KANBAN "BLUE BOOK" EXTRACT

# The Microsoft XIT Sustaining Engineering Story

# CHAPTER 2

# A Former Athlete's New Challenge

“Kanban only works with small, co-located teams!” This was a widely held belief, around the time of the publication of the first edition – a belief that lingers to this day. It is, of course, a myth! It was an assumption based on an understanding that standing “looking at the board” was the core, essential element of the approach. Consequently, there were plenty of so-called “experts” willing to state publicly and categorically that Kanban didn’t work with geographically distributed organizations. If people couldn’t stand together in front of the board, it was postured, then clearly Kanban had nothing to offer. There is a huge irony in this myth. To explain why, the next two chapters tell the story of how we got started with Kanban...

Dragos Dumitriu is friendly, jocular, Romanian-American with a winning smile and an enthusiasm for life that seems to compel people to like him and attract them to follow his lead. Tall, bald, solidly built and spreading only slightly with the onset of middle age, he cuts a dashing figure in his handmade, tailored European suits and expensive sunglasses. Though 20 plus years living in the United States has softened it, he still has a distinct eastern Europe accent. As a package, there is something just a little intimidating about Dragos: something that says “I’m in charge. There will be no nonsense!” You can imagine smalltime gangsters running the other way at the sight of him, as he emerges from a large BMW, in downtown Bucharest.

His physique is a legacy from his past, as a young athlete, in the Romanian Olympic team. As a young adult, he owned and managed a fitness center in his native Romania, worked as a stunt double in movies, and as a personal body guard. He’s the personification of the expression “larger than life character.” Moving to New York with his then wife, a successful doctor, he took a low paying job in a psychiatric hospital and two years later had risen to manage it. Following his wife to Fargo, North Dakota, a remote city in the northern central part of the United States, perhaps best known for the movie and spin-off TV show of the same name, and infamous for its bitterly cold winters, he joined Great Plains Software as a project manager. Already well into his 30s, it was his first experience in the IT industry.

Following the acquisition by Microsoft to create what is now known as Dynamics, Dragos transferred to Seattle in 2003 and found himself in the IT division as a program manager. The following year, ambitious for a challenge, he volunteered to take command of the small sustaining engineering team in the XIT business unit – a team that was known for having the worst customer satisfaction record across all of Microsoft’s IT organization.

## XIT Sustaining Engineering

At the time, Microsoft was divided into 7 different businesses, each with its own profit & loss statement.

Units such as Windows, Office, Developer Tools, MSN, Hardware, Xbox and Dynamics were all treated like separate businesses. In addition, there was a corporate headquarters unit containing shared services such as human resources, finance, facilities management and security. Each of the seven business units plus the corporate function had its own IT function for a total of eight distinct IT services groups. XIT served corporate shared services and provided IT support and applications for services such as finance and human resources. It was led by Dale Christian, General Manager, who was later to serve as CIO of Avanade, and later still, CIO of The Bill & Melinda Gates Foundation. The Sustaining Engineering team was a small team tasked with minor feature upgrades and bug fixes “off-cycle” and outside of major releases and application upgrades. From an accounting perspective, XIT Sustaining Engineering, was an operating expense, while project teams working on the major project portfolio were considered capital expense. These are two different budgets and while capital expense is an asset, operating expense is pure cost. This had an impact on both policy constraining behavior and decision making.

The team Dragos volunteered to lead was located in Hyderabad in India, in a so-called “captive” center or campus, built by the outsourcing firm TCS, specially for Microsoft. A few short years earlier, Microsoft had made a strategic decision to outsource its IT function. IT wasn’t core to Microsoft’s mission or its identity rather it was an enabling function. It was reasonable then that IT could be provided as a service from afar. It was hoped that existing developers and testers working in IT on Microsoft’s campus near Seattle, could be repurposed to work on products within one of the other seven business units. Most of this switch had happened in 2003 and what remained of IT was largely a vendor management organization, consisting mainly of individual contributors with the job title program manager. Dragos’ job was to lead and manage the small 6-person Sustaining Engineering team working in Hyderabad. For context, Seattle is, depending on the time of year, either 12.5 or 13.5 hours behind Hydrabad. This time difference creates both challenges and opportunities when managing vendors in India remotely from the west coast of the United States. The advantage is that things can happen overnight. The disadvantage is that synchronous communication such as conference calls are challenging to schedule, and that by Friday in Seattle, it is already Saturday and the weekend in India. There are effectively only 4 days per week available for managing across this time difference.

As mentioned already the team had the worst record for customer service across all of Microsoft’s IT functions. This organization had stubbornly refused to show improvement. After the switch to a new team offshore, at the TCS facility in Hyderabad, things had not improved. So, all of the personnel had changed, the management had changed, and the service was now provided by a vendor with a master services agreement and yet things had not improved. So hopeless was the performance of XIT Sustaining Engineering that the program manager position had been vacant for some months – no one wanted it.

Into this scene Dragos arrived, ambitious, always up for a challenge, a born leader, and keen to make a mark, be recognized and hopefully rewarded with greater responsibilities in future, he volunteered for the job. A few of his colleagues thought he was crazy.

> *A friend of mine is a former Winter Olympian from the Austrian team. She competed in the luge at the Salt Lake City Olympics in 2002. Nowadays, she's a coach for the Austrian national youth team (18 years and under). Chatting with me in 2009, she advised me that when looking to hire new personnel for my business, "always look out for athletes."*

*"They have discipline. They know how to set goals. They are motivated. They know how to measure performance and they take an organized approach to training and improved performance." I thought of Dragos, how well he fitted this description and how valuable these characteristics were to his role at Microsoft.*

At this point, it's worth reflecting on why I spent so many words familiarizing you with Dragos. I want to lay the foundation to dispel another myth: the myth that Kanban would only work when it was led by remarkable, larger than life, characters like Dragos. The results Dragos achieved as you'll learn in chapter 3 are remarkable. Heck, I wrote two chapters of a book about them! It's been easy for people to dismiss these results as uniquely attributable to his character and not the method he was following. As you'll see towards the conclusions of chapters 3 and 4 this simply isn't true. Dragos doesn't have to be in the room in order for you to garner the same scale of results. You need to follow his method, his way of thinking. You need to follow the Kanban Method. Yes, leadership is truly required but you don't have to be a former Olympian for it work for you.

Dragos and his team were responsible for the software maintenance for the XIT business unit. He was the program manager for the XIT software maintenance service, known as Sustaining Engineering. His team provided two basic services: minor upgrades (known as change requests); and defect fixes. The team (shown in Figure 2.1), consisting of 3 software developers, 3 testers and a local function manager (2nd left in the photograph), developed minor upgrades and fixed production bugs for about 80 cross-functional IT applications used by Microsoft staff throughout the world.

I had joined Microsoft's Developer Tools division in September 2004 and hence Dragos and I were colleagues in different business units. We were yet to meet.

Figure 2.1 Dragos pictured with the XIT Sustaining Engineering team in Hyderabad circa February 2005

## The Problem

By summer of 2004, senior management and customers were out of patience. Something had to be done! Dragos volunteered to take charge. He loved this sort of challenge. He spent his first few weeks watching, learning, understanding, observing, and examining data from their tracking system. He wasn't tasked with filling the shoes of his predecessor. He was asked to do more than just fill the position and crank the handle of the existing broken process. Dragos was told to make changes - to fix whatever it was that was broken.

He quickly understood the customer dissatisfaction to be rooted in long lead times, unreliable delivery, and broken promises for what seemed like small, highly achievable, and often important changes and bug fixes. His team maintained applications such as the human resources employee records system and the payroll system. These were used by finance to enable salary payments to most of Microsoft's global workforce. To understand the nature of their work, let's consider a strawman business initiative and how it might impact XIT's applications. Let's imagine that Microsoft plans to open a new office in San Juan, Puerto Rico. Puerto Rico is a protectorate of the United States. They use the US Dollar. In many ways, Puerto Rico is similar to one of the states of the United States. Hawaii had similar status until 1959 when it became the 50th state of the United States.

You can imagine a business initiative such as opening a new office in San Juan will create impact for all of XIT's corporate shared services customers: finance will need to make payroll for Puerto Rican employees; human resources will need to store employee records for those employees as well as facilitate recruitment on the island; facilities management will need to provide an office building and allocate space to departments, and offices to individuals; while security will need to secure the premises and have a capability to print employee badges, and enable security scanners on entrances and exits.

One of the organizational dysfunctions is that often the high-level business objective or initiative was opaque to the workers in XIT. The business initiative would manifest as requests for support from each of the shared services who would in turn work with their product managers to push requests for changes to IT systems through to the sustaining engineering team. Requests would therefore appear in isolation, apparently independent, when in fact they may have been value in understanding them as a dependent set. Sustaining Engineering were set up as order takers, and the orders were for small changes, delivered in short order. Context was missing.

Fixing this bigger more strategic dysfunction was not part of Dragos' remit. His job was to make Sustaining Engineering better order takers, to make them fit enough to deliver what was asked of them.

So, you might imagine that requests might look like, "support Puerto Rican address format in the employee records system" or "support Puerto Rican tax withholding for payroll for Puerto Rican employees." These will break down into details such as "Add Puerto Rica to the drop-down menu of States of the United States on the employee address form". Any lay person familiar with using personal computers could understand that from a business owner's perspective these seem like simple little changes. Why then were they taking months? And why was the IT group constantly breaking delivery promises? It is easy to understand the frustration of the customers in finance, HR and the other shared corporate services.

