KanBan and it’s foundations (JIT&Lean)

KanBan first emerged in post II World War Japan as a very important component of “Toyota Production System”, a groundbreaking shift in approach to **production process management** in **car development.** This system know also as JIT (just in time) was invented & implemented in Toyota by group of engineers led by Taiichi Ohno.

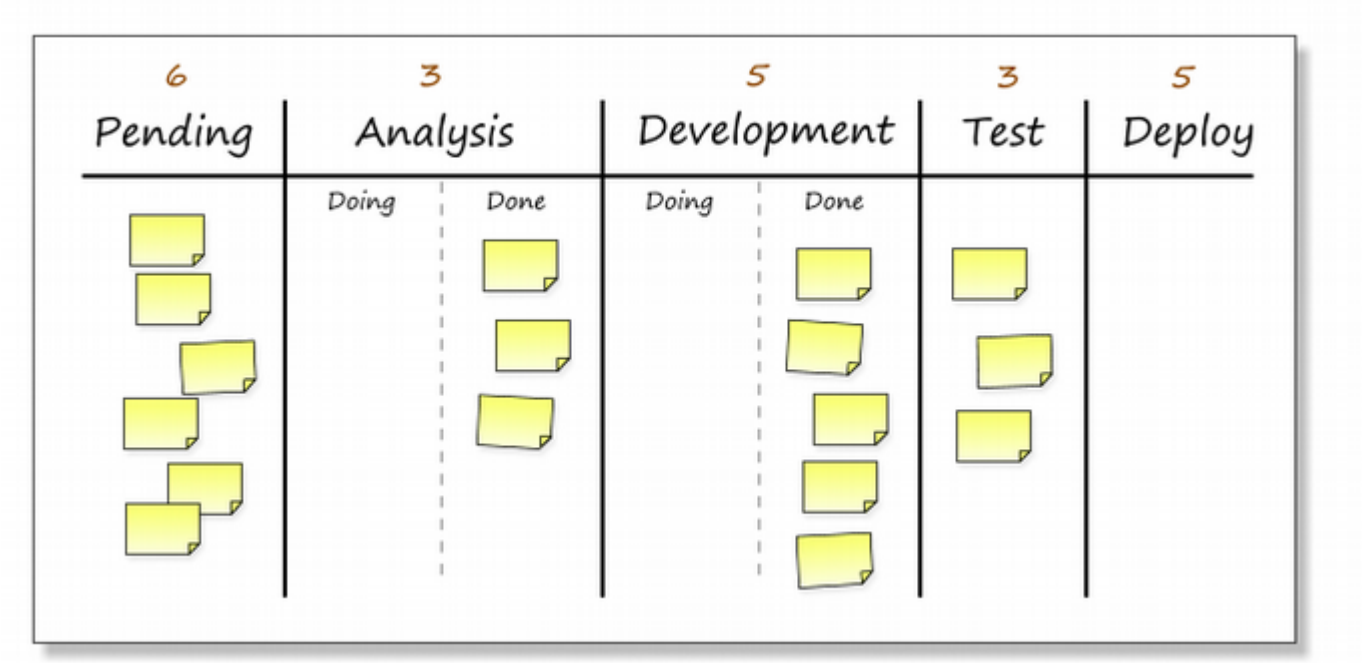
In yearly 90’ JIT production process management was confronted with other manufacturing systems in the book “The Machine That Changed the World” (by James P. Wormack, Daniel T.Jones & Daniel Roos). That confrontation led to the definition of what we today understand as **LEAN** defined in its principles:

1. *Specify value from the standpoint of the end customer by product family.*
2. *Identify all the steps in the value stream for each product family, eliminating whenever possible those steps that do not create value.*
3. *Make the value-creating steps occur in tight sequence so the product will flow smoothly towards the customer.*
4. *As flow is introduced, let customers pull value from the next upstream activity.*
5. *As value is specified, value streams are identified, wasted steps are removed, and flow and pull are introduced, begin the process again and continue it until a state of perfection is reached in which perfect value is created with no waste.*

It’s hard to catch at first glance how exactly improvements in the process of car manufacturing can be adopted in software engineering but as you may already recognize LEAN principles were one of inspirations for borth **SCRUM inventors & Agile Manifesto signers.**

KanBan board, WiP limit, pull system

The easiest way to express KanBan it’s by **clearly showing** a full **map of the value stream** on **KanBan board** (later defined as “board”). Board should be visible & easily obtainable by all developers that are responsible for all steps that occur through value stream.



Direction of value stream – left to right only!

