form as the fundamentals of your business change. Follow through the rest of the sheet with any changes this causes.

What’s the Point?

Now you know what you want to build. Let’s go.

Building a Development Team

What Can be Outsourced

Before we dive into this discussion: no matter how you choose to develop your product, you won't succeed unless you have strong leadership. When I say "strong leadership," I mean entrepreneurial types who get stuff done. An outsourced development team can be a big piece of this puzzle; however, you can't outsource everything.

The best companies we work with have strong leadership in-house. If your team's leaders don't set a strong direction for the product and can't quickly make decisions then the technical team responsible for building your product will struggle, whether they're in-house or outsourced. It's hard to meet expectations built out of sand.

It's critical to have leadership in the following roles in-house:

1. Marketing
2. Sales
3. Product management

Now, that's not to say you need different people for each role. Indeed, we've had clients where one person was responsible for all of the roles above.

The Three Models

For software development, we generally see organizations work with three models:

Outsourcing All Development	An internal product manager works with an outsourced development firm, like SmartLogic, to build the product. No matter the product manager's title, his or her job is to set the direction for the outsourced development firm's efforts.
Insourcing All Development	All application development is performed in-house. Companies following this model hire and retain on staff their entire technical team, from a CTO down to entry level developers, and continue to pay their salaries on a roughly indefinite basis.
Using a Hybrid Model	You've got a technical project lead in-house and possibly one or more developers, but you pull in an outsourced firm when you need to increase how quickly the team is able to build the product.

Outsource vs. Insource vs. Hybrid Comparison

The table below illustrates high-level use cases for each model. After the table, we'll go into detail on the relative pros and cons for each of the three models.

DISCLAIMER: there is no one-size-fits-all approach. What's in the table below is written from my experience. Have I seen deviations? Absolutely. Do what works for you.

Constraint	**Insource**	**Outsource**	**Hybrid**
Time	Longest ramp-up period	Shortest ramp-up period	Short ramp-up, long-term resources in-house
Budget	Less expensive, fixed long-term costs	More expensive up front, more flexible long term	In the middle
Long-term plans and predictability of workload	Must have a consistent flow of work	Good for both development spikes AND/OR a consistent flow of work	Fixed amount of required work with the ability to increase or decrease
Management's experience hiring and managing technical people	Expert	Novice to expert	Intermediate to expert
Management's experience developing software products	Expert	Novice to expert	Intermediate to expert

You Should Outsource All Development If:

- **Your team has limited experience managing the software development process and hiring technical people.**
    - It's much easier to hire one software development firm rather than hire ten software developers.
    - Working with a trusted outside firm can simplify the process and save you time, money, and stress.
    - It's difficult to hire individual developers nowadays. On the other hand, there are many software development firms to be hired.
    - Software development—like any technical or specialized profession—is difficult

and requires a specific skillset. A professional team will know what it's doing.

- **You need to build your minimum viable product or deploy a large number of features relatively quickly.**
  - It can take months just to find and hire the right technical talent, and that's not even including how long it will take for them to build the product. An outsourced firm will already have a team of talented developers ready to go.
  - Further, outsourced teams have worked together before and (should) have a consistent process from project to project. Because of this, they can focus on what they're supposed to: getting features out the door.
- **You have a lot of features that you need to build right now, but you're uncertain what the future will hold.**
  - Outsourced firms expect projects to come and go. As such, they are equipped to handle the common scenario where a client needs to ramp up or down.

You Should Insource All Development If:

- **Your team has extensive experience managing the software development process and hiring technical people.**
  - Having a team member experienced in the development process and technical process will help ensure things go off without a hitch.
- **You have enough time to hire and ramp up developers.**
  - It takes time to find the right talent. If you have time to dedicate to recruiting and onboarding then insourcing makes sense.
- **You are able to accurately predict your development needs**
  - When you have developers on the payroll you want them working to deliver features; you don't want developers "sitting on the bench" with nothing to do.
  - Similarly, a certain number of developers can only work a finite number of hours per week, which may translate into only a certain features delivered on a weekly basis. If you are in control of how many features you need delivered in a given week, then you can plan ahead for how many developers you will need.