## Current Capability

The Sustaining Engineering team had an average five-month lead time on change requests and this, along with their backlog of requests, was growing uncontrollably. Not only was the average lead time already unacceptable, it was likely that for any one item the lead time from commitment to delivery was between 6 weeks to greater than 1 year. As a service, they were slow and unpredictable. They had a habit of promising delivery dates and then failing to meet them.

## Constraints

The programmers and testers working for TCS were following the Software Engineering Institute's Personal Software Process/Team Software Process (PSP/TSP) methodology. Microsoft mandated this contractually with TCS. This choice had been made by Jon De Vaan, the Vice President of Microsoft's Engineering Excellence group. Jon reported directly to Bill Gates in his role as Chief Architect as well as Chairman of Microsoft. Jon De Vaan was a big fan of Watts Humphrey[1] of the Software Engineering Institute at Carnegie Mellon University. Humphrey had been recognized for his contribution to the professional of software engineering as a recipient of the National Medal of Technology awarded by the President of the United States. Humphrey was the creator of the Personal Software Process/Team Software Process and De Vaan had been looking for an opportunity to experiment with it at Microsoft. Unable to gain traction for it on product teams, he had been granted the opportunity to run his experiment with the IT division. Consequently, contractually obliging TCS to follow it. Jon De Vaan was an early developer at Microsoft and a trusted friend of Bill Gates. Sometime later, when the Windows Vista project went off the rails and had to be reset as Windows 7, it was to Jon that Bill turned as the person to lead the recovery. In 2004, as head of Engineering Excellence, no one was going to challenge the preferences of Jon De Vaan. This meant that changing the process used by the Sustaining Engineering team, changing their software development lifecycle method, was not an available option. This constraint turned out to be a stroke of luck! Dragos was forced into starting with what the team was doing now.

The perception was of a team that was badly organized and managed. As a result, senior management was not disposed to provide additional money to fix the problem.

Sustaining Engineering were order takers for small, short-order requests seen in isolation, they were a cost center, their hands were tied regarding their choice of working processes, and management was unwilling to provide additional funding to enable improvements – there was no appetite to throw more people at the problem.

By sheer coincidence Dragos had discovered my first book, Agile Management for Software Engineering. Impressed with what he'd read, he asked for my advice. I arranged to visit him in his office in building 115 on Microsoft's campus in Redmond, WA in the leafy green eastern suburbs of Seattle. The interaction, interview and analysis described below has been formalized into the first steps of the STATIK (systems thinking approach to introducing Kanban) method. This method is described fully in chapter pp.

## Visualize

To begin to understand the problems, I asked Dragos to sketch the workflow. He drew a simple stick-man drawing describing the lifecycle of a change request, and as he did so, we discussed the

[1] https://en.wikipedia.org/wiki/Watts_Humphrey

problems. Figure 2.2 is a facsimile of what he drew. The PM stick figure represents Dragos.

Requests were arriving uncontrollably. Four product managers represented and controlled budgets for the customers functions named previous, such as Finance, Human Resources, Facilities Management and Security. Request were for small upgrades but also included production defects (problems discovered in the field by end users). These defects had not been created by the maintenance team, but by the application development project teams. These project teams were working on the major project portfolio and their work was considered a capital expense, or an asset. Those application development teams were generally broken up one month after the release of a new system, after the end of the so-called "warranty period," and the source code was handed off to the Sustaining Engineering team for further maintenance. While many readers will recognize this dysfunctional pattern. We weren't in a position to do anything about it. Our job was to make the sustaining engineering team better and faster at fixing bugs, not to help XIT as a whole, reduce the quantity of defects created. This was therefore another constraint. We were not in a position to shape demand or enact changes that would reduce demand. Sustaining Engineering were order takers.

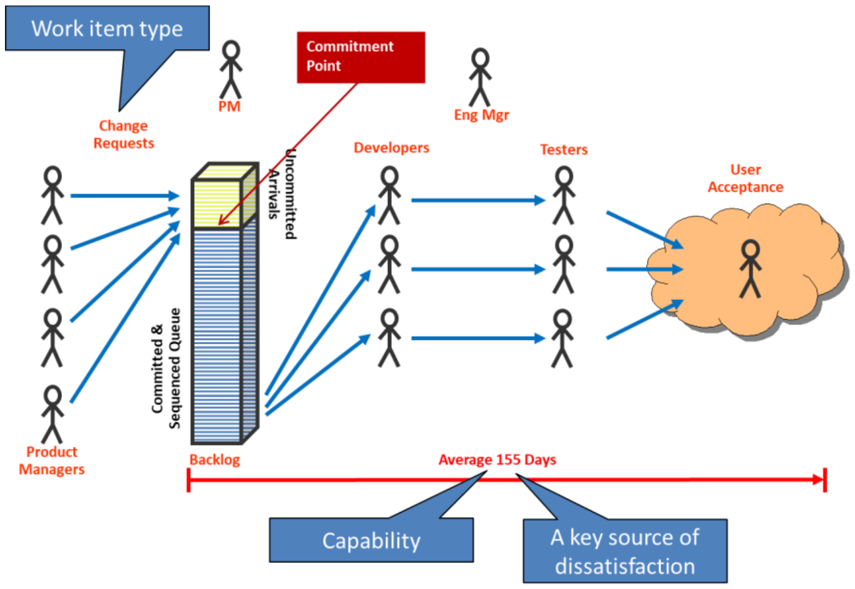


Figure 2.2 XIT Sustaining Engineering Workflow

## Demand & Capability Analysis

When each request for a change or defect fix arrived from a product manager, Dragos would send it to India for an estimate, as illustrated in figure 2.3. The policy was that estimates had to be made and returned to the business owners within 48 hours. This would facilitate making some return-on-investment (ROI) calculation and deciding whether to proceed with the request. Once a month, Dragos would meet with the product managers and other stakeholders, and they would reprioritize the backlog and create a project plan from the requests.

Due to the service level agreement to return estimates within 48 hours, they preempted existing planned work already in-progress. Effectively, gathering information for future speculative work, was treated with greater urgency and importance than completing existing planned and committed work.

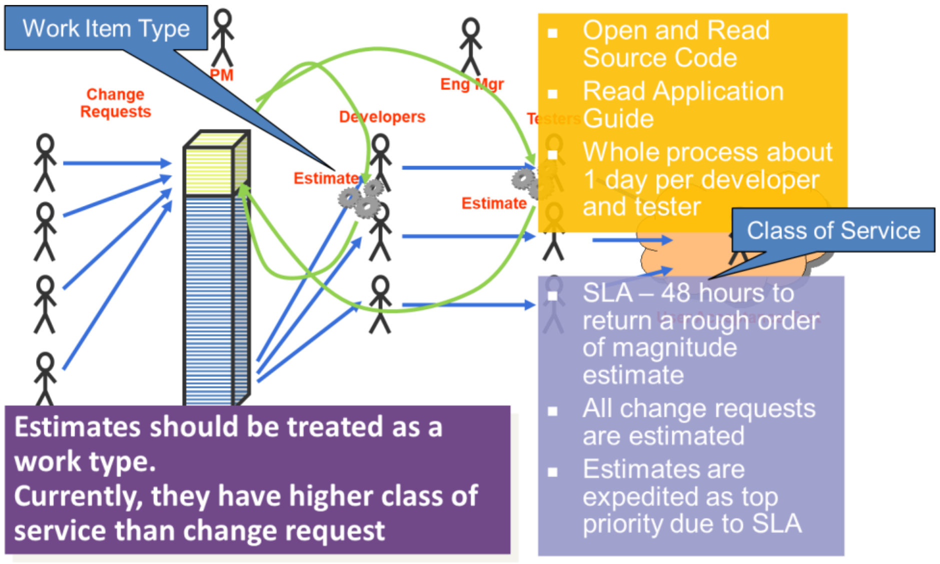

Work Item Type
PM
Change Requests
Eng Mgr
Developers
Testers
Estimate
Estimate
Open and Read Source Code
Read Application Guide
Whole process about 1 day per developer and tester
Class of Service
SLA – 48 hours to return a rough order of magnitude estimate
All change requests are estimated
Estimates are expedited as top priority due to SLA
Estimates should be treated as a work type.
Currently, they have higher class of service than change request

Figure 2.3 Illustrating how requests for estimates disrupted planned work

The estimates for new incoming work were consuming a lot of effort. Despite being referred to as “rough order of magnitude” (ROM) estimates, the customer expectation was actually for a very accurate estimate, and team members had learned to take great care over preparing them. The root cause of this was that estimates were used both as input to ROI calculations and hence prioritization decisions, but also as a means to cost a request, for the purpose of interdepartmental budget transfers.

Bizarrely, payment for the work done by XIT Sustaining Engineering was made based on the estimate rather than the actual time spent doing the work. It appeared from our analysis that Microsoft corporate functions were effectively "paying" XIT in advance for each request. While this seemed truly bizarre, we decided to let it go unchallenged. You need to pick your battles, and a candid discussion with a VP of Finance who had made the policy decisions on how sustaining engineering work should be accounted for, i.e. as operating expense, and how it might be paid for from requesting business unit budgets, was not something that either of us relished. We didn't have the pay grade or the respect within the firm to even suggest such a meeting. Like the contractual requirement to use the Software Engineering Institute's PSP/TSP methodology, the finance policies were, at least for us, an immovable object, a constraint around which we had to work. We had to be successful despite these rules. It wasn't acceptable to point the blame at them and wash our hands of further responsibility.

