code{4}lib JOURNAL

Issue 32, 2016-04-25

ISSN 1940-5758

How We Went from Worst Practices to Good Practices, and Became Happier in the Process

Our application team was struggling. We had good people and the desire to create good software, but the library as an organization did not yet have experience with software development processes. Work halted. Team members felt unfulfilled. The once moderately competent developer felt frustrated, ashamed, helpless, and incompetent. Then, miraculously, a director with experience in software project management and an experienced and talented systems administrator were hired and began to work with the team. People in the group developed a sense of teamwork that they had not experienced in their entire time at the library. Now we are happy, excited, and energetic. We hope that you will appreciate our "feel-good" testimony of how excellent people and appropriate processes transformed an unhealthy work environment into a fit and happy team.

By Amanda French, Francis Kayiwa, Anne Lawrence, Keith Gilbertson and Melissa Lohrey

Introduction

This paper is unusual in that we have tried to incorporate the unique personal perspectives of the members of our team. Throughout the paper, thoughts and opinions of team members appear in blocks of text that address some aspect of our technology or processes. The voices included are those of a systems administrator, a director, a collections specialist, and a software developer.

Our focus is primarily on how we changed our workflow, and we will also give details about the technologies that support our new process.

Background

We are the team responsible for VTechWorks, the institutional repository at Virginia Tech, hosted by the libraries. Our team is made up of a director, a systems administrator, a metadata specialist, a content specialist, and a software developer, with additional assistance from student employees and occasionally other teams. Due to the nature of library work, several of us have many other responsibilities. Also, because of this workload, as well as the openness of our new system, the roles are less rigid than our titles imply; for example, we all have access to do software development work, and we all have access to change metadata.

VTechWorks was brought online using the open source DSpace software in the Fall of 2011, at a time when only one of the current team members was employed at the library. It was a project of the former Digital Library and Archives department, as a challenge by the new dean that had started in the Spring: build an institutional repository by the end of the year.

Although the library at Virginia Tech had a history of innovation in information systems, with an early ETD system, the VTLS ILS, and ImageBase, the preceding dean had chosen to concentrate efforts in areas other than technology development. Therefore, the technology infrastructure wasn't as robust as one might have expected from an organization in 2011, and some of the culture and processes to support software development weren't yet in place, even though there were already skilled technologists and managers. The decision was made to launch VTechWorks, even though IT infrastructure and processes weren't fully developed. This was a good decision at the time, as we were in need of a repository and couldn't wait: another technology group on campus, outside of the library, had already started an institutional repository based on Fedora 3 technology, but this repository failed to thrive due to problems with content recruitment.

Today, VTechWorks has grown to over 50,000 items, with initial boosts by repackaging previously existing digital collections, such as Electronic Theses and Dissertations, into the repository. Equally important to those of us who work on the system, our technology infrastructure and work processes have also grown and improved over the years. We'll show what's changed and discuss our unique perspectives.

Systems

When VTechWorks was first launched, there was a single development server. Due to a lack of hardware resources, the server was actually an old desktop computer that had been reclaimed from an employee that had left several years prior.

There was initially no source control, but this was set up soon by a systems administrator after the developer arrived.

The developer was the only person on the VTechWorks team with access to read or change the source control, and the developer handled all deployments to test and production systems.

When Francis (our systems administrator) arrived, he made great strides in our systems that ultimately helped with our workflow. Our source control was moved to GitHub, the standard site for open source software development, based on the Git version control system. All team members now have read and write access to the code. Additionally, it is easier for us to collaborate with other organizations using DSpace. Shortly after the source code was migrated to GitHub, we also began tracking active issues with the GitHub issues tool built into the project.

We do have some files stored in a locally hosted instance of GitLab, an open source alternative to GitHub, due to security concerns, including a new system for quickly provisioning test instances of DSpace. Francis created our Local Development Environment (LDE) infrastructure so that all team members have a VTechWorks development environment running on their own machines, in a VirtualBox VM. The LDE system is based on the Vagrant + DSpace project, but uses Ansible instead of Puppet.

VirtualBox allows us to run virtual machines, which in our case emulate servers, within our standard Mac systems that we were already using although some team members with older systems needed to upgrade to at least 8GB of RAM. We use vagrant in our project to configure and deploy the virtual box machine. For our Local Development Environments, the work that vagrant performs includes provisioning a virtual machine with 4GB of RAM that runs Ubuntu Linux, setting up a private network so that we can connect to the virtual machine from the host machine, granting access to the DSpace source code folder, and running an Ansible playbook.

