

Taming the Customer Support Queue

A Kanban Experience Report

Karen Greaves
Product and Technology,
Fundamo
Durbanville, South Africa
kareng@fundamo.com

Abstract—Dealing with customer support issues can often affect Agile teams negatively. It can impact both their focus on new features and their predictability. This paper shows how Fundamo made use of a Kanban system to take control of their customer support issues and drive quality improvements in their Agile development process.

Keywords: Kanban; agile; customer support; defects; quality

I. INTRODUCTION

Fundamo is a leader in mobile financial services products. We provide a white label¹ product that enables our customers to offer mobile wallet and mobile banking transactions in their service offering.

Fundamo's customers are either banks or mobile network operators. We have a comparatively small number of customers, but each of them has large numbers of customers who are the users of the Fundamo product. As a result our relationship with each of our customers is vitally important to our business. The success of Fundamo is dependent on the success of our customers.

Fundamo began in 2000. By 2008 we had two major versions of our product, both substantially customized for individual customers. The mobile payments industry was growing. Fundamo sought investor funding to capitalize on its existing versions by creating a more generic mobile financial services platform which would be easily extensible and configurable to cater to current and future needs in the mobile financial services markets.

Fundamo received funding for two years to build Fundamo Enterprise Edition (FEE), which would incorporate the functionality of the two existing versions (R3 and R4), and position Fundamo well for the future.

Fundamo separated the Product and Technology teams, who were responsible for building the core FEE product, from the Professional Services teams, who worked more closely with customers to customize and configure FEE according to specific customer needs. The Fundamo Professional Services teams serve as the Product teams' internal customers.

Fundamo used Scrum in the Product team to build FEE, and released several versions throughout the two-year rebuild. It was difficult to persuade existing customers to move to the newer version while it lacked some of the functionality of the older versions, and therefore new customers were targeted for the initial versions of FEE.

Because of quality issues, none of these customers had managed to go into production by July 2010. However, the focus of the development team was on completing functionality in the two-year time frame.

A separate support team, mostly located offshore in Pune, India, was tasked with dealing with issues from customers trying to launch while the development team focused their effort on completing features.

In July 2010, the two-year rebuild was over, the investment funding was spent, and the team now needed to get customers into production. As a result of funding being over, the available teams were significantly reduced. The number of issues being raised exceeded the number being closed, and a large backlog of issues existed.

It was at this time that I was appointed as the Software Development Manager in Product and Technology at Fundamo. My first mandate was to get support under control, and get the first customers into production before the end of 2010.

II. UNDERSTANDING THE PROBLEM

The first challenge was to get a clear understanding of the size of the problem before we could select an approach to deal with it. Fortunately, Fundamo had been using Atlassian's Jira software to track all issues. This provided the data that I used to determine the current state.

We had a team of three people in a Level 2 (L2) support team who attempted to resolve issues that did not require code changes, and a team of four people in a Level 3 (L3) support team who focused on fixing issues in the code. All but one of these people were working off-site in Pune, India. Despite having seven people allocated to support, the trend of the support queue was growing at a rate of about 50 issues per month, as shown in Figure 1. Even after a concerted effort in March 2010 to fix a number of issues before a release, the trend continued. It was apparent that the current process was not working.

Since investment funding was over, the plan was to revert to much smaller teams, working only in Cape Town. The existing support team would no longer be available, and the development teams that had been split between offshore and on-site team members would now consist of only the on-site members.

Ultimately we faced a 60% reduction in team size. A significant change was clearly required.

¹ A white label product or service is a product or service produced by one company (the producer) that other companies (the marketers) rebrand to make it appear as if they made it.

