Developers and Validators Working Remotely Together on a Single Kanban Board

BECKY BERNDT, INTEL CORPORATION

Sharing a Kanban board is a great start to get teams working together on a common goal, even if the team consists of people in different roles such as developers and validators. Collaboration does not happen automatically. We took many steps to not just get everyone using the same board, but everyone working towards a common cycle time goal of 14 days from start to fully testing a work item, the same cycle time that was nearly a year long at the start.

1. INTRODUCTION

Revolutionizing the world of memory and storage as we know it today takes collaboration. Being on the cutting edge means shifting timelines left, and producing high quality work in less time. Unfortunately our team's new development was taking longer than expected to validate, and even then, not thoroughly. This had to change or we would never be ready to release this amazing product.

Developers and validators needed to combine forces, cooperate, and work towards common goals in order to close this gap and stop revisiting code that was written as much as a year ago.

Our team needed to share and visualize common goals, work together on a daily basis to reach those goals, and needed a way to know if we were succeeding. We chose Kanban, a well-known process used by many agilists to visually show how a team is progressing.

2. BACKGROUND

I am blessed to be a software engineer working on the manageability code for Intel's recently announced 3D XPoint™. It's exciting to be part of something that will revolutionize memory technology. It's crucial that the quality be outstanding. Working as an agile scrum team to shift the timeline left, it was important to have clear, open, and often communication as requirements changed. We needed to earn the right to continue calling ourselves BSTEs (Best Scrum Team Ever, but we pronounce it besties).

We developers have our set of unit tests, but appreciate the importance of a full blown validation suite of tests. However, it was unclear how much was covered in our local validation team's tests, and the validation team was unable to react when features or requirements, and thus, our code changed. So imagine our frustration when we had a local validation team that was not open to taking the risk and trying something new by joining forces and becoming one cohesive team.

Fast forward after a restructure, and we now have a new, but remote, validation team. We knew we had to set this up for success with both sides working closely from the beginning. It's not how either the current development team or the new validation team was used to working, and the validation test suite they inherited was incomplete.

3. MY STORY

I have been a software engineer at Intel for 6 years. The last few years I have also been ScrumMaster for our 3D XPoint manageability software team located in Colorado. We used Scrum for our development for several years. We reached a point where I felt it was important to be able to react quicker to change. A little over a year

Becky Berndt, email: BeckyBerndt@intel.com

ago, I took a brief Kanban session during an agile conference. After doing a little more research, I thought it was something that could work for us. Showing the expected benefits from incorporating Kanban practices to convince my team lead and manager to allow us to make the change was easier than I expected. They could see my vision and were interested in giving it a try. Almost exactly a year ago, just a couple of months after the conference, it was time to start. Never having followed Kanban processes before, we were all experimenting together, which turned out to be a great thing. Already having our work items in Rally, we easily set up our new Kanban states, created our acceptance criteria for each column, gave some Kanban flow training, and off we went, still figuring some of it out along the way.

We had a development team that worked well together for a couple of years on this project. We were disappointed with what we felt was a lack of collaboration with our local validation counterparts. The developer group and the validation group each had their own scrum teams, with their own rituals, and little crossover. The validation group was invited to the development stand-up, review, planning, and retrospective meetings, but attendance was rare and communication was poor. We developers had no idea how much of the code was getting tested, and felt like things were being hidden from us. The validation group was frustrated when specs and code behavior unexpectedly changed. We talked about it during stand-ups, but with few common gatherings the communication failed. We wanted both sides to be on the same team, communicating and planning together, giving status updates together, and talking about priorities together. Unfortunately, our attempts to get the local validation to join forces to achieve better collaboration failed. I've always told my team, "If it doesn't work, we'll change it." What's the worst that could happen? We fail and go back to the way we used to do things?

About 9 months ago, after a team restructure, our validation work was transferred to a new validation team across the globe. There was an eight hour time difference. They were new to our team, our product and to our agile practices. The majority of existing functionality had no associated tests, and tests that did exist did not run, so there was a lot of work to be done. Given the history, some development team members were not sure the new team was up for the task and were afraid of starting all over with a new group of validation people, not knowing anything about them, or what to expect, especially when they wanted to start with a whole new framework. Rumor had it the new validation team was not capable of doing the job. Needless to say, I was concerned, but we were all willing to take the risk together. Turned out, our new validation team was eager and willing to take on the challenges. That's the last time I listen to rumors.

