[DRAFT]

Lean and agile essentials for startups

A must-read guide before your tech startup fails

Martin Chapcak

Introduction

I guess many people told you that 'software development' is a young and difficult discipline. I was also told the same. When I was in university, I applied for the software development management course. The first class was crucial for me. A seasoned professor, with many practical skills, came to the class and got my full attention within the first minute. "Most software projects fail! They are delivered later, cost more than expected or do not have required functionality." The professor explained that people have been building wooden houses for thousands of years, with iron for hundreds of years, but what about software? Just a few decades!

Software Development is indeed difficult, but the causes of failed projects are rarely technical. Unfortunately, Software Engineering rarely teaches how to manage software development. People are used to their outdated (mostly wrong) behavior. I have witnessed some managers that quit their stable corporate jobs, hoping to build a startup. But their mindset, shaped by a corporate lifestyle, force them to build a Titanic (corporate) instead of speed boat (startup). This publication plays an essential role of a lean and agile transformation of your team or company. Whether you are founder, CEO, CTO, developer or product manager, this publication is here to guide your next software project to success.

Story

I did not care about agile, planning, estimation or other management skills at the beginning of my career. I was passionate about code and I did not care about anything else. I was on the right track to become an adequate coder. By acquiring coding skills, my understanding of software was more broad and in-depth. I came to believe that knowledge of programing is not enough and I started to explore other areas. I quickly learned one thing -Software management is crucial for a project's success! I thought that this is the area where I can efficiently bring the most value. I know of many developers with ZERO management skills and managers with NO software understanding. So I enrolled in a Project Management class in university, taught by Senior Project Manager from a global corporation. He lectured about real projects and real problems. He described everything from planning to pricing and actual execution. But during one class, I heard the shocking truth: "Once we get this detailed estimation, we have to add some time buffer. Industry standard buffer is 30%." I was shocked. It took so much effort and time to make an accurate estimate for all tasks and then he messes up that by some constant. But it was not his fault, he just adapted to the environment he was working in.

This book is for people involved in planning or managing software teams. Readers should be self-aware enough to realize that they are potentially headed in a wrong direction with their team, or even the entire company.

Software development process

You do not need any development processes if you are working on a project alone. You can hack things on the way, make quick decisions and have control over software. But once your team starts to grow, you will need some methodology or process which will guide you. Two most

popular methodologies used to develop software (and only two I know :) are “waterfall” and “agile”. These methodologies are often misunderstood and misleadingly defined.

Waterfall

Long time ago, software was made with process called Waterfall. All development was divided into sequential phases. Analytics wrote detailed specification documents, architects prepared big models of architecture, developers developed systems and testers tested them. Then operations teams deployed system to servers and another group took over maintenance.
Requirement -> Analysis -> Design -> Implementation -> Testing -> Deployment -> Maintenance.
This framework was working very well for some companies, but not so well for other. The biggest pitfall was that it took many months or years to go from requirement phase to deployment. This long period caused most of the problems.
Waterfall gives false belief that the product will be delivered in a given time. It looks attractive to management because it seemingly predictable. Companies love to hear that software will be done in the given timeframe, based on detailed specifications and with a fixed price. It sounds too good to be true.
It is not a surprise that many companies are using waterfall again and again. Just as a long time ago, some companies were doing well, but most of them were failing with their software projects. Why are people, whether in small startups or big corporations, always attracted to the waterfall?
[http://www.ambysoft.com/surveys/success2013.html]
http://blogs.gartner.com/nathan-wilson/the-end-of-the-world-as-we-know-it/
http://www.scruminc.com/yet-another-waterfall-project-failure/

Birth of waterfall

It is 1970 and Dr. Winston W. Royce writes “Managing the development of large software systems”. The document is later recognized as the ground stone of waterfall. Right at the 2nd page, he writes “I believe in this concept, but implementation described above [waterfall] is risky and invites failure.” Yes, here it is! Here he describes how waterfall is bad for big systems. At the same page, he continues “ [in case of problem] Either the requirement must be modified, or a substantial change in the design is required. In effect, the development process has returned to the origin and **one can expect up to 100-percent overrun in schedule and/or costs**.”
Looks like everybody have read only first page and nobody looked at the second one! Why would anybody use waterfall for a big project? Only one reason, I can come up with, is waterfalls’ misleading beauty. Predictability is every manager’s dream. Clearly defined steps with linear execution. It fits beautifully in every slideshow. It is perfect for Gantt diagram to estimate delivery date. Waterfall is clean, well specified and without surprises. But it has one problem: It just does not deliver. We knew that 50 years ago and it still holds up today.

Waterfall successful use-case scenario

Waterfall fails in large scale software projects, but what about small ones? It, surprisingly, works quite well. Is the startup you are building big or small software project? I bet you think it is big, but the reality is more likely to prove that your beta app should be a small project. Even if this

pre-condition is met, small projects still tends to get significantly delayed. So why do these projects fails? Why does the waterfall not work for small project where it should? My theory is harsh: incompetent people and lack of resources. Waterfall requires at least 3 steps before actual coding. But reality in startup tells me that we cannot expect more than two phases. Company simply does not have enough people with required skills and enough time to execute at least 3 steps before coding. Once the startup violates rigid waterfall process, the next destination is always hell.