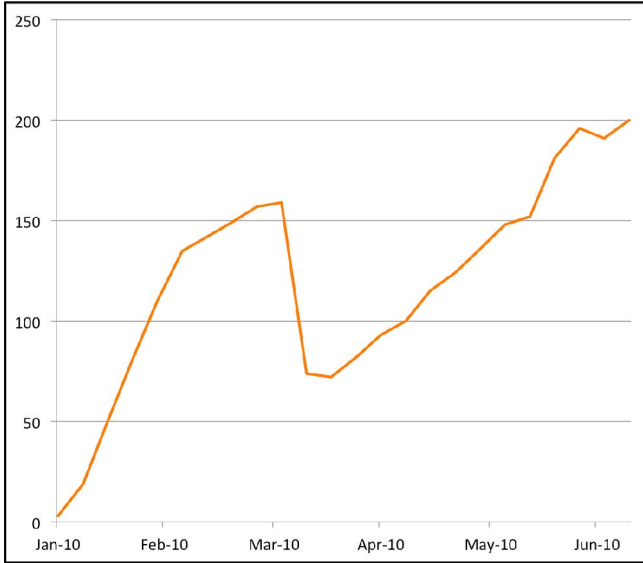


Figure 1: Growing trend of outstanding issues in the main support queue

III. PLANNED APPROACH

A. Structuring the teams

Given the reduction in team size, we decided to group people into three teams rather than the five teams we had previously.

The first challenge was to decide how to structure these three teams to deal with both new development and customer support. My view, from many years working in the software industry, was that it was essential for the future quality of the product that the same developers were responsible for supporting the product and developing it further. Without this, there would be no incentive for the developers to consider the impact of supporting a feature when developing it. However, since the teams were using Scrum during their development cycles, it was important for them not to be interrupted during their sprints with other work. The decision was to nominate one person per team to be dedicated to support for a particular sprint. At any one time there would be three developers allocated to support.

B. Simplifying Jira

Secondly, I felt it was important to simplify the support structures we had in place. We modified Jira to limit the priorities to only three (Severity 1, Severity 2, Severity 3), rather than the default setting of five (Blocker, Critical, Major, Minor, Trivial). This was in line with our support agreements with customers, which would make tracking Support Level Agreements (SLAs) easier.

The new levels were:

- Severity 1: 10% or more of clients cannot transact, OR Functionality seriously impacted, OR Financial integrity possibly compromised
- Severity 2: Less than 10% of clients cannot transact, OR Limited impact on functionality, OR performance impaired, OR a workaround exists

- Severity 3: No material impact on Client transactions OR Can work around it

We did a simple mapping exercise to convert existing issues to the new priorities. Blocker and Critical issues became Severity 1, Major issues became Severity 2, and Minor and Trivial issues became Severity 3.

We had previously maintained six different queues of issues in Jira, as shown in Figure 2. Prioritizing work across these six different queues was challenging. Some queues ended up being ignored as no one was assigned to monitoring them. We had separate queues for L2 and L3 support. Issues were replicated in the L3 queue when L2 engineers could not solve the issue. This caused problems because issues were not synchronized. If the replicated issue was resolved, it was a manual process to close the linked issue, which meant it was error prone.

We decided to combine all six queues into only two queues. The first queue was dedicated to all customer-support issues, which were any issues found by our customers. We define our customers as anyone outside of the Product team. This could be people in our Professional Services division, sales staff or our end customers. The second queue was for issues found internally by our development teams while developing and testing new features. We felt it was important to separate these two, as the customer-facing queue would relate to released versions of the software and would give us an indication of the quality of our released code. The second queue was for internal use only; it did not necessarily indicate quality of our released software, as issues were logged here during the development process.

The customer-support queue would be managed by the new support process. The three people nominated from their teams to work on support would address these issues. The development queue would be managed by the teams themselves as part of their normal development sprints. We decided to take a hard-line approach to quality in development and agreed that any bugs found would be fixed in the same sprint that they were found, regardless of whether they related to the features being delivered in that sprint. The goal was that the development queue would be empty at the end of each sprint.