You Should Use a Hybrid Model If:

- **Your team has moderate to extensive experience managing the software development process**.
  - Working with an outside team gives you access to people who have been doing software development for a while. The team should have sound processes, and can provide guidance and expertise. Their distance allows them to provide objective feedback on the way you do things.
  - Your internal people can continue to grow and learn, becoming even more valuable.
- **Your internal team of developers isn't large enough to build your product or deploy your features as quickly as you'd like.**
  - Rather than scrambling to hire more developers, you can easily and quickly ramp

up your development capacity with an outside firm. Then take your time to find the right developers in-house and scale back with the outside firm, or scale back development as a whole by using less time from the outside firm instead of firing people.
    - You might also want to use an outsourced firm to supplement your internal team if your product involves skills your people lack, such as developing APIs or mobile applications. Your people can learn these skills for the future while working with the right outsourced firm.
- **You have a consistent need for software development for the next few years, but every now and then, your need will spike.**
    - Bring enough developers in-house to handle your consistent demand. Then manage the spikes by working with an outsourced firm.
    - This way, you won't need to hire and fire developers on a regular basis. Your developers will also do better work if they know their jobs are secure.

Make the Right Decision for Your Situation

Take my advice here as a guideline. You might not fit into any one category, and when it comes down to your decision, you have to do what works best for your specific, unique situation.

Hiring an External Development Firm

Let's assume you've decided you're going to hire an outsourced development team. It's critical to get this decision right because you will be spending a significant amount of time and money with said team. Much like marriage, you should court a few vendors before committing to a long-term relationship with any one vendor. Also much like marriage, divorcing a vendor is costly, timely, and emotionally taxing.

The process I recommend is as follows:

1. **Get organized:** prepare a document you can send to each vendor you'll interview. We'll provide more details on this below.
2. **Find potential suitors:** you'll want to talk to 3 or more prospective development partners.
3. **Interview them:** have one or more conversations with each company. Let your gut help inform your decisions. Watch out for the red flags mentioned later in this section.
4. **Get proposals:** proposals should at a minimum include
   a. an estimated cost for a specified amount of work;
   b. an explanation of the firm's process; and
   c. other pertinent information associated with what you can expect when working with the development firm.
5. **Sign a deal:** pick a winner and get moving; time is of the essence!

Get Organized with a Development Prospectus

To save yourself time—and help organize your own thoughts—consider putting together a document to share with the vendors you will interview. Note: this should NOT be an old-school voluminous RFP or RFQ; those are outdated modes of communicating what you're trying to build. Prepare a short document that at a minimum covers the following:

1. Your core hypotheses worksheet, or at the very minimum, a bulleted list of features you want to build (this does not need to be an exhaustive list, an overview is perfectly fine)
2. Key team members and contractors along with brief bios and respective areas of responsibility
3. What you've accomplished to-date (wireframes? prototype? secured financing?)
4. Timeline and important dates to consider

If you don't have all of this documented, expect any competent vendor to extract this information from you over the course of your initial meetings.

Key Roles and Responsibilities When Outsourcing Development

When you're discussing key team members it's helpful for the vendor to have insight into the people in the following roles:

Role	Primary Responsibility	Client or Dev Team?
Product Owner (PO)	Authorizes and accepts (or rejects) user stories added to the development queue	Client
Finance Person	Authorizes and executes Statements of Work (SOW's); cuts checks	Client
Technical Lead	Manages the external development team's efforts and manages all day-to-day interaction with client stakeholders	Dev team
Developers	Write code and ship features	Both or only the dev team

Caution Against Prepping too Much Before Development

While you clearly need to think through your product and have a good idea of what you're trying to build it's often counterproductive to define features that will be built months down the line. The only thing that is constant is change; the features you want built into your product will change.

Good things to prepare:	**You may be going overboard when you have:**
1. A development prospectus. 2. A small prototype. Just keep in mind that when Boeing builds a 747 prototype the prototype doesn't make its way to Southwest's fleet. 3. A product roadmap, which will inform features to be built down the road. Keep in mind the roadmap may change; be flexible.	1. Fully fleshed out wireframes and corresponding designs. 2. Complicated PSDs mapping out most functionality and screens in the application. 3. Lengthy requirements documents, including functional specifications and other similarly detailed documents.

