Talk Ain't Easy - Round-the-World Agile Without *Any* Talk

GERARD MESZAROS, FeedXL PTY LTD

FeedXL is a 6 year old SaaS product that lost its only developer and had to be resuscitated. High labour costs in NA and AU caused us to consider off-shore development with both bad and good results. But we have persevered and now have people in 5 time zones each about 3 hours apart collaborating using cloud-based tools and we *never* actually talk.

1. INTRODUCTION

FeedXL is a small Software-as-a-Service product being built and operated by a team of half a dozen part-time people located in 5 different time zones spread around the world. While it was originally built in Australia, development is now done in India and Romania with technical oversight from Canada with marketing and customer support in Australia and New Zealand. When the original developer of five years decided he'd had enough and wanted to pursue other ventures, I was brought in to take over technical leadership and find people to support and extend the software. I was chosen because I was the brother of one of the principles and an expert in agile software development having done Test-Driven development (TDD) and eXtreme Programming (XP) since 1996 and written a book on automated unit testing [Meszaros]. This report describes our experiences of moving from 3 people in two time zones (2 hours apart) to half a dozen people operating in 5 time zones spread around the world. On top of all that we did it with almost zero face-to-face communication.

2. BACKGROUND

FeedXL is a small Software-as-a-Service product originally built by a single developer, Rod, in Java using Spring, Struts, Hibernate and MySQL over the space of a year and a half to the first deployment. It has evolved extensively over its 6 year lifespan based on user feedback. Customers around the English-speaking world can enter the details of their horses and what they are being fed. They get detailed reports about the nutritional content of the diets and how well they are meeting the horses' nutritional needs.

Over the last few years, Rod had a child and his enthusiasm for FeedXL waned. As a result very few new features were added to FeedXL despite a growing backlog. Finally, as part of a personal pivot in 2013, Rod concluded he was tired of working in startup mode and decided to "get a real job" and try consulting. My sister, who was one of the principles, asked me to take over technical leadership and help them find people to support and extend the software.

3. FEEDXL 2.0

The story of FeedXL 2.0 starts mid 2013 when my sister made the trip back from New Zealand (where she runs a sheep farm) to Canada to see our ailing father. Over the course of her 3 week visit, we had numerous "pivot" discussions about the future of FeedXL without Rod. We scheduled conference calls with Nerida, Sue's horse nutrition partner to discuss the joint venture's future. After several discussions, we made the decision to reboot the joint venture as a new company, FeedXL PTY LTD (Australia) and find people to take over support and development from Rod.

3.1 Getting Started

The first step was to get an understanding of the code base and the skills we would need in whoever was to do the technical support and development. Because Rod had built the system single-handed, there was no documentation to refer to. We needed to create just enough documentation to get people new to the code base started. So I did some Skype interviews with Rod about the existing product code to get an understanding of the technologies used and how the code was organized. I wrote up an architecture overview document based on this while he figured out what tools were actually needed. He set up the development environment on a new laptop and took screenshots of every step so I would know what tools were needed and how to set them up. Then I got the development environment set up on my laptop (Eclipse plus a dozen plug-ins, MySql, Git, etc.) and updated the documentation. Now I was ready to start looking at the code.

I found a project containing several hundred abandoned JUnit tests and started trying to get them to run. I also started looking around for someone to whom we could outsource daily software development. Being a very small company, we really couldn't afford to hire an onshore developer in Australia, the USA or Canada so that left us the option of offshore development.

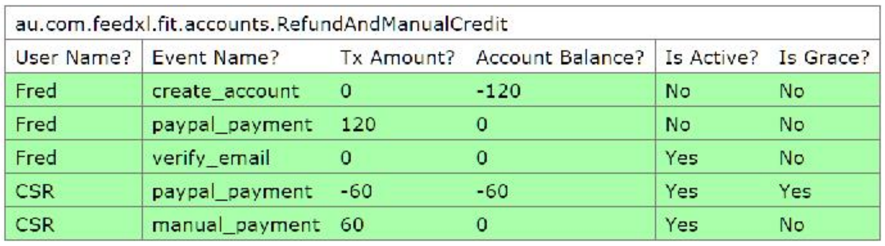
Principles used: Just Enough, Just-in-Time Documentation

3.2 Nailing Down a Development Process

Because of my background as an agile consultant, there was no question about whether or not we would be doing agile. So the real question was "What kind of agile?" Given that our developer(s) would have to support the product in production as well as build new functionality on top of a code base they had very little knowledge of, I decided that it didn't make sense to try to predict what we could get done in a fixed time-box as prescribed by Scrum. Kanban, with its continuous flow of work items and just-in-time queue replenishment seemed like a much more natural fit.