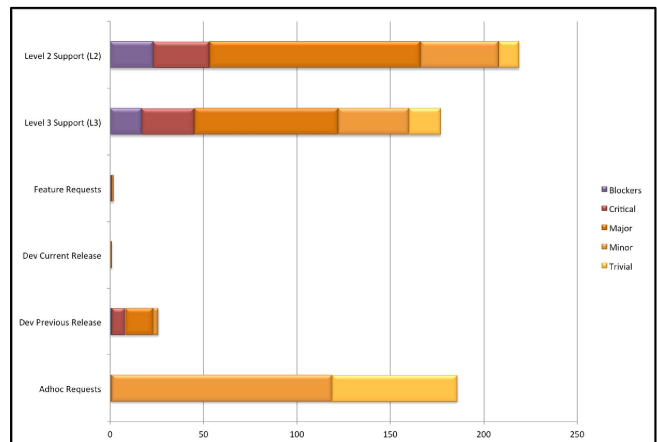


Figure 2: Outstanding issues across all queues in July 2010

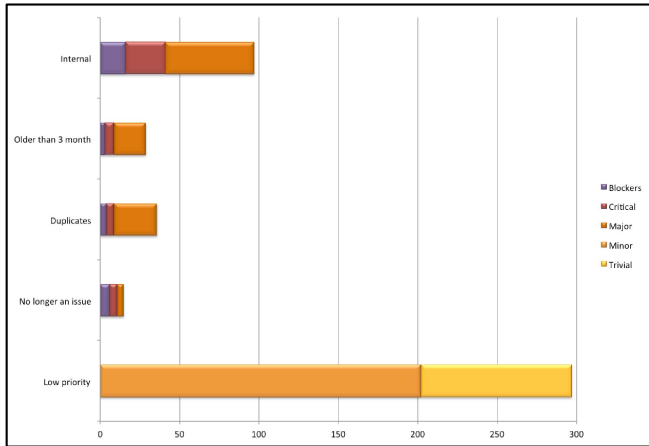


Figure 3: Issues that were closed in the cleanup exercise

C. Using Kanban

Although we were using Scrum for our development work, we chose to use Kanban for support. I had attended a three-day Kanban training workshop with David J. Anderson, author of “KANBAN, Successful Evolutionary Change For Your Technology Business” [1], earlier in the year, and found that aspects of Kanban made it a better fit for support. Firstly, Kanban allowed for issues to be added to the work queue at any time, rather than on a fixed cadence as in Scrum. Secondly, Kanban allowed for releases to be decoupled from the sprint cadence. We would have the flexibility to release a patch at any point, rather than at the end of the fixed-length sprint. Finally, Kanban provided a mechanism – cycle time – to predict how long it would take to fix issues without requiring the teams to estimate issues up front. Teams felt they were unable to estimate issues until they had spent time investigating the problem. Although I had been trained on how to set up a fairly complex Kanban board with buffers, work in progress (WIP) limits, multiple types and classes of services, I chose to start with a basic board. David’s advice had been that it was easier to allow the board to emerge and add complexity when necessary, rather than starting with complexity you didn’t need and trying to remove it later. Our initial board had only one type, namely Bugs; and three columns, namely To Do, In Progress, and Done. The board had no WIP limits.

We did however make sure to use a Cumulative Flow Diagram (CFD) from day one. An example of our CFD is shown in Figure 10 at the end of the paper. We simply recorded the number of issues in each column at the same time each day, just after our daily meeting. This is what we used to measure cycle time and make predictions around our defect fix rates.

D. Stakeholder Engagement

The final piece of the initial plan was to face up to the harsh reality that we had more issues than we could deal with. An

important concept I had learned from David was balancing demand and throughput. It didn’t matter what you did; if you had more demand than throughput you would choke the system. It was time for an honest assessment of what was important to Fundamo and its customers.

Given the stage of the product lifecycle we were in, there was no capacity to deal with minor cosmetic issues; we had customers unable to launch the product in the market. At this stage, we were not going to focus on issues that had been logged by internal people that were not directly preventing a customer from launching the product.

I set up some meetings with our internal stakeholders, the Professional Services teams responsible for getting our customers into production. We talked frankly about the stage we were at, and agreed that our common goal was to get our customers into user acceptance testing or production before the end of the year so that we could start to collect some revenue from the product.

We agreed to work closely together and get their assistance in prioritizing the issues towards this goal. To do this we agreed to a weekly prioritization meeting where we would review the current outstanding issues and agree on what issues we would tackle next. This meeting drove the issues that we added to the Kanban board.

