Lean Software Development with **Kanban**

Free ebook

Dimitar Karaivanov
Kanbanize CEO and Co-founder

kanbanize

Why should you read this case study?

For the past 28 months, we were able to push out **26 new releases**, with just two gaps in December, when we celebrate Christmas. As a matter of fact, the versions were there, however, we decided against releasing them because it might have meant very few people would make use of the changes during the holidays.

With an average of six new features per month, we have released **150+ new features** to production. The living proof appears in our Kanbanize Blog (make sure to subscribe for updates).

As of this moment, at Kanbanize we have a total of **one (1) open customer issues.** Yes, this is not a joke, we have almost no open customer issues for a code base exceeding a million lines. Of course, there are issues that we are just not aware of, but out of the reported ones, we've fixed them all.

We have around sixty open internal issues, most of which are low priority. We are depleting these quite quickly, so in around four months we expect to have **zero open internal issues as well**.

We managed to create an eight-digit company in three years and effectively multiply the initial funding **by more than 30 times.**

The numbers show that we must be doing something right and this book talks about the engineering part of this modest success.

Okay, let's do this!

Contents

The Beginning

I co-founded Kanbanize because I envisioned the exponential adoption of the Kanban method some 6 years ago (the name is not a coincidence). Back then (2010), I became a Lean champion at a big German company. We were a team of Lean thinkers helping the RnD department (and to some extent the entire company) change course from Waterfall to Lean and Agile delivery methods.

To this day, I am not sure how I got this role and to be quite open, I have no idea why Christoph chose me before other much more experienced managers in the company. I guess it was a stroke of luck that would later prove even more significant than I anticipated at the time. Christoph came from the biggest German software company - SAP. He was famous for achieving outstanding results there and that's why our CTO invited him to take up the SVP position.

I met Christoph for the first time a couple of months after he joined the company. We talked about my team, which was responsible for the performance of the product suite. I remember showing off the good results that we had got during the past two years, but he didn't seem too impressed. I realized later that he was probably not that interested in what we were actually doing, but focused on assessing the people he could count on later. He had plans that none of us could have suspected.

His plans turned out to be simple but revolutionary for the situation that we were all in. He was on his way to establish a promotion process for the entire product suite (20+ products working together). This basically meant the following:

1. Each product team had to have a nightly build and as many automated tests as possible.
2. All product builds had to be integrated each night and tens of thousands of automated tests were executed against the entire product suite.
3. The teams were not allowed to work on other things unless their builds and tests ran smoothly. Even if a single test was failing, the issue was marked as a blocker for the whole suite and had to be resolved with the highest urgency.

Now, I want you to evaluate the chances of success of this approach, while keeping in mind that it took some of the teams more than a month to build their products. That's right - the last successful build for some products was more than a month old and these same guys had to not only build every day, but also run automation tests, none of which could fail. Quite revolutionary indeed.

In the beginning, people thought Christoph was not serious about the promotion process and even ridiculed his approach. However, they soon realized that he wasn't joking and started working hard to get this whole thing done. It took everyone a lot of time to get there, but after an

year things started to look better. We had a well-oiled promotion process of consistently producing stable and well-integrated suite builds, representing 20+ products working together.

A lot of success followed. We were able to release a super high-quality release on time and that had not happened for many years prior to the changes. We started adding more tests to the promotion process - performance tests, upgrade tests, installation tests, etc. In my team we were able to detect a performance regression on the next day after the release was introduced. Just think how many companies have performance tests in the first place and then think how many of those companies can detect regressions throughout the promotion cycle of 20 products. We were nailing it down every single day and, in fact, there were a lot of performance bottlenecks fixed without even raising promotion blockers for them. This was a sign that the product teams could actually spend time improving the product's architecture and not just put out fires, as they were used to.

Watching all of this from a first person perspective and actually being an active part of the transformation got me really fascinated. I was amazed by the effectiveness of Christoph's ideas. That got me further into Lean and I started to develop and test my own ideas in the teams with which I worked. That's how I learned about Kanban and this was the element that changed my life for good.

...

Meanwhile, one of the main initiatives that we had with the Lean Thinkers team was the "Feature Management" topic. The problem there was that there were many user stories distributed across 20+ teams and there was no reasonable tracking as to of what was happening on the feature level. In other words, only the corresponding product manager was partially aware of what was happening in the team and there was absolutely no way to figure out the progress outside this group. Apart from that, there were many cross-team features that were usually delayed, due to the lack of synchronization and focus holding back some of the teams.

What we accomplished was to establish a mechanism that would allow us to track the overall progress of all feature work for more than a dozen different products, split into five investment areas. This mechanism would allow us to report weekly on how many features have been completed, how many were being worked on, how many user stories were in progress, etc. all of this distributed by investment area (management wanted to know where money was being invested).

All the data was presented using a monstrous excel spreadsheet, which was generated once a day from a central database. There was a working student whose primary job was to get the data extracted from a tool starting with J and then get it imported into the central database, from which the spreadsheet was seeded. Then, we would use this spreadsheet to run our weekly meeting, during which we would come up with observations and suggestions for improvement.

There was also a separate meeting of all VPs of the corresponding investment areas to discuss the feature status and take actions to unblock given features, if they appeared stuck.

This solution worked really well compared to the chaotic environment in the past, but, of course, some things were far from perfect:

- Product managers lacked the visibility into what was happening in RnD and were always unhappy, because “the development teams were very slow”.
- There was no mechanism to stop teams from starting new work, which caused a dramatic increase of the features that were started, but not finished. At one point we had to forbid starting new features so that we can get at least something out of the door.
- The features that had to be worked on by more than one team were frequently left behind and special synchronization efforts were always necessary.

Being involved with the Feature Management initiative, while at the same time experimenting with Kanban in my teams, made it really easy for me to see that using a Kanban system on the global feature level would solve a lot of problems (and expose a lot of new ones). Had we implemented such a system, we could have:

- Ways to manage work in progress limits so that we achieve better flow and eventually better throughput from the same people and resources.
- Transparency into the current progress, thus reducing status reporting and actually making the reports much more accurate.
- The cycle time and block time metrics to support a Kaizen culture. Kaizen is only possible if you have baselines and ways to track changes, be it positive or negative.

Pursuing this idea further, I started searching for better alternatives that would allow us to map this whole management process. Looking at the different tool offerings, it became evident that tooling support was years away from what I envisioned my organization needed. That was understandable. After all, we were quite innovative with what we were doing and tooling had not yet caught up.

That’s how Kanbanize was born – out of necessity. I was lucky to have Christo, a childhood best friend of mine and now CTO and co-founder of the company, working on the first version. He came up with a prototype after six months of work and I went ahead to demo it to the Lean Thinkers team. I actually proposed Kanbanize as the official feature management instrument in the company, but the project didn’t fly. It’s just that the company needed much more than we could offer with the first immature version of the product (this was more than 5 years ago).

However, Christo and I could recognize the potential of this solution and we could probably see what others didn’t, so we decided to quit our jobs and completely dedicate ourselves to the dream to “Kanbanize the world”. This was, and still is, the best professional decision I have ever made in my life.

Getting an investment from a local venture fund (thanks Eleven!), on-boarding Biso as a third co-founder and quitting our lucrative jobs was the easy part. When we found ourselves in the situation with an ugly beta-version, just $125,000 in the bank and a total revenue of $3,000, we realized that we needed to act fast. We just had to find ways to make maximum impact with the least amount of money possible and make the company profitable. Oddly enough, this looks pretty much like the situation which Toyota were in when they started their automotive projects. They were a small Japanese company that wanted to build cars, but didn't have a lot of money and know-how. That is how they created Lean - they just didn't have the luxury not to.

Full of fear and hesitation we continued with our endeavor to Kanbanize the world. We became a team of five guys, the three co-founders and two employees. We were getting some traction, but it was far from what we needed. That's when Christoph came back into the picture.

One day, I got an unexpected call from the HR manager of the office in Sofia. She told me that Christoph wanted me to take the position of a Managing Director (the position from which Biso had resigned in order to join Kanbanize). I declined right away, because I was fully dedicated to Kanbanize, but I felt I should thank Christoph for this generous offer. I sent him a short "Thank you" email. He replied. I replied again, and a couple of months later he became an investor in the company. After all, it was a company conceived on the basis of his ideas, so his place on our board was a well-deserved one.

Shortly after that, things just started to happen and although it was not easy, the world is a much better place now. But if there is one thing that made us succeed, it is the Lean Thinking that we've mastered so well. My goal for the case study is to convey as much of this "Thinking" as possible. Learning through experience is priceless, but if I could save you even a month of mistakes, I would consider my mission successful.

Literature

Reading "Lean Thinking" by James J. Womack and Daniel T. Jones was how I initially became acquainted with the concepts of Lean. The ideas it presented made so much sense that I started reading book after book on similar subjects. Many of the things we do today would not have been possible without some of the books I have had the privilege to read. That is why I would like to share a distilled list of "must-reads" with you right away:

- The Goal by Eliyahu M. Goldratt and Jeff Cox
- Lean Thinking by James J. Womack and Daniel T. Jones
- The Toyota Way by Jeffrey Liker
- Kanban by David J. Anderson
- The Principles of Product Development Flow by Donald G. Reinertsen
- Lean Product and Process Development by Allen C. Ward and Durward Sobek
- Lean Software Development by Mary and Tom Poppendieck

Find the time to read these books. They will be well worth your time.

Case Study: Lean Software Development at Kanbanize

Mastering Lean has been our task at Kanbanize for the past six years and, in this chapter, I am going to share what we've learned and what is still a challenge. This is going to be a very practical course and I hope you will find useful tips that you can directly apply to your work.

Before we start, I would like to quickly go through the reasons why Kanbanize was created in the first place. The main idea for the tool was to ease the process of breaking down bigger chunks of work into smaller pieces and to ensure seamless tracking not just on the user story (task) level, but on the feature/project level too.

The issues that Kanbanize was meant to tackle of individual features were and still are: (or projects). into the progress

- Lack of visibility
- Too much context switching between multiple projects or tasks.
- Slow development processes and overall poor productivity.
- Lack of mechanism to prevent teams from uncontrollably starting new work and therefore harm the overall performance of the company.
- Inability of the product management to proactively work with the engineering teams due to the lack of real-time status updates.
- Lack of actionable metrics to be used for continuous improvement (Kaizen) initiatives.
- Lack of consistently up-to-date status reports available not just for the management but for everyone involved in the project.
- Hard to use tools that were never meant to be used by product management, but just engineers.
- Show what engineering is working on so that product management won't always think that developers are slow and lazy.
- Product management engaging with individual engineers directly, due to the lack of visibility of the actual progress.

It took us more than ten years at corporations like SAP, Johnson Controls, Software AG, ProSyst and six years at Kanbanize to get to where we are today and it won't be exaggerated to say that we are still in the beginning of the journey. However, knowing how much effort and knowledge it took us to get to that stage, I would like to share some of our experience, so you can make the transition not in six, but in probably one or two years.

Quick Company Overview

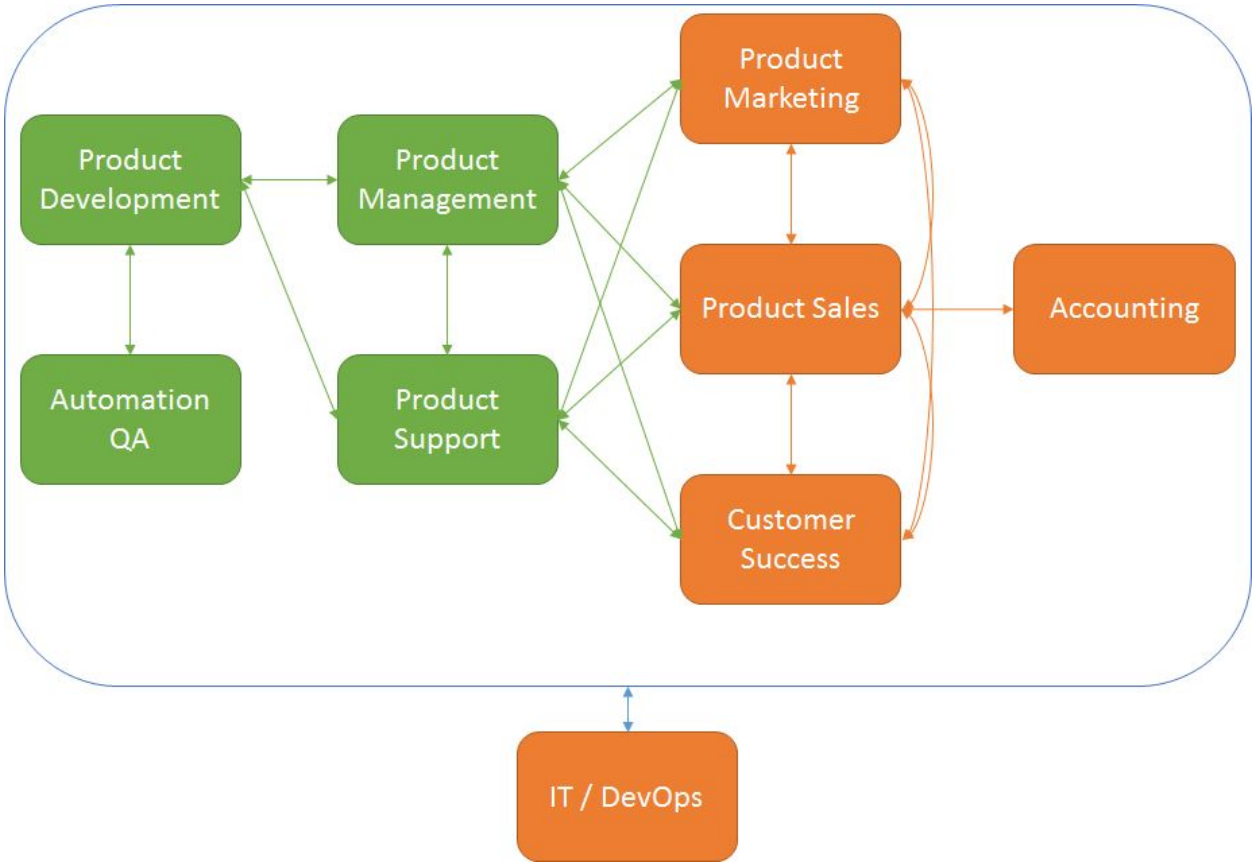
We are a product company and for the moment we are working on a single product, but we may change this status really soon. Our release takt time (how often we deliver value to our customers and how often they are ready to accept it) is one month, which means that we roll-out

new features to all our users every four weeks or so. Some of you may recognize a Scrum sprint here, but what we do is actually quite different. Instead of planning what we will be able to accomplish in a month, we do our best to accomplish as much as possible and, when the takt time comes, we just release what has been completed. This is a fundamental difference between Scrum and a Flow-based method, such as Kanban. I urge you think more about that, it may sound like a minor difference, but it is, in fact, a huge gap in productivity between these two approaches.

Our monthly release goals are:

- Zero open customer defects (achieved consistently)
- Zero high severity internal defects (achieved consistently)
- Less total internal defects than last release (achieved consistently)
- At least two major new features and at least two minor (achieved consistently)
- Zero regression defects in each release (work in progress)

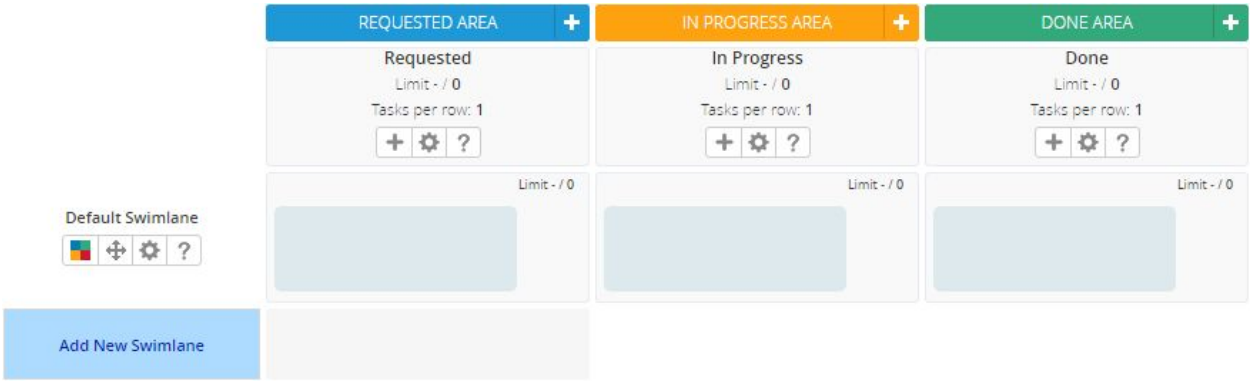
To give you a better perspective about the company structure, let us explore what service teams (or also services) we have in the organization and how they interact with one another.



Service teams (at Kanbanize)

The use of the word “services” is intentional. We do have teams in the company, but we think of them as services. A service has an input, processing steps and an output. In order to have all services working well together, we have established rules to control how services take input from other services, how work is actually being completed and what the output should be. We can compare our company to a computer program that has the different modules talking to one another via predefined interfaces. We do this despite the fact that we are all in the same office. The support person may be sitting next to you, but instead of asking them to do something for you, we insist that people create tickets in the Support Kanban board, which is essentially the input for support. The same principle is with each and every service in the company, be it marketing, sales or product management. This does not mean that we discourage face-to-face communication. There are many situations when filing a ticket and discussing it personally is the right thing to do and we do it quite often.

One feature we have in Kanbanize, which many users consider a limitation (at first), is the predefined sections of the Kanban board. If you try to edit a board in Kanbanize, you will see that there are three sections on that board - REQUESTED, IN PROGRESS, and DONE.



Requested, In Progress and Done areas in Kanbanize

Why would we force everyone that uses our software to have to choose which column belongs in which area? The answer is quite simple - each Kanban board is meant to be a service and each service has an input (REQUESTED AREA), processing steps (IN PROGRESS AREA) and output (DONE AREA). Having this separation, we can perform a much more sophisticated analysis of the data in the board and provide feedback to the users.

Unsurprisingly, all services within our organization use Kanban (and Kanbanize) to do their job. It is our universal delivery tool, which, integrated with email, turns out to be a really powerful system that can be employed to serve various goals. However, this book is about product development, so we will mainly discuss the services marked with green in the image above and only partially touch on the orange ones. Let us get started...

Product Management

As shown on the Figure above, product management is one of the central services in a product company. It is practically the bridge between product development and sales/marketing. Being a central service, product management has to take care of multiple initiatives such as:

- Articulating the product vision and strategy to all other services
- Capturing customer feedback and converting it to meaningful features
- Reporting on the current status to all interested parties
- Actively researching new technology to stay up to date with the new trends

So how does a product manager come up with the product strategy in the first place? How does she capture ideas, in a way that is easy to work with, without spending too much time maintaining huge backlogs? What can be done to easily report on the progress? What metrics are important?

These are all important questions and we will try to answer them all with concrete examples from our daily work. First, we will go through the backlog management, then we will cover the actual delivery model, which is in essence an Upstream Kanban implementation, and we will finish with a set of important metrics to monitor.

The Product Backlog Board

According to Lean, maintaining an ordered list of ideas (backlog) is waste. It is waste because working with hundreds of different ideas, always trying to refine them and prioritize them, takes time. If they never get implemented, this time is simply wasted. However, you still need a way to capture your ideas and customer feedback. We do that with the so called “All Features” Kanban board. This is where we capture all the ideas that we come up with as well as the customer feedback we receive. The board structure looks like that:

Proj Mgmt and Analytics	Run-times and Automation	Links and Dependencies	Email and Collaboration	Ease of Use	Administration	Integrations	Process Governance	Mobile	Misc
High Priority									
Medium Priority									
Low Priority									

The Structure of the Product Management Backlog in Kanbanize

The explanation of the columns listed above:

- Proj Mgmt and Analytics - this column contains ideas or requests related to how managers track status and report on it. Also, this column covers the requests related to the analytics module.
- Runtimes and Automation - if you have used Kanbanize you should know that the runtime automation policies are the heart of our software. This is the way to automate your processes with the help of super-flexible business rules. This column captures ideas/requests regarding potential improvements in that area.
- Links and Dependencies - requests related to card linking and card dependencies. These requests usually come from project managers trying to implement a portfolio board (we will talk about that in a minute) or some sort of a schedule.
- Email and Collaboration - as the name suggests, this column covers the collaboration part of the tool. This is mainly the Email Integration piece and the Moxtra collaboration module (chat + screen sharing meetings) with which we are experimenting.
- Ease of Use - this one is easy to figure out. Here, we file requests that do not affect a concrete area of the software but, rather, the overall user experience and usability.
- Administration - again, the name speaks for itself. This is where we keep all requests related to administration.
- Integrations - a lot of companies integrate Kanbanize with their own systems via the developer's API and this is the column for such requests.
- Process Governance - everything related to permissions or implementing concrete behavior in the system is kept in this area.
- Mobile - work related to the iOS and Android Kanbanize apps.
- Miscellaneous - anything that does not fall in some of the categories above.

