# The Innovative Software Factory

**About the differences between software companies that innovate and those who just develop customer requests.**

**By**
**Ori Eisen**

**August 2017**

Version 1.1

## Introduction

In VC-backed software companies, the expectation is to run fast, produce an MVP, and get a first-mover advantage in the market.

When the team is composed technical founder/s, who can translate their vision to code, this happens by design.

Even when the team grows to a few developers (6-8), in the first ramp up after funding. The tight environment and working in close quarters do not slow things down.

As soon as you expand to a larger team (10+), and more so when you have virtual members – a fracture may appear that was not there before…

This paper is aimed to discuss how an innovative software company can setup their org chart, processes, and expectations. So they can keep running for a long time as if they were still in the basement.

## Understanding the problem

When you play Space Invaders, you are both the player, the analyzer, the doer, the strategist, the decision maker… you are everything in one mind.

This assures the fastest translation from vision (what you want to see happen) to actions.

Imagine that now you are told to hand the joystick to another person. This person is told not to argue with you, and simply do as you say…
You are still the analyzer and decision maker of what to do next – however now you move from a "mind of one" to a "mind of many". Try playing like this and you will see a few things happen:

1) The original player now feels they are slower, as they need to transfer the instruction to achieve the vision to another mind
2) The new player, Player 2, is feeling frustrated at times, as he/she need to quickly listen and act based on what Player 1 is asking them to do
3) Because the game keeps moving and changing, Player 1 finds him/herself in constant state of flux and appear to Player 2 to be… errm… not exactly sure of what they want. Always asking for something different than what they asked for… and the thoughts in Player 2's mind soon shift to "does this Player 1 know what they are doing?!"

Sounds familiar?

Now imagine that as the team grows, this only cascades further. As Player 1 is telling Player 2 "have your teams do this…" which simply now makes Player 2 seem like they do not know what they are doing to the teams below them and on and on.

If this describes how your company looks and operates, you are not alone.

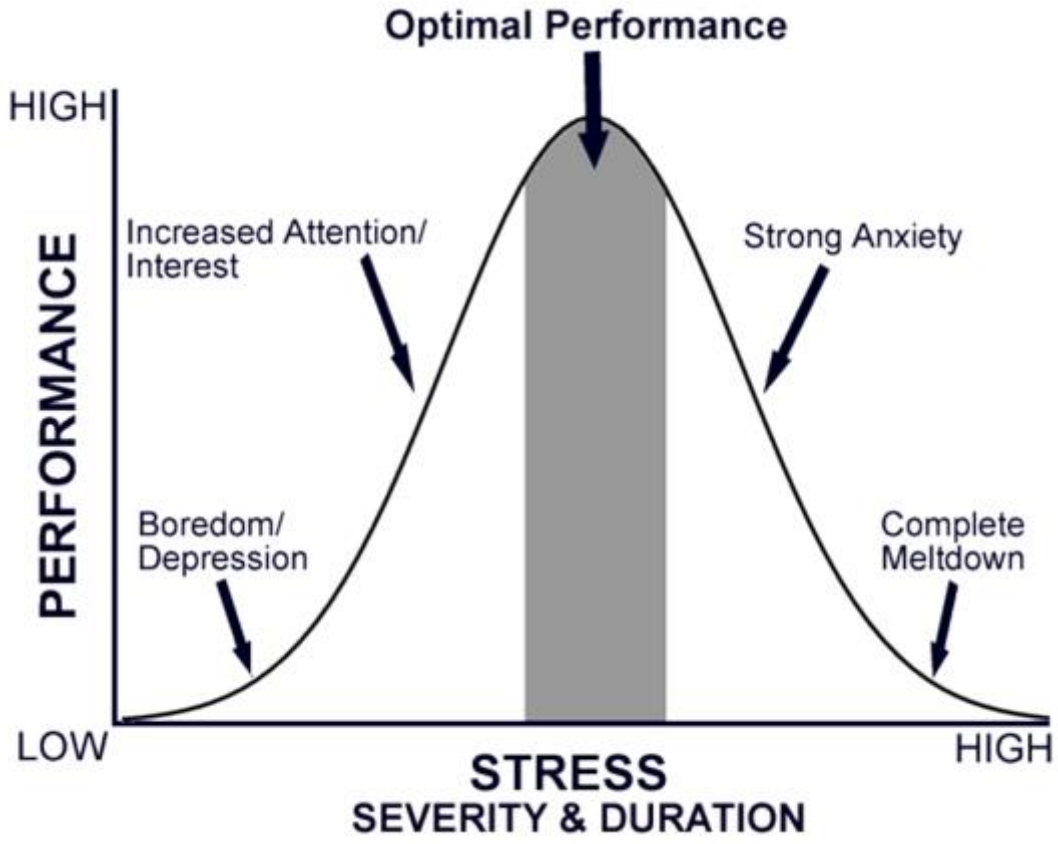
The move from a mind of one, to a mind of two, to a mind of few, and then to a mind of many – is often overlooked. Especially when you get the funding you so wanted, and told your investors "we can now hire more people and scale faster…"

