Discovering Sustainability in Software Development

"People who have managed to intervene in systems at the level of paradigm have hit a leverage point that totally transforms systems."
Donella Meadows

This paper is about our process of discovering and learning when trying to implement Lean principles and the Kanban approach for sustaining engineering in a software project. We have realized that a method of continuous understanding of our own reality is the key to leverage our processes at the level of paradigm. Here I describe how we are applying this model in our environment and the toolset that has been emerged from our efforts in organizing this obtained comprehension.

The ability of a system to self sustain is probably one of the most difficult to achieve. At Phidelis, a small software company in Brazil focused on financial and academic solutions for big universities and medium-sized schools, we realize that "sustainability" is a key concept in allowing customers to value long term relationships, by the establishment of a mutual trust culture. Sustainability is a key element to a competitive advantage in our niche. Most of our customers came to us after three or even more bad experiences with previous software supplies. One important cause for almost all of the supplier replacements was associated with a lack of potential sustainability on a long-term basis. The customers already know how software is strategic to maintain the health of the business. They want to be sure that they are taking the right chance.

Sustainability, in this context, means basically the continuous effort to build a mutual trust relationship capable of generating, in customers, a feeling that the software supplier can: (1) maintain their business processes working efficiently on a long term basis, and (2) create the necessary new business opportunities or operational improvements that help them to stay updated with the technological requirements demanded by our current society. So, as a software supplier, this is our main purpose today: create a self-sustainable system capable of generating these feelings in our customers.

We know that we have a very difficult job on our hands. But, if you have an easy goal, certainly it would not be a problem to be solved with the Lean approach. We believe that Lean is about pursuing a long term philosophy. And that is what we are trying to do, which brings us to a more important message: we don't focus on the right and wrong of our process. We prefer believing that our current process is just the best way to do our job considering the current conditions of time, business challenges, technical and knowledge restrictions and resource constraints.

System Thinking

One of the heaviest influences on our mindset formation process is the belief in systemic theories. The best way to define and manage a process is getting a deep knowledge of the systemic relations of its components. Value, Flow and Waste are Lean elements, but before that they are also systemic components of the process, and this comprehension is far more important than any guidebook for us.

I had access to the first work that caused a huge impact in our mindset in January of 2008. The "Twelve Leverage Points to intervene in a System" (Meadows, 1999) was

originally written in 1997 by Donella Meadows, a scientist and system analyst involved with the study of complex systems like environmental and economic structures. Before that, I had already been influenced by some other authors with a systemic bias in the Agile Software Development area, specifically Jim Highsmith and his study of Complex Adaptive System and Emergence (Highsmith, 2000), David Anderson with his analysis in Theory of Constraints applied to software development (Anderson & Schragenheim, 2003), and Mary and Tom Poppendieck in translating Lean concepts to the software development world (Poppendieck & Poppendieck, 2006).

Meadow's work is quite interesting in many ways to software development. She lists twelve possible places to intervene in a system in order to create intentional changes in its behavior. If you read the list carefully, you will notice an impressive well structured way of thinking about your system, besides clues that guide you when you are looking for practices, methods and philosophies to improve your own model of work.

Let's take a closer look at some places on her list. For example, one of the key practices of the Kanban approach is the establishment of limits to the volume or size of the work that advances to each stage of your process. You can find some support for this practice in her list by analyzing the eleventh point: "The size of buffers and other stabilizing stocks, relative to their flows." She exemplifies: "Business invented Just-in-time inventories, because they figured that vulnerability to occasional fluctuations or screw-ups is cheaper than certain, constant inventory costs - and because small-to-vanishing inventories allow more flexible response to shifting demand (...) There's leverage, sometimes magical, in changing the size of buffers." We definitely had a huge leverage when we started to define and manipulate the buffers' sizes. Kanban was the instrument, System Thinking, the reason.

The tenth point, "The structure of material stocks and flow and nodes of intersection", is also interesting and is about Flow, one of the most important Lean concepts: "(...) the leverage is in understanding its limitations and bottlenecks and refraining from fluctuations or expansions that strain its capacity." Although this point was not ranked as one of the best ways of creating leverage, the author is clear about the reason for that as related to the huge challenge in manipulating flow in physical systems, like plumbing structures, traffic systems or even logistical operations. But for software processes and services, the flow of work has an enormous potential to create fast improvements.