A Caution Against Wanting to Start Yesterday

Not to say that time isn't of the essence, but if you don't take the time to set the stage for a healthy partnership, you will lose a lot of time down the road—and may even need to go through the messy process of starting again with a new vendor.

Finding Prospective Vendors to Interview

Play the field before engaging a vendor.

Seek out community leaders in town and ask them to send you a few suggestions for companies to hire. This way you'll know the firms are reputable and do the kind of work you need. Ultimately the decision of who to hire is yours, so you may as well ask for a few suggestions as opposed to just one. Or, of course, you can search online and find the firm with the best eBook.

You can find community leaders by asking a local incubator, asking the organizers and sponsors of local tech groups on meetup.com, and/or asking investors.

Once you've got a couple leads on development firms to interview, see if you can narrow that down to several prospective companies with which you'd consider working. Visit their websites to find out more information about the work performed by the companies. Stalk the principals on LinkedIn and see if you have any mutual contacts who could offer blunt feedback.

Interview the Vendors

After introductory formalities (discussing your product, team, etc.) use the guide below to help shape your discussions with vendors. Of course, you'll have different follow up questions with each vendor, but here are questions I'd suggest asking every vendor with whom you speak.

A note on using point systems and other quantitative evaluation methods: unless you're the government and you *need* to measure contenders, your gut is often a better indicator of which firm to choose when compared with seemingly arbitrary scales.

Question	**What to look for**
What type of projects do you work on?	Look for vendors who have experience building unique products, nc building simple websites or configuring Content Management Systems (CMS's). There's different expertise and project management for CMS vs. custom app development. If the vendor doesn't have a team of people very experienced in custom app development, be wary. ***Red Flag: A company that only builds CMS's.***
Have you created libraries to streamline software development?	Experts don't reinvent the wheel. They stand on the shoulders of giants and leverage existing frameworks for common tasks such as authentication, testing, server deployment and the like. However, this is sort of a trick question: small tools, scripts, and code libraries that augment the process can be very helpful. Experts will have created small tools that help with very specific and isolatec tasks. For example, our team created the rspec_api_documentation tool to help document APIs. ***Red Flag: A completely custom approach to every problem, and on the opposite end of the spectrum: any "one size fits***

	all" solutions.
What tools do you use to streamline development?	Feel comfortable with firms that use widely respected tools, and bonus points for tools that are open source. Be extremely wary of building your product off of proprietary tools (even if they're "open source") that are not widely supported. This could leave you in the lurch if you need to stop working with the current firm. For firms using Ruby on Rails, ruby-toolbox.com is a great resource for researching tools. ***Red Flag: Something proprietary that nobody else has ever heard of.***
What framework will you use to build our application?	Look for vendors that use web application frameworks such as Ruby on Rails, Django or CakePHP. These are frameworks for creating custom applications, like your product. Stay away from strangers with candy and vendors who say they'll use WordPress, Joomla, Drupal, Magento, [insert name of CMS or eCommerce framework here] as the backbone of your product. CMS's should only be used for content sites, and eCommerce frameworks should only be used for eCommerce sites. A Cessna with a single propeller is great for traveling a couple hundred miles at speeds under 150 mph. But I wouldn't retrofit it with jet engines and then try to fly across the Pacific at 600mph. ***Red Flag: DotNetNuke, Joomla, Drupal, ExpressionEngine, WordPress, and (let's be honest) .NET.***