**Optimizing a high-performance team**

There is an old US Navy productivity research study about adding more people to the same task. At first it adds productivity, and then it stops adding productivity and then it hurts productivity…

The reason is that many tasks get done fastest when one person does them. With each person added, you add a layer of "management". The task may get jeopardized if the team does not communicate/coordinate.

Moreover, constant communication and fear of missing a deadline, or total failure are adding stress. A team without any stress will not be productive as they will suffer from boredom. A team with constant over-stress will suffer from burnout…




Now remember that we wanted to be innovative? Not just do the tasks we are asked to do. To be innovative and get the team to think and contribute, they need to be engaged in a way that allows it.

You cannot say "develop this, and if by chance you have a better idea as you are working on it, so we may get some innovation…"

To get the team to engage and allow them to be innovative – time and context needs to be provided. Otherwise you are only giving lip service to innovation but chances are it will not manifest.

To get to a point that you can have an innovative team of software developers – you need to first see all these conflicting forces, and then start planning how you want to operate.

I call this The Innovative Software Factory.

**Why "Factory"?**

The word factory has a lot of connotations, which may sound old fashioned, industrial and… not what hip software companies should be modelling.

Factories have been around for centuries and many operational research studies have helped make them more efficient and productive. What a software team can learn from the anthology of manufacturing research is priceless.

The first lesson is about separation of concerns, so everyone is not doing the same task. If the team opened a restaurant, and they all could cook, greet the guests, wash dishes and take orders – doing everything at the same time by one person is not efficient.

There is a difference between being able to do any of the tasks in the restaurant, and running it this way.

Same with software development… There are basic skills every person on the team must be able to do, and then, they should man their station during the "manufacturing hours". Thus, not everyone is testing, or everyone is monitoring production, or everyone reviewing incoming support tickets…

At a drop of hat a member of the team should be able to switch their context/concern/station, but there should be a clear delineation and purpose set by the manufacturing manager.

To get to high-performance manufacturing, you need to have a disciplined process.

In the movie The Founder, they show what it took to perfect the first MacDonald's. You can see the tireless efforts to perfect what each team-member needs to do, when and how, and the order of operations.

While any day the person who is toasting buns could be manning the fryer, when they man the fryer they are focused on this one task – as part of the whole process. Think about it as the OS of the team.

In the Software Factory, you may have three levels of players:
Line operator, supervisor of a team, and the factory manager.

I know, it sounds off when manufacturing is mentioned next to software…
Will use these to illustrate and make the point, you can call things what you want.

When the software factory has one or two employees, they effectively do everything. They could be manning all stations on one shift, and see the

manufacturing process from beginning to end. Still, in each point in time, when they are doing something – they are doing ONLY ONE thing.

That is separation of concerns.

**How can a "factory" be innovative?**
It can't.

There is nothing else to say. No factory has ever been innovative.

The focus of manufacturing is to repeatedly make the same things, in the most efficient way. If everyone on the factory floor would tinker and experiment, you would get no output, and the sign should be switched from "factory" to "laboratory".