Ansible, a deployment and configuration automation tool, allows us to perform further configuration on the virtual machines. First, Ansible installs all of the software prerequisites for building and running DSpace, including Java, Apache Tomcat, Ant, PostgreSQL, and others. Ansible then configures PostgreSQL and initializes the DSpace database. Finally, Ansible performs the DSpace install process and starts the Tomcat application server when the install is completed.

The Local Development Environment System has worked well for us. Now, every team member has a place to experiment and unfettered access to look at and modify the source code. Each person can do this at any time, even when the official development server is in use for testing or demonstrating new features. When we're finished experimenting, we can easily save the state of our local development environments to disk, or delete it completely using vagrant.

At the time of these changes, the old desktop system that we previously used as our development server was retired. It is now running on server class hardware, and is configured with the same technology as the LDE, so that our local development and production systems are very similar.

In addition to the development server, we now have a pre-production system to use for user testing while integration work is happening on the development server. The production server was also migrated to a similar setup, though it does have some differences: for example there is a dedicated database server on a separate node.

The deployments to our integration, pre-production, and production servers are now handled by systems administrators, typically at the request of Amanda, who serves as the product manager.

	First Try	Transitional	Current
Development server	Old desktop system	Old desktop system; developer and student developer also created individual setups on personal computer	Vagrant, Ansible, and VirtualBox used to give similar development environments to all team members.
Source code	No source control	On private svn server, then private git server.	Most of the source is on GitHub, with a few things specific to our infrastructure on a private GitLab server.
Issue	No issue	Issues on Redmine. Many issues went unentered.	Issues stored with source code on GitHub and GitLab, or in the backlog spreadsheet.

tracking	tracking	Issues were not reviewed regularly.	Nearly all issues and change requests recorded. Issue lists reviewed regularly.

Documentation

Initially, there was no official set of documentation. Notes, when available, were shared via e-mail from the systems administrator to the developer.

Eventually, team members began to create documentation and add it to a private Redmine wiki, but this was not seen as a priority.

When Francis created our Vagrant and Ansible setup for DSpace, he created detailed setup documents so that the environment could be recreated. The setup instructions were stored along with the source control project. Others improved the documentation as it was used, making edits for errors or clarification.

This initial documentation project served as a basis for most of our documentation efforts. Any time we have a new system or a new procedure, we document it, and in most cases put the documentation in source control, either in README files or on the project wiki.

This works well for us because everyone our team has a strong urge and motivation to document anyway. Each of us is detail-oriented. It's also reassuring to know that there are others on the team that can handle work in case we need to take time off or devote ourselves to other projects.

Documentation has become an intentional effort now, and we are considering dedicating entire sprints to documentation improvement efforts.

	First Try	Transitional	Current
Typical documentation practice	Shared via e-mail on a need to know basis.	Some documentation in Redmine wiki and in source control, created as team members had time, but not as a priority.	A big set of documentation shared on public GitHub and private GitLab projects. Other documents shared with all team members on Google Drive. Team members improve documentation as they make use of it. We are discussing dedicating entire sprints to improving documentation.

Openness

As our team has evolved, we have become more open. We're open with each other, in that we share what we are working on and how we do things. We regularly work with others in the library to improve metadata, content, and systems. And we've again started to work with others in the DSpace community, attending conferences, contributing small changes, and testing fixes.

	First Try	Transitional	Current
Source code	Source code stored privately and access available only to developer.	Source code stored privately. Write access available to developer, and to student developer after convincing the Information Technology department that the student needed access.	Most code stored in public, on GitHub. All team members have read/write access to the source code, and contribute. The team has resumed making small contributions to DSpace core.
Intra-Team Collaboration	It felt like there was no person on the team – developer, manager, content specialists, systems administrators – that had the power to make positive changes to the product.	Developer hid from other team members.	Team members are productive during meetings, assist each other as needed, and enjoy the company of each other. Team members feel free to

			share suggestions for improvements.