This is how the actual board looks in Kanbanize (this is a huge board. In order to get a sense of it in its entirety, it has to be minimized to the point where nothing is readable, sorry).

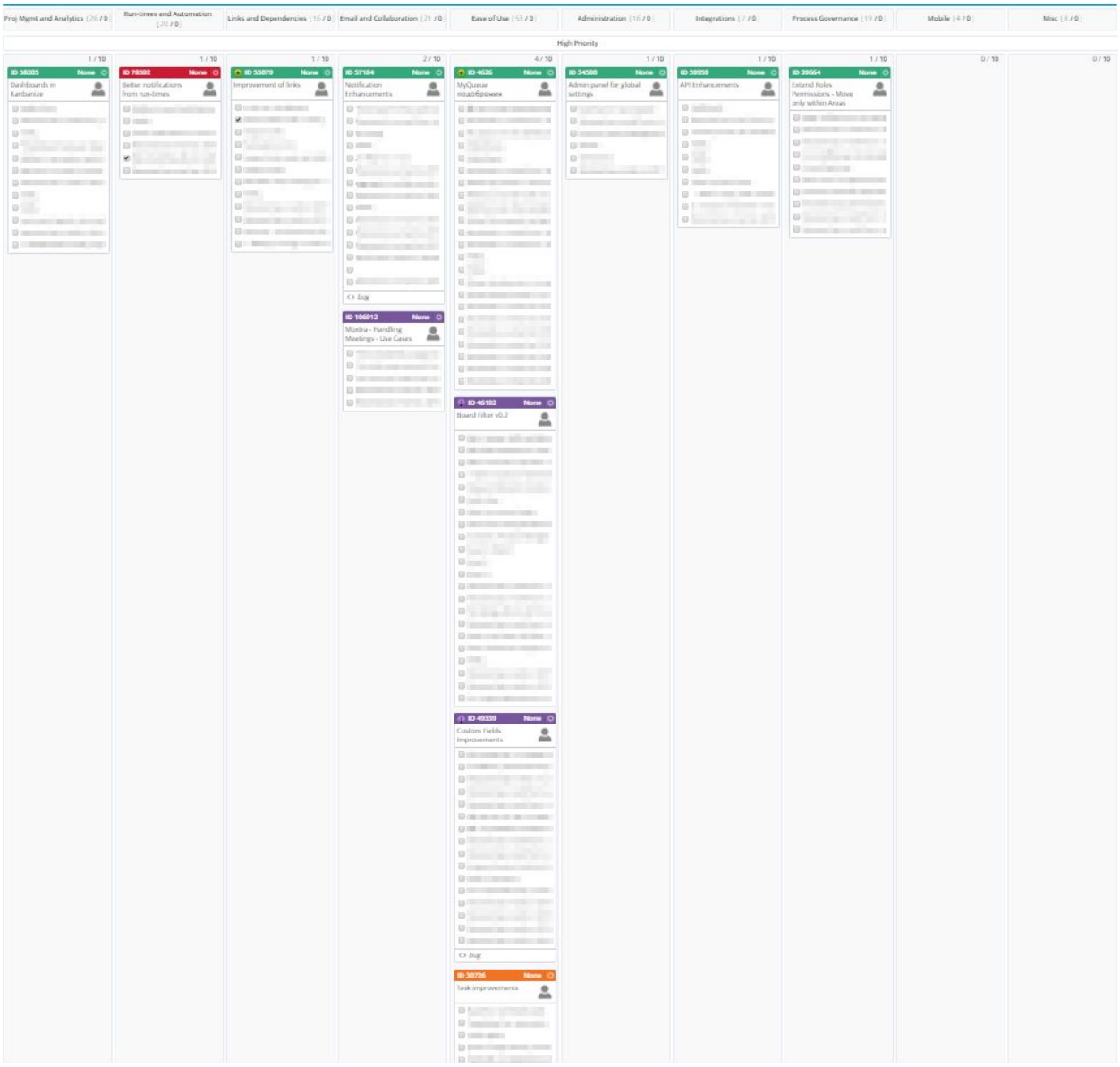
The Product Management Backlog in Kanbanize

Let me ask you a question. Can you point out the area where we get most requests/ideas for? I bet you can. Can you point out the second and the third? I think you can do that quite easily too.

You must be **seeing** the benefit of mapping your backlog in such a **visual** way. The keyword here is seeing, because you can get a lot of meaningful information out of the board, despite the fact that you cannot read a single character. Visualization of work is the first core practice in Kanban and this example is just further proof that visualization is a really powerful tool.

Another effective approach, which is quite specific to the use of Kanbanize, does a really good job for us too. It is a sort of voting system that allows us to immediately see which requests are

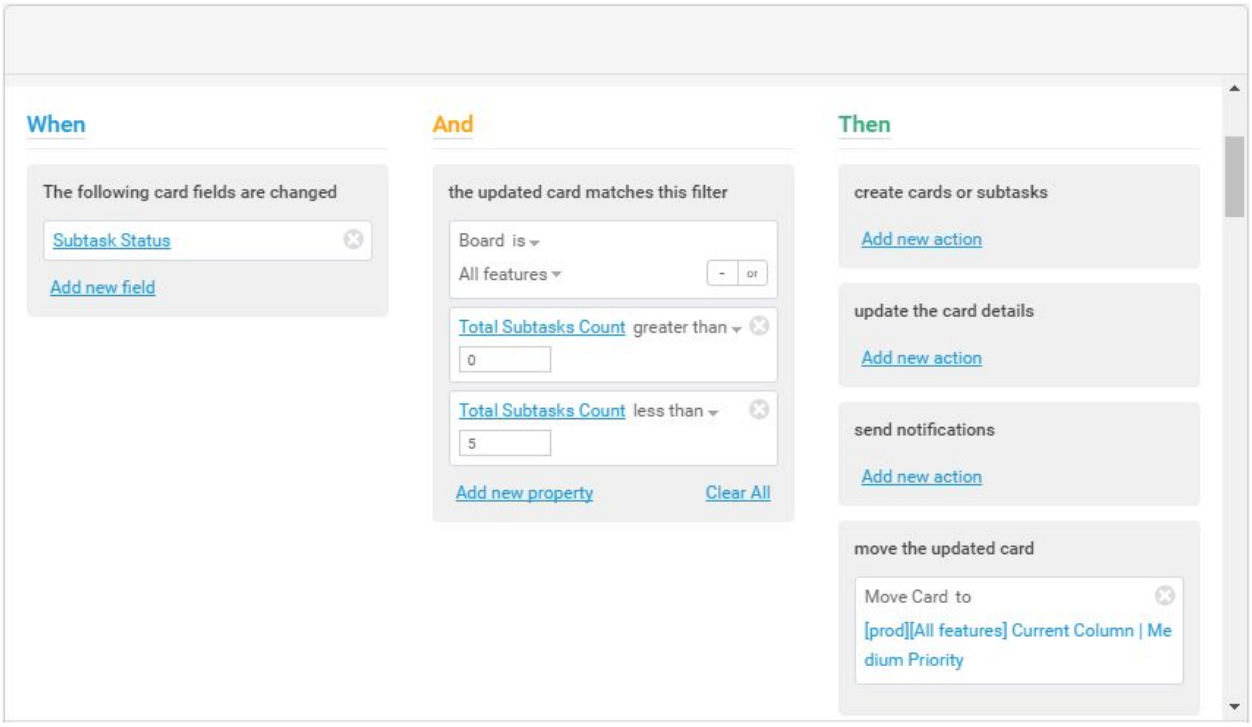
most popular. The realization is really simple - whenever a customer requests a new feature, a new card is created on this board. If the feature has been requested before, this means that such a card already exists on the board and, in such cases, we just create a subtask in that card. In other words, a card with no subtasks has been requested just once, but a card with four subtasks means that the feature has been requested five times in total. Here is how the first swimlane (high priority) of the board looks in reality:



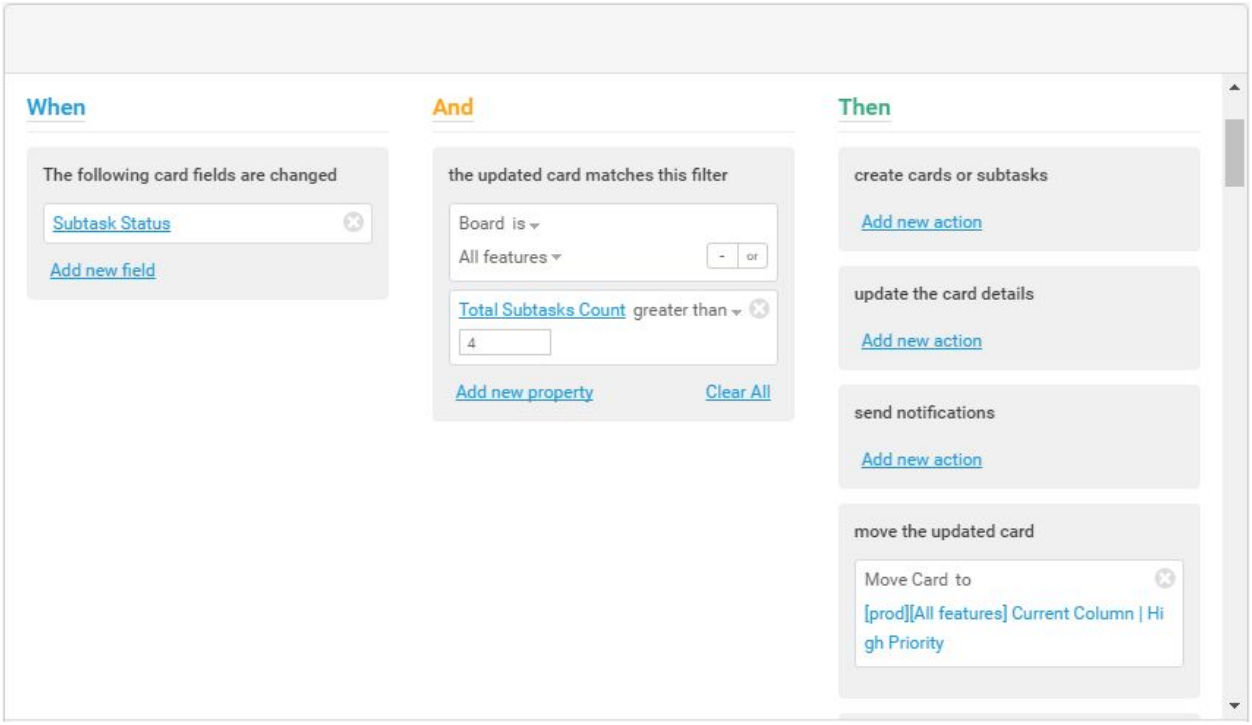
The Product Management Backlog in Kanbanize with Subtask Details

Visualization is, once again, very helpful here (I have obscured the subtasks text, because this is real communication coming from customers). We can very quickly see which of the requests

are most popular by just comparing the height of the different cards. The cards with more subtasks are larger and this directly means that more people have requested this enhancement. Now comes the “sexiest” part. We have a runtime policy which counts the number of subtasks on all cards and if a card has up to 5 subtasks, the policy moves the card automatically to the second swimlane. Another policy is configured to move the cards with 5+ subtasks to the first swimlane (high priority). This is how we automatically maintain a list of the most important features to work on. Here is how the policies look:



Subtasks automation policy in Kanbanize



Subtasks automation policy in Kanbanize

The only thing needed here is the discipline to consistently capture requests as subtasks and attach them to the correct card. It takes some time, but unless you use an automated voting system, there is probably nothing more effective than what is suggested here. We are not big fans of automated voting systems because feedback is only meaningful when you know who gave it. With automatic voting systems you rarely know this information and, at least at Kanbanize, we are hesitant to take decisions based on such feedback.

One potential issue with this approach is the fact that information may be duplicated if more than one person works on such a board. That is why you should always use keywords in the title of the card in order to make it easy to filter. What we always do is filter by a given keyword, check if such feedback already exists and, only then, decide whether a new card should be created or a subtask should be attached to one of the existing cards.

The Feature Management Board

In the previous section, we explored how ideas and feedback can be collected and prioritized in a visual backlog. In the current section, we will see how we actually run the service that feeds the development teams with work.

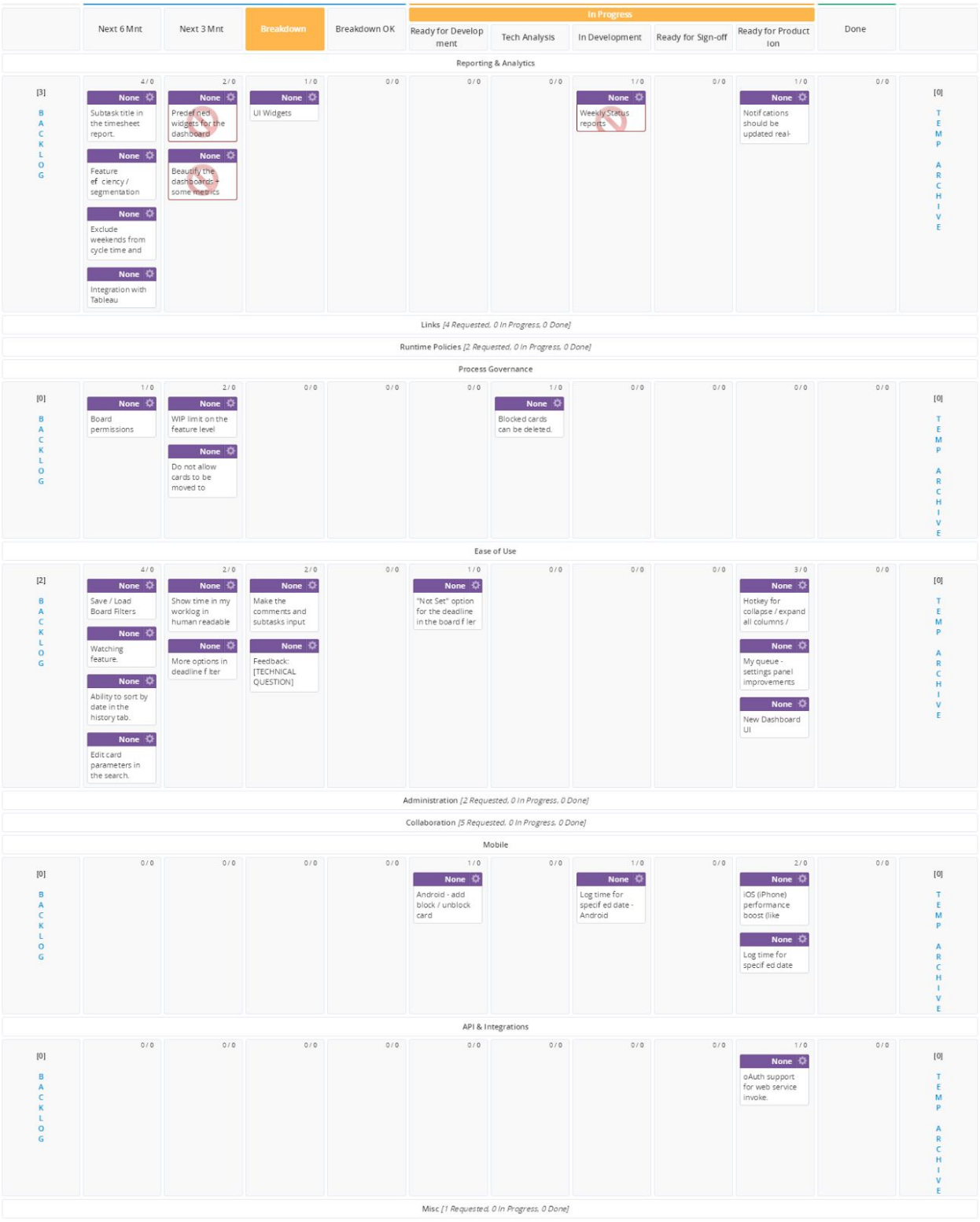
The board where we track the progress of features is called “Kanbanize-features Roadmap”. It is not a roadmap in the real sense of the word, but provides some generic overview about when features can be expected on production. **It is important to mention that this board is a sort**

of MASTER BOARD and it works in combination with other boards. In our particular case, the other board that is involved in the product development process is the board of the development team. Here is a scheme of the roadmap board structure:

Next 6 Mnt	Next 3 Mnt	Breakdown	Breakdown Done	In Progress					Done
				Ready for Dev	Tech Analysis	In Development	Ready for Sign-off	Ready for Production	
Reporting and Analytics									
Links and Dependencies									
Runtime Policies									
Ease of Use									
Administration									
Collaboration									
Mobile									
API & Integrations									
Miscellaneous									

The Kanbanize-features Roadmap Board Scheme in Kanbanize

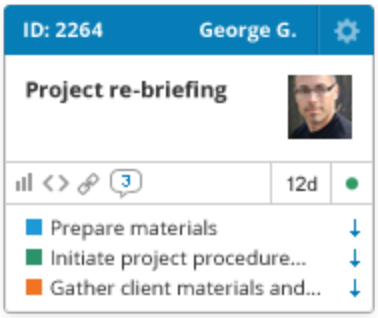
And here is how the actual board looks:



The Kanbanize-features Roadmap Board in Kanbanize

It is obvious that the swimlanes represent the different categories the features can fall into, but it is more interesting to take a look at the columns and their purpose.