And the list goes on, "positive and negative feedback loops, information flow, the rules of the system, the goals of the system," and the effect of "constants, parameters and numbers." In fact, all of these topics can be easily seen in our work system. You can visualize the results of our systemic approach by looking at our electronic Kanban Board (Figure 1). This board represents what we think about our system offering an important visual management tool. The board helps us in making all the work visible, in maintaining our focus on the way things flow through the process, and in signalizing important information about the state of the system.

But, for us, the biggest impact occurred when we decided to intervene in the place cited by Meadows as the second most powerful to change a system: "the mindset or paradigm out of which the system arises." So, we did that when we started to think about our process and our business in Kanban and Lean terms. Below you will see what we have discovered since that.

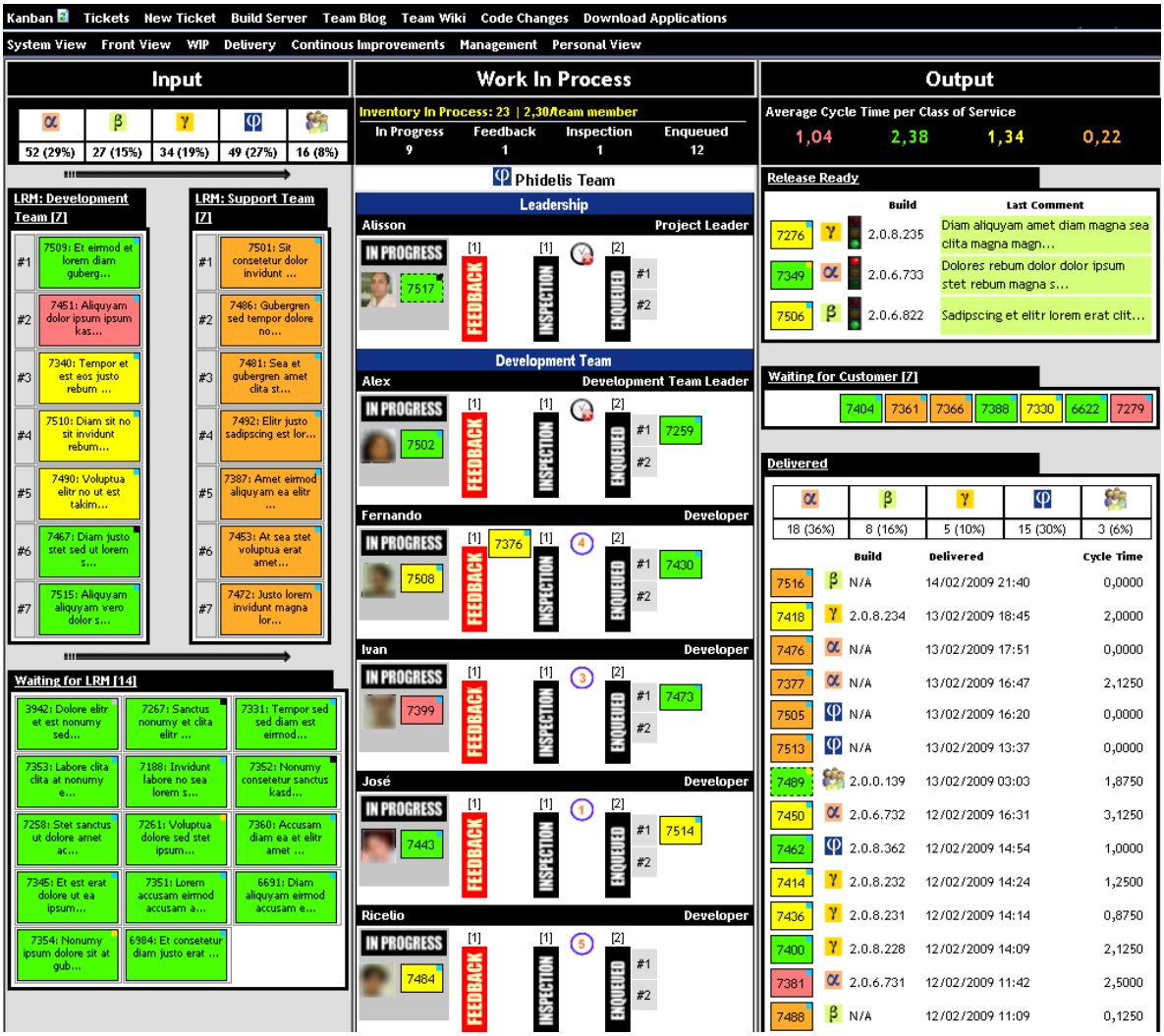


Figure 1: Our Kanban Board in System View mode

Finding Value

It is not easy to find Value without going in depth to understand your business through the eyes of the customer. The software that you produce is not Value by itself. The Value can be found in the business opportunities that can be created by the software. We felt this when we built a new system for on-line enrollment of students in primary and secondary schools a few years ago. The process of enrollment is one of the most critical for these institutions. Parents must pay fees, sign contracts, select customized services and choose the new classroom for their kids on the school website. In times of Internet and Web 2.0, this was not such a big deal for us. However, assuming the customer perspective it was not enough just publishing the new software on-line. We had to work with them defining strategies to adapt and spread the new service through the organizations, doing customization for different realities, flexible data extractions and quick support and problem solving. The sensation of Value comes from the whole package of services, not just the new features. The result of this approach creates the business opportunity for schools to have from 20% to 30% of the enrollment process free of operation costs in the first year, and almost 50% in the second year. No more

lines or delays at the school, neither an overloaded administrative staff. Value is frequently clear like this.

What we are trying to do is subordinate our system to Value generation, not feature generation. Our game is not about features, it is about services. Delivering features is just part of the whole package. We need to embrace other activities like being oriented by business results, flexible rules, custom procedures, effective support and quick problem solving.

Demand Management

Once we try to orient our system to generate Value by delivering the right features supported by the right services, we must care about how to organize and control our demand in such a way that we can organize, classify, prioritize and get the required comprehension as it is entering into the system. Some practices that we have discovered as important to manage the demand are:

1. Reduce variability by classifying the demand in classes of services

Classes of Services represent different types of demand, address different standard-work definitions and follow different patterns of flow. They can reduce variability because all members of a Class of Service (we call them Tickets of Service) have similar characteristics. It means that they share similar rules inside the same class. Different classes have different purposes, timing and people required.