Students

Previously, the student developer position reported directly to the software developer. This worked well when the full-time developer was able to dedicate sufficient time to training the student, but there would be difficulty when students were hired at a time when the full-time developer was not available to train or guide them. Additionally, the student developer would sometimes be given independent assignments from multiple team members, without any guidance on how to prioritize those assignments.

The student developer position is now fully integrated with the team. She participates in the stand-up meetings when it fits with her work schedule. Team members make sure that there is always a list of tasks for her to perform.

Additionally, in recognition of the value that student developers can bring to our project, our student developer wages have gone from $11.00, to $13.00, and now to $15.00 an hour within the last year. This also helps us to be competitive with other student jobs on campus.

	First Try	Transitional	Current
Team Dynamics with Student	Assistant to developer	Work with other team members as needed.	Integrated into team meetings, processes, and priorities.
Student developer pay	Unpaid intern	$11.00 /hr	$15.00 /hr
Student recruiting process	Friend of a friend	Developer lists job, collects resume, performs telephone interview	Job listed in multiple places. Scoring matrix created for candidates. Programming knowledge questions asked. Entire team plus one outside member evaluates candidates.

Testing and Fixing

Initially, there was no test process, formal or informal. After the first set of changes, a simple theme change, the site was displayed in a meeting of library directors for feedback and redesign. For later changes, there was no test process, other than testing performed independently by the software developer.

Bugs and change requests were initially handled as they came in, but this system eventually failed with over 100 open, unaddressed, and unprioritized tickets in the tracker, and a single developer, with other responsibilities, available to handle them.

We've improved upon these practices: now, all team members have input on what they will work on, but items for a potential sprint are prioritized with the assistance of the team at the sprint planning meeting, and based upon team discussion.

With all team members contributing, and a documented deployment process, we are now able to release to the production server approximately once a month. Each change is reviewed by someone else on the team. We test before and after each production deployment, and have created a standard test script.

	First Try	Transitional	Current
Frequency	Problems resolved as they are reported.	Test on major releases (about once per year)	Test branches Test before all deployments (about once per month)
Test location	Test on production server	Test on development server	Test on local development environments, on at least one test server, and on production
Test depth	Serendipitous, cursory testing	Test a checklist of items based on previous problems and basic uses.	Use a written test plan based on a longer list of previous problem reports, Add serendipitous testing. Experimentation with Selenium IDE tests.
People	Few people,	Team, plus a cataloguer and an open	Student developer for minor releases.Team for major releases.

involved	incidental.	access specialist.	Preparing for additional usability and user certification testing.

Workflows

Amanda French, our new director, worked to improve workflows on our team.

We went from having no team workflow – essentially chaos when many problem reports and change requests came in – to having a highly organized workflow based on the scrum framework.

The elements of scrum that we find particularly helpful include:

- product backlog – a place to document all potential upcoming changes to a project
- sprints – time-bounded development units where specific project tasks are accomplished
- planning meetings – meetings held to set accomplishable goals for each sprint
- task board – a visual representation of the status of each task in the sprint. Ours is made of post-it notes on a whiteboard, with areas for to-do, doing, done, and waiting.
- Stand-ups – frequent, often daily brief meetings where participants discuss what they've done since the last meeting, what they'll work on now, and any obstacles

Many of our processes and technologies have improved gradually over time, but the best and most revolutionary changes came when two new hires came in Spring of 2015 and modernized our team.

One of these hires, Francis Kayiwa, Senior UNIX Administrator, worked on modernizing the systems used for development and production. His thoughts, along with those of Amanda French, and Anne Lawrence and Keith Gilbertson, both of whom have been involved with the VTechWorks project for several years, are offered below.

Francis Kayiwa's Thoughts: Delivery of Service

"*I am not just a proponent of empowering everyone in the project. I would actually rather not be involved in any project that doesn't aim to do this.*

My entire career has been around maintaining other people's work. Over time I have worked towards ensuring stakeholders are not only given power to work in a way that suits them, but more importantly are reminded of what part of the project they will be responsible for maintaining. The benefit here is no one person feels the burden of the project, and when done right everyone will have a good understanding of how to avoid disasters.

This kind of work now falls under the auspices of the "difficult to define" DevOps. For this article we will describe it as the ability for Developers, Application Administrators, and Operations people to meet periodically to define the way to work most productively.