3.2.1 Product Design (What to Build?)

Sue and Nerida have no formal background in software development or even specification. In the past they had described to Rod what they wanted to be able to do and he had filled in the blanks based on the domain knowledge that he had acquired from them. This wasn't going to work with an offshore developer because they wouldn't have the depth of domain knowledge Rod had accumulated so we were going to have to provide them with a more detailed description of what we wanted. Nerida liked using spreadsheets to describe the nutrition calculations so I decided to convert them into one or more Fitnesse test tables that the developer can automate as he implements the logic.



au.com.feedxl.fit.accounts.RefundAndManualCredit					
User Name?	Event Name?	Tx Amount?	Account Balance?	Is Active?	Is Grace?
Fred	create_account	0	-120	No	No
Fred	paypal_payment	120	0	No	No
Fred	verify_email	0	0	Yes	No
CSR	paypal_payment	-60	-60	Yes	Yes
CSR	manual_payment	60	0	Yes	No

Figure 1: A sample Fitnesse example with input and expected output column-sets

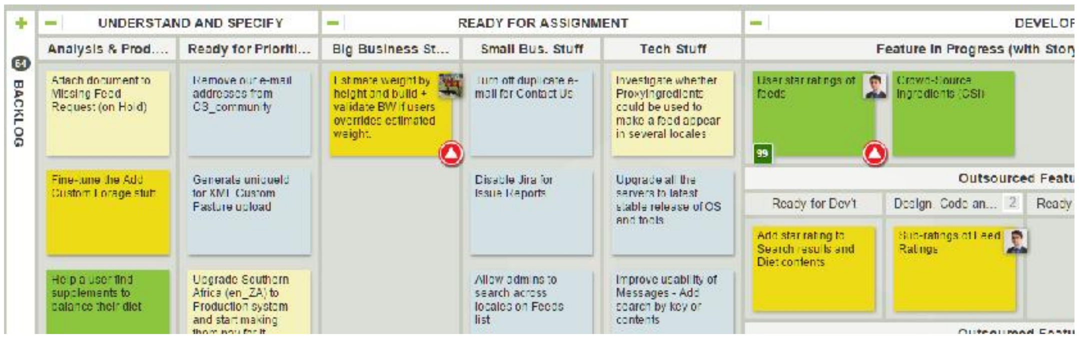
We needed a way to describe the workflows, screen changes, etc. Not just for the developer but also to come to an agreement between ourselves (Sue, Nerida and I). I looked around the web and found a web-based visual prototyping tool that allowed us to build clickable interactive user interface (UI) prototypes relatively easily. Having found one that worked well enough, I went with it based on the lean principle of Just Enough, Just in Time. We would use it in real work as a way of trying it out. A 30-day free trial sealed the deal!

Later, I also used Robot Framework (RF) tests/examples as a way to specify overall workflows using the keyword capabilities of RF to hide the UI details of each step in the workflow while highlighting the differences in the flows or various scenarios. This provides the developer with an overview of the functionality.

3.2.2 Tracking Our Backlog and Work-in-Progress

We also needed a way to keep track of the features and stories we wanted to build. Rod had used Jira for this but we found it too cumbersome to use and the agile plug-in was incompatible with our version. The amount of technical effort required to bring our locally installed instance up to date didn't seem justified so I tried several online tools and settled on LeanKit Kanban.

I customized one of the template boards to provide us with an area where Sue, Nerida and I could put new ideas for features and another area where items currently being built were tracked. In between I created some lanes where we could put the next few items we wanted built. Once an item was placed here, we need to invest some business time in clarifying what we want built before we can put it into the Ready for Development lane from which the developer would pull. As everyone was busy with their other jobs, we would only start to flesh out these ideas a week or two before the developer(s) ran out of work.



