Vireo ETD system deployment experiences

Peter J. Nürnberg¹, John J. Leggett², S. Mark McFarland³
¹Texas Digital Library, ²Texas A&M University Libraries, ³University of Texas Libraries
¹peter.nuernberg@tdl.org, ²leggett@library.tamu.edu, ³m.mcfarland@austin.utexas.edu

The Vireo electronic thesis and dissertation (ETD) submittal and management system has been in use at several universities throughout the United States. Vireo was developed over the last several years at the Texas Digital Library (TDL,) a consortium of public and private institutions throughout the state of Texas. The TDL has recently undertaken a Vireo productization effort. The result of this effort is both an updated software system and accompanying support in the form of documentation, training and support infrastructures. In this paper, we describe this effort. We first review the methodologies we used to undertake the effort itself. This review includes a discussion of the development and testing methodologies we employed for the Vireo system software, including those practices we found especially useful or problematic. We then describe the deployment of the software and support. Specifically, we describe the strategies we used to accommodate the roll-out of software to nearly 20 institutions nearly simultaneously, as well as how we managed the risks inherent in such a deployment. We also describe the development of training and support materials that allow us to scale these activities to our user community. Finally, we reflect on the unique challenges that the productization of an ETD system engenders relative to generic productization efforts. We conclude with recommendations for other organizations faced with development, deployment, maintenance and support of ETD systems.
Introduction

The Texas Digital Library (TDL) is a consortium of higher education institutions in Texas that provides shared services in support of research and teaching. The TDL began in 2005 as a partnership between four of the state’s largest ARL universities: Texas A&M University, Texas Tech University, the University of Houston, and the University of Texas at Austin. Currently, the consortium has 19 members, representing large and small institutions from every region of the state. The goal of the TDL is to use a shared-services model to provide cost-effective, collaborative solutions to the challenges of digital storage, publication, and preservation of research, scholarship, and teaching materials. Among the services the TDL provides its members are: hosted digital repositories; hosted scholarly publishing tools; development of a “Preservation Network” to secure multiple copies of digital items at geographically distributed nodes; training, technical support, and opportunities for professional interaction; and, electronic thesis and dissertation (ETD) management software and infrastructure (Vireo).

The Vireo project [Mikael et al. 2009] was started at Texas A&M University. TDL assumed responsibility for Vireo in 2006. It has been under active development at TDL since that time. Until 2009, most of this effort has been toward adding functionality. This development was done primarily with a small team of graduate student workers. Starting in October 2009, development was transferred to a larger team of professional software engineers. This larger team undertook a productization effort that spanned nearly 13 weeks. It is this productization effort that is the focus of this paper.

We begin by describing our development efforts, including our methodology, strategies, and results. We then consider the deployment efforts surrounding the efforts of our development. We conclude with lessons learned from our experiences.

Development

In this section, we describe the development methodology we used in the latest round of Vireo development. We begin by describing the state of Vireo before we began the latest round. We continue by describing the scrum process [Schwaber and Beedle 2001] of development and our local adaptations of it. We also describe the process of refactoring [Fowler 1999] code, both generally and as applied to Vireo, as well as the documentation we produced as part of the productization effort. We conclude by describing our testing procedures.

Vireo prototype

When we started our latest round of Vireo development, the code suffered from various problems. Although basically functional, there were numerous known defects. The code had grown more through accretion than careful planning, with new features and defect fixes applied with an emphasis on quick turnaround instead of maintainability. Over time, this focus resulted in a code base that was unnecessarily complex and brittle. New changes were difficult and time-consuming to make, and often resulted in new defects being introduced. Although turnaround time for defect fixes was still acceptable, the defect rate in new code had become problematic.

As a result of these issues, we decided to change our approach to Vireo development. We instituted a feature-freeze, concentrating only on fixing known defects. We moved development from one Vireo specialist to a team of developers (initially with three members, with a fourth joining the team about halfway through the development.) We also adopted a different development methodology with a focus on short bursts of development punctuated with ample
opportunities for feedback and course correction instead of the relatively long periods of development favored previously. Finally, we added a formal build engineering process as well as multiple layers of testing, both of which were previously missing.

Scrum

We adopted scrum as our development methodology for our Vireo work. Scrum is an agile software development methodology. Agile methodologies are so-called because, unlike traditional software engineering methodologies, agile methodologies are constructed to expect change in requirements. They provide tools for customers to change their preferences or requirements for a project, as well as for developers to cope with this change.