Waterfall conclusion

Pure waterfall simply does not work for big projects, but could be used for smaller apps. Waterfall is rigid, slow and resource-hungry way of delivering small projects. Startups rarely follow waterfall's best practices, resulting in failed projects. It is important to point out that even if waterfall delivers what was specified, it does not evaluate the quality of the specification. In other words, you can build an app nobody needs. Is there some way to build a software with limited resources and product evaluation?

Agile

New, cool, disruptive, silver bullet, popular and effective. Yes, everybody is talking about great agile. "Did you hear about Scrum? You must do Scrum! Everybody is using Scrum! If you do not know how to start with agile, pick the Scrum. Buy a book, get training and certification to become an agile master ... oh sorry ... Scrum Master!".
Yes, agile seems like a big hype, but it is also able to deliver in time and within budget most of the time!
I believe that agile is the best methodology suitable for majority of software projects. But why only a handful of companies really do agile successfully?
[http://intland.com/blog/agile/failure-of-agile/]

Lean & agile

These terms are not the same neither are they equal. People usually hear agile first, and later, they discover lean. I have read a nice definition of agile and lean by Tami Reiss from Pivotal:
[http://blog.pivotal.io/labs/labs/agile-vs-lean]

Agile
A development process that emphasizes short iterations and focuses on delivery of functional software.
Flavors of agile may include continuous deployment/integration, test driven development, pairing, scrums, sprints, or many other facets that are too long to list here.

Lean
A process framework with an attempt to minimize risk and waste while maximizing customer value.
It is often achieved through a tight feedback cycle with minimal investment between loops. Lean protocols have been employed to optimize manufacturing, construction, and pretty much any business process you can think of.

http://blogs.gartner.com/nathan-wilson/welcome-to-the-post-waterfall-era/
http://blog.toolshed.com/2015/05/the-failure-of-agile.html
http://agilemanifesto.org/

Waterfall or lean & agile

Building software is difficult. Old fashioned waterfall requires lots of management to create specifications and then even more to build and deliver a project in time. And because waterfall consumes lots of time, it is likely that specifications will change during development. Outdated or changed specifications is one of the biggest causes of pain. If managers and developers do great work and finish projects, there is still a risk of somebody saying: “We do not need it anymore”.
Agile and lean try to eliminate these problems. Software and business requirements are constantly evolving. The problem is that people and businesses are used to work in the linear waterfall way. The challenge is to transform from the “Good manager should work on big and time-consuming projects” way of thinking to “Good team quickly delivers solution to a client”.

Effectivity

Methodology itself is not as much important as result. However, the main focus of software teams should be efficiency. You have to focus on value. Following a set of rules gives you a better idea of how to be more efficient.

Pareto Principle (80-20 rule)

The rule is that 20% of action (work) produces 80% of results. Therefore, remaining 80% of work produces just 20% of result (and is just basically wasting your time). Your goal is to do just the right 20% of work which gives you that sufficient 80% of result. Identify that critical scope and eliminate rest.
There are two steps to apply 80-20 rule. First step is to qualify the activity which gives you the most results. Step two is to try eliminate other activities by simplifying them to its minimal form or by delegating.

Parkinson's law

“Work expands so as to fill the time available for its completion.”
[https://en.wikipedia.org/wiki/Parkinson's_law]

Having 3 months to do some work means you will spend 3 months working on that, or probably more. If the main feature is easy to build, you will always find so many unimportant functions to add to fill that 3 months deadline. Having strict deadline for building software which would usually take 6 months forces you to be creative, cut all non-essential work and deliver in time. On the contrary, you can also build crappy software with bugs and blame tight deadline.

Parkinson's law in startup and enterprise project

Example

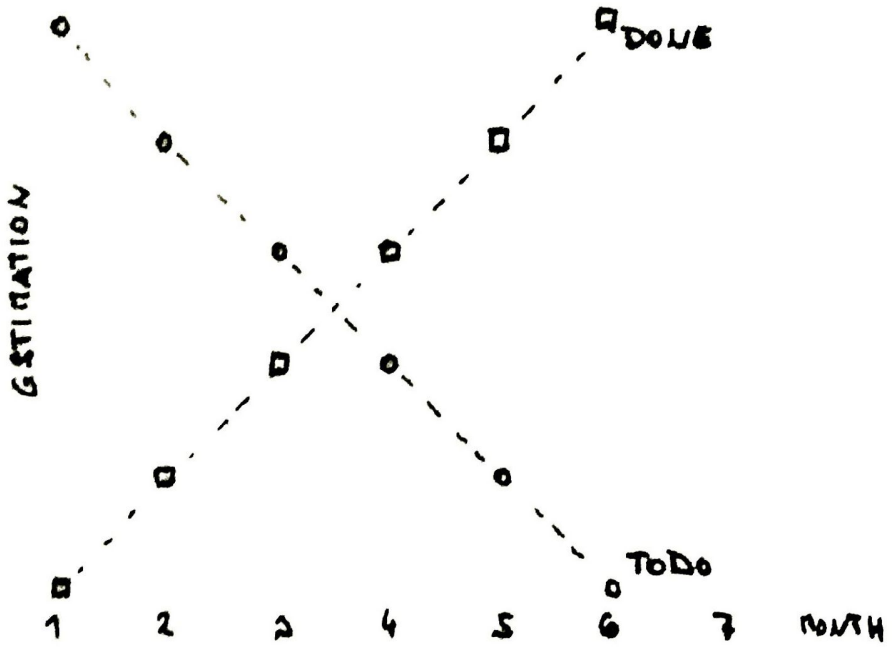
A Client asks you to build a chat app in 2 days. Just two days for the whole app, which should be then released to the public. How do you develop that app? You start with a short analysis. Ask as many questions as you can. Who is user? What problem do I want to solve? How should the app look like? Blue or black? You make quick mockups, minimize the app just to three screens, find the simplest back end as service (baas) supporting sockets, work with simple JSON structure, build fast and test often. You show final app to your client 48 hours later! I know you can do it. Just challenge yourself and find the easiest way how to make it happen. An app like that would be far from perfect but you would have a functioning product which can be improved later after launch.

