The Lean Startup - A focus on Practices

by Jerome Kehrli

Written in January 2017

A few years ago, I worked intensively on a pet project : AirXCell.
What was at first some framework and tool I had to write to work on my Master Thesis dedicated to Quantitative Research in finance, became after a few months somewhat my most essential focus in life.
Initially it was really intended to be only a tool providing me with a way to have a Graphical User Interface on top of all these smart calculations I was doing in R. After my master thesis, I surprised myself to continue to work on it, improving it a little here and a little there. I kept on doing that until the moment I figured I was dedicated several hours to it every day after my day job.
Pretty soon, I figured I was really holding an interesting software and I became convinced I could make something out of it and eventually, why not, start a company.

And of course I did it all wrong.

Instead of **finding out first if there was a need and a market for it**, and then *what should I really build to answer this need*, I spent hours every day and most of my week-ends developing it further towards what I was convinced was the minimum set of feature it should hold before I actually try to meet some potential customers to tell them about it.
So I did that for more than a year and a half until I came close to burn-out and send it all to hell.

Now the project hasn't evolve for three years. The thing is that I just don't want to hear about it anymore. I burnt myself and I am just disgusted about it. Honestly it is pretty likely that at the time of reading this article, the link above is not even reachable anymore.
When I think of the amount of time I ~~invested~~ wasted in it, and the fact that even now, three years after, I still just don't want to hear anything about this project anymore, I feel so ashamed. Ashamed that I didn't take a leap backwards, read a few books about startup creation, and maybe, who knows, discover *The Lean Startup* movement before.
Even now, I still never met any potential customer, any market representative. Even

worst: I'm still pretty convinced that there is a need and a market for such a tool. But I'll never know for sure.

Such stories, and even worst, stories of startups burning millions of dollars for nothing in the end, happen every day, still today.

Some years ago, Eric Ries, Steve Blank and others initiated ***The Lean Startup*** movement. *The Lean Startup* is a movement, an inspiration, a set of principles and practices that any entrepreneur initiating a startup would be well advised to follow. Projecting myself into it, I think that if I had read Ries' book before, or even better Blank's book, I would maybe own my own company today, around AirXCell or another product, instead of being disgusted and honestly not considering it for the near future.
In addition to giving a pretty important set of principles when it comes to creating and running a startup, *The Lean Startup* also implies an extended set of Engineering practices, especially software engineering practices.

This article focuses on presenting and detailing these **Software Engineering Practices from the Lean Startup Movement** since, in the end, I believe they can benefit from any kind company, from initiating startup to well established companies with Software Development Activities.
By Software Engineering practices, I mean software development practices of course but not only. Engineering is also about analyzing the features to be implemented, understanding the customer need and building a successful product, not just writing code.

Table of Contents

1. The Lean Startup

The Lean Startup is today a movement, initiated and supported by some key people that I'll present below.
But it's also a framework, an inspiration, an approach, a methodology with a set of fundamental principles and practices for helping entrepreneurs increase their odds of building a successful startup.
Lean Startup cannot be thought as a set of tactics or steps. Don't expect any checklist (well, at least not only checklists) or any recipe to be applied blindly.

The approach is built around two main objectives:

1. Teaching entrepreneurs how to drive a startup through the process of steering (*Build-Measure-Learn* feedback loop).
2. Enabling entrepreneurs to scale and grow the business with maximum acceleration

Lean Startup Practices

The Lean Startup methodology can be divided in two sets of practices:

1. The **steering practices** : designed to minimize the total time through the Build-Measure-Learn feedback loop and
2. The **acceleration practices** : which allow Lean Startups to grow without sacrificing the startup's speed and agility

This is developed further in 2. The four steps to the Epiphany.

1.1 Origins

The Lean Movement

Lean thinking is a **business methodology** that aims to provide a new way to think about how to organize human activities to deliver more benefits to society and value to individuals while **eliminating waste**.
Lean thinking is a **new way of thinking any activity** and seeing the waste inadvertently generated by the way the process is organized

The aim of lean thinking is to create a **lean enterprise**, one that **sustains growth** by aligning customer satisfaction with employee satisfaction, and that **offers innovative products** or services profitably while **minimizing unnecessary over-costs** to customers, suppliers and the environment.

The Lean Movement finds its roots in Toyotism and values **performance** and **continuous improvement**. The Lean Movement really rose in the early 90's and the lean tradition has adopted a number of practices from Toyota's own learning curve.
Some worth to mention:

- **Kaizen** (Continuous Improvement) : is a strategy where employees at all levels of a company work together pro-actively to achieve regular, incremental improvements to the manufacturing process. The point of Kaizen is that improvement is a normal part of the job, not something to be done "when there is time left after having done everything else", that should involve the company as a whole, from the CEO to the assembly line workers.
- **Kanban** (Visual Billboard) : is a scheduling system and visual management tool used in *Lean Manufacturing* to signal steps in their manufacturing process. The system's highly visual nature allows teams to communicate more easily on what work needed to be done and when. It also standardizes cues and refines processes, which helps to reduce waste and maximize value.

Plus strong emphasizes on *Autonomation*, *Visualization*, etc.

The Lean Startup

The author, I should say *initial author*, of the Lean Startup methodology, Eric Ries, explains in his book "*The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*", that traditional management practices and ideas are not adequate to tackle the entrepreneurial challenges of startups.

By exploring and studying new and existing approaches, Ries found that adapting *Lean thinking* to the context of entrepreneurship would allow to discern between *value-creating activities* versus *waste*.

Thus, Ries, decided to apply lean thinking to the process of innovation. After its initial development and some refinement, as he states, **the Lean Startup** represents a new approach to creating continuous innovation that builds on many previous management and product development ideas, including lean manufacturing, design thinking, customer development, and agile development.

1.2 The movement

I would highly recommend this enlightening article - The History Of Lean Startup - that does a pretty great job explaining how and why the following guys got together and initiated the *Lean Startup Movement* (aside from a few things I do not agree with).

(2013)
The Four Steps to the Epiphany
Successful Strategies for Products that Win
Steven Gary Blank
Steve Blank
(2011)
Business Model Generation
Alex Osterwalder
THE LEAN STARTUP
ERIC RIES
Eric Ries
(2011)
RUNNING LEAN
ITERATE FROM PLAN A TO A PLAN THAT WORKS
ASH MAURYA
Ash Maurya
(2012)

Blank, Ries, Osterwalder and Maurya are the founders or initiators of the *Lean Startup Movement*. Eric Ries is considered as the leader of the movement, while Steve Blank considers himself as its godfather.
Osterwalder and Maurya's work on business models is considered to fill a gap in Ries and Blank's work on processes, principles and practices. In Steve Blank's "The four Steps the the Epiphany", the business model section is a vague single page. Furtherly, Maurya's "*Running Lean*" magnificently completes Blank's work on *Customer Development*. We'll get to that.

1.3 Principles

In my opinion, the most fundamental aspect of Lean Startup is the *Build-Measure-Learn* loop, or, in the context of the *Customer Development Process*, the *Customer Discovery - Customer Validation - Re-adapt the product* loop.
The idea is to be able to loop in laboratory mode, mostly with prototypes and interviews, in an iterative research process, with as little costs as possible, about the product to be developed. A startup should spend as little investment as possible in

terms of product development as long as it has no certainty in regards to the customer needs, the right product to be developed, the potential market, etc.
This is really key, before hiring employees and starting to develop a product, the entrepreneur should have certainty about the product to be developed and its market.
Premature scaling is the immediate cause of the Death Spiral.

Before digging any further into this, below are the essential principles that characterize *The Lean Startup* approach, as reported by Eric Ries' book.

Entrepreneurs are everywhere

You don't have to work in a garage to be in a startup. The concept of entrepreneurship includes anyone who works within Eric Ries' definition of a startup, which I like very much BTW.
His definition is as follows :

A startup is a human institution designed to create new products and services under conditions of extreme uncertainty.

That means entrepreneurs are everywhere and the Lean Startup approach can work in any size company, even a very large enterprise, in any sector or industry.

Entrepreneurship is management

A startup is an institution, not just a product, and so it requires a new kind of management specifically geared to its context of extreme uncertainty.
In fact, Ries believes "*entrepreneur*" should be considered a job title in all modern companies that depend on innovation for their future growth

Validated learnings

Startups exist not just to make stuff, make money, or even serve customers. They exist to learn how to build a sustainable business. This learning can be validated scientifically by running frequent experiments that allow entrepreneurs to test each element of their vision.

Innovation accounting

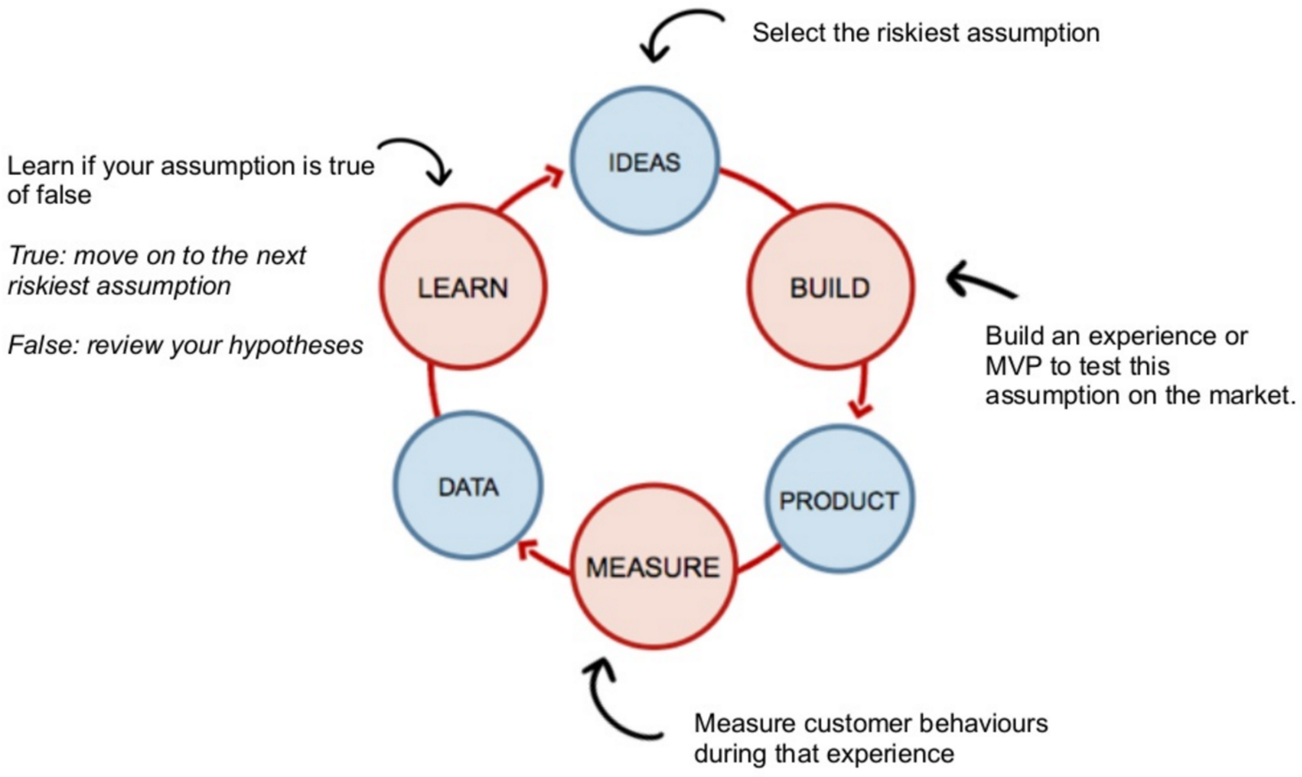
To improve entrepreneurial outcomes and hold innovators accountable, we need to focus on the boring stuff: how to measure progress, how to set up milestones, and how to prioritize work. This requires a new kind of accounting designed for startups- and the people who hold them accountable.

