STARTUP FLOW

Get your team up and running as quickly as possible.

Louis Chatriot / Stanislas Marion / Charles Miglietti

Startup Flow

Get Your Team Up and Running as Quickly as Possible

This version was published on 2012-05-15

Contents

Preface

Who Is this Book For?

If you are a **small team beginning a technical entrepreneurial project** and you are struggling to organize your workflow, especially if some of you are working remotely, this book is for you. It was written with both business people and developers in mind as they'll need to use common tools, however there are some more *technical* chapters which are marked as such so that business people can skip them.

Why You Need this Book

You need it because **it will save you a ton of time and make your workdays much more pleasant**. Indeed, when you start working on a startup you quickly realize that you are going to need some tools to organize your workflow. Especially if you have cofounders, you will find yourself meeting too often, repeating yourself, having a hard time following what everyone is doing, and this will be very frustrating for all of you.

In order to fix these problems, **you'll need to choose your tools and design an efficient workflow** through trial and error. **That takes a lot of time**, given how many tools there are and how hard it is to discriminate between them without actually trying them. **We already went through that process, and we compiled our findings** in this book so that you can skip that part and start getting things done with great tools that will boost your productivity. Some of these tools solve issues that you might not even be aware of yet.

What Is in this Book?

This book is a collection of the **opinionated choices** we made. We focused on what we think are the most important components of a startup workflow:

- Communication
- Knowledge sharing and reuse
- Codebase management

With these 3 themes in mind, we offer a selection of tools, one tool for one problem. The tools that we promote are the ones that we use extensively everyday and that we chose after weeks of iteration. Furthermore, they all work very well together. We will describe these **synergies** at the end of the book, so that you can take full advantage of your work environment.

Communication

Communication between teammates is a hard problem. It is necessary yet so complicated to get right. A large portion of communication should be asynchronous, enabling you to avoid useless interruptions that hurt your and your teammates' productivity. You also need to use different channels dedicated to different use cases.

Knowledge Sharing and Reuse

All the knowledge that one finds interesting and usable should be findable by the other teammates, without him having to send a link by email or Skype every time. If knowledge is not shared or not reusable, it's worthless and you will lose a lot of time. This is especially true for Eric Ries' validated learning[1], on which the whole team should be constantly focused.

Codebase Management

In a technical startup, coding is a vital activity. The developers need to be able to code effectively on their own, but also to collaborate seamlessly. They need to be able to review one another's code to ensure overall quality. Teams that don't do that will hit a wall.

If you do all these 3 things right, your team will be so efficient it will be able to build the next Instagram in a week. No kidding!

What this Book is Not

This book is **not** a collection of tutorials. We will not guide you through the process of installing and using everything in details. Instead we point you to quality external resources.

It also doesn't pretend to be the *be-all, end-all* guide on team workflow for every case imaginable. However we feel it is the absolute best for small teams working on a tech project, especially if the team doesn't have an office yet.

And, finally, it is not a startup guide. We won't give you any advice as to how to get your business off the ground since we're not qualified for that! If that's what you're looking for, we recommend Eric Ries' *The Lean Startup*[2], a must-read in our opinion.

Why We Wrote this Book

As we started working on ideas and on our startup, we faced the problem we cited earlier. Our lack of organization was holding us back. So we spent two weeks researching and trying different tools.

[1]http://www.startuplessonslearned.com/2009/04/validated-learning-about-customers.html

[2]http://www.amazon.com/The-Lean-Startup-Entrepreneurs-Continuous/dp/0307887898/ref=sr_1_1?ie=UTF8&qid=1334839735&sr=8-1

After a few iterations we finally reached a very efficient workflow and this investment is now paying off big time. At first this was just an internal effort, but judging from the sheer number of questions[3] you[4] can[5] find[6] on Quora[7], we understood that many people could benefit from our research, and decided to write this book.

How to Contact Us

This is the first version of this book and any feedback that can help improve it will be genuinely appreciated! You can send us an email at book@needforair.com[8] or tweet us @NeedForAir[9]. You can also check our blog[10] out.