How will I know where we stand on scope and budget?	With the right project management system, you should be able to go to a website, such as Pivotal Tracker, and see the status of all features. This should be SUPER easy for you to do. For knowing where you stand on budget, consider how often the firm invoices and how budget is reported back to you. SmartLogic, for example, invoices weekly. Invoicing weekly protects you from getting shocked at the end of the month, or even worse, the end of phase in the project. Whether your firm uses this method or simply tracks hours in a spreadsheet, they should have a clear answer to this question. ***Red Flag: Any budget or scope updates that feel infrequent enough that the information would be "too late."***
How does the estimation process work?	Estimating a budget after just a few sales discussions can be hard to do, especially with products for startups who may need to chang their strategy given customer feedback. That's why communication is so important. The whole point of agile development is that the product will change. Expect a ballpark and as your product changes, make sure the firm can tell you how changes in scope will affect cost. ***Red Flag: We can give you a fixed-price bid.***
Are your project managers technical?	Can it work for a non-tech person to manage a technical project? Probably, but it's a risk. Some people might say developers should be focused on coding and not managing client relationships and questions. But separatin developers from the client means you've got more chances for miscommunication and mistakes. The ideal project manager is a developer with project management experience. The PM should be supported by a simple process. *Everybody* at SmartLogic writes code. ***Red Flag: The answer "No. Besides, developers don't know how to talk."***
How often can I access the latest version of the app?	A development shop focused on developing products will have a continuous integration (CI) system in place. CI is an automated process by which changes to code are deployed to various servers and tested against an automated test suite. Features and code should be deployed continuously—and automatically—so that all stakeholders can see the most up-to-date code in action.

	Be wary of having to ask to see progress or features in development. The last thing you want is to pay for development for long periods of time without being able to see and review features; this increases the chances of being unpleasantly surprised. ***Red Flag: Anything less frequent than several times a week, and if it seems like they don't actively want you see and approve work frequently.***
Describe your QA process.	You want a company that has an automated QA process and that practices Test-Driven Development (TDD). You should feel confident that there won't be regressions (new features can't break old features.) Be wary of companies that have manual testing processes; they don't scale. ***Red Flag: Anything with a manual component*** ***Atomic Red Flag: the answer "the client is responsible for testing code changes."***
Describe a situation where a project did not go well. What happened and what did you change?	If a vendor tries to tell you that they've both (1) never faced any issues on a project; and (2) bore no responsibility for said issues then they are lying or you are their first customer. Nobody is perfect and no project goes off without a hitch. You want a vendor that can act as a trusted partner and that can be honest with you. You also want a vendor that is constantly improvin its process. At SmartLogic we are extremely quick to make changes. Generally those changes result in improvements to our process and client relationships. When that's not the case we're just as quick to change and try something new. What you don't want is a vendor that repeats the same mistakes and is unable to learn from said mistakes. ***Red Flag: (1) We've never really had a client dissatisfied with our work; or (2) it was all the client's fault; or (3) no clear explanation on steps taken to mitigate risk of the situation happening in the future.***

What to Expect in a Vendor's Proposal

First things first: before you even receive a proposal you should have a good idea of the vendor's process and what it would mean to work together.

The proposal should very clearly describe how you will engage the vendor and how work will proceed. There should be no surprises in the proposal because you typically will have had several discussions with the vendor before receiving a proposal. What to expect:

1. **Financials:** Both an estimated cost for a specified amount of work along with important billing information (invoicing frequency, payment terms, etc.).
2. **Timeline:** Both start date and projected completion date (if applicable).
3. **Work plan and development process:** What's the next step? Once engaged and the development team is ramped up, the proposal should be clear of the next steps.
4. **Project risks:** Each project will face its own unique risks. You want a vendor that is pragmatic and cognisant of the risks your specific project will face. Because the vendor is an expert in application development the firm should have foresight into risks your project may encounter along with tactics for mitigating said risks.

Notes on Budget and Timeline

First: **If it sounds too good to be true then it probably is.**

Angelina Jolie does not want to date you.

Secondly: rarely, if ever, can you judge two development firms based solely off of budget and timeline. No two development shops are born equal, a lower budget or faster timeframe often indicates lower quality as well. Further, in general be wary of firms that send lowball offers. Nobody wins in that scenario.

Finally: go with your gut. Realize that as your product changes so too will your budget and timeline.

Pick a Vendor and Sign a Deal

Once you've selected the firm that will help bring your product to reality you'll need to deal with various administrative things—contracts, picking dates for meetings, etc. Most of this is straightforward and simply takes time.

It is standard operating procedure to use the vendor's Master Services Agreement and corresponding Statements of Work. The MSA and SOW's should reflect how the vendor works. If you—as a client—insist on coming with your own legal documents then you are implicitly working to manage the process, which may have negative consequences down the line.