Build-Measure-Learn

The fundamental activity of a startup is to turn ideas into products, measure how customers respond, and then learn whether to **pivot or persevere**. All successful startup processes should be geared to accelerate that **feedback loop**.

1.4 The Feedback Loop

The feedback loop is represented as below.
The five-part version of the *Build-Measure-Learn* schema helps us see that the real intent of building is to test "*ideas*" - not just to build blindly without an objective.
The need for "*data*" indicates that after we measure our experiments we'll use the data to further refine our learning. And the new learning will influence our next ideas. So we can see that the goal of Build-Measure-Learn isn't just to build things, the goal is to build things to validate or invalidate the initial idea.


Select the riskiest assumption
IDEAS
Learn if your assumption is true of false
True: move on to the next riskiest assumption
False: review your hypotheses
LEARN
BUILD
Build an experience or MVP to test this assumption on the market.
DATA
PRODUCT
MEASURE
Measure customer behaviours during that experience

Again, the goal of *Build-Measure-Learn* is not to build a final product to ship or even to build a prototype of a product, but to **maximize learning** through incremental and iterative engineering.
In this case, learning can be about product features, customer needs, distribution channels, the right pricing strategy, etc.
The "*build*" step refers to building an 3.2.1 MVP (Minimal Viable Product).
It's critical here to understand that an MVP does not mean *the product with fewer features*. Instead, an MVP should be seen as the simplest thing that you can show to customers to get the most learning at that point in time. Early on in a startup, an MVP could well simply be a set of Powerpoint slides with some fancy animations, or

whatever is sufficient to demonstrate a set of features to customers and get feedback from it. Each time one builds an MVP one should also define precisely what one wants to test/measure.
Later, as more is learned, the MVP goes from low-fidelity to higher fidelity, but the goal continues to be to maximize learning not to build a beta/fully featured prototype of a product or a feature.

In the end, the *Build-Measure-Learn* framework lets startups be fast, agile and efficient.

1.5 Business Model Canvas and Lean Canvas

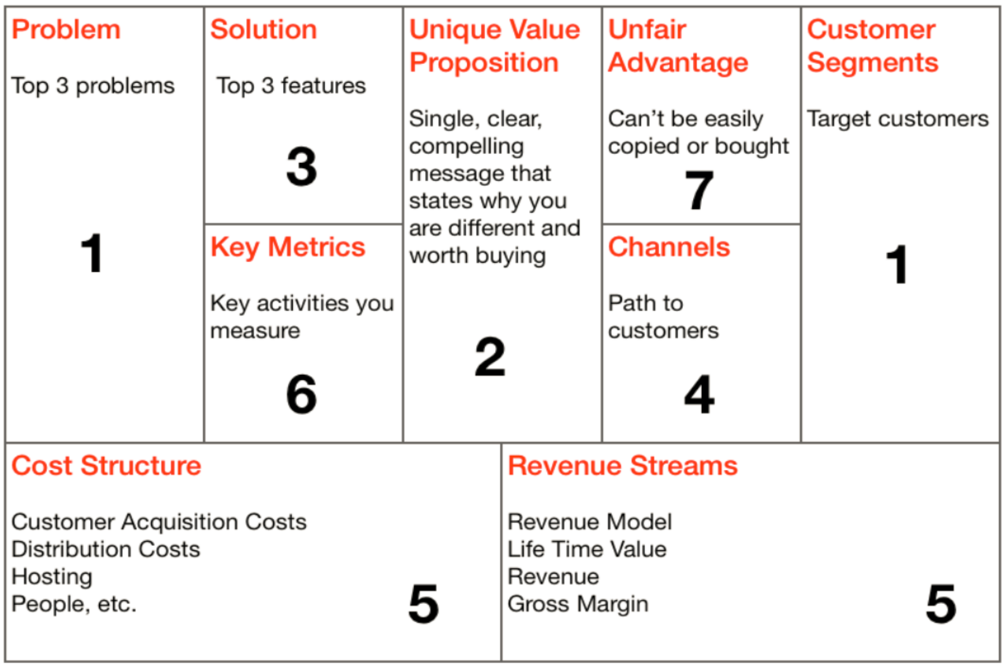
Evolution on Business Models and the relative processes were surprisingly missing or poorly addressed from Ries' and Blank's initial work.
Fortunately, Osterwalder and Maurya caught up and filled the gap.

Business Model Canvas

The Business Model Canvas is a strategic management template invented by Alexander Osterwalder and Yves Pigneur for developing new business models or documenting existing ones.
It is a visual chart with elements describing a company's value proposition, infrastructure, customers, and finances. It assists companies in aligning their activities by illustrating potential trade-offs.

Lean Canvas

The Lean Canvas is a version of the Business Model Canvas adapted by Ash Maurya specifically for startups. The Lean Canvas focuses on addressing broad customer problems and solutions and delivering them to customer segments through a unique value proposition.



So how should one use the Lean Canvas?

1. **Customer Segment and Problem**
   Both Customer Segment and Problem sections should be filled in together. Fill in the list of potential *customers* and *users* of your product, distinguish customers (willing to pay) clearly from users, then refine each and every identified customer segment. Be careful not no try to focus on a too broad segment at first, think of Facebook whose first segment was only Harvard students.
   Fill in carefully the problem encountered by your identified customers.
2. **UVP - Unique Value Proposition**
   The UVP is the unique characteristic of your product or your service making it different from what is already available on the market an that makes it worth the consideration of your customers. Focus on the main problem you are solving and what makes your solution different.
3. **Solution**
   Filling this is initially is tricky, since knowing about the solution for real requires trial and error, build-measure-learn loop, etc. In an initial stage one shouldn't try to be to precise here and keep things pretty open.
4. **Channels**
   This consists in answering: how should you get in touch with your users and customers ? How do you get them to know about your product ? Indicate clearly your communication channels.

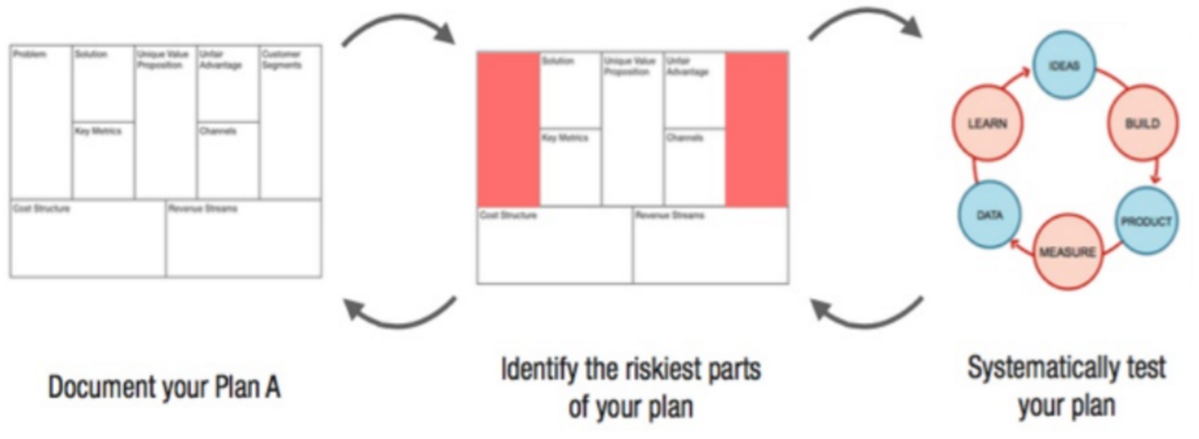
5. **Revenue Stream and Cost Structure**
   Both these sections should also be filled in together.
   At first, at the time of the initial stage of the startup, this should really be focused on the costs and revenues related to launching the MVP (how to interview 50 customers ? Whats the initial burn rate ? etc.)
   Later this should evolve towards an initial startup structure and focus on identifying the *break-even* point by answering the question : how many customers are required to cover my costs ?

6. **Key Metrics**
   Ash Maurya refers to Dave McClure Pirate Metrics to identify the relevant KPIs to be followed :
   Acquisition - How do user find you ?
   Activation - Do user have a great first experience ?
   Retention - Do users come back ?
   Revenue - How do you make money ?
   Referral - Do users tell others ?

7. **Unfair Advantage**
   This consists in indicating the adoption barriers as well as the competitive advantages of your solution. An *unfair advantage* is defined as something that cannot be copied easily neither bought.

Lean Startup : test your plan !

Using the new "Build - Measure - Learn" diagram, the question then becomes, "What hypotheses should I test?". This is precisely the purpose of the initial Lean Canvas,



And it brings us to another definition of a startup:

A startup is a temporary organization designed to *search* for a repeatable and scalable business model.

And once these hypotheses fill the Lean Canvas (Or Business Model Canvas), the key approach is to **run experiments**. This leads us to the next section.

1.6 Customer Development

The Customer Development process is a simple methodology for taking new venture hypotheses and getting out of the building to test them. *Customer discovery* (see below) captures the founders' vision and turns it into a series of business model hypotheses. Then it develops a series of experiments to test customer reactions to those hypotheses and turn them into facts. The experiments can be a series of questions you ask customers. Though, most often an MVP to help potential customers understand your solution accompanies the questions.

Startups are building an MVP to learn the most they can, not to get a prototype!

The goal of designing these experiments and minimal viable products is not to get data. The data is not the endpoint. Anyone can collect data. **The goal is to get insight**. The entire point of getting out of the building is to inform the founder's vision.
The insight may come from analyzing customer answers, but it also may come from interpreting the data in a new way or completely ignoring it when realizing that the idea is related to a completely new and disruptive market that even doesn't exist yet.

Customer Development instead of Product Development

More startup fail from a *lack of customers* rather than from a failure of Product Development.

The Customer Development model delineates all the customer-related activities in the early stage of a company into their own processes and groups them into four easy-to-understand steps: *Customer Discovery*, *Customer Validation*, *Customer Creation*, and *Company Building*.
These steps mesh seamlessly and support a startup's ongoing product development activities. Each step results in specific deliverables and involves specific practices.

As its name should communicate, the Customer Development model focuses on developing customers for the product or service your startup is building. Customer Development is really about finding a market for your product. It is built upon the idea that the founder has an idea but he doesn't know if the clients he imagines will buy. He needs to check this point and it is better if he does it soon.

2. The four steps to the Epiphany

Shortly put, Steve Blank proposes that companies need a **Customer Development process** that complements, or even in large portions replaces, their *Product Development Process*. The *Customer Development process* goes directly to the

theory of Product/Market Fit.
In "*The four steps to the Epiphany*", Steve Blank provides a roadmap for how to get to Product/Market Fit.