In the software startup, we have a built-in conflict that is not talked about much. It is the innate conflict between high-performance manufacturing and innovation.

They are enemies of each other. They do not like each other. They are in direct conflict with one another, always.

<pause for effect>

"Are you saying that my best developers cannot be both great software makers and be innovative?"

No. I am saying they cannot be both at the same time. Just like they cannot be chefs in the kitchen and the greeter at the door, at the same time.

No one can.

To be innovative, there must be a vision. Something that we are NOT doing today in the factory. Something that has yet to be perfected to a point that you can manufacture it repeatedly.

The vision is what got the investors excited, and now needs to be manufactured. So, let's separate the two:
**Vision** – what we are set to accomplish, so the world will be better, and customers will want to buy. This is a fluffy thing, which is not a well-defined product spec. It is a vision.
**Product** – how we will enable the vision, through a product/service that is a well-defined spec. The spec may change over time based on how the market evolves.

This shows TWO forces at play, which can be distilled to two roles:
Vision "manager" and Product "manager". But we need to account for the Factory Manager who manufactures the product. We will use Vision/Product and Factory manager as our two main roles. One defines why/what to build, and the other how/when it will be built.

For example:
Vision = a world without harmful automobile emissions
Product = Tesla Model 3, priced for the average buyer
Factory = a facility to make the Model 3 and use the BEST processes and technology for optimum manufacturing yield

You can apply this to Apple, Tesla, DocuSign and Twitter. In all cases, you have a vision of how the world ought to be, and then a product that is designed to enable the vision. And then… someone needs to BUILD this product or service in the most efficient way. To get it to market in a fast, stable, accurate and economical way. And without bugs.

**The Room Where It Happens**

For the investors to get what they want, the startup to get what it wants, and the world to get better – the vision/product/factory must be aligned.

This is not a simple thing to do.

Even if the world was stable, it would be hard. However, the world keeps changing and thus, the vision may morph, which by design will have the product morph, and by design have the factory morph.

All three forces should be aware of it, so there is little frustration when things evolve. The key is to ALWAYS begin with the vision as the "true north" at any point in time. Then, the product manager should spec a product that delivers the vision. Then, the factory manager should spec how to deliver the product.

Anything else will lead to fractures in the organization that are a result of misalignment. If the teams do not see these forces, the constant changes could whipsaw the organization.

I propose that to get going, the product manager and the factory manager meet in one room, daily.

To discuss what the product needs are, and if there is any "disturbance in the force" that caused the vision to change. Most days, the vision will not change, and there will only be product spec/manufacturing discussions.

The daily meetup should help foster communications that are constant and limit misalignments.

While there should also be a delineation between product management and software manufacturing, there should also be a symbiotic relationship on the trade-offs.

The Factory manager can help assess the LOE on any feature, story, or epic. The product manager may use these assessments in the decision process. Feature prioritization is made based on all the factors attacking the business or the quick wins that are opportunistic.

The product and factory manager should discuss the WHAT first, and then the HOW. Sometimes just the LOE of the feature may not fit into the business timeline needs, and thus be pushed down in priority. In other times, if three small features can be done in one day, they may get prioritized so there is a clean backlog to continue from.

Disciplines from other fields can be used to software development. Studying best practices in medicine, manufacturing, insurance risk and financial yield can help the product manager and factory manager get to new level of high-performance.

**The Trade Offs**

The product manager can only get to pick two of these three knobs: Speed, Quality, Features.

If the request is for X number of features, with Y quality, the Factory manager will say what is the speed.
If the request is for speed and features, the the Factory manager will say at what quality level.
If the request is for speed and quality, the Factory manager will say how many features.

**How to simplify the tradeoffs and risks?**

The first thing is to agree to a language of what means what. Every feature that enters development has a degree of risk associated with it.

Even the tiniest change…

Case in point is the AWS shutdown of the East Coast because of one semi-colon that was not reviewed and caused a domino-effect that is catastrophic.

