DOI:10.1145/2755503

Article development led by acmqueue
queue.acm.org

An agile process implementation.

BY PHELIM DOWLING AND KEVIN MCGRATH

Using Free and Open Source Tools to Manage Software Quality

THE PRINCIPLES OF agile software development place more emphasis on individuals and interactions than on processes and tools. They steer us away from heavy documentation requirements and guide us along a path of reacting efficiently to change rather than sticking rigidly to a predefined plan. To support this flexible method of operation, it is important to have suitable applications to manage the team's activities. It is also essential to implement effective frameworks to ensure quality is being built into the product early and at all levels. With these concerns in mind and coming from a budget-conscious perspective, this article will explore the free and open source applications and tools used by one organization in its quest to build process and quality around its projects and products.

How a software development team implements process and handles its application life cycle management can have a huge bearing on the success of the product it is developing.[2] A lean and efficient mode of operation is the key, especially for early-stage start-ups. Many commercial-grade products and frameworks are available for integrating all aspects of your development process: requirements management, project planning, defect management, test case creation and execution, automated testing, load testing, and so on. But these tools can be expensive and may be too much of a burden for some organizations' idea of agility.

Other solutions exist that can support your agile philosophy and not make any dent at all in your hard-earned bottom line. An abundance of free and open source tools have been available for quite a while now. Many companies and organizations are employing open source tools to meet their business needs.[1] These applications often have large communities of active and engaged contributors documenting the features of the product and releasing enhancements on a regular basis. We recommend proper due diligence in examining the suitability of the tools for your needs and determining their track records of performance, maintenance, and growth. If they pass that check, open source tools can more than meet the needs of small project outfits, funded research and development teams, early start-ups, and even larger-scale enterprises.

Project Management and Issue Tracking

The Telecommunications Software & Systems Group (TSSG)[5] previously used tools such as XPlanner for project planning and Bugzilla for defect management. While individually these tools were quite functional, the lack of integration in terms of gathering requirements, planning, logging defects, and so on became an obstacle to our desire for a more streamlined and agile way of working. Expecting developers to look at one system for bugs to be fixed and another to determine what features to work on was inefficient. The Verification and Validation (VnV) team made efforts to link defects with features but this was manual and laborious. Planning of daily and weekly activities was harder because of the standalone nature of the tools.

This difficulty led to the creation of a subteam of tech leads and process champions to review available tools that might be better suited to our needs. Their remit was to find a tool that would allow us to manage our activities across multiple projects efficiently, was adaptable to our fluid agile ethos, and had zero cost.

