PARASOFT

How Service Virtualization Eliminates Functional Testing Roadblocks at Staples

By Sanjay Chablani, Head of SQA Biogen Idec, reflecting on the service virtualization implementation he oversaw as Director of eCommerce Quality Management at Staples

Staples is committed to making everything easy for its customers, but ensuring positive customer experiences on their eCommerce site is far from simple. Functional testers must contend with the high number of dependent systems, subsystems, and services that are required to complete almost any eCommerce transaction—but rarely available for dev/test purposes. Learn how the eCommerce functional testing team leveraged service virtualization to more rapidly and more exhaustively test complex transactions across highly-distributed systems…

As the director of QA for the Staples eCommerce site, my team and I were responsible for ensuring that every transaction involving the eCommerce application operated seamlessly and reinforced the company's commitment to providing an easy customer experience.

Like so many modern applications, the eCommerce application interacted with hundreds of other systems and services, all of which were reused widely across the company and had multiple integration points. This complex distributed architecture was great for providing consistent reusable functionality across the organization and enabling us to minimize application footprints. However, it definitely complicated our team's ability to perform the functional testing needed to ensure that end-to-end transactions across the eCommerce site met expectations.

Some of the main challenges we faced with our functional testing included:

- **Environment availability:** With today's enterprise systems, being able to fully replicate environments for testing is no longer a possibility. Replicating a fully-connected environment multiple times would require too much money and effort—and sometimes it's simply impossible (such as when we need access to not-yet-implemented components being developed in parallel). Nevertheless, having complete test environment access was critical to our ability to execute almost any test. Inability to access just one dependent subcomponent could impact the testing of our entire eCommerce site.
- **Environment scheduling:** Since the subsystems, subcomponents, and services that we (and many of the other divisions) relied up on were all owned by different groups, we were always competing for access to these constrained resources.
- **Access to realistic test data:** To complete a single test, we needed to have test data refreshed and synchronized across the various services, subcomponents, and subsystems. Considerable effort and coordination among different teams was needed to get the right test data populated across all the systems.

Our initial attempt to overcome these challenges involved stubbing. At first, we built some pretty basic stubs, then we moved to "semi-intelligent" stubs, which we deployed in our test environments so they could stand in for the actual services. However, stubs had their limitations. They are highly static and they typically require the involvement of developers who are quite familiar with how the applications interface.

When we heard about service virtualization, our group quickly recognized its potential to address our functional testing challenges, and we were the ones that took the lead on the company's service virtualization initiative. What initially drew us to service virtualization was how much easier it would be—in terms of recording traffic, deploying virtual assets that simulated this traffic, then making those virtual assets available for on-demand provisioning.

The goal of our service virtualization initiative was to create a library of virtual assets that would represent service providers and service consumers, including the message bus (service proxy). Essentially, if a system was not something we were responsible for testing, we would create virtual assets for it. This would allow us to simulate any dependency involved in our various end-to-end tests. To promote the development of such a rich library, we made virtual assets a required deliverable within the SDLC.

Ultimately, we found that service virtualization provided us the following benefits…

Faster release cycles

With service virtualization, we could start testing earlier in each cycle, and reduced the time required to execute our test plans. This was especially critical on parallel development projects, such as when the Retail, Warehouse, and eCommerce teams were all working on functionality related to online ordering with in-store pickup. This was a complex project with a very aggressive timeline.

Using service virtualization to simulate resources that were still being developed, each team's development and testing could move forward without waiting on the others. With the virtual assets, we could start integration testing much earlier than if we had to wait for all the dependent components to be completed. This helped us get everything running smoothly even before we integrated all the completed components. Ultimately, we not only completed the project on budget, but actually ended up deploying it two weeks early. This success really helped us promote service virtualization adoption across the company.

Greater control over environment stability and application behavior

We could complete test scenarios that were previously impossible (due to data privacy guidelines) since service virtualization enabled us to mimic behavior that was very data dependent. Moreover, we could not only trust that the dependencies we needed to access would be available, but also exert additional control over them (e.g., to test corner cases, error conditions, etc.) to achieve greater test coverage.

Ability to test on our own schedule

We no longer had to wait on others to complete applications being developed in parallel, and we no longer had to compete with other teams in order to schedule limited windows of access to highly-constrained shared resources.

Reduced issue and defect resolution times

Before, we had to look at all the different subsystems to try to figure out what was causing a problem. By turning on different virtual assets, we could really isolate different subsystems and quickly zero in on the root cause of the problem.

About Parasoft

This resource was published by Parasoft, industry-leader in Service Virtualization, API Testing, and Development Testing.

Parasoft researches and develops software solutions that help organizations deliver defect-free software efficiently. We reduce the time, effort, and cost of delivering secure, reliable, and compliant software. Parasoft's enterprise and embedded development solutions are the industry's most comprehensive—including static analysis, unit testing, requirements traceability, coverage analysis, functional & load testing, dev/test environment management, and more. The majority of Fortune 500 companies rely on Parasoft in order to produce top-quality software consistently and efficiently as they pursue agile, lean, DevOps, compliance, and safety-critical development initiatives.

Upon entering the API Testing marketplace in 2002 with Parasoft SOAtest, Parasoft pioneered service virtualization capabilities beginning with the industry's first "stub server." Since then, Parasoft has been leading the service virtualization marketplace, as recognized by Gartner, voke, Forrester, and Bloor, as well as awards including the Jolt Grand Prize (2012), Info-Tech: Trend Setter and Market Innovator Awards (2013), and Wealth & Finance Award for Most Innovative Software Vendor (2014).

Contacting Parasoft

USA	Phone: (888) 305-0041	Email: info@parasoft.com
NORDICS	Phone: +31-70-3922000	Email: info@parasoft.nl
GERMANY	Phone: +49 731 880309-0	Email: info-de@parasoft.com
POLAND	Phone: +48 12 290 91 01	Email: info-pl@parasoft.com
UK	Phone: +44 (0)208 263 6005	Email: sales@parasoft-uk.com
FRANCE	Phone: (33 1) 64 89 26 00,	Email: sales@parasoft-fr.com
ITALY	Phone: (+39) 06 96 03 86 74	Email: c.soulat@parasoft-fr.com

OTHER See http://www.parasoft.com/contacts