Every feature has risk… however not every risk is equal. We can borrow from the field of financial risk management on how to quantify it.

We can classify each feature into three types of risk:
UI – Cosmetics – Risk level 1-3
Logic – Conditional code – Risk level 4-7
Architecture – Infrastructure/API – Risk level 8-10

Cosmetics
Say the product manager talks with the software factory manager about a typo in the interface. He asks to change a label on a UI field from "Naem" to "Name". They classify is as a UI change, that should be the lowest level of risk.

Everything being equal, and without introducing flukes or freak accidents, this kind of change should not pose severe risk to the product, and should be done is a short amount of time.

If we just change a static UI label and give it a smoke test, it CAN go to production. Little risk, big gain, little time.

Logic
If the name of the label is a result of logic/conditional code:
If Variable A >1
Then Label = Naem
Else Label = No Name
End If

If this is the where the label is calculated, the risk level goes higher by design (even though to the end user the end result is the same).

Why?
When we introduce any change to logic/conditional code, we may also introduce a chain reaction to other parts of the product, and introduce inconsistencies. We may introduce a domino effect inadvertently.

Thus, if the change is not to a static label, but rather to anything that is conditional, it must be discussed with a higher level of risk, albeit a relative risk.

Architecture

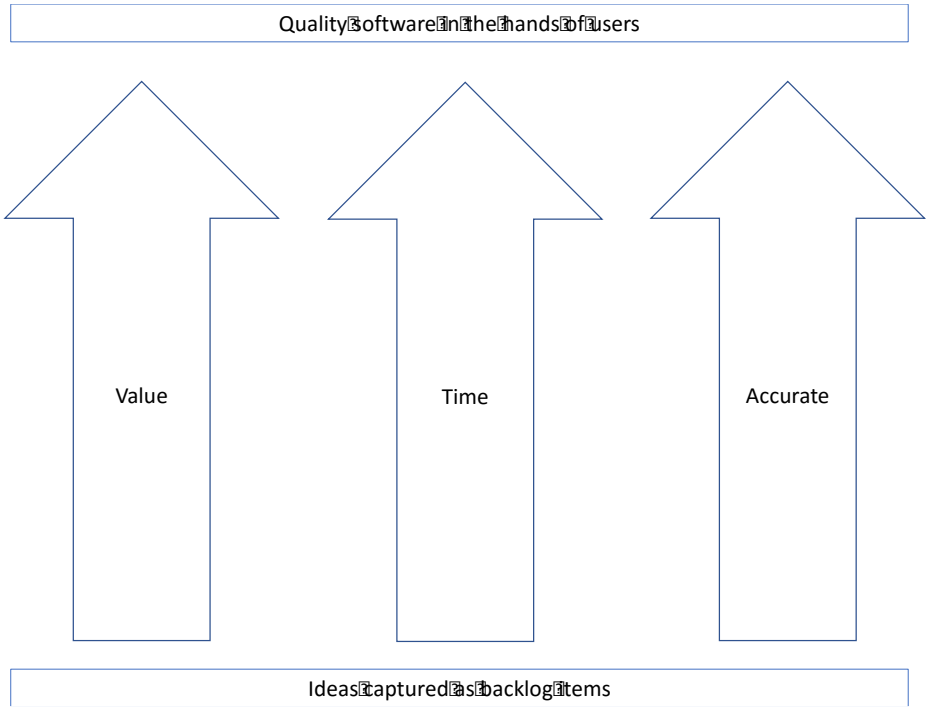
If the label of this field is derived from a field name in a database… then it cuts through all layers of the product (storage/logic/UI) and represents the highest level of risk.

Again, to the end user the result may seem the same in all three cases, but the product manager and software factory manager should choose different paths/tools and risk assessments.