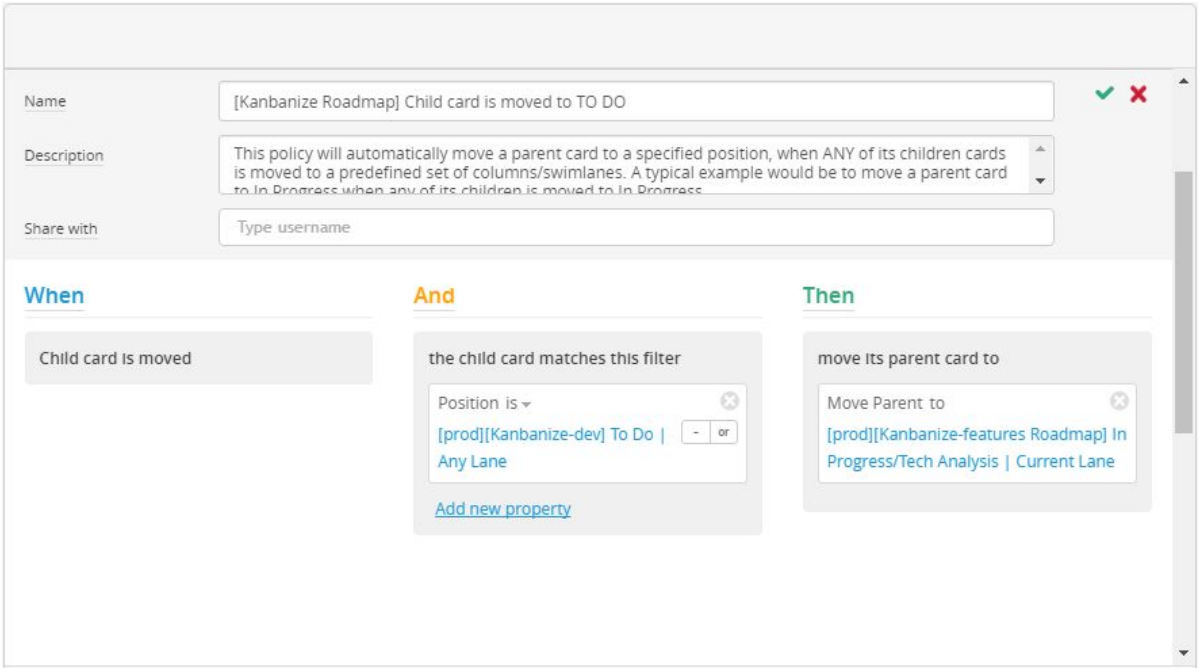
- Next 6 months - these are the items that we plan to tackle in the next six months. This is not considered committed work, but rather the features that are important enough to get out of the backlog. However, they are not important enough to be scheduled for immediate implementation.
- Next 3 months - these are the items that we plan to tackle in the next three months. These are effectively the most important, non-commited features. They have been identified for implementation in the closest future, but no work has been done on them yet.
- Breakdown - this is the first stage where actual work is being done on a feature. The Breakdown column indicates that product management is working to breakdown the feature into tangible user stories. In the context of Kanbanize, this means that the product manager creates separate cards, which are linked as “children” to the feature, which is, in turn, the “parent” of all the user stories. This process usually requires the cooperation of one or more senior members of the actual development team plus the help of a UX/UI expert. If no such help is provided, the product manager might refine work that is practically impossible to deliver or would require too much time. We strongly recommend that you perform the breakdown process in a small working group of at least one product manager, one senior engineer and, if possible, a UX/UI designer.
- Breakdown Done - the breakdown process has been successfully completed and tangible user stories are available. At this stage, the user stories are not present on the actual development board, but are linked as child cards to the parent feature. Here is how a feature with a couple of child user stories looks (the stories are the three rows at the bottom with colored squares in front of them):



A Feature with Three Children User Stories in Kanbanize

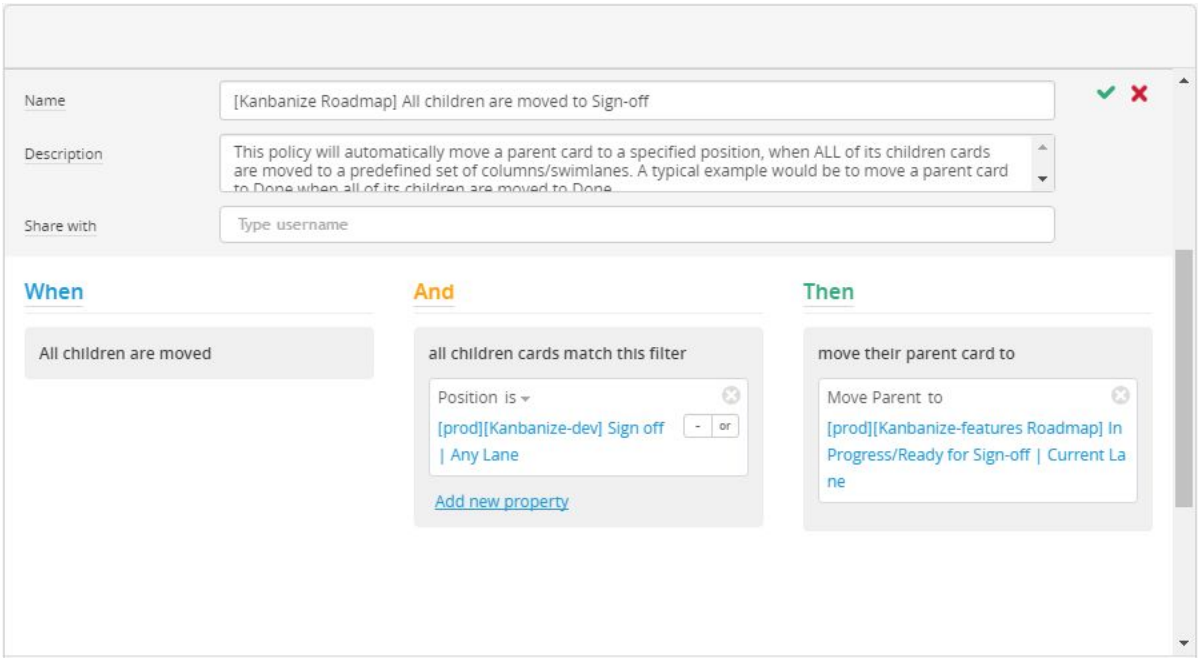
- In Progress / Ready for Dev - this is the first column of the IN PROGRESS area on the board and it is, in fact, the first column where items are considered committed. Whatever enters the Ready for Dev column should be implemented. When a PM moves the card to this column, he or she has to also make the user stories available to the dev team. We are planning an automation piece to help product managers with this action because right now it is manually executed.
- In Progress / Tech Analysis - cards get moved here automatically, due to a pre-configured runtime policy. The policy basically says: “When any of the children cards (user stories) is moved to the In Progress column of the Development board, then move

the parent card to the Tech Analysis column in the Roadmap board. The policy looks like this:



Child Card is Moved Runtime Policy Set Up Panel in Kanbanize

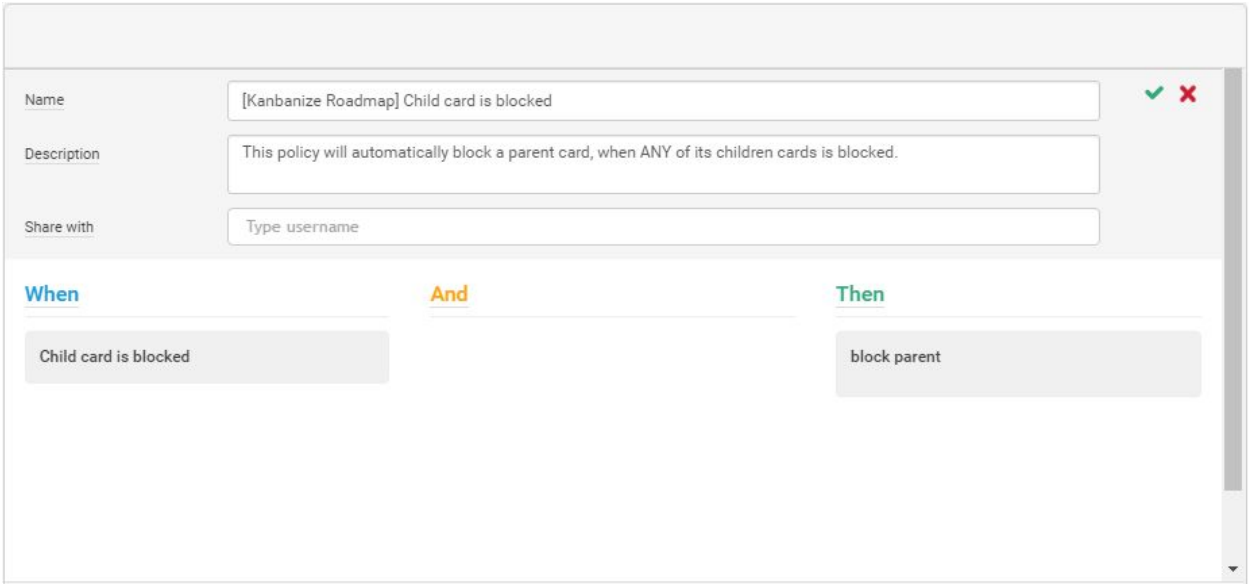
- In Progress / In Development - when the developers start coding any of the user stories of a given feature, the feature is automatically moved to this column. This is again achieved via a runtime policy, which is almost identical to the previous one.
- In Progress / Ready for Sign-off - this is a synchronization point between all children user stories. A product manager should only sign off the feature if all the user stories have reached the corresponding column on the development board. We implement the synchronization with a different runtime policy, which waits all children to reach to a given state on the development board and only then moves the parent feature to Ready for Sign-off on the roadmap board. Here is how the policy looks:



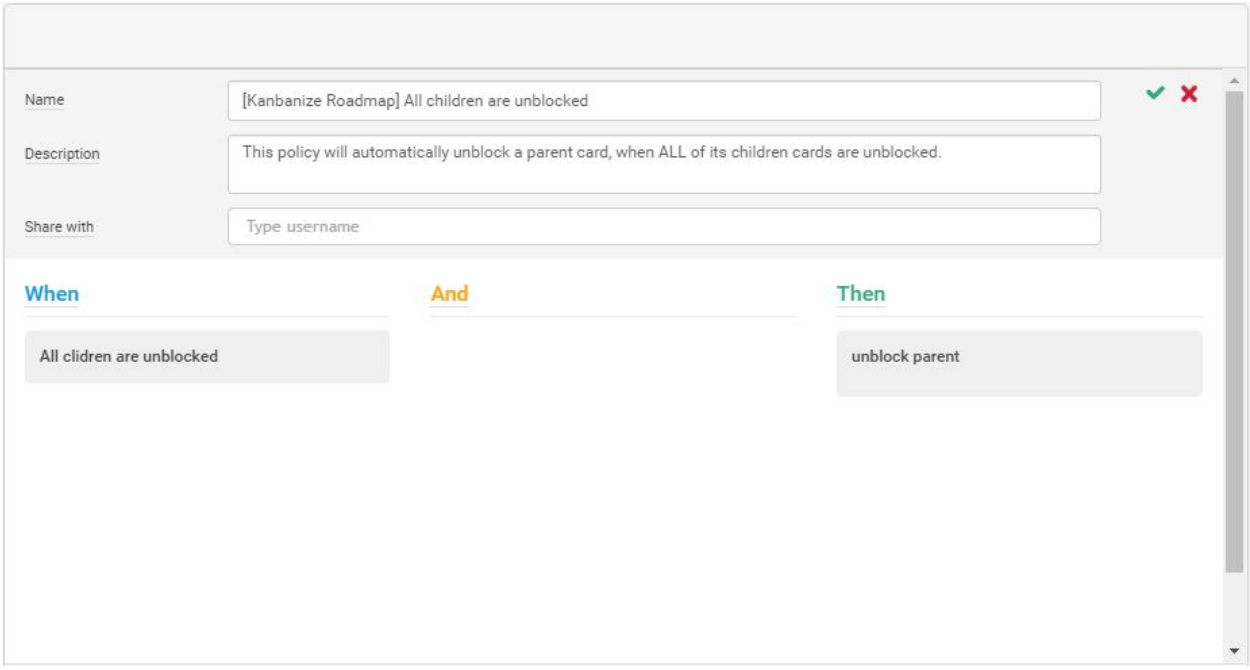
All Children Cards are Moved Runtime Policy in Kanbanize

- In Progress / Ready for Production - similarly to the previous column, when all children cards reach the Ready for Production column on the development board, the parent feature gets moved to this column on the roadmap board.
- Done - identically to the previous column, when all children cards get completed on the dev board, the parent feature gets moved to Done on the roadmap board.

One runtime policy that was not presented above is the “Child Card is Blocked” policy. It basically blocks the parent feature when any of the children user stories is being blocked and unblocks it when the last blocked user story gets unblocked. When this runtime policy is implemented, the system will automatically block or unblock the features on the roadmap board and will track the times the features were blocked. Here is how the two runtime policies look:

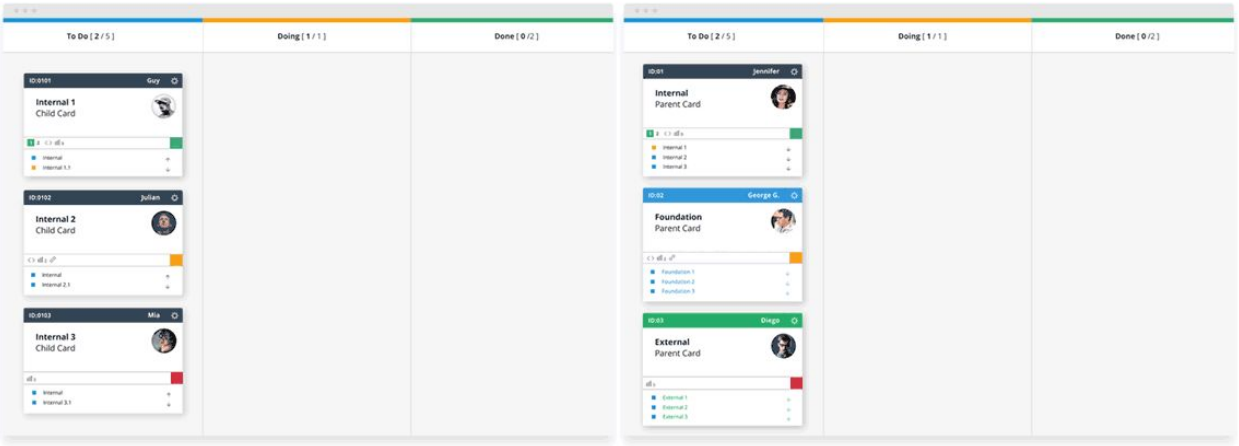


Block card automation policy in Kanbanize



Block card automation policy in Kanbanize

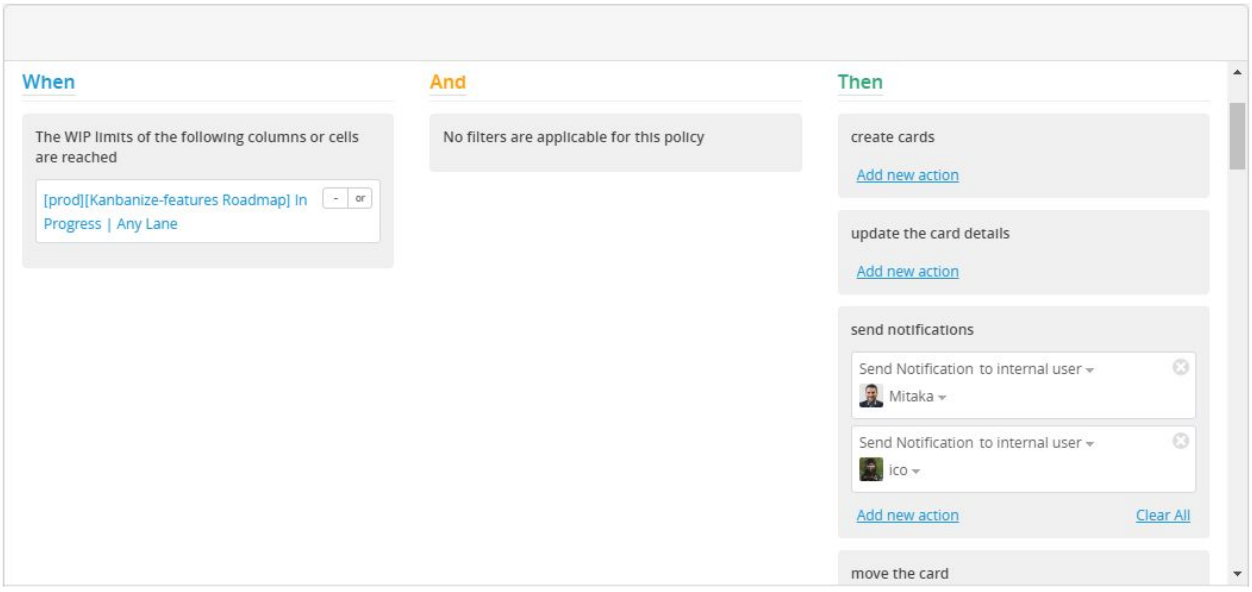
This is a nice animation to visualize how cards are being moved, blocked and unblocked automatically in reality (not visible in the PDF version of the case study):



Portfolio automation policies + block card policies

Managing WIP

One of the greatest benefits of the approach presented above is the ability to manage WIP on the global feature level. If you take a look at the *Figure* above, you will notice that the entire In Progress area is marked with a yellow color. This is because the WIP limit has been reached. When a WIP limit on this board is reached, another policy is triggered and it sends email notifications to myself and the lead of the development team. We can then take actions to make sure that the WIP limit does not get exceeded and therefore keep our Cycle Times in control. The policy looks like this:



WIP Limit is Reached Runtime Policy in Kanbanize

The real brilliance of this approach shines when an organization has multiple development teams working on different user stories, but the user stories are part of the same feature. What happens in reality, when no such system is implemented, is that teams work on user stories completely detached from one another, which results in a lot of started features and the lack of completed ones. Implementing the WIP limit on the global level prevents you from this situation and makes the entire organization much more efficient.

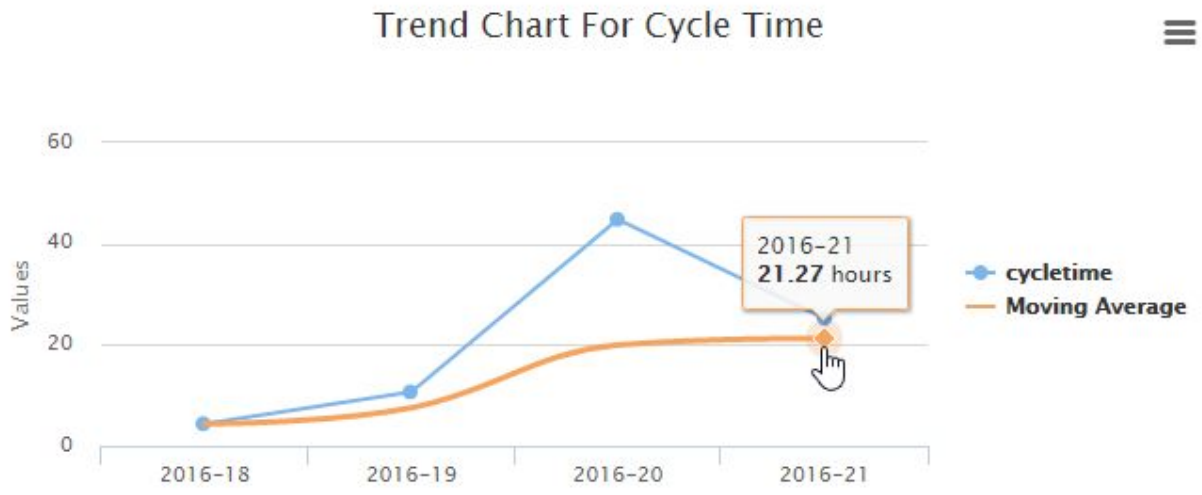
In the near future, we will be implementing an enhancement that will communicate to the developer that if she starts a given story then the parent feature will also be started and this will exceed the limit on the features board. This addition will free the PMs from the task to constantly monitor WIP limits and will actually help the development teams with timely feedback.

Metrics

Improvement is only possible through controlled experimentation. However, experimenting would not be possible unless we have established metrics to support our decisions. Having the right metrics out of the box is one of the key benefits of having an electronic Kanban system or simply said Kanban Software. Kanbanize does that for you behind the scenes and thus makes your lives much easier. Here are some key metrics that you should be monitoring:

Cycle Time Trends

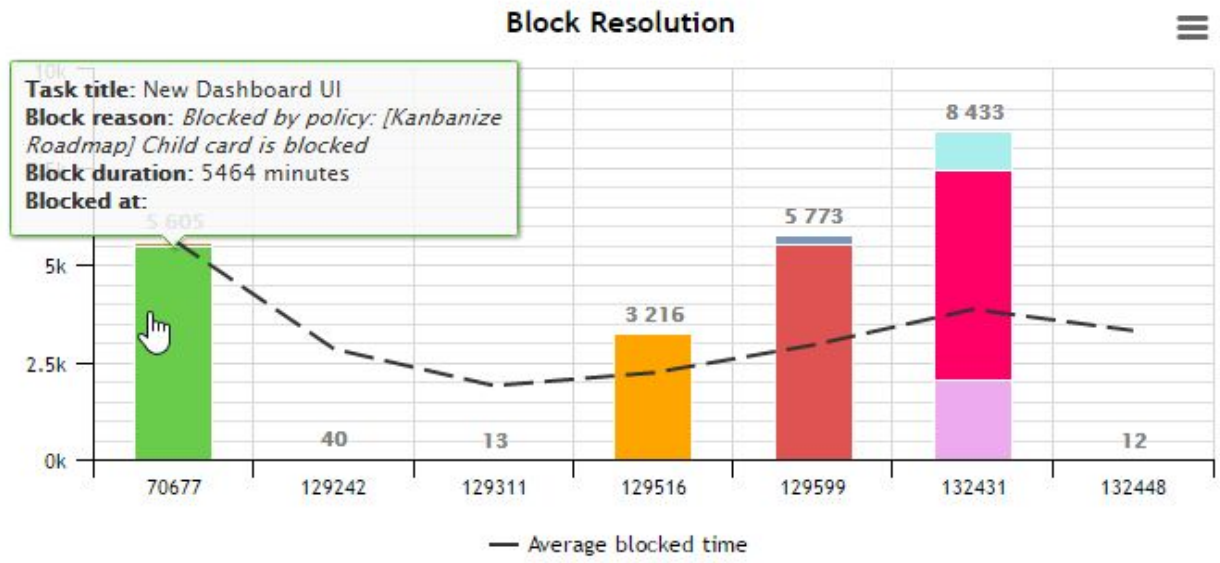
In Kanbanize, we have a chart that directly shows you the cycle time trends for a given board (or part of a board). Simply said, if your cycle time is growing and there has not been any significant change in your team or process, then you are doing something wrong. On the contrary, if under stable conditions the cycle time is going down, then you are doing things right and you should keep it up. Here is how the cycle time trends chart looks like:



Cycle Time Trends Chart in Kanbanize

Block Resolution Time

Another important metric is the Block Resolution Time (how much time the cards on the board have been in the blocked state). Using the Block Resolution Time chart, you can easily review how much time that was and directly get the measurable delay that could have been avoided (block time means **waste**).