Acknowledgments and Inspiration

We would like to thank our reviewers who helped us improve this book:

- Charles[11] for his precise and constructive feedback
- Clement for his early support and his key remarks on the introduction
- Jeff[12], who managed not to fall asleep spotting typos during a flight
- Jonathan[13] and the Vidal[14] R&D team for their very comprehensive and detailed review of the book
- Martin[15] for his thoughtful advice on marketing and audience building
- Rand[16], for being the quickest to give us feedback after receiving our email. And it was insightful too!
- Raphael[17] for is business-oriented eye

[3]http://www.quora.com/Project-Management/Which-is-the-best-project-mangement-time-mangement-tool-and-why
[4]http://www.quora.com/What-is-the-best-way-for-a-startup-distributed-team-to-handle-document-management
[5]http://www.quora.com/Which-Project-Management-tools-are-most-useful-for-a-small-startup
[6]http://www.quora.com/What-are-the-best-productivity-tools-for-entrepreneurs
[7]http://quora.com
[8]mailto:book@needforair.com
[9]http://twitter.com/#!/NeedForAir
[10]http://needforair.com
[11]http://twitter.com/#!/Gorintic
[12]http://twitter.com/#!/jfpetitniv
[13]http://twitter.com/#!/un_john
[14]http://www.vidal.fr/
[15]http://twitter.com/#!/martinpannier
[16]http://twitter.com/#!/randhindi
[17]http://www.linkedin.com/pub/raphael-de-talhouet/15/572/a7a

- Safta[18] for her nice non-geek point of view

We would also like to thank Jason Fried[19] and David Heinemeier Hansson[20] for their book *Rework*[21] which helped us and inspired us tremendously. We recommend you read it!

[18]http://www.linkedin.com/pub/safta-de-hillerin/47/711/a57

[19]http://twitter.com/#!/jasonfried

[20]http://twitter.com/#!/dhh

[21]http://www.amazon.com/Rework-Jason-Fried/dp/0307463745/ref=sr_1_1?ie=UTF8&qid=1334839838&sr=8-1

Communication in a Small Team

Internal communication is hard to do efficiently, and bad communication will dramatically slow you down. Let us describe a few concepts that are the foundations on which we chose our tools.

Synchronous and Asynchronous Communications

Communicating synchronously means that all the interlocutors are communicating at the same time. This is why it is called synchronous. This is usually the case for all voice-based communications, such as face-to-face conversations, phone calls and conference calls. In a synchronous conversation, people expect an instantaneous response from the other participants.

On the other hand, asynchronous communication entails letting people take part in the conversation when they want. That could be a few minutes later, a few hours later, or even a few days or weeks. For that to work, asynchronous communications need to rely on a written exchange. It is not possible to have an oral conversation where inputs arrive every few hours. They just don't work with delay. Examples of asynchronous communication media are letters, text messages, emails, chat or carrier pigeons.

Most people and teams use both forms of communications naturally, but often applied to the wrong usecases. It is not rare for instance to see people send an email and expecting a quick answer, or conversely to see people engage in oral conversations that really should have been asynchronous because the matter is not urgent. In practice, long conversations tend to take the shape of meetings which are notorious productivity killers, while if done in an asynchronously they can yield better results and be less disturbing at the same time.

Most Communication Should Be Asynchronous

The massive use of text messages illustrates the usefulness and the advantages asynchronous communication offers over an oral conversation. Itâ€™s quick and efficient, one can respond hours later and keep track of history. In most cases your communication with your teammates should be asynchronous for the following reasons.

Asynchronous communications don't disturb people. By sending a message to someone, you let her read it and respond when she sees fit. You don't bother the person you are communicating with, nor the other people around you that might overhear your conversation if you're talking in an open space for instance. Distracting your teammates strongly hurts your productivity as a team and is conducive to tensions. Furthermore, synchronous conversations have a high tendency to drift completely off-topic, whereas their asynchronous counterparts are more stable.

Asynchronous communications being written, they are stored somewhere (chatlogs, emails, etc.) and you can go back to it later, to refresh your memory or to bring someone new to the debate. This is important because humans are inherently bad at recalling conversations in an impartial way.

Most topics of communications are not urgent, and most shouldn't require the focus of two or more people at the same time.

When to Be Synchronous

There are a few cases though when asynchronous is just wasting time and won't cut it. Here they are:

- Urgent matters
- Crucial decisions that need lengthy explanations and argumentations. These should be very rare in practice.
- When interlocutors are effectively chatting to each other in synchronous mode and the discussion is going nowhere. In that case it usually feels obvious a vocal conversation will be more efficient and the best thing is to pick up the phone or start a Skype call to settle the matter.

Segmentation by Use Cases

Being asynchronous is not enough. If you don't compartmentalize your conversations by use case, you'll end up using email for everything, and we all know how this ends. Instead you should have different channels for different topics. We have a chat room for everyday conversations (and subchannels inside the chat), Github for code-related conversations, Trello for project management conversations, and email for reports and formal announcements. That way it becomes easy to follow on everything that's going on, and you don't face a mountain of emails everyday. We will detail each of these channels in this book.