Don't be a Micromanager

You're paying your vendor the big bucks because they know what they're doing. Have faith in your decision by trusting them to do the best job for your company.

Thoughts on Hiring Development Firms for Equity

Don’t do it.

Empirically I can say that it’s only something less mature development firms do. It’s a very complicated type of deal to work out given that development on custom products never really ends. Unless the firm is paying their employees with your equity, there will always be an imperative for them to work on billable client work, putting your company at a disadvantage.

How the Sausage Gets Made

Now that we've discussed how to think about what should go into your product and how to hire a team to build that product, let's talk about how the product actually gets made.

Development 101: Maximize Long-Term Throughput

The objective of your development team should be to maximize its long-term throughput, which I define as the rate of features being delivered per unit of developer time. We ensure consistently high throughput at SmartLogic by structuring efficient processes around the following key **tenets** of the development process:

1. **Meetings & Communication:** Meetings and communication serve two vital purposes: (a) enable developers to know what to build; and (b) make business folks aware of immediate or potential bottlenecks.
2. **Development Backlog Management:** The backlog is an ordered list of all features on the development schedule; a comprehensive list of what the developers will work on in the near-term.
3. **Risk Management:** There are sundry risks that can and will adversely impact throughput. Maximizing long-term throughput requires being cognizant of risks under the team's control and taking measures to mitigate their impacts. It's also helpful to think of risks that are not under the team's control.
4. **The Process of Writing Code:** last and CERTAINLY not least is the process by which developers write code. There are various best practices for writing code that we cover below.

Continue reading to learn about tactics you can take to ensure maximum success for each tenet described above, and furthermore specific tools you can use as you work to execute the tactic.

Why Long-Term Throughput?

We optimize our development for long-term throughput because unlike with short-term projects with a defined lifetime or scope, product development isn't one-and-done. If your product is successful, you'll be releasing features ad infinitum. Quality, sustainability, and consistency for your time and budget are more important than rushing along towards features or cutting budgets whenever possible. With a steady, consistently high long-term throughput you'll be able to predict where your budget will take you in terms of your product.

Tactics vs. Tools

We outline the tactics we use along with some possible tools. **Tactics** are the pillars of the process. Tactics change on an infrequent basis (e.g. every several years)**.**

The **tools,** on the other hand, are mutable. They can and should change as better or more convenient tools become available.

Meetings & Communication

Meetings and communication ensure consistently high throughput by:

1. Eliminating bottlenecks
2. Filling the development backlog
3. Aligning expectations

Tactics	Tools
Hold **Epic Meetings** when you need to fill up the development backlog and discuss the next several iterations' worth of work.	In-person meetings or virtual meetings via Skype/Google Hangout
On a regular and more micro basis, e.g. weekly, hold **Iteration Meetings** to ensure that priorities are aligned.	In-person meetings or virtual meetings via Skype/Google Hangout
Standup Meetings are the most micro type of meeting you can have. We hold them every morning.	In-person meetings or virtual meetings via Skype/Google Hangout
Constant **Team Collaboration and Chat** allow questions to be answered as quickly as feasible.	Sitting in the same physical room, Campfire, Hipchat, Jabber
Tell us how you really feel; **communicate effectively**.	Ask all stakeholders for their Project Temperature (see below)

Project Temperature

At the end of each iteration meeting we ask each person at the meeting to assign a *project temperature* to the project and explain their rationale. The project temperature is simply a rating of 1-10 of how well the individual thinks the project is going. 10 means everything in the world is just peachy whereas 1 means somebody should be fired immediately.

We do this because oftentimes you can go through an entire meeting without actually sharing your concerns or praises about a project. It's important to know why someone is happy or unhappy. You'd be amazed at what can come to light when people need to assign a simple numeric value to their feelings on a project. Catching concerns early keeps the project from going in the wrong direction for too long. On the other hand, it's important to know what the team is doing right.