BACKLOG 64

UNDERSTAND AND SPECIFY

Analysis & Prod....
- Attach document to Missing Food Request (on Hold)
- Fine-tune the Add Custom Forage stuff
- Help a user find supplements to balance their diet

Ready for Prioriti...
- Remove our e-mail addresses from CB_community
- Generate uniqueId for XML Custom Feature upload
- Upgrade Southern Africa (en_ZA) to Production system and start making them pay for it

READY FOR ASSIGNMENT

Big Business St...
- Estimate weight by height and build + validate BW if users overrides estimated weight.

Small Bus. Stuff
- Turn off duplicate e-mail for Contact Us
- Disable Jira for Issue Reports
- Allow admins to search across locales on Feeds list

Tech Stuff
- Investigate whether ProxyIngredients could be used to make a food appear in several locales
- Upgrade all the servers to latest stable release of OS and tools
- Improve usability of Messages - Add search by key or contents

DEVELOP

Feature In Progress (with Story
- User star ratings of foods 99
- Crowd-Source Ingredients (CSI)

Outsourced Featu

Ready for Dev't
- Add star rating to Search results and Diet contents

Design, Code an... 2
- Sub-ratings of Feed Ratings

Ready

Outsourced Featu

Figure 2: Our LeanKit Kanban Board showing business, input, and development column-sets

We don’t do any formal estimation other than distinguishing between features (which then spawn a bunch of user stories) and small user stories or small technical work items (technical debt reduction and development process/environment improvement stuff.) We will occasionally ask the developers how much work they have left on their current task so we know how soon we need to have the next user story ready for development.

Principles used: Visible Workspace; #NoEstimates

3.2.3 Limiting Work-in-Progress

Each of our developers works on one business backlog item at a time but they may do some technical infrastructure improvement work in parallel if that simplifies their job. We have a large number of items in the backlog but we don’t do any “grooming” work on them until they’ve been move to an Understand and Specify column. By the time it moves from here into the Ready for Assignment columns, we have pretty much committed to doing them. We don’t announce features to our users until they are about to be deployed to our Beta server so we don’t have a concept of “committed”.

Principles used: Just Enough, Just-in-Time requirements definition; Pull-Based Workflow; Single Piece Flow

3.3 First Attempt at Outsourcing – Generic Indian Company

My first attempt at outsourcing was with an India based company specializing in web-based applications and web sites. I checked out the references on their web site and everyone seemed really happy. So I engaged them in discussion and they assured me that they had people with lots of experience using Java, Struts, Hibernate, etc. to build web applications. They sent me a few resumes and we arranged a phone interview (via Skype) of the most likely candidate. His verbal communications skills weren’t the best but we decided to give it a try. He started working for us on March 3rd 2014. I sent him the document Rod and I had created outlining how to get the development environment set up and gave him access to the source code repository in GitHub.

The first task I assigned was to get all the old JUnit tests running again; I figured that this would let him become familiar with the code before asking him to add a new feature. He really struggled with this and after 2 weeks, I gave up trying to coach him on JUnit testing and got them working myself; I asked him to start building the first feature. At the end of the first month I debated not renewing the contract but was loath to abandon the investment we had already made. So we extended for a 2nd month. I had made it clear we wanted daily updates on progress and near-daily code drops so we could test what he was building. I provided a lot of feedback on the software design but saw very little improvement in either behavior or code quality so I finally pulled the plug after 3 months.

So now we knew what we weren’t looking for: “Just” a Java, Spring, Struts, Hibernate Developer. We needed much better communication skills.

Lesson: Technology is less important than communication skills when hiring

Principles (not) used: We had failed to “Fail Fast”.

3.4 Second Attempt at Outsourcing – Developer Found Through Personal Contact in India

I asked a fellow agilista in India, Naresh, if he knew anyone with good communication skills and agile coding skills. It took him a good month but he found Vik, who he offered us half time at a price similar to Amit full time. Having recognized that quality was more important than price, we decided to give it a try. It was agreed that he would work mornings (Indian time; GMT-12?) which would be afternoon in AU and NZ.

Right from the outset, it was clear that his communication skills were much better than A's. As with A, we gave him a HotGloo prototype of what we wanted built and he asked questions via Slack. Initially we tried scheduling at least one Skype-based voice meeting each week but audio quality wasn't great, getting everyone online at the same time proved difficult and we found that answering questions via Slack worked well enough.

We sliced the stories as small as we could and asked Vik to deploy working code regularly. He would typically deploy the code into our test environment at least once per week so we could provide feedback. At this point, we only had JUnit tests for the code and all full-system testing was done manually.