Joining forces was great, but of course there was more work to be done. As we all know agile methods, including the Kanban board visual, don't fix issues, they just make them easier to see. Validation inherited a test suite that had few working tests. They had to build a new test framework, and import existing tests into this framework, had many tests to update, and many more to implement. This all needed to occur while they were validating new work, oh, and learning what the heck this product was and what it was capable of doing.

To jump start the new team members, they got an overview of the product, and freedom to ask anything, any time. This premise still holds true today. We share stand-ups every morning with the entire team, so we should know exactly what everyone has accomplished and what they may be stuck on, which in turn, tells the team what is changing.

The merge started with a short Kanban process training, having both developers and validators joining in daily stand-ups and retrospectives, and adding 'test in progress' and 'test done' columns to our Kanban board. The ideal Kanban flow would have work items tested and accepted before the developer pushes code to the master repository. This ensures the work is complete and correct, and practically bug free. I fully believe that once we get to this point, we should have no new bugs filed by our validation team, making the team more efficient and saving a lot of time fixing defects in old code. The code changes are fresh in everyone's mind and this would avoid task switching back and forth between old and new code changes. However, as I mentioned, code that was created as much as a year ago is what was currently getting tested, so we had a long way to go.

It has been difficult to change the thinking that we are no longer a development team and a validation team, but we are one combined team. I will say that again. We are one combined team. If one fails we all fail. We succeed as a team.

We talked about having developers help build up the test cases. The Poland validation team includes some contingent workers as well as Intel employees. Given the way things worked with contingent workers in Poland, and commitments the developers already had in place, it wasn't an immediate option. Developers were, however, able to answer questions quickly, share documentation that is kept in a shared document repository, and teach feature behavior in order to help get the new members of the team up and running quickly. In the meantime, in order to allow the time to fill in missing tests, I decided to have an initial WIP limit

of 20 for our 'development done' and 'test in progress' columns. High enough to allow some wiggle room, but not so high that we would never revisit this if the number of items in this column continued to increase.

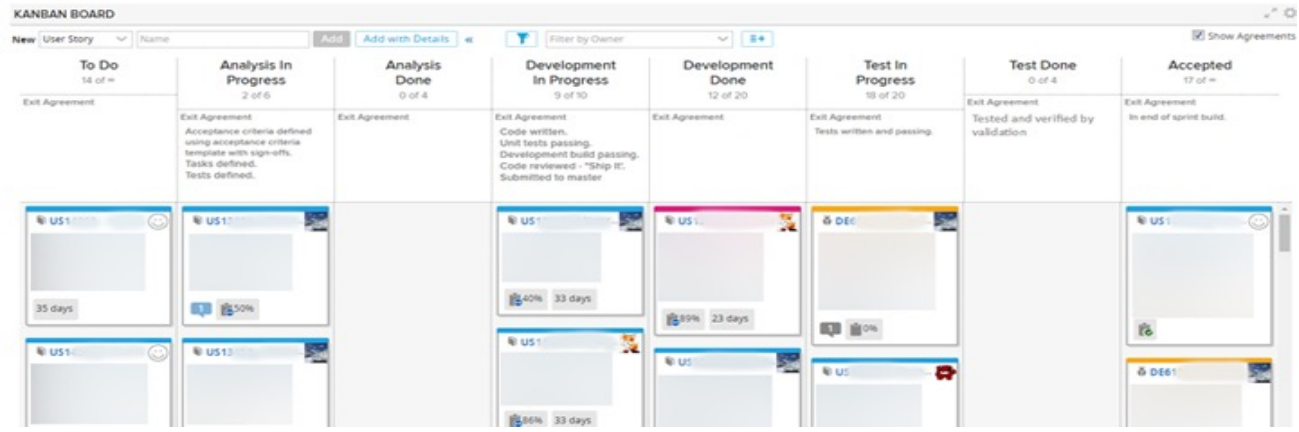

Figure 1. **Modified Kanban Board Combining Development and Validation**

Our goal was always to have our developer's code tested by the validation team before the code gets checked into our master repository. This would keep the number of defects down and developers would spend much less time task switching to revisit code that was modified weeks or months ago now that they have moved on to other work items. In order to do this, our cycle time (days from 'analysis in progress' to 'accepted') would have to decrease…. a lot. With 3 week integration builds, our cycle time of 25+ days starting with the new team was just not going to cut it. We were ready to accept a week in development and a week in test as the most time any work item should take. So, for many months our cycle time goal was 14 days. We also knew that our cycle time would get bigger before it got smaller, since we added the validation acceptance as part of the flow.