Project Management - Do It Like a Pro

Todo Lists

When working on a project, you often think about a new idea or a bug to fix while doing something else and you want to keep track of all these. That's why todo lists were invented. Todo lists are so simple and modular that you can organize and manage all your work with them. For managing our short, middle and long term workload we chose to use Trello[22].

Why Do We Use Trello ?

The common problem with todo list software is they actually end up getting in the way of real work. The classic tragic ending is when people are happy because they just prioritized and assigned 50 bullet points without getting any work done. Two weeks later the todo list is abandoned because it's too much work to maintain.

But Trello is different. We tested Asana[23], RememberTheMilk[24], and simple google docs. None of them survived more than a week. Trello has survived for more than 2 months now and is still going strong. The reason is that at heart Trello is a simplified and collaborative version of Excel: you get columns and cells, but with a fantastic user experience. You'll see that people are naturally used to that kind of visual data-structure and will use it effectively. You can catch a global overview of the progress of your project in one glance. So throw away those 23657 post-its that have been sitting around on your desk for 2 weeks and start using Trello now!

How Does Trello Work ?

On Trello, you create a board (just like a physical one), accessible by all teamates from their computers and phones. Each board is composed of a certain number of lists. On each list every teammate can add items (called 'cards' that are just like post-its), move them within the list or across lists with a simple drag-and-drop, assign them to one or several persons, and archive them when they're taken care of. Notifications are sent for each modification to keep you updated.
What really makes Trello user friendly is its two layers of information. By looking at a board you have all the top level information you need to keep track of the work in progress. If you need more details on a task you can zoom in to the card view where you may add comments, have a discussion with your teammates, browse the card history and so on.

[22]https://trello.com/
[23]http://asana.com
[24]http://rememberthemilk.com

Further Information

Trello's guide[25] is great for getting started.

Real Life Example

We use Trello Boards for this book, our product and our blog[26]. Let's detail the process we used for this book. At first we jotted down our ideas for the tools we wanted to cover. We created a `Nothing done` list and a card for each tool. Then we created the following lists: `Ready for Review`, and `Reviewed`. Any time one of us started working on a new chapter, he would assign himself to the card so that it was easy to see what was being worked on at any given time, and what was left to do. Once the first draft of the article was done, the person who wrote it would move the card to the `Ready for Review` column. The other would now review the chapter and assign themselves to the card once the review was done. Once we agreed on a final version for a chapter, the card was moved to the `Reviewed` list. Any time someone had a question or advice on an item that was still in the `Nothing done` list, he could ask a question or write a piece of advice in the card. Further discussions took place on Github pull requests. This very simple process worked wonders.

For those interested in a usage example in a larger and more mature startup, Uservoice[27] published a great blogpost[28] that details their product management process.

[25]https://trello.com/guide
[26]http://needforair.com
[27]http://uservoice.com
[28]http://www.uservoice.com/blog/founders/trello-google-docs-product-management/

Version Control - Safely Iterate on Your Code

technical

This is probably the **single most important tool** you will need apart from your brain, your hands and your computer. You might not realize it now if you never had to use a version control tool before, but you'll see.

SVN users, don't skip this chapter! It explains **why Git is genuinely better than SVN**.

Version Control, What Is It ?

When you learn how to code, you start naming your files like `my-cool-program`. Then you want to make it better, but you are scared to overwrite the previous version because it might not work. So you wisely name your new version `my-cool-program-v2`. You show it to a friend of yours who thinks what you built is really cool and he is going to try and make it even better! So you email him the file and the next day, lo and behold, you get a `my-cool-program-v3-reviewed-by-friend` in your inbox. What if you want to accept only a few of his modifications? What if after doing this you want to rollback to your `v1`? It gets really messy real fast and then it gets worse, and then worse again.

So then you think there must be a better way to do this mustn't it? As any good programmer, you want to automate stuff so you can focus on your code instead of managing versions.

The good news is that there are tools to deal with this problem. They are called Version Control Systems (VCS). Using a VCS allows you to experiment with your code while always keeping a stable version easily. They allow you to show your teammates and the world what you worked only when it is ready, without messing up the program for them in the process.

The Ultimate VCS: Git

Git has been invented by none other than Linus Torvalds[29] a few years ago, and is the most elegant and convenient solution out there. It is straightforward, easy to use and incredibly powerful. The good thing is that the learning curve is very smooth. You can start using it for simple tasks at the beginning and learn more complex stuff as you go.