Lesson: Good communication skills are essential, Frequent Delivery (to test environment)

3.5 Third Outsourcing Experience – Csaba

While giving a talk at Craft 2014 in Hungary, I was invited to visit Timisoara in Romania to do a talk at the local user group. While there, I did a bit of free consulting at the host company and one of the things we did was to get a start on building some end-to-end tests using Robot Framework. On the drive back to the nearest train station in Hungary my host and I discussed FeedXL and that led, several months later, to me offering him a part-time job.

When he was ready to start, I gave him access to the documentation that Vik and I had written and asked him to fix any errors and omissions he found while following the instructions. I roughed in a few Robot Framework tests of the overall workflow and proposed that he automate those tests as a way of learning how the application works and the technologies involved.

Initially, he would work 2 hours every evening after his day job. That would have him online mid morning Mountain Time (GMT-7) when I was the only one available. Whenever he had a technical question for Vik in India, he would have to wait until the next day for a response. This slowed things down too much for him so he eventually switched to working early mornings as the work day in Romania only starts about 10am (GMT-2). So he would work on FeedXL from 7 to 9am each morning which would coincide with a part of Vik's four-hour day in India. This allowed them to collaborate much more effectively and also meant he was online mid afternoon Australian time.

Principles used: Incremental Improvement (of documentation and process)

Lesson: Co-location isn't essential but real-time communication *is* important

3.6 Fourth Outsourcing Experience – Student Team

The local university issued a "Call for Projects" for its 4th year computer engineering students to take on as part of a 2 semester course. I spoke to the professor and he told me that they were having trouble getting good student project proposals from industry. We had some "pie in the sky" ideas for functionality that we wanted to build but hadn't had time to really figure out what we wanted. Calculating that it would be a good experience for the students to do some product design, we put in a couple of proposals and one of them was accepted: a way to "crowd-source" the database entries for commercial feeds so that users could get access to new feeds faster without waiting for our CSR (Sue) to enter the data for them. We gave them free reign to do the product design with the constraint being that the data for a feed needed to be validated by either a CSR or other users before it was made available to anyone other than the person entering it and that their design needed to be usability tested. We also asked for the functionality to be delivered incrementally so we could start testing.

The students came up with several alternative solutions which they reviewed with us and landed on a hybrid solution for which they defined a road map with multiple releases. Initially, we didn't get any kind of code drop for over a month and when we did, there were a lot of bugs and misunderstandings. Eventually, they did release software every Sunday evening (MT) and sent us a status report indicating what the release included. We would test it and provide feedback via e-mail. Due to constraints in how the students worked, we did not have real-time interaction via Slack. The plan they come up with involved building one screen at a time and there were enough bugs to keep us from deploying to beta. We kept submitting bugs to them but the fixes for these were mixed in with the new functionality they had developed which made it hard to deploy to beta (we really wanted to but could not do so with any confidence due to the new functionality delivered.)

Lessons Learned: Implement a thin slice of the whole workflow before fleshing out each screen

If we were to do it again, we would insist on starting with a thin slice that covered the whole workflow rather than releases that each release focused on a single page in the workflow. We would also ask them to join our Slack group rather than having their own as we found the level of interaction was much less rich and real-time and that resulted in slower turnaround on issues.

Principles (sort of) used: Frequent Delivery, Just-in-time requirements

3.7 Improving Our Understanding of Business Logic in Legacy Code

After building several new features, our two part-time developers had started to get an understanding of how the system worked overall. But there were many areas they hadn't needed to touch at all and some were critical to understand. One example was the customer account lifecycle: how were accounts created, paid for, graced, and expired? When a customer account ended up in an unexpected state, I had a Slack discussion with Rod about the logic surrounding it and he stressed the importance of order of operations when giving a customer a credit and a refund. I captured the logic by creating some concrete examples which I then ran by Rod and Sue for verification. Not wanting these examples to just passively sit in some documentation directory somewhere, I decided to write them up as candidate Fit[Mugridge] tests. When Vik was doing some work in the general area, I asked him to spend some time automating the Fit tests by implementing the tables as Fit "fixtures". This required refactoring the business logic because it was too closely tied to the user interface but the result was half a dozen self-verifying examples.