Initially it was hard for validation to keep Kanban work items up to date because everything happened so fast. Developers were implementing features and fixing defects, while requirements were changing. Validation was new to the process and didn't want to flood the Kanban board with stories such as completely redoing the test framework and updating tests, so they kept work items off the shared board. They felt that since their items were unrelated to development work at the time, it didn't add anything useful for the overall process and they should not clutter up the Kanban board. I think it would have been better to have these items on the board for full transparency, a better understanding of what the teams were up to, as well as potential places for other team members to jump in and help. We now put all validation work items, including items that have no development impact, and all development work items, including those with no validation impact, on the Kanban board for all to see.

4. PROBLEMS

The eager new validation team and re-energized development team became one manageability unit. Both sides happily jumped into stand-ups and retrospectives. Problem solved, right? Everything we have been doing as BSTEs ("besties", or Best Scrum Team Ever) is perfect, right? Well…

As expected, things were a bit bumpy at first. We had a few obstacles right away. There was now an 8 hour time difference. Validation had a small team and their leadership kept changing. They had to continue to launch the existing test suite on the old framework while bringing the new framework up-to-date. The existing tests needed to be pulled into the new framework and fixed, so no immediate improvement on the gap between development and test of a new feature was realized.

There was confusion on priorities. Development priorities were not always the same as validation. We had one Product Owner so priorities were expected to match.

Working cultures were different. It was hard to break down the barrier of "it is our job to do …", or "they should…" In fact, we still hear that coming up now and then.

5. GETTING EVERYONE INVOLVED

We kept our 3-week sprint cycle and retrospectives after incorporating Kanban, which proved to be important. Our team's practice for over a year has been to have a different member lead the retrospective at the end of each sprint. The new team members jumped right in. Retrospective leaders can search for techniques or get creative and make up something fun. It is sometimes hard to keep the attention of creative people in a retrospective, but our team rocks at coming up with these. We have to watch our time, as we often run out of time since we are so involved in the game. Having varied techniques for retrospectives means pulling various types of ideas and information from the team. For instance, the "What I know, I don't know" focused on the team's knowledge base around the product, procedures, development and release plans, etc. This fired off a set of bi-weekly team sessions for sharing the missing knowledge. The "Worst Possible Situation" retrospective was a fun exploration of what the worst possible outcome of our next stack integration would be. We talked about cats getting into the lab and knocking things over, causing a meltdown of the entire city, and black holes. Granted, not likely, but fun for the team, and did actually make us look deep into a part of the code that could be made more secure. During the "Breakup Letter" retrospective we each wrote a break-up letter to something we dreaded doing or was inhibiting progress. This allowed us to vent and focus on ways to speed things up or improve a process or action. Let's just say some people are a bit colder than others with their breakups. One team member's original version of "Apples to Apples" allowed for some creative releases in a fun, slightly competitive game while still leaving with some good actions on ways to improve. Several others pointed us to process changes to get more communication going during the analysis phase to make sure everyone agrees what it means for a specific work item to be done, eliminating confusion and reworking the item at the end of the cycle.

We tracked cycle time each sprint and reviewed statistics during our retrospectives as well as came up with team agreed practices to help reduce cycle time.