**What makes a Software Factory innovative?**

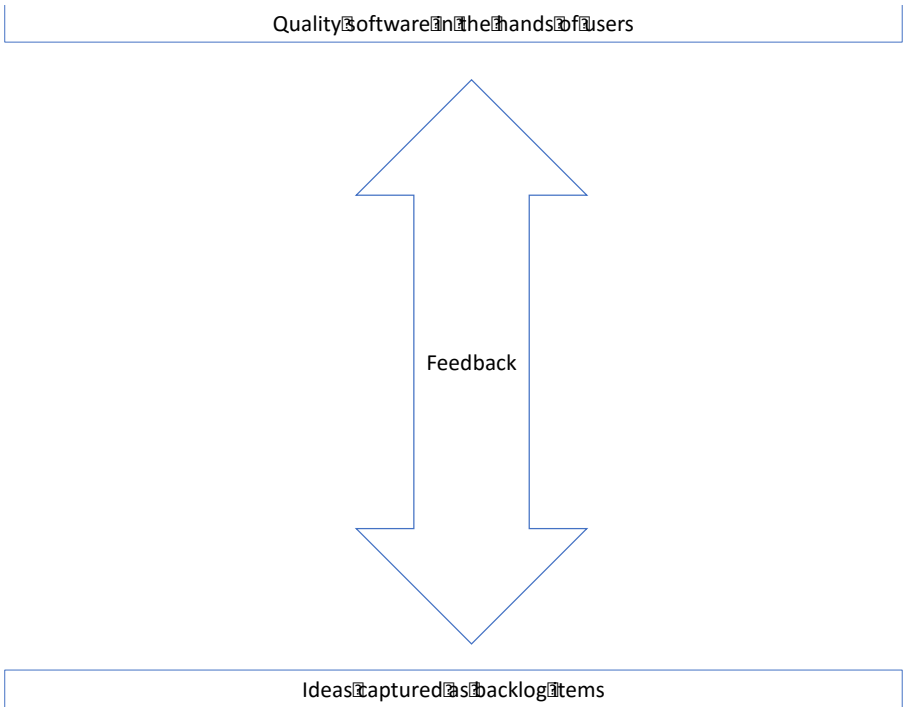
Remember the movie The Founder, and how they perfected the manufacturing floor layout? We need to start there.

The high-level view:




This says that from the moment an idea for a feature is conceived, we want it to get into the hands of users as intended (accurate), in a timely fashion, and to provide value.

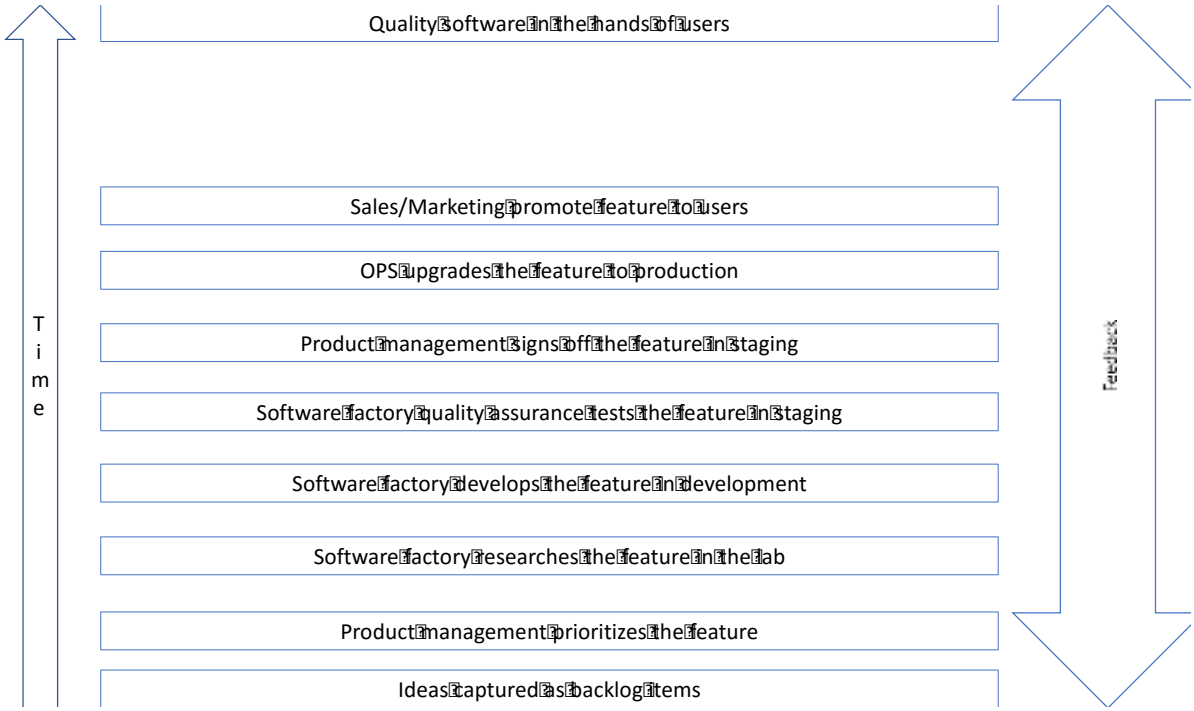
After the feature is in users' hands, we want to get feedback about how they like it, and does it add the value we intended