Imagine a similar scenario. You are the developer and your client asks you to build a chat app. But now, your deadline is 200 days.

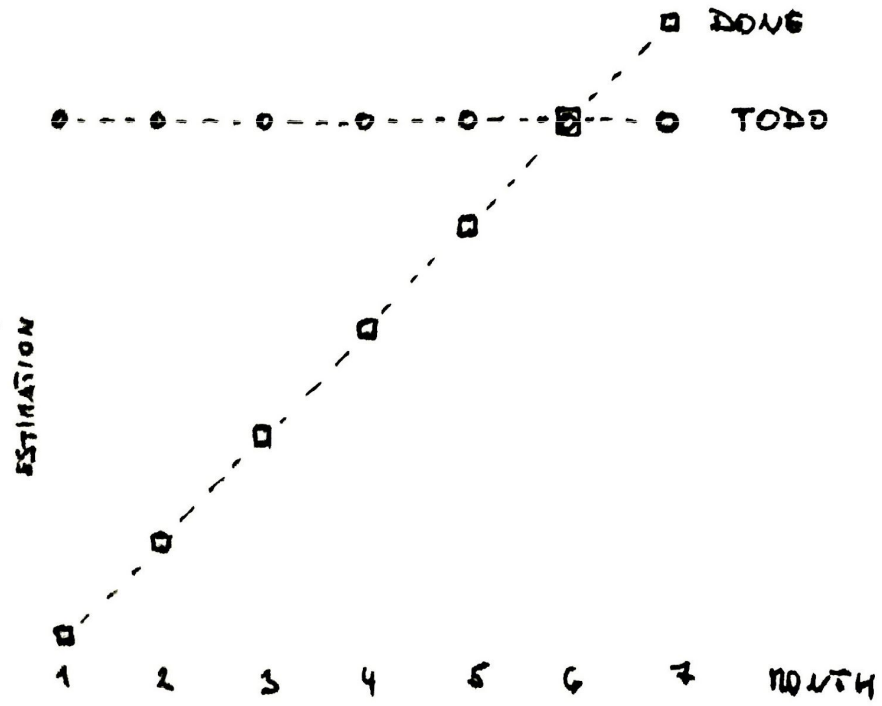
Example
You have 200 days to build chat app, that is one hundred times more than previous deadline! You can spend more than nine months on that one app. If we think about time as money, you have 200 man-days. You hire a designer and you both have 100 days. And then, because you have plenty of time, you start work on the best chat app ever made! Chat function is essential, but what else to add? You have almost 5 months! Emoji icons, stickers, in app purchase of stickers, 3rd party stickers, animated stickers, sending photos, sending videos, sending voice messages, voice call, video call! Best chat ever, right? It should be prepared to scale! You need the best cloud back end solution, maybe building custom back end! What about new cutting edge technology you never used before? Yeah, let's build the biggest chat app ever! Let's build a big, slow, bulky app with functions nobody use! The funniest thing is that you will deliver after deadline, I bet on that! You have one hundred times more time on the chat app and you are late, because this is what happened in long term projects!

Do you understand the Parkinson's law now? Slightly related to this is "scope creep". Issue that your scope is getting bigger during execution, which delays delivery date. The cause can be external (stakeholder) or internal (dev. team). The rule of thumb is to not expand the current scope, but to iterate after first release.

Expectation



Reality



You will never finish your project/product unless you finish all your TODOs.

Agile is your product rocket fuel

Do you think that agile development is not for you? Then either you are a big corporation with a huge budget, or you will be out of business very soon!

Growth

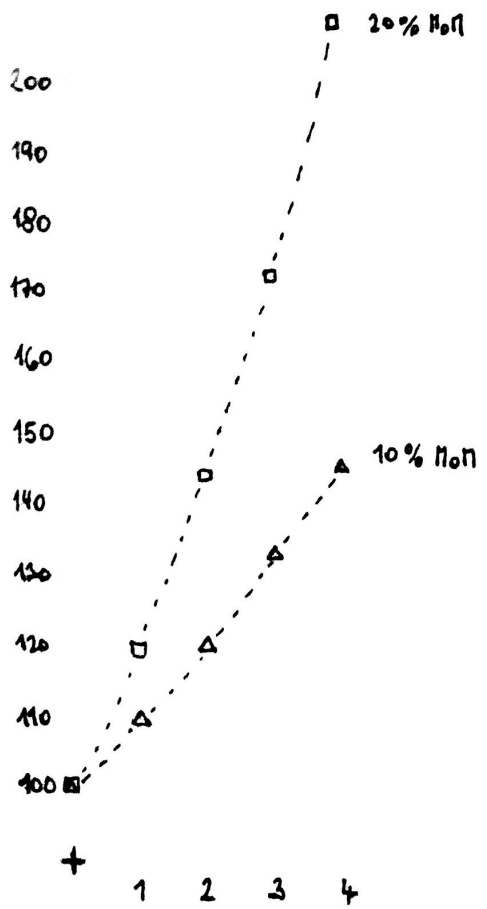
Growth works like interest in your bank account, it compounds.

