

2009

PACIFIC NW SOFTWARE
QUALITY
CONFERENCE



MOVING
QUALITY
FORWARD

OCTOBER 27-28, 2009

Conference Paper Excerpt

From the

CONFERENCE
PROCEEDINGS

Permission to copy, without fee, all or part of this material, except copyrighted material as noted, is granted provided that the copies are not made or distributed for commercial use.

ProdTest: A Production Test Framework for SAAS Deployment at Salesforce.com

Bhavana Rehani
brehani@salesforce.com

Kei Tang
kei.tang@salesforce.com

Abstract

Salesforce.com is the market and technology leader in Software as a Service (SaaS) [1] and Platform as a Service (PaaS) [2]. SAAS can be defined as a model of software deployment where an application is delivered to customers via the Internet. The company's portfolio of Salesforce.com CRM applications has revolutionized the ways that customers manage and share business information over the Internet, while the company's Force.com PaaS enables customers, developers and partners to build powerful on-demand applications [3] for any business process. The SaaS and PaaS on-demand models present unique challenges during new feature deployment. Unlike on premise software, new features are rolled out to numerous customers at the same time. Also, this rollout happens within a planned maintenance window. This means the R&D group has a very short period to ensure that the new release is not going to adversely impact customers' critical business operations. In order to verify that the release is good to go, QA engineers run a suite of sanity tests[4] during the maintenance window. Previously, testing used to be a time consuming manual process involving numerous QA engineers. To reduce this effort, the QA organization has developed a new automation framework (ProdTest) for executing tests on the production environment. ProdTest was developed as a testing framework at Salesforce.com to run automated tests on production environments during releases. This framework allows engineers to write both API and UI tests. Since the initial rollout, ProdTest has gradually evolved into a complex and sophisticated tool for writing automated tests. It has also become an integral part of our deployment process and has provided both measurable and intangible benefits, such as more predictable test runtime, fewer people involved in a release and an increased confidence in the release decision. This paper discusses the development of ProdTest including the technical, process and people challenges faced while building it. We also discuss the benefits to the organization and provide an insight into our future direction.

Biography

Bhavana Rehani is a senior quality assurance engineer at Salesforce.com. She joined Salesforce.com in 2006. She has learned a great deal by working with various teams on automation initiatives within the organization. She loves discovering innovative ways to automate test cases that are normally thought of as un-automatable. She especially enjoys utilizing JavaScript for testing complex web UIs. Bhavana earned her Masters degree in Software Engineering from Carnegie Mellon in 2006. She has also worked in the field of online learning at CDAC-Mumbai, a scientific society of the Department of Information Technology, Govt of India, from 2000 to 2004. Bhavana's interests include test automation, mentorship, educational technology and web programming.

Mr. Tang has over a decade experience in the software industry working in the aerospace, financial and CRM sectors. He has extensive experience in test harness design and implementation, web services and API testing, as well as developing and applying methods for model-based approaches to support defect removal and test automation. Mr. Tang is currently a software quality assurance manager at Salesforce.com, the leading provider of cloud computing services. He oversees the quality for the next generation UI development. He earned a B.Sc. in Aerospace engineering from University of Michigan at Ann Arbor, and a MS in Aeronautic and Astronautic Engineering from MIT.