My goal as part of the operations team was to be able to make our institutional repository, VTechworks, be a highly repeatable environment for all involved in the project. I wanted all involved to be able, with the minimal amount of documented effort, to stand up and tear down a replica of our production institutional repository. I wanted the entire group to understand that there's nothing magical about the production server. In order to do this we needed to define what our preferred development environment would look like.

We selected Vagrant and Ansible tools to do this. We walked through the "Dspace Way" of setting up a new instance. We worked towards making sure setting up a new instance would require no human action in order to have a replica of the environment. This process was iterative and continues to be improved upon.

The most important thing I brought to all of this was my ability to facilitate. It was more important for me to have everyone comfortable working with a repeatable environment than it was for me to use any particular technology."

Amanda French's Thoughts: Implementing Scrum

"*I began at the Virginia Tech Libraries as Director of Digital Research Services in late April, 2015. My director-level position includes responsibility for several areas, the two largest of which are digital humanities and the institutional repository. My background is really in digital humanities, but I've picked up some theory and practice of project management along the way, especially but not solely project management of software development projects. I first heard about the Agile Manifesto and "extreme programming" in 2008 when I was at Emory Libraries running the reserves service and managing the development of a locally developed reserves system there, ReservesDirect. Really, it was the developer there, Jason White, who taught me about how to manage software projects. At his recommendation I read the classic book The Mythical Man-Month, and I remember nodding vigorously at many of its insights. I also remember beginning to hear the terms "agile" and "iterative" used much more often in the broader culture at that time to indicate desirable qualities. At Jason White's suggestion, we adopted a vaguely Agile sprint-based workflow, but honestly it wasn't strictly necessary, since there were only the two of us involved with developing the software (and both of us had other projects and duties), which made communication and prioritization and evaluation simple. We also weren't subject to requests for new features or even bug fixes, since the production system was relatively stable and users were used to it.*

I have, however, spent a lot of time around software developers in the last ten years, first at NCSU in the Digital Library Initiative group, then at Emory, then at the Roy Rosenzweig Center for History and New Media, and what has become increasingly clear just from listening is that

- *there are an infinite number of possible features and potential bugs in any piece of software,*
- *it isn't possible for most users to judge how much time and effort developing a feature or fixing a bug will require, and*
- *it's often not much easier for a developer to judge how much time and effort developing a feature or fixing a bug will require, either.*

Coming from an English PhD program, too, I couldn't help but liken the difficulties with writing code quickly to the difficulties of writing anything quickly; there are an infinite number of topics you could write about about, an infinite number of books and articles you could read to help you write it, and it's very difficult to judge how long it will take to answer a research question you pose yourself or how long it will take to produce a passable piece of prose. As a teacher of writing, I had a great deal of success asking my students to write one-page weekly papers instead of one or two longer papers each semester; the papers were better, the students became more used to writing and got much more feedback from me and from each other, and leaving writing until the last minute became at once more difficult and less disastrous. That assignment might have been called "an iterative process" if it were in the tech sphere. Similarly, when I learned about the unconference model and later headed up the unconference THATCamp (The Humanities and Technology Camp), I appreciated the innate agility of it: by waiting until the very morning of the event to decide what to do, and to make the decision together, participants would be able to address issues that were on their mind that very day instead of having to read aloud a paper that they had first thought about several months before.

Therefore, when I came to Virginia Tech, where I was for the first time in charge of more than one person at a time doing software development, and where (as I soon learned) there was a lengthy backlog of feature requests to address, it seemed to me that I should learn a lot more about methods based on Agile principles. I read Scrum: A Breathtakingly Brief and Agile Introduction by Chris Sims*; I read The Elements of Scrum by Chris Sims and Hillary Louise Johnson; and I perused the slides from a 2011 Digital Library Forum presentation by Naomi Dushay (on behalf of Bess Sadler), Tom Cramer, and Jessie Keck – Agile Methodology, Innovation, and Quality. Because I was lucky enough to come in at the same time as a new Unix systems administrator, Francis Kayiwa, who wanted to overhaul the whole technical backend of the boxes our systems resided on, we did that first, which already meant that we would have new GitHub-based workflows. Once we were through upgrading DSpace, getting set up on GitHub, and getting used to developing on local virtual machines, the time seemed right to introduce a new way of working. It didn't hurt that just at the same time we were due to move into new offices, as well: why not get all the changes over with at once?*