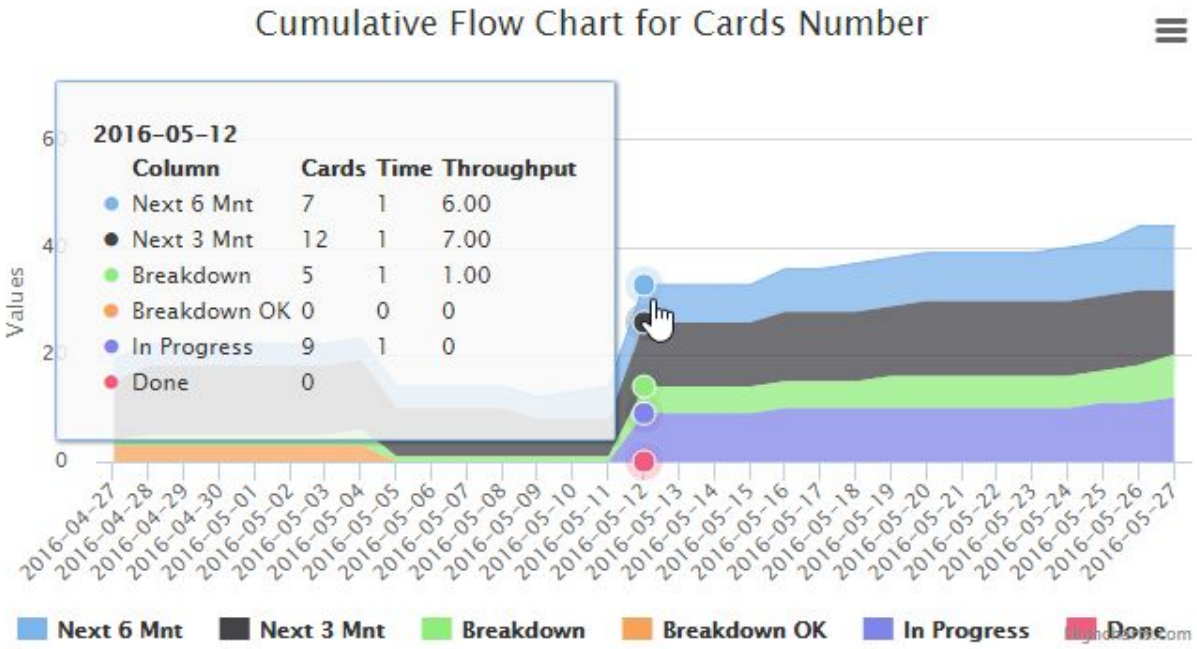


Block Resolution Time Chart in Kanbanize

Cumulative Feature Flow

The cumulative flow diagram (CFD) is one of the most powerful tools in Lean and Kanban. It visualizes the arrival rate of new requests, the levels of work in progress and the departure rate (throughput). We have a really good article on the Kanbanize Blog that explains how this chart works in details. I recommend that you check it here.

One of the disadvantages of the cumulative flow diagram, in general, is that it only cares about the number of items, irrespective of their size. This means that, if we complete ten small features that bring very little value to the customer, it would look ten times better compared to one feature that is ten times larger than each of the smaller features. That is why the 'Flows' chart in Kanbanize allows you to draw the CFD not just by using the number of cards, but also the card size (or any other numeric field). Apart from that, the chart can show you what the average WIP, Cycle Time and Throughput were at any given point in time by hovering over the specific part of the chart you're interested in.



Cumulative Flow Diagram (CFD) in Kanbanize

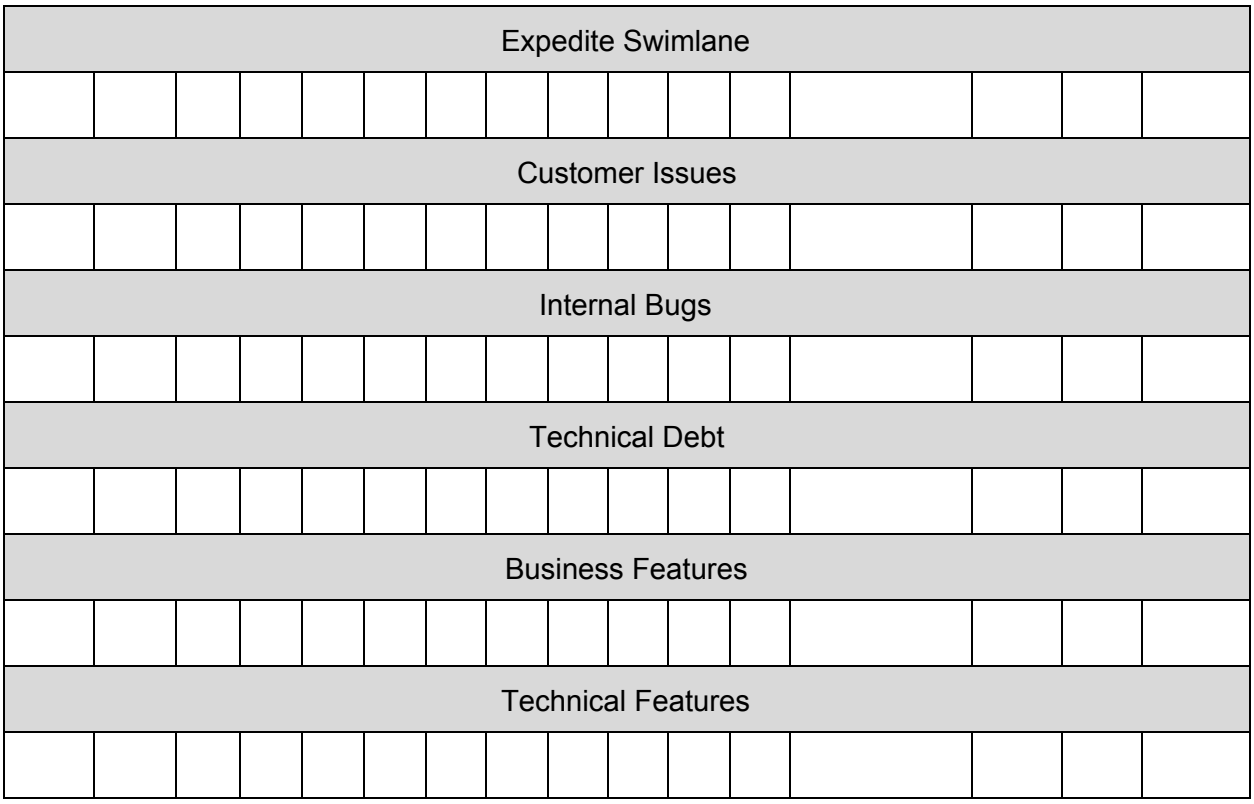
Product Development

The next service that we are going to talk about is product development. This is where ideas turn into reality. As Jock Busuttil says in his book [The Practitioner's Guide To Product Management](#) *"Developers are miracle workers. They turn a product vision into reality and make it look easy."*

It may look easy to the outside world, but everyone who's been involved with development knows that this is a really hard thing to do, especially with limited time and resources. I dare to say that we have a brilliant team at Kanbanize and these guys really do miracles to make things happen. I have shared some of the achievements that we feel quite proud of in the very beginning of this paper. Make sure to check them in case you have skipped that part.

One of the tools that the team uses on a daily basis, without which we would have been far from this achievement, is of course the "Kanbanize-dev" Kanban board. This is where value is born in our company. The board looks like this (some of the column names are coded to save space, explanation below the table):

1. Requested		2. To Do				3. Development				4. Sign Off		5. Ready for Production	6. Production		7. Done
1.1	1.2	2.1	2.2	2.3	2.4	3.1	3.2	3.3	3.4	4.1	4.2		6.1	6.2	



Kanbanize-dev Kanban Board in Kanbanize

Let us go through the columns on the board and explain what each columns stands for:

1. Requested (The next things to tackle for the dev team)
    1.1. Business Requirements - all cards which are being requested by either a customer or a product manager sit in that column. At this point, they are just ideas that are going to be implemented in the “near future”.
    1.2. Ready for Tech Design - this step testifies that the card is a priority now and whoever has capacity should take it and start working on the technical design for implementation.
2. To Do (The highest priority things to work on)
    2.1. Tech Design - in this step, the developers work on the technical design for a given card. They have to figure out what kind of changes or new code has to be developed. In other words, the developers write down what they intend to do when they start coding.
    2.2. Ready for Review - when the developers finish working on the technical design, they put the card in this column. This signals to a senior team member, that the design of the card has to be reviewed.
    2.3. Review - a senior developer pulls the card to this column when she starts working on the actual review. If the review passes successfully, the card is moved to the next column. If not, the card is blocked and the author of the design

is notified that they have to pull the card back to tech design and consider the remarks from the interview.

2.4. Ready for Coding - the technical review was successful and the card can be started. This is the last step up until which a card can be deferred. Each consecutive column contains committed work.

3. Development (Work in progress)
   3.1. Coding - the card sits here while the developer works on implementing the code.
   3.2. Testing - the card is moved here when the developer has finished working on the feature and starts writing unit or automation tests.
   3.3. Ready for Code Review - when the developer has finished implementing the feature or bug, when the tests have been written (and have passed successfully), then the card is pushed for a code review.
   3.4. Code Review - a senior team member is reviewing the code of the new feature. If the changes are good, the card is moved to the next column. If not, the card is blocked and the author of the code is notified to fix the issues.
4. Sign off (The developers believe the feature is ready and no known issues exist)
   4.1. Ready for Sign off - a successful code review leads to this column. If the card being worked on represents a feature, product management is notified (via the runtime policies explained in the previous section). If it is a bug, a senior team member performs the sign-off operation to certify that the code can go to production.
   4.2. Signing off - a product manager reviews the feature. If it is successful, the product manager notifies the developer that the story can continue to production. Otherwise, the product manager creates subtasks in the user story and blocks it. It is up to the developer to unblock it and pull it back to coding, so that the necessary changes are implemented.
5. Ready for Production - a column that contains all cards that can be deployed to production during the next release cycle.
6. Production (the new code is on production)
   6.1. To be Tested on Production - when a release is pushed out to the public, all cards from the previous column are moved in batch to this column. Then, automated and manual tests are executed to make sure that all changes are available on production and that no regression in functionality has been introduced.
   6.2. Ready to Archive - the code has been successfully tested on production, the card can be archived.
7. Done - this column contains cards that are completed, but that do not lead to changes in the code base (non-reproducible issues, research stories, etc.)

Now, let us talk a bit about the swimlanes too. It is critical to understand that, in our case, the top swimlane means highest priority, the bottom swimlane means lowest priority. A short explanation is below, but definitely go through the details afterwards:

- Expedite – Super urgent cards. Everyone helps to get the item expedited. Usually a critical customer defect, security related issue, etc.
- Customer Issues – Defects reported by customers.
- Bugs – Internal bugs found by somebody on the team
- Technical Debt – Things that we should have done
- Customer / Business Features – New features to be introduced in Kanbanize
- Technical Features – Usually DevOps tasks, deployment-related activities, database schema changes, etc.

More details about the swimlanes

Expedite is pretty much the most critical area on the board. In fact, it is treated with such priority, that everyone stops doing what they're doing and helps to get the issue expedited. A sample expedite item would be a production-down situation, significant performance degradation, a hacker's attack, key customer escalation, board-level escalation, etc. What is special about this swimlane is that we allow it to exceed the WIP limits, because these are almost always issues outside of our control and we cannot predict when and how many of them will occur. However, whenever we get an item to expedite, there's always a root-cause analysis to reflect on why things happened the way they happened. Expediting things all the time is a terrible idea and you should do your best to control the amount of work that flows through this swimlane. Most of the time this swimlane should be empty. We have expedited features only a few times in our history, only when we had a really good reason. We strongly advise against that practice. The only acceptable form of expedite work is critical issues.

Customer Issues are the real health indicator of your product or service. If you get huge amounts of issues piling up in the "Requested" area of your Kanban board, you have a very urgent problem to solve. Never, ever allow that to happen, otherwise you'll soon be in serious trouble. Customer issues are the single most important thing on your Kanban board no matter how minor they are. If someone cared enough to report them to you, you must fix them as fast as you can. Yes, sometimes you have to make compromises with issues like "This image should be one pixel to the left", but generally speaking, customer issues must be #1 priority for your team.

Bugs (internal issues) are not as critical as customer ones, but that's a trap that most of the development organizations fall into. There's a widespread saying "If the customer hasn't seen it, it doesn't exist", which is not the preferred line of thought as far as we're concerned. Usually, you could divide internal bugs into two categories: old ones sitting there for ages and new ones that we've created a few days back. While you could compromise on the former, you MUST hunt down and relentlessly fix the latter. That's how you achieve a big win. First, when you stop piling up issues, they don't grow old and you control the size of the backlog for open issues. Second, at some point in time, the old issues will be too old to fix, which would give you the holy right to simply discard them. Combined with the fact that you've continuously prevented new issues from aging, it would result in emptying your bugs backlog. Can you imagine zero issues

in your backlog? Can you?

Technical Debt (TD) is a somewhat abstract term. You could Google its official definition, but, to us, this is a flaw in the product design / architecture that would eventually lead to longer cycle times and high complexity in development and support. In fact, some product architectures are so bad that companies simply decide to ditch them and re-create the product from scratch. Monstrous form of waste, but not that uncommon, especially in contexts where the communication between business and technical people is not good. You can think of TD as a mortgage – you skip your monthly payments long enough and then you are suddenly homeless.

Business Features is the fourth swimlane, which makes it less important than the other three above, but it is definitely very, very important. This is where value gets generated and, since we are usually paid to generate value, it is crucial that we flow work through this swimlane in a fast and predictable manner. In fact, the swimlanes above are a sort of investment that we make for the sake of being able to consistently produce value without being swarmed with customer complaints, escalations and firefighting activities in the long run.

Unfortunately, very few people out there understand these priorities and the development teams are usually forced to work on business features while dozens of customer defects are piling up, architecture is getting worse and worse and the complexity and the maintainability of the code are becoming a spaghetti nightmare. When that happens, development becomes slower and the business folks start throwing blame on the developers for being lazy: “How come you were able to produce 10 features last month and now you only produce 4?”.

Being behind schedule forces the business folks to throw new developers on the team, but, as we know, nine mothers cannot give birth to a child in one month. “Amazingly”, results are even worse because new folks on the team always means a delay in the short run and even more technical debt due to the lack of experience or domain knowledge. With the increased levels of technical debt we become even slower and get even more new people on the team, which is an obviously vicious cycle already.

The conclusion is clear: keep your customer issues, internal issues and technical debt under control and you will have enough time to work on new business features in the long run.
It requires a lot of discipline (trust me when I say a lot), but it’s worth getting there and that’s a promise.

Technical Features are lower on the list of priorities, but again, it doesn’t mean they’re unimportant. As a matter of fact, technical features may often make it to the expedite swimlane if we have an automation piece that blocks the release. The main purpose of this swimlane is to visually separate technical features from business features and to allow better tracking of the given work type. In general, you could merge this swimlane with the business features one, but we prefer to have it on its own because the business features are still more important than the technical ones. As a rule of thumb, if you have work items that share the same priority and

abide by the same set of rules when being worked on, grouping them in a separate swimlane is a good idea.

Managing WIP

The board as you see it in the previous paragraph is the result of a four-year evolution. In the beginning, it had just a couple of columns and two swimlanes. We evolved it into this huge thing because of our need to constantly improve and measure things.

One of the things you have probably noticed is that we have a lot of “Ready” columns. These are nothing but queues. Knowing how important these queues are for any development process, we visualize them separately and also measure the cycle time of all cards in these queues. This is where we find a lot of valuable data for future improvements.

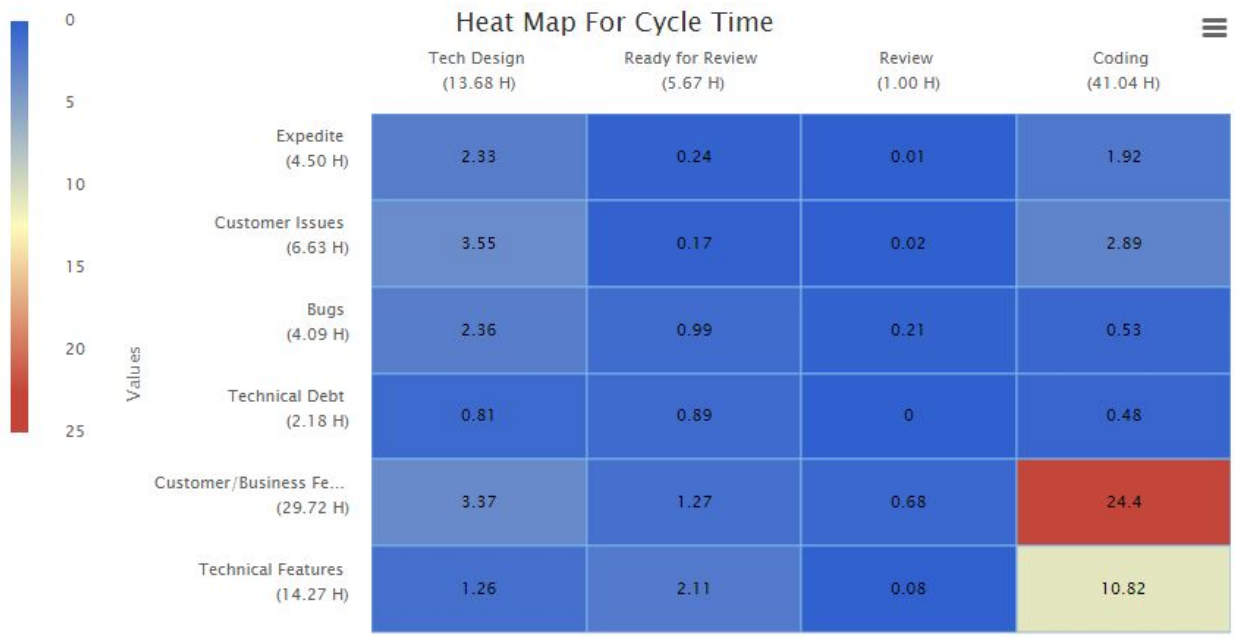
When it comes to limiting work in progress, we have implemented quite an aggressive approach. We have a limit of eight on the TO DO column and ten on the DEVELOPMENT column. This means we only allow one card per developer at a time. As I mentioned before, it requires a lot of discipline to handle such process and it takes some significant time to train everyone how to do it. The good part is that it is also better for the developers themselves to have such limits because they protect them from frequent context switching due to shifting priorities or simply bad management decisions. Once the developers subscribe to the idea, then it is easy.

Metrics

The metrics that we talked about in the previous section (cycle time trends, cumulative flow diagram and block resolution time) are in full power here as well. The key thing to think about, though, is that you need to have the size attribute accounted for when calculating the average cycle time, for example. Some features may be very small, others may be very big. One trick that we use at Kanbanize is to have all features estimated AFTER they have been completed. We just put a number that corresponds to the real scope (sometimes things look quite different in the beginning). Then, we can use this size parameter to measure the cycle time for features of similar size. Some managers are tempted to have all work precisely estimated up-front and this is sometimes required, but if you can avoid that, please do so.

Cycle Time Heat Map

One of the metrics that we use internally, and that is available as a chart in Kanbanize, is the cycle time heat map. It looks like this:



Cycle Time Heat Map Chart in Kanbanize

This chart is a small treasure that reveals which parts of your Kanban board take most of your cycle time, block time or even logged time (if you ask people to log time spent on a task manually). It helps us identify the slowest parts of our processes and focuses our attention to particular areas that we should improve. We use this chart especially to track the time we spend in queues, which, as mentioned before, is the most significant source of waste in a system.

Product Support

A product company like Kanbanize can only prosper if it provides not just good, but excellent support. We are fanatic about support and we truly believe that customers are our number one priority, which, by the way, is also visible from our product development priorities listed above. Our focus on customer support is something we insist on emphasizing. As a testament to how much we believe in its importance, I personally go through each and every support request that gets raised (I don't think too many CEOs would prioritize the same way). I do realize that the CEO of a huge company cannot do that, but I have the luxury to be able to cope with the load and I will continue to do that for as long as I can. I do suspect that I will be able to do that for quite some time, because we maintain very high quality of our product and the support requests are not that many anyway.