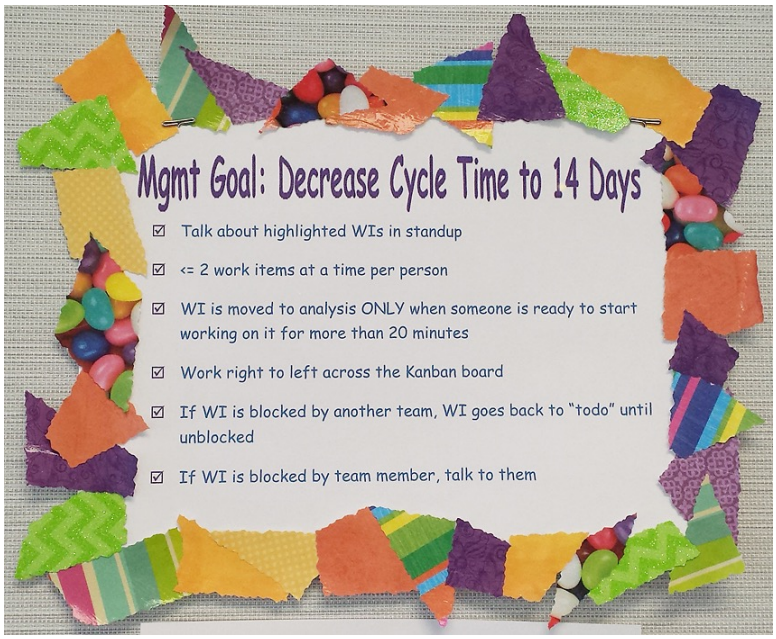

Figure 2. **Cycle Time Goal Agreement**

It was no surprise that our cycle time increased before it decreased. Our average sprint cycle time went from 10 to 28 to 60 days over a 4 month period. We understood this was because tests were being written, frameworks were being put into place, and we are now measuring through the validation acceptance for the first time. Some of the stories that were on the Kanban board when we combined forces were now working their way off the board. We continued to review cycle time and talk about ways to decrease it, including being sure story sizes were reasonable. At the time the new validation team joined, developers were making changes to a complicated feature using various work items to track these changes. Validation was using one work item to create all the tests for this complicated feature, and it turned out to be a much bigger job than expected and could not be completed in a single sprint, or even several sprints, so the item sat in one place on the Kanban

board for several sprint cycles. Splitting user stories helped us better see actual progress by having the combined work items move across the board again on a regular basis. Seeing items moving to the 'accepted' column is also great for the team morale. There is an excitement people feel while dragging their work items to the 'accepted' column.

We struggled with communication around acceptance criteria. That is, what does it really mean for a work item to be done. If development and validation are not in sync, the result will be defects filed for unexpected behavior. We started out by talking about work items after stand-up as they were pulled into the 'analysis in progress' column, and adding the agreed upon criteria to the work item description. This was hard to do without being able to research the story first, and people did not want to stay after stand-ups for this. We then allowed them to take the responsibility of getting it done outside of team meetings. This didn't seem to always happen and we still ended up with questions when it came time for testing, sometimes causing defects and rework. We then moved to requiring a template with more information to be included in the work item description before it can be pulled into 'analysis done'. Details include the expected behavior once this work item is complete, as well as both a validation and a development person signing off in agreement. This at least forced some kind of conversation, but we still seemed to miss the purpose of the conversation, and that is to look at what must change for this particular work item, and what the ending behavior will be, agreed on by everyone. This made it a little easier to see that the discussions happened, as well as knowing the impact to each side, if any. When this got done, it worked well. When it didn't there was once again confusion when we got to the testing phase. Remember, the code was already checked into the master repository, so when questions and issues arose, defects were filed since developers had already moved on to other work items. Let's face it. We tend to focus on what the current priority or work at hand is, so getting stories analyzed that you won't work on for a month or reviewing defects for something you did a month ago is not top on our lists. We still get defects filed due to lack of communication on the agreed upon expected behavior once the work item is completed, or because a work item was not analyzed completely. All the more reason to require that code is verified before it is checked into the repository.

When the team was smaller, a 15 minute stand-up was often enough time to ask questions along with talking about accomplishments and plans. As the team size grew, we just couldn't cover it all in 15 minutes, anymore. However, immediately after stand-up became a great time to hold discussions. In person discussions, or over the phone seemed to get the quickest responses. However, sometimes we needed more details on the system setup, and email worked just fine for that. We also used to allow stand-ups to run 5 minutes overtime as we were getting started, but decided to curb the discussion and keep the 15 minute time limit in order to keep them short, and to allow some team members to attend their next meeting without having to leave while discussions are still happening during our team stand-up.