1 Introduction

Salesforce.com is a leader in Software as a Service (SaaS) for business applications and is an innovator for Platform as a Service (PaaS). With on-demand models, Salesforce.com takes care of the deployment, delivery and upgrade of the applications so customers can focus on their own business execution. The unique challenges the Salesforce.com R&D group faces in supporting the on-demand model is very different than those for on-premise based software. One important difference is that the deployment of new functionalities takes place during a short maintenance window for numerous customers at the same time. Salesforce.com delivers a major release approximately three times a year, which gets rolled out to all customers. Due to high impact, it is crucial that we take maximum care while testing every new release. During the release, we must make decisions to move ahead or to rollback at every stage in a very short period of time. In order to ensure that the release is good to go to production, we carry out sanity tests on the new release before giving the final go ahead. Testing the release on the production environment is crucial since the production environment brings about additional complexity. For example, a production system of our size, that services over 150 million requests per day, has numerous points of failure in subsystems like the cache server, search engine, database server, app servers and various batch processes. Although Salesforce.com has an extensive automated suite of tests that runs on our internal test environments, these tests cannot uncover issues specific to our production environment. For example, a search related test that passes on the internal environments can fail on production if the production search process is switched off or incorrectly configured. During a release, the production environment undergoes numerous configuration, code and database changes. As a result, there are hundreds of tests that we need to run on production before signing off on the deployment of the release. Previously, these tests were run manually and required every QA engineer to participate. This was not scalable as the number of production instances and features were growing rapidly. Our solution was to create a production test automation tool (ProdTest) to eliminate the manual testing effort.

The following sections describe the evolution of ProdTest, the challenges faced in its development and technology used. We also describe the benefits achieved and our future direction.

2 Problem description

2.1 Previous process

During deployment of major releases, QA engineers need to verify the sanity of features they own. Previously, this involved a long and laborious process of manually verifying the sanity test cases on production during the maintenance window. The following summarizes some of the challenges faced during release time:

1) Meeting the time pressure of our maintenance window:

One of the biggest challenges of the manual testing approach was to keep within the time allocated for the maintenance window. With many people running tests at different phases of the release, it took too much time to find the issues, thus leaving very little time to get them resolved. With ever increasing features and complexity, completing everything during the maintenance window became risky.

2) Requiring numerous engineers on call:

At least one QA representative from each functional area was required to be on duty during the release. This meant at least 40 QA engineers working late to ensure deployment success over several weekends; three to four times a year.

3) Scaling for new instances:

The number of production instances for Salesforce.com has been increasing rapidly. Although this has allowed us to better accommodate our growing customer base, it has also added to the QA verification effort. Since each instance requires repetition of the verification process, a manual approach is not scalable.

4) Avoiding human error:

The manual testing approach was more prone to error. A new QA engineer may miss a critical test case resulting in defects and customer issues, which was unacceptable because quality is the number one goal for our R&D organization.

5) Keeping track of the communication:

Tracking of the testing process was difficult because so much back and forth was required between the various teams. Thus, when real issues were uncovered, it took longer to resolve them.

6) Managing stress:

Downstream effects included high levels of tension among the team and the need to plan vacations so that they do not conflict with major releases, etc.

3 New process proposed

In order to overcome the problems faced during manual verification, the QA organization devised a new test automation initiative called ProdTest. ProdTest was conceived as an automation framework allowing QA engineers to automate their sanity test cases and run them on production. The goals, technical details, challenges and benefits of ProdTest are discussed below.

3.1 ProdTest goals

The aim of ProdTest was to create a production validation tool to execute a suite of sanity test cases against the on-demand production environment using external API and UI requests. The tool would allow us to:

- Always meet the maintenance window to help us maintain availability[5]
- Reduce the number of QA engineers needed
- Remove the need for repetitive tasks
- Expose bugs that tend to appear only in the complex production environment
- Report test results and failures
- Automate both API and UI tests

4 Building ProdTest

In order to resemble the production environment accurately, ProdTest was developed as a client application which utilizes Salesforce.com's public web service APIs. These web service APIs are designed for use by our customers and partners to build their custom integrations. [6] Using the public APIs helped us 'act like the customer' without requiring additional functionality from the app server.

4.1 Technology

We started building the ProdTest framework as follows:

- Designing and implementing an Eclipse-based tool and UI for running tests
- Integrating with the testing framework JUnit
- Integrating with Selenium, the JavaScript based UI test automation framework [7]
- Importing our API wsdl for use by our test code
- Creating test utilities
- Adding configuration files to identify production environments

4.2 Process

Agile processes such as a daily scrum are widely adopted in Salesforce.com[8], so we utilized scrum for the development of ProdTest. We formed a new scrum team, which had a backlog, tasks and sprints. The team consisted entirely of QA Engineers. It was a flexible member team, such that various people participated in sprints at different times, based on their availability. This allowed ProdTest to grow continually with inputs from numerous QA engineers.