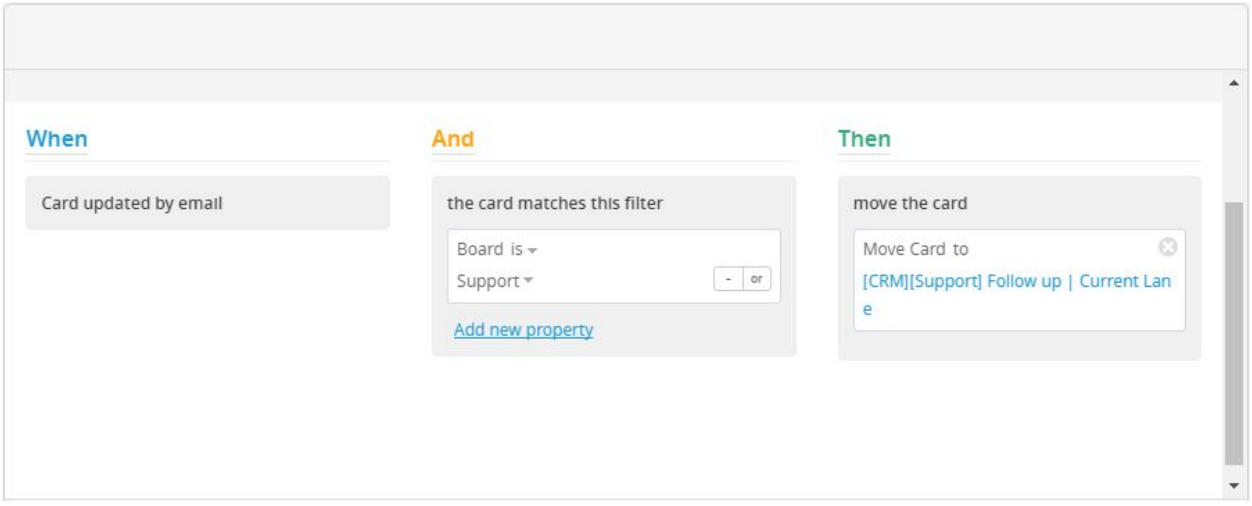
You might be surprised to learn that we do our support with the help of Kanbanize. With the Email Integration possibility, you can basically turn a Kanban board into a ticketing system that scales more than anything you have seen before. Why? Because of the Lean and Kanban principles behind - visualization, which puts any problems on the surface, limiting WIP, which allows us to establish flow through our system and perfection - always doing our best to improve the KPIs that we track. This is how the Support Board looks:

Requested	Follow-up	Working on	To Customer	To Rnd	To CSM	Done
High Priority						
Medium Priority						
Low Priority						

The Structure of the Support Board in Kanbanize

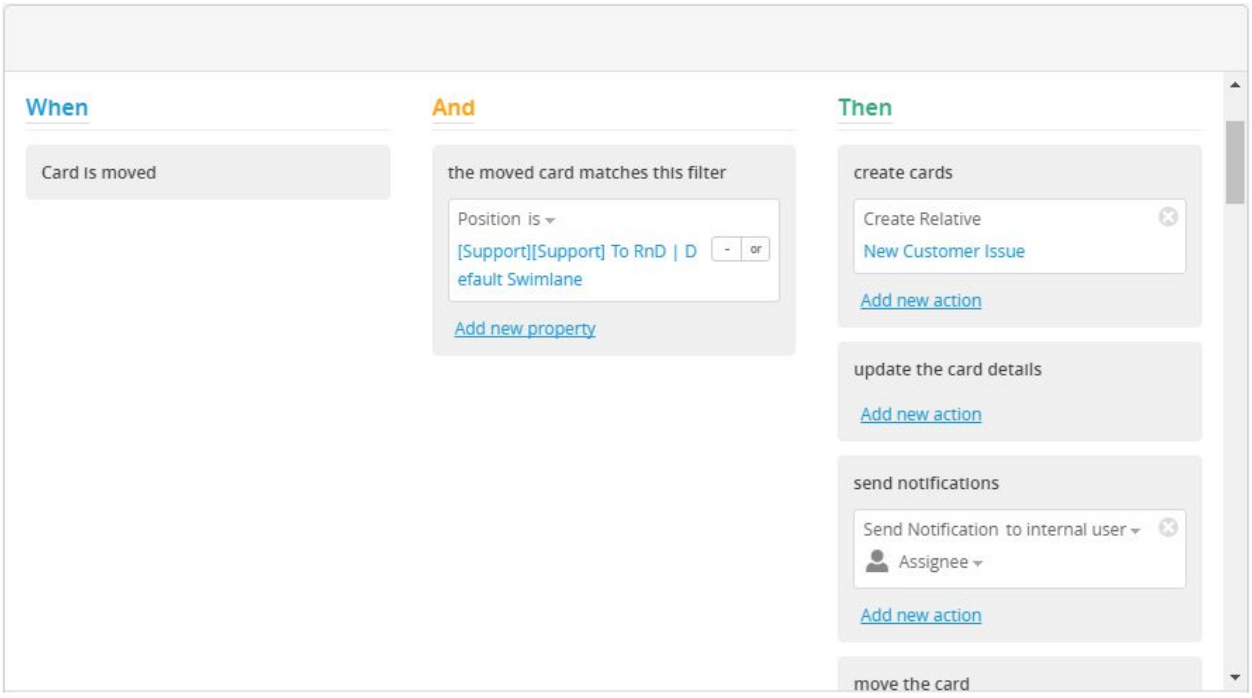
Below is the explanation about the columns:

- Requested - this is where all new cards enter the Kanban board.
- Working on - when a support engineer starts working on the card, the card is moved to this column. What happens in this column is that the support engineer can add a comment to the card, which is automatically sent as an email to the customer.
- Follow-up - when the customer replies, the card is automatically moved to this column. It means that a support engineer has to take care of the card because there is new information. The automation policy that triggers the automatic move looks like this:



Card is Updated by Email Runtime Policy in Kanbanize

- To Customer - sometimes the support guys will need more information from the customer or will wait for a confirmation that an issue has been resolved. When we wait for a reply by the customer, the card is moved to this column.
- To RnD - if a customer has reported an issue and the support people figure out that it is a bug, the card is moved to this column. When that happens, a runtime policy is triggered, a linked card (relative) is created on the Kanbanize-dev board (in the Customer Bugs swimlane) and a notification gets sent to the Assignee of the card. Then, it is the responsibility of the developers to fix the issue and notify the support team by adding a comment to the dedicated card to confirm the issue has been fixed. Then, the support team notifies the customer about the final resolution. Here is how the policy looks:



Card is Moved to Follow-up Runtime Policy in Kanbanize

- To CSM - sometimes a customer might need special attention, which is a task for the customer success team. Following the example above, this column creates a linked card on the CSM board, so that a customer success representative takes care of a certain request.
- Done - The card is completed or, in other words, the ticket is closed. When the cards reach this state, they are manually checked for feature requests. If such exist, they get collected in the “Product Backlog” board, which was the first service that we discussed in this chapter. Finally, cards get archived.

The swimlanes on the board are quite self-explanatory, but let us cover them shortly too:

- High Priority (Critical) - these are cards that require urgent attention. When a high-priority email is sent to the support alias, the card is directly created with high priority. Then, a runtime policy moves it to the top-most swimlane.
- Medium Priority - by default, all cards are created in this swimlane.
- Low Priority - if a low importance email is sent to the support alias, the card is created with a low priority. Then a runtime policy moves it to this swimlane.

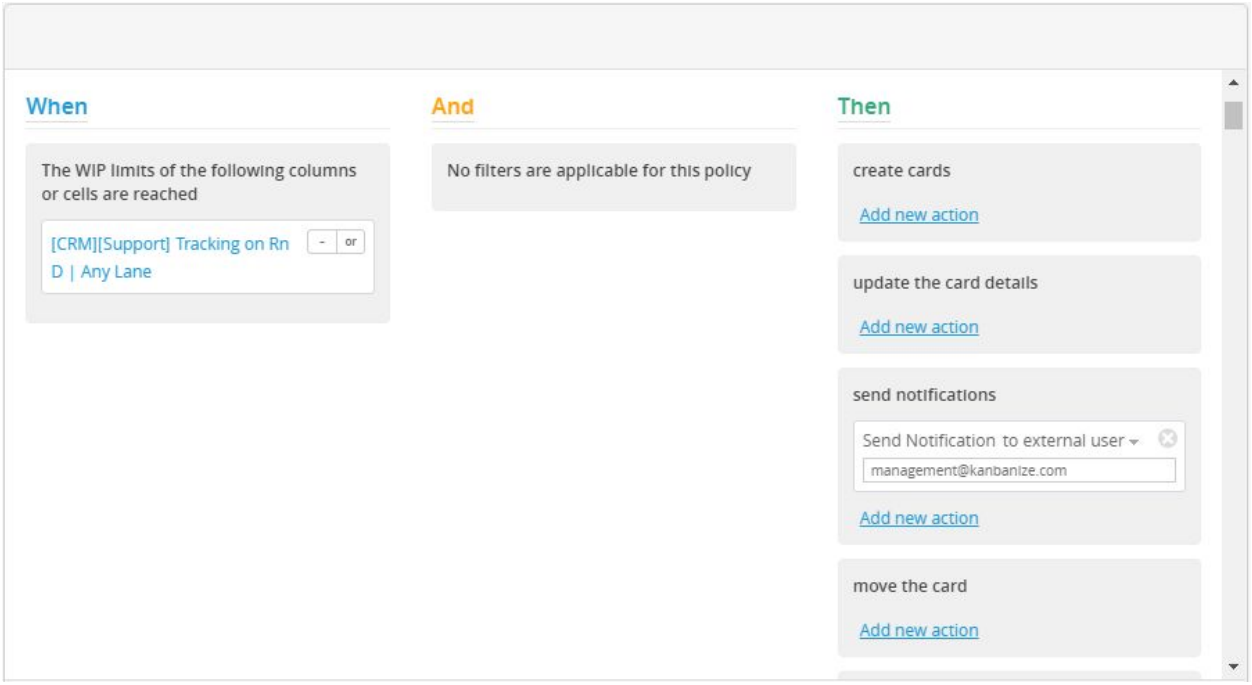
Managing WIP

With services like product support, it is a challenge to maintain strict work in progress limits, just because you do not have full control over the process. However, this does not mean that you should not put any limits at all. For example, we do have WIP limits on our support board as follows:

- Working on - 5
- To RnD - 10
- To CSM - 10
- To Customer - 10

Here is the explanation of why we have decided to apply these limits:

Working on should be limited, just because we want the support engineers to be as efficient as possible.
To RnD is limited to 10, which is a number that signals a problem. If we ever get 10 issues referred to RnD, this means a big problem in our organization. If that happens, a runtime policy triggers and sends a warning to the management of the company that customer issues are going up and nobody in the entire team wants that to happen (not because of fear, but because everyone understands that this means waste and unhappy customers). The valuable runtime policy that makes this possible looks like this:

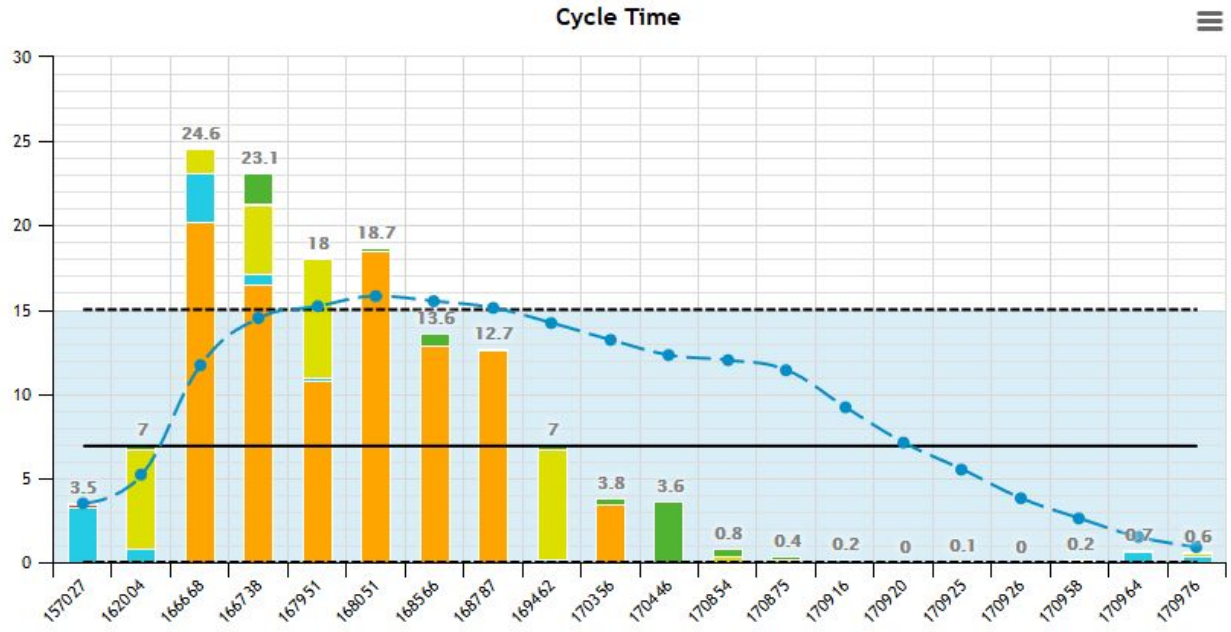


WIP Limit is Reached Runtime Policy in Kanbanize

To CSM is limited as well, similarly to the RnD team column, in order to make it possible to track the dependencies there.
To Customer is limited because we want to have issues resolved and not forgotten. When we have a limit on that column, the support guys are somewhat forced to work proactively with the customers even if they have not replied to the original question. A friendly reminder is always welcome, isn't it?

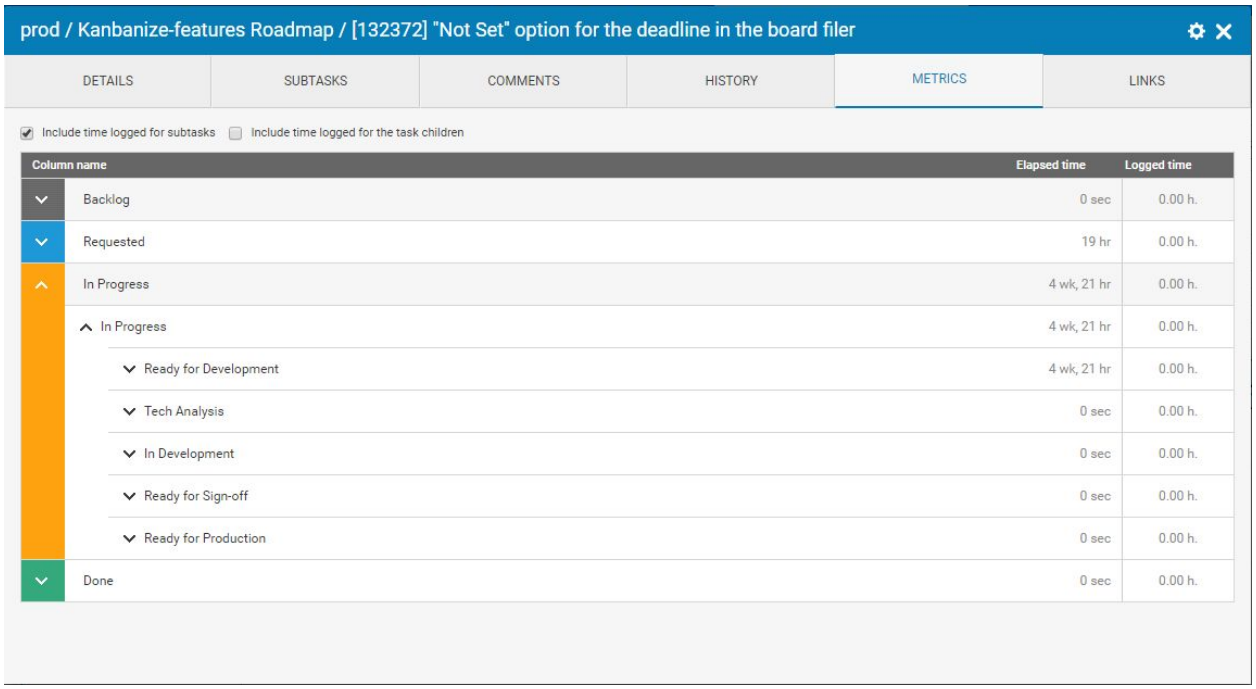
Metrics

The three most significant metrics for this service are the overall Cycle Time, the initial response time and response time for the Follow-up column. All these can be tracked using the charts that we discussed above with one addition - the Cycle Time per Card Chart. It shows the cycle time for each and every card on the board and it looks like this:



Cycle Time Chart in Kanbanize

The nice thing about this chart is that you can very quickly see which cards have taken the most time and, also, where that time has been spent. Then, if you need to investigate further, you would just click on the bar representing the card and the card's details will open. Having the HISTORY and METRICS tab on the card, you can check where the time has been spent by column:



The Metrics Tab of a Card in Kanbanize

Automation QA

Automation QA is the last service that gets attributed to the product development cycle at Kanbanize. This is a central and very important step in the process. It ensures that the product covers all major scenarios and integrations.

In the past, we had the QA guys testing everything that was going through the development board, but we found out that there were two major drawbacks to this approach:

- There was not enough QA power to test everything on the dev board and at the same time develop new automation tests to increase our test coverage.
- The developers primarily relied on the QA people to test their changes and, as a result, there was too much ping-pong between them.

So, we were basically in the situation that required one QA person to every two or three new developers. We thought we should find a way to encourage our developers to be more careful when passing their code to the next phase. That is how we decided to experiment with the crazy idea to exclude the QA guys from the development process and dedicate them to the creation of hundreds of automation tests. The QA people are currently responsible to review the testing steps that each developer suggests during the Technical Design step. That is how we make sure we get their feedback and verify the corner cases which, usually, only an experienced QA person would consider. In a way, the QA guys act as quality consultants teaching the developers what to test, but they spend most of their time actually work on automation tests.

How was this idea born? There is a great story about the automotive industry, where Mercedes Benz became famous for spending as much time fixing issues with a given car, as was needed to build the car in the first place. This is how they ensured the superior quality for which the brand is famous, but this was definitely not the most cost-effective approach. TOYOTA, on the other hand, had a different philosophy. They were building quality into the product so they wouldn't have to spend any time fixing issues once the car was out of the production line. This is what we did with our development process. We decided to do whatever is necessary to start building quality into the actual development process as to not have to fix issues afterwards.

After this change, the developers became the last possible step where quality could happen. This created stress, of course, but it also gave us the leverage to establish new testing frameworks that made it possible for the developers to write tests on their own. They also started running all tests before they commit their code to the official branch, which is another step to ensure better quality. Overall, the change was very positive and, although it was really risky, we do believe it was (and still is) the right thing to do.

The QA guys are a small development team on their own. Apparently, working on test infrastructure, developing complex automation tests and helping the dev team requires some

significant organization as well. You must have guessed how they make it happen ... with their own Kanban board in Kanbanize. It has the following structure:

Requested		In Progress				Done
To Do	Plan	Coding	Testing	Ready for Review	Review	
Expedite						
Bugs						
Features						
Infrastructure						
Technical Debt						
Misc						

The Structure of the Automation QA Board in Kanbanize

The columns represent the following steps:

- To Do - new cards that are requested from the QA team. At this point, they are not committed work, just things that are identified as important.
- Plan - the QA guys evaluate the task and make a plan of how to execute it. This is a mixture between technical design for the automation tests and a review of this design by the QA lead.
- Coding - the tests are being implemented.
- Testing - to make sure that the new tests work properly by actually running them.
- Ready for Review - the new tests are ready to be reviewed by the QA lead.
- Review - the QA lead is reviewing the tests.
- Done - the card is completed and the test is now part of the official automation suite.

The swimlanes of this board deserve some special attention too:

- Expedite - there are three reasons for cards to appear here:
    - Card from the Dev board that needs a QA review (we already discussed that the QA people at Kanbanize serve as quality consultants for the dev guys)
    - Card from the Dev board is ready to be tested on production (this usually happens after a release and the entire team helps to test all changes on the production servers)
    - There are failed tests from the nightly test suite. When this happens, the QA team investigates the failures with highest priority. If there are issues with the tests, then the failing tests are fixed right away. If a bug in the application is discovered, then an expedite card is created in the Dev board too and the developers fix the bug with highest priority. In other words, we are doing whatever has to be done to have all tests passing again. Only then do we continue with our usual work.
- Bugs - when a bug is "Ready for Production" a card will be created here. Create a test or update an existing test to regression check the scenario. This is our take on continuous improvement - when a bug has been found, we cover it with a test so that we never release it to production again (in the form of a regression).
- Features - planned regression tests for features are placed here.
- Infrastructure - infrastructure changes are tracked here.
- Technical Debt - things that we should have done, but have not been done.
- Misc - items of various types - meetings, manual regression testing, etc.

Metrics

Apart from the standard Cycle Time and Block Time metrics that we have discussed above and which are applicable to the QA board as well, there are additional metrics that the team collects and reviews on a weekly basis.