So what makes the difference between a software factory and innovative software factory?

We need to zoom in to the manufacturing floor, to see it.

Here is a proposed layout of how an idea of a feature should make its way to a user.




The difference between a software factory and the innovative software factory can be seen in two main areas:

1) Where do new ideas come from before being prioritized for development?
2) How much time/money/resources are used in the factory lab?

Sales teams that visit customers often get a litany of features the customers want. Some of these end up in the backlog and get prioritized.

It is easy to fall into the trap of: customers want it, then let's prioritize it.

If all your prioritized featured are coming from customers, you are a software factory. Period.

It is not bad to be a software factory; many companies operate very healthy businesses doing so. However, do not expect innovation to come out of it.

To be innovative, you cannot have all the ideas come from customers/users. Henry Ford said: If I asked my customers what did they want, they would say "more horses…"
If all your features are a result of customer feedback… you have no edge, you are not relevant.

Your competitors can simply play "Software Factory" and give customers what they want, and keep the relationship.

Another way of saying this is: the lightbulb is not a result of perfecting the candle.

To get real innovation, an idea MUST go through the lab to be tinkered with, before it is scheduled to be manufactured. Do not confuse research that is done as part of normal operations of the factory with real research.

Real research is based on an idea that NO customer asked for. Its based on ideas that may or may not pan out. It attempts to completely re-think how things are done, and not mere improvements.

**Innovative Software Factory**: Ideas get into the backlog from internal thought, ideation, research and customer suggestions.
**Software Factory**: Ideas get into the backlog from customer suggestions.

The second way to know if you are innovative or not, is to measure the time/money/resources you are investing in the software factory lab.

Say you want to have a machine-readable code to scan with your app. You can spend little to no effort with buying a library that does that. You may be able to get an open-source project, or even a function that is built into the OS. There is no innovation in that.

It is a good way to get from point A to point B, with spending the least number of calories.

However, if you want to have a different scanning experience than what is on the market, you need to innovate. You need to take some of your engineers and ask them to spend time and come up with a NEW way of scanning, which fits your objectives.

Your objectives may be:
Strategic (I want people to know this is our differentiated scan)
Operations (I want this scan to be faster than what is available in the market)
Efficiency (I want the payload to be small, and take less storage/network bandwidth)
You need to say what you want to innovate, and to time-box it, always.

You can end up with three outcomes at the end of the time-boxed session:

1) No innovation at all. Nothing worked, and we spent the resources in vain.

2) Innovation on the current available technologies was found, which is evolutionary but not revolutionary.
3) Innovation that opens a new way of doing things (light bulb vs better candle)

Do not confused this kind of research, with the case of a customer asking you for a scanning feature, and the team is comparing all the options that are in the market to the client's requirements. This is NOT research per se, and should not be interchanged with spending time on improving the status quo.

**So what is the best methodology to use?**

It is like asking what is better a hammer or a screwdriver?