- Absolute interest is driven by size of your bank account. Compounded growth is driven by frequent payment of interest.
- Absolute growth is driven by the size of your business. Compounded growth is driven by frequent execution. In software, execution is synonym to deployment.

[https://medium.com/uber-for-x/startup-your-engines-43a6ec57891b]

MoM growth[%]	1. Month	2. Month	3. Month	4. Month
10	110	121	133	**146**
20	120	144	173	**207**

After 4 months with 20% monthly growth, your business is 42% (207/146) larger compared to 10% month growth. It should not be surprising that you want to maximize your growth.



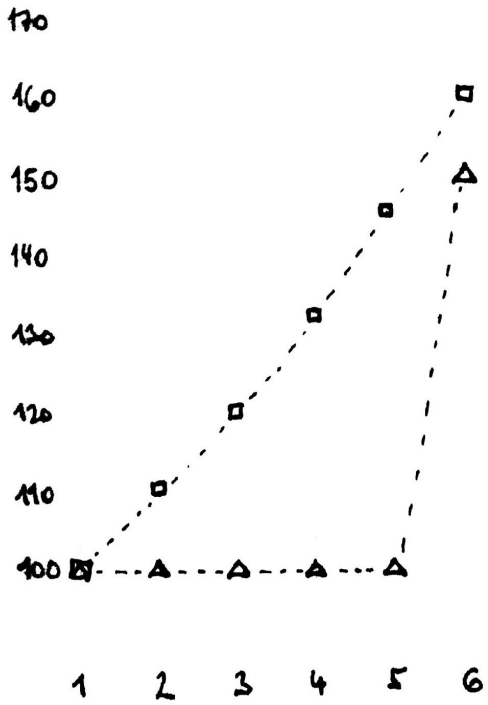
But you also have to get all the growth juice as soon as possible. In the next example I will show you why you need to grow in small increments now rather than a lot later.

Example

Your revenue at the beginning is 100 [M$?:)]. You have 5 independent tweaks where each one will take 1 month to develop and increase your conversion rate by 10%. If you release these improvements together at the beginning of the 6th month (waterfall), your revenue will go up by 50%. If your release feature every month (agile), your revenue will increase by 61%.

Method		1. Month	2. Month	3. month	4. Month	5. Month	6. Month
Waterfall	release	no	no	no	no	yes	no
	MoM revenue[%]	0	0	0	0	0	50
	Total revenue	100	100	100	100	100	**150**

Agile	release	yes	yes	yes	yes	yes	no
	MoM revenue [%]	0	10	10	10	10	10
	Total revenue	100	110	121	133	146	**161**



This math is not here to show you how much more you can make using agile methodologies. This simple short-term calculation is here to demonstrate one of the benefits of agile.

Product matters

Let's be honest. Companies often build crappy, slow, ugly and too difficult software. I do not know of any developer or company who wants build bad software, but they usually do. It hurts to see how much time, energy and money they waste. Please, do not waste people's time. I am sure you hope to develop software people will love, in short time and with limited resources. I have good news for you. You can build a great product with little money and a few good developers.

Bigger is not Better

Three talented people are enough to develop any software product. If you think you can not develop a product with one small team, you are not going to develop it with big team.

Story
One company I was working for, had a working product which was maintained just by two developers. Company decided to build a new version of the product from scratch and to do it quickly, they hired more developers (including me). I have experienced growth from 3 to 6 developers. Expectations were optimistic and we hoped to develop the new version in just a few months. We were a good team with great support from managers. We were full of energy and ready to build a remarkably great product.
Six months later, we even did not start to work on the new version. We got twice more work to do on the old platform. Therefore, the development of the new platform was postponed every month. Stakeholders had always found some missing functions which were more important than the new version. They knew we had more developers, so we could do more work. Required features were usually small, but the overall complexity of product increased rapidly and every new line of code slowed us down. More functions, more code, complexity and also more bugs. It took 10 months to start building new version. We were 10 months late with twice the team's size! That was just beginning of the project. I will keep some juicy details about finishing product for my next book.
Note: I think that building a new version of a product from scratch is generally a bad idea. Based on my experience, slow transformation of code from v1 to v2 is usually the best approach.

The problem is not in how many people you have. The problem is in the way you develop a product.

Big picture - How to develop software/product

6 steps

Step 1: Listen to your customers
I cannot stress this enough. You must meet customers and listen to their needs. Your vision might be important but your customer satisfaction is way more critical.

Step 2: Try to find a solution
This is your show time! Show off your visionary solution. Write, draw, prototype, sketch, mock ... do whatever it takes to find the best solution.

Step 3: MVP
Take your solution and cut it down to MVP -> minimum viable product. Put off all cool features you would love to have, but are not essential to solve a problem. The goal is to define the smallest scope which solves a problem. All other features you dream about can be implemented later, they are not part of MVP.

Step 4: Build

This is the longest part. Lock a few developers in the room for a few weeks with MVP specifications and observe what happens :D.

Step 5: Show time
Take your MVP, show it to client, and listen.

Step 6: Repeat
Go back to step 1. If you feel like you have created right solution for client, you obviously wouldn't build new MVP, just iterate over existing MVP and maybe add some features.