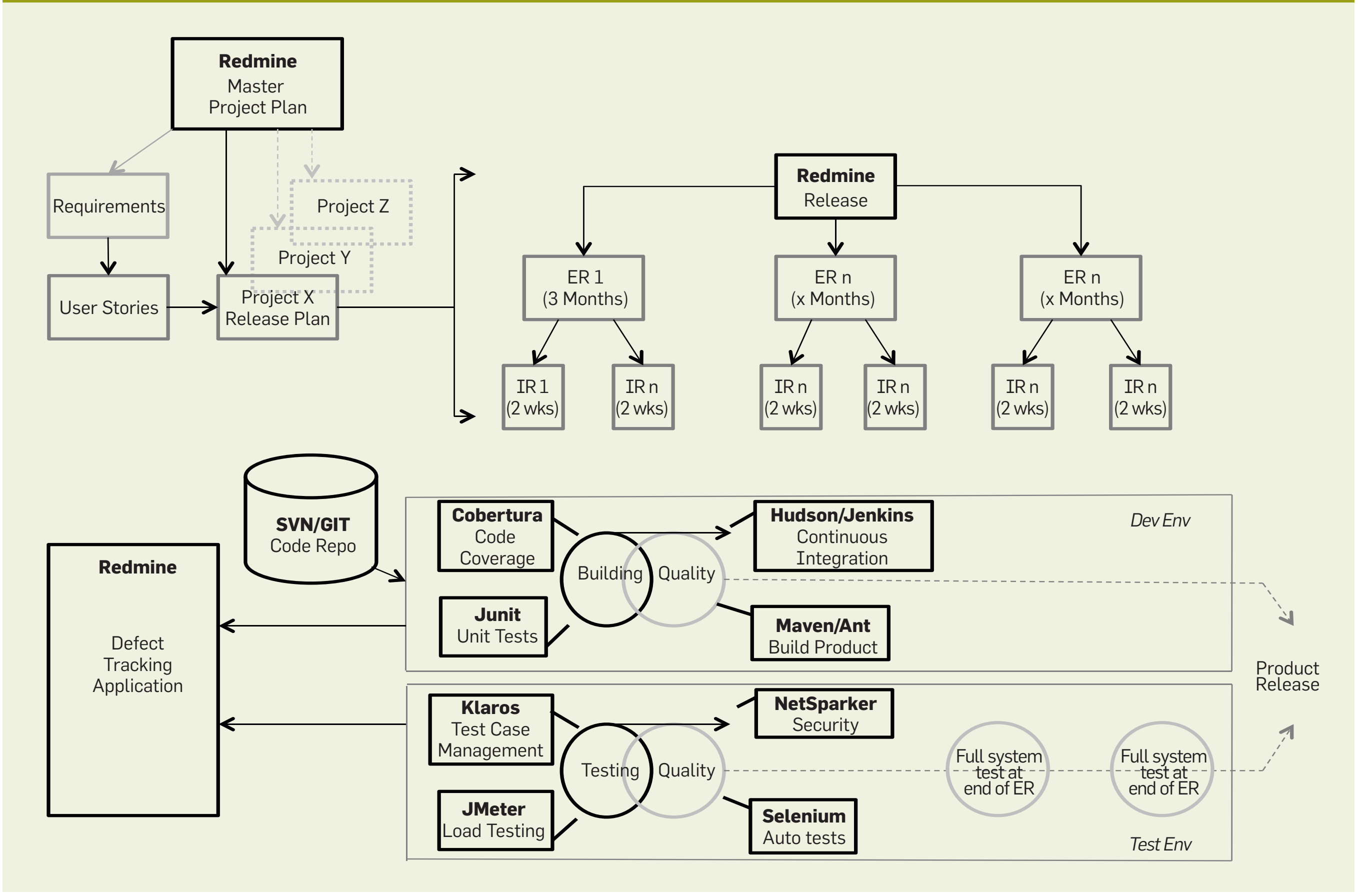
This eventually led to the rollout of Redmine as the organization's primary tool for project management and issue tracking. Redmine is an open source application and is released under the GNU General Public License v2. Written using the Ruby on Rails Web application framework, we have installed our implementation on an Ubuntu VM. Utilizing the open source Apache Web server, TSSG staff access the tool via any common browser. The application's MySQL database is backed up regularly because of the widespread use and critical stature of this tool within the organization. The integral nature of Redmine within the TSSG agile process along with the other tools we use is depicted in the accompanying figure.

At the requirements gathering stage all project requirements are documented in Redmine as user stories. These are initially stored in the project backlog before being allocated to specific engineering releases and biweekly iterations. The user stories are then broken down into subtasks and subfeatures so that distinct and measurable elements of work are planned and executed to achieve the overall goal. These pieces of work are allocated to individuals on the team to be completed within a specific time period.

The Redmine tool facilitates many of the other agile methods used by our engineering teams, including time management, defect tracking, email integration, task boards, sprint cards, and so on.

While there is an active community of developers working on Redmine and creating plugins of value, this is somewhat of a double-edged sword. We have found in the past that upgrading the core product can be problematic due to these plugins. The plugins are not always compatible with newer releases, leading to headaches during the up-

Open source and free tools mapped to the TSSG agile process.



grade and broken functionality for features that teams were previously using.

Code Repository and Continuous Integration

One of the first tasks for a project team is to set up a source code control framework. It is critical to maintain a central area for the team to check in code and have it backed up regularly. Tracking versions of your code will allow you to roll back to a previous revision if required. Spawning branches as the project grows and tagging builds to allow hot fixes are likely requirements. In TSSG, we have used locally hosted versions of both the Apache Subversion system and the Git repository, and both tools have met the needs of our projects.

Another handy integration point we have set up for projects is between Git and Redmine. This allows us to view the Git repository, commits, and stats directly from our project management system.