Each of these high precision estimates was taking about one day for each developer and tester to produce. The fear of getting it wrong was driving them to do analysis and design work just to develop an estimate. Of course, this analysis and design work was thrown away, and not preserved such that it might be re-used later.

At this time the requests for estimates ranged between 18-25 per month. We quickly calculated that the estimation effort alone was consuming 7 or 8 working days per person each month. Consequently, 33 to 40 percent of capacity was being used to assess the viability of uncommitted work. At least one third of capacity was used to speculate about future work, in preference to working on coding and testing for current committed work. There was no governance over the number of requests for estimates and hence the impact of estimating was potentially unlimited. Estimating these new requests pre-empted existing work and caused delay. Given that it was ungoverned, it had the potential, though apparently it had never happened, to completely halt all current committed work. Consequently, estimating randomized plans made for any given month, and resulted in work being completed behind schedule. In fact, demand for estimation was sufficiently high, and its impact pre-empting committed work so great, that XIT Sustaining Engineering were incapable of delivering anything against plan. Their currently delivery capability was effectively 0% on-time.

While the demand for estimates was 18-25 per month, the number of items being delivered was around 6 per month[2], as shown in figure 2.4. The backlog which had 80 or more items in it as of October 2004 was growing though not as quickly as it should be, if we looked at the demand for estimates. What was going on?

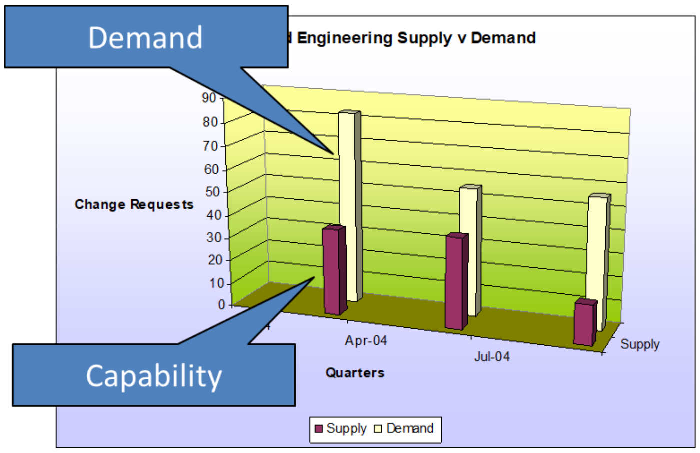


Figure 2.4 Demand for change requests versus capability to supply over the previous 9 months

A study of all items closed, whether complete, discarded or abandoned mid-flight, showed that 48% of requested submitted were never actually delivered. This explained why the backlog wasn't growing as rapidly as might be expected. Nevertheless, the growth was typically greater than 6 per month. Of the items never delivered 26% represented items discarded during planning because they had a poor ROI, or because they were "too big". Items estimated as greater than 15 days of engineering had to be redirected to a major project in the portfolio in order to be accounted for as capital expense. This governance rule was intended to enforce the notion that maintenance work, accounted for as operating expense, was only ever for small items. Historically, the "too bigs" were only 2% of demand. Hence, low ROI represented 24% of demand. The remaining 22% were abandoned and closed with the reason "overtaken by events". This was often caused by the decommission of an application or intranet site. For example, in 2003, there was a huge earthquake and subsequent tsunami off the coast of Sumatra, Indonesia. The tsunami wave took the lives of over

[2] In 2 of the 3 quarters shown in figure 4.3 the delivery rate is approximately double at around 12 per month. This gives a false impression of capability. During this six month period, Microsoft management had doubled the staffing level in an attempt to reduce the backlog and allow TCS taking over as the vendor to start their contract with a relatively small backlog. From July 2004, staffing had returned to historical levels and the delivery rate returned to similar historical levels of approximately 6 items per month. Unfortunately, we do not have a chart of this earlier period.

250,000 people in Indonesia, Thailand, Sri Lanka and South East India. At the time, Microsoft had created a website to enable employees to make donations and these were distributed to charities such as the Red Cross. This site was no longer required some 18 months later, and it was decommissioned. Other such examples, were often season in nature or for one-off events.

We can summarize our demand and capability analysis as follows:

- Speculative work requiring an estimate and business case 18-25 per month
- Actual committed work, planned and sequenced 9-13 per month
- Work actually delivered each month, approximately 6 items

While we can summarize delivery capability as

- lead time was on average 5.5 months and growing at a rate of at least 0.5 months per month
- Items delivered against original planned and committed dates was approximately 0%

While only 6 or so items were delivered each month, the entire backlog was reprioritized and re-planned each month. While approximately 12 items were discarded or abandoned, a similar number were committed, sequenced and added to the plan implemented as a Gaant Chart in Microsoft Project. In Kanban language, work was committed early, at the monthly planning meeting after the request was submitted. At the time of commitment, each item would have a proposed delivery date. Around 6 items would be delivered over the next month. However, the committed backlog would be at least 80 items. New requests would have arrived meantime, and all the undelivered work would be reprioritized at the next monthly planning and the new plan with new dates for each item, recommunicated to stakeholders. It was likely that a typical request would be replanned 4 or 5 times prior to delivery. This was a key factor in customer dissatisfaction and a lack of trust in the sustaining engineering service. They were simply not capable of keeping their promises.

The problem was two-fold: they were committing too early, to too much; and even for the immediate month ahead, they were over-committing because the disruptive effect of the requests for estimates wasn't be taken into account.

## Flow Efficiency

The requests were tracked with a tool called Product Studio. An updated version of this tool was later released publicly as “Team Foundation Server Work Item Tracking.” The XIT Sustaining Engineering team was similar to many organizations I see in my teaching and consulting work—they had lots of data, but they were not using it. Dragos began to mine the data and discovered that an average request took 11 days of engineering (a combination of development and testing time) as shown in histogram in figure 2.5. However, lead times of 125 to 155 days were typical. More than 90 percent of the lead time was queuing, or other forms of waste. They were only 8% flow efficient. While this sounds very poor, we've come to realize that often the starting condition for improvement is well below this. Hakan Forss[3] and Zsolt Fabok[4] have both reported starting flow efficiency numbers of 1-2%. These numbers are widely accepted amongst the Kanban coaching community as typical.

[3] Hakan Forss, Lean Kanban France, Oct 2013

[4] Zsolt Fabok, Lean Agile Scotland, Sep 2012, Lean Kanban France, Oct 2012

The good news, whether we have 8% as we did in 2004, or a much lower number, is that there is a huge potential upside. Improving flow efficiency and dropping lead times, should be a matter of identifying and eliminating sources of delay.

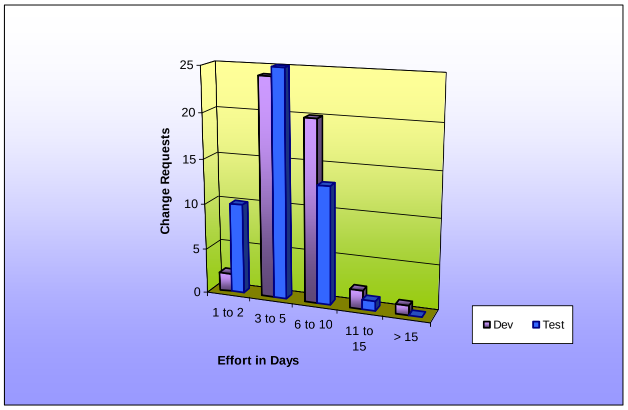


Figure 2.5 Histogram showing actual development & testing time per change request

## Additional Work Item Type

In addition to change requests and defect fixes, there was an additional type of work, known as production text changes (PTCs). The origin of these was text changes to on-screen dialog boxes or web pages. This had grown to include graphical or web page design changes and eventually expanded to involve modifying values in tables used to drive business logic in applications or XML files used for configuration or reference while an application was running. We later discovered that for example, the tax tables, for the payroll system, which contain the income tax deductions to be withheld by the employer, fell into this category of work. The acronym PTC was meaningless! The common element was that these changes did not require a developer and were often made by business owners, product managers, or the program manager, but they did require a formal test pass, so they affected the testers. PTCs all had a common workflow. However, their nature, the business risks associated with one item compared to another varied greatly: changing a department logo on an intranet web page clearly does carry the same risks as deploying the new withholding tax tables on the payroll system. And this was another problem, PTCs were given their own class of service. They were all "top priority", effectively, expedite requests. At the time, we didn't understand why. We didn't have enough insight into the true nature of PTCs. Inherently, it just seemed wrong. Why were "text changes" being expedited? It was a red flag but at the time we chose to ignore it. All we knew was that PTCs were disruptive, that they pre-empted planned and committed work, and they affected our ability to deliver that planned work on-time.

I asked Dragos about the nature of arrival of PTCs and the volume of demand. His response was that they were unusual and weeks would go by without a single request then without much warning a whole batch would arrive. There sporadic nature and demanded class of service made PTCs a problem. A problem that today we teach Kanban practitioners the skills to understand and design for adequately. In 2004, we ignored them. As you will learn in the next chapter, we got away with it for two reasons: because the improvement in general performance was so great that there was capacity to cope with PTCs; while their arrival impact was much less disruptive, merely a ripple because of the

WIP control and deferred commitment benefits from using a kanban system.