Example:

Step 1: Listen to your customers
Customer has a problem: he needs to get from one city to another, whenever he wants, as quickly as possible.

Step 2: Try to find solution
So you decide to build a car. You start with drawing board and draw the most beautiful car you have ever seen. And you add features you would like to have in a car … many features. Music player, automatic opening door, parking camera, etc.

Step 3: MVP
MVP … OK so what is a car? Engine, 4 wheels, 1+ seat, steering wheel. I think that is all we need -> MVP.

Step 4: Build
Build that ugly, barebone car.

Step 5: Show time
Let your client drive your car.
If your client has needs to move from city to city, he will not demand a music player, or parking camera. You will solve his problem and he will be happy. If your customer gets angry when you tell him that music player is not available, then his real problem is not to move from place a to b. Maybe he already has a car and you are just building a solution which already exists. Maybe his problem/need is to move from place a to b with the highest comfort possible. He actually wants great a sound system, 5 zone air condition and seats with massage function.

Step 6: Repeat
You need to build a luxury limousine, which takes a long time, or you can pivot. Build luxury car sound system.

No ‘fix scope’ projects anymore

What have you learned from previous chapter? That building software is hard. You must work smart, keep things simple, plan for small scope and set up short deadlines. Previous parts were focused more on how to build products, but sometimes your challenges are about project management. Situation when you need to estimate costs of project or set deadlines. I will give you a small example of how a project planned with fixed both time, and scope can harm your product.

4 most important project questions

Vision: The most important. Ask yourself what problem do you attempt to solve?
Scope: How to solve the problem? Description of solution.
Time: When does it have to be shipped to customers? How long will it take to develop?
Money: How many developers do you have?
[https://support.office.com/en-ca/article/Every-Project-plan-is-a-triangle-2b74c21b-a406-4727-8d74-26648a56924a]
Scope, time and money can be fixed or variable. Variable is when your team can freely estimate a given element. Once you put an estimation on a variable element, it starts to behave almost like fixed one. Fixed is considered when stakeholders strictly estimate a given element before talking to a team. If time is the variable, stakeholders will ask you how long will it take to build that. Once you give estimations, you also create strong expectations. Money is variable in theory but size of the development team (which affects money) is usually more or less fixed. Throwing more money on your short project will not make it shorter. Hiring more people for already running and delayed project rarely increases productivity.

Fixed scope and time

Having all elements fixed puts big pressure on developers, especially if the deadline is tight. You can be almost sure they will not finish before deadline. They will probably ship later. Where did you get that deadline? How do you know how long it takes? If they have too much time, they are not effective (Parkinson's law). If they have too little time, they will deliver low quality software, miss some features from fixed scope or just be late. None of that is good.

Fixed scope, variable time

You give developers freedom to set their deadline. Result depends on their abilities, background, personality and understanding of business needs. The team has to do detailed analysis on tasks and hold several meetings to estimate time and commit to a date. Their deadline decision puts constant pressure on them. Because of that, and to feel safe, the team will also allocate some buffer time they expect to have in an emergency case (emergency case happens in all projects, all the time).
Having a buffer in projects just shows how terrible your planning is. Imagine you spend lots of time on estimations and then you just mess up this precise number by adding some buffer (usually 15-30%). If you do not believe in your estimation why do you do estimation at all?

Fixed time, variable scope

Variable scope claims that vision is the most important. Discuss your vision with developers, emphasize the problem you want to solve and observe what solution they propose. This works for smaller projects and you should consider the risk that solutions do not need to meet your expectations. Scope can be slightly defined but it is not required. Your team can build something different from you might think but the solution has to always solve the main problem. You can get creative with your solution but there is also a big risk factor in team quality. Experienced teams should deliver good products in time! Inexperienced team fails badly.

Variable scope and time, no deadlines, no estimates

Give your team a vision, without specific tasks and without asking for any deadline. When you have a problem and conventional solution does not exist or does not work, it is time for disruptive solution. Development becomes a constant stream of small experiments without fixed deadline. It makes no sense to put deadlines or scope pressure on a committed and experienced team.

Scope and time conclusion

First two approaches with fixed scope puts pressure on a team and are tied to waterfall and similar variations. Fixed scope requires more planning and preparation. The more you plan, the more difficult it is to change direction later. And because you know changing direction later is difficult, you plan even more and you are digging deeper in the hole.
Last two methods, with variable scope, are tied to agile methodologies. Loose scope is variable in details but not in vision. Iterative development is required along with an open mind and willingness to change.

Iterative development

Essence of agile, but often misunderstood or ignored. Iterative development means that you build a small piece of functionality, release to the customer, see how it works and continue with another small piece.

Example

Your scope contains features A, B, C, D sorted by priority. Your plan is to build all features but you are also aware that something may change and you will not able to finish it all. You should start with feature A, build just the most important part, far away from perfect, and show to the customer. When the customer likes it, you can improve it and later start with task B. If your deadline is approaching, you have already built features A, B and C. Most important function A is well built and B+C are satisfying for customers but not perfect. D is not so much important for client so it is OK to release it little later.

Up and running

You want to have your servers up and software running! Source code in your code repository is useless. Product managers and developers got paid but result of their work is just sitting in a repository and doing nothing. Imagine all that wasted time of developers if software is not “Up and running”. Your goal should be to build fast, release often and get valuable feedback. Find

and eliminate every distraction which can postpone this process. Once you lose control over scope, project size will grow, expectations will be much bigger and delivery time longer. Advice to release often sounds simple but many people try to do opposite. They want to show how huge work they have done. Avoid big fancy releases with champagne unless it is really necessary! Release as soon as possible! Keep your servers up and new code running in production.