Successful Nightly Test Execution

Each night, the entire test suite containing more than seven hundred end-to-end and API tests is run. We track how many of the tests pass or fail and what the root cause for the failure was. The data is tracked manually in a table like this one:

Nightly			Tests Count			Number of Failing Tests			
			Total	Skipped	Failed	Bug	App Change	Broken Test	Unstable Test
Day 117	5/27/2016	Friday	721	1	13	3	0	7	3
Day 118	5/28/2016	Saturday	736	1	10	8	0	0	2

Day 119	5/29/2016	Sunday	736	1	20	0	0	20	0
Day 120	5/30/2016	Monday	736	1	22	0	0	22	0
Day 121	5/31/2016	Tuesday	736	0	11	0	0	9	2

Automation QA Metrics for Passing/Failing Test in Kanbanize

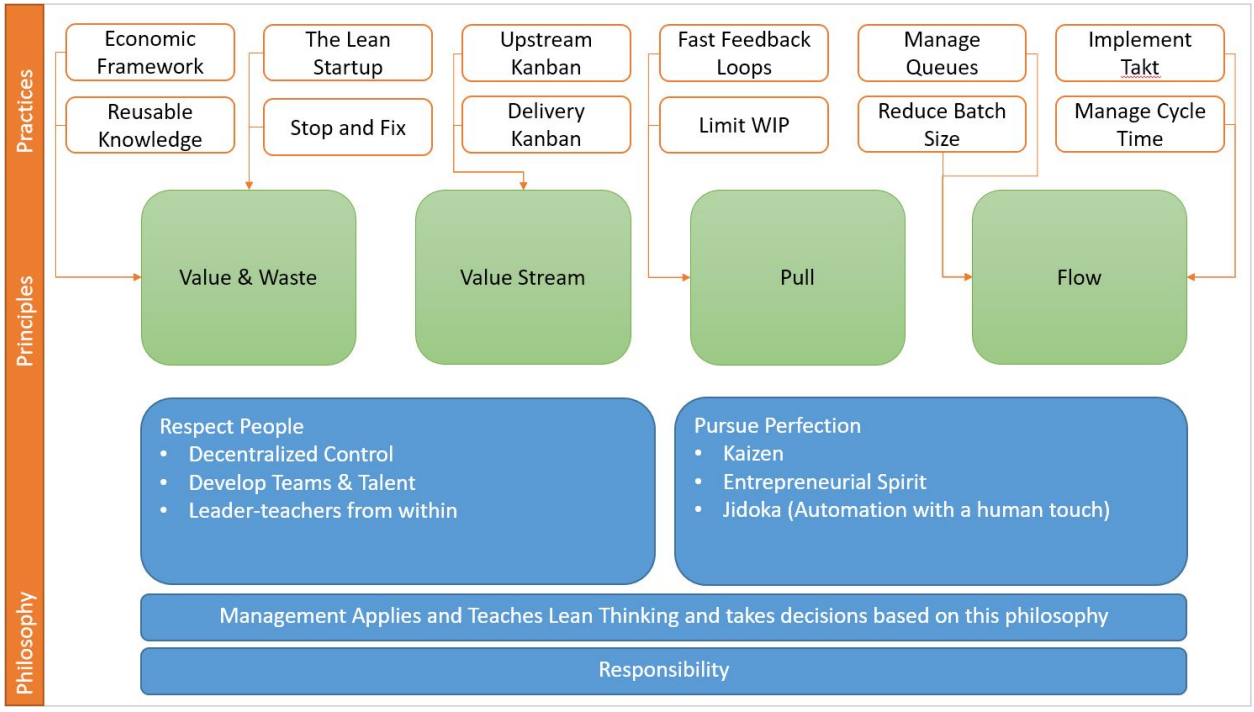
Having this information available, we can measure what the health of the product is on a global level. If we start getting too many failures caused by bugs in the product, then a “warning light” flashes in our heads. This means we are attempting to complete more than what we are actually capable of. This warning basically means “SLOW DOWN”. We review these metrics at each weekly team and strategy meeting and, by that, ensure we can quickly react to such negative trends.

Our Philosophy, Principles and Practices

The case study from the previous chapter represents the mechanics of what we do. However, realizing all this in practice requires a much deeper understanding of the underlying philosophy, principles and practices. This last chapter is dedicated to our view of how software should be created, which is, indeed, a rather complex mixture of different components.

The image below provides a good overview of all the different ingredients that contribute to a healthy, effective and efficient system:

- Our philosophy in blue (at the bottom)
- Our principles in green (in the middle)
- Our practices in orange (at the top)



Discussing each and every box on this graph in details will likely result in a few hundred pages, so we will not go into detail about every single one. However, I would like to cover as much as possible, at least on a basic level. This should give enough food for thought to the inexperienced Lean Thinker, but even the experienced one could learn something valuable.

Philosophy

Responsibility

I have survived two life-changing experiences that completely transformed my views of the world. The first one was a free fall from 10+ meters height, which put me to bed for more than three months. Broken spine, hands, legs… a real beauty. I was really lucky to have lived through it and recover completely. That's how I learned where the thin line between bravery and stupidity was.

The second thing that shook my world was my divorce. It was not a nasty divorce, we just admitted to ourselves that we were not happy and decided to go our own ways. No arguments, no quarrels, no conflicts, we are actually still pretty close friends. That being said, when you are putting an end to a long-lasting relationship, it is natural to ask yourself a lot of questions. Being a truly rational person, I was trying to analyze why things turned out the way they did, but I always reached the point where "I'd done nothing wrong". That's when I got acquainted with the work of Christopher Avery and the "Responsibility Process". The "Responsibility Process" answered all my questions… I would never be able to explain this better than Dr. Avery himself, so I suggest you check this video (if you are reading this on paper, switch to Youtube and search for "The Responsibility Process™"). You will thank me later.

I am not telling you all this as a call for sympathy. If you want to implement Lean, you need to have your organization acting out of responsibility and that's a given. If you lay blame, or tell good stories, things will not work. However, if you do this right, you will succeed with or without Lean. Lean will just make things happen faster.

Management Applies and Teaches Lean

It is somewhat hard to admit that, but it took us more than four years to realize this. In the beginning, we were only team of a couple of people and we did not need any special conditions to do what we were supposed to do. Then, we started to grow and many new people joined the company. That's when things started to crack. The processes were no longer working, we were getting worse results than before and it looked like we were speaking different languages with the people with whom we were supposed to be on the same page. Moreover, we did not have time to do what we were doing before, which hurt the overall performance of the company.

Trying to figure out how to improve this situation I read a really good book - The E-Myth Revisited by Michael Gerber. He emphasized the importance of training and how it affected the ability of organizations to scale. That's when the solution just occurred to me - we needed to teach Lean to the people with whom we work and hire in the future. At that point in time, this seemed so obvious that I couldn't go beyond the "shame" zone for a while.

We started a series of workshops within and outside the office with the sole goal of teaching what we had learned. To this day, these teachings are being received really well and it is great how people engage with the new things they are learning. Our discussions changed, the arguments became backed up by the right theory and principles. The overall situation became, and still is, much better compared to what we used to have before.

Right now, we are in the process of transforming the company into a "learning machine" and we do realize that the role of the leaders is a major one. Teaching and learning has proven to be the right way for us so far and it will most probably work for you in the long-run as well.

Respect People

This might sound like stating the obvious, but managers or business owners tend to forget that all great things were made by somebody. Innovation doesn't just happen - it is created by people. We should train them and then trust them to do the right thing. We must treat everyone as if they were a volunteer who came to help us. Treat your colleagues as equals. Be friends. They will respond with their best.

Decentralized Control

If I asked you to think about a structure with a super strict hierarchy, following orders without ever questioning them, I bet you'd think about the military. Historically, most of the battles were fought by commanders who instructed their troops what to do and when to do it. But as the German strategist Helmuth von Moltke said: "No battle plan survives contact with the enemy." Suffering severe casualties, the military had to re-think their ways. They realized that they needed to only provide a global plan and then leave it to the troops to take the smaller, but still

important, decisions along the way. That's how the US marines or any similarly structured organizations were created. You get extremely well-trained and capable people, give them an objective, define the constraints and let them do their job.

This is what businesses can and should do as well. They need to make sure that everyone is qualified for their job and then create a productive environment to allow the rest to happen. One way that companies do this is by establishing a few very simple rules. I remember reading another great book on that topic - Made to Stick by Chip and Dan Heath. The authors dissected why some ideas survive and others die. I remember an example of an airline that enabled thousands of employees to make their own decisions by giving them one simple rule: "We want to be the low-cost flight company.". From then on, everyone could tell if a given decision would bring the company closer to its goal or not and acted accordingly.

Develop Teams & Talent

We will talk a lot about knowledge and how we generate it in our company. That is why I'd rather keep this section pretty short. Just remember that you need to make sure everyone in the organization learns to do their job better. If they stop learning, your organization is aging.

Leader-teachers from within

The only way an organization can scale effectively is through strong leaders, who express the behaviors expected by the company culture. If a company fails to create the leaders it needs, sooner or later, it will be in great danger. It happened even to Toyota, when they grew their operations outside of Japan - they just couldn't create leaders fast enough and that resulted in some major messes.

We believe that people, and especially leaders, should try very hard. We do not work with "sissies" and, on the rare occasion of having hired one by mistake, we let him/her go really fast. As a famous woman on Youtube once said, "Ain't Nobody Got Time For That". The question here is how you make people "try hard" and like doing that they're doing it. Quite honestly, we don't have a complete answer to that question quite yet, but we will figure it out. For the moment, I can only suggest that you check Dan Pink's Ted talk on motivation. It is a true eye-opener.

One thing is certain, though. We would only hire a leader from the outside if we need some exceptional talent, that we have not yet managed to breed internally. We strongly believe in this approach and we are prepared to wait patiently for the time when we will be reaping the benefits of this long-term strategy.

Pursue Perfection

Kaizen

Kaizen is a Japanese word used to represent the culture of continuous improvement. We are really good at this and that's what makes things possible around here. We don't tolerate

inefficiency and wasteful activities. We pursue and eliminate them relentlessly. This is possible through great commitment from the leaders of the organization and the people who support them. The engine for this is, of course, the responsibility process I referenced earlier, because, quite often, the root cause of a problem is a person. If that person acts out of blame or justification, things wouldn't work. Everyone in the company is (or will soon be) proficient in the responsibility process and has granted permission to others to tell him or her when they are not above the line of responsibility.

Each team has adopted a data-driven approach to improvement. We collect metrics and analyze them during our weekly meetings. Apart from that, we hold a weekly all-hands meeting where we discuss potential improvements.

As it is with the responsibility process, if you get this right, you will sooner or later succeed. This is a critical part of the adoption of Lean and, unless you get everyone to adopt this idea, your efforts might be doomed to fail.

Entrepreneurial Spirit

Working with brilliant people is a must if you want to achieve long-term success. However, brilliant people have special needs. At some point in time, they want to create something on their own. They want to have their ideas heard and see them come to life. That is why we welcome entrepreneurial spirit at the company. We love it when somebody has a good idea and wants to work to make it happen. To be honest, I wish we had more of this, but we are making some steps to ensure that the energized people have the time to think about innovation. I guess it's a matter of time to have our own 20% free time for cool ideas, we just need to figure out how exactly we want to do it.

Jidoka (Automation with a human touch)

The Jidoka concept was invented by Sakichi Toyoda, the Founder of the Toyota Group. He figured out how to stop the automatic looms as soon as a defect was introduced. This simple invention allowed Toyota to separate the human from the machine and have just a single worker manage many machines. Not only that, but the wastes from defective goods were greatly minimized. These two contributed to huge economic improvements and made Toyota a much more profitable company.

At this point, you may understand why we are investing so much work in the run-time policies (business rules) module in Kanbanize. They are our Jidoka - they allow us to detach the human from the machine and have the machine report to the human, when it needs attention. This is a really powerful concept, which is considered one of the pillars of the TPS house.

Principles

Value & Waste

This might be a blunt statement to throw at you, but making money is a necessary goal for every business out there. Feeding the starving children around the planet, preventing global warming and colonizing Mars are noble causes (no irony intended), and many corporations are notable activists, but in the end, they have to make money to support what they believe in. Money comes from the customer and the customer pays for value that they get from a product or a service. Therefore, the goal of a Lean business should be to generate more value for the customer without increasing the cost for the generation of this value.

At first, this may sound like an oxymoron, but this is, in fact, what Taiichi Ohno did in TOYOTA in the second half of the 20th century. This is what James Womack says about him in his book "Lean Thinking" - "*Taiichi Ohno (1912– 1990), the Toyota executive who was the most ferocious foe of **waste** human history has produced*". Taiichi Ohno was devoted to eliminating **waste** (*muda*) from the production process in his company, because he realized that this is how more value could be generated for the customer. He was able to recognize multiple activities in the design, development and production processes, the removal of which could improve the overall economics of the company. Working relentlessly to remove various types of waste, he identified the seven types of *muda*:

- Defects
- Overproduction
- Over-processing
- Motion
- Waiting
- Transportation
- Inventory

The table below shows how these seven types of waste map to the production process in a factory and to the development process in a software company:

Waste	**Production (Factory)**	**Software Development**
Defects	Broken parts	Bugs
Overproduction	Too many parts	Features that nobody uses
Over-processing	Spending a lot of time on a given task	Unnecessary complex algorithms solving simple problems

Motion	Unnecessary movement of the worker	Unnecessary meetings, procrastination, extra effort to find information
Waiting	Waiting	Waiting
Transportation	Moving parts and materials from one place to the other	Task switching / Context Switching, Interruptions, Bad automation
Inventory	Undelivered products or parts	Code that has not been tested and integrated. Ready features not delivered to customers.

It is important to differentiate between waste in its pure form and waste, which is “necessary” in order to get things done. For example, testing in software development is a non-value adding activity, but it is virtually impossible to produce complex software without testing. Can you imagine flying on a plane knowing that its software was never tested? I’d rather walk.

Let us summarize the different activities you can have in a Lean implementation:

- Pure waste - any form of waiting can be described as pure waste. Also, projects that get developed, but never delivered to a customer, defective goods, etc.
- Necessary waste - activities that our customers have no interest paying for, but are required to get things done in a quality manner. These are all sorts of activities like planning, testing, tracking, documenting, reporting, invoicing, etc.
- Value - what the customer has interest paying for.

When we look around, we can clearly see a lot of waste being generated around us. Even a non-experienced Lean thinker will notice the multiple meetings where nothing gets decided, the dozens of defects filed by customers, the people waiting for their boss to assign work and so on. This is both bad news and good news. The bad news is that the majority of companies are spending too much time and money on wasteful activities and the good news is, that yours does not have to be one of those companies. If you train your thinking in Lean, things will suddenly start to improve, thus giving you the competitive advantage against your competition.

To quote James Womack again, “*Lean thinking must start with a conscious attempt to precisely define value in terms of specific products with specific capabilities offered at specific prices through a dialogue with specific customers.*”

Why would he say that? Is it even close to reality that companies do not really know what value is? The striking answer to the question is that most of the product companies have a really hard time defining real value for the customer. This is the reason why we see some companies succeed big time, while others fail. The ones that succeed have been able to first, identify what

value means for a given customer segment and second, deliver a product that represents this value. The thing is, that only the customer can define value in its entirety and sometimes even the customer cannot do that. I bet you have heard the famous quote by Henry Ford, "*If I had asked people what they wanted, they would have said faster horses.*". Yes, people sometimes do not know that they want something until they see it. This was the case with the PC, the iPhone and many other innovations. All these facts make it really hard to guess what value means for the customer.

However, one thing is certain, the customers are not willing to pay more if they can pay less (with some exceptions for countries built on top of oil barrels). The customer doesn't care if you have to pay more to your vendors or if you had a bad quarter and you have to delay their order. It's your own problem and you have to do whatever it takes to meet their expectations. Cost is always a constraint for product development and one of the key aspects of Lean, when it comes to cost, is to put the customer first. You should be asking yourself how much would the customer be willing to pay for a product. Then calculate the cost for the production of the product, provided that no **waste** exists. Provided that there is a reasonable margin between these two, it is worth pursuing the possibility. As long as you are able to remove significant waste from your development process, everyone will be happy. The customer will be happy to buy the product at a cost they perceive as reasonable and your company will also be happy because of the profit. Removing waste from the development process also makes it possible to push back on competition, as long as they are not doing better than you are.

Before we close this chapter, let us touch on the idea of "total product" or, also referred to as, "whole product". This concept is important because it is the equivalent of value. Customers enjoy value in its entirety and not delivered piece by piece. Yes, there are technology geeks, who enjoy getting their hands dirty, but most of us do not. I have an example of a real-life experience, which will definitely keep me from buying from a certain brand again.

When I was moving into my new home, I had to purchase all sorts of electrical appliances. Of course, this also meant a fridge. I had the fridge delivered to my house, but the delivery guys just put it on the street, handed me the invoice and left. So I had to adjust my expectations and figure out how to bring the appliance inside with nobody around to help me. Luckily enough, I am a relatively strong man, so I could lift it and manage to bring it in. This wouldn't have been the case for many other men, not to mention the average woman. After some fitness, I put the fridge in the right spot, unpacked it, and.. Oh my! There was no cable to plug the thing in! I had a fridge with no cable! I was sure that the delivery company had made a mistake, but, taking a look at the manual, I realized that the appliance had no cable included and I had to call an electrician to install it. You can imagine how pissed I was, right? I needed the fridge, NOW! So I got in my car, drove 10 kilometers, purchased a cable, went back home and attached the cable myself using some skills acquired during the course of my engineering degree. Unfortunately, my misery did not end here. The bloody appliance did not start even with a cable attached to it. Digging half an hour through the manual, it became evident, that I had to not only connect the cable, but also connect two plates inside the fridge itself. I had to remove its back side, figure

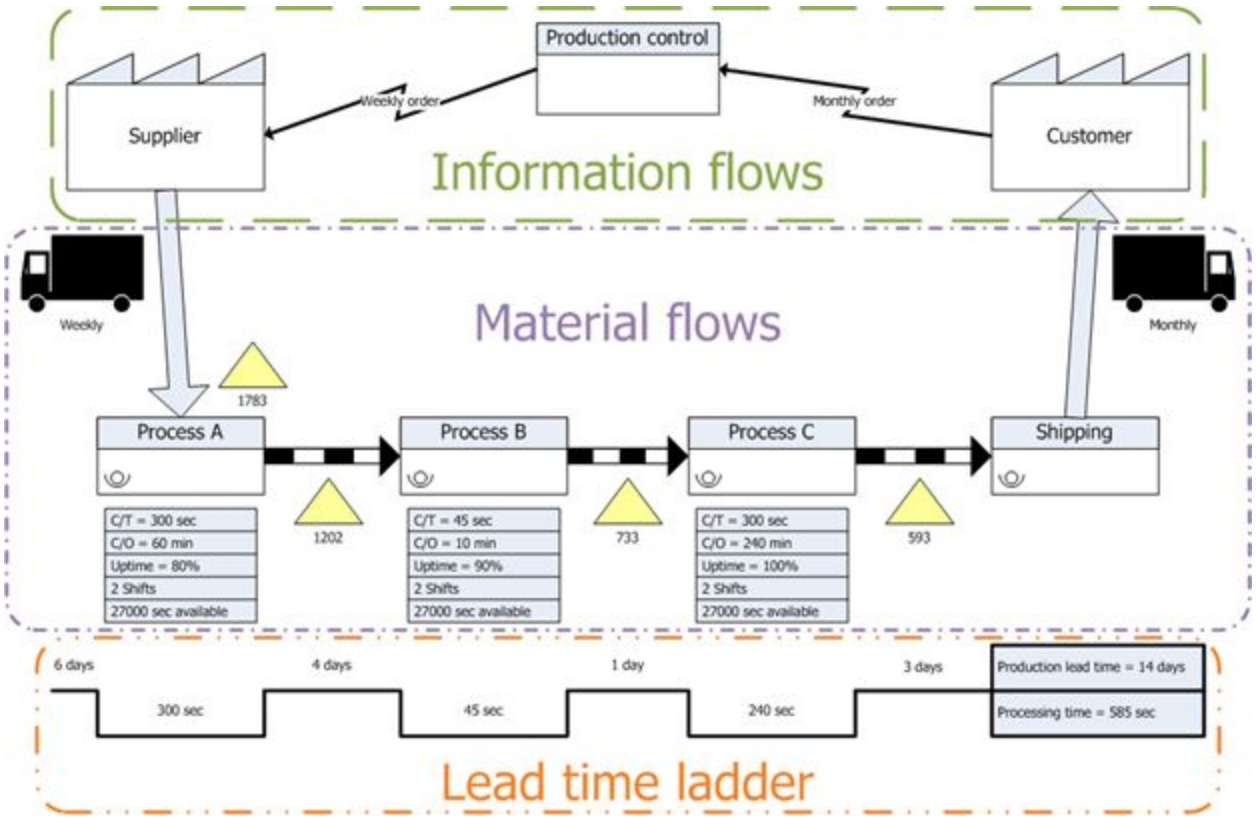
out which plates had to be connected and connected them. Finally it worked! I was happy with my success, but these guys will not get my money again. I swear!