An important concept for understanding scrum is empirical process control [Ogunnaike and W. H. Ray 1992] – the idea that, for complex processes (such as software development), an iterative approach based on feedback and correction is required. The alternative, defined process control, is suitable for very well-understood processes. Traditional software development methodologies such as the waterfall model are more deeply influenced by defined process control than agile methodologies.

A second important concept in scrum is the role of product owner. The product owner in scrum has been described as the “single chopable neck” – the person responsible for defining and prioritizing the work for the project. Ideally, the product owner is a fully-empowered customer representative from outside the development organization. For various reasons, we did not have such a person for our recent Vireo work. Instead, we chose a TDL employee who is not a member of the development team and who frequently interacts with customers. The impact of our choice on our work is described below.

At the TDL, the work that needs to be done is given to the team by the product owner in the form of user stories. A user story [Cohn 2004] is a very lightweight form of a requirement. It informally communicates a business need to the development team. As Cohn says, a user story can be thought of as a “reminder for a discussion” between the product owner and the team. We record each user story on a 4”x6” white notecard. Generally, each of our user stories has the form: “As a <user role>, I want to <action>.” On the back of each notecard, we record acceptance criteria given to the team by the product owner. The acceptance criteria define when the work related to the story is done. Collectively, all user stories not yet completed are referred to as the product backlog. For our most recent Vireo work, most of the stories concerned fixing known defects – the remainder were requests for new functionality.

The work of a scrum team is divided into sprints or development iterations. For the most recent Vireo work, we used sprints that were three weeks long. Within a sprint, there are daily stand-up meetings called scrums. Each sprint starts with a planning meeting and concludes with a review and retrospective. In the middle of the sprint, there is a mid-course correction. We also use the middle Friday of a sprint as a “lab day” on which non-development duties of the team are done. See Fig. 1 for a graphical depiction of how we organized our sprints.

At the planning meeting, the product owner prioritizes the most important stories in the product backlog. The team estimates the complexity of each of these high priority stories in terms of story points. Story points are an arbitrary measure of complexity. A story point does not necessarily correspond to a particular length of time. It is important that measures are consistent (i.e., all 1 point stories are approximately equally complex; all 2 point stories are approximately twice as complex as any 1 point story; etc.) It is also important that the team know their velocity, or the number of points that can be expected to be completed within a sprint. With the velocity and the complexity estimates for the high priority stories, the product owner
can choose which stories are most important and fit within the velocity. The team then either commits to completing the choice of stories in the upcoming sprint, or further negotiation occurs (e.g., some stories are re-estimated) until the team can commit to the work chosen by the product owner.

The idea behind this negotiation and commitment process is as follows. Broadly speaking, there are three variables within development: time, scope and quality. In a process in which the customer dictates both time and scope (i.e., “do this much work in this much time”), the free variable is quality. A team may deliver the requested work within the requested time, but the quality of the delivered work may suffer if the demands of the customer were unrealistic. In scrum, we hold quality constant (we always want high quality) and dictate time (in this case, we defined sprints of three weeks). The team can then vary the scope by negotiating the work that will be completed.

Figure 1. A typical sprint at the TDL.

After the stories are chosen, the product owner can leave. The team continues with deaggregation, breaking the user stories into a set of concrete tasks. Each task should be between 1 and 8 hours – smaller tasks need not be tracked, while larger tasks should be further deaggregated. We generated a 3”x5” colored notecard for each task.

On each development day, the team meets for a scrum. At the scrum, every team member reports on what they did (which tasks they completed) since the last scrum, what (which tasks) they are planning to do until the next scrum, and what impediments they have to their work process. Team members do no significant work that does not correspond to a task card. If new work arises, new task cards are generated.

The team used a large (4’x8’) cork-board to track the state of each story and task. Stories and tasks could be in one of several columns: unstarted; specify; execute; test; confirm; or, complete. All stories and tasks start in the “unstarted” column. When a task is first taken by a team member, it is moved into the “specify” column. Once the task is well-defined, it is moved into the “execute” column. After any necessary development related to a task is complete, it is moved into the “confirm” column. Once in the confirm column, a team member who has not previously worked on the task checks the work done so far to confirm that it has been done correctly. After this double-check is complete, the task is moved into the “complete” column.
Story cards also move across the columns of the board. They are promoted from one column to the next once every task associated with the story has been moved at least that far across the board. (For example, a story card should appear in the “execute” column only after all tasks associated with it are in the “execute,” “confirm” or “complete” columns.)