4.3 People

ProdTest was adopted as an important initiative by our technology division. It received great backing by senior management, adopted as an organizational goal and its importance emphasized. Writing ProdTest tests was not only greatly encouraged by the managers, but writing prodtests was included as a mandatory step before release. This greatly helped QA engineers to adopt and enhance the framework quickly.

4.4 Evolution of ProdTest

Initially, ProdTest had a small set of features. In time, these evolved into a more complex and sophisticated test set. The challenges and limitations faced helped us customize ProdTest to best meet our needs. Some of these challenges and steps taken to overcome them are discussed in the next section.

5 Challenges and solutions

Time taken to run tests:

We quickly adopted ProdTest and the number of automated tests increased. However, due to the increasing number of tests, the test suite took longer and longer to complete. In order to decrease the time taken to run ProdTest, we've taken the following steps:

- Categorize tests so that the most important (high impact feature) tests run before the others. We introduced 5 categories
 - **DVT** - Data validation tests after the upgrade
 - **prebatch** - Sanity tests run before batch services are up,
 - **postbatch** - Sanity tests run before the site is internally up,
 - **postwww** - Sanity tests that run after the site is internally up, and
 - **extended** - More detailed tests that are run after site is live

Categorizing our tests into various stages ensured that we could communicate the issues found as early as possible before proceeding to the next stage of the release.

- Support multithreading in ProdTest so that multiple tests can be run simultaneously on the same machine. This was especially challenging for Selenium tests. However, multithreading helped us reduce the test run time by making use of multiprocessor systems. We also saved time of opening and closing browsers for each Selenium test.

- Within a couple of releases, we had a better idea about maintaining test data, structuring test code and categorizing tests. We documented and shared these best practices with test authors so that they can write more efficient tests.
- Pre-create and maintain test data to avoid creating it during tests.
- Conduct additional code reviews to eliminate or modify redundant or inefficient tests.

Lack of support for certain types of tests:

Certain tests that are more complex or highly dependent on application server code cannot be easily automated with the limited public API based functions available in ProdTest. For a very small subset of tests, we needed to implement some one of automated solutions that utilized some portions of our internal APIs too.

Maintaining test data:

The automated tests in ProdTest run on pre-determined sets of data in the production environments. Keeping the data up to date for every release was a cumbersome task. Eventually, we solved this problem by creating tools to minimize the effort in data configuration.

Categorizing tests correctly:

Proper categorizing helps to minimize run time but it is important to educate and inform test authors what these categories really mean. A number of tests used to be annotated as prebatch, postbatch and postwww but should have fallen in the extended category. A longer pre-live test run meant a longer wait for allowing the site to go live. Extended tests run after the site has gone live and help discover lower priority defects that can be fixed in minor releases.

Eliminating false failures:

One of the most important challenges faced is eliminating false failures. Failures may result due to various environmental or data related problems. Differentiating between false failures and real issues that cause a test to fail continues as an ongoing effort. In order to reduce this effort, we've emphasized the need for maintaining a clean state of test data by having tests clean up after themselves. To identify environment issues quickly, some of the tests fail with an appropriate message such as 'Ensure the _____ process is up'.

5.1 Training

ProdTest has been an ongoing effort to reduce the number of QA engineers working during a release. The team was able to create the initial ProdTest infrastructure within 3 months of dedicated effort, and as a result, we have eliminated all manual test effort. Now there are two testing roles to support the release: the test master and test owners. A test master is responsible for triggering the test runs during various stages, gathering test run results and coordinating with test owners for re-executing the failed tests. Test owners, representing various functional teams, are standing by to manually verify any failed test to ensure that the failure was legitimate. We organized several training sessions in the form of dry runs to familiarize everyone with the new process.