A concept we are currently exploring is a peer review of code before commit. This would especially apply to junior developers paired with more experienced colleagues. Tools such as Gerrit and Review Board are being examined for suitability in this regard. While it may bring extra overhead before commit we feel it can add extra quality to our products in the long run. Code reviews and peer programming are philosophies that TSSG promotes, but are too often overlooked due to time constraints. If these tools can facilitate code review and peer programming in a relatively seamless manner, we anticipate trialing their use on a subset of projects before pushing them out for general use.

A key element of our agile process is to build our code regularly. Continuous integration will execute all the unit and acceptance tests at build time, and makes regular builds available for verification on the test environment. In the TSSG we have used Hudson and more frequently Jenkins for this. Easily installed and configured, Jenkins builds can be triggered by code check-ins (to the repositories mentioned earlier) or at regular intervals. A user-friendly Web-based interface makes it easy to view a list of builds including the latest. You can view the changes made in any build, and a comprehensive overview of the test results for each build is also presented. Jenkins can also be configured to send email messages to team members on build pass and/or fail. Of course, under-the-hood open source tools such as Ant and Maven are actually generating the builds.

In the TSSG we promote the concept of test-driven development and, while the choice of tool for unit testing will vary depending on the technologies being used, JUnit is the most common. On top of this, we measure the percentage of our code that is covered by tests using tools such as Cobertura. A Cobertura plugin for Jenkins allows the code coverage to be analyzed from the Jenkins build Web page. This element of our process often generates the most heated discussions between the VnV team and the engineering teams. Our goal is a 90% level of code coverage, but there is always debate about the difficulty of achieving this target with certain technologies. Also, the time and resources required to reach the last few are often questioned in terms of return on investment. However, the team mentality promoted in our agile philosophy means that consensus is always reached.

Another aspect of the TSSG's agile process methodology is an independent code review team. This team examines the project's implementation of a build framework using the tools described in this section with a view to building quality into our code from the ground up. They may also advise the developers to implement code analysis using open source tools such as Checkstyle, Gmetrics, or CodeNarc.

Automated Testing

The majority of the projects in the TSSG start as funded research. However, as the project evolves and the solution stabilizes and matures, the product will generally take on a more commercial nature. As this stage approaches we have found it beneficial to implement Selenium automated test suites on the components developed to date.

Our backend implementation is Selenium 2.0 WebDriver running on a Windows server. The Selenium IDE is installed as a Firefox browser add-on for recording the test cases. To allow the results of a test suite to be emailed to the relevant team members we have installed two other pieces of freeware, MDaemon Free mail server and SendEmail, a lightweight command line email client.

The tests are executed in Firefox and recorded via the Selenium IDE. The tests are then saved on the server side. A single file to list all the individual test cases to be executed as part of a test suite is created. A batch file is developed to execute the entire suite and scheduled to run at desired intervals. The batch file calls the Selenium tool and tells it which suite to run (and on which browser) and where to save the results. The final step is to email the results to the distribution list.

Quite often you will find that other users have encountered roadblocks in automating elements of their tests. To overcome this they may have coded a 'user extension' for Selenium and made it openly available. Contained in a flat file configured in your local environment, these coded solutions provided by individual contributors can prove to be very useful. One of the earliest extensions we used was the beatnicClick function. This handy extension acted as a workaround for an issue with timeout failures on certain native click activities, allowing the test to execute the desired functionality seamlessly. Another example of a user extension we have implemented in our framework is for flow control logic. The ability to easily include `goto` statements and `while` loops has helped improve the robustness and coverage provided by our automated test suites.

The Selenium framework implemented in TSSG allows us to execute functions in a browser session, record those steps, and then replay them automatically. The test steps can then be extended to include checks for specific text or elements on the site. You can capture and store variables for reuse within the test suite. Pauses and waits can be added to replicate real user behavior. Screenshots can be captured. Data-driven testing is supported; that is, external files with lists of parameters can be fed into the tests. Other more advanced features include the use of JavaScript snippets, for example to manipulate data or strings.

Although the recording of the tests must be carried out in Firefox, the test execution supported by the Selenium WebDriver allows us to run the tests against Firefox, IE, Chrome, Safari, and other common browsers.