At the mid-course correction, the team considers whether or not they are on schedule for meeting their commitment. If they believe they will not meet their commitment, they can schedule a meeting with the product owner and re-prioritize (if necessary) the remaining work in light of the reduced velocity of the team. In our case of the most recent round of Vireo development, none of the mid-course corrections resulted in re-scoping the work for a sprint.

Developers often have other duties, such as attending departmental meetings, taking professional development courses, upgrading software, or just digging out from under accumulated email. We set aside the middle Friday of every sprint as a “lab day” for developers to attend to these other duties.

At the sprint review meeting, the stories chosen for the sprint are demonstrated to the product owner. These demonstrations are public, though in our case, only the final demo was attended by anyone other than the team and the product owner. The product owner decides if the acceptance criteria for each story were met. If so, the story is removed from the backlog. Otherwise, the story remains on the backlog available for re-prioritization in future sprints. There is no notion of a story being “partially complete” – either the product owner agrees the criteria for a story were met or not.

Finally, the team (without the product owner) holds a retrospective on the sprint. At this meeting, the team reflects on what went well during the previous sprint and what new things they would like to try. These “new things” might be in response to perceived weaknesses of the recently concluded sprint, or might be small adjustments to the work process. For example, after the first vireo sprint, our team decided that members should bring task cards with them to the scrum every day to ensure that no unnecessary work was being done. This was in response to an observation by the team that occasionally, some members engaged in “gold-plating” – the practice of adding more functionality than was asked for by the product owner. Such gold-plating work, since it was not asked for, tended not to have corresponding task cards. They also made a relatively small change by moving the time of the daily scrum from 10am to 9:30am.

**Refactoring**

Parts of the Vireo code base prior to our most recent work were deemed sufficiently complex that refactoring (simplifying) the code was necessary before any major changes could be undertaken. Refactoring is the process of improving code in a systematic way. Refactoring should be semantically neutral – i.e., the resultant code should not have any new behaviors or fix any existing defects. Instead, the improvements in refactored code generally concern simplicity. By making code simpler without affecting the behavior of the code, it becomes simpler to add new functionality or address existing defects. One can think of refactoring as “cleaning up” code.

As our team began the refactoring process, we faced the additional challenge that the existing code did not have accompanying unit tests (see below). Unit tests are a practical prerequisite for refactoring, since they allow an objective definition of semantics. In the absence of unit tests, it is difficult to guarantee that any refactoring undertaken by the team was semantically neutral. Instead, we made a best effort analysis of the current behavior of the complex portions of the code and applied a series of small, well-defined refactoring techniques. Strictly speaking, this may be more correctly referred to as “restructuring” rather than refactoring, since there was no objective measurement of semantic drift. In principle, correctly applied refactoring should not
introduce new defects. We found, however, that our team did introduce some changes in behavior that, lacking a clear specification, could be classified as defective. These have since been addressed in a follow-on maintenance release.

**Documentation**

The TDL made major strides in documenting Vireo during the latest development cycle. There are four basic types of Vireo documentation: inline code comments; a developer wiki; a user wiki; and, training videos.

Firstly, the team did a better job of documenting the code itself. This documentation allows developers better insight into how the code should function. When TDL developers go back through the code to address defects, this code can be helpful in reconstructing the thought processes of team members during previous work. It is also helpful for developers outside the TDL. (Currently, several universities outside of the TDL have signed agreements to beta test Vireo. In September 2010, Vireo will be open sourced. Therefore, the audience of developers external to TDL is already significant and set to grow.) Secondly, the team generated wiki documentation aimed at developers and administrators. This documentation covers such topics as high-level architecture, configuration options, and interface specifications. Thirdly, there is a publicly available wiki aimed at end users (specifically, librarians and graduate school representatives) as well. Finally, there are numerous videos posted on YouTube that demonstrate Vireo use.

**Testing**

The TDL team focused heavily on testing during the latest Vireo development. There are several types of testing (unit, integration, and QA) that were put into place.

Unit tests are software that test compliance of a single piece of the system with a specification for that piece. Especially in refactored code, the team followed an interface-driven design approach that consists of three steps. Firstly, a contract for an object is specified by a Java interface. Secondly, a unit test for compliance to this contract is built. Finally, an implementation of the contract is written and then continually refined until it passes the unit test for the contract. These tests are usually small and quick to execute. We configured the continuous integration server we use, Hudson (see <http://hudson-ci.org/>), to execute these tests every time a developer checks in code to our version controlled source code repository. If any unit tests fail, the entire team is notified via email. This allows defects to be detected early and addressed promptly.