Before I decided to adopt Scrum officially, however, I spoke with a few people beforehand about potential problems, most usefully the head of Virginia Tech University Libraries' IT Services, John Borwick, who had participated in Scrum processes at Wake Forest. There were three major concerns I had:

- *Scrum is really designed for teams who are developing new software, not for teams who are supporting production IT systems. I wasn't sure whether Scrum processes would work in that context.*
- *The VTechWorks team doesn't only work on the DSpace software; we identify and acquire content and check metadata quality and respond to user requests as well. Some reports averred that Scrum worked well for other work besides software development, but I wasn't sure it would.*

- *Much of the seemingly endless debate around Scrum online concerned whether it was okay to depart from the established roles and meetings and structures; many people claimed that if Scrum failed, it was because that team didn't do it correctly. I knew, in particular, that my own role mapped most closely onto Product Owner, but that we didn't have a Scrum Master.*

John's opinion on these three points was that Scrum would probably work for production IT (although he's more a fan of ITIL, about which he pointed me to some fresh reading), that it would probably work for other types of work as well, but that it was crucial to have a Scrum Master if at all possible. He also advised that the Product Owner role might best be filled by a user.

Ultimately, I decided to forge ahead, and to take the risk of doing without a Scrum Master. I put together a brief presentation for the four-person team about Scrum, identified a place for a physical Task Board, created a product backlog in a Google Doc spreadsheet from the 100+ issues in the old issue tracker, and put a bunch of meetings on the calendar: "sprint planning," "sprint showcase," "sprint retrospective," and of course the "daily stand-up," which in our case is at noon. Together we decided on a month-long sprint cycle (sort of: more on that below), and we began sprinting in September 2015. At Keith Gilbertson's suggestion, our sprints and sprint releases are named after animals in alphabetical order (aardvark, bichon frise, chipmunk, dogue de Bordeaux, emu, frog …); we plan our work at the beginning of the calendar month and usually (though not always) deploy changes to production at the end of the sprint and calendar month.

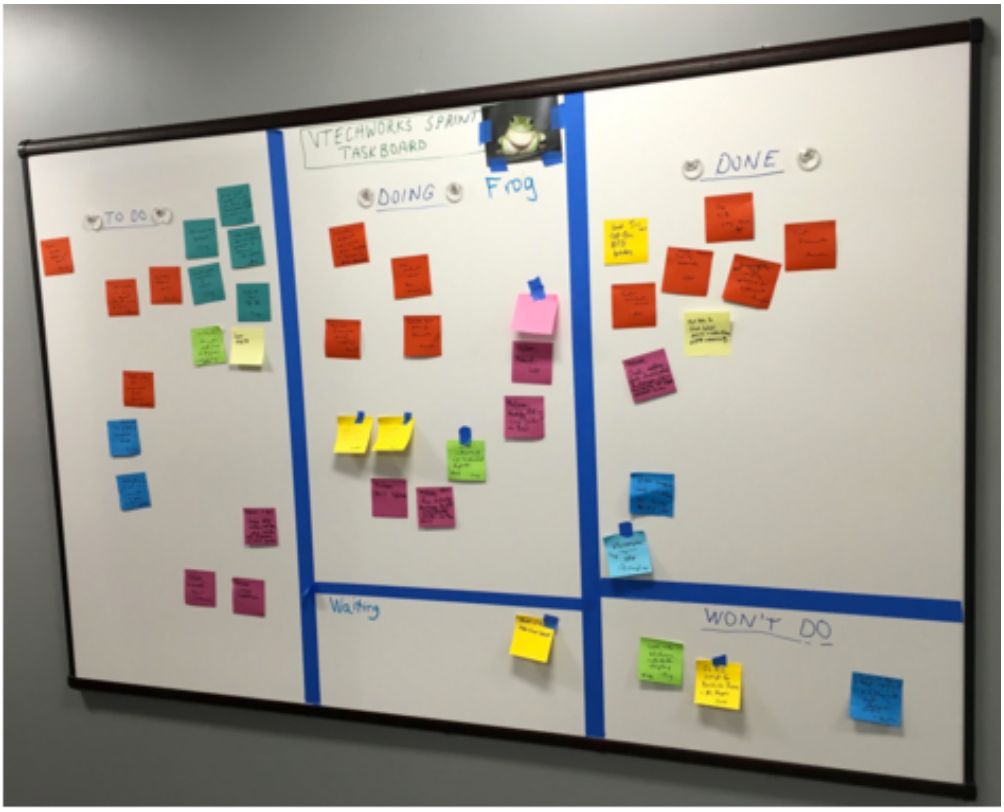

Figure 1. Taskboard.