2.1 Overview

The Path to Disaster: The Product Development Model

The traditional product development model has four stages:

1. concept/seed,
2. product development,
3. beta test,
4. and launch.

That product development model, when applied to startups, suffers from a lot of flaws. They basically boil down to:

- Customers were nowhere in that flow chart
- The flow chart was strictly linear
- Emphasis on execution over learning
- Lack of meaningful milestones for sales/marketing
- Treating all startups alike

What's the alternative? Before we get to that, one final topic is the *technology life cycle adoption curve*, i.e. adoption happens in phases of early adopters (tech enthusiasts, visionaries), mainstream (pragmatists, conservatives), and skeptics. Between each category is a *chasm*, the largest is between the early adopters and the mainstream.
Crossing the chasm is a success problem. But you're not there yet, "customer development" lives in the realm of the early adopter.

The Path to Epiphany: The Customer Development Model

Most startups lack a process for discovering their markets, locating their first customers, validating their assumptions, and growing their business.
The **Customer Development Model** creates the process for these goals.

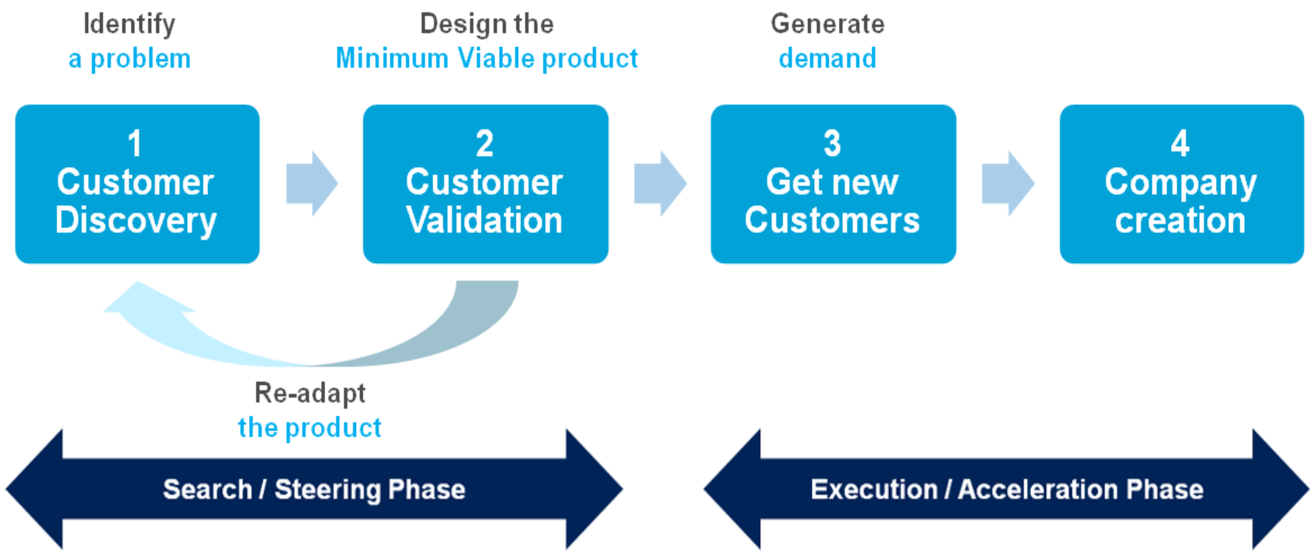
2.2 A 4 steps process

The four stages the *Customer Development Model* are: customer discovery, customer validation, customer creation, and company creation.

1. **Customer discovery**: understanding customer problems and needs

2. **Customer validation**: developing a sales model that can be replicated
3. **Customer creation / Get new Customers**: creating and driving end user demand
4. **Customer building / Company Creation**: transitioning from learning to executing

We can represent them as follows:



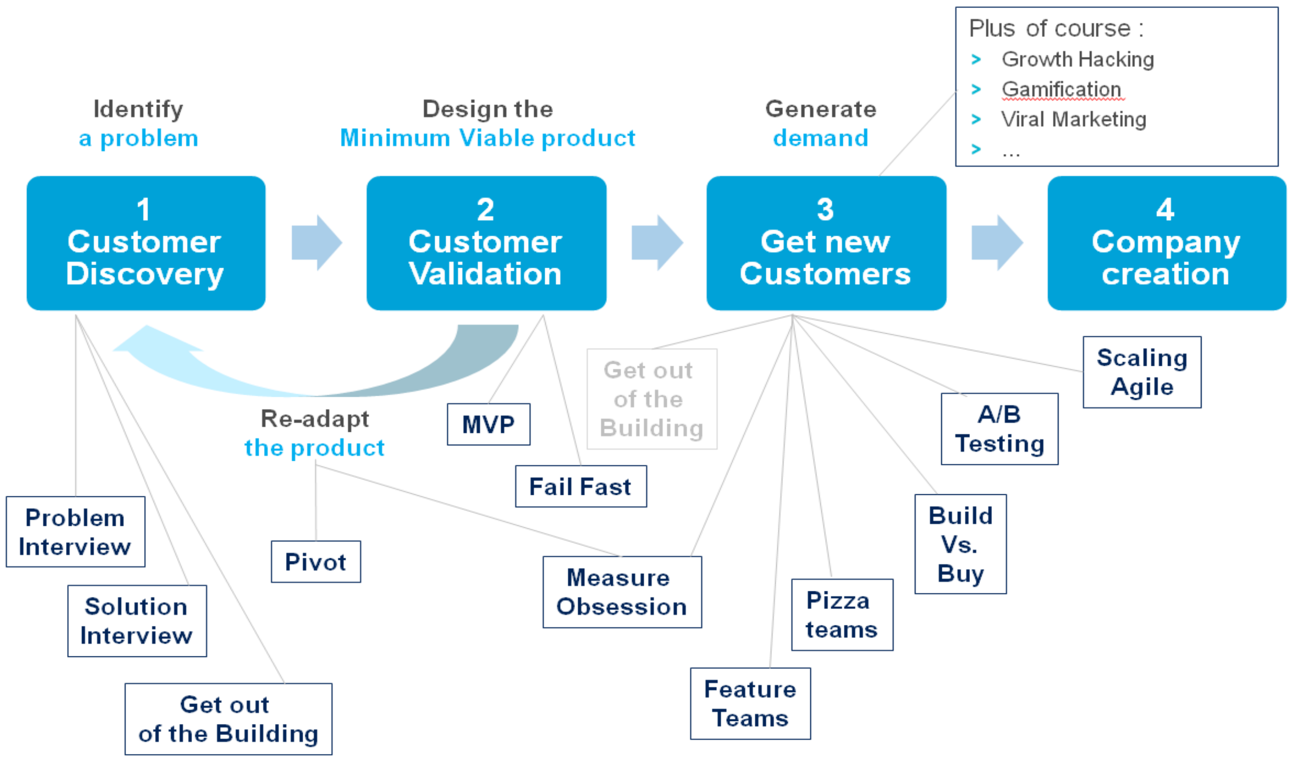
I won't go any further in this article in describing these steps, their purpose and reasons.
To be honest Blank's book is pretty heavy and not very accessible. Happily Blank's did a lot of presentations around his book that one can find on youtube or elsewhere. In addition, there are a lot of excellent summaries and text explanations available online on Blank's book and I let the reader refer to this material should he want more information.

Instead, I want to focus in this article on the **Software Engineering Practices** inferred from The Lean Startup approach, since, again, I believe they are very important for any kind of corporation with an important Software Development activity.
And yet again, Software Engineering practices go beyond solely Software Development practices, but cover every activity in the company aimed at identifying and developing the product.

3. Lean startup practices

So I want to present the most essentials principles and practices introduced and discussed by *the Lean Startup* approach.
These principles and practices are presented on the following schema attached to

the stages of the *Customer Development* process where I think they make more sense:



Important notes

- I attached the practices to the step where I think they make more sense, where I think they bring the most added value or should be introduced. But bear in mind that such a *categorization* is highly subjective and questionable. If you yourself believe some practices should be attached to another step, well just leave a comment and move on.
- Also, there are other practices of course. I mention here and will be discussing below the ones that seem the most appealing to me, myself and I. Again my selection is highly subjective and personal. If you think I am missing something important, just leave a comment and move on.

The rest of this paper intends to describe all these engineering - mostly software engineering - practices since, again, at the end of the day, I strongly believe that they form the most essential legacy of the Lean Startup movement and that they can benefit any kind of company, not only startups.

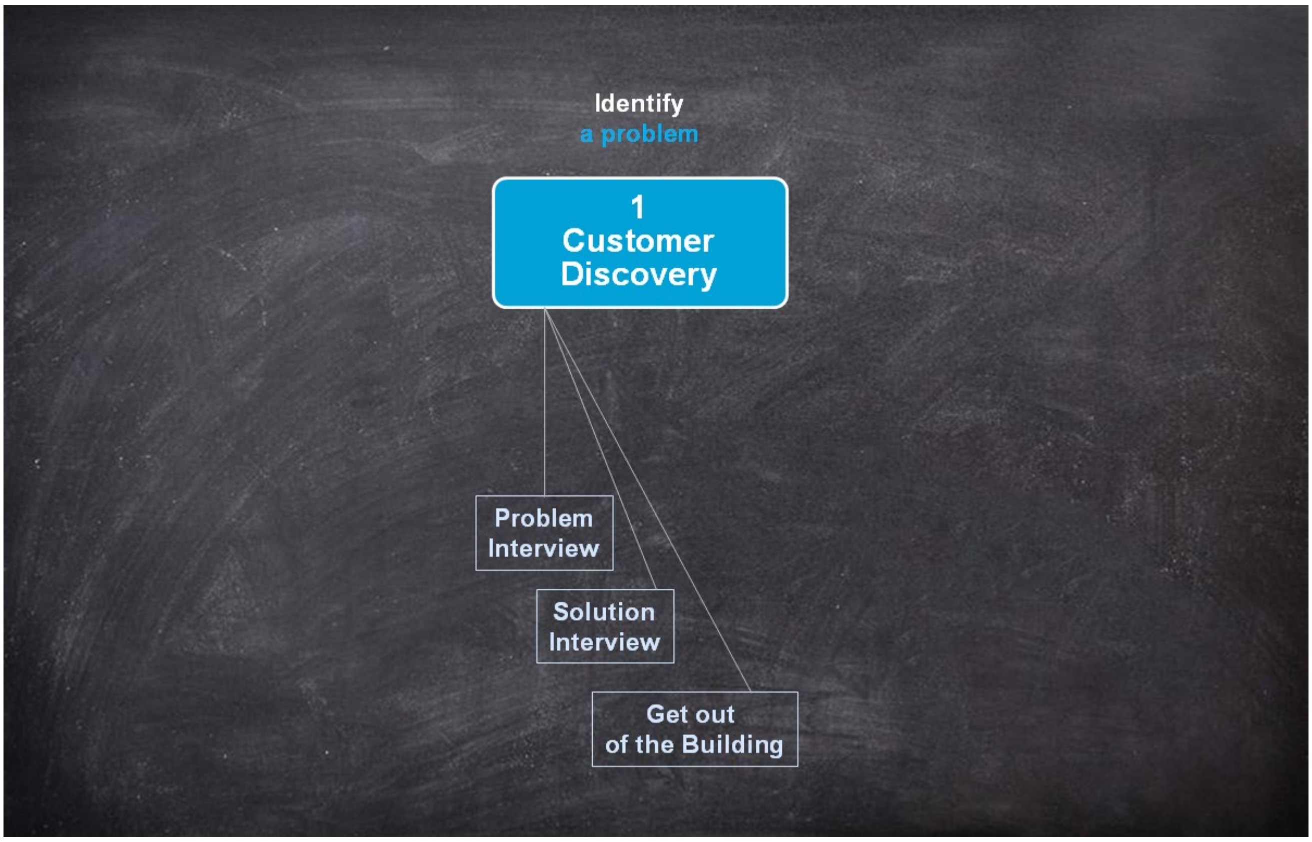
3.1 Customer Discovery

Customer Discovery, focuses on understanding customer problems and needs. Its really about searching for the *Product-Solution Fit*, turning the founders' initial hypotheses about their market and customers into facts.