## From Understanding a Problem to Designing a Solution

Now that we understood the problems and the constraints within which we had to work, the focus turned to what we might do about it. You'll learn what Dragos chose to do, and how he enabled it to happen in chapter 3.

## Takeaways

The first known and documented kanban system for intangible goods, professional services was implemented with the XIT Sustaining Engineering software maintenance team at Microsoft, starting in 2004.

It used an electronic tracking tool. Sometimes this is referred to as a “virtual kanban system.”

It was implemented with an offshore team at TCS in Hyderabad, India.

Workflow for a service should be sketched and visualized.

The process should be described as an explicit set of policies.

Demand & capability analysis should be conducted

Sources of customer dissatisfaction should be identified and understood

Analyzing flow efficiency helps us to understand the potential improvement possible

Typically flow efficiency is very low, e.g. 5% or less before any interventions are made

# CHAPTER 3

# "Do you think they'll go for that?"

“So, our proposal is that we’re going to stop estimating, and stop planning, and ask them to trust that this will magically result in everything being delivered within 30 days?”
“Yes! Do you think they will go for that?”
Dragos and I looked at each other across his office, in building 115 of Microsoft’s campus in Redmond, Washington. It was a dark, dreary, cloud-covered, rainy day in the fall of 2004.
“No. Probably not!”

This is the story of what Dragos did and how he got it done. The results are now legendary. The delivery rate of change requests jumped by 230%, while lead times fell from an average of 5.5 months to a mere 12 days, and on-time performance rose to 98% against a 25-day service level agreement. Dragos was promoted then later headhunted when Dale Christian moved from GM of XIT to the position of CIO at Avanade. In two moves, in two years, he was now Senior Director for Global IT Operations of the Accenture/Microsoft joint venture company, from his position as program manager of a 6-person team, only two pay grades above a university graduate at Microsoft. The XIT Sustaining Engineering service team moved from having the worst customer service record within Microsoft’s IT group to the best, while Dragos was rewarded with the division’s process improvement award for the 2nd half of 2005.

## How Policies Affected Performance

The team was following the required process that included many bad policy decisions that had been made by managers at various levels often in isolation and without due consideration for the wider impact on the service as a whole. It is important to think of a service and its workflow as defined by a set of policies that govern behavior. Someone has the authority to override or change policies. They are under management's control. For example, the policy to use PSP/TSP was set at the executive vice-president level, one rung below Bill Gates, and this policy would be hard or impossible to change. Policies on accounting and budget transfers were made by a mid-ranking executive in the Finance department and these policies would also be difficult to change. Policies on prioritization and the use of ROI calculations in businesses cases were made by the PMO and required of product managers. While not impossible to change, neither Dragos nor I had the pay grade or the influence to affect change there. However, many other policies, such as the policy to prioritize estimates over actual coding and testing, were developed locally and were under the collaborative authority of the immediate managers. It is possible that these policies made sense at the time when they were implemented; but circumstances had changed, and no attempt had been made to review and update the policies that governed the team's operation. There was scope to change some policies and affect improvements in performance despite the other constraints.

## No Estimates

After some discussion with his colleagues and manager, Dragos decided to enact two initial management changes. First, the team would stop estimating. He wanted to recover the capacity wasted by estimation activity and use it to develop and test software. Eliminating the schedule randomization caused by estimating would also improve predictability, and the combination would, he hoped, have a great impact on customer satisfaction.

However, removing estimation was problematic. It would affect the ROI calculations, and customers might worry that bad prioritization choices were being made. In addition, estimates were used to facilitate inter-departmental cost accounting and budget transfers. Estimates were also used to implement a governance policy. Only small requests were allowed through system maintenance. Larger requests, those exceeding 15 days of development or testing, had to be submitted to a major project initiative and go through the formal program management office (PMO) portfolio management governance process. We will revisit these issues shortly.

Estimates were disruptive and affecting ability to deliver against promised dates. The lack of predictability was affecting customer satisfaction. Had Dragos chosen to fix this one issue of predictability then perhaps he would have made a different choice. Deciding not to estimate was a choice made to give back at least 1/3rd of capacity spent on it and improve predictability. There were actually four choices available to us for consideration: stop estimating; time slice estimation activities, separately from value-added committed work delivery; isolate estimation with a specialist role of "estimator"; develop a hybrid system of passing a specialist estimator role around from one team member to another with a fixed cadence such as weekly. Let's consider each of these approaches in turn…

- To stop estimating altogether is the most radical choice. It requires that we introduce a service level agreement. This gave back wasted capacity, but it required a new agreement, a new contract, with the customers. It is the boldest choice.

- The time-slicing approach of containing estimating, prioritizing and planning, into a fixed time period then task switching between customer-valued work, and such planning, is the approach used in the Agile software development lifecycle methodology, Scrum. To have made such an approach work at XIT, Dragos would have needed to allocate 8 days per month for estimating and planning, and then spend the remainder of the month coding and testing. This approach would have improved predictability and helped dramatically with customer satisfaction, but it did nothing to address the approximately 1/3rd of capacity being sucked away by estimation effort.

- The choice of assigning a specialist could also have worked quite nicely in this case. Dragos could have informed the local manager at TCS in Hyderabad, that one of the developers and one of the testers were to be permanently assigned to analysis and design in order to provide estimates. A simple policy change! This would have prevented the other two developers and two testers from being disrupted and resulted in significant on-time delivery improvement. However, it would also have made it abundantly clear that 1/3rd of capacity was being used for estimating.

- The third option of passing the estimation responsibility around from one team member to another, on a weekly basis, might have been more acceptable to the team than assigning a specialist but it still didn't recover the capacity being wasted on estimation.

Only the choice to stop estimating altogether freed up capacity. While customers were unhappy about unpredictable, unreliable delivery and broken promises, they were also unhappy about lead times. Lead times were growing because demand exceeded capability to supply. As a consequence, there was a need to produce more. Recovering 1/3rd of capacity was a means to produce more and directly address the growing backlog and lengthening lead times. It presented an interesting trade off, in exchange for replacing individual estimates and delivery date promises with a service level agreement (SLA), 50% more work would be completed and there was some chance that the growing backlog could be tamed with long delivery times brought under control, such that product managers and sponsoring customer departments like HR would find it fit-for-purpose.

Choosing not to estimate, as part of the changes made at XIT Sustaining Engineering was a choice made because of specific circumstances, and it was a choice made while considering 3 other options. Any of the other options were possible and viable and would have contributed to fixing a significant issue with customer satisfaction. Regardless of which choice we had made, we would still have been using a kanban system. This story would still have represented the prototype for what developed into the Kanban Method.

In the early days, Kanban was often cited as the “no estimates” method. This created some fear and trepidation amongst a traditional project management audience, while raising tribal hackles in the Scrum[5] community who had ritualized their Planning Poker and other estimation techniques. Choosing whether to estimate or not should always be a consideration for the policies that define a class of service. The risks associated with work should always determine whether it is better to proceed with what you know, or delay to gather additional information before making a commitment. A request for an estimate is a request for information speculating about the cost or time required to complete a piece of work. This information may be useful for risk management in some situations, where in others, it makes little difference to the good governance of the entity and can therefore by avoided. With XIT, their customers were used to consuming IT services, defined by service level agreements (SLAs). As a consequence, Dragos was in a position to make his customers a straightforward offer and trade, “if we switch to an SLA with a defined lead time expectation, then in return, we will deliver you around 50% more completed requests each year.”

## Limit Work-in-Progress

Dragos also decided to limit work-in-progress and pull work from an input buffer as current work was

[5] Scrum is an example of a generic class of prescriptive processes known as Agile software development methodologies. In software engineering, a methodology is defined as a description of a process workflow, together with a defined set of roles to be played and the responsibilities those roles carry in the execution of the work.

completed. The input buffer was sized to anticipate the maximum delivery rate within the one week period between replenishment meetings, i.e. it was just big enough to insure that the developers were never starving for work and consequently never idle. He chose to limit WIP in development to one request per developer and to use a similar rule for testers. As the Personal Software Process (PSP) already recommended this practice, it was in fact, the policy already in use. He inserted a small buffer between development and test in order to receive the PTCs and to smooth the flow of work between development and test, as shown in Figure 3.1. This approach of using a buffer to smooth out variability in size and effort is discussed in chapter 19[6]. The buffer was arbitrarily sized as 5. We didn't know how big to make it so we made a guess and decided to empirically adjust its size as we observed how it worked. If a large batch of PTCs arrived, it is likely the buffer would overflow, having the effect of stalling upstream development work. Developers would have to wait until the testers cleared the batch of PTCs and kanbans became free in the buffer to allow finished development work to flow forwards.

Note: This is a policy choice. One change request per developer at any given time is a policy. It can be modified later. Thinking of a service as a set of policies is a key element of the Kanban Method.