These are some of our classes of service:

Class of Service	Description	Value Perspective	Visual Signalization
New Value	A new feature that adds value to a customer by creating new business opportunities or operational improvements.	Value	
Improvements to Sustain	When a modification is necessary to maintain a working business operation. The new feature will not add any new value to the customer but, the software must be changed to adapt itself to some new scenario.	Waste	
Support Operations	Actions to help customers in using the product.	Waste	
Problem Solving	When, for any reason, the customer cannot use the current features of the product appropriately.	Waste	

The colors of the cards are important. Identifying the demand through the color helps in getting some insights from the system without going in depth in analyzing complex reports or graphs. Yellow, Orange and Red map directly to waste. We know when the system is stable, because the board is becoming greener.

2. Classify the demand in sizes to reduce variability and adjust measurements

We don't estimate at all. Instead, we try to classify the work in three particular sizes:

Size	Applicability	Time Associated	SLA	Visual Signalization
Reference-1	Assigned to a Ticket of Service that can be delivered from one to two days at the most;	1 or 2 days	3 days	
Reference-3	Three times more complex or difficult than the reference.	3 or 4 days	5 days	
Reference-5	Five times more complex or difficult than the reference.	5 to 10 days	15 days	

Our SLA (Service Level Agreement) policy is defined by these references. We look at the demands from a "sizing" perspective to make the establishment of a reference for SLA possible, not to create a commitment. If a team member realizes that the classification is imprecise, we just change it, which also implies moving the demand to another SLA category. What matters about this classification is just the capacity to measure the velocity considering the volume of work required to deliver each item. This is related to the mutual trust relationship required to run this kind of approach. SLA is not a commitment at all, it is a reference. The customer has to trust in our evaluation of sizing and be able to negotiate based on scope instead of time.

3. Prioritize demand in different levels of importance according to time

The "Prioritization Filter" technique is very useful in making continuous planning possible and in helping the team to organize the demand in front of the system. We use filters in four stages to facilitate the prioritization process:

a) LRM Area: Demands in this stage reach the "Last Responsible Moment." It means that the best time to inject the demand in the system has come. Average Cycle Time and worker availability is used to analyze the best opportunity to do that. Only 7 items can stay here at the same time.
b) Waiting for LRM: The next 14 items that are just waiting for a new empty space in the LRM queue. These items need more attention than the others to avoid delay.
c) The Front of the System: All the rest of the demands that are stationary in front of the system waiting for an opportunity to be injected.
d) The Archive: Demands that expire or become unnecessary as time goes on. This type of approach allows the customer to privilege some demands over other ones. This creates a backlog of old demands where the value is

questionable. We maintain the expired demands on this stage just to keep track and history. Eventually we can restore a demand from The Archive and work with it again if the customer re-evaluates its value after some time.