The Problem-Solution Fit occurs when entrepreneurs identify relevant insights that can be addressed with a suggested solution. As Osterwalder describes it, this fit happens when there is evidence that customers care about certain problems that need to be solved or needs, and, there is a value proposition designed that addresses those needs.

In Customer Discovery the startup aims at understanding customer problems and needs and, also, to ideate potential solutions that could be valuable based on the findings. Similarly, Osterwalder calls these problems and needs as *jobs, pains and gains*.

The three practices I want to emphasize at this stage are as follows:



3.1.1 Get out of the building

If you're not Getting out of the Building, you're not doing Customer Development and Lean Startup.
There are no facts inside the building, only opinions.

If you aren't actually talking to your customers, you aren't doing Customer Development. And talking here is really speaking, with your mouth. Preferably in-person, but if not, a video call would work as well, messaging or emailing doesn't.

As Steve Blank said "*One good customer development interview is better for learning about your customers / product / problem / solution / market than five surveys with 10'000 statistically significant responses.*"

The problem here is that tech people, especially software engineers, try to avoid going out of the building as much as possible. But this is so important. Engineers need to fight against their nature and get out of the building and talk to customers as much as possible; find out who they are, how they work, what they need and what your startup needs to do, to build and then sell its solution.

In fact, so many engineers, just as myself, spent months of working on a prototype or even a complete solution, sometimes for several years, before actually meeting a first potential customer, and discovering the hard way that all this work has been for nothing.
As hard as it is, Engineers should not work one one single line of code, even not one single powerpoint presentation before having met at least a twenty potential customers or representatives and conducted formal 3.1.2 Problem interview.
After that, it's still not a question of writing lines of code, it's a question of investing a few hours - not more ! - in designing a demonstrable prototype for the next set of interviews, the 3.1.3 Solution interview. That prototype doesn't need to be actually working, it should only be demonstrable. A powerpoint presentation with clickable animations works perfectly!

Again, getting out of the building is not getting in the parking lot, it's really about getting in front of the customer.
At the end of the day, it's about *Customer Discovery*. And *Customer Discovery* is not sales, it's a lot of listening, a lot of understanding, not a lot of talking.

A difficulty that people always imagine is that young entrepreneurs with an idea believe that they don't know anybody, so how to figure out who to talk to ?

But at the time of Linkedin, facebook, twitter, it's hard to believe one cannot find a hundred of people to have a conversation with.

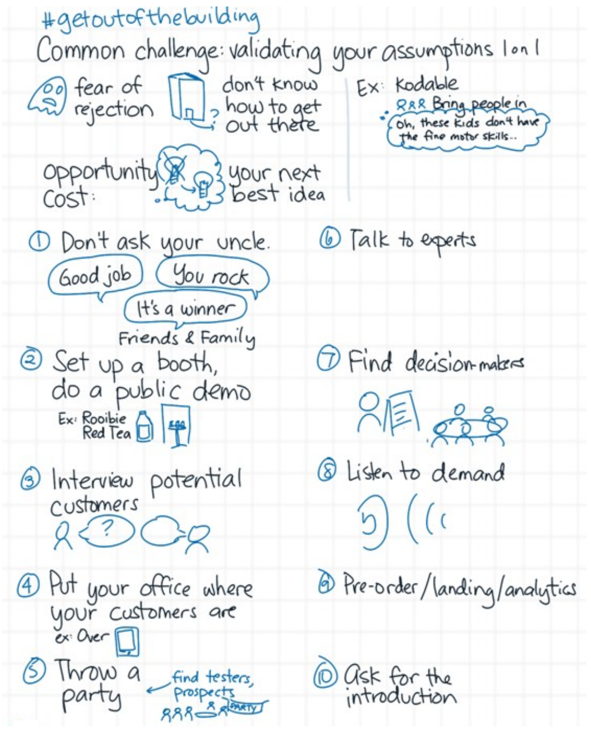
And when having a conversation with one of them, whatever else one's asking (3.1.2 Problem interview, 3.1.3 Solution interview), one should ask two very important final questions:

1. "*Who else should I be talking to ?*"
   And because you're a pushy entrepreneur, when they give you those names, you should ask "*Do you mind if I sit here while you email them introducing me ?*"

2. "*What should I have really asked you ?*"
   And sometimes that gets into another half hour related to what the customer is *really* worried about, what's really the customer's problem.

Customer Discovery becomes really easy once you realize you don't need to get the world's best first interview.
In fact its the sum of these data points over time, it's not one's just going to be doing one and one wants to call on the highest level of the organization.
In fact you actually never want to call on the highest level of the organization because you're not selling yet, you don't know enough.
What one actually wants is to understand enough about the customers, their problems and how they're solving it today, and whether one's solution is something they would want to consider.

A few hints in regards to how to get out of the building:



3.1.2 Problem interview

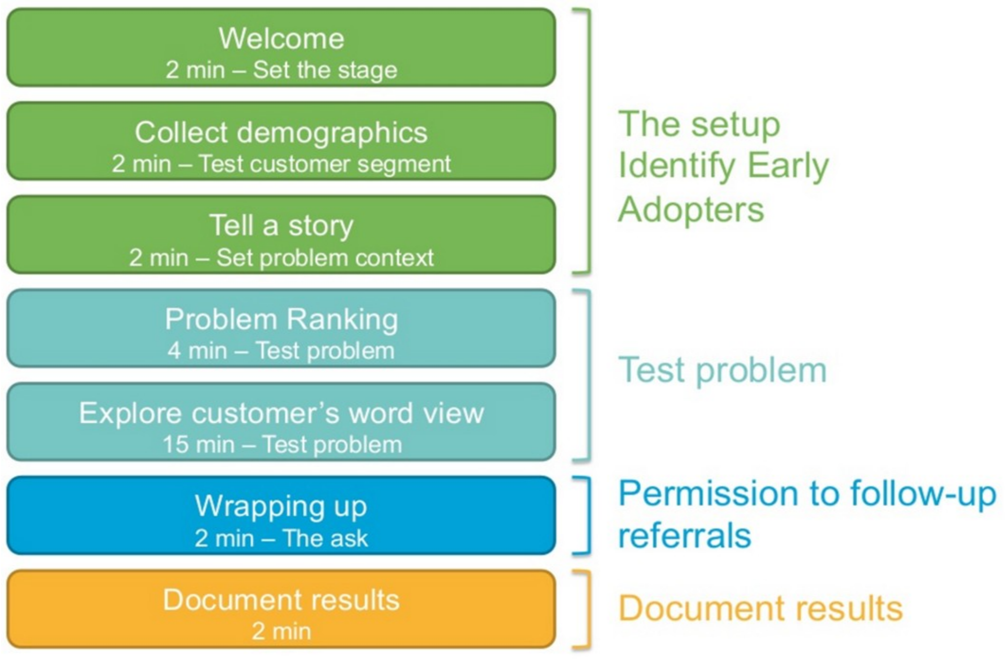
Problem Interview is Ash Maurya's term for the interview you conduct to validate whether or not you have a real problem that your target audience has.

In the Problem Interview, you want to find out 3 things:

1. **Problem** - What are you solving? - How do customers rank the top 3 problems?
2. **Existing Alternatives** - Who is your competition? - How do customers solve these problems today?
3. **Customer Segments** - Who has the pain? - Is this a viable customer segment?

Talking to people is hard, and talking to people in person is even harder. The best way to do this is building a script and sticking to it. Also don't tweak your script until you've done enough interviews so that your responses are consistent.
The main point is to collect the information that you will need to validate your problem, and to do it face-to-face, either in-person or by video call. It's actually important to see people and be able to study their body language as well.

The interview script - at least the initial you should follow until you have enough experience to build yours - is as follows:



If you have to remember just three rules for problem interviews here they are:

1. Do not talk about your business idea or product. You are here to understand a problem, not imagine or sell a solution yet.

2. Ask about past events and behaviours
3. No leading question, learn from the customer

After every interview, take a leap backwards, analyze the answers, make sure you understand everything correctly and synthesize the results.
After a few dozen of interviews, you should be a able to make yourself a clear understanding of the problem and initiate a few ideas regarding the solution to it.
Finding and validating your solution brings us to the next topic: the *Solution Interview*.

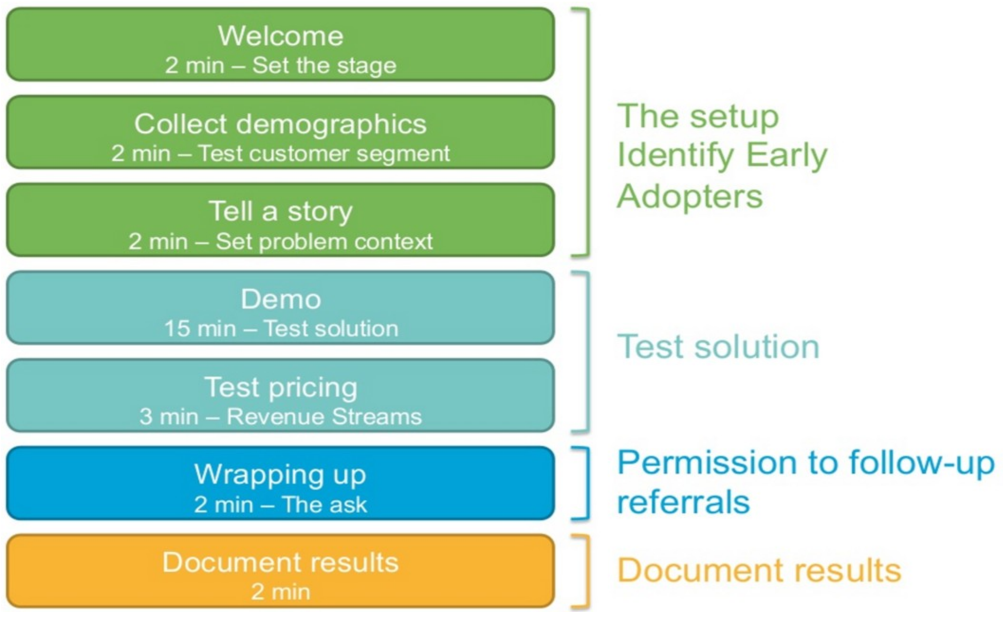
And what if a customer tells you that the issues you thought are important really aren't? Learn that you have gained important data.