On certain projects we encountered issues with Selenium not being able to interact with Windows-specific pop-up dialogues. While Selenium provides a feature to handle browser pop-ups (via getWindowHandles and switchTo.window), the framework cannot handle non-browser components. To deal with this we used another open source automation tool, iMacros. Using iMacros to handle the interaction with the external dialogue, the iMacro script was saved as a bookmark in the browser and called from Selenium, allowing seamless automation of the full test.

Selenium works well for our VnV test team, who utilize its record/playback authoring of tests with some extensions as described previously. But it can also be a powerful tool for developers who need to create tests in Java, Ruby, Python, and other languages using the Selenium API.

The most significant implementation of Selenium automated tests by TSSG was on the FeedHenry project. Working within an aggressive biweekly release cycle,[3] the comprehensive automated test suite maintained by the VnV team gave the fledgling company the confidence and space to release features for its enterprise mobile application platform every two weeks. RedHat's €63.5 million acquisition[4] of FeedHenry (www.feedhenry.com), which started as a small research project, is a vindication of both the TSSG's research and innovation model and the successful implementation of our agile process and tools.

A key element of our agile process is to build our code regularly. Continuous integration will execute all the unit and acceptance tests at build time, and makes regular builds available for verification on the test environment.

Load Testing

Performance monitoring is an important part of the software life cycle at TSSG as many of our products will go to live trial and full commercialization. We conduct load testing in two ways. Endurance testing involves evaluating a system's ability to handle a moderate workload for a longer period of time. Volume testing, on the other hand, subjects a system to a heavy load for a limited period of time. Both approaches have identified bottlenecks, bugs, and component limitations in certain projects.

Apache JMeter is an open source load-testing tool designed to load test functional behavior and measure performance. It was originally designed for testing Web applications but has since expanded to other tests. JMeter provides the control needed on the tests and uses multiple threads to emulate multiple users. While JMeter is not a browser, and does not function like one, it can emulate browser behavior in a number of ways—caching, cookie management, and header management.

JMeter includes an HTTP proxy which, when enabled, can be used to record scripts when testing websites. The idea is to point the browser to the JMeter proxy so that JMeter can sniff the requests made through the browser and store them under the specified controller in JMeter. The information derived can be used to create a test plan. JMeter will then play back the HTTP requests to the server, but we can simulate any number of users doing this. We also use Fiddler and Firebug to dig deeper into Web traffic when required.

Unfortunately, JMeter does not execute the JavaScript found in HTML pages. Nor does it render the HTML pages as a browser does (it is possible to view the response as HTML). So, depending on the Web application, it may be necessary to incorporate Selenium tests in a separate test run if you are testing the client-side business logic and use JMeter for the backend components. JMeter also has an extensive set of third-party plugins that extend its functionality with an array of graphs, new load delivery controllers, and other functions that act as add-ons to the original JMeter package.

Recently we have had a requirement to perform load tests across an Openfire IM server instance to test rules from a policy engine we are creating as part of a new project. In this scenario we used Mockaroo to generate a large number of test users with realistic characteristics. We then used Tsung test scripts to fire load at the test environment. Tsung's Erlang foundation means it uses fewer threads than JMeter, allowing larger-scale testing, but it is trumped by JMeter's superior GUI, which allows easy configuration of tests and analysis of results.

Test Case Management

The TSSG has chosen the Klaros test management system for creating and executing manual test cases. A community edition that is free for unlimited use provides all the functionality required to maintain manual test cases across multiple projects.

Windows and Linux versions are available, and we have installed our instance on a Windows server. The downloadable executable will install an Apache Tomcat application server and a file-based database Apache Derby (although this can be changed to more robust solutions such as MySQL or PostgreSql). Users can interact with the test case system via any common Web browser.

The free edition of Klaros supports multiple projects being maintained simultaneously. Within these projects the test case manager can build up comprehensive test steps, test cases, and test suites. Varying test environments can be maintained for each project, for example, for different OSes or application servers. Also, versioning of the product being tested is supported via the concept of tracking the system under test (SUT).

At execution time, the tester can decide to complete a single test or a full test suite. The test run can be suspended and resumed as needed. The result of each test (pass or fail) and any errors observed are captured at every step. After the test suite has been completed, an easy-to-read report is generated in various formats (pdf, html, csv). This report will display the date, test environment, system under test, a summary table of the test case results, a pie chart of the results, and a more detailed listing of the individual test cases with details such as execution times.