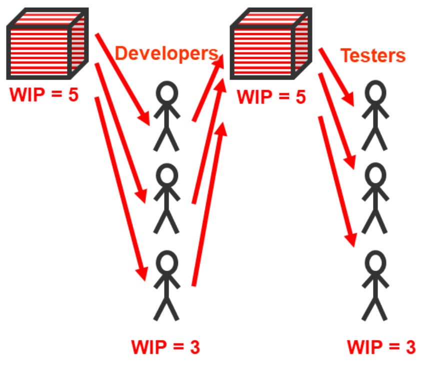


Figure 3.1 a kanban system for the sustaining engineering workflow

## No Planning

The monthly planning meeting was to be abandoned and replaced with a more frequent meeting to replenish the kanban system. There would be no more Gaant charts and no more early commitment to everything in the plan. The backlog of requests would remain uncommitted until an item was pulled into the kanban system at the replenishment meeting by consensus agreement of the four product managers and Dragos as program manager. Dragos had to think about the cadence for interacting with the product managers. He thought that a weekly meeting to replenish the kanban system would be feasible. It was planned as a conference call, as the topic of the meeting, would be the simple replenishment from the backlog, of empty slots, free kanban, in the input buffer. In a typical week there might be three slots free in that buffer. So, the discussion would center around the question, "Which three items from the backlog would you most like started next for delivery

Software engineering methodologies describe which role performs which function, who they collaborate with, who carries responsibility and accountability and how work is handed off from one individual or team or collaborators to the next. Often methodologies have very specific and detailed guidance on techniques to be used for specific activities.

[6] Of the 1st edition

within 25 days?" It's a simple question and it should facilitate a short meeting. This cadence is modeled in Figure 4.5.

Note: *Cadence* is a concept in the Kanban Method that determines the rhythm or frequency of an event. Planning, prioritization, delivery, retrospectives, and any recurring event can have its own cadence.

He wanted to offer a "guaranteed" delivery time of 25 days from commitment - the point when a request was accepted into the kanban system and placed in its input buffer. This 25 days service guarantee was considerably greater than the 11 days of average engineering time required to complete the job. The statistical outliers required around 30 days, but he anticipated very few of them; 25 days sounded attractive, especially compared to the existing lead time of around 140 days. He expected to hit that target with regularity, building trust with product managers and their customers as he went.

So traditional planning with a Gaant chart, with anticipated start and end dates for each request, and therefore specific promised delivery dates for each item, were to be discarded and replaced with a simple service level agreement with a 25 day or less service level guarantee on lead time from commitment to delivery.

Sitting in Dragos' office, we had an understanding of the problems and a design for a solution. We looked at each other, Dragos was giggling,

"So, our proposal is that we're going to stop estimating, and stop planning, and ask them to trust that this will magically result in everything being delivered within 30 days?"

"Yes! Do you think they will go for that?"

"No. Probably not!"

It was going to take more than a strong logical argument to get people on board.

## Who might object and why?

Let's consider each of the changes in turn and think about how the proposal might be received in isolation.

First, we propose to stop estimating. Developers and testers find estimation disruptive and it impacts their ability to good, high quality work. Also, they are professionally qualified in software development and testing, they have degrees and certifications in this subject. No one ever asked them to study, sit an exam or acquire a certification in estimation. An ability to estimate is not core to their identity or how they derive their professional pride or self-esteem. If we tell the team in Hyderabad that we no longer require them to make estimates, they will celebrate.

Next there is the program manager who facilitates making the plans and own the plan constructed in a Microsoft Project Gaant chart. If we tell the program manager that estimates will no longer be produced and that Gaant charts are no longer required, there is likely to be some resistance. It is highly likely that the program manager as a self-image of project manager, and many such people in that position were members of professional organizations such as the PMI[7] and held credentials and qualifications such as the PMP[8] for which they had studied and passed an exam. To suggest that we remove the practice of planning and the production of a Gaant chart from these people, would likely be interpreted as an attack on their identity, a show of disrespect, and an indication that their skills and hence, they, were no longer valued. However, in this case, the program manager was Dragos, the former Olympic athlete, stuntman, bodyguard, and psychiatric hospital manager. He wasn't vested in any of this professional project manager identity. And so we got lucky – Dragos was the change agent, he was the instigator, not someone objecting and being obstructive. Had this not been the case, it could all have died there and then. Perhaps we wouldn't have Kanban as a management method adopted globally? Perhaps there would never have been a first edition of this book, or any other book on the topic.

Lastly, we have the product managers. There role had three main aspects to it: managing the budget on behalf of their customer and business owners; assisting customers to elaborate requirements and performing business analysis; providing good governance over the budget by building business cases and prioritizing the work based on optimizing the return on investment.[9] The equation used was as follows

$$\text{ROI} = \frac{\text{Business value}}{\text{Cost}}$$

$$\text{Cost} = \text{hour rate} \times \text{estimated engineering hours}$$

Without an estimate, there would be no value for the denominator in the ROI equation and consequently it would be impossible to calculate. Stopping estimation denied the product managers the ability to complete their business cases and to perform their prioritization function by stack ranking requests by ROI[10]. And this is a key reason, why we believed "they" would go for it.

Our second proposal was to implement a kanban system to pull work from an uncommitted backlog. Rather than making early commitment, we intended to defer commitment.

In this case, the developers and testers are unmoved and unaffected by the change. So, we wouldn't expect any resistance. And once again, Dragos was the program manager and the change agent. However, this was a change for the product managers and their customers in each respective business unit. They were used to being given firm plans within a couple of weeks of submitting a request. However, they were also used to the idea that the plan was worth nothing and that the sustaining engineering team never delivered anything when promised. Our approach to this first to show them the abandoned and discarded data. Only 52% of requests were ever delivered. Why commit to all of them when 48% of them never made it into production? This technique has proven powerful and persuasive in many implementations since then. It is particularly persuasive when you

[7] Project Management Institute

[8] Project Management Professional

[9] This method of prioritization intended to maximize return on investment is described in the Project Management Body of Knowledge published by the Project Management Institute and it is widely adopted globally as the standard way to prioritize professional services, and knowledge worker activities.

[10] Note: This was usually performed in an Excel spreadsheet using the column sort function. ROI was a simply ratio achieved by dividing two numbers. Business value it was assumed could be reduced to a simple dollar amount. This is actually standard practice in product and project management.

have those affected do the data mining and discover for themselves just how many requests they make are never implemented. Sometimes it is necessary to put a definition, and explicit policy on “abandoned”. What does it mean to be abandoned? If a request is older than 6 months, or 12 months, or 13 months, or 2 years then is it abandoned? Where is the organization’s tolerance and threshold for, “if we haven’t gotten to it yet, we probably never will.” When you make this data explicit, it becomes hugely powerful.

The change to a pull system was coupled to the adoption of a service level agreement, effectively aggregating delivery risk across all requests rather than making fragile, individual commitments based on speculation. The business units were used to consuming other IT services defined by a contract, a service level agreement, including guarantees on delivery lead times. So, in this case, they were being asked to switch modes and see this work different. Rather than see it as a series of mini-projects, instead they should view it as a continuous service. This argument appeared to work and raised little objection. Things had been broken for so long, why not try this alternative, yet familiar approach?

Lastly, we proposed to stop planning. Once again, this made little to no difference to the developers and testers. They were used to picking up their work from a sequence defined in a project plan. Instead, they would pick up their work from a buffer, defined in their existing tracking tool, Product Studio. And once again, the program manager was Dragos, so no resistance from him. What about the product managers? From their perspective they were being asked to attend a weekly replenishment meeting rather than a monthly planning meeting. Other aspects of planning which they owned such as preparing business cases, and producing a prioritized backlog were unaffected, assuming there could be resolution to the ROI calculation conflict.

Actually, the planning meetings were long and laborious with a large plotted Gaant chart on the table being marked up in pencil. These meetings weren’t anyone’s idea of fun and they took many hours. A short, 15-20 minute, call once per week, sounded like significant relief, assuming everything else worked effectively and they continued to look professional, competent and effective in their roles.

Finally, we haven’t considered Dragos’ upline managers. What did they think?

By all accounts, Dragos’ immediate manager had misgivings and was fearful of the consequences. The next two layers up simply died laughing, “your going to stop planning and stop estimating and everything will be fine???” Once, they’d sobered up a bit, they were able to reason about it, “This service has been broken for a long time. A succession of former managers has been unable to fix it. Moving it offshore didn’t fix it either. This sounds utterly crazy but we put you in the job to make changes. If these are the crazy changes you want to make then at least we should give them a try.” So, the senior management were prepared to hold their breath and wait and see.

However, there was still the issue of how to enable the product managers to continue making their business cases, and prioritization decisions without an estimate. The solution to this was the spark of genius that together with Dragos’ diplomatic skills and personality were to enable the first kanban system implementation at Microsoft.

## Shuttle Diplomacy

Dragos arranged to visit each of the product managers in their offices individually, and then their immediate manager. He wanted to get each of them on-board with our proposal without the influence of their peer group or social pressure causing them to close ranks and conservatively stick to their existing modus operandi. If he could get them on-board individually then he'd hold a group meeting for the official kick-off of the change initiative and the rollout of the kanban system.

Along with the basics of a sketch of the workflow and the proposed kanban system, figure 3.2, together with a description of the replenishment meeting, Dragos brought the chart showing the distribution of engineering effort for requests over the past year, previously shown in figure 2.5 and repeated here for your convenience as figure 3.3.