3.1.3 Solution interview

In the Solution Interview, you want to find out three things:

1. **Early Adopters** - Who has this problem? - How do we identify an early adopter?
2. **Solution** - How will you solve the problems? - What features do you need to build?
3. **Pricing/Revenue** - What is the pricing model? - Will customers pay for it?

The key point here is to understand how to come up with a solution fitting the problem, step by step getting to the right track with your prototype and also understanding what could be a pricing model.



A *demo* is actually important. Many products are too hard to understand without some kind of demo. If a picture is worth a thousand words, a demonstration is probably worth a million.

Identifying early adopters is also key.
Think of something: if one of the guys you meet tells you that you definitely hold something, ask him if he would want to buy it. If he says he would definitely buy it when it's ready and available, ask him if he would commit to this. If he says he commits to this, ask him if he would be ready to pay half of it now and have it when its ready, thus becoming a partner or an investor.
If you find ten persons committing on already paying for the solution you draw, you may not even need to search for investors, you already have them. And that is the very best proof you can find that your solution is actually something.
And customers or partners are actually the best possible type of investors.

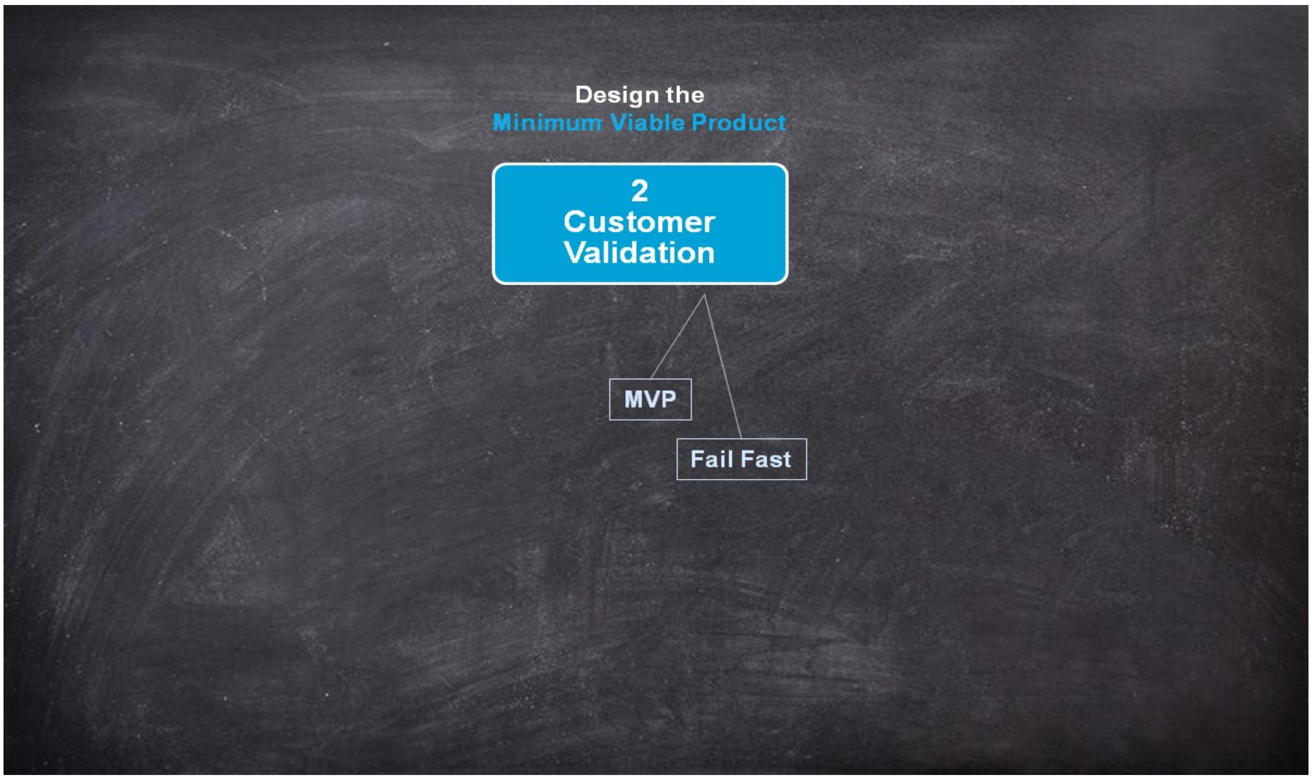
3.2 Customer Validation

The second step of the Customer Development model, *Customer Validation*, focuses on developing a sales model that can be replicated. The sales model is validated by running experiments to test if customers value how the startup's products and services are responding to the customer problems and needs identified during the previous step.
If customers show no interest, then the startup can 3.3.2 Pivot to search for a better business model.

Customer Validation needs to happen to validate if the customers really care about the products and services that could be valuable to them. This second step is hence really about *Product-Market Fit* which occurs when there is a sales model that works, when customers think the proposed solution is valuable to them. This should be proven by evidence that customers care about the products and services that conform the value proposition.

Blank believes that *product-market fit* needs to happen before moving from Customer Validation to Customer Creation (or the *Search Phase* to the *Execution Phase*).

The two practices I want to emphasize at this stage are as follows:



3.2.1 MVP

The **Minimum Viable Product** is an engineering product with just the set of features required to gather *validated learnings* about it - or some of its features - and its continuous development.
This notion of *Minimum Feature Set* is key in the MVP approach.

The key idea is that it makes really no sense developing a full and finalized product without actually knowing what will be the market reception and if all of it is actually worth the development costs.
Gathering insights and directions from an MVP avoids investing too much in a product based on wrong assumptions. Even further, The *Lean Startup* methodology seeks to avoid assumptions at all costs, see 1.4 The Feedback Loop and 3.3.1 Metrics Obsession.

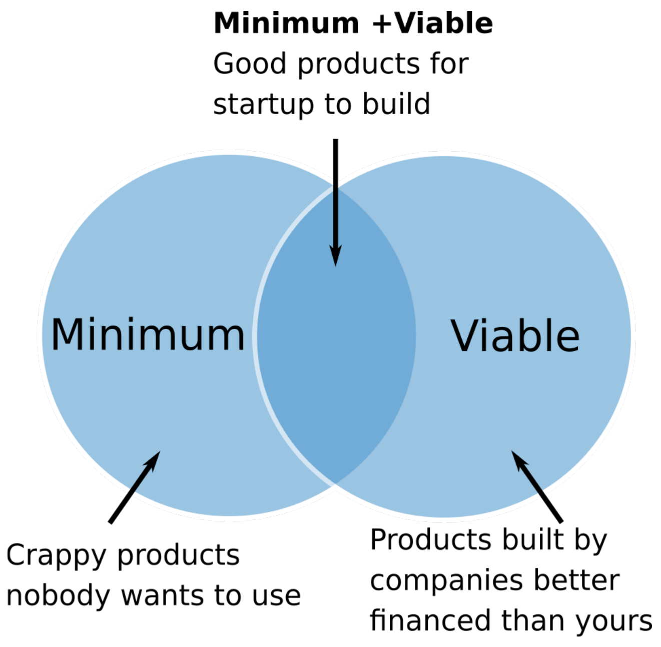
The *Minimum Viable Product* should have just that set of initial features strictly required to have a valid product, usable for its very initial intent, and nothing more. In addition these features should be as minimalist as possible but without compromising the overall *User Experience*. A car should move, a balloon should be round and bounce, etc.
when adopting an MVP approach, the MVP is typically put at disposal at first only to *early adopters*, these customers that may be somewhat forgiving for the "naked" aspect of the product and more importantly that would be willing to give feedback and help steer the product development further.

Eric Ries defines the MVP as:

"The minimum viable product is that version of a new product a team uses to collect the maximum amount of validated learning about customers with the least effort."

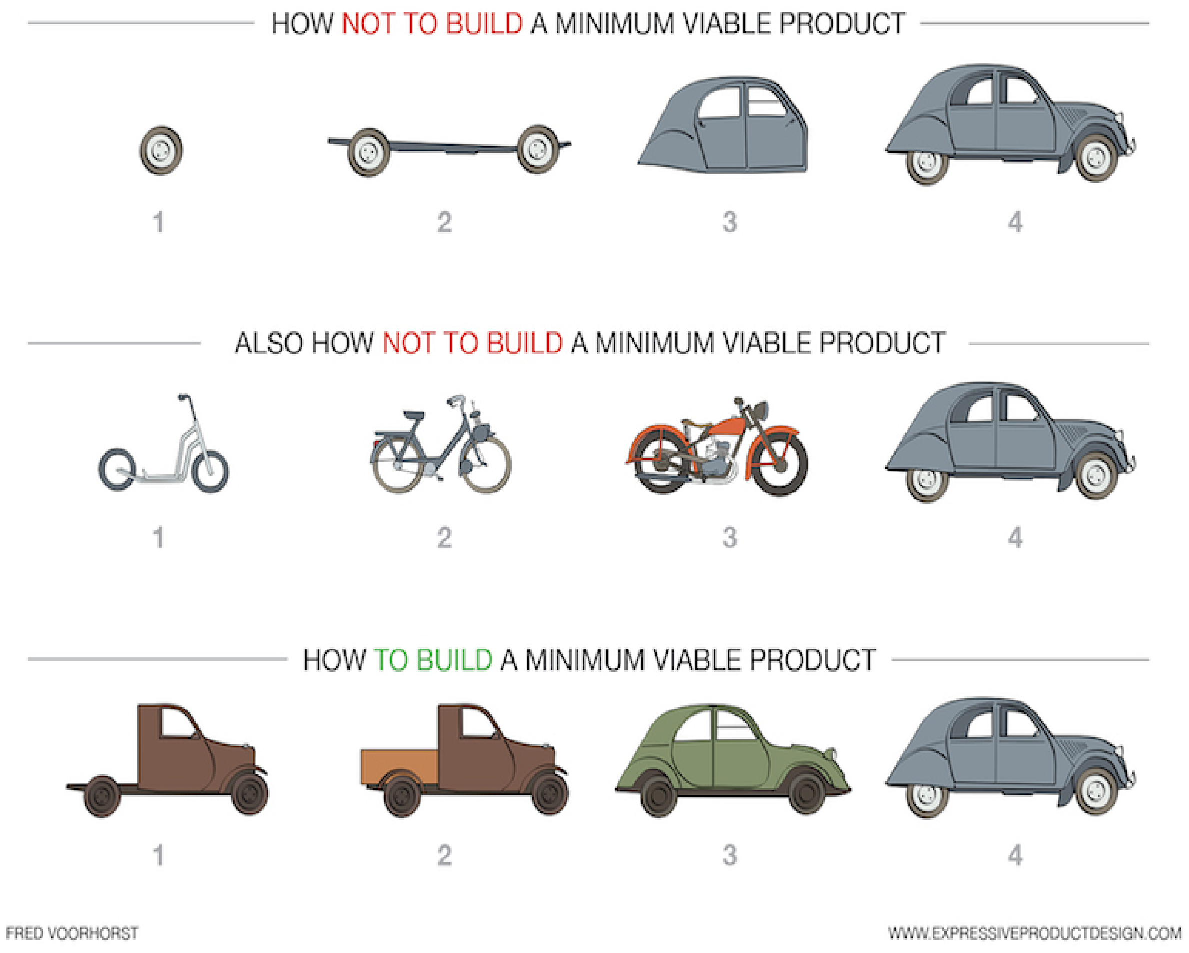
The definition's use of the words *maximum* and *minimum* means it is really not formulaic. In practice, it requires a lot of judgment and experience to figure out, for any given context, what MVP makes sense.