Integration tests test several parts of the system under a use scenario. In our case, we generated an integration test for every defect to ensure that we successfully addressed these defects during our development. These tests were semi-automated. We used a tool called Selenium (see <http://seleniumhq.org/>) to help execute scripts of actions within a web browser. Developers could run these scripts periodically. Along with each integration test script, the team documented the expected outcome of running the script.

The team also generated a quality assurance (QA) test plan. This QA test plan also relied heavily on Selenium scripts. The aim of the QA tests, however, were expected paths through the system (e.g., there were tests for ensuring that: an administrator could log in; options on a particular screen could be set or unset; and, items could be exported.) There were 31 QA tests in total. These tests were run near the end of every sprint.
Deployment
In this section, we describe the deployment of the result of the latest round of Vireo development. We describe the scope of the deployment, the architecture used in the deployment, and our training and support efforts.

**Scope**
We manage two types of deployed Vireo instances. “Labs” instances are intended for users to familiarize themselves with the system. These instances allow staff members to submit sample ETDs and test various execution paths through the software without affecting students or interfering with colleagues. “Production” instances are those that are set up for “real” use by students and staff.

TDL, as mentioned above, currently has 19 members. We decided to deploy a labs instance and a production instance of Vireo for each of these members, as well as two labs instances and one production for our own internal use, for a total of 41 instances. Some of our members have decided to run their own production instances. TDL deployed a production instance for these members as a backup if needed.

There are many difficulties inherent in managing 41 instances of any piece of software. Many of these difficulties stem from the fact that there can be differences in the instances. For example, different instances require different configurations – e.g., students at different institutions should be presented different lists of available degrees. Also, the use load for different instances can vary widely – some labs instances are rarely if ever used, whereas some production instances manage many hundreds of submissions each semester. Finally, prior to our most recent development round, different instances were running different versions of Vireo, since some institutions requested specific customizations that would not be appropriate for other institutions.

We simplified our deployment procedures considerably to help address some of these difficulties. Most configuration options are stored within the Vireo database; therefore, upgrading the Vireo software does not affect these options. Options that can vary between instances (e.g., the destination for submissions that are published) are stored in our version control system. We have consolidated the instances onto a single database server with substantial computing resources. Lastly, we now insist that all TDL hosted instances of Vireo be upgraded nearly simultaneously. (We say “nearly,” since there can be small variations. We generally upgrade all labs instances first, wait a short period of time to allow user testing, and then negotiate specific downtimes with partners for production instances within a relatively short time box – say, one week. Thus, production instance upgrade times vary by at most one week, which vary from lab upgrade times by at most one more week.)

**Architecture**
We have two (essentially) identical servers that run our Vireo instances: one for production instances; the other for labs instances. Each is a Sun T5220 with 1 physical and 64 virtual CPUs and 32 GB RAM running Solaris 10. Each of these two machines is backed by its own database server. Each database server is a Sun V490 with 4 physical CPUs and 16 GB or RAM running Solaris 10. The database servers use two filers, containing a total of 112 disks comprising 4 aggregates with a total of c. 25 TB capacity. Fig. 2 illustrates this architecture.

For every member, TDL hosts a production and labs copy of both a Vireo and an IR. We use SWORD [Allinson et al. 2008] to move submissions from Vireo to the corresponding IR. We
then harvest from these IRs into our federated repository (also hosted on the production server mentioned above.) Currently, the Vireo and IR instances we host run a special TDL modified version of DSpace 1.5.1. Our database servers run Postgres version 8.4.

Training
Part of our deployment efforts is the ongoing support for Vireo training. The TDL holds a number of classes at minimal cost to our members. These classes are run at a training facility hosted by one of the TDL members. The Vireo classes run several times per year, and have an average of 5-10 attendees. Some of these classes are taught by TDL staff; others are taught by staff from TDL members who have attended previous Vireo classes. One of the TDL labs instances of Vireo is used to support these classes. This Vireo installation is primed with specific submissions at the start of each class, and restored to its original state after the class completes. The classes are intended to cover student, graduate school and library workflow as well as configuration and administration. Informal feedback from class attendees have indicated that the Vireo classes are helpful to attendees, but have also pointed out areas for improvement.