I'm definitely happy with the way it has turned out; we're making significant progress on the backlog of issues, and I think that Scrum does work well even in the context of a production IT system and for non-technology work such as correcting metadata or acquiring content. I also think that while it would be nice to have a Scrum Master, someone who would make sure that I'm not asking too much of people in my role as Product Owner, we're managing pretty well without one. To make up for the lack, I try to give team members a choice of what to work on and I try to check in often to make sure they're not overwhelmed.

Nothing is perfect, of course, and there are a few new problems that have been introduced. The first problem is that running VTechWorks is only part of my job, and the Scrum process is so powerful (those daily meetings!) that it tends to concentrate my efforts on VTechWorks at the expense of other things I should be doing, things that are not part of the Scrum process. It's still not entirely clear if we have a three-week sprint or a four-week sprint: I tried to set it up so that we had a three-week sprint and then a week off so that I could focus on other things and the staff could too (learn new things, write things, clean up old tasks, revise documentation, take a break from each other), but in practice it doesn't seem to work that way, and we wind up with four-week sprints in which the last week is devoted to testing and bug fixing and code review and deployment and, yes, a few things we didn't get done. The second problem is that while Scrum does function pretty well for non-technology work, there is a sort of

emphasis on and privileging of the technology work that is hard to overcome. I'm not sure whether that's innate to the Scrum process or is just a problem with my own prioritizing: some month or months soon, I'd like to plan almost the whole sprint around content acquisition, metadata correction, or learning new skills, or writing and revising and migrating and archiving documentation. But meanwhile, we're still very focused on learning DSpace and recreating or fulfilling old feature requests. The third problem, and perhaps the least important, is that we're still not very good at judging how long it will take to accomplish particular tasks, nor at how much we'll be able to get done in the course of a month. We decided not to bother with burndown charts and other tools designed to help us estimate such things, because while we do have a lot of feature requests and projects, so far we haven't had a firm deadline for any particular feature or project that we've been unable to meet.

What is clear is that we're getting a lot done, and even when we don't get everything done that we thought we might, well, next month is a brand-new month, filled with possibility, and we've gotten a lot done in the past month, which is satisfying. And even sometimes fun."

Anne Lawrence's Thoughts: The Scrum Experience

"*I have degrees in Industrial Engineering and Library Science, with an interest in efficiency and the continual improvement of processes and systems. I appreciate that I can contribute to the improvement of our repository in a variety of ways: improving the software, improving workflows, administration, ingesting content, etc.*

I have practiced Getting Things Done, somewhat loosely, for about 8 years and believe strongly in the utility of getting things out of my head and into an external system. Our system allows this principle to extend to our work team. This system not only allows me to free up my brain of things to remember, it facilitates communication of my information and detail to our team. Similarly, I am informed of my co-workers' and supervisor's projects and priorities.

Everyone's ideas and suggestions are welcome; they may not all be implemented but will be considered. We are encouraged to take on projects that are a personal stretch and allow us to build skills and knowledge. Our git/vagrant system creates some overhead to working, but provides a good safety net for developing and experimenting. I thought the number of meetings would be overwhelming but it's OK. I greatly appreciate that meetings are generally held to their allotted time (generally 15 minutes or 1 hour).

Clear communication is required to do complicated or more involved tasks. High level wishes have to be translated into coding details. GitHub, daily stand-ups, and the planning meetings go a long way to meeting that need. Three of us are in the same physical space, which is a help. But some members telework, which can inhibit communication. So far, that has not been a big problem. They communicate via email, video, and the Slack-like tool Mattermost. We seem to have a good level of communication about issues.

Our director acts as product manager and scrum leader. It might be better with a larger group, but it works and allows each person time to report on their work and express their ideas.