Principles used: Incremental improvement (of test coverage)

3.8 Improving Customer Support

In the past we used a shared gmail inbox for incoming customer support mail and we used Jira for ticket tracking. Whenever a user used the FeedXL app to report an issue or request a missing feed, the app would create a Jira ticket to track the issue and it would also send an e-mail to our shared inbox with the "from" set to the user's e-mail. The CSR (mostly Sue) would reply to the e-mail to contact the customer and would update the Jira ticket to match the status. This worked fine when most of the customer support was done by Sue but others found Jira a pain to use and wouldn't close their tickets. Also, finding out whether someone had replied to a request required searching the outbox. As more of us started doing support from various time zones, it became more work to determine which tickets still needed action.

One day we discovered FreshDesk; we set up a free account and tried it out by forwarding the support e-mail address to FreshDesk and "poof" we were up and running instantly. Whenever a user reports an issue or does something that requires manual intervention (like uploading a feed analysis), FreshDesk receives the e-mail from FeedXL and creates a ticket automatically. Any of us can log into FreshDesk and act on the tickets and everyone else will see the full history in one place because all e-mail replies are sent directly from the application. Using FreshDesk has allowed us to bring on part-time helpers as CSR's; they have less to learn and they have a bunch of closed tickets they can study to see how we handle specific situations. Switching to a help-desk application wasn't anywhere on our list of planned work but it leapt to the top of the queue when we saw how much value it would provide for so little investment.

Principles used: While planning is useful, plans are obsolete the minute they are created (continuous planning)

4. RESULTS

FeedXL 2.0 is now confidently moving forward implementing and deploying new features on a regular basis. We have a lot more confidence in our ability to tackle complex new features and to deal with unexpected technical issues. Our around-the-world agile process is functioning well with good teamwork happening despite the lack of any regular voice communication.

4.1 Code Branching Strategy and Build Pipeline

We use the GitFlow branching model to manage all code changes. Developers deploy and test their code changes locally in Tomcat before pushing their code changes to GitHub in a feature branch (e.g. "feature/NameOfFeature".) When the developer wants feedback on what they have built, they deploy their feature branch to the Dev server; several features can be deployed at the same time and they share a database with a subset of basic data an no user data.

When we are happy with the functionality, we merge the feature branch into the main "develop" branch and create a "release" branch (e.g. "release/1.9.5") which we deploy to the Staging server. This acts as a "dry run" of deployment process to ensure that it will run smoothly on top of a complete snapshot of real user data.

Once we are satisfied that everything is working properly, we deploy the release branch to our Beta server which shares the database with our Production servers. This allows our CSRs and early adopters to try the new functionality without impacting the majority of users. Feedback on the feature may result in changes which go through the same process (though the Staging step may be skipped for small changes with no DB impact.) When we are confident that we have built the right functionality and that it is working correctly, we deploy the release branch to our Prod servers.

4.2 A Day in the Life of FeedXL

To better understand how we work around the world without any talk, here is a typical July day in the life of FeedXL. (Because of local daylight saving times, the time difference between us varies by the season. A "*" next to the time indicates Daylight Savings Time is in effect. Colouring courtesy of timeanddate.com.)

Alberta	New Zealand	Australia	India	Romania
Wed 7:00 AM *	Thu 1:00 AM	Wed 11:00 PM	Wed 6:30 PM	Wed 4:00 PM *

I wake up at 7am Mountain Time, grab some breakfast and check my overnight e-mail and Slack messages. If I don't have anything else to do, I'll go into FreshDesk and see if there are any customer support requests that I can handle (login problems, bug reports, etc.) or anything indicative of system problems. If there are any questions for me in Slack from my devs, I may answer them now or later in the day; I have until about 11am before anyone else is likely to be up unless Csaba happens to be working late at night in Romania. I may also have some discussions in progress with my business partners in New Zealand (Sue) and Australia (Nerida).

Wed 1:00 PM *	Thu 7:00 AM	Thu 5:00 AM	Thu 12:30 AM	Wed 10:00 PM *

Sue is up and checking her messages from bed while she drinks her first pot of tea. We may engage in some chatting via Slack in addition to some work-related stuff. I'll tell her if there is anything urgent needing her attention in FreshDesk.

Wed 3:00 PM *	Thu 9:00 AM	Thu 7:00 AM	Thu 2:30 AM	Midnight Wed-Thu *

Nerida is up and checking messages before the children and her work drag her away. We may have a 3-way #business-owners Slack discussion or discuss a feature in the Analyse & Prioritize column of our LeanKit Kanban board.