E. Cleaning out issues

In July, just before getting started, we did a mass cleanup of Jira. We closed issues identified by the following heuristics:

- Issues that had been logged internally but were unrelated to a client project.
- Issues older than three months, under the assumption that if the issue had not been escalated during that time, it was likely to not be holding up a customer project.
- Issues that had been duplicated in more than one queue
- Issues linked to other issues that had already been closed.
- All minor and trivial issues.

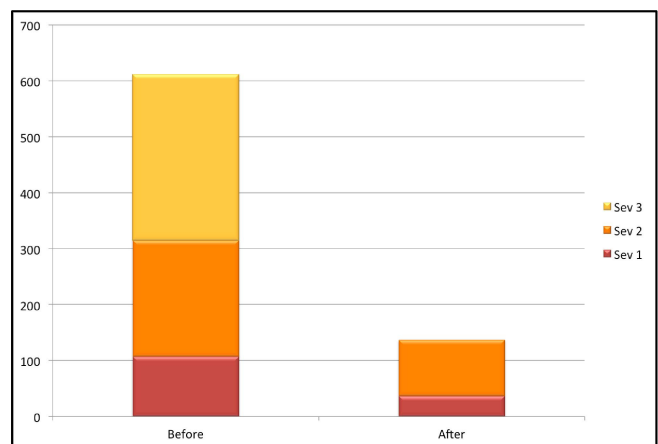


Figure 4: The impact of the cleanup on the outstanding issues

The end result, shown in Figure 4, was a single queue with 37 Severity 1 issues, and 100 Severity 2 issues. Addressing these issues seemed like a manageable task.

IV. HOW THINGS EVOLVED

A. Making predictions

On Monday, August 2, 2010 we started our Kanban support process with one developer from each team. Unfortunately it soon became clear that three developers were not able to fix issues fast enough for the customer projects we had running. Although we were focused purely on a single customer, their planned launch date in September was under threat. After an escalation from the customer, we put nearly all hands on deck, and upped the capacity to eleven developers on August 18, 2010. At the same time we used the average cycle time and the rate of incoming issues from this customer to predict when we would reach zero Severity 1 and 2 issues. The prediction showed us what we all knew in our gut to be true. There was no way this customer was going to be able to launch on time. This was difficult to digest. It is impossible to go to a large global customer and tell them to delay their launch, which included television commercials, without giving them a new date we could be certain they would be able to launch.

I tracked statistics daily and updated the prediction of when we would complete their issues. I tracked how the prediction changed from day to day. The results of these predictions are shown in Figure 5.

In the middle of September, the prediction indicated we would be done in mid-October. Feeling more confident, having announced this delay to the customer, we dropped the support team down to six developers. We urgently needed to continue developing new features for other customers. We expected that the incoming rate of issues would start to decline as we got nearer to the end of the project, and that we would cope with less people.

We were wrong. Issues continued to stream in and by the end of September it was clear that it was more likely that they would only be able to launch in late November. Once again we had to go back to the customer and announce a delay. Fortunately the predictions stabilized around 18 November, and the customer set a launch date of November 29, 2010.

By November 9, 2010 we succeeded in closing all their Severity 1 and 2 issues. We started working on issues for the next client planning to launch. On November 29, 2010, as planned, the customer launched the product successfully and had no issues during their televised launch. Production was stable, and very few issues were raised with the support team. Most importantly, we had won back the confidence of the business in our ability to predict when we could deliver. Everyone agreed the Kanban approach was working.