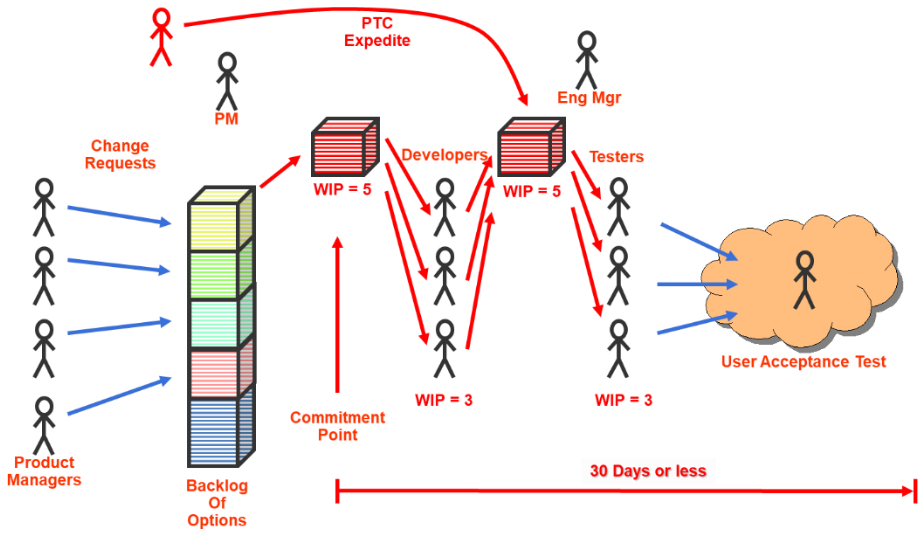


Figure 3.2 The full solution proposed for the XIT Sustaining Engineering workflow

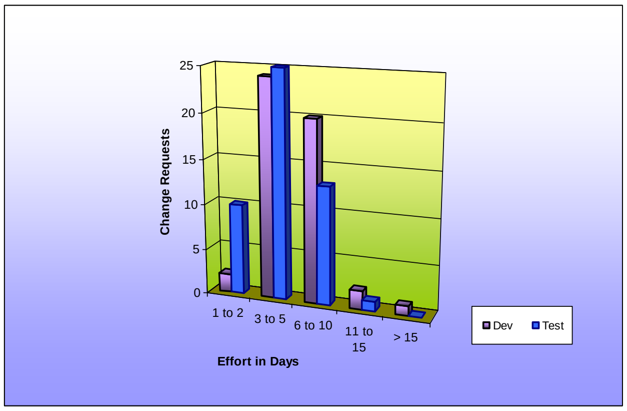


Figure 3.3 Histogram showing actual development & testing time per change request

Dragos suggested to each product manager that the distribution of effort was within a relatively narrow range and most requests fell within the range 3 to 10 days of development and test with a mean of just under 6. Given the volume of requests and that we fully expected this volume to increase dramatically, that it was reasonable to replace a specific, deterministic, though still speculative estimate, with an average extracted from recent historical data based on actual hours spent. An average of actual hours spent is a fact where an estimate for any individual item is merely speculation.

$$\text{ROI} = \frac{\text{Business value}}{\text{Average Cost}}$$

Essentially, if the product managers were willing to accept that cost varied within a tight range and effectively ignore this variation, then all the other benefits of improved productivity and predictability could be enabled. We weren't asking them to change anything about their own job, and how they were working. Their identity, self-esteem, social status, respect with the organization, and professionalism wasn't being called in to question or threatened in any way. Instead, we were asking them simply to accept average cost as a fact, and its value as good enough to enable them to make effective prioritization decisions.

Actually, this technique works well when there is significant asymmetry in the problem. When all business values, significantly outweigh, any cost incurred, then the outcome of a stack ranking of the ratios is not very sensitive to variation in the cost. Cost can effectively be ignored. Where cost estimates have true value is when this asymmetry doesn't exist, and costs are actually relatively close to the payoff labeled "business value." Ironically, this condition of relatively symmetric payoff and cost is quite common in IT systems for shared service and back-office functions such as finance and human resources. So, estimates of project costs are important when governing an IT portfolio for back-office systems. However, with system maintenance and sustaining engineering, small requests often have a huge impact - such as deploying the tax tables for the new fiscal year - and hence the asymmetric payoff requirement is most certainly met in this case. In 2004, however, we weren't that sophisticated, and neither were the product managers. They all bought the argument and agreed.

The game was on! Kanban was green lit for deployment at Microsoft's XIT Sustaining Engineering department. It was October 2004.

## Implementing Changes

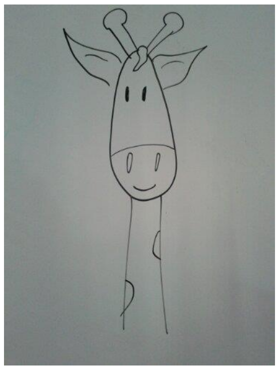
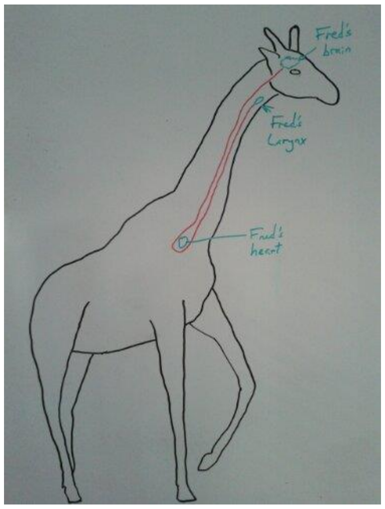
So, the changes were enacted. Dragos had their instance of Product Studio instrumented using stored procedures in its database to enforce the kanban system WIP limits and signal free slots and the ability to pull work using database triggers that sent automated emails. He cancelled the monthly planning meetings and scheduled weekly conference calls to replenish the kanban system. New work requested were no longer sent to Hyderabad for estimation.

It began to work. Requests were processed and released to production. Delivery times on new commitments were met within the 25-day promise. The weekly meeting worked smoothly, and the input buffer was replenished each week. Trust began to build with the product managers. Customers began to see requests deployed to production quickly and within the promised SLA.

## Evolutionary Relics

An evolutionary relic is something left behind by evolution that no longer serves any purpose but for which there isn't a mechanism to remove it. Such relics appear in our own bodies. Our Coccyx bone on the end of our spine is the obviated connector for a tail, and our appendix is left over from an herbivore species from which we evolved into modern humans. There is some argument that our gallbladder may be a similar relic. It seems we aren't quite sure what it is for, but just like our appendix, if it goes wrong it can be rather serious and life-threatening. Evolutionary processes leave behind artifacts and behaviors that are hard to explain and serve no purpose.

Paul Klipp, an American from Chicago, living in Krakow, Poland and founder of Kanbanery, a kanban software tool, explained the concept on his blog[11] on March 6th, 2013, after attending the Kanban Coaching Professional (KCP) Masterclass,

> *I'll enlist the help of a giraffe. His name's Fred:*
>
> 
>
> *Like all mammals, Fred has a larynx controlled by his brain, and Fred is the product of evolutionary change. Fred's larynx is just inches from his brain, because it's at the top of his neck and so is his head, as you might expect. Fred's getting impatient, so he bellows for me to get to the damn point. His brain got impatient first and sent the impulse to bellow right down that nerve to his larynx. A short trip? Not really. Silly evolution decided that the best way to route a nerve between one thing on the top of his neck and another thing on the top of his neck was to wrap it around his aorta first.*
>
> *Here's Fred's laryngeal nerve; it's about 15 feet long:*
>
> 
>
>
> *Now who decided THAT was a good idea?*
>
> *That's where evolution gets you. It's a hell of a lot better than being a fish, at least from the giraffe's point of view, but the evolutionary path from fish to giraffe has some constraints. The corresponding nerve in a fish makes sense. A straight line between a fish's brain and its gills passes the heart, so the nerve crossing behind the heart is pretty sensible. Here's the thing, though. Evolution starts with the existing processes and systems and changes them incrementally. Re-routing a nerve is not an incremental change; it's a revolutionary change.*

If true evolutionary processes are at work, awkward solutions evolve over time. You wouldn't intentionally design a nerve to run down a giraffe's neck and back up again. It isn't logical or efficient, but it is robust. The concept of "survival of the fittest" in evolutionary biology indicates that a solution was fit for its environment. For us, we seek to evolve fit-for-purpose business services. Being fit-for-purpose is likely to indicate an ability to survive and continue. The ability to respond to stress in the environment and continuous evolve to remain fit for the ever-changing environment is what Nassim Nicholas Taleb labeled anti-fragility. Kanban as a means to wire a business with evolutionary DNA provides a means to anti-fragility.

[11] http://paulklipp.com/blog/evolutionary-change-better-than-a-kick-in-the-nuts/

Meanwhile, if you walk into a company and everything is too neat and tidy, all of the process are efficient, lean and devoid of artifacts or activities that seem to serve little purpose, have little or no value and may have been obviated by circumstances of new techniques, then you are looking a designed environment - the process consultants have been in, designed a new process, installed it, perhaps through the use of position power, and then left. These designed solutions are fragile and the businesses using them are fragile. Why?

When resistance is overcome using positional power, it is highly likely that employs were acquiescing while their behavior is actually passive-aggressive. When management attention is turned to something else, they'll quietly revert back to the old ways. They had no ownership in the changes, and they haven't internalized it. It hasn't become "how we do things around here." It isn't part of their identity individually or as a group. Evolutionary change is robust, while design and managed change is fragile.

## Prioritization - the evolutionary relic at XIT Sustaining Engineering

Returning to Dragos' story, we'll recall that he wasn't asking the product managers to change how they were working, they would continue to make business cases and calculate ROI using their own estimate of business value, and the IT engineers estimate of costs. They would continue to column sort their spreadsheet to provide a stack ranking of change requests from highest to lowest ROI. They had accepted the viability of using an average value for cost, effectively meaning that all change requests were ranked by their business value.

