Cloud CRM’s Evolution and Impact on QA

As cloud-based customer relationship management systems such as Salesforce.com go mainstream, quality assurance and testing teams need to reinvent their skills and processes to ensure timely and effective application deployment and return on investment.

Executive Summary

Cloud computing is not a completely new technology; rather, it represents a change in the software ecology for solution creation, distribution and consumption. Cloud architecture has already had disruptive effects on the techno-business landscape, particularly in pushing established vendors aside as new players emerge. The cloud’s most critical change is the amount of power and flexibility provided to application designers, content creators and end users. The traditional roles of developers, testers and end users in the software development lifecycle (SDLC) model have become interchangeable in some respects; all roles must adapt to survive, if not thrive, in this new age.

This white paper discusses enhanced testing techniques that have delivered business benefits amid challenging cloud application testing engagements and which can be adopted across a wider spectrum of projects.

The Cloud: A Brief History

The cloud revolution started with the software as a service (SaaS) model. In this approach, the vendor provides end users a full software suite that is hosted on the vendor’s premises. These end users typically do not need any additional hardware or infrastructure, except network connectivity and an Internet browser to run the applications. Web-based e-mail services such as Hotmail are among the earliest examples of SaaS. However, the full potential of the model was realized only after multi-tenant architectures came into existence.

In the multi-tenant model, each client (usually, an enterprise) shares computing resources provided by a host service provider, but has separate and distinct data and code bases. This enables companies to secure sensitive business information even when the same application platform is being used by other companies. Moreover, any client can customize and add extra features to its instance of the application, without impacting other customers that are sharing the same server and database. New features can be implemented by installing third-party plug-ins or writing custom code. In some cases, clients can build unique versions of the application, while using...
the computing resources and APIs provided by the vendor. This approach is known as platform as a service (PaaS). PaaS offers organizations an unprecedented amount of power and flexibility. It also allows user companies to focus on their core business rather than investing time and money in building IT infrastructure.

Infrastructure as a service (IaaS) takes this concept to the next level, by virtualizing the entire hardware layer — including the server, file-based storage, firewalls, load balancers, IP addresses, virtual local area networks (VLANs) and software bundles (see Figure 1). For example, Heroku, provider of a cloud-based build and deployment platform, relies completely on virtual network services that Amazon EC2 provides to run its own software stack.

Running applications over SaaS, PaaS or IaaS brings immense benefits to the enterprises in terms of Cap-Ex and Op-Ex reduction, but when it comes to the project lifecycle — and testing in particular — businesses need to radically overhaul IT methods and thought processes to extract critical performance advantages.

The Cloud Difference
Why does the cloud matter? Take the example of Salesforce.com, one of the most popular cloud-based customer relationship management (CRM) systems in use today. Salesforce offers an end-to-end CRM and contact management system. When an organization purchases a license from Salesforce, it gains access to many pre-built modules, database tables and storage spaces provided via Salesforce’s cloud-based CRM platform.

For small-scale businesses, out-of-the-box Salesforce applications often cover all business requirements. However, large-scale businesses often require customization of the company’s CRM application to match their business processes, and they must integrate Salesforce with other enterprise applications. Regardless of whether internal IT teams or third-party consultants support these customizations, standardized and well-established practices for software development still apply for efficient delivery. This is a key requirement across the SDLC, from specification and design through coding and testing.

The above statement does not indicate that a strict Waterfall model needs to be followed; any standard iterative model can be used. The primary issue here is that standard SDLC models require a greater amount of time spent on different phases than the quick deployment most cloud vendors suggest when it comes to application implementation. The expectation disconnect typically ensues after the cloud vendor’s promise that the application can be used as is, with just a little bit of tweaking. Timelines tend to extend well beyond the promise of immediate time-to-value, an issue for most companies looking to deliver quick-hit returns on investment after making large upfront outlays on software licensing. Therefore, IT teams need to figure out a way to shorten the cycles while still retaining software quality.

Even within IT teams, different groups face different sets of challenges. Historically, until three-tier client server architectures emerged, IT roles were fairly straightforward. A group of business analysts specified the business requirement. Another
group of architects designed the application. Developers wrote the code, and testers verified that the codes worked based on the specifications of the business requirement.

This no longer applies in a cloud project, in which, based on the specific application, 30% to 70% of the application is delivered out of the box by the cloud vendor. The business analyst who writes the requirement must understand the nuts and bolts of the platform. This analyst needs to understand the feasibility of implementing a specific customization requested by the business to provide valuable feedback on time. The developer, who is implementing the requirement, needs to decide whether a new feature can be implemented through built-in system capabilities (commonly referred to as configuration change) or must be coded from scratch. The tester responsible for validating the system needs to decide where to spend the limited available time based on the risk, which depends on the way a system change was implemented.

Therefore, every group needs to be intimately familiar with the tasks and decisions of the other groups in order to plan their activities efficiently and accommodate the project’s overall budget.

Independent Testing for Cloud Applications

A valid question is often asked by project stakeholders: If the roles are so blurred, is there any justification of the cost incurred by maintaining a separate testing team? The answer to this is both yes and no. Yes, because regardless of the type of activities performed by a testing team, the fundamental principles of testing remain in place; that is, to objectively assess the application and expose critical flaws before end users use the application in a production environment. No, because traditional testers do not really add much value in this case. Only a specialized group of professionals who are trained and experienced in the architectural details of the cloud platform used to develop the application, and who also understand the concepts and philosophy of testing, can produce the kind of result called for in a cloud project.

Some steps can be taken to ensure that testing teams do not become a bottleneck in the otherwise fast-moving cycle of SaaS and PaaS implementations. What follows is a look at the tools and techniques applied by an empowered testing team and at the kind of values added in a real-time project.

Proposed Lifecycle for Cloud Application Testing Projects

![Diagram](image)

- Requirement Specification
  - Field specification, formula/trigger/workflow description and security grid reference created by business analyst.
- Development
  - Significant portion of testing done before actual system testing cycle begins.
- Testing
  - System testing cycle consumes much less time than usually required.
  - Field Validation
  - Formula Validation
  - Trigger & Workflow Validation
  - Security Validation
  - Web Service Validation
  - Integration Testing
  - Functional Testing
  - Regression Testing
  - User Acceptance Testing

Figure 2
A Novel Approach: Redefining the Testing Process and Tester’s Role in SDLC

In an effort to reach out to individual customers, a North America-based insurance company decided to implement an automated e-mail marketing system. This system required users to schedule individual or recurring campaigns with complex parameters for identifying eligible users by using its Salesforce CRM system. The capability needed to be delivered to sales reps within 18 weeks, before a nationwide promotional event was to kick off.

The company has several different lines of business and subsidiaries. Individual user contact data is strictly guarded by the unit that owns it. Strict data security policies mandate a complex set of rules