B. Further cleanup

In October, after we had fixed a large number of issues in support, we got the agreement from our stakeholders to

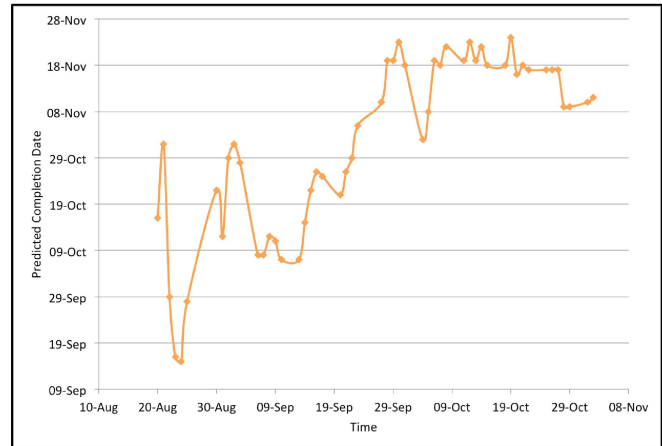


Figure 5: Tracking predictions of when we would close all Severity 1 and 2 issues for a particular client

close all issues on previous versions of the product released before July. Our feeling was that the quality of the product had increased dramatically, both through a focused support effort, but also through our development approach of fixing issues when we found them. We all agreed that the effort of going through the old customer issues to see which were still valid would not be valuable. Any issues remaining would resurface, and if important would be fixed quickly. Now that our stakeholders saw us attending to their urgent issues quickly, they felt comfortable closing older issues. This helped us drop the outstanding issue total down to 78 issues, which took us closer to our end goal which was to have no backlog of issues and to deal only with issues immediately as they are logged.

C. Team changes

Some significant changes took place during this time as we refined our process. One of the first was a change to how we rotated people through the support team. Within the first month, teams noted that the people on support felt isolated from their teams, and were mostly working alone. They asked if we could change the process so that a whole team worked on support at any one time. Since it was already clear that three developers was not enough capacity to deal with support, we made the change. In hindsight it seems an obvious change. We knew from Scrum that cross-functional, self-organizing teams work far better than individuals. This change allowed those already close-knit teams to work together in support. Initially we agreed to rotate the teams each sprint. However, by December 2010 the teams suggested the rotation only happen once a month, since after a two-week sprint on support they had just found their stride, and felt they would perform better if they continued for another sprint.

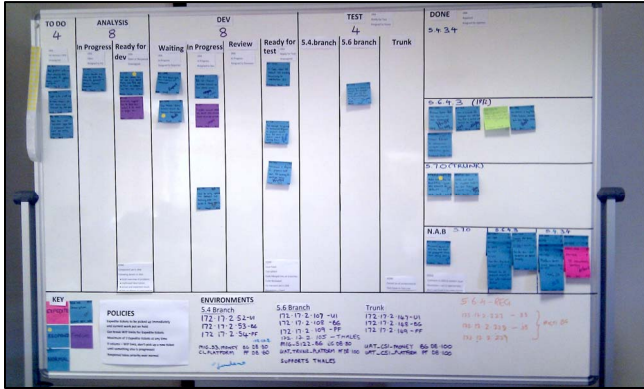


Figure 6: Our Kanban board in March 2011

In November 2010, we had added a full-time support coordinator whose responsibility was to manage the process. She quickly came on board and took over meeting with our internal stakeholders to understand priorities and populating the work queue as needed.

D. Board Evolution

As mentioned earlier, we started out with a very basic board. As you can see from Figure 6 our board has advanced significantly. The following section describes the changes that took place. The timeline is illustrated in Figure 7.

August 23 – the team added a test column. Although we started with a basic Scrum-style board, the team realized that they were working exclusively on issues, and instead of creating tasks for coding and testing, the single defect ticket progressed through the process steps on the board. It became clear that we needed to see which issues were in development and which were being tested. The test column was also split into multiple branches as fixes needed to be tested on the supported released versions of the product and the version currently in development.

September 20 – after seeing a large number of issues in the test column and only one tester in the team, I suggested it was time to add some WIP limits and a buffer in front of test called “Ready for test” so that the team could help out when tests got stuck, rather than working on new issues. With 5 developers in a team, we set the development WIP limit to 8 and the test limit to 4. We agreed it made sense for developers to mostly work on one thing at a time, but they occasionally worked on two if the two bugs were related or they were waiting for customer feedback on one issue, and thus we set the limit to 1.5 times the number of developers and rounded up to 8. For the test limit we just asked the tester on the team what they thought was reasonable. Since the tester often tested a number of issues together, they felt a limit of 4 was reasonable.