The exception to discussing work item details during stand-up is for those that have been color coded hot pink or have discussion notes added in CA Technologies (formerly Rally). These can be color coded by any team member for various reasons. Perhaps it is a high priority item, it has been on the board a long time and we need to get it moving, or it is missing analysis information but somehow got moved across the board. Once everyone gives their accomplishments and plans, these are up for discussion during the remainder of stand-up, if there is time. If time runs out, they are welcome to stay on the call longer to get these discussed.

Mornings are prime time on the Colorado side. There is a short window between when we can get US people into the office in the morning, and the time Poland people leave at the end of their day. This is when we hold our stand-ups (Monday-Thursday), and this is when the majority of our discussions happen. We respect Poland's Friday evenings and choose to not hold stand-ups on Fridays. Convincing some US people to start their day earlier was not easy. As a team, we initially decided to start stand-ups a little earlier than normal. A few months later, after a team member and manager attended a retrospective from Poland, hungry and past dinner time, we decided to start retrospectives even earlier. Coming in early once every 3 weeks is totally doable, and keeps the Poland side from staying late into their evening. Our validation folks are troopers. They never complained.

Even though we started sharing stand-ups and retrospectives, it is sometimes difficult to get to know remote team members, when you only talk as a team for 15 minutes or so each day. We started using video conferences for our retrospectives, allowing us to get to know each other better by seeing the way the team members work with each other, and being able to see the facials and the energy on the screen. This makes retrospectives so much more entertaining and helps bond the entire team. Since the video conference rooms are in different buildings on both the Colorado and Poland side, and they cannot be reserved for just 15

minutes, it is not practical to use them for our stand-ups, but I am working on other ways to incorporate video into our stand-ups.

Spreading the knowledge has also helped decrease cycle time. The developers have the philosophy that anyone should be able to pick up the next most important work item and be successful, regardless in what part of the code the change is needed. Validation initially assigned certain people to test specific pieces of the manageability application. They found it helpful and freeing to start sharing the knowledge of the various pieces throughout, so things are not held up if there are 4 items to be tested in one level of the application, and none in another. They now can also pick up the highest priority work item to validate, regardless of where in the code it happens to be.

Our Product Owner started holding a weekly sync meeting with the technical leads to be sure all priorities were aligned. In addition to getting everyone in sync, we are also able to coordinate and pull in defects that are blocking a lot of validation tests onto our Kanban board to be fixed sooner rather than later, unblocking validation tests, which results in quicker acceptance of stories.

6. ARE WE THERE YET?

I was just about to give up on our 14 day cycle time goal, thinking there was no way we could hit this since our 27 day cycle time increased to 60 days and our team makeup had changed several times. However, patience and persistence paid off. After some team changes and not checking for a couple of sprints, with the help of a teammate, I put together a scatter chart showing cycle times for each completed work item, differentiating between new work and defects. We were all surprised to see an average sprint cycle time of 15.32 days. We could see that our cycle time per work item was getting more consistent, with the exception of a few outliers. Still not quite where we want to be but so much better. Now the challenge was to figure out what we could do to get even more consistent.