In a perfect world, a product manager and factory manager agreed on the following:
What is the best feature to work on next
How to categorize the risk of this feature
How to pick the two knobs between: speed/quality/features
Whether it requires research time (not every feature requires research)

They are now still faced with the choice of process of manufacturing this feature. Think about it like picking a tool from a tool box…

The software factory has many tools to choose from, and perhaps the best plan is to choose the tool that is best for the task. There should be no religious wars about which tool to use, as it is a derivation of the task at hand.

Some tasks, like moving from one datacenter to another, or switching from an RDMBS to NOSQL, or switching your code base from Java to .NET – may benefit from a waterfall planning process… It may require pre-planning and documentation prior to "jumping in".

While most tasks in the factory do not require the "big guns", there should be a discipline of when to use what. Not all features are created equal, and thus, may benefit from different processes to reduce time, risk and expense to the factory.

Features that are low risk could use a single dev, using a SCRUM methodology and have a build ready in one hour.

A feature that has conditional code, may benefit from paring two developers side by side.

A feature that moves your code base to another language may need a full project plan for a year of work, broken into epics that can be developed and tested in smaller chunks. You need to show how everything comes together before the first line of code is written.

There is nothing wrong with doing so, when the features call for it. The only thing "wrong" is thinking that every feature should be using the latest buzzword/fad methodology…

This "one size fits all" approach is not giving the factory manager and dev team the options they need. It's like saying: you will build cars, but can only use pliers, because pliers are cool and hip now.

Back to the knobs, you should pair the tool/process with what is important to you. If you use agile, you are saying that being quick to market, and developing an MVP is your goal. That is fantastic.

However, if you are building software that will be autonomous on Mars and must be tested in many different failure modes – you should not optimize for speed, and rather on the number of scenarios you wish to survive and the quality of the testing to simulate real-life.

**Decisions. Decisions.**

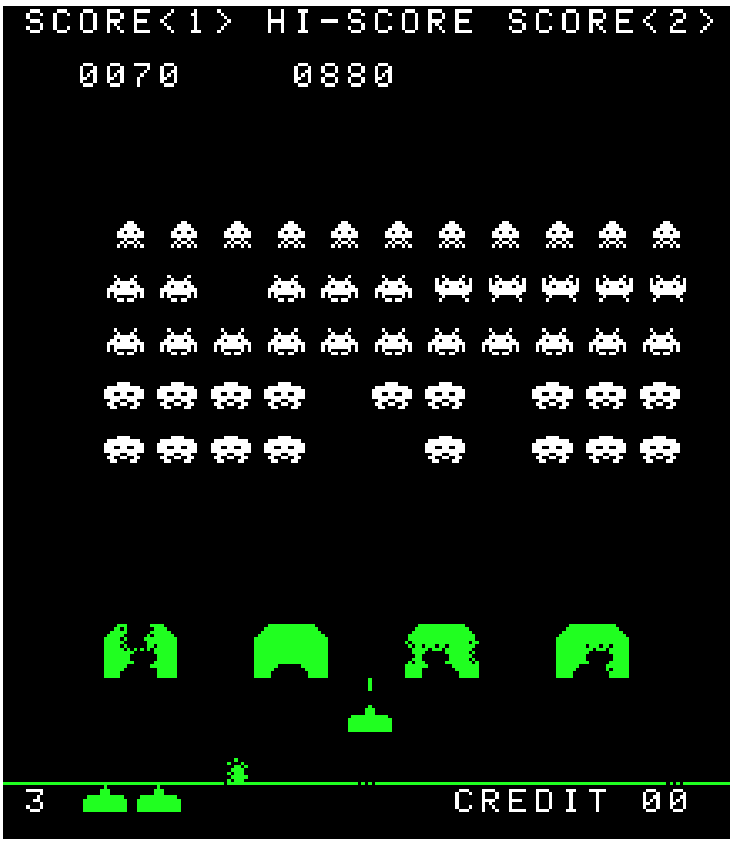
So how does the product manager decide which feature to prioritize?

Ever played Space Invaders?

It is similar. So many different inputs to consider before you decide what to attack next.