October 18 – the Product Owner noticed that the support team had fixed an issue that a customer reported, which was actually a conscious design decision. He realized that he needed to get more involved in support and ensure that

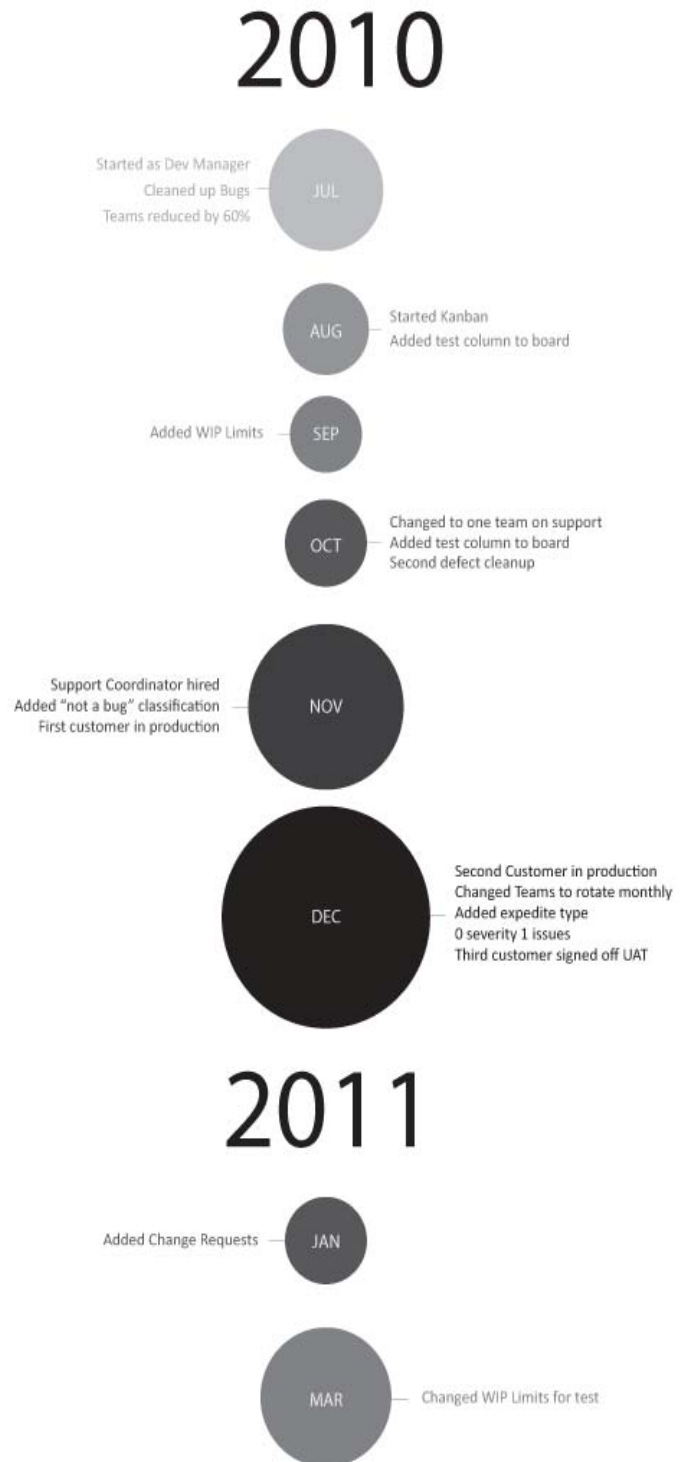


Figure 7: Timeline of changes. The sizes of the bubbles for each month indicate the number of changes in process and events that occurred during that month.

issues customers were reporting as bugs made sense before developers fixed them. We needed a space for him in the process. We added an analysis column in front of development, and a "ready for development" buffer column. We didn't add any limits to those columns initially, since we knew he would probably look at issues in batches when the team needed more issues.