>—>—->—>—->—>—>->—>—>—>—>—>—>—>—>

Now we have a map of the development of our product. In our case that value of the product is working software so we can imagine that every sticky on board is a small “batch” of potentially working software (later defined as “feature”). Every feature that is planned for development has to “travel” through the board from left to right through all steps (pending, analysys, development, test, deploy) before it can be delivered as a value to the end customer. At this point you may ask where is the catch? It does not look like a hard thing to implement this kind of workflow. There are (at least :D) two caches. One is implementing Work in Progress Limit (WiP Limit) , second one is implementing a Pull System.

Work in Progres Limit (WiP limit):
There is a number above every step name on board and that numberdefines maximum amount of “stickies” (features) that could be developed **at any given stage of value stream**. It may be counterintuitive to recognize that limiting work in progress may actually help to deliver value to end customers faster. The truth is that it really helps and this happen due to understanding of Little’s Law which states:

> Little's Law reveals that in general, for a given process with a given throughput, the more things that you work on at any given time (on average), the longer it is going to take to finish those things (on average).

Pull System:

In "normal" push system a person responsible for the analysis step would just **PUSH** a feature that has just been analysed by him to development. In this situation the developer is **not in control** of his own work in progress limit. He either waits for features to "jump" into development (which is a waste of time) or he is covered by a huge amount of features to develop at once thus development step become a bottleneck. (resolving bottlenecks also creates a waste of time).

As you see on the board in two of the existing steps (analysis & development) there are additional sub-steps – doing & done. In a pull system **developer is PULLING** feature to develop from a done sub-step of analysis step. This gives him (developer) control over his own work in progress limit. He can start work on new feature **only if** he has a capacity which **he creates** by finishing work on previous feature.

Creating a **pull system** and **constantly inspecting & adapting it** *theoretically*:

- eliminate possibilities of waste of time due to waiting for work and bottleneck resolution;
- gives the possibility to develop & deliver *flawless* value to the end customer.

Of course in reality we are not able to eliminate waste totally but mastering flow of work through KanBan board is a great tool to keep waste on a minimum possible level.

You may think again that it's not a big deal to implement WiP & Pull system but please confront your expectation with information that it took **10 YEARS (50'-60')** for Toyota to fully implement the JiT method for manufacturing. They have struggled with countless bigger (4-month strike of ALL manufacturing workforce due to **NOT accepting** that they have to now learn how to operate all machines in certain production line) or smaller (how to exchange information between workstations that are louder than airplane engines) problems during transition from "standard" way to this new JiT way of production.

SAILINGBYTE

Summary: Pros, cons of Kan Ban

Summary:Pros, cons of Kan Ban
Even if something is very hard to achieve it does not mean that You would not pursue it. Implementing KanBan in organizations/teams now is way easier due to the now long history of do's and don'ts scripted in multiple books about LEAN in practice.

KanBan is a **very flexible** but **hard to master** tool to:

- developing and sustaining both complex products (for example software development) & "not so complex" products as cars etc;
- fast & precise recognition where & when bottleneck occurs (thus giving possibility to focus ONLY on those problems that interfere with value delivery, focusing on other problems is a waste);
- minimizing amount of wastes that occur through certain work/production flow;
- smooth delivery of value to end customer;

Pros of Kanban:

- KanBan is radically non-prescriptive and easy to apply to **any given** You can spend 10 minutes to draw the first draft of KanBan board for your workflow and you can already **start using it**. There are no roles, artifacts, set of responsibilities, secret meetings etc. KanBan is adapting to You. Not you to KanBan.
- Once Kanban workflow is established a certain flow starts to dictate pace for work items to travel through the value stream. This gives a prediction of how fast a certain amount of value can be delivered to the end customer.

Cons of KanBan:

- KanBan is radically non-prescriptive and easy to apply to any given workflow **but it is not** by all means an easy process to sustain! By definition KanBan it's a constant improvement seek and waste reduction cycle. There is no "end" to the KanBan adaptation process and if an organization is not willing to continue that process it **will not** find the true reasons of occurring wastes & bottlenecks. "Lack" of strictly defined meetings structure, roles and responsibilities can be also frustrating because KanBan demands **absolute attention** to details and mutual understanding of lean principles

from **every** value stream step **team member**. Forcing KanBan “wrong way” (without absolute understanding of its purpose by all participants) may become a case when mutual responsibility to seek perfection is misunderstood by individual responsibility to point a finger at anyone who makes mistakes.
- Once KanBan workflow is established a certain flow starts to dictate pace for work items to travel through the value stream. This gives a prediction of how fast a certain amount of value can be delivered to the end customer.

That is exactly what it is – a prediction. Now imagine that a company X is asked by a client to deliver a complex, fully developed product in a strictly defined period of time . If X will accept such a request **during adaptation** of KanBan they are exposing themself to risk of not knowing the real throughput of their team. They will be under enormous pressure to deliver on time which will for sure dramatically grow cost AND lower quality of end product. (as 2x growth in the amount of developers **is not going** to magically deliver product 2x faster, quality of end product must suffer)