Each moment in time in the game provides a different situation and inputs, which causes you to shoot at a different target.

For Example:
Where would you shoot at now?

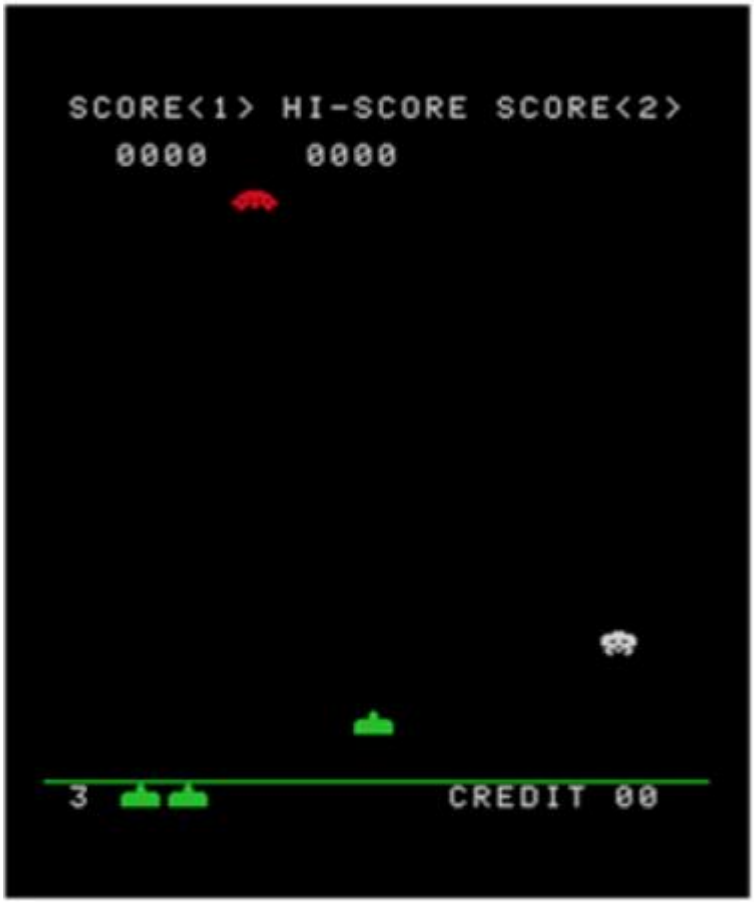



The game speed is slow, there are many invaders to hit, the shields are intact – life is good.

Where would you shoot at now?


You are still in the slow speed, and an opportunity arises to make a big win of point (money in real-life). But it means you need to be distracted from the invaders to hit it. It's a long shot, the target is moving fast and there is a low chance of hitting it. But if you hit it… you are a hero.

Where would you shoot at now?




The last invader is moving super fast, it has one more row before you are toast, and the space ship arrive… should you be distracted and go for glory or focus on the invader?

Quick, before it is…


In real-life, every day, every hour, represents a different set of inputs for the product manager. In rare cases, new input will cause the vision to change. For example, a new law that says that whales are now an endangered species, and your vision is to bring the health benefits of whale oil to the world…

You better think again about your vision, and your product, and not worry so much about the next feature ;-)

Most days, the input may affect your product but not your vision. For example, if your product uses SHA1 as an encryption method, and now it is obsolete – you need to prioritize changing it to another method.

Right after you left the daily stand-up meeting where you asked the team to change from SHA1 to SHA256… you get a call from Microsoft.

"Hello? NewCo? Yes, this is Microsfot and we chose you to present at our annual customer event. To do that, you need to first integrate to our cloud authentication…the show is in two weeks…"

This is akin to the space ship showing up and you decide that it is worth going for it. So the product manager and the innovative factory manager should both recognize that every decision is only good for the current set of inputs. Lastly, much like you cannot explain every single shot you make in Space Invaders, as your decisions

become innate – great product managers react instantly to the input and can make instant decisions.

Making an instant decision that makes years of experience (playing the game) and also gut feeling, is the difference between good product managers and great ones.