November 29 – we noticed that a large number of the issues we dealt with in support turned out not to require code changes. These issues were either misunderstandings or configuration problems. To track this we split our "done" column horizontally and separated issues fixed in particular patches from those we classified as Not A Bug (NAB). This gave the team a great visual indicator of the ratio between fixes and non-fixes.

December 10 – now that we had a customer live in production, we introduced an "expedite" class of service. Any Severity 1 production issues would be on a pink ticket. We added explicit policies around expedite tickets. We also added a "waiting" column, to be able to easily see if tickets were waiting for customer feedback. These tickets still count toward the development WIP limit, as we want to avoid tickets waiting for too long. Teams felt that sometimes we were not doing code reviews of fixes, so we added a review column as a specific reminder of that step in the process.

We also formalized our policies by displaying them on the board, and added definitions of done to each column of the board. Our policies included the following:

- there could be no more than three expedite items on the board at any one time;
- we could break WIP limits for expedite items; and
- reopened items took precedence over any items except expedite items.

January 28 – support had been working so well that over December we had cleared all Severity 1 issues, and had only a few Severity 2 issues remaining. Our customers felt that at this point their projects were being most impacted by features that were identified late in the project, rather than defects. We decided to allow for small change requests in the support team as well as defects. Our only caveat was that the changes needed to be small in size, less than 2 days' work. If we introduced items significantly larger than the defects then our process would not flow as smoothly. We added a new color on the board to represent change requests. The team felt that for this they would like to ensure they did some group design sessions, and so the team agreed to hold a design session similar to Scrum's sprint planning after each weekly prioritization meeting.

March 3 – we had not anticipated the impact of adding change requests on the testing portion of our process. The tester mentioned that he was struggling to keep up with the team, as he usually needed to do more work for change requests than for defects. After some discussion, it seemed the team was unaware of the bottleneck in testing, and therefore had not come to his aid. We agreed to reduce the

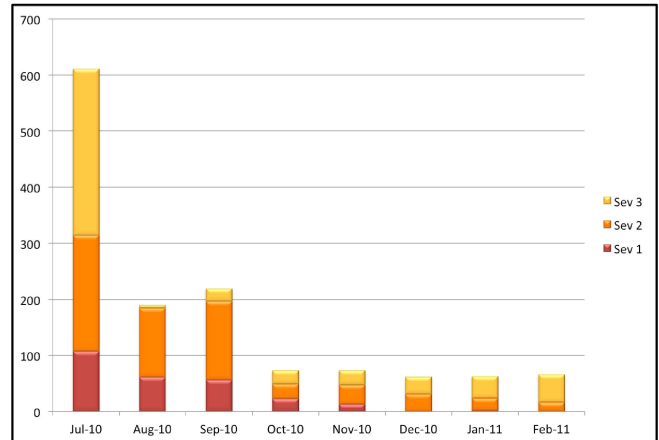


Figure 8: Outstanding issues in the support queue at the end of each month

test WIP limit to 2 to make it more obvious when the tester needed help from other team members.

V. RESULTS

Looking at Figure 8 it is obvious that our issue queue is substantially smaller than it once was. Most of the big wins here were due to closing issues we would not fix. I believe this was about matching our demand to our throughput, and focusing our effort only on the most valuable items. Both are strong principles in Agile software development.

Reducing our queues has had two significant impacts:

- Our teams believe it is possible to reach our goal of a backlog of 0, and are motivated to get there.
- Our customers see us turning around issues they log today much more quickly than we did in the past, and therefore support the new process.