Failure

Most startups fails, that does not come as a surprise. Imagine, you are an investor and you meet a college student who says “I want to build a startup which enables people to share their videos. We allow anybody to upload unlimited video for free and we will make money from advertising.” Sounds like a bad business model for me. I would not invest. But I am wrong, because this platform called YouTube turned to be great business. Or, you meet another young entrepreneur who says “I want to build a page where people can publicly share their everyday thoughts by writing short status updates. It is completely free and we have no idea how to monetize it.” I would not invest into … Twitter. As a VC, you could also hear “We will make software which shares your files between computers. We know there are many competitors already but we will make it much better.” Everybody tries to do it better than competitor. Why do you think you will be successful mr. Dropbox?

Many successful startups threw away their first idea and pivoted into new and successful stage. It is likely that you will be completely wrong with your assumptions at the beginning. You should expect many small failures which lead you to success or bankruptcy.

Prevent painful failure

Some entrepreneurs come and say: “My team will build solution S to solve customer’s problem P and ship product in time T.”

Usually T is very long period. Did they prove that their assumptions are right? How do they know customers have this problem P? How do they know, solution S really solves given problem? How do they know their market does not change during that long T period and their solution will not become obsolete?

I have heard many entrepreneurs saying that failure is good because you will learn from mistakes and do better next time. I agree with that, but there are different types of failures. Some failures are like small experiments and it is necessary to experience them. Others are big, painful and you want/should avoid them. Agile and lean is here to give you framework which favours small experiments and avoids expensive ones.

Fail in the waterfall

Example

You think about feature A, you make it complex and robust, with perfect design and sliding effects. You had plan to develop it in 3 months but because it is so robust, new and prepared to scale to 1 million customers (even no one is using it right now), it takes 6 months. You deliver this bulky piece of code to clients and hope they will like it. But they just do not use it so much, it is complex (and maybe buggy), and you have this huge expensive code which does not solve the problem. Just a few people use and it has to be maintained forever! You can not just delete

a half of a year worth of your work (managers says) to make your code clean/lean and easy to maintain. Welcome to the road to hell.

Failure case: You have wasted half of a year and you will spend a lot of time on maintenance.
Success case: Customers wait 24 weeks for polished the features they love.

Fail in the lean

Example

You think about feature A, you make it as small and easy as possible. You implement that, ship to customers in 2 weeks and watch metrics for evaluating outcome. If customers use this feature, you can repeat process. If they do not use feature, it is probably not useful for them. You can/should delete this function which nobody use and start to think about different feature B.

Failure case: You have wasted 2 weeks.
Success case: Customers wait 2 weeks for simple features they like and
customers wait 22 weeks for polished feature they love.

Fail fast

Painful failure is one which costs you lots of money. I hope, this small mathematics exercise, will make it pretty obvious that failing in agile is cheaper than failing in waterfall. We can use simple math to show how to minimize tragical failure.

P(success) - probability of making "successful product" [%]
P(success) = P(problem exists) * P(correct solution) * P(right market timing)

P(problem exists) - *probability of correct analysis that "problem exists" [%]*
P(correct solution) - *probability of "correct solution" which solves problem [%]*
P(right market timing) - *probability of right "market timing" so market stays same for given period T" [%]*

FC - cost of failure [money]
FC = DC * D
DC - cost of one MD (day of development) [money]
D - number of days [day]

You want to have high probability that product is right P(success) and low cost of failure FC. The most practical way to minimize failure cost is to develop and ship as fast as possible. Time (D and P(right market timing)) reflects scope size. Make scope small and you will fail fast. Building startup is about how many trials and errors you have before you run out of budget.

Building startup is like stock trading

An experienced trader does not bid all his capital on one trade. He knows that the risk is too high. A professional trader buy stock and sets a limit for his loss (Stop-Loss Order). The trader sells his stock with a small loss if stock goes the opposite way he anticipated. The trader tries to hold his position in profitable trade as long as possible so he can maximize profit. Be like a professional trader and limit your loss by limiting time you need for feature development.

Implementing lean and agile

You already know you have to do agile, right? You have read many articles how agile/Scrum is great, how it boosts your productivity and makes customers happy. It is not new. Kanban/lean is here since 1960s [http://en.wikipedia.org/wiki/Kanban] [http://www.toyota-global.com/company/toyota_traditions/quality/mar_apr_2004.html]. Scrum itself since 1995 [http://www.scrumguides.org/scrum-guide.html#acknowledgements-history]. So why just a few companies do agile successfully while most of them fail? Agile is not about framework, methodology or kanban board. Agile is about the mindset of delivering quality software to the customer.

What is quality software

"Quality is whatever is important to stakeholders" Stakeholders include also your customers and honestly, most people do not know their customers as well as they would think. You do not see into people's mind and even customers do not know what they want! Customers tell you what they think they want, but it can be very different from what they actually want. Do not be stupid, egotistic or naive to think you know exactly what customers want. Go to higher conscience level and accept that you just do not know. That is not problem, that is an opportunity. You have the ability to learn about your customers through research and get data which shows their needs. You should do real research, not just develop some functions based on your own (often wrong) assumptions.