Development Backlog Management

It's critical to keep a close eye on the backlog because the backlog very clearly defines what work will be performed by the design and development team. Of course, if you're paying for their time, you want your employees/development firm to only work on high-priority items. You need to prioritize your developers' work to align with your customer/stakeholder needs and your overall business agenda.

Here's how we manage our development backlog:

Tactics	**Tools**
Write **user stories** (more on this below) in a commonly understood language and in a consistent format.	We use the **Gherkin** format to write user stories.
Keep track of the list of **user stories** and their relative priorities.	Though they may start on a whiteboard or index cards, we eventually document all **user stories** and their relative priorities in **Pivotal Tracker.**
Go through the **life cycle of a user story** from conception (writing it with the team) to full-grown independent adulthood (deployed to production).	Pivotal Tracker does a great job tracking the **life cycles** of user stories.
Have the product owner **accept or reject all delivered stories.**	Automated **email notifications** are sent to the product owner as stories are delivered.

What's Pivotal Tracker

Pivotal Tracker documents all features and bugs and is a running portrait of what is in the project. Your live product is the summation of everything in Tracker.

User Stories

Essential for a smooth development process, user stories are the most atomic units of describable work that developers and "business" people alike can understand. User stories are the common language understood by both developers and non-developers. Developers ultimately write code to implement user stories.

We use the *Gherkin* format for writing user stories:

In order to <do something of value>
As a <user type>

I want to <perform some function>

User stories are added to the **development backlog**, which is a list, ordered by relative priority, of user stories. If a user story is on the backlog, it is on the development schedule. If a user story isn't on the backlog then the developers will not write code implementing the user story.

Developers should write user stories in a consistent manner and so that all technical complexities are considered. Being thorough allows developers to assign accurate estimates to the user stories.

Estimating User Stories

Part of writing user stories is estimating the complexity of writing code to implement the story. This helps you plan development and design time, as well as budget costs for a given set of work.

User stories are assigned "point" estimates. Points describe the relative complexity of a story. For example: a 4-point story is probably twice as hard to implement as a 2-point story.

There are several point scales you can use to estimate the work around a user story:

- Linear (0, 1, 2, 3)
- **Exponential (0, 1, 2, 4, 8)**
- Fibonacci (0, 1, 2, 3, 5, 8)

We use the exponential scale, but it's up to your team to find out what works best for you.

The Lifecycle of a User Story

State	Description
Not Started	Developers haven't started work (coding, thinking, whatever) on the story. It's free for any developer to take.
Started	A developer takes ownership of the story and has started architecting, writing tests, and generating code to implement the story.
Finished	Code is written and checked into the repository for review by the technical lead and ultimately deployed to the staging server for review by product owner.
Delivered	The technical lead has approved the story and deployed it to the staging server for review by the product owner.
Accepted	The product owner has reviewed the story and declared that the story was built to his/her expectation.

Rejected	The story is not to the product owner's expectation and is sent back to the developers. This is not a big deal; rejections happen all the time. The important thing is that the product owner approves all work before it gets deployed to production.

Accepted stories can then be scheduled to go to production. Developers own the Not Started to Delivered phases whereas the product owner owns the Accept/Reject phase.

Risk Management

Throughout the course of writing code to develop your product you will encounter several risks that will impede progress and therefore result in lower short-term throughput.

In this section we focus on risks that affect the development team (e.g. using a new API) as opposed to risks that more generally affect your company (e.g. Google entering your market and competing with your product).

Because history repeats itself we know many of the risks your project will face. Of course, there are unpredictable risks, but below, I'll cover the common ones we've encountered.

A Note on Off-Shore "Resources" (Read: Developers)

I've been asked this question several times in my professional career: "Can we hire these Indian/Ukrainian guys to develop some components of the software for cheaper than your rate and have you manage them?"

My answer? No. Would you ask your surgeon if you could bring in an outside team of unproven nurses so you could have a cheaper operation? It could work, but it's a big risk.

Most software development firms' developers are trained in a given process and work efficiently as a team. Your firm won't have the same level of control over the process of an outside team beyond telling them which features to develop that week.

Potential Risks of Any Development Process

Note: When we say “third-party,” we mean adding another group or individual to the equation, whether they’re outsourced or in-house, which can slow down your outsourced firm (as we’ll explain in detail below).