Wed 10:00 PM *	Thu 4:00 PM	Thu 2:00 PM	Thu 9:30 AM	Thu 7:00 AM *

Vik and Csaba come online. If anything has occurred during the day (their night) that needs immediate attention, we will have left them @named messages in the #dev-team channel in Slack to get their attention. For stories in development, Vik and Csaba may ask questions about how certain cases should be handled from a business perspective (Sue and Nerida's input required) or from a technical perspective (my input required.) Sue and Nerida may be available for short discussions if they are aren't otherwise occupied by their other businesses or Nerida's children. Otherwise they'll respond when they come on later in their day. I'll try to be available to participate in the discussions even if I'm travelling or on vacation.

Midnight Wed-Thu *	Thu 6:00 PM	Thu 4:00 PM	Thu 11:30 AM	Thu 9:00 AM *

I start getting ready for bed and give the developers a "Last call for questions". We wrap up any discussion that need my participation and bid each other "good night" even though it is morning for them. Csaba typically drops off about this time as his 2 hours are up and he needs to go to his day job. Vik will work for another hour or two leaving questions or a status report in the appropriate Slack channel.

5. WHAT WE LEARNED

Some of the following are things we already knew but had the opportunity to "relearn"; others are things we learned fresh.

5.1 Get the Right People on the Bus

Finding the right people to do remote development is hard. Attitudes and communication skills are more important than familiarity with specific tools and technologies as the latter can be learned while the former are innate. Having the right people on the bus and getting the wrong people off the bus quickly is essential. When we started considering bringing on people from other locales, we worried about how we would manage being even more distributed. Now that we have experienced working with people in 5 time zones each at least 2 hours apart, we are less concerned with locale and more about the actual person, their communication skills, and their flexibility and motivation; it definitely helps when people are flexible enough to ensure some overlap of our work days.

5.2 Yes You CAN Build Relationships Online

While Sue and I know each other very well offline, and Sue and Nerida first met in university, the rest of the relationships are almost purely online ones. None of us have met Vik and only I have met Csaba. Despite this, we have built very good relationships on Slack. We have a number of topic-specific Slack channels where we discuss things like the story being developed or the results of testing. We also have a General channel in Slack where we post less work-related items. This is where we trash-talk each others' cricket or rugby teams.

As people come online, they'll say "good morning" even though Slack does provide an indication of who's online at any point in time. And we'll say "G'nite" to each other as we drop off. In between we may enquire about the weather in a particular location or even ask what time it is (because it varies at different times of the year.) We enquire about each other's children and pets. We even discuss politics when something noteworthy happens (like the recent end of the 44 year dynasty in Alberta.)

5.3 Communicating What to Build is Hard

When building FeedXL 1.0, Rod had to learn about the problem domain, and once he was fairly comfortable with it, he would take fairly vague descriptions of what Sue and Nerida wanted and fill in the details. This did result in some less-than-ideal solutions because there wasn't clear understanding between Sue and Nerida what would be built. Rod, being an uber-geek, would sometimes make user interface design choices that the less technically-literate users had problems understanding. But it mostly worked because everyone was in almost the same time zone (Sue in New Zealand is 2 hours ahead of Rod and Nerida in Australia.)

5.3.1 Documents are Good for Consensus-Building

As we moved into outsourcing, I needed to teach Sue and Nerida how to express what they wanted more clearly so that there was less ambiguity in the requirements for the offshore developers to fill in. It was also important in coming to a common understanding amongst ourselves of what we were proposing and gives us something with which we can do lightweight usability testing. Even with this level of "up front" product design, the developers still found lots of things to ask about or test cases we didn't think of. The product design still evolved as it was being built. Developers often proposed changes either via screenshots or by dropping code into the test environment

5.3.2 Text-Based Conversation are Sufficient for Clarification

It is helpful to start the back and forth "conversation" before the developer is ready to start building to avoid being held up by fundamental questions due to our highly time-distributed locations.