Measure everything

Key principle of Lean and agile is Build -> Measure -> Learn loop. Measuring results is important ingredient to successful Lean and agile but many people just simply skip this step and later fails terribly. Firsts question we have to ask is: "What is our result we want to get? What do we aim for?" Second question is: "How should we measure that?" Answers, I will give you, are simplified and covers just one small part of this big topic. However, this helps you to get clues as to what to focus on and how to start.

What is result

Result is how many Story Points we finish in Scrum sprint.

I can imagine the manager's report which point out that the team finished tasks worth 30 story points in sprint. Nothing can be more wrong than focusing just on working hours, story points or any other metrics tied to time. It shows quantity of work done but not effectiveness. We often want to finish more tasks and deliver more functions. But are all tasks really important? What if

we skip tasks which do not make a better product and does not satisfy the customer? Counting delivered Story Points is a common practice to measure your team, it says nothing about your product.

Result is how we solve customers' problems.

If you want to build a software which solve customer's problems, you must measure your customer's satisfaction. If you release a new version of the product with ten new features and people do not use them, your result is zero ... no matter how many hours you spent on that or how many tasks were finished. Feature impact indicator should be on top of your lean metrics.

Measure impact

Every code you write should make something better, faster, easier, nicer or cheaper. These improvements can be measured, but we usually do not measure it. We do not want to spend extra time on features, believing that our solution is already good. We think that our progress is how much work we do, but truth is that progress is how much better product we create by measuring key indicators. Build analytics into your product once it is released to public. Think about indicators for every feature/improvement you make.

Example

You add an export function into your app. Every click on export button will trigger an event in your analytics. You check your analytics after a week and almost nobody clicks the button. Now you know you have some problems. You would not know about the problem without measuring clicks. And time needed for development would be wasted on functions nobody uses. You decide to make the button more accessible, deploy code and look at analytics again. You can also send emails to customers about the new features later and again see of they use it. If they do not use the functions, then you made the wrong feature, lesson learned. You can delete that function, keep your code and UI clean and look for ways on how to improve your research about the new functionality. If they use that function, you have done a great job and you have proof of that. One MVP trick is to build a button with analytics even before actual functionality. You can measure if users click on that button or not. Think about how much it costs to test an idea with this fake button when compared to building a new export function.

Politic campaign success

Story

A famous example of well executed project is the donation page of Obama campaign. Developers decided to use simple technology like HTML and little of Javascript instead of advanced languages like php/ruby/java/.net. They did not have any scaling problems because they used a simple static page, deployment was fast and no complex architecture was required. The team could be focused on the most important task to make the campaign successful. They ran hundreds of a/b tests to tweak the page to its maximum performance (conversion rate). The campaign was successful because of short build-measure-learn loop time period, not because of some new magic JS framework.

[http://kylerush.net/blog/optimization-at-the-obama-campaign-ab-testing/]
[http://www.wired.com/2012/04/ff_abtesting/]

Go to Hackathon

Hackathon is good and a cheap way how to train your lean thinking! Teams contributing at hackathon has to be extremely agile or they will fail. I believe that joining a hackathon is more beneficial than another Scrum training/seminar by industry leader. Your dev team transformation can start on their first hackathon. Goal at the hackathon is to prove concept and vision. It is not about business plans or robust architecture. Teams have to find what is your problem and make solution. Defining real problem and solution by team can sometimes take half of your time and you end up just with a few hours for execution. Imagine building a viable product in much less than 48 hours! Hackathon teams are able to do it! They have limited time therefore they became highly effective. Do you remember Parkinson's law and Pareto Principle?

Story

At the one Hackathon, we have built a service which shows municipal announcements related to location on a map. We had team of six people. One was "business" oriented, one graphic designer and 4 developers. We have built cloud web app (SPA AngularJS) which communicates with back end through API. Backend contains four microservices spread across three servers and using NoSql database. We have built a modern application on top of the hottest scalable technology. That was amazing! No bullshit, just two days of hard work! A product which works and people can use. The App was not perfect, had some bugs and some parts were incomplete, but we tested that people liked the idea. We got feedback from mentors and we were offered financial support to finish the app! That is a hackathon baby!

The good thing about hackathons is that there is no time to think about anything else except the fastest solution. Quite the opposite in an average dev team. I found ego-driven tendency of developers to show off their "great" skills at every time possible. This is the reason why we sometimes have bulky and difficult code for a simple problem. Every lead developer knows how to get the time he needs to build a perfect solution, they have enough power to bend project estimation to fit their vision. Many developers simply can not write simple code instead of an over-engineered perfect solution. Their pride and drive force them to build a spaceship instead of paraglide. First trigger for change can be joining a hackathon and see how great of an app can be created just in a few hours. Accept agile and let your ego go.
Clients do not care if you are agile or not, what technology you use, where you host your app, how many meetings you have or how much software tests you write**.** Clients are selfish. They want just product which solves their problem!

Scrum is not agile

Do not tell me that Scrum is agile. It is not! Scrum is just a framework, guide. Set of rules and recommendations. Scrum is often implemented by companies wanting to do Agile. It is the easiest way to tick that box and make managers and shareholders happy. I have seen many software teams claiming they do agile but later failing and moving to some sort of waterfall wrapped in agile sauce and accepting struggle associated to waterfall. Some people say they can not do Scrum because it does not fit to their business, customers, CEO, developers, company

culture, country, time zone, religion and mainly because it affects lunchtime on Friday! I am just joking ... or maybe not that much. Excuses, excuses, excuses. Some companies try Scrum and they end up with the simplest form of Kanban. They just put already existing tasks on kanban board and they claim to do agile development. Do you know why Scrum is the most popular agile framework? Because you can tweak it to the point where you follow Scrum practices and, at the same time, your high level development progress looks exactly like waterfall. People call it Water-Scrum-Fall
[https://www.scrumalliance.org/community/articles/2015/june/water-scrum-fal] It is usually first step toward agility, but often also the last one.