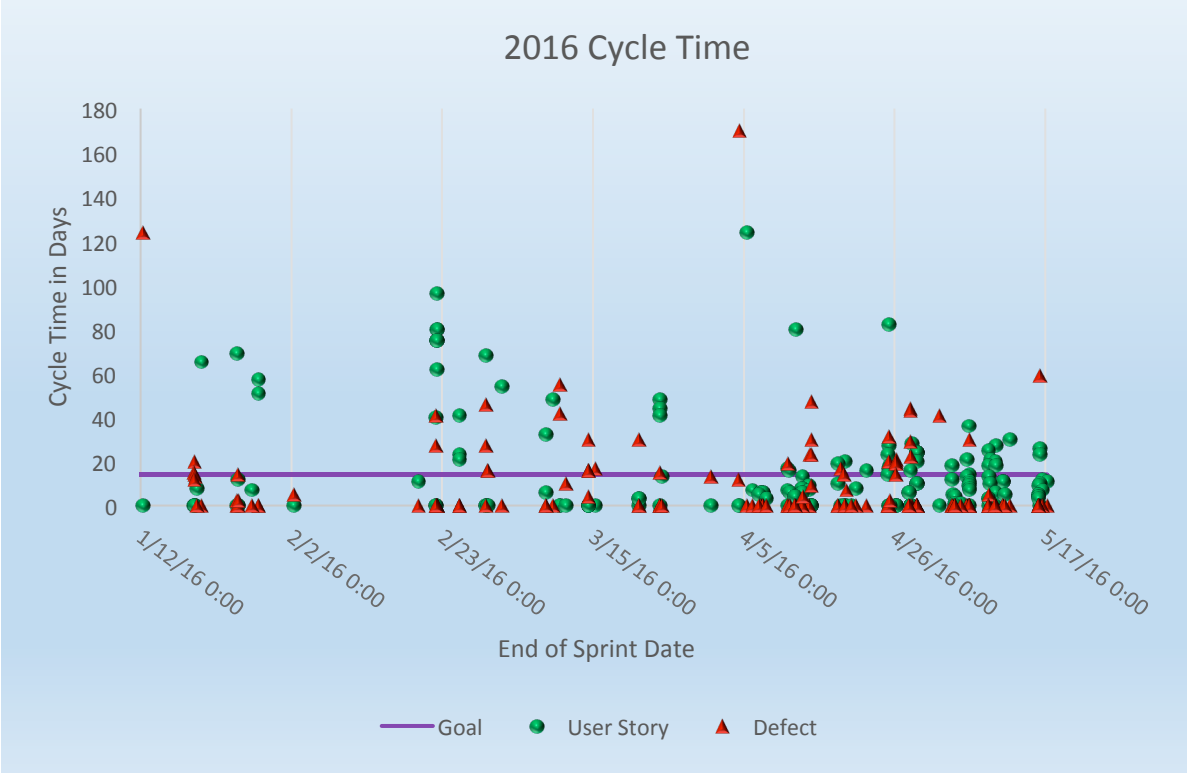

Figure 3. **Cycle Time Scatter Chart**

One of the items in discussion today is that attention to analysis details is still not being recognized. Email is not answered quickly when the subject is not in the radar of the recipient. We are either not having the complete discussions around expected behavior once the work item is complete, or the affected teams are not

fully understanding the impact on code or tests. There are still questions, and even story blockages, happening in the right side of the Kanban board for things that should have been caught earlier.

We are already committed to some current user stories, but now have set a date after these commitments should be complete from which going forward we will not allow code to be checked into our master repository until it is tested by both development unit tests and validation test suites. This should force more thorough discussions during the analysis phase, before checking in code, and should avoid any new defects since nothing will get checked in until it passes both sets of tests.