Figure 8 however does not show the real impact of the new process. Figure 9 shows the number of incoming issues alongside the resolved issues. Issues closed during cleanup exercises in July and October 2010 are separated out. Here we can see a marked difference after August. The team was resolving all incoming issues as well as making a small

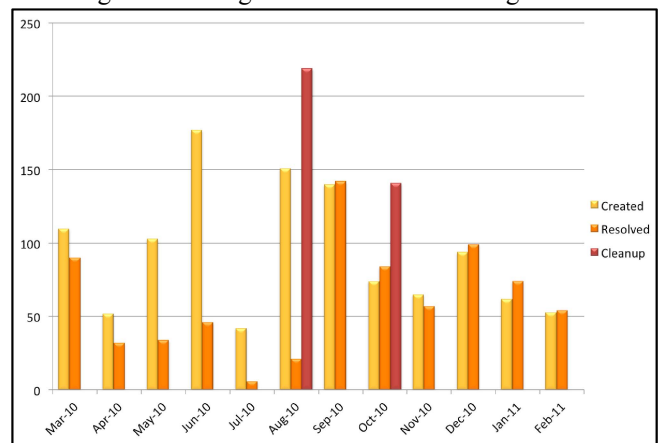


Figure 9: Comparing the number of issues created with the number resolved

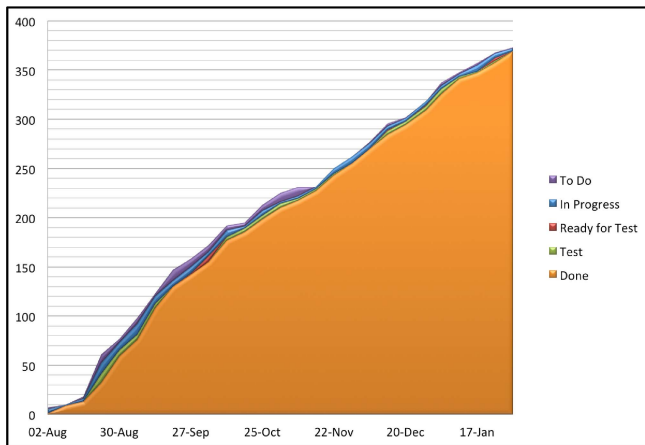


Figure 10: Our Cumulative Flow Diagram (CFD) a simple mechanism to measure the impact of any policy or process changes

impact on the existing backlog. This shows that under the new process our throughput matches our demand and therefore the queues are under control.

VI. CONCLUSION

In conclusion, I believe there are a few key insights to share. As an avid Agilist, I believe in the power of self-organizing teams; however, the entire support process has been managed by either myself, or since November, our full-time Support Coordinator, Kershnee Govinder. To make a process like this work well requires constant interaction with stakeholders, communication, reprioritization, and adherence to the agreed policies. Someone other than the team doing the work can do this best, since this minimizes interruptions for the team. Also this enabled us to maintain a consistent relationship with the business despite the fact that teams rotated through support. The greatest value of our Support Coordinator is that she maintains the relationship with our stakeholders, regardless of whether the underlying teams have changed. Having this single interface has also freed the teams to focus on the work. My first insight is not

to underestimate the amount of work required in managing a process like this.

My second insight is to know that regardless of how well your plan is conceived, it will change as you execute it and learn more about the work. It is perhaps better to start with a less formal process and Kanban board so that there is less reluctance to change it. Although we now use Greenhopper as well as a manual board to manage our issues, I believe that starting with only a manual board meant that it was simpler to change, which in turn made it easier for the team to make changes as necessary.

It is important to remember that the best people to improve the process are the teams doing the actual work. We hold fortnightly retrospectives with the teams and the support coordinator to discuss the process and find ways to improve. We have also held retrospectives with our internal stakeholders to see what could be improved in the process for them. Although I offered guidance on how the mechanics of a Kanban board could support something they were grappling with, the teams themselves identified the problems, and implemented solutions.

Finally, to me, the greatest benefit of Kanban is that it provides a mechanism to quantitatively measure the effects of any changes in your process. From day 1 we measured a daily CFD, and kept a comments field to track changes in the process, the teams and the environments. All of the data in this paper was reconstructed from both the CFD and our Jira database. It is very simple to track this, and has provided great insights into the impact and effectiveness of our process changes. We are able to quantify both our demand and throughput and ensure these stay balanced by changing explicit policies, as a result our support queue keeps flowing and we achieve good turnaround times on customer issues.

REFERENCES

- [1] David J. Anderson, "KANBAN, Successful Evolutionary Change For Your Technology Business" 2010