5.3.3 Synchronous Text Conversations Beat Asynchronous Ones

While Slack lets us leave messages for others to read whenever they come on line, asynchronous messaging results in the requester having to wait for their answer. How long they have to wait depends on which time zone they are in and where the answerer is located with the worst case being nearly 20 hours (plus another 48 if it happens to be Friday evening in the answerer's location.) Therefore it really helps to have people synchronize their schedules to minimize response times. The other thing that works is to always have a background task that each dev can work on if they are blocked waiting on an answer. This does imply task-switching which reduces focus and efficiency; our developers typically have a technical task (development environment improvement) they can work on while they are waiting for a response.

5.3.4 Voice Communication isn't Essential

Very occasionally (at most once per quarter, but typically only once or twice per year), we'll have a voice call amongst the business owners, usually through Skype. But otherwise all discussions are done using text-based messaging, mostly Slack. Documents are shared via Drop Box and/or Slack depending on their permanence. We also attach documents to LeanKit items. While this makes it easy to find the document associated with a feature or story, in practice this doesn't work as well as DropBox because LeanKit forces you to download a copy before you can open it resulting in multiple copies of the document which can get out of date.

5.4 Incremental Delivery is Essential; Undeployed Code is Waste

With everyone spread around the world, the best way to keep everyone synchronized on what is being built is to deliver working code to our test server very frequently. Keeping the stories small allows tested code to be deployed to Beta where our users would provide new information about it. When the stories were too large or weren't finished (e.g. had bugs) before moving on to other stories, we weren't able to test and deploy to Beta resulting in a backlog (inventory) of untested code.

5.5 Cloud-based Tools are a Game-Changer

It is ridiculously easy and cheap to run a distributed company on the modern internet. Our total monthly bill for all our cloud-based services runs significantly less than $2,000 USD. We use a core set of cloud-based tools to synchronize our activities. We focus on tools which can be set up quickly, have a low learning curve and low usage overhead. Most of these tools can be found through a quick web search, signed-up for with a free account, learned in a few minutes and used the same day. They can also be dumped quickly and cheaply if they don't work out. Very agile.

5.6 Time Zones are More Important than Distance

The cloud-based tools make synchronizing across distance relatively easy. But the bigger issue is the time difference. Not only are time differences a major pain, they keep changing as various time zones switch between daylight savings and standard time. Between the North American summer and winter seasons, I can be as little as 4 hours and as much as 6 hours ahead of New Zealand. But really, I'm 18 to 20 hours behind NZ since it is tomorrow already. Which is something to keep in mind as the week unwinds because it is already Saturday in NZ and India when I get up on Friday morning. I have bookmarks for a multi-time-zone meeting planner in my browser's toolbar [timeanddate] and I consult it regularly.

6. NEXT STEPS

We are bringing on another developer provided by the same company as Vik. He'll be working part-time from home too. As a small company, we would rather have two part timers rather than a full timer because it improves coverage when people are sick or on vacation.

People suggest process changes on a regular basis and we'll try almost anything once. We are continuously improving the level of automation of common tasks. Before FeedXL 2.0, deploying a new release to the server required many manual steps. Now we have a deploy script that can be run from a single command. Whereas we had to run database upgrade scripts manually, now they are run automatically by Liquibase. We would like to automate everything so that anyone who joins the project can do internal and public server deployments at the push of a button. We also want to set up cloud-based load testing so we have a better idea of our capacity and load thresholds. Continuous deployment is still just a dream but I could see trying it another year out.

Now that we have rebuilt our ability to support and extend FeedXL, we have many ideas for new functionality. We have already added some social networking features such as Ratings and Reviews of feeds so our members can share their experiences using a particular feed product. Our university team built the capability for members to propose changes to existing feeds and add brand-new feeds; we need to finish usability testing this set of features and fine-tune it so that we can deploy it to beta.

7. ACKNOWLEDGEMENTS

Thanks to my sister Susan Meszaros for inviting me to participate in FeedXL 2.0 and giving me free reign to try various-sometimes bleeding edge agile practices before I inflict them on my consulting clients. Thanks to Vik and Csaba for humouring my requests to use these practices and for suggesting process and tools improvements on a regular basis. And thanks to my shepherd David Grabel who made many suggestions for improvements to this paper and to Susan Meszaros for reviewing it for both form and content.

REFERENCES

Meszaros, Gerard G. "xUnit Test Patterns – Refactoring Test Code" Addison Wesley Profession, 2007
Mugridge, Rick & Ward Cunningham – "Fit for Software Development"
http://www.timeanddate.com/worldclock/meeting.html