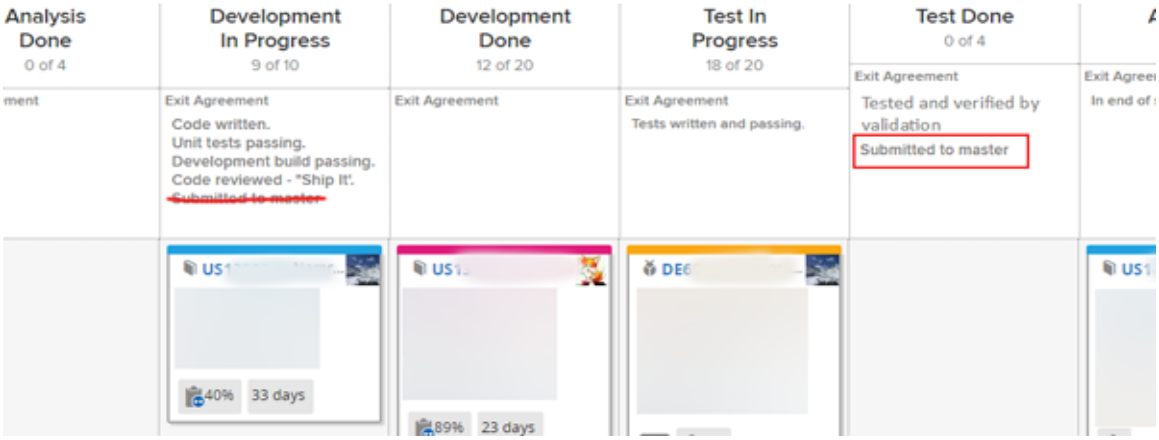

Figure 4. **Future Kanban Board Requiring Test Acceptance Before Code Check-in**

After this date we will take only the highest priority code change requests, and instead spend time cleaning up our defects and working on some development code or build process improvements. By looking at our scatter chart, we noticed that defects seem to have shorter cycle times than user stories, which makes sense. Most of the defects are filed by validation so tests likely already exist for them so there should be little time between modifying code for a defect fix and being able to actually test and accept it. This should help shorten the time between when an item in 'development done' can be moved to the 'accepted' column. We have some logistics to work out on how to handle builds for this, but I am confident that this amazing team will find a way to make it work, as usual.

As of May, the gap between development completing a work item and validation picking up that same work item was about 1.5 (3-week) sprints. So much closer than a year, but we still need to do something to shorten that time gap. It is hard to break down the barrier of "it is our job to do <xyz>". If developers cannot help by designing tests, which one could argue is not a good idea since the tests will work just like our code, there are things developers can do. They can be there to answer questions, and speed the knowledge transfer with rapid style communication such as video or over the phone. The decision we made is to stop new development, beginning on a specific date after our current commitments are complete. We will accept only a critical few code change requests, and just work on defects and some items to improve the development process that would have no impact on validation. We plan to do this for 2 sprints giving validation time to develop more tests without being bombarded with more rapid changes.

I have created physical team Kudos boards that are displayed both in Colorado and in Poland. This is where team members can thank another person for helping them with something, showing appreciation, or congratulating a team member for a job well done. I will admit it is difficult to form new habits, but I am enjoying posting to it, and people always enjoy reading that something they did was meaningful to another. Team members are slowly following my example and posting kudos cards of their own. I am hoping this board gets used by other teams in our location, and can become a building-wide kudos area. Everyone likes to know that they are appreciated.

Figure 5. **Colorado Kudos Board**

I am also starting to look into trying some lean techniques that may work with our current processes. Always looking to improve the way we work, but I guess that's a paper for another time.

7. WHAT WE LEARNED

Communication is the key. The more we increased communication, the smoother things went, and the closer we got to our goals. Communication for us takes place in daily stand-ups retrospectives, knowledge sharing, phone calls, IM, and emails. Real time communication gets the quickest answers, and using the time just after stand-ups is important because of the time difference.

Our window of shared work hours is minimal, and we need to take full advantage of the common working hours. Our team respects each other, and for that I am thankful. I pay close attention to be sure the same respect is given to all members, and we are sure to include the remote team members in discussions and decisions. They are important in the success of our product.

Breaking stories into more manageable sized pieces allows for smaller code reviews, and validation test updates at one time. Smaller work items take less time to complete, which keeps items moving across the Kanban Board. This keeps team members from being tied to a single story for too long which allows the team to more quickly react to priority changes.

Retrospectives continue to be valuable. We acknowledge the challenges in a fun way that keeps the team involved, then we dig into a few selected challenges and find ways to overcome them.

We don't just share a Kanban board. We also share work items. Developers and validators analyze each work item together, and both sides may have tasks on the same work item.

We share resources such as specs, requirements, and best known methods in a common location. Our codebases and builds are available to the entire team.

8. ACKNOWLEDGEMENTS

Thank you to my manager, Denise Vitt, who has shown confidence in me and allowed me the opportunity to lead the team as we take risks by adjusting our processes on a regular basis, as well as having encouraged me to submit this paper. Thank you to our architect and product owner, Tiffany Kasanicky, for not only introducing me to agile processes when I first joined Intel, but also being there to bounce ideas, and help keep the team moving in the right direction. Thank you to my entire team for being BSTEs, for being great communicators, and for always being willing to jump in and get involved in order to better ourselves and our product. Thank you to my shepherd, Rebecca Wirfs-Brock, for her encouragement, valuable insights, and quick feedback. You helped me organize my thoughts and not stress out, which is why I was able to complete this paper.