Risk	Factors that increase risk	How to mitigate the risk / what to expect
Working with Third-Party Designer(s)	**The designer is moonlighting.** They won’t have enough time to commit to your project and will probably flake on you. **The designer has limited HTML/CSS/production experience.** They’re designing in a vacuum. Don’t hire MC Escher to design your house. While the designs may be aesthetically pleasing they may not be practical or easily implementable.	Have a plan B. Anticipate that you will get to a point where you outgrow the designer’s capabilities or availability. This will slow down your project and cost you more. We often see clients paying the third-party for unusable work and then paying us to redo it.
Working with Third-Party Developer(s)	**The developer is moonlighting.** See above. **The developer has limited full-stack development experience.** Time we spend training other devs is time we bill; training slows us down and adversely impacts our throughput in the short-term. **The developer is not well-versed in best practices.** When everyone understands and follows the same set of best practices you can move extremely quickly and be confident that you are getting extremely high quality code. Unless the external developers’ quality of work is unquestionably on-par with or better than the firm you’ve hired, they’ll adversely impact throughput.	See above. Furthermore, code is more expensive than design and more integral to your project. So the risk and the bad effect on your product and budget is magnified.
An Existing Design,	**There are unrealistic aspects to the design.** The design may reflect bells,	Don’t spend too much time on wireframing and design

Wireframe, and/or Mockup	whistles and "nice to haves" but in all likelihood will need to change to reflect reality: budget, time and what the end-users really want. **The design will change.** As the application is built to whatever design is provided, there is 100% likelihood that everyone on the team will want to change various aspects of the design. This is a good thing; however, there is a cost associated with it.	before development. If you do spend a lot of time on this phase, don't get attached, and don't expect it to save you time down the road.
Using a Backend API as a Datastore	**The API is slow.** As the frontend application is built it may become evident that optimizations need to be made to the backend API in order to provide a better experience to the end user. Performance improvements will need to be made to both the backend API and the frontend application. Your team will also need to collaborate with the backend API developers in order to design and develop the most cost-effective optimizations. **The API will need to be modified.** It's almost always the case that the API won't have all of the fields and methods necessary to support 100% of the desired functionality in the frontend application. Frontend developers will need to spend more time interfacing with the backend API developers in order to identify all required functionality.	Expect that the developers will need to spend time focusing on (1) improving the performance of the API; and (2) modifying the API to reflect the needs of the frontend.
Integrating with unknown APIs	The more APIs you integrate with, the higher the risk. Of course, for many products, this is inevitable. You should just be prepared.	If your budget is tight, perhaps you can do without the integration with the unknown API.
Working with an Existing Codebase	It may seem like building on an existing codebase, whether it's live or halfway built, will save you time and costs. However, this is often not true.	Depending on your situation, it might be better to scratch the existing codebase and start anew. Also realize that

	We see this happen in two forms. **You've fired a vendor or employee.** Unfortunately, companies do not typically switch development vendors because the vendor was doing an excellent job. In this situation, working on an existing codebase is akin to asking contractors to perform structural improvements on a large 100 year old apartment building that has no blueprints. It's simply a risk and there is no telling what may be uncovered. **You're growing and need to ramp up development.** This is generally a good thing. It's typically less risky than taking over code that was poorly written. However, there's still extra time needed to integrate developers into the existing project, and sync up on process and best practices.	any estimate from an outside firm on building on your existing codebase is tantamount to shooting in the dark. How many times has your car mechanic given you an accurate initial estimate? You never know what problems you'll see in the engine (or the code) until you get under the hood and start doing the work. The bigger the app, the bigger the risk.

The Process of Writing Code

Everything else leads up to this point. Using best practices when writing code accomplishes the following:

1. Prevents regressions, wherein new code causes bugs in old code
2. Helps ensure consistently high development throughput
3. Minimizes technical debt—the long-term consequences of writing bad code that must be refactored in the future
4. Mitigates the risk of error and bug-prone code reaching production
5. Allows developers to work on features instead of bugs
6. Makes the product easier to augment in the future
7. Makes the development and product team happy

Here are some of the best practices our team ascribes to—some teams may follow different best practices but this is what works for us.