The reason why I told you this story is to give you a different perspective about value. I did not just need a fridge. I needed someone to bring it in for me and, even more importantly, I needed to be able to use it. I did not want a fridge, I wanted cold beer!

So, the first principle of Lean asks you to redefine value in a complete and meaningful way for your customers. Value is what the customers really want, along with the price they are willing to pay. Having this information, you should focus your entire process towards recognizing value adding activities vs. non-value adding activities and work relentlessly to maximize the former and minimize the latter to meet these expectations.

Value Stream

The value stream map (VSM) is a tool used in Lean to visualize the flow of information and materials on their way from idea to delivery of a real product to the customer. A sample value stream map is displayed in the image below:



Value Stream Map (source Wikipedia)

Value stream maps are primarily used in manufacturing and not so much in knowledge work, so I will go through this principle briefly. The key points in a VSM are the stations (processes)

where value is added to the product, the queues between each station and the cadences with which information and materials arrive, move through and depart from the system.

The example portrayed in the picture above is strictly related to manufacturing, but I will explain it just so you get a sense of the idea.

- There are three major stations/processes: A, B and C.
- Each of them has the following characteristics:
    - Cycle time (C/T): the time required for the production of one piece of the product.
    - Changeover time (C/O): the time needed to configure the process/station to do a different job.
    - Uptime: the amount of time the station is available to run jobs.
    - Shifts: how many shifts work on that station.
    - Seconds available: Out of one day (24 hours), how many seconds is the station available, provided that uptime is 100%. This is important if this station is included in more than one of the value streams.
- Every two stations are separated by a queue (the yellow triangle). Queues are where we store the result of one station before it enters the next one for processing. Queues are a signal for some sort of delay in the system, which should be kept at minimal possible levels.
- We have the incoming and departing material cadences illustrated as trucks for delivering or shipping goods (1 week and 1 month respectively).
- Same goes for information cadences, which are represented by weekly and monthly orders.

It is important to note that the VSM is an invention by the American industry and not part of the Toyota Production System (TPS). I consider it a very good practice, but it is not a fundamental principle as the other three. VSM is a good way to ease the adoption of Lean simply because it adds structure to the way you produce goods. Having clear descriptions of your current processes allows you to observe the workflow and pinpoint areas of the value stream that are problematic. Then you can benchmark how much time and effort these stages require and work with your teams to improve. Without such structure, it is rather hard to run any improvement initiatives.

Let us mention that the VSM almost perfectly maps to a Kanban board. If we take all the steps and the queues from the VSM and draw them as vertical columns of a table then we achieve the following structure, which, resembles a typical Kanban board:

Supplier	Q1	Process A	Q2	Process B	Q3	Process C	Q4	Delivery

Pull

Once upon a time, project managers used to plan all activities months in the future, by the hour for their teams. If you were a developer, you would know what you were supposed to be doing six months from now at 3 pm on Monday. Such estimation and planning hardly ever work due to uncertainty, varying demand, change in requirements, specialization and so on. This does not mean that it never happens, of course. Many customers would not award a project to a vendor that does not have a detailed plan, which forces many people to do it, even though they know that it will not work.

Lean takes a different approach. Instead of building things on schedule, quite often ahead of time, you build something as soon as the customer orders it, which is, in essence, the PULL principle. This implementation requires a great level of flexibility and very short turnaround times, but it's a competitive advantage and manufacturing companies have been doing it for decades now.

But it is not just manufacturing companies that benefit from the PULL principle. I would like to give you a really nice example of the PULL principle in our daily lives. Where I live, we have a great pub called Ale House. It is unique because it has real beer taps built into the wall. Take a look at the picture below:

Ale House - A Restaurant with Beer Taps

This is one of the best examples of elimination of waste I have ever seen in a restaurant. The customers want some beer (value)? No problem, they get it the minute they want. No need to wait for an order, no need for the bartender to pour beer into the glass of the customer and the customer has constant feedback how much they have consumed (there are small displays showing liters). Last but not least, the beer is a local one, brewed in the pub's basement and is really good. I wish I had a built-in tap at home too.

Another example would be the process of pre-ordering music albums, which is quite popular these days. The bands and the labels that they record with allow pre-ordering because it gives them more information about the expected sales. This, of course, helps. There is no point investing money to release all across the world, if pre-orders come from one or two countries only, right?

One of the most typical examples is what we see every day at the supermarket. As a matter of fact, supermarkets are probably the most advanced implementations of PULL. Supermarkets can generate significant losses if there is not enough stock in place as well as if there is too much stock that ends up dispensed because it reaches its expiration date.

Organizing your business according to the PULL principle is a powerful economic advantage. Producing only what the customers want, at the time they want it, prevents the muda of overproduction and inventory. Relating this principle to product development means a handful of things, but probably the most important one is to listen to what your customers say. If they want something, then you better build it for them, because they will be ready to pay.

If you wonder how to implement the PULL principle, here is the three-step recipe:

- Use Kanban boards to track all the work
- Set WIP limits for all columns on the boards
- Do not exceed these limits

Flow

The principle of flow urges you to create a value stream through which information and products never stop ‘flowing’. If you have the entire process blocked by the interrupted flow of either information or products, waste occurs. You should design your workflow in a way that allows constant movement from its beginning to its end. This is hardly ever possible in knowledge work, such as programming, but the less waste you allow in the process, the more competitive your company will become.

One of the frequent occurrences of waste is between developers and QA engineers.The developers would work on something, then consider it done and push it for testing. However, the QA guys are working on something else and the feature that has just been developed has to sit in the waiting queue until the QA team can get to it. If the QA guys could start testing the new feature right away, then there would be no wait time and the FLOW through the value stream would be smooth. That is why we had to detach the QA people from the actual development process at Kanbanize. At first, this may sound like a really bad idea, but smoothening the flow makes things work much better than you would expect.

Another example of interrupted flow is any review process. For example, the technical design review process, which is usually performed by the architecture group or by a senior member of the team. These people are usually really busy and cannot swiftly respond to review requests from other team members. What happens in reality is that a job ready for review will spend some time waiting for that review, during which time the engineer working on it will start working on something else (people won’t usually stay idle). Then the architecture group will perform the review and, in the best case scenario, will approve it for further development. However, the engineer has already switched context and cannot immediately switch back to the job that has just been reviewed. This is, once again, an interruption of the flow and leads to higher work in progress levels and eventually to higher cycle times.

Improving the FLOW in a given system is a challenging task, but, by following these three steps, you can start addressing issues one by one. The first step is to focus on the product you are producing and keep in constant sight of how little pieces of value are being added from the very

first to the very last step of the process. An example would be a feature that you want to “flow” through your system. To make that feature pass from design to implementation to completion in the fastest possible way, you make sure to constantly observe its state. Whenever the feature is blocked for some reason, you and your team make sure to unblock it. Blocking can happen due to missing or unclear requirements, delayed feedback or reviews, not enough hardware resources, no QA available, etc.

The second step is to envision how the little pieces of value from the previous step could be added as quickly as possible. This often means ignoring some of the existing limitations that you have observed in the previous step. One approach here would be to expedite a feature once, just so you get a baseline of the ideal cycle time / lead time for a feature of this given size and complexity. Then compare this time with the average time for all other features, that have not been expedited. The delta that you get is the potential to improve by managing flow correctly.

The last step is to come up with concrete improvement proposals that would allow you to reach the ideal state, which was defined in the previous step.

Practices

Economic View

Being the active product manager at Kanbanize, I used to find it really tough deciding what to prioritize for development and what not to. When you have a lot of ideas and requests on your hands, it is difficult to decide whether you will work on a feature right away, put it in the backlog or drop it completely. It all starts with your product strategy. If a feature request contradicts your vision for the product, then you simply discard it, irrespective of whether you can get cash for it or not.

If a feature request does not contradict to the product vision, then you have to evaluate the potential economic impact. Will that feature bring you any return right away, how much can it bring in total, will implementing this feature instead of anything else bring you more value or not, how much effort do you need, is the return on investment good enough? These are all questions that you need to answer before deciding if and when you proceed to implementation. I know that it all sounds like common sense, but ask yourself whether you have a concrete economic framework to evaluate features. More often than not, the answer would be no.

Creating an economic framework is not an easy task, because it is hard to quantify the different goals of a project - increased quality, better performance, more robust architecture. These are all different characteristics of a product and we cannot compare them, unless we convert them to the same unit of measure. Not surprisingly at all, the common unit of measure is profit.

If we should quantify one thing, which affects profit, then this should be the cost of delay. If we can calculate how much 10% better quality would cost us and compare that to the cost of the

delay, which we would need to achieve better quality, then we will know if pursuing better quality is a good target or not. The same goes with better performance, more robust architecture and so on.

Therefore, the answer to the question of which feature to start implementing now and which feature can wait becomes a bit easier to guess. We prioritize the features with higher delay cost before the features with lower delay cost. If two features have the same cost of delay, we implement the smaller feature first, because we reduce the total risk and thus improve the overall economics again.

This raises another important topic - using a FIFO (first in first out) algorithm to sequence your features, as manufacturing companies do, is a really dangerous idea. You should always sequence work based on its potential cost of delay, which will guarantee that you minimize risk and increase profits.

Quantifying economics using the cost of delay is an attractive concept, but there are a bunch of situations where you won't know the cost of delay. This is what each SaaS company is probably facing most of the time. For example, we, at Kanbanize, push new features out each month, but we do not really know the cost of delay for 90% of them. We are still trying to figure out how to calculate the cost of delay of the new batch-update feature vs. the cost of delay of the new runtime policies module vs. the cost of delay of the new flexible self-reporting service. However, this does not mean that we never try. On the contrary, we are doing our best to assess how many customers will potentially use this feature, how much money it will costs us to develop it, what else we could deliver, etc. You have read about our economic framework in the "Product Management" section of the previous chapter. It is definitely not an absolute calculator that can be used to project actual profit, but it does a great job for us. If you could implement this in your context, I strongly advise you to do so - even if it is not perfectly accurate, it will still save you a lot of money.

Reusable Knowledge

One of the best quotes by Allen Ward is *"Create effective knowledge that the organization can use over and over, so that the expertise created becomes its primary asset"*. So, what stands behind this thought?

First and foremost, one of our goals with every project should be that we create reusable knowledge. Not just implement the requirements, not just deliver the product on time, but also learn how to do that better each time and document it. All chief engineers at TOYOTA have the goal not only to deliver the new car within budget, scope and time, but to also contribute to the global pool of knowledge in the company. When such knowledge is created and is universally accessible, building innovation becomes much easier, because you don't have to discover the wheel each time.

Generating knowledge, as multiple products get created, leads to the challenge of how you store this knowledge. At TOYOTA, they used visual "Trade-off curves" to represent how two (or more) characteristics of the system affect one another. Traditionally such information would be stored in IT knowledge management systems, which ease the discovery of information. To make this information usable, I would personally recommend including it as part of a formal training, which all employees should be entitled to attend.

Realizing the need to create more knowledge in our company, we have implemented the idea called "reading bonus". The rules are very simple: Each person who reads three books for the year and presents what they have learned, receives a 50% bonus to their salary at the end of the year. If they read four books, the bonus grows to 75% and if they read five books, the bonus goes as high as 100%. All people who reached 100% also get a special T-shirt for extraordinary achievements. This year, we are running this experiment for the first time and I still cannot share data about how well it works, but I believe it is going to be a success. One thing that we will probably change in the future is the monetary reward, but we are still evaluating a more engaging and interesting approaches.

Finding ways to boost the creation of **reusable** knowledge in your organization is a key instrument to enhance innovation. If we have the reusable knowledge in place then we would not have to innovate everything, but just some parts of the system. This, in turn, speeds up the entire development cycle and creates more value for the organization. But be careful! Allowing your people to work on innovation during 15% of their time, without the proper mechanisms in place to capture and exploit the accumulated knowledge, may turn into an expensive source of waste.

The Lean Startup

You must have heard about The Lean Startup by Eric Ries. His book revolutionized how products in the startup world are built but these concepts are actually universally applicable. There are a bunch of key ideas in the book, but I will share the ones that I believe are most important to product development in general.

Test

Testing your assumptions may start before you have invested a single dollar building a product. When Google Glass was about the get started, they were able to validate some of the core ideas in just a couple of hours. What they did was to prototype the actual device with some wires and plasteline. That's how they figured out what the maximum weight of the gadget can be. What they also realized was that the nose was able to carry more weight than the ears before it started to feel uncomfortable. Another test that they performed at Google was to test how it would feel to control things with "Minority Report"- like gestures. They realized fairly soon that holding your arms up in the air for more than a couple of minutes was a challenge and they scratched the idea. Can you imagine how much money they saved with these two simple tests? Probably a lot…

Another alternative to test your assumptions before you have built anything is to actually talk to potential customers and get to know their problems better. This is a great way to get a feeling about features, pricing and potential issues. Nowadays, it is becoming more and more popular to “crowdfund” ideas, which is something like pre-ordering a non-existing product. When the targeted funds have been collected, the founders build the product, because they know that they have already X items pre-ordered and hence have the cash needed.

Minimal Viable Product (MVP)
This idea is so simple that it is still inexplicable to me how and why thousands and thousands of products never started like this. Building an MVP is the task of building a prototype, which is the smallest possible chunk of code or material, that would allow you to validate your assumptions about the product. When you have the MVP in place, then you release it for real usage and start learning whether building the product as you envisioned it actually makes sense or not. If your MVP is good enough, you should actually start making real cash out of it, which is a clear sign that you are doing things right.

Be careful, though, many startups fail because they release a crappy MVP. It is true what Reid Hoffman (the founder of LinkedIn) says *‘If you are not embarrassed by the first version of your product, you’ve launched too late’,* but if you release a complete piece of junk, nobody will ever pay attention and you will fail to actually learn anything from it.

Pivot
Having the MVP out the door is an exciting milestone, but, quite often, teams will realize that not many customers need the product they are building. In this case, they have to reconsider their entire strategy and “pivot”. To pivot means to change course and to challenge your main hypothesis about the product and the market you are in. You come up with an alternative hypothesis and repeat the cycle of learning and validating. You do this in a loop until you succeed (or you run out of cash). As the probability of depleting your cash reserves is always behind the corner, you should be “failing fast” if failure is an option in the first place. This is a popular concept coming from the Lean Startup focused on fast learning cycles and embedding the new knowledge in the further development of the product.

Stop and Fix

Stop and fix is a practice coming from the early days of Toyota. Actually, this practice evolved from the Jidoka concept, which we talked about earlier. This is fairly simple concept - do not propagate defects to new products. In other words, if you know that something does not work right, make sure to fix it before you continue with the rest of your work. If you have carefully read the case study, you must have noticed the part where failed tests create cards in the Developer’s expedite lane. This automation piece represents exactly this principle.

Upstream Kanban

The upstream Kanban takes care to help us realize what we want to build. It is the pre-commit moment where we evaluate ideas and apply the economic framework to ensure you will get maximum return of investment.

Set-based concurrent engineering is a practice which I find particularly interesting, as it can be easily applied in the upstream part of a value stream. It came from TOYOTA and it is a fantastic tool to help with uncertainty.

The way that Toyota does product development is not that trivial. They start with a set of options that could work. Then, in the progress of developing the product, they would aggressively remove the weakest options when data to support such decisions became available. Eventually, only the best option remains and therefore it is used further in the development process. This process is known as set-based decision making or also set-based design.

There is a similarity between set-based and upstream Kanban. In the upstream we do not commit to work. We evaluate ideas. The more a given idea goes downstream, the more knowledge we collect about it. At some point in time we can decide to discard it, as a non-valuable addition to our product. If it survives the entire value stream it goes to the delivery Kanban.

Delivery Kanban

The delivery Kanban is where we implement things. The items that enter the “In Progress” area on the delivery Kanban must be implemented, because we have validated the idea in the upstream value stream and we have reached to the conclusion that it has business value. The delivery Kanban is the manufacturing line where we only care about speed and efficiency. It must be clarified that an item may get dropped if it has not crossed the line between “Requested” and “In Progress”, but that should not happen frequently, as this would be a signal for ineffective upstream. The delivery Kanban in our company has been described in details in the “Product Development” section of the case study.

Fast Feedback Loops

One thing I have always considered extremely wasteful is the huge number of meetings that people in big companies believe are required to get something done. Not only is time being wasted, but feedback is also delayed, which, in turn, creates more risk for the actual product. That is why I would like to suggest a meeting structure that we have found to be really productive. It will certainly differ from what you have in your context, but if you find some of the rituals interesting, feel free to adopt them. Only one is mandatory and it is the first on the list.

Daily standup meeting

If you are not aware of the concept, I urge you to read more about this type of meeting, it is probably the best thing that came out of Scrum (no offence intended). It is a must-have for every modern team and if you are not doing it already, this is the first thing you have to change starting from today.

In short, the daily standup has a few characteristics:

- It is held every day at the same time.
- Everyone is expected to arrive on time and delays are not tolerated. If you find it hard to get everyone on time, schedule the meeting for a weird time, e.g. 9:37. You wouldn't say 9:37 unless you really meant it, right?
- The meeting must not last more than 15 minutes.
- People stand and do not sit. This makes it easier to have the meeting limited to 15 minutes (nobody likes standing for too long).
- Every team member reports on what they have accomplished yesterday, what they intend to accomplish today and whether they are experiencing any issues.
- This meeting is not status reporting for the boss. This is a meeting from the team for the team.
- Leaders and managers also participate and report to the team.

The daily standup meeting has a few key benefits:

- Improves information flow between the team members. Quite often someone will be working on a task that another team member has experience with. In such cases, it is quite easy for them to exchange ideas or schedule a separate follow-up discussion. As a matter of fact, this is probably the single most important indicator to measure the effectiveness of your standup meetings. If everyone is eager to give their "status" and leave, then you are probably doing it wrong. If people spawn off-line discussions after the standup, then you are good.
- Creates a rhythm for the team and creates a focus to deliver something valuable each day.
- Adds a bit of healthy peer-pressure. Nobody feels comfortable reporting "No Progress" every day. This is a good way for the team to hold each and every member accountable for their work.

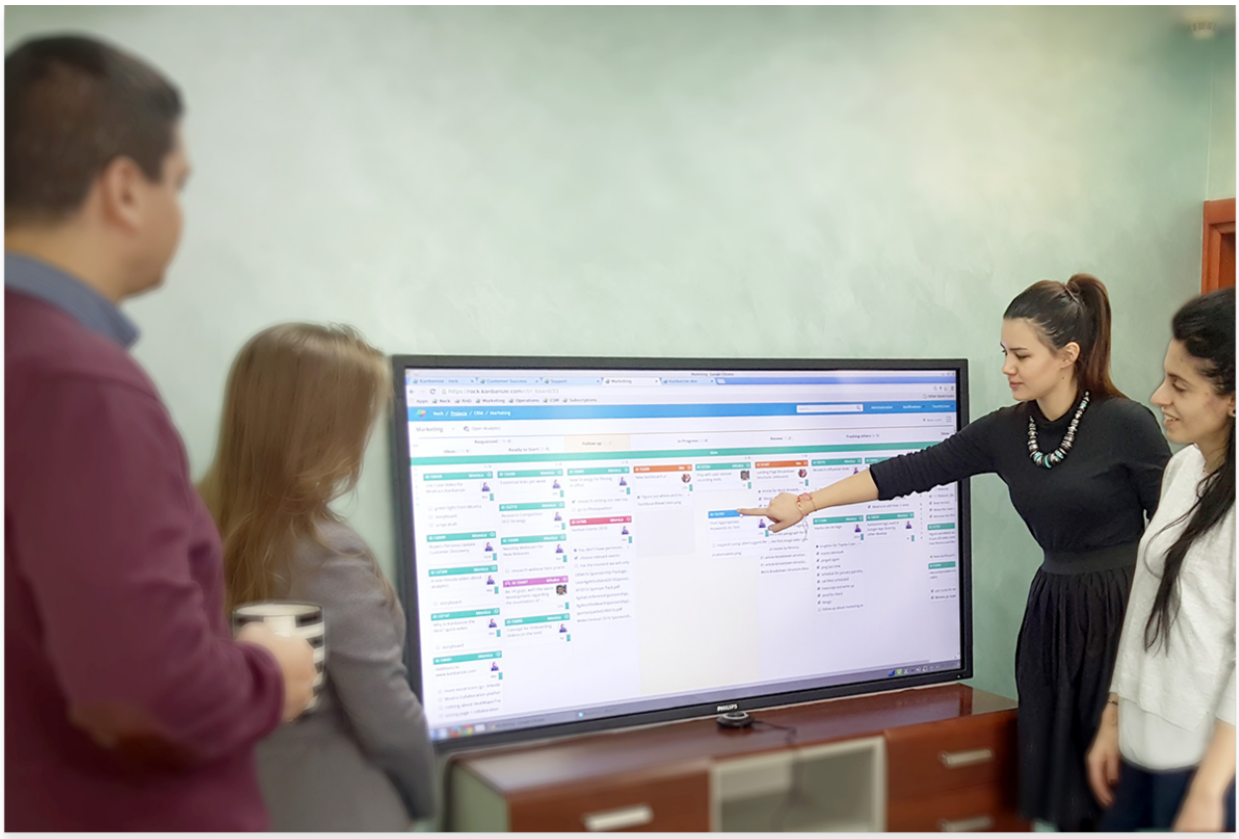
We have four different stand-up meetings in the company, all of them held at different times, but all of them in the time range between 9 am and 10:30 am. The maximum duration of each stand-up is 15 minutes and we use our 65'' touch screen to 'visualize our words'.