This model also helps to create flow and allows the start of a Pull System mechanism. Look at the picture in Figure 1. It shows the screenshot of our Front View board. The arrows exemplify some possible paths of flow.

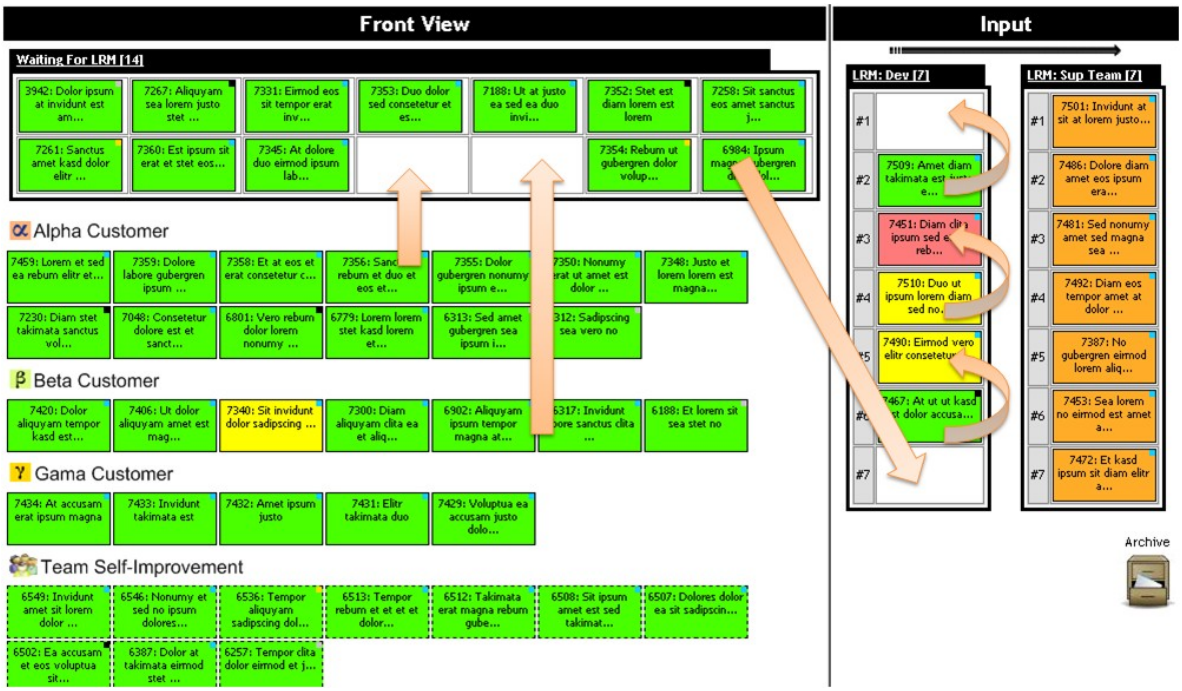


Figure 2: Flow in Demand Management area

You can visualize the beginning of a Pull System here. A signalization is used to indicate the need of a new prioritization action and the need to pull more work from the previous stage. The last stage (LRM) is the only one where ordering matters. The team will look to the LRM queue and take the one at the top as the next work to do. When this happens, an empty space appears. So, the Product Owner has to pull another card from the previous stage to fill the space. When he moves a card from the "Waiting for LRM" stage to the "LRM" queue, another empty space is created in the first one. So he has to pull one card from the Front to fill in the hole.

4. Subordinate demand to capacity and to the limits of the system

The pulling mechanism explained in item 3 creates flow. The important concept here is that this mechanism allows the subordination of the demand to system capacity. You can't move a card downstream if there is no empty space to fill. The empty spaces are created by the action of the team to pull more work. Once the team members are limited to work one Ticket at a time, the system helps to avoid more work to be injected before the current work being delivered.

5. Continuous planning and monitoring to maintain the Demand synchronized with our business criteria

Continuous planning is part of our Business. Our worst and more chaotic phases come when we have tried to manipulate this systemic behavior. We plan every week to maintain focus on short time goals, but there is no fixed scope. Instead, we work with a "fixed WIP" concept. WIP is an acronym for "Work In Progress." When a demand enters into the WIP stage, we do our best to work with that until delivering it. This establishes a cadence of frequent delivery. The previsibility demanded by traditional planning is replaced by the knowledge acquired in making the work done continuously.