The following chart is pretty helpful in understanding why both terms *minimum* and *viable* are equally important and why designing an MVP is actually difficult:


Minimum +Viable
Good products for startup to build
Minimum
Viable
Crappy products nobody wants to use
Products built by companies better financed than yours

When applied to a new feature of any existing product instead of a brand new product, the MVP approach is in my opinion somewhat different. It consists of implementing the feature itself not completely; rather, a mock-up or even some animation simulating the new feature should be provided.
The mock-up or links should be properly instrumented so that all user reactions are recorded and measured in order to get insights on the actual demand of the feature and the best form it should take (3.3.1 Metrics Obsession),
This is called a **deploy first, code later** method.

Fred Voorhorst' work does a pretty good job in explaining what an MVP is:


HOW NOT TO BUILD A MINIMUM VIABLE PRODUCT
1
2
3
4
ALSO HOW NOT TO BUILD A MINIMUM VIABLE PRODUCT
1
2
3
4
HOW TO BUILD A MINIMUM VIABLE PRODUCT
1
2
3
4
FRED VOORHORST
WWW.EXPRESSIVEPRODUCTDESIGN.COM

(Fred Voorhorst - Expressive Product Design - http://www.expressiveproductdesign.com/minimal-viable-product-mvp/)

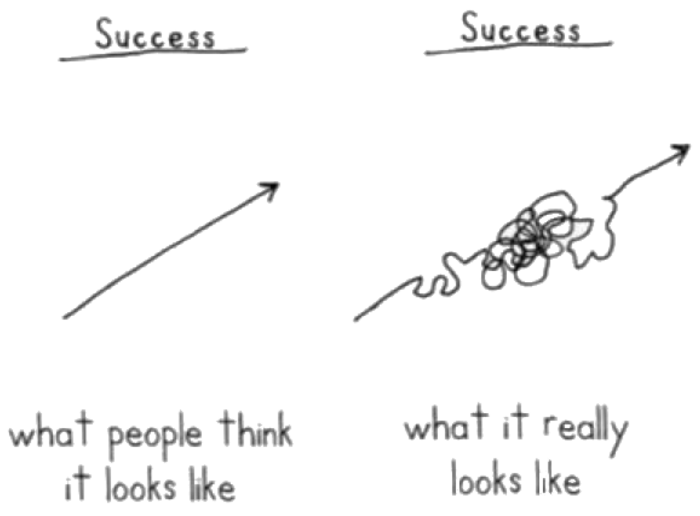
Developing an MVP is most definitely not the same as developing a sequence of elements which maybe, eventually combine into a product. A single wheel is not of much interest to a user wanting a personal transporter like a car, as illustrated by the first line.
Instead, developing an MVP is about developing the vision. This is not the same as developing a sequence of intermediate visions, especially not, if these are valuable products by themselves. As an example, a skateboard will likely neither interest someone in search for a car, as illustrated by the second line.

Developing an MVP means developing a sequence of prototypes through which you explore what is key for your product idea and what can be omitted.

3.2.2 Fail Fast

The key point of the "**fail fast**" principle is to quickly abandon ideas that aren't working. And the big difficulty of course is not giving up too soon on an idea that could potentially be working. should one find the right channel, the right approach. Fail fast means getting out of planning mode and into testing mode, eventually for every component, every single feature, every idea around your product or model of change. *Customer development* is the process that embodies this principle and helps you determine which hypotheses to start with and which are the most critical for your new idea.

It really is OK to fail if one knows the reason of the failure, and that is where most people go wrong. Once a site or a product fails then one needs to analyze why it bombed. It's only then that one can learn from it.
The key aspect here is really learning. And learning comes from experimenting, **trying things, measuring their success and adapting**.
An entrepreneur should really be a pathologist investigating a death and finding the cause of the failure. Understanding the cause of a failure can only work if the appropriate measures and metrics around the experiment are in place.



Now failing is OK as long as we learn from it and as long as we **fail as fast as possible**. Again, the whole *lean* idea is to avoid waste as much as possible and there's no greater waste than keeping investing on something that can ultimately not work. Failing as fast as possible, adapting the product, pivoting the startup towards its next approach as soon as possible is key.
But then again, the big difficulty is not to give up too soon on something that could possible work.

Fail fast,
Learn faster,
Succeed sooner !

So how do you know when to turn, when to drop an approach and adapt your solution ? How can you know it's not too soon?

Measure, measure, measure of course!

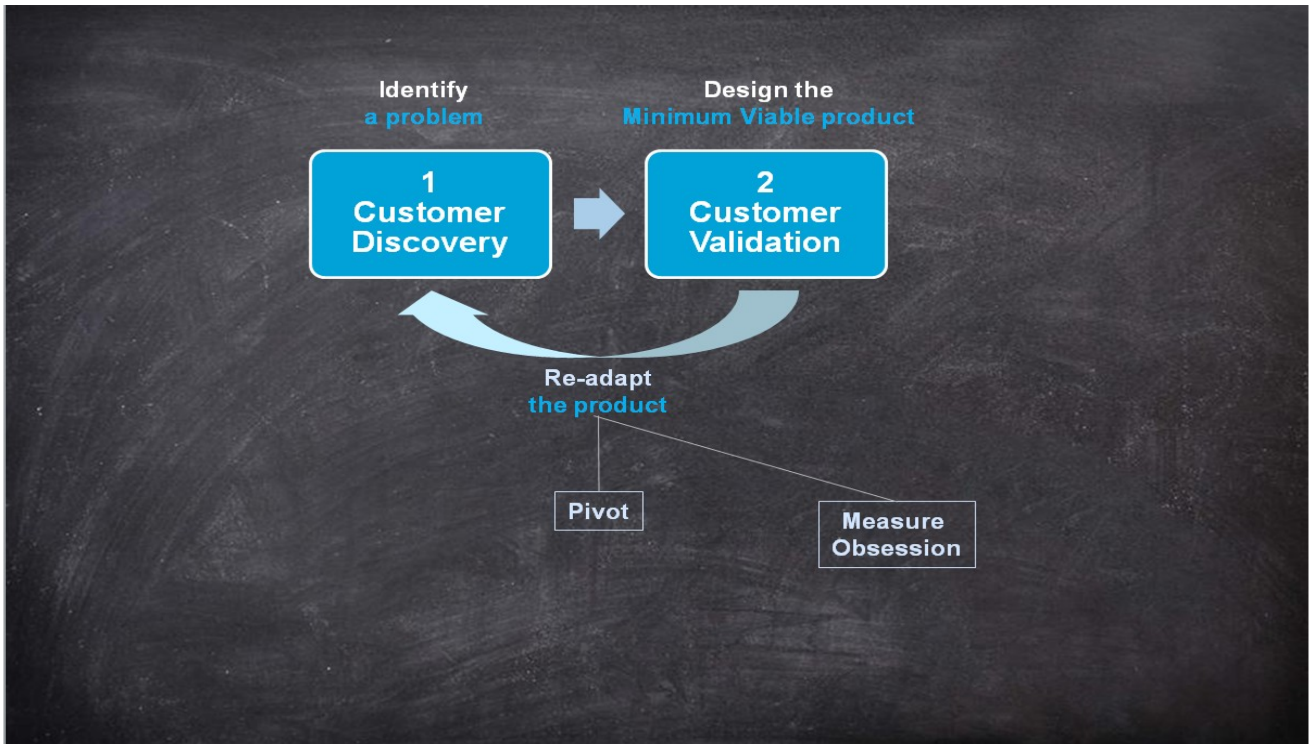
The testing of new concepts, failing, and building on failures are necessary when creating a great product.
The adage, "*If you can't measure it, you can't manage it*" is often used in management and is very important in *The Lean Startup* approach. By analyzing data, results can be measured, key lessons learned, and better initiatives employed.

3.3 Re-adapt the product

Customer development isn't predictable; you don't know what you're going to learn until you start. You'll need the ability to think on your feet and adapt as you uncover new information.
Adapting, in my opinion, is really re-adapting the product to the new situation, to the new knowledge you gained from the previous steps. And re-adapting the product, your solution, your approach is pivoting.

But I want to emphasize here that pivoting, or re-adapting the product, should only happen with the right data, the precise insights that give a clear new direction. Metrics and insight are essential.

The key practices here are as follows:



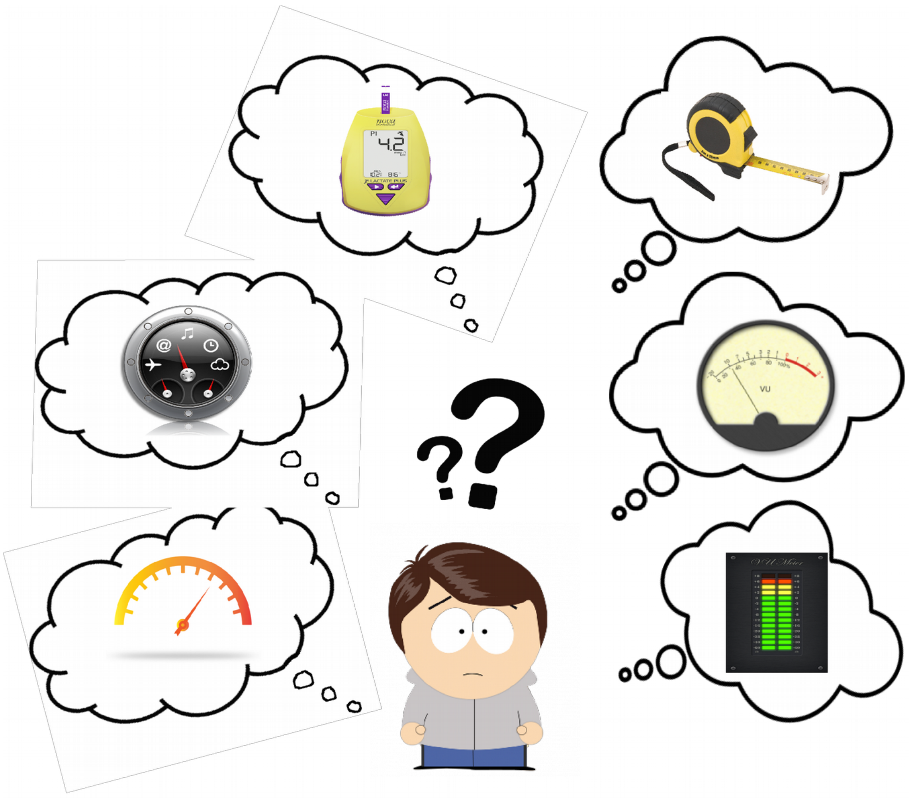
3.3.1 Metrics Obsession

In the *build-measure-learn* loop, there is measure ... *The Lean Startup* makes from measuring everything an actual obsession. And I believe that this is a damn' good thing.
Think of it: what if you have an idea regarding a new feature or an evolution of your product and you don't already have the metrics that can help you take a sound and enlightened decision? You'll need to introduce the new measure and wait until you get the data. Waiting is not good for startups.

This is why I like thinking of it as a **Metrics Obsession**. Measure everything, everything you can think of!