5.2 Best practices

In order to minimize test run time, the following best practices were shared with test authors:

1. Selenium test cases take longer to run (due to additional work in spawning browsers) so write API tests whenever possible.
2. Be conservative of what goes into Go Live Tests such as prebatch, postbatch and postwww since these categories need to have the shortest runs possible.
3. Avoid doing test data setup in your tests. Create required data in the test organization early, much before tests are run on it.
4. Spread out test cases to run on different sets of data to minimize interference.
5. Assign an owner for each set of data.
6. Categorize test cases into separate categories, since our design allows different categories to safely run in parallel.
7. Create duplicate sets of test data to ease quick troubleshooting of test failures. In this way, if one set of data becomes corrupt or lost, one can rely on the second set.

6 Benefits of ProdTest

We significantly reduced the number of QA engineers needed during the deployment window by eliminating manual verification. This was our biggest achievement.

We also removed a great deal of stress for engineers by eliminating the need to manually run all the sanity tests within a very short time frame. Instead of running the tests, a subset of the test owners can be in a stand-by mode and concentrate on resolving any issues found by ProdTest.

Another major benefit is that ProdTest provides better deployment test coverage. Various complex user scenarios can now be added into the test suite, which were previously impossible to run manually due to time considerations. The added coverage helps uncover bugs that are specific to the production environment. For example, we've quickly caught errors due to processes being switched off, virtual IP configuration errors, etc. Having an automated run provides a predictable runtime for the sanity tests, which becomes extremely important when planning a system downtime.

Since the development of ProdTest, we have used it in other ways that yield additional benefits that we did not anticipate at first. For example, ProdTest is now used for other minor releases including patch releases, releases for database upgrades, etc. Running ProdTest for these releases at no additional cost ensures the highest quality for all our releases. Also, we run ProdTest a few times daily on our internal testing environments. These internal automatic runs help validate that the changes checked in for new feature development have not caused regressions. This helps us continuously monitor the quality of our current release in development.

7 Future Direction

ProdTest has come a long way. Today, it has over 1700 tests. These helped us reduce our deployment time and resources, and identify critical issues quickly. However, as the number of tests and deployment instances grows, we hope to further reduce the run time for critical tests. We are investing in tools that automate test data setup. The QA organization aims to achieve a very high level of dependability for ProdTest, such that we do not encounter any false failures due to environment issues, data issues, etc. This should help the QA organization reach our ultimate goal of being able to hand ProdTest over to the technical operations team, who can use it during deployment without any involvement from QA.

8 Conclusion

ProdTest serves as an excellent example of the strength of test automation. Today, almost all software companies invest resources towards software automation. ProdTest demonstrates how we can extend automation into different environments, which were previously not easy to test. Tools like ProdTest can bring about improvements in both process and product quality if designed and planned for correctly.

Aside from the intended benefits discussed in the paper, the ProdTest initiative helped us adopt and improve a slightly different type of agile development process. The ProdTest scrum team has been a flexible team consisting of several in and out members working at different times, based on their time availability. In spite of this, we have learned to take on new ProdTest commitments, task ownership, add requirements, and manage the product backlog in an effective and direct manner. This not only paves the way for future ProdTest related innovation and lays the foundation for kicking off other similar initiatives within the QA organization.

References:

- [1] Software as a Service, http://en.wikipedia.org/wiki/Software_as_a_Service
- [2] Platform as a Service, http://en.wikipedia.org/wiki/Platform_as_a_service
- [3] On demand software, http://www.webopedia.com/TERM/O/on_demand_software_delivery.html
- [4] Sanity testing, http://en.wikipedia.org/wiki/Sanity_testing
- [5] Trust.salesforce.com for Salesforce.com instance availability <http://trust.salesforce.com/>
- [6] Salesforce.com Web Services API http://wiki.apexdevnet.com/index.php/Web_Services_API
- [7] Selenium -Web Application Testing System <http://selenium.openqa.org/>
- [8] Babinet and Ramanathan, Dependency Management in a Large Agile Environment, Agile 2008 Conference, Aug 2008