WIP Management

Managing the Work In Progress is definitely the most difficult and critical process in this approach. The WIP is always over pressured. One of the jobs of the Team Leaders is to control this pressure.

The Team Member area

We have to start the description of our way to manage WIP showing the Team Member area. Each work cell member has his own area to organize tickets in progress.

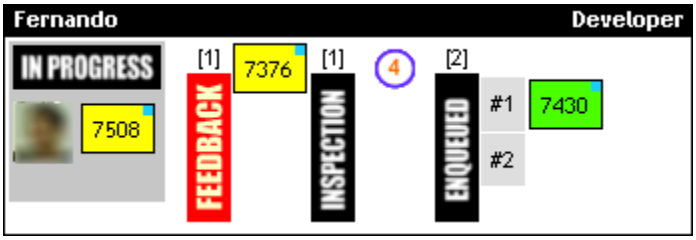


Figure 3: The Team Member area

Each team member can hold five cards at the most. More than three cards is a sign that the team member is overloaded and needs help. The Team Leader should help to identify this situation and intervene. The five card policy is more necessary than desirable.

The "In Progress" area holds the ticket that the team member is working on at the moment. The feedback area shows a ticket that has not passed in the quality assurance policies. This area is critical (that is the reason why it appears in red on our board) because of our code integration process. When the developer finishes his implementation, he commits the change to the real branch of code that is used in the production environment. If this code is released in bad quality, the customer will suffer from problems with the product.

Negative Feedback Loops

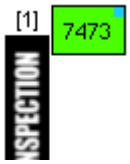
At the moment he sends the code to the repository, the whole system - formed by our work system and the product itself - is potentially in an "unstable state." So, we need to restore the system to its natural stability level as soon as possible.

We recognize this as a systemic "negative feedback loop" behavior. As the systemic analyses suggests, a negative feedback loop requires:

a) A goal: A stable and “ready to release” software to deliver.

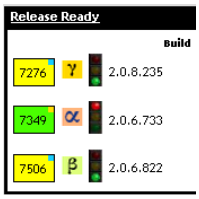
b) A monitoring and signalizing device: to show that we are out of the goal.
Our Kanban board offers some signalization to guide the team in restoring the system to its ideal state.

A new version of the product can’t be released without passing through an inspection process executed by another team member.



So, when a ticket is moved to “done,” the board assigns that ticket automatically to another member, by choosing the first one in the inspection members queue. This is quite interesting because people usually create queues to manage “work.” In this case, we have a limited queue for “Team Members.” It is limited to the number of Team Members that are able to do inspections (mainly the more experienced Team Members or, which is more common, all the members of a mature team). The limit of the queue is also flexible to absorb variation in the size of the team (vacations, trips, resignations, etc).

This “unstable mode” can be amplified if the “Inspector” finds some problem when analyzing the changes. He analyzes, mainly, the adhesion of the new implementation to the customer request and the adhesion of the changes to the standard work definitions. In this case, the inspector sends the ticket back to the source for adjustment. Now we have the feedback area filled with a card.



If the same code branch was changed by another developer and this change passes to the quality assurance policy before the system returns to its original state, we signalize this drawing a “Stop the line sign” in the release ready area. At this moment, we know that one of our code branches is in an “unstable mode” and the team has to run the appropriated procedures.

c) a response mechanism: to put the system back to the required goal:

Team Members have to look at the inspection area at least twice a day. They don’t have to stop what they are doing to run inspections when they receive a card. Our moments to stop and do inspections are:

- In the morning, before the beginning of a regular work day;
- After lunch, before the beginning of afternoon work;

- Before the start of a new work item.

The inspection process takes one hour in average and each Team Member does about 3 or 4 inspections per week. This is enough to have all Tickets of Service inspected after completed. Team Members are oriented to run a specific check-list that involves an analysis to verify if the service really matches the original customer request, if the standards were followed and if there is no improvement that could be made to make the service more compliance with the customer's request.

If everything is fine, the Ticket of Service advances to a "Ready to Release" stage, where the Customer Support Team has to prepare and execute the delivery process. Otherwise, the team receives a message like this on their desktops:

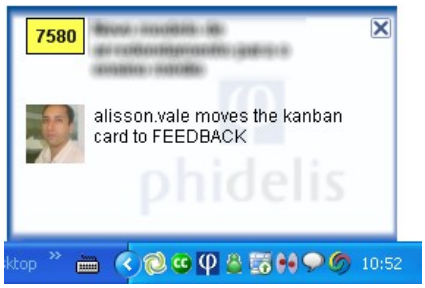


Figure 4: A pop-up screen appears when a Kanban card is moved.
This is important for fast responses when problems are found during inspections.