I would prefer if all our Git work were in GitLab (limited to the Libraries staff) rather than in the public GitHub. We use a large percentage of the functionality in GitHub for code development, the wiki, and issue tracking. GitHub has very nice features which make it easy to add screenshots, create pull requests, relate issues and pull requests, etc. We used Redmine previously for issue tracking and to create a wiki. However GitHub/Lab integrates better into our code development work, particularly since we have a fork of the DSpace GitHub repository.

The backlog is well implemented as a spreadsheet on Google Drive, which makes it easy to edit, while ensuring that it is backed up. Separate sheets for the customer help form, content, technology, metadata, and documentation projects aid us in focusing on each category at a time during product backlog meetings. Team members are free to add issues to it, recording requests from users and their own ideas or tasks. As I moved a sizable portion of my personal to-do list to the backlog, I appreciated that the list of tasks I had carried personally would now be shared and prioritized as a group. The product backlog allows me to list items without bothering anyone in the moment, knowing that they will be considered in the product backlog meeting.

I like the daily stand-up for several reasons:

- *It's short — I can do anything for 15 minutes!*
- *I prefer to stand and stretch after sitting so much.*

- *Tasks are much easier and much less uncertain if I can check in with others, especially my supervisor, as questions and concerns arise.*

Since we all prefer to work in uninterrupted stretches, I am sometimes hesitant to disturb co-workers, even to negotiate a time to meet. And tracking down people can be even more difficult. Rather than not communicate or delay communication too long, it's much easier and more efficient to plan it into the day, as stand-up meetings. I can save up most issues for the meetings, without having to disturb co-workers at other times. I give the group a summary of my work and bring up issues that would benefit from communication with them. As needed, I provide additional documentation and details in GitHub issues, email, or Google Docs. And I truly appreciate hearing from my co-workers about their work and news at this meeting. Because they are so useful, we have stand ups during our "off sprint" week. Our manager tends to concentrate on her other responsibilities during that week, but since we stay on VTechWorks during that week, communication is still needed.

The monthly policy and procedure meeting isn't part of the scrum, but it allows us time to improve our procedures and discuss policies and other issues in depth. I find it useful to have this time away from strict task completion in order to handle administrative issues of the repository.

So, in short, I am very pleased with our current system and team climate. I feel it fosters communication, a team spirit, and improved results."

Keith Gilbertson's Thoughts: Redemption

"*For most of the prior three years at Virginia Tech, I had been the bad egg and the bottleneck on the VTechWorks team.*

I started my career working primarily as a software developer. Prior to libraries, my jobs had been in either very small, very focused organizations, where communication was simple, or in very large organizations with well-defined software development processes.

My work from 2007-2009 on institutional repositories at OhioLINK (the Ohio Libraries and Information Network) served as a bridge into software development for academic libraries. At the time, a significant amount of software development, configuration, deployment, and maintenance happened at OhioLINK. The organization was small and flexible, but very productive. For example, when I left OhioLINK, a team of one developer, one manager, and a shared systems administrator were providing installation, customization, support, and upgrades for approximately a dozen institutional repositories in the state, with once-a-month guidance from a team of librarians serving at Ohio universities.

When I began working in academic libraries proper, I experienced somewhat of a culture shock. Most libraries aren't sized as enormous corporations, but they can be ambitious, and often have multiple priorities that compete for staff time. Also, while I don't know if I could say that the same is true today, I will risk stating that 10 years ago, many libraries, even when staffed by talented technologists, had organizational cultures that would have made completion of software development projects difficult.

When I arrived at Virginia Tech, it was clear that there were extremely competent managers with a history of success, and that the organizational competencies in software development didn't yet match the technology aspirations that came with a new dean.

I failed in that environment. I was overwhelmed with all of the responsibilities and challenges. I felt incompetent because I wasn't able to complete everything that everyone wanted from me.

Now, things are much better. Because of the changes that we have made in our technology and processes, everyone on the team participates in improving VTechWorks. I'm not overwhelmed, and I feel confident again that I can help others who will also help me. I'm also available to work on other projects that will benefit the library.

The most astonishing thing for me is that I have learned that I can be a "people person". I used to seek time alone because I feared that every interaction with people would create demands that I wouldn't be able to fulfill; now I'm a team player and enjoy working with my colleagues again, so much so that when I must miss a meeting, I feel disappointment instead of relief. I'm happy to be here."