First is the management stand-up meeting at 9:00, where we gather with my two co-founders and we report to one-another what we've been doing yesterday, what we plan to do today and what the status in our teams is. It is a brilliant way to sync on current priorities and identify some cross-team gaps that would otherwise take weeks to find. Last but not least, we do 30 push ups

to start the day with slightly lower levels of testosterone in our blood. This last one was not a joke.

Then the automation QA guys meet to discuss their internal progress covering the product with automated tests. We used to have them in the Dev stand-up, but the team grew too large for a single 15-minute meeting, so they spawned a new one.

Third come the developers and then a joint stand-up between the customer success, sales and marketing teams. This last joint stand-up is a really effective ritual, because sales and marketing have a lot of intersection points, but, in reality, are quite detached from one another. That is why we run this meeting together - to improve the flow of information between the teams. This is how the marketing/sales standup looks (or at least a part of it):



Stand-up meeting at Kanbanize (From left to right: Jason, Marina, Monica, Gabby)

Weekly team meetings

Each of the teams in Kanbanize has an internal meeting dedicated to KPIs. The meeting is held on Monday or Tuesday. We collect a lot of data which is being discussed and analyzed at the team level. We also use half of the meeting to nominate the highest priority items for the week, so that we always have the right thing to work on. It is something like a mini-replenishment meeting.

Strategy meeting

After all of the team meetings have been held, me and my co-founders spend a day, every second week, discussing the strategy of the company, going through our team metrics and reporting on experiments that we are currently running. This may sound like too much for more mature organizations, but we are still a young company and we are laying a lot of ground work to ensure rapid growth in the future.

All-hands meetings

Each month, we have two one-hour all-hands meetings which are either dedicated to learning more about Lean or discussing important company topics.

All-hands workshops

Each quarter, we organize a workshop outside the office. It is a whole-day event where we discuss strategic topics with the teams and we also work on potential ideas for improvement. It is a combination of planning meetings, strategy meetings, risk review and team building.

As you see, we have adopted quite a number of rituals, but we find these absolutely necessary to keep our senses alert for potential issues or opportunities. Also, we are a company that PULLs a lot of feedback and ideas from the people who work with us and we are doing our best to ensure everyone has a channel to convey a message through, which also includes regular 1-to-1 meetings.

You should, of course, choose what feedback loops to implement in your context. Some of the meetings might not make sense for you, but most of them should be valuable. Remember, the stand-up meeting is a must have at almost all levels!

Limit WIP

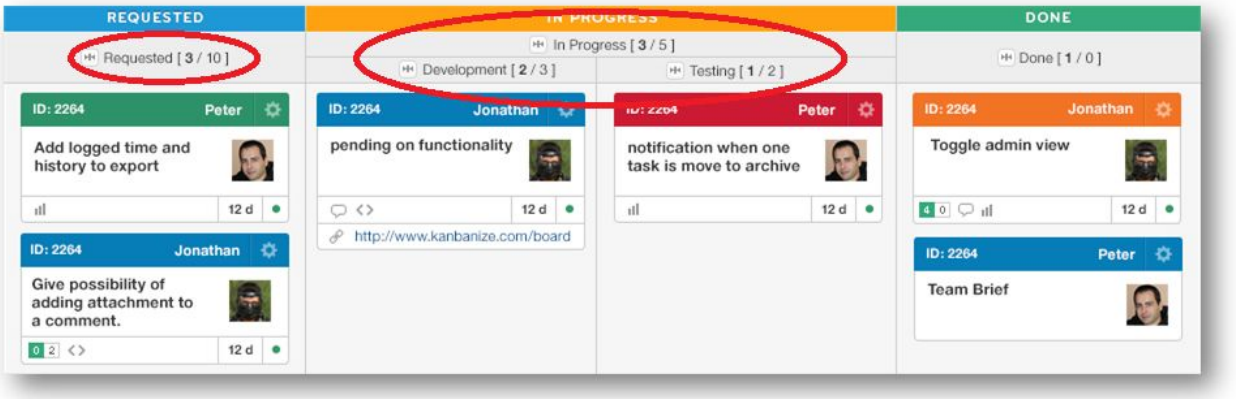
Limiting work in progress is probably the best thing about Kanban. This is what makes your teams super efficient and allows you to deliver value to your customers faster than ever before. By applying strict WIP limits on our RnD processes at Kanbanize, we were able to improve our delivery times with more than 300%. It takes a lot (and I really mean a lot) of discipline to be able to do that and, unfortunately, many teams fail, but with some help from our side you will be in the position to improve your processes dramatically.

People are better at starting work than at finishing it. We all get excited to build a new prototype of a super-cool thing, be it a gadget, a program feature, a blog article or anything else. However, we hate it when we have to document it, fix it, test it, debug it and so on. The Pareto principle is a good explanation about this phenomenon - we simply get 80% of the work done in 20% of the time and then have to polish all the details for 80% of the time, which, quite often, tends to be an annoying, tedious job.

What happens when you have a bunch of talented, creative and motivated people? You leave them working without any formal structure and you find them spawning hundreds of new ideas, testing out hypotheses, evaluating frameworks, but not really delivering value to the customer. This is, of course, an exaggerated situation, but not too far from reality.

Another situation, which actually happens quite often: A team is really busy pushing a new release out the door. Then, a high-priority request comes from the management. The team is forced to stop working on the current items and begins working on some other items. If you have nothing to keep the system in order, it will quickly revert to chaos and a lot of the items that have been started will end up forgotten and therefore never completed. This means **waste**.

Kanban has the means to control such situations or basically any other situation that would result in starting more new work rather than completing work in progress. Namely, using the WIP limit concept. Setting up a WIP limit means that you only allow X number of work items in a given column on your Kanban board. This is how WIP limits look on a real Kanban board:



Kanban board with WIP limits

Combined with the rule that “what goes into In Progress must get done”, limiting the number of cards in each column is what makes the Kanban board a PULL system. This becomes so because when you fill in the limits with items, you can only start new work when a customer (which may be an internal team too) PULLS some of your work (i.e. a card gets done and delivered). When that happens, you have free capacity to move a new card into In Progress.

To fight off any remaining skepticism, let us also talk about a mathematical equation that explains the proportional relationship between WIP and cycle time. It is known as Little’s formula, named after John Little, the MIT professor who first proved it. His formula tells us that the average number of customers in the store *L*, is the effective arrival rate λ, times the average time that a customer spends in the store *W*, or simply: $L = \lambda W$. Since this formula can be applied to any stable system, we can use it for a Kanban system too, where L would be the work in progress, λ would be the throughput of the system (Little’s Law assumes a stable

system so the arrival rate and departure rate are identical) and W would be the average cycle time. Then we get the following equation: $WIP = Throughput\ x\ Avg.\ Cycle\ Time$ or also $Avg.\ Cycle\ Time = WIP/Throughput$. This last equation clearly shows that as long as we have a stable system (none of the parameters change dramatically) then the greater WIP would result in greater cycle times.

For those of you who need a more concrete example in product development, we could replace the customers in the store with the bugs that a software development team has to fix. Then Little's law would be transformed to: the average time that we need to fix a bug is equal to the number of bugs currently being worked on divided by the rate at which we usually fix bugs. So, if you have your team working on 30 bugs and they fix 10 bugs per week on average, then the average cycle time for fixing a single bug would be 30 / 10 = 3 weeks. If you reduce the work in progress to 10, then the average cycle time for a single bug would become 10 / 10 = 1 week. Isn't it wonderful that you can reduce the cycle time by a factor of three, by just applying a WIP limit? I think it is.

If you have no WIP limits then there is no safety net to revert to when chaos is about to occur. However, when the limits force you to keep certain levels of work in progress, you think twice before switching to a new task or saying YES to your boss. As proven by Little's law, this makes you much more efficient and allows you to complete work at a much faster pace.

Talking about bosses, it is crucial to mention that they have to buy the idea about WIP limits. If yours still live in the multitasking world, the only thing that you can do is to try to educate them. This is a funny game, which takes not more than 5 minutes to play, but is the best instrument I have ever seen and used to educate people on the inefficiency of multitasking. Highly recommended.

Manage Queues

Let us start this paragraph with the blunt statement that queues are one of the major factors contributing to the duration (cycle time) of a given job or entire project. If you recall the Value Stream principle, we talked about queues and they were represented by the yellow triangles on the Value Stream Map (VSM).

The teams that start measuring how much time jobs spend in queues on average are often surprised to discover values ranging between 50% - 90%. Even we, at Kanbanize, who are absolutely aware of this phenomenon and deliberately try to minimize queue time, see that some jobs get delayed in queues quite a lot. Can you imagine what would happen if you could magically get rid of all queues in your processes? You would be gaining 50%-90% of the project's cycle time, which, I presume, sounds like fiction (but it is not).

Above all, queues are an economic problem. As stated above they lead to longer cycle times (and hence risk), which is definitely an economic problem, but they also delay feedback, decrease quality and motivation.

When feedback is delayed due to larger queues, we risk to generate waste. A typical example here is the architecture review process. What happens often is that a draft version of the design is submitted for a review. However, the lead engineer or architect is busy and cannot review the design proposal. Facing the alternative to sit idle, the engineer either proceeds implementing the design, keeping fingers crossed that it will be approved or switches context to something else. When the review is completed, the engineer has to get back to that task (switch back context). Switching contexts is a form of waste, which we have already proven with the game presented in the Limit WIP paragraph in chapter 3. What if the design is bad but has already been partially implemented? Then we have to either move on with what we have, just so we do not lose it, or scratch it and start again. These are also forms of waste, which almost always lead to poor quality too.

And what about motivation? Would you feel motivated to work on a design, when you know that it will take weeks to get it reviewed and approved for actual implementation? Likely not.

How Do Queues Occur?

Queues in a given process occur right before a step with limited capacity and/or high utilization. Actually, capacity utilization is the single most important factor for the occurrence of queues. This is somewhat natural, because when a process is run at 100% utilization, any new work would automatically sit on the “waiting” queue until someone has free capacity to take it.

The interesting part is that capacity utilization affects the size of the queue exponentially: going from 80% utilization to 90% utilization would double the queue size; going from 90% to 95% will double it once again. Since the queue size also affects the cycle time of each new job, you have to be careful what percent utilization you operate your processes on. If cycle time is important – choose lower utilization, which would guarantee a quick turnaround time for important new jobs.

Typical Places to Look for Queues

In reality, a company that is not trying (hard) to implement Lean will have “ghost” queues all over the place. I say “ghost” queues, because nobody realizes that they exist and that they greatly affect the ability of the business to execute. To make it easier for the non-experienced Lean thinker to find queues in their work, let us present a couple of typical places where they occur in almost any company out there.

Product management

Product managers usually have a big queue of non-refined ideas collected by themselves or the marketing team. A lot of ideas could potentially elevate the company to a new level, as long as product management has the capacity to refine them into real business cases and potentially

work with the engineering teams to realize them. Failing to do so results in a lot of missed opportunities or sometimes to direct losses when customers churn because their feedback was not heard.

This, of course, does not mean that product managers should break down each and every idea into tangible user stories. This would be waste.The idea is that PMs should have the capacity to, at least, evaluate the ideas according to the economic framework, which we already talked about. Only when an idea has proven to be valuable enough to pursue, then the product manager should invest their time to create the necessary user stories and proceed to implementation.

Sustaining / Support teams

In a typical product development organization a sustaining team will have a huge backlog of customer issues. If things are really bad, they will be growing day to day and the only way to fight them off would be to either defer many of the low-priority issues or just bring new engineers on the team. Both ways are highly ineffective as far as economics is concerned.

The only reasonable way to tackle sustaining queues is to 1) maintain their size so that they never grow out of control and 2) gradually pay off the debt you've accumulated. We will explore a concrete example of this behavior in the last chapter, which will be presenting how we do it at Kanbanize (as of this moment we have only one customer issue open and we will be closing it pretty soon).

Business reviews / management reviews

When operating in a field of great risk it is usually required that all jobs get reviewed by a senior team member. A concrete example might be a code review by a team lead, sign-off by a product manager, test cases review by a senior QA, etc. These types of queues are quite bad not only for the economics, but also for the morale of the team. No person is happy to work on something, then pass it for a review, start working on something else and then receive the results of the review two weeks later (when they'd forgotten half of what's been done).

Quite frankly, this is an area for improvement for our own RnD team and I cannot be of much help proposing concrete steps for improvement. I know what we will end up doing - decentralization of control, but we are still thinking how exactly to do that. Maybe in the second edition of this book, if there is such, I will have something more to share on that topic.

These are, by far, not all the places where you can find queues in a company. Queues are basically everywhere and we, as Lean thinkers, should do our best to first, start spotting dangerous queues, second, evaluate the economics of these queues and third, figure out how to minimize the potential losses they might cause.

Managing Queues

Managing queues is hard, but at the same time simple. All you need to do is put strict WIP limits on those queues and try not to exceed them. Note that sometimes queues might be good. If you have an obvious bottleneck in your process, you don't want it to sit idle. In such cases it is a good idea to put an upstream queue, so that the bottleneck always has somewhere to PULL from.

Reduce Batch Size

To explain what batch size is, let us take the basic example of how DHL, FedEx and the rest of the courier services work. When someone sends a parcel to the other side of the world, the courier will not fly a plane for the parcel alone, but will try to transport as many parcels as possible. This will save the company a lot of cost for transportation and will allow it to offer more competitive pricing for its clients. What the company does is to "batch" as many parcels as possible and pay the transportation cost just once. This effectively means that the company executes one transaction over a big batch of items.

The problem comes when the batch becomes too big. Then, the receiver of the parcel may have to wait months to get it, because it takes a lot of parcels to fill an entire plane. However, many people would not send parcels if they were to travel two months. In turn, this would undermine the core business of the carrier. Therefore, the companies are sometimes forced to decrease the batch size in order to meet the expectation of their clients.

Now the logical question is: "Where is the balance between the economy of big batches vs. the shorter cycle time associated with smaller batches?"

Before we attempt to answer this question, let us take a look at another situation. Imagine you need to undergo a surgery of both eyes. The doctors offer you to try out a procedure with new laser equipment, which is completely painless, does not cause damage to the eye and is relatively cheap. The only problem is that nobody else has attempted such an operation before and there is no real evidence that it will work. So, the million-dollar question here is: If you agree to that surgery, would you have it on both your eyes at the same time or would you rather have one eye operated, then recover successfully and then go for the second one? In other words, would you rather go with a batch size of 1 operated eye and twice the money or would you prefer to have a batch size of 2 operated eyes and save half of the money? I believe that most of the people, would go with the smaller batch size because the risk would be much smaller.

It is fair to claim that the economic risk of flying a plane full of goods is much bigger than a plane delivering a single parcel. If the plane is kidnapped by aliens then, in the former case, we would have to compensate thousands of recipients while, in the latter, it would be only one. Generally

speaking, big batches always carry a bigger risk and that is why Lean suggests minimizing these batches as much as possible.

Minimizing the batches is economically viable if we are able to optimize the transaction cost. If we insist on decreasing the batch size for the sake of decreasing it, we would be wasting a lot of money. So, how do we decrease the transaction cost?

- One way to decrease the transaction cost is to implement cadences that allow the involved parties to plan their activities better. One example is the regular meetings that we all have - we just know that we have them and we accommodate for that in our schedule. If all these meetings were ad-hoc, at different times of the day, then we would waste a lot of time trying to find a good slot in which everyone is available, waiting for people to join (some of them will forget for sure), catching up with the last progress on the fly and similar situations.
- Standardization is another alternative. If you have implemented standards for all the important activities related to a given transaction, people working to complete it will never have to wonder what to do. They just follow a check-list of items and mark them off one by one. However, be careful with standardization. It is okay to standardize procedures, but it is dangerous to standardize work in the attempt to eliminate variability. This only works in manufacturing.
- Repetition. Once a transaction has been completed a sufficient number of times, the transaction cost will also go down, because the people will know better what to do and how to do it. It is just like throwing a dart a thousand times - you become better and better each time.
- Innovation. A few years back we started talking about drones carrying spare parts for their larger brothers or even delivering our Amazon orders. Is this whole single-parcel-plane thing still a stupid idea?

It is clear that batches will continue to exist because they save companies a lot of money. However, reducing them also reduces the associated risk. With this information available, we should be doing our best to decrease the batch size as much as possible without hurting the economics of our processes.

Implement Takt

Takt is the time interval at which customers are ready to accept value from the vendor. In our case, the takt time is one month, because we have discovered that our customers prefer that period. Theoretically, we could be pushing code to production much more frequently, but a lot of our customers prefer not to see the system changing every day.

I mentioned this before, but I would like to second it. Takt is not a sprint or an iteration. You don’t plan takts, they just come. When the takt comes you get the goods you have produced

and deliver them. We do not plan that we will release 4 features or 30 story points or whatever. We just do our best to deliver as much as possible.

This approach is superior to iterations for a couple of reasons:

- You save yourself the hassle of planning and estimating work. You could do it if you want to, but we do not do it, because we believe it is waste. All we need is a rough idea of what is expected to be ready for the next shipment, but this is by no means a commitment or anything of that sort.
- There is no fear of failure. When you plan an iteration and your plans are wrong, then there is a sense of failure. People get demotivated and some managers use this kind of events to punish someone. This is obviously wrong and shouldn't happen, but it does.
- The Parkinson's law that "Work expands so as to fill the time allocated for its completion" cannot exist. We all know these guys who completed their sprint ahead of time and decided to play some Starcraft. This cannot happen with a flow-based approach, because you always have something to work on.
- You can achieve Heijunka - leveling of the work. A typical scrum team will start the sprint slow and then towards the end of the iteration will work late hours to compensate unexpected delays. This does not happen that often with a takt-based approach, because the work just goes on and on in an endless flow.

Manage Cycle Time

Some of you may wonder what's the thing that keeps teams from falling below a certain performance level. After all, when you don't have plans, things can take a really long time, can't they? The trick is to measure and manage cycle time. If you know the average cycle time per work type per size, then you will know when a given task is taking too long. The team will know it too, and this will work as a self-controlling mechanism.

How do we know the size, when we have not estimated it? Well, do not estimate it, state it. How do you state it? You just wait until the task gets done... That's how you remove the guesswork from the equation. If you want to make the sizing exercise less dependent on personal judgement, use the judgement of just one person. If this person is the technical lead of the team, she will be able to size items properly and will be consistent in her approach. We do this with our team and it works pretty well.

If you follow these simple steps you will know exactly how the team is doing and even how each person on the team is doing. I know that some of you find the personal productivity a taboo-metric, but we prefer to know exactly how much each team member is contributing. If somebody delivers more you have to reward him/her more compared to others. Otherwise it won't be fair.

Last but not least, managing cycle time is much easier than managing throughput. To reduce the average cycle time, you just have to lower the WIP limits of the related process steps and

leave the rest to the Little's law. Of course, you cannot do that forever, but most teams have quite some room for improvement. Now you tell me what a team should do to start delivering 50 story points instead of 30 :-).

Closing Words

If you have reached this page, there is not much left for me to say, except THANK YOU! It took me more than a month to compile this read and I hope that it has brought you at least some value.

Lean is a journey and I will be happy if this paper got you started. Keep Kanbanizing!

If you want to connect with me for a demo of Kanbanize, some tough question, consulting session or just a chat, find me on LinkedIn, or follow me on Twitter.

Special thanks to Biso, Christo and Marina for providing their feedback and making this paper a much better read. Special thanks to Monica for the hard work on the editing part. You guys rock :)