The team member that originally executed the job receives the Ticket and is oriented to give immediate response to address the issue raised in the inspection. Meanwhile, any build after that in the same code branch remains frozen until the problem is completely solved.

Overload

One of our main concerns today is team member overloading problems. This happens when someone has to be worried about more than one Ticket at the same time. In practice the team member can use a personal queue to hold tickets that he has already committed. This personal buffer has to be managed very carefully because it:

1. can potentially raise a "task switching" behavior;
2. decreases quality and cycle time;
3. increases inventory levels in WIP;
4. You can loose the opportunity for another team member to pull the work earlier as he becomes free, slowing down the flow.

On the other hand, we have to work with this buffer while some dysfunctional situations continue to occur in our environment:

- A team member sometimes needs to interrupt the work in progress, because something more important suddenly appears. It´s also rare, but when it happens, team members hold the current card and start another one.
- A team member eventually speeds up the flow by pulling more than one ticket very closely related, dealing with all of them at once, which is rare but it happens occasionally;

To deal with this problem, we have a system that sends us a sign when the WIP area becomes overloaded (Figure 5). When the average cards per Team Member becomes bigger than 3, the team members have to change their behavior following a policy that orients them to not pull more work from the input area. They start to pull work held in the enqueued area of other members to decrease the WIP to normal levels. When this level becomes more reasonable, they return to pull from the LRM Area as they usually do.

Figure 5: Monitoring WIP

Business Activities

Perhaps you think at this point that this approach is about managing the Tickets of Services. It is not. This approach starts to work much better when you realize that managing the bigger chunks of work is also very important. To manage only the units of work maybe you just need a group of people and a traditional manager. But, to manage chunks of work with a business purpose you need a Team and Leaders.

This can't be a machine that eats demands and spits out software and services. We think of our ticket management approach as a way to create flow. But the system, as a whole, should be evaluated by its capacity to generate value. So we need a process for putting together several units of work (the ones that flow through the system) in a broader and business-centered structure for management. Today, we call this structure "Business Activities."[1]

A Business Activity is just a set of Tickets of Services that shares the same business reason. It is our unit of management and it is visually represented in a Parking Lot Chart (Figure 6).

[1] This term was borrowed from Feature Driven Development (Palmer & Felsing, 2002), but is slightly different because they are not composed only for features, but for any of our Tickets of Service.

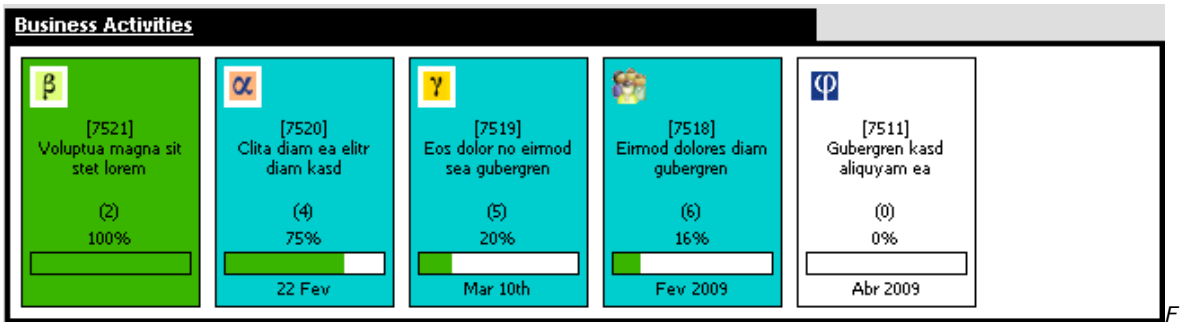


Figure 6: Business Activities are represented in a Parking Lot chart.

We use these structures to evaluate progress and to manage time and scope while we are continuously trying to deliver business solutions. A teamwork culture is determinant when we are doing this.

Team Work

Business Activities and Tickets of Services must be executed and delivered by collaboration among team members. Our Team Model is formed by work cells with Team Leaders and Team Members. A central leadership is responsible for maintaining the synchronized working of the cells, to be sure that the system is working with the most important things at the moment, to encourage continuous self-improvement and to help the teams in strategic and root cause analysis.