And repeat a hundred times:

I will never ever again think that
Instead I will *measure* that ...



Or as Edward Deming said :

"In god we trust, all others must bring data"

Imagine you work on a website. You should enhance your backend to measure, at least: amount of times a page has been displayed, count of users and different users displaying the pages, amount of times a link or button has been clicked, by who it

has been clicked, how much time after the containing page has been displayed, what is the user think time between 2 actions, what is the path of navigation from each and every user (actually build the graph and the counts along the branches), etc.
Measure everything! Don't hesitate to measure something you do not see any use for now. Sooner or later you will find a usage for that metrics, and that day, you better have it.

How to choose good metrics ?

Honestly there is no magic silver bullet and it can in fact be pretty difficult to pick up the right metric that would be most helpful to validate a certain hypothesis.
However, metrics should at all cost respect the three A's. Good metrics

- are **actionable**,
- can be **audited**
- are **accessible**

An **actionable metric** is one that ties specific and repeatable actions to observed results. The *actionable* property of picked up metrics is important since it prevents the entrepreneur from distorting the reality to his own vision. We speak of *Actionable vs. Vanity* Metrics.
Meaningless metrics such as "How many visitors ?", "How many followers ?" are vanity metrics and are useless.

Ultimately, your metrics should be useful to **measure progress against your own questions**.

3.3.2 Pivot

In the process of learning by iterations, a startup can discover through field returns with real customers that its product is either not adapted to the identified need, that it does not meet that need.
However, during this learning process, the startup may have identified another need (often related to the first product) or another way to answer the original need.
When the startup changes its product to meet either this new need or the former need in a different way, it is said to have performed a **Pivot**.
A startup can *pivot* several times during its existence.

A *pivot* is ultimately a **change in strategy** without *a change in vision*.
It is defined as a structured course correction designed **to test a new fundamental hypothesis** about the product, business model and engine of growth.

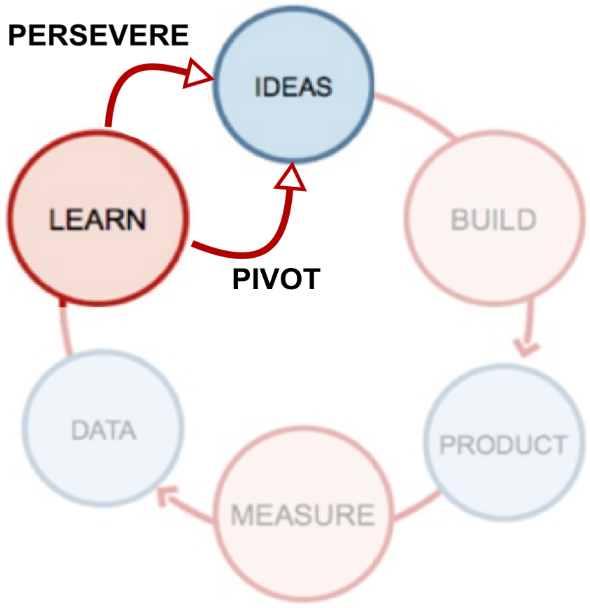
The vision is important. A startup is created because the founder has a vision and the startup is really built and organized around this vision. If the feedback from the field compromises the vision, the startup doesn't need to pivot, it needs to resign, cease its activities and another startup, another organization aligned to the new vision should perhaps be created.

There are various kind of pivots:

- **Zoom-In :** a single feature becomes the whole product
- **Zoom-Out :** the whole initial product becomes a feature of a new product
- **Customer segment :** Good product, bad customer segment
- **Customer need :** Repositioning, designing a completely new product (still sticking to the vision)
- **Platform :** Change from an application to a platform, or vice versa
- Many others ...

Pivot or Persevere

Since entrepreneurs are typically emotionally attached to their product ideas, there is a tendency to hang in there too long. This wastes time and money. The pivot or persevere process forces a non-emotional review of the hypothesis.



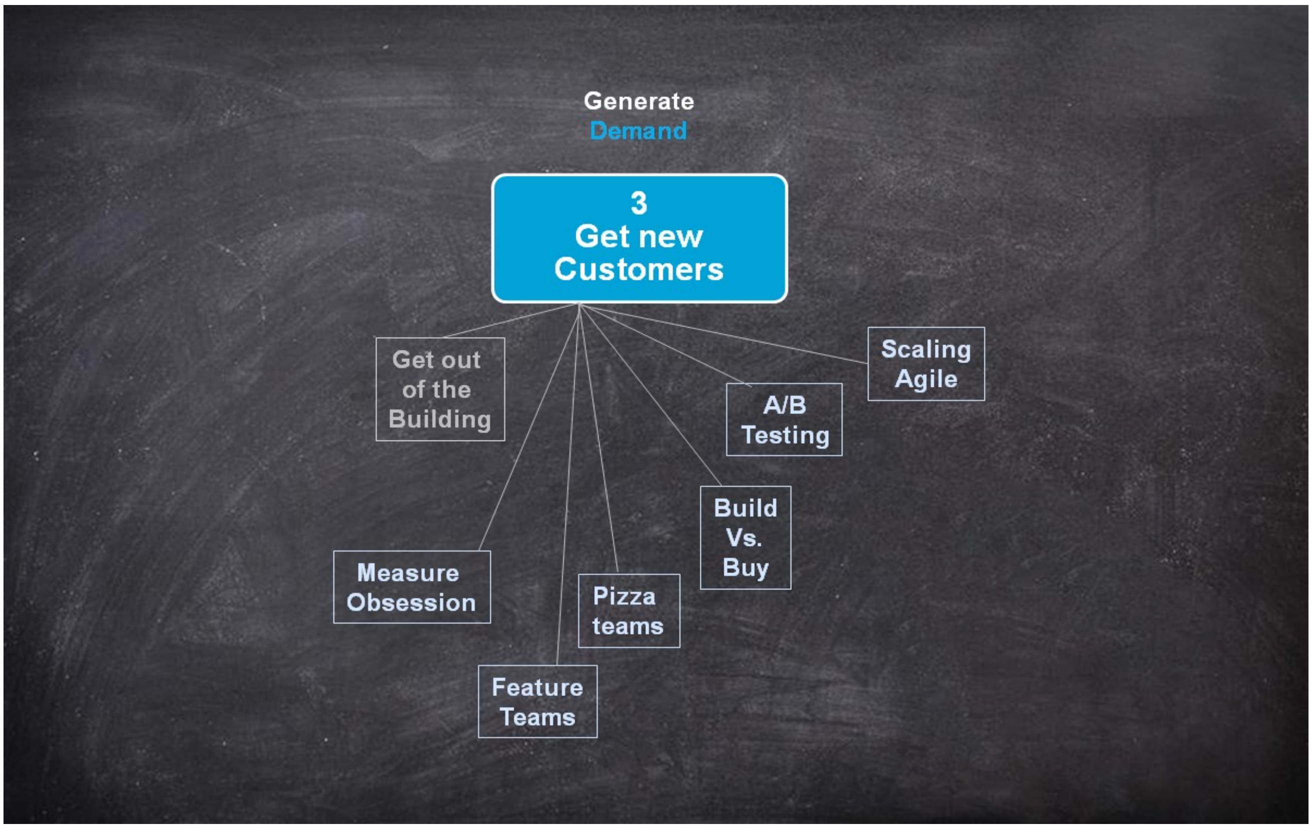
Unsurprisingly, knowing when to pivot is an art, not a science. It requires to be well thought through and can be pretty complicated to manage.
At the end of the day, knowing when to pivot or persevere requires experience and, more importantly, metrics: proper performance indicators giving the entrepreneur clear insights about the market reception of the product and the fitting of customer needs.

One thing seems pretty clear though, if it becomes clear to everyone in the company that another approach would better suit the customer needs, the startup needs to pivot, and fast.

3.4 Get new customers

The third step, the Customer Creation step, to "*start building end user demand to scale the business*", is the precursor to achieve *Business Model Fit*. Therefore, the Business Model Fit stage can be understood as validating the value for the company, where as the product-market fit focuses on validating the value for the customer.

The set of practices I deem important here are as follows:



Again, attaching some of these practices here or in the next and last step can be subjective. In my opinion, the startup needs to embrace this Lean and Agile principles and practices before it attempts to scale its organization, hence the reason why I consider these practices at this stage.

3.4.1 Pizza Teams

Jeff Bezos, Amazon's founder and CEO, always said that a team size shouldn't be larger than what two pizzas can feed, two american pizzas, not italian, needless to say.
This makes it 7 +/- 2 co-workers inside an Agile Team.

More communication isn't necessarily the solution to communication problems - it's how it is carried out. Compare the interactions at a small dinner - or pizza - party with a larger gathering like a wedding. As group size grows, you simply can't have as meaningful of a conversation with every person, which is why people start clumping off into smaller clusters to chat.
For Bezos, small teams make it easier to communicate more effectively rather than more, to stay decentralized and moving fast, and encourage high autonomy and innovation. Here's the science behind why the two-pizza team rule works.

As team size grows, **the amount of one-on-one communication channels tend to explode**, following the formula to compute number of links between people which is $n (n - 1) / 2$.
This is $O(n^2)$ (Hello Engineers) and is really a *combinatorial explosion*.
If you take a basic two-pizza team size of, say, 6. That's 15 links between everyone. Double that group for a team of 12. That shoots up to 66 links.
The cost of coordinating, communicating, and relating with each other explodes to such a degree that it lowers individual and team productivity.

Under five co-workers, the team becomes fragile to external events and lacks creativity.
Beyond ten, communication loses efficiency, cohesion diminishes, parasitism behaviors and power struggles appear, and the performance of the team decreases very rapidly with the number of members.

The right size for an Agile Team is 7 +/- 2 persons.

3.4.2 Feature Teams

Let's first have a look at what is the other model: *Component Teams*.

Component Teams

Components Teams are the usual, the legacy model. In large IT organizations, there is usually a development team dedicated to the front-end, the Graphical User Interface, another team dedicated to developing the Java (Or Cobol :-) backend, a team responsible to design and maintain the database, etc.
A Component Team is defined as a development Team whose primary area of concern is restricted to a specific component, or a set of components from a specific layer or tiers, of the system.
Prior to Agile, most large-scale systems were developed following the component team approach and the development teams were organized around components and subsystems.

The most essential drawback of *Component Teams* is obvious : most new features are spread among several components, creating dependencies that require cooperation between these teams. This is a continuing drag on velocity, as the individual teams spend much of their time discussing dependencies between teams and testing, assessing, fixing behaviour across components rather than delivering end user value as efficiently as possible.
An important direct consequence of this dependency is that any given feature can only be delivered as fast as can be delivered the component changes by the slowest (or most overloaded) component team.