Story
I was in a situation where one important client came into office and manager (let's call him product owner) went directly to Scrum board and started to talk about our great development process. That single board with post-it sticky notes was the best marketing tool to get client excited about dev department. It is cheap and it looks like we know what we are doing. These magical colorful notes are the most impactful thing team get from Scrum. Somebody even may think that their product will be developed in time and budget just because of that kanban board. That rarely happens. Not because of Scrum or kanban board! Product is made by people and if they do not have agile/lean in their blood, no methodology will help. Doing daily standups, sprints and play planning poker is much easier than seriously think about agile. It is a good starting point, but the hardest action you can do is to sit down and think about agile for an hour. Scrum is nice, but complex framework. If you are new in agile, starting with Scrum is not bad step. But you should turn your interest out from Scrum in favour of agile as soon as possible. You do not get agile and lean mindset just by practicing Scrum. Focus on agile core values.

Agile core values

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

[http://www.agilemanifesto.org/]

People

Your company is nothing more than a few legal documents, some hardware and people. Your product is the result of employee's work guided by company processes. Focus on your employees, support them, trust them and give them opportunity to show off. People in your company are always the most important element.

Decision makers

I am sure every product manager wants to build the best product on the market. Decision makers have often good intentions, but they might be driven by their ego. Sometimes, people do not realize they are not building product market wants, but product they want. Support people who favour the best solution instead of pushing their own one.

No ego, no problem

Many problems are caused by humans' egos. Find people who welcome change, people who want to explore and learn. To hire a product manager, who thinks he knows exactly what people want, is the best way to build a product that nobody will use. Admit that you do not know what customer wants and try to find their need through experiments. For example, to accept that Scrum in your organization is not working is much better than to pretend that everything is fine. Find honest people who want to learn, build and change themselves.

Find evangelist

Hiring good people can be cost-effective way to increase the quality of your product. If you have team where lead developer does not understand agile, cost of development can be ... I am sure you already know the answer. It's easy to get stuck in a waterfall we are used to. It is critical to have at least one agile evangelist in a team who will support and guide other team members. This role can be called Scrum master and his job is to supervise agile development in an organization.

Company Culture

Last but most important. Your culture is what, and how, you do things daily. Scrum process does not make your organization agile. Your culture should support and emphasize agility. Right change means improvement and improvement means success.

Story

Once, I was at job interview at Rakuten. Rakuten is one of the biggest E-commerce company in Japan. CEO Hiroshi Mikitani has written a few books related to success and e-commerce. As a potential employee, I was asked to read at least one of his books and write a summary. This sounded strange and my first reaction was "... Yeah crazy Japanese with their over-commitment to company." Then I looked at books I read in past years. Most of them were books about "Success" (whatever it means) written by successful people. I have realized how much is this CEO smart and how strongly committed to unite company culture. Every new person in this corporation knows in detail about CEO and his values. They, in theory, do not need any paper describing company culture and values as any other big corporation. They have the book.

Embrace change - Kaizen

Kaizen is a method to introduce small and constant measured changes into an organization. You can easily introduce around 55 improvements in a year by making one small step every week!

The best solution

Perhaps you have guessed that I am strong believer in the lean way. I have tried to sell you lean and agile as the best solution for every software related issue. Unfortunately, lean is not cure for all problems. But it can keep your product above water so it does not sink. My motivation is simple -> no failed projects anymore. I have seen many failed teams, project, startups, companies. I feel like people are born with "waterfall" mindset. I see same trivial mistakes and illusional assumptions all the time. I understand you have your own interests. And I also know,

that releasing MVP, puts you into vulnerable position. But I believe, there is not any other simple way. Ask yourself, when were you last satisfied with your product development and release?

Epilogue

Congratulations! This is the end of the book but the beginning of your journey. I have nothing more to tell you and you have nothing to wait for. Nobody will push you forward or pull you closer to a success. You have to do it on your own. Be bold and move forward.

I started this book with quote “Most software projects fail!”, so I have to admit, that in my short career, I would consider most of the projects I was involved in as failure. However a few projects which were very close to, what I would call success, involved close communication with clients and short iterative development cycle. Delayed and failed project were ones with hundreds of tasks in TODO and release date in far future. So take my advice with some pinch of salt because I have to admit, I wasn’t fully satisfied with the outcome of any projects I was involved in.

This book was shortly written to give you an intense resource. It is not enough to read this book just once. You would most likely forget most of the knowledge you received and nothing would change. Re-read every chapter several times to deeply understand the meaning behind sentences.

It does not matter how old you are, what degree or experience you have. Only what matters is result. Do not get stuck into the loop called Paralysis by analysis. Pick one thing from this guide you like the most and implement it today, not tomorrow. Make action and observe reaction. Following agile core values is the noble path. Do not wait, your competitors do not. We are alive because of evolution. Start your evolution today and make every day one small step.

Thank you for reading. I believe in your success!