Our Kanban Board is also an important instrument in facilitating collaboration. Each Ticket of Service or Business Activity is executed by a couple of people who must be involved to deliver the service. The board facilitates the communication around each service by offering a central point for conversation and documentation of the work. Figure 7 shows an example of a ticket with an extensive conversation about the solution to a problem.

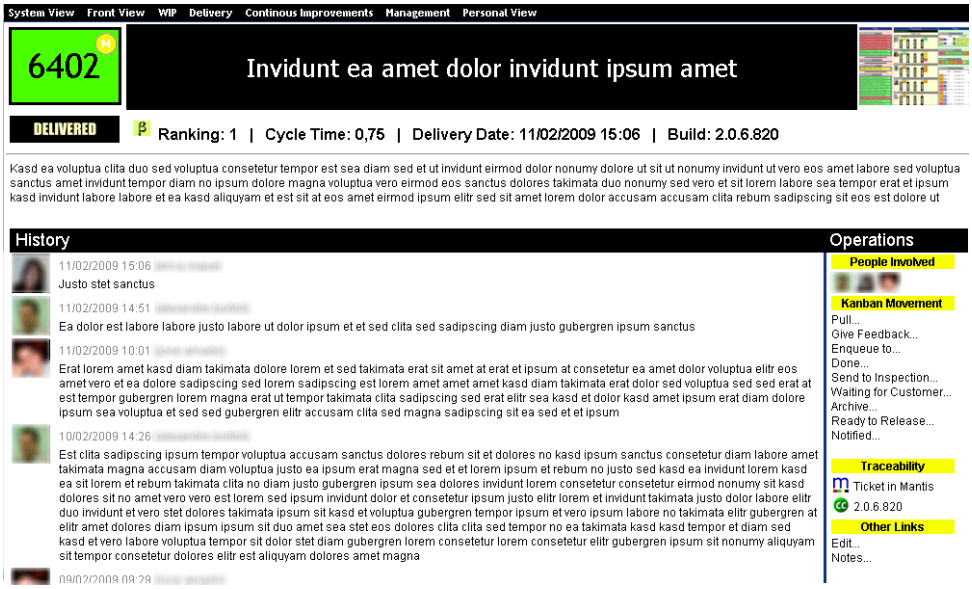


Figure 7: Three people collaborating in the execution of a Kanban card.

Collaboration is also important on a daily basis. Stand-up meetings, quick meetings with a few people involved in solving a problem and periodic meetings with the entire group to analyze the process and make it better are all important to create an environment that encourages team work.

Delivering

When any Ticket of Service gets out of the WIP area, it means that it is ready to deliver to the customer. The delivering process can be as simple as sending an e-mail to the customer with information about the executed service, or as complicated as reaching the database with an administrative account to run some script, or even installing a new version of the product on a production server. The common ending place for all possible actions of delivering is a formal notification to the customer.

Release Management

Release per feature[2] can be a nightmare if you don't have a good system to deliver new releases of your software. We release software every day, frequently even more than once a day. A good automated system to deploy is mandatory for success in this practice.

We have customers with different realities. Some of them have their own host structure, another ones uses our own data centers[3]. So, our release management process must be automated for

[2] Release per feature is a common practice in Kanban implementations. It means that you release a version of your software for each new implemented feature. You don't wait for a larger set of accumulated features to release.

[3] Our products are totally web-based, which allows us to host the software for small customers that are not capable of hosting it by themselves.

both cases. What we have, mainly, is a “one-step-deploy” system. This system is first executed in our continuous integration process.

When a developer changes the code base, the build server runs a lot of check procedures and actions to generate a new build, which means compiling the entire application, running automated tests, checking the integrity of XML configuration files, checking new database structures and some other specific automated validations to avoid deployment issues on the customer hands. Before the build is complete, the continuous integration script runs the one-step-deploy process in a stage server. Our installer application is executed in the background in the same way it will be done on the customer servers. The stage server is used by a team member to do inspections and by the customer support team to simulate exercises in the product.