Meanwhile, they'd bought into deferred commitment, they had no objection to switching to weekly replenishment meetings and dropping the time-consuming monthly planning meeting. ***"So, let's all do the Kanban!"***

However, as soon as we start with Kanban, their prioritization work will instantly become an evolutionary relic. Why? At a replenishment conference call, they may be asked to, "pick the one item you'd most like for delivery within the next 25 days." This isn't a request for the item with the highest return-on-investment, rather it is a request based on urgency or timeliness. An item deemed important but perhaps not with the highest ROI, is likely to get selected. For example, "support Puerto Rican address format for employee information form within the employee records application". This isn't a request with a particularly high ROI. How do we even calculate the "business value" of such a request and put a dollar value on it? Even if we do cook up some method to devise a number, it's unlikely to produce the highest ROI. And yet, it will get picked! Why? Because the Puerto Rican office is planned to open at the end of next month and we will need to be able to record details of the new employees hired for that office.

Kanban replenishment questions are about urgency and timeless, not return-on-investment. Product managers may have a spreadsheet filled with data and stack ranked and column-sorted by the ROI calculation, but when it comes to the crunch, and making a decision during the replenishment conference call, they'll find the item they most want "for delivery in 25 days or less," will not be the item in row 2 of the spreadsheet. Consistently, they'll find their top picks are coming from further down their list.

Their efforts to prioritize have been obviated. They are now selecting items from their pool of available options based on the cost of delay of those items. What is the cost of delaying the new Puerto Rican office because we can't on-board the employees. Cost of delay isn't the same as return-on-investment. Effectively both methods are now in use: cost of delay; and return-on-investment. One has obviated the other. The practice of calculating return-on-investment has become an evolutionary relic.

This approach of leaving (some) existing practices in place while introducing new practices to replace them is standard technique in applying evolutionary change. Effectively, ROI and Cost of Delay are two species for the purpose of prioritization, or to use less ambiguous and more precise language, sequencing of work. These two methods, the incumbent and the insurgent, will compete, like two biological species compete to be fittest for the environment.

Imagine that ROI is the red squirrel, while cost of delay plays the role of the grey squirrel. Across Europe and the British Isles, the two species will compete. Red squirrels, native to most of Eurasia with a range from Portugal to Mongolia and including the British Isles, have been in decline in Europe for centuries. Initially, through hunting and shrinking habitat, their decline has been rapid in the latter half of the 20th Century. This accelerated decline is widely due to the arrival of grey squirrels. The confusingly named "Eastern Gray", is actually an invader from North America where it takes its name from its territory and the extent of its range along the eastern seaboard of the continent and inland to the mid-west. Officially, it is recognized as having been introduced to the British Isles in 1948 though there are examples of earlier introductions, perhaps the earliest by, Herbrand Russell, the 11th Duke of Bedford[12]. He introduced greys from New Jersey to his Bedfordshire home, Woburn Abbey, in 1890. Starting from the south of England, greys spread throughout the British Isles and to mainland Europe.

Grey squirrels are significantly larger and more aggressive, particularly during springtime, and they compete with the reds for nuts and other food. However, the bigger issue is that greys are carriers of squirrel pox virus. This disease is deadly to red squirrels. The grey is in effect creating the reverse effect of the conquistadors from Spain arriving in Central America where up to 90% of the population perished from European diseases carried by the invaders. Red squirrels have been in severe decline. In the UK most of the population is now isolated to Scotland and Cornwall, while in Europe, red survive mostly on the Iberian Peninsula and appear to have been protected, thus far, by the Pyrenees mountain range. What happened with the squirrels is in biology known as a DMI (a 'disease mediated invasion').

With an evolutionary change approach in the workplace, the "Red Squirrel - Grey Squirrel Strategy"

[12] http://www.telegraph.co.uk/news/earth/wildlife/12122377/11th-Duke-of-Bedford-blamed-for-unstoppable-grey-squirrel-invasion.html
http://www.bbc.com/news/uk-england-beds-bucks-herts-35417747
http://www3.imperial.ac.uk/newsandeventspggrp/imperialcollege/newssummary/news_25-1-2016-15-38-49
https://www.ft.com/content/6d9fd8f0-c9ff-11e5-be0b-b7ece4e953a0

as we've come to call it, is used to reduce resistance. We don't ask individual or groups to give up a particular practice, such as prioritization based on ROI, because "we don't think they'll go for that!" Instead, we let the old practice continue, while we introduce into the environment, the practice that we anticipate becoming its successor. If the new practice, such as selecting and sequencing work based on urgency or timeliness, through an understanding of cost of delay, is successful, then we expect the older practice of sequencing based on return-on-investment, to die out. However, with stubborn environments, often where there is a tightly knit, highly cohesive social group, with a conservative, risk averse, culture, or where a practice is particularly strongly associated with the identity, self-esteem, ego, or social group status of individuals, then the old practice tends to stick around. While the old practice has been obviated, and no longer plays a role in successful outcomes, it survives. Time spent on it wasteful overhead and yet it remains. It's an evolutionary relic - something hard to explain left behind by evolution change in action.

## The abandonment guillotine

What happens to items that are never sufficiently important and sufficiently urgent - items that simply never get selected in a replenishment meeting. A few months after the initial roll out, Dragos recognized that a new policy was needed: Any item older than six months was purged from backlog, closed as "abandoned." There was now an explicit abandonment guillotine policy. If it wasn’t important enough to be selected within six months of its arrival, it could be assumed that it wasn’t important at all. Such policies work on the assumption that every request for work has a mother - the person who initiated the request. If the mother truly cares and the item is truly important, then it will be resubmitted.

## Dropping estimation had one further obstacle to adoption

You may recall from the previous chapter that there was a governance rule concerning operational versus capital expenditure which stated that work requiring more than 15 days of engineering must be routed to a project in the major project portfolio and accounted for as capital expense. If we don't estimate, then how would we know whether something is too big or not?

This was solved by accepting that some might sneak through. We refer to this as the "credit card security" solution. Credit card companies don't try to completely prevent fraudulent transactions on credit cards - doing so, would make it so challenging to use a credit card, that many of us would revert to cash or find other modern means of making payments. Instead credit card companies build an allowance for fraud into their business models and they pay for using the margin they charge the merchant for accepting credit card payments. When any of us use our credit cards, a percentage often 3- 4.5% is retained by the card company and not paid out to the merchant. Some of this money is allocated to insure against fraudulent transactions. Credit card companies worked out that it was better to take the risk of some bad things happening rather than eliminate the possibility but have their business shrink significantly.

Historical data told us that these were less than two percent of requests rerouted as too big. Hence, to retain the estimation effort to eliminate this two percent risk was bad economics. We were spending 30-40% of our capacity on estimation. If the governance rules on accounting were the only remaining reason to retain estimation, then it was a very bad bargain - who would pay 40 to insure against a potential loss of 2? So, instead we decided to let the "too bigs" into the system and catch them later.

Developers were instructed to be alert, and if a new request they started to work on appeared to be large, and they estimated that it required greater than 15 days of effort, then they should alert their local manager. If the item was confirmed with high confidence to be too big then it would be rerouted to the major project portfolio. The risk and cost of doing this was less than one-half of one percent of available capacity. It was a great tradeoff. By dropping estimates, the team gained more than 3o percent of capacity at the risk of less than 1 percent of that capacity. This new policy empowered developers to manage risk and to speak up when necessary!

Note: This is a common theme in the Kanban Method. The combination of explicit policies, transparency, and visualization empowers individual team members to make their own decisions and to manage risks themselves. Management comes to trust the system because they understand that the process is made of policies. The policies are designed to manage risk and deliver customer expectations. The policies are explicit, work is tracked transparently, and all team members understand the policies and how to use them.

## What happened next?

The first two changes were left to settle in for six months. A few minor changes were made during this period. As mentioned, a backlog purge policy was added; the weekly meeting with product owners also disappeared. The process was running so smoothly that Dragos had the Product Studio tool modified so it would send him an email when a slot became available in the input buffer. He would then alert the product owners via email, who would decide among themselves who should pick next. A choice would be made and a request from the backlog was replenished into the kanban system within two hours of a slot becoming available.

## Looking for Further Improvements

Dragos began looking for further improvement. He’d been studying historical data for tester productivity from his team and comparing it with other teams within the XIT services at TCS in Hyderabad. He suspected that his testers were not heavily loaded and had a lot of slack capacity. By implication the developers were a significant bottleneck. He decided to visit the team in India. He sat in their office for two weeks observing. On his return he instructed TCS to make a headcount-allocation change. He reduced the test team from three to two and added another developer (Figure 4.6). This resulted in a near-linear increase in productivity, with the throughput for that quarter rising from 45 to 56 change requests completed and deployed to production. He had correctly assessed that there was slack capacity in test. Two testers were sufficient to handle the work coming from four developers.

Microsoft’s fiscal year was ending in June 2005. Dale Christian, the General Manager, and his leadership team were noticing the significant improvement in productivity and the consistency of delivery from the XIT Sustaining Engineering (software maintenance) team. Finally, management trusted in Dragos and the techniques he was employing. My phone rang!