[29]http://en.wikipedia.org/wiki/Linus_Torvalds

Installing Git

You will find simple installation instructions here[30]. You can either install from source, with a package manager or with an executable, depending on your OS.

How Does Git Work?

Basically what Git does is track modifications of your files and allow you to take a snapshot of the current state of your files at any given time. When you have a number of snapshots, you can navigate between them, and follow the history of your code.

The inner workings of Git are well explained in the Pro Git book[31]. You should read the first three chapters to understand how a modern VCS works. You will learn what a repository, a commit, a branch, a merge, and a checkout are. After this, you will be ready to start using Git, and soon you'll feel really comfortable with it. **It is crucial that you read these three chapters to understand what version control is and how it works**. This won't be a waste of time: as you begin to use Git's more advanced commands, you will find more than helpful to have a real understanding of what you are doing.

Git versus SVN

This paragraph is geared towards SVN users, not version control beginners

If you have used SVN before and you're wondering why you should switch to a new system, then here's why:

- Branching[32] in Git is simple, natural, fast and makes for great unobstrusive collaborative workflows
- Merging is a breeze: ever had problems with merging in SVN? Having to choose between a reverse merge, a branch merge, a something else merge in TortoiseSVN? In Git, merge just works, another reason to go on a branching spree!
- You can work and commit **without having to be connected to the server**. I'm not kidding, you're not dreaming. Yes, you really can keep track of different versions on a plane!
- The CLI is fairly straightforward and lightweight
- Github[33] was designed to work with Git and makes your team even more productive

[30]http://progit.org/book/ch1-4.html
[31]http://progit.org/book/
[32]http://en.wikipedia.org/wiki/Branching_%28software%29
[33]http://github.com

Feature Branch Workflow

Scott Chacon, author of Pro Git book[34], and other fine folks working at Github[35] recommend a workflow[36] based on feature branches. Basically one starts a new branch from master every time one wants to develop a new feature. This allows to keep an always deployable master branch, and always have a clear view of what features are being developed. It also mitigates the risk of two branches overlapping in scope and ending up conflicting.

Real Life Example

Charles wanted to build the *handling PATCH requests* feature for our API. He created a branch named `handle-patch-request` and started coding. After some time, there was an urgent bug *some special characters not displaying properly* to fix, so he simply commited what he was working on, switched to the new feature branch `hot-fix-special-characters`, fixed the bug and pushed to `master`. He then resumed his work on the *handle PATCH requests* feature. Meanwhile, Stan reviewed his hot fix and merged it into `master` to make it go live right away. It really was that easy.

[34]http://progit.org/book/
[35]http://github.com
[36]http://scottchacon.com/2011/08/31/github-flow.html

Who Are We?

Need for Air[37] is composed of Louis Chatriot, Stanislas Marion and Charles Miglietti. Sharing the same passion for the internet and entrepreneurship, we quit our day jobs at the same time to launch a company together. We are working on making good content more easily discoverable through better curation. Previously we learned to work together on our first (and only so far) freelance mission for, Captain Dash[38], Laurence Parisot[39] and the Medef[40].

Louis Chatriot

I'm a long-time geek who's always wanted to understand how things work, especially the fields of math, physics, computer science and economy.

After 6 months as a software engineer at Robinsons Participations and about 2 years as a management consultant for Bain & Company, I decided to launch my own company with Charles and Stan.

I graduated from Ecole Polytechnique (majored in CS) and have a MS in Management Science and Engineering from Stanford University.

Stanislas Marion

I've been meaning to start a company for a long time. I quit my job without a firm idea nor a cofounder. It didn't take long to find both. In the past I successfully played online poker at middle to high stakes, which taught me a lot about risk-taking, long-term focus and short term swings, adapting and the value of money.

I worked for about 2 years at a Parisian startup[41], where I learned a ton about small team dynamics and problems. I graduated from Ecole Centrale Paris and have an MS in Management Science and Engineering from Stanford University.

Charles Miglietti

I do what I love and I'm always on the lookout for a good challenge. Being passionate about computer science and math, as well as innovation,I became more and more interested in entrepreneurship.

[37]https://twitter.com/#!/NeedForAir
[38]http://captaindash.com
[39]https://twitter.com/#!/laurenceparisot
[40]http://www.medef.com/medef-corporate.html
[41]http://yseop.com

After graduating from Ecole Polytechnique, I spent a year as an embedded systems engineer at Withings[42], one of the most promising French startups at the moment. Inspired by this experience, I decided to quit to start my own company.

[42]http://withings.com