Also available in the community edition is integration with other open source testing tools. Redmine, described earlier in this article, can be configured as an issue management tool so that tickets can be created directly in the chosen bug tracking and project management tool. The continuous integration servers Hudson and Jenkins (described earlier) can also be integrated with Klaros. By installing the Klaros plugin in Hudson or Jenkins, the results of test cases generated at build time can be imported into Klaros via a REST interface. The import is tagged with the appropriate project, test environment, and SUT IDs and the results can then be reported and analyzed as required. Similarly, test results from test automation tools such as Selenium can also be imported and added to the manual test results.

Challenges

Supporting mobile usage of the apps and features produced by our projects is an essential element of our quality offering. The matrix of test scenarios can grow quite large when you want to cover phones and tablets across iOS, Android, and other platforms. Add in the spectrum of manufacturers with their own variations, and comprehensively validating app quality can be a challenge.

On certain projects we have dabbled in the pool of commercial offerings that provide a wide range of devices along with automated test functionality, Device Anywhere and Perfecto Mobile being the most popular. However, long-term use of these products was not financially sustainable on projects with tight budgets. We have made some attempts to automate and script tests against Android devices using ADB and monkeyrunner. MonkeyTalk also looks like a reasonable prospect for both iOS and Android. However, the resources required to build and maintain automated mobile test suites have not always proved justifiable. Mobile projects with agile requirements and aggressive deadlines have not yet proved suitable for large-scale automated testing, while some lack of confidence in emulator testing was also a concern. On the flip side, the advent of wearable technology such as glasses and watches broadens the horizon even further and may warrant a more strategic approach.

Another aspect we have identified for improvement in our process is security testing. Many of our projects are in the areas of e-health, finance, and personal digital data and have obvious requirements in terms of protecting user data. Protecting our own intellectual property and hosted services are also ongoing concerns. We have used Netsparkers community edition along with other free tools on certain projects to try to identify vulnerabilities to cross-site scripting, injection attacks, and so on. The code review team, mentioned previously, examines projects' use of static analysis tools such as Checkstyle and FindBugs to find flaws at that level. However, we feel leveraging existing TSSG expertise in this area can help provide a framework and process for a more bulletproof offering to the overall VnV process.

Conclusion

The TSSG operates without baseline funding and must compete nationally and internationally to finance projects. With a commitment to product excellence epitomized by the VnV teams' role on all projects, the use of free and open source tools to build layers of quality into our applications is a necessity. The many integration points between the tools described are an added benefit in pursuing a zero-cost solution to application life cycle management. The tools described in this article are only some of those used within the TSSG process. Monitoring the evolution of existing and new zero-cost tools is an ongoing task in our freeware and open source strategy. C

Related articles on queue.acm.org

Adopting DevOps Practices in Quality Assurance
James Roche
http://queue.acm.org/detail.cfm?id=2540984

Building Collaboration into IDEs
Li-Te Cheng, et al.
http://queue.acm.org/detail.cfm?id=966803

Breaking The Major Release Habit
Damon Poole
http://queue.acm.org/detail.cfm?id=1165768

References

1. Baldwin H. Reasons companies say yes to open source. (Jan. 6, 2014); http://bit.ly/1uJ7uYK
2. Clarke, P. and O'Connor, R. The influence of SPI on business success in software SMEs: An empirical study. *J. Systems and Software 85*, 10 (2012), 2356–2367.
3. Dowling, P. Successfully transitioning a research project to a commercial spin-out using an Agile software process. In *J. Software: Evolution and Process 26* (May 2014), 468–475; doi: 10.1002/smr.1611
4. Kepes, B. Red Hat acquires FeedHenry to move into the mobile space. *Forbes* (Sept. 18, 2014); http://onforb.es/1CXd4WF
5. TSSG; www.tssg.org/.

Phelim Dowling has 18 years of experience in the IT industry. He joined the TSSG Verification and Validation team in 2006 and has facilitated on projects such as Muzu, FeedHenry, Kodacall, and more recently the EU FP7-funded OpenLab initiative.

Kevin McGrath joined the commercialization team within Waterford Institute of Technology's TSSG in 2006 as a verification and validation engineer. He has worked on FeedHenry SWAGGER, COIN, SSGS, EGP, and Billing4Rent, an EU-funded project.