"David, it's Dragos. Dale loves what we are doing. He sees the results. They've been reviewing the annual budgets and I've been told I can hire two more people. So, I'm about to email TCS and ask them for two more developers."

"I don't think I would do that," I replied.

"No?"

"I think there is a danger that two testers can't handle the workload arriving from six developers. I think, based on my admittedly only superficial understanding of your data, that two more developers will turn testing into a bottleneck and you won't get all the benefits you are expecting. My gut feeling is that you should go for one of each – a new ratio of 5 developers to 3 testers. I think that will work."

I was using my knowledge of the Theory of Constraints and bottlenecks to give this advice. The approach is explained in more depth in chapter 17.

Dragos added one more developer and one additional tester in July 2005. By the winter of 2006, the results were significant as shown in figure 3.6 and 3.7.

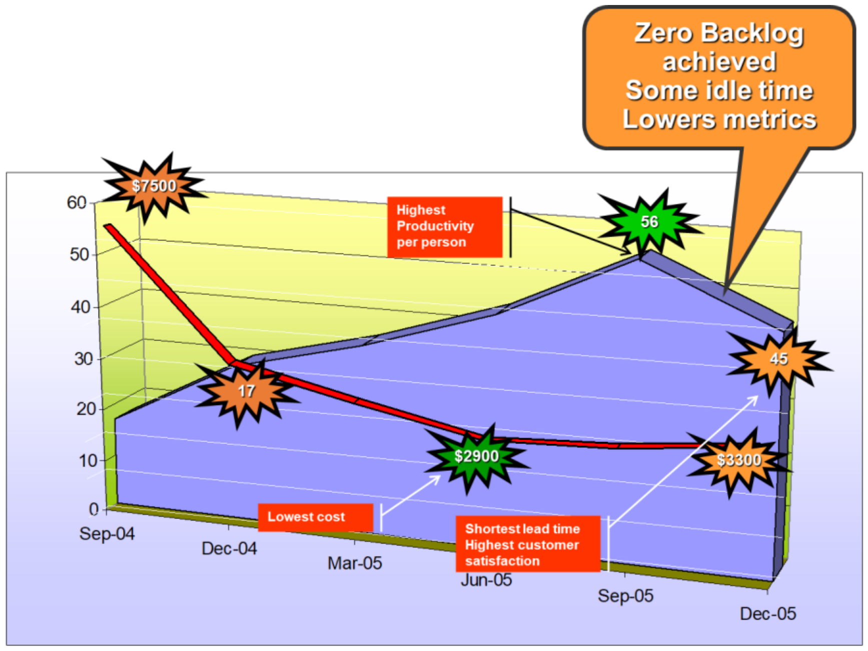


Figure 3.6 XIT Sustaining Engineering, delivery rate of change requests versus cost per change request

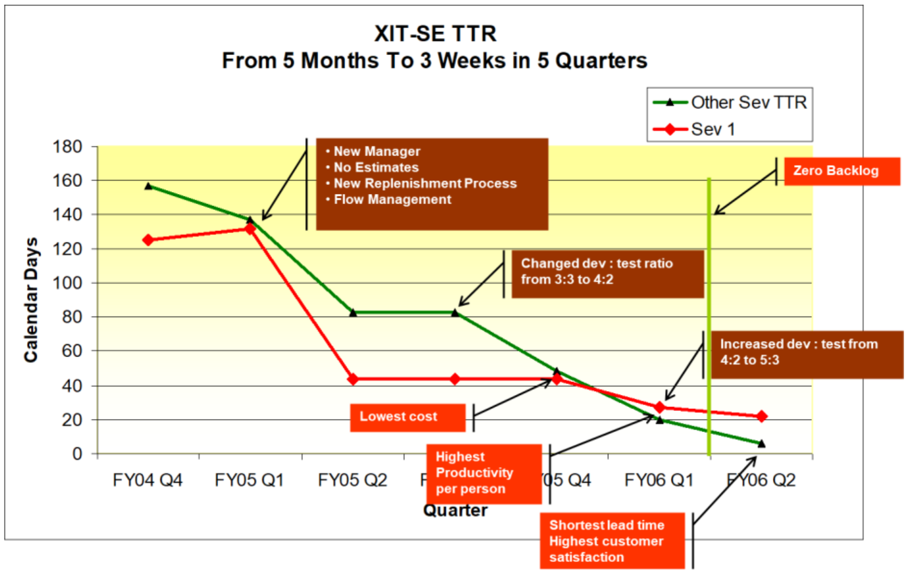


Figure 3.7 XIT Sustaining Engineering, Time to Resolve (TTR) or average lead time per change request from commitment to deployment

## Results

The additional capacity was enough to increase throughput beyond demand. The result? The backlog was eliminated entirely on November 22, 2005. By this time the team had reduced the lead time to an average of 14 days against an 11-day engineering time. The due-date performance on the 25-day delivery time target was 98 percent. The throughput of requests had risen more than threefold, while lead times had dropped by more than 90 percent, and reliability improved almost as much. No changes were made to the software development or testing process. The people working in Hyderabad were unaware of any significant change. The PSP/TSP method was unchanged and all the corporate governance, process, and vendor-contract requirements were fully met. The team won the Engineering Excellence Award for the second half of 2005. Dragos was rewarded with additional responsibilities, and the day-to-day management of the team was handed off to the local line manager in India, who relocated to Washington State to work on the Microsoft campus.

These improvements came about in part because of the incredible personality and managerial skills of Dragos Dumitriu, but the basic elements of Kanban were key enablers: mapping a value stream, analyzing flow, setting WIP limits, and implementing a pull system. Without the flow paradigm and the kanban approach of limiting WIP, the performance gains would not have been possible. Kanban enabled incremental changes with low political risk and low resistance to change.

By the fall of 2005 I began reporting the results initially via my blog, then at a Theory of Constraints conference in Barcelona, and again in winter of 2006, at a Lean Product Development conference in Chicago. That year others began to pick up the concept and replicate it. Most notably, Eric Landes at automotive component manufacturer, Robert Bosch's South Bend, Indiana facility where he replicated our results with a team doing software maintenance of intranet applications. At the time, what we referred to as "a virtual kanban system for software engineering" was gaining broader adoption and awareness. It wasn't yet the full Kanban Method as we know it today. That wasn't to emerge until 2007 but the results at Robert Bosch validated that the approach was replicable and that it did not require David or Dragos' leadership in order for it to work.

The XIT story shows how a WIP-limited pull system was implemented on a geographically distributed IT service using offshore resources and an outsource vendor. The implementation was facilitated with a software tracking tool. There was no visual board and many of the more sophisticated features of the Kanban Method described later in this book had yet to emerge. Nevertheless, what manager could ignore the possibility of similar results? Adopting a "start with what you do now" evolutionary approach to change using kanban systems was clearly an approach worthy of reporting publicly for others to replicate and something that we both wanted to try again!

## Takeaways

The first Kanban implementation at Microsoft improved productivity by greater than 3x, dropped lead times by 92% and improved on-time delivery by 98%

Policies affect performance. Some policies, set by senior executives, must be treated as constraints and can't be easily changed

Stopping estimating was a choice, other options were available: isolate the disruption of estimation in a time slice; isolate the disruption of estimation using a specialist estimator role; combine the other two options with a specialist role which rotated between team members. These other options were rejected because they improved predictability but didn't recover any wasted capacity.

Input buffer sizing should anticipate the maximum anticipated delivery rate in the time period between replenishment meetings. The goal is to insure that the first activity in the workflow is never starved of new work to start, and its workers are consequently never idle.

A buffer between two activities may be desirable to smooth flow due to variability in local cycle times in each activity

Sometimes initial buffer sizing can be arbitrary. From empirical observation over 10 years, a WIP of 5 is often a good starting position. From there, the size can be adjusted up or down as observations are made on how heavily it is used

Abandoning traditionally planning, and switching to deferred commitment and a service level agreement for delivery expectations is a core concept in Kanban. Traditional planning often encourages early commitment, leading to re-planning and re-scheduling of work. This can be a source of dissatisfaction for customers.

Once a proposed kanban system is designed for a service delivery workflow, it is important to anticipate who might object to making the switch

Generally, people who will raise objections are those who have their identity, self-esteem, or social status anchored in the skillful execution of a specific practice. A suggestion to change or remove such a practice will meet with resistance

Shuttle diplomacy and meeting with individual stakeholders to explain the proposed changes is highly recommended. Get individuals to agree and commit to the changes before holding a group kick-off meeting

Evolutionary change processes can leave behind strange historical artifacts or obviated practices. These are known as evolutionary relics

An approach to changing a practice that will invoke resistance and defensiveness from some individuals is to introduce the new, hopefully replacement practice, alongside the incumbent.

The approach of introducing the intended successor, alongside the incumbent practice, is known as the Red Squirrel - Grey Squirrel Strategy

Use of a time limit on submitted requests, is useful to prevent backlogs growing to a large and unmanageable size. The time limits are known as Abandonment Guillotines

Sometimes there may be an advantage to letting something bad happen, so long as it can be detected quickly and its impact minimized, than spending a lot of effort upfront to prevent it from happening at all. Risk avoidance can be more expensive and wasteful than risk mitigation. This concept is refer to as the Credit Card Fraud Solution.

When adding people or automation equipment to a service delivery workflow and kanban system, it is important to consider where to place the additions, and to avoid accidentally creating a bottleneck that limits the value and improvement produced