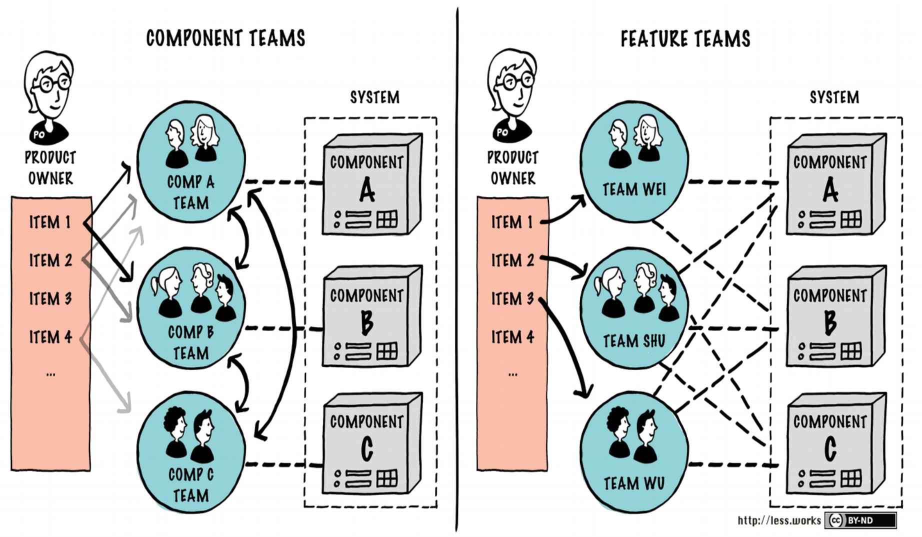
Feature Teams

As such, in an Agile Organization, where the whole company is organized around Feature backlogs or Kanban, it makes a lot more sense to organize the various development teams in **Feature Teams**.
Feature teams are organized around user-centered functionality. Each and every team, is capable of delivering end-to-end user value throughout the software stack. Feature teams operate primarily with user stories, refactors and spikes. However, technical stories may also occasionally occur in their backlog.
A feature team is defined as a long-lived, cross-functional, cross-component team that completes many end-to-end customer features, one by one.

More Information on Feature Teams:

- From SAFe - Scaled Agile Framework
- From LeSS - Large Scale Scrum framework

The difference between both models is well illustrated this way:



(Source : https://less.works/less/structure/feature-teams.html)

A pretty good summary of the most essential differences between both models is available on the LeSS web site:

component team	**feature team**
optimized for delivering the **maximum number of lines of code**	optimized for delivering the **maximum customer value**
focus on increased individual productivity by implementing 'easy' lower-value features	focus on high-value features and system productivity (value throughput)
responsible for only part of a customer-centric feature	responsible for complete customer-centric feature
traditional way of organizing teams - follows Conway's law	'modern' way of organizing teams - avoids Conway's law
leads to 'invented' work and a forever-growing organization	leads to customer focus, visibility, and smaller organizations
dependencies between teams leads to additional planning	**minimizes dependencies between teams to increase flexibility**
focus on single specialization	focus on multiple specializations
individual/team code ownership	**shared product code ownership**
clear individual responsibilities	**shared team responsibilities**
results in 'waterfall' development	**supports iterative development**
exploits existing expertise; lower level of learning new skills	exploits flexibility; continuous and broad learning
works with sloppy engineering practices-effects are localized	requires skilled engineering practices-effects are broadly visible
contrary to belief, often leads to low-quality code in component	**provides a motivation to make code easy to maintain and test**
seemingly easy to implement	seemingly difficult to implement

(Source : https://less.works/less/structure/feature-teams.html)

The Analogy with a Star Trek team makes surprisingly and funnily a lot of sense.

Think of a Star Trek spaceship. The crew is constituted by Commanding Officers, Medical Officers, Medical Staff, Engineering Officers, Engineering Staff, Science Officers, Scientists, etc.
These different functions, competencies and responsibilities are grouped together to work towards a common objective, its continuing mission: *to explore strange new worlds, to seek out new life and new civilizations, to boldly go where no one has gone before*.

Now imagine if Starfleet had instead put all the Commanding Officers in one ship, all medical staff in another ship, and so on. It would have been pretty difficult to make those ships actually do anything significant, don't you think ?
This is precisely the situation of *Component Teams*.
Just as with a Star Trek Ship, it makes a lot more sense to put all the required competencies together in a team (or ship) and assign them a clear objective, implementing that feature throughout the technology and software stack.

3.4.3 Build vs. Buy

This dilemma is as old as the world of computers: is it better to invest in developing a software that is best suited to your needs or should you rely on a software package or third party product that embed the capitalization and R&D of **another** software editor in order to - **apparently** - speed up your time to market ?

In order to be as efficient as possible on the build-measure-learn loop, it is essential to master your development process. For this reason, *tailor made* solutions are better because the adoption of a third party software package often requires to invest a lot of resources not in the development of your product, but instead in the development of workarounds, hacks and patches to correct all the points on which the software package is poorly adapted to the specific and precise behavior required by your own product feature.

In the case of a startup, this aspect is catastrophic. Investing in the development of hacks and workarounds around a third party product, a product that one has in addition to pay for, sometimes depending on the number of machine or users, instead of developing the startup's core business, should just not happen.

This cost aspect is particularly critical of course when scaling the solution. When one multiplies the processors and the servers, the invoice climbs very quickly and not necessarily linearly, and the costs become very visible, no matter whether it is a business software package or an infrastructure brick.

This is precisely one of the arguments that led LinkedIn to gradually replace Oracle with a home solution: Voldemor.

Most technologies that make the buzz today in the world of high performance architectures are the result of developments made by the Web Giants that have been released as Open Source: Cassandra, developed by Facebook, Hadoop and HBase inspired by Google and developed at Yahoo, Voldemort by LinkedIn, etc.

Open-Source software is cool

Of course the cost problem doesn't apply to Open-Source and free to use software. In addition, instead of developing workarounds and patches around Open-Source Software, you can instead change its source, fork it and maintain your different baseline while still benefiting from the developments made on the official baseline by merging it frequently.

At the end of the day, integrating an Open-Source software, in contrary to Editor / Closed Source Software, is pretty closed to developing it on your own, as long as you have the competencies to maintain it on your own should you need to.
Open-Source software is cool, go for it!

3.4.4 A/B Testing

A/B testing is a marketing technique that consists in proposing several variants of the same object that differ according to a single criterion (for example, the color of a package) in order to determine the version which lead to the best appreciation and acceptance from consumers.
A / B testing is used to qualify all kinds of multivariate tests.

An A/B test evaluates the respective performance of one or more partially or totally different versions of the same product or functionality by comparing them to the original version. The test consists in creating modified versions of the functionality by modifying as many elements as desired.
The idea is to split the visitors into two groups (hence the name A / B) and to present to each group a different version of the functionality or the product. Then, we should follow the path of the two groups, their appreciation of the functionality by means of ad'hoc metrics, and we consider which of the two variants gives the best result with respect to a given objective.

For instance, in order to tests if a *trial first approach* is more appealing and leads eventually to more sales than a mandatory buying:

BUY
TRY
20%
CONVERSION
47%
CONVERSION

The A/B test enables to validate very quickly the idea of introducing a trial period for a feature or a product.

3.4.5 Scaling Agile

Transforming a startup into a company, changing and scaling its organization is a unique, and yet challenging, opportunity to make it an agile organization keeping the *lean* genes on which it has been built.
The *agile* aspect here is essential and the approach here actually has a name: **Scaling Agile**.

Scrum and *Kanban* are two agile frameworks often used at the team level. Over the past decade, as they gained popularity, the industry has begun to adapt and use Agile in larger companies. Two methods (among others) emerged to facilitate this process: **LeSS** (Large Scale Scrum) and **SAFe** (Scaled Agile Framework). Both are excellent starting points for using Agile on a large scale within a company.

Both approaches differ a little but also have a lot in common: they consist of scaling agility first among multiple agile team within the R&D or Engineering department and then around it, by having the whole company organizing its activities in an agile way and centered on the engineering team, the product development team.
I won't be describing these both approaches any further here and I let the reader refer to both links above.

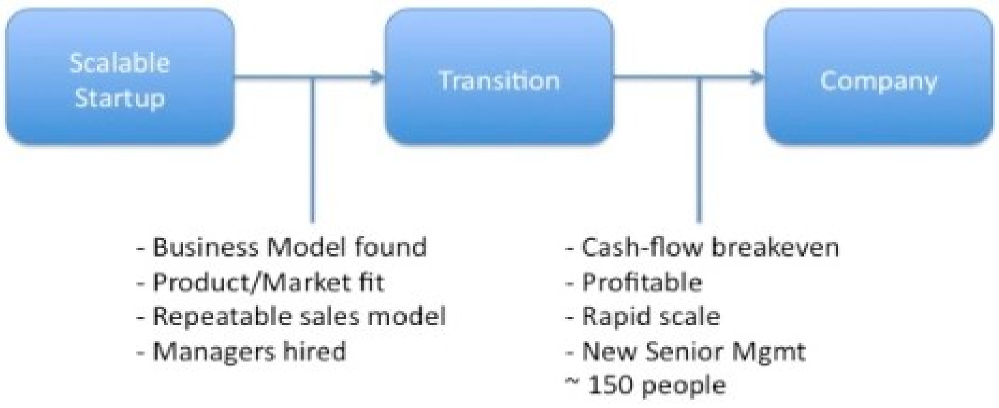
I just want to emphasize how important I believe that is. Scaling Agile is key in aligning business and IT engagement models.

3.5 Company creation

Company creation is the end phase, when all assumptions have been confirmed or adapted, when the product is build in an acceptable form, when the break-even point it reached, and the startup should evolve to a corporation. When that moment is reached, startups must begin the transition from the temporary organization designed to search a business model to a structure focused on executing a validated model.

Company creation happens at the moment the company can transition from its informal, learning and discovery-oriented Customer Development team (startup, temporary organization) into formal departments with VPs of Sales, Marketing and Business Development.
At that moment, these executives should focus on building mission-oriented departments that can exploit the company's early market success.

This is a change of bracket. We think of *Company Creation* since it is really a question of creating a company, from what was "only" a startup. The temporary organization should evolve towards a sustainable and viable organization.



Describing anything further in regards to *Company Creation* exceeds the scope of this article focused on *Lean Startup Practices*.

I can only recommend reading Steve Blank's article on the subject (or the big chapter in the "*Four Steps to the Epihpany*"):

- A Startup is Not a Smaller Version of a Large Company
- The Elves Leave Middle Earth - Sodas Are No Longer Free
- The Peter Pan Syndrome - The Startup to Company Transition
- What Do I Do Now? The Startup Lifecycle

4. Conclusions

The Lean Startup is not dogmatic. It is first and foremost a question of being aware that the market and the customer are not in the architecture meetings, marketing plans, sales projections or key feature discussions.

Bearing this in mind, you will see assumptions everywhere. The key approach then consists in putting in place a discipline of validation of the hypotheses while keeping as key principle to validate the minimum of functionalities at any given time.

Before doing any line of code, the main questions to ask revolve around the triplet :
Client / Problem / Solution
Do you really have a problem that is worth resolving? Is your solution the right one for your customer? Is he likely to buy it? For how much ? All the means are good to remove these hypotheses: interviews, market studies, models, whatever you can think of.

The next step is to know if the model that you came up with and have been able to test on a smaller scale is really repeatable and extensible.
How to put a product they have never heard of in the hands of the customers ? Will they understand it as well with its use and benefits ?

The Lean Startup is not an approach to be reserved only to mainstream websites or fancy internet products. Innovating by validating hypotheses as quickly as possible and limiting financial investment is obviously a logic that can be transposed to any type of engineering project, even if it is internal.
I am convinced that the practices and principles from **the Lean Startup** approach should be more widely used to avoid so many projects burning so much money and effort before being simply dropped.