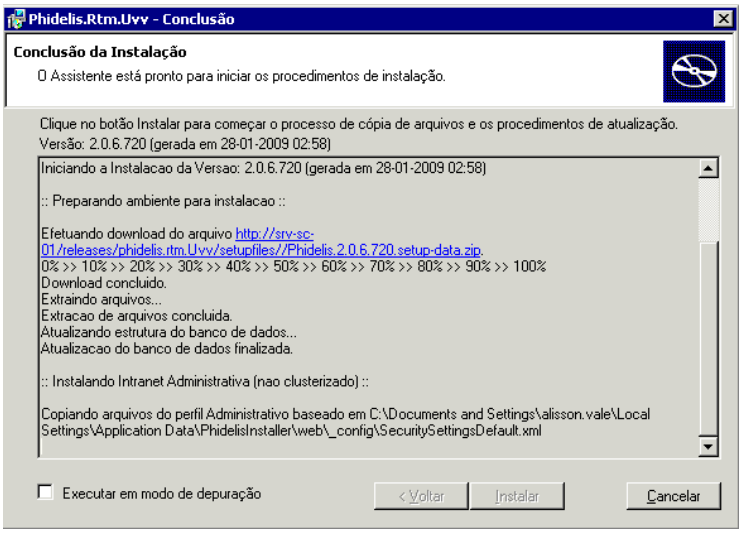


Figure 8: Screenshot of the installer application.
It Needs to be flexible to automate all required deploy procedures.

To fit perfectly in our process, we had to create our own installer application to achieve high levels of flexibility. It is based on a proprietary script language[4]. We can add automation for any operation that we have to do in the customer environment such as downloading compressed package files from our servers, copying files to different servers, starting and stopping windows services and other applications, updating database data and structures, XML configuration files and web server configurations, generating log reports and some other actions needed to avoid manual procedures in the deploy process.

When the deploy procedures are done and formal notification happens[5], the Cycle Time measure is determined.

Cycle Time

[4] It is a XML script pretty similar to the one used in tools like Ant, NAnt or MSBuild.

[5] Bigger customers run deploy procedures by their own way using our deploy instructions. In this case, the cycle time is measured when they receive the notification of a new available release.

We press the start button of our Cycle Time clock when a Ticket of Service enters into the WIP Area, and press it again to stop when the notification happens. Cycle Time is a measure that allows us to evaluate several aspects of our process:

a) Strong variations: When the average cycle time suddenly changes with a high level of variation, this is a sign to evaluate what went wrong, by discussing what happened with the team. Usually it occurs when a demand remains in WIP for a long period of time.
b) Decreasing the time it takes to do something: Getting shorter cycle times are always better if the quality and standards remain the same. It means we have learned to deliver faster, usually improving practices or automating repetitive tasks.
c) Previsibility: Knowing how much it takes to do some types of operations makes us more comfortable to plan and to establish strategies to handle with Business Activities.

Figure 9: Cycle time measures (in days) are visible in the Kanban board. The color points you to the context.

Another key point of the Cycle Time use is the possibility of analysis using different aggregators. For example, we can measure the Cycle Time per class of service (Figure 8), which gives us an idea about the average time per type of work, but we can also play with other dimensions, like getting cycle times for measuring how long it takes in average to create a new report, to fix a bug, to add new ways for product configuration, to create a new screen for simple data maintenance, and others. It is just a matter of classifying the Tickets of Services and knowing when to start and stop the cycle time clock.

Final Words

We know there are a lot of improvements that we have to implement in our process. Long term planning is not perfect yet, standardization and people training definitely could be better. We also still don't know how to scale this model from tens to hundreds or thousands of customers. I think that is the power of Lean and Kanban approach: the power of giving you a direction to follow through the understanding of some principles to apply in these scenarios as they come up.

We really have not reached the end yet. Even as I am writing this, our process is changing. New problems are arising and new opportunities are coming. More important than having a guidebook to read when facing these problems and opportunities is being able to build the capacity to write your own guidebook, adapting your process to the reality instead of trying to manipulate it to use a specific process. That is what we think Lean and Kanban are about.

References

Anderson, D. J., & Schragenheim, E. (2003). *Agile Management for Software Engineering: Applying the Theory of Constraints*. Prentice Hall PTR.
Highsmith, J. (2000). *Adaptive Software Development*. Dorset House Publishing.
Meadows, D. (1999). *Leverage Points: Places to Intervene in a System*. Acesso em March de 2009, disponível em Sustainability Institute: http://www.sustainabilityinstitute.org/pubs/Leverage_Points.pdf
Palmer, S. R., & Felsing, J. M. (2002). *A Practical Guide to Feature-Driven Development*. Prentice Hall.
Poppendieck, M., & Poppendieck, T. (2006). *Implementing Lean Software Development From Concept to Cash*. Addison Wesley Professional.