Tactics	Tools	Benefits

Use a **source code control system** to manage the actual files being produced by the development team.	We use **git**, specifically **GitHub**. Other systems exist (subversion, CVS, SourceSafe) but are inferior to GitHub.	GitHub is built on git, a Distributed Concurrent Versions System, or DCVS, which makes it easy to branch and merge code. This makes teamwork easier. Teamwork, in turn, improves code quality. Additionally, GitHub provides a pull request feature (benefits are described below). Even if you don't use GitHub, *some* version control system is essential to a team of professional developers. Without a source code control management system, you're at risk of losing track of which code is current. There are so many good reasons to use revision and version control that there are whole books about it.
Ensure another developer **reviews and approves code** before the code is deployed to production.	We use **pull requests** to accomplish this.	Using a pull request workflow to introduce changes to production code makes it likely that: ● The delivered code is clear, because the reviewer—who was likely not involved in writing the code—needs to be able to understand the changes made ● That the delivered code is correct, because the reviewer has a fresh perspective on the new code, and is—ideally—looking to find any flaws In practice, code delivered this way is more maintainable, and also requires less maintenance.
Use behavior-driven development (BDD).	We use **Cucumber** and **Capybara** for running automated acceptance tests.	Behavior-driven development produces tests that serve as the code specification, which, when done properly, is understandable to a client or non-technical stakeholder. When non-technical people can understand what the developer is trying to achieve, it becomes easy to determine whether development efforts are providing value. In short, BDD fosters communication, which in turn allows the value of development to be maximized.
Use test-driven development	We use **RSpec** to write and test application	Test-driven development, in reference to the general process of writing automated tests

(TDD).	scenarios for Ruby. We use **Mocha** and **Konacha** to test javascript. We use **JUnit and Android extensions** to test Android.	before the actual implementation, helps developers avoid code regressions, and permits rapid, confident refactoring. With a robust, thorough automated test suite in place, a developer is able to produce highe quality code, more quickly, while reducing the number of bugs that get delivered.
Use Continuous Integration (CI).	We use **Borg (a dedicated CI server),** with the **Jenkins** infrastructure to streamline our integration of code and automated testing.	Continuous integration prevents integration problems when multiple developers are working on the same project. It also helps developers continuously apply testing to improve the quality of software.
Use pair programming.	We use a **dedicated pairing server.** We use **tmux** sessions to share a terminal window, and we use **Vim** for code editing.	Pair programming helps to bring developers up to speed on a new project, allows developers to combine strengths, and helps ensure that code is error-free.

Conclusion

Turning your idea into a product is an arduous process. If you do it right, you will make mistakes, adjust your assumptions, and learn more than you ever expected you would. Whether or not you're experienced in building products, the right outsourced development partner can help you avoid some of those mistakes, and build a more successful application. If you're going it alone, find advisors and ask for help and feedback often—ideally, before your funding runs dry and you're desperate.

Some key takeaways for building a product in any situation:

1. **Change is the only constant.** Be deliberate about building your product, but don't stick to your original plan when all signs are pointing in a new direction. Constantly test your assumptions.
2. **Prepare for roadblocks.** Especially if your project involves some of the risks we covered earlier, hope for the best, but plan for your worst-case scenario.
3. **Communicate constantly.** Communication may be billed as a soft skill compared to coding, but without communication, your product development will flounder.
4. **Take the time to find the right development team.** Even if deadlines are tight, taking the time to hire the right developers, or the right outsourced firm, will save you a huge amount of time and money down the road.
5. **Always strive to improve your process.** The philosophies, tactics, and tools described in this eBook will change over the next few years. In fact, we've had to go back and change some of them while writing. Don't stick to a process just because that's how you've always done it. You could be missing out on a better way.

Contact the Author

I'd love to answer any questions you have about this eBook—or a web and mobile application you're building.

Contact me, and SmartLogic, in your favorite way:

Yair Flicker
http://smartlogic.io/
yair@smartlogic.io
443.451.3001
@yflicker
LinkedIn
@smartlogic
facebook.com/smartlogic