Conclusion

The primary lesson learned from our experience is that having a process for our team, much like we have processes for our own

individual work, is helpful. Knowing which items we are going to concentrate on at any given time helps us to follow through with our own individual work processes, because we're less likely to be distracted by incoming work that might seem urgent because it has landed in our email inboxes. Instead of distracting us, new feature requests go into our backlog spreadsheet, and newly reported bugs – unless they are showstopping – are placed into our GitHub ticketing system. The most important requests will be selected for attention at our next sprint planning meeting.

The process is not perfect. For example, some group members have expressed concern that we under-emphasize task prioritization. Members are assigned specific tasks each week, and each task is, in theory, equally important. In reality, this is often not the case and some projects are clearly more time-sensitive or essential than others. Due to the challenges of correct time and effort estimation, often not all of the work selected for a sprint can be finished. Therefore, we recommend that groups trying scrum also assign priorities to tasks within the sprint planning meeting.

And we do recommend that groups try scrum, and adapt it to their own needs. In particular, we recommend documenting all change requests in a product backlog. We recommend sprints, and holding official sprint planning meetings to determine, based on the product backlog, the goals of each sprint. We recommend selecting a primary person of responsibility for each task during the sprint, and posting this information on a task board, whether this is done with post-it notes or virtually.

Finally, even those of us in our group who initially disliked meetings recommend holding brief, regular stand-ups where each group member quickly explains work done since the last meeting, what will be worked on next, and any obstacles faced. This gives group members an opportunity to share victories, and to ask for help when it is needed. In our case, this regular, structured contact where we report on our successes and help each other achieve goals has allowed some members to have a stronger sense of being a part of a team than they have ever felt before.

Resources

Allen D. 2015. Getting things done: the art of stress free productivity. 2015 ed. Penguin Books.

Brooks FP. 1995. The mythical man month. Anniversary ed. Addison-Wesley Professional.

Dushay N, Sadler B, Cramer T, Keck J. 2011. Agile methodology, innovation, and quality. Presented at 2011 Digital Library Federation Meeting; 2011 November 1; Baltimore. Slides available at: http://www.slideshare.net/ndushay/agile-for-digital-library-projects.

Selenium IDE. [Internet]. SeleniumHQ; [cited 2016-3-20]. Available from: http://www.seleniumhq.org/projects/ide/.

Sims C, Johnson HL. 2012. [Internet]. Scrum: a breathtakingly brief and agile introduction. Available from http://www.agilelearninglabs.com/resources/scrum-introduction/.

Sims C, Johnson HL. 2011. The elements of scrum. Dymaxicon.

Pottinger H, Donohue T, Prado R. Vagrant + DSpace. 2016. [Internet]. Available from: https://github.com/DSpace/vagrant-dspace/blob/master/README.md

VTechWorks Wiki. 2016. [Internet]. Available from: https://github.com/VTUL/vtechworks/wiki.

About the Authors

Former CLIR Postdoctoral Fellow **Amanda French** is currently Director of Digital Research Services and Associate Professor at Virginia Tech University Libraries, where she is helping to build digital humanities infrastructure and is running the institutional repository, VTechWorks. From 2010-2014 she was THATCamp Coordinator and Research Assistant Professor at the Roy Rosenzweig Center for History and New Media at George Mason University. Her particular expertise consists of making humanities content (both cultural content and scholarly interpretation of that content) openly available online, as well as introducing scholars to the various methods of and issues with making humanities content openly available online.

Francis Kayiwa is a UNIX/Linux systems administrator with over 15 years of experience, and increasingly serves as a software developer and project manager with expertise in Perl and Python, and exposure to Ruby. His current interest is DevOps automation of server infrastructure using Ansible and Salt, while exploring Chef/Puppet.

Anne Lawrence is a Repository Collections Specialist at Virginia Tech.

Keith Gilbertson is a Digital Technologies Development Librarian at Virginia Tech.

Melissa Lohrey is a Repository Collections Specialist at the Virginia Tech Libraries. She assists users from Virginia Tech (and around the world) to deposit and obtain access to scholarly and cultural heritage resources in VTechWorks, Virginia Tech's institutional repository. She received her BA in Psychology from the University of Maryland, Baltimore County and her MLS from the University of Maryland, College Park. Before arriving at Virginia Tech, Melissa worked as a student assistant in the University of Maryland's Special Collections Department and as a Metadata Specialist at the National Agricultural Library.