Support
Our current support model for TDL applications is based on a three tier approach. The initial point of contact for users is the help desk. A TDL member institution staffs this help desk with a full time employee. This employee attempts to resolve straightforward issues – we refer to this

Figure 2. Deployment architecture of Vireo at the TDL.
as tier 1. Any more complex issues are escalated to tier 2 by our help desk employee. Tier 2 issues are discussed within the TDL production team, and delegated as appropriate. The production team consists of one full time employee staffed by a TDL member, one TDL developer (who allocates up to half of his time to production issues) and several other employees who provide guidance on system administration and customer communication issues. Items deemed too complex for the production team are referred to the Chief Technical Officer, who either retires the issue (if it cannot be addressed by TDL) or escalates the item to tier 3, in which case it is allocated as a task to the development team. (In scrum parlance, it becomes the basis for a new user story.) Since the development team rotates among several different projects, support issues at tier 3 may not be resolved in a timely manner.

Lessons
In this section, we first consider some of the challenges we faced during our Vireo productization efforts. We then provide some recommendations, both for development and deployment.

Challenges
We faced several challenges in our recent productization effort. Some of these are common in any projects; others are common in general software projects; and, still others seem particularly magnified by the nature of development of an ETD system.

The TDL underwent substantial organizational change during the latest Vireo development. Such change always introduces unexpected side effects and negative productivity consequences. In our case, most of the Vireo development team consisted of newly hired TDL employees. The latest Vireo development was an early (if not the first) experience the team members had working together. We also reorganized the TDL “horizontally” instead of “vertically” – i.e., we created one large team and cycled the team through each outstanding project instead of putting single team members in charge of each project. This meant halting other projects while the Vireo productization was ongoing. We believe a horizontal organization fosters teamwork and results in a higher overall throughput, though at any given moment, work is only being done on one of many outstanding projects.

The change to scrum-style development was significant for us, since most of our staff were unfamiliar with scrum. This impacted all facets of our organization. Our system administration staff was responsible for maintaining more development tools, such as our continuous integration server. We standardized the computing environment for our developers (Java in Eclipse on Ubuntu Linux) to facilitate pairing of developers. Our communications team worked hard to explain the scrum model to our partners, setting their expectation regarding when work would be done, eliciting requirements through user stories, and organizing public sprint reviews (including webcasting the final Vireo sprint demonstration.)

ETD systems such as Vireo have multiple audiences. TDL is funded partly through contributions from the libraries at our partner organizations, so we view librarians (especially catalogers) as a very important audience for Vireo. Representatives from graduate schools perhaps have the greatest amount of interaction with Vireo, and are thus an important audience. Finally, students submit their work through Vireo, and are therefore a third important audience. In the best of circumstances, managing the needs and priorities of disparate audiences is problematic. We feel this is especially so for Vireo, since we did not have an explicit representative of the student audience in our recent development work. We tended to compensate for this by prioritizing defects that affected students highly. Librarians and graduate school
representatives also have well-aligned but differing goals for Vireo. We have found that many librarians view Vireo as a valuable tool for metadata collection, whereas many graduate school representatives view Vireo as a workflow management tool for ETD related processes. For our recent work, we did not feel compelled to prioritize one of these views more highly than the other. Going forward, we will need to consider this tension to prioritize requests for new features.

**Recommendations**

Firstly, we recommend scrum as a development framework. Scrums worked well for us. We found that it provided many benefits, including a strong customer-focus, quick turnaround for development, and ample, formalized opportunities for improvement. We also greatly benefited from making the demonstrations public. This allowed our customers to see the status of our work and provided the development team valuable feedback. Secondly, we recommend not only making the support ticket system public, but actively training users in its use. Our support system uses a ticketing model. These tickets are publicly viewable; however, we had not yet communicated with our customers how or when the system should be used. As a result, most customers do not know the status of tickets they submit without explicit feedback from our helpdesk. We believe that a public and often-used ticket system helps customers, the production and support team, and developers.

Thirdly, we recommend setting up a semi-formal mechanism such as a user group for collecting such feedback as early in the productization (or even development) effort as possible. We have recently started the Vireo Users Group. One of the functions of this group is to act as a virtual meeting place for Vireo users to discuss their experiences and share best practices. We hope to be able to use the results of some of these discussions to inform our prioritization of development and maintenance work. Since the Vireo Users Group was not yet active when the work described here began, we were forced to rely on our own perceptions of the relative priorities of the outstanding defects instead of getting these priorities directly from our users.

In general, all of these recommendations can be seen as results of promulgating the organizational value of transparency. Scrums make the development process transparent. A public ticket system makes the support process transparent. A users group makes user story priorities transparent. We have found that pursuit of transparency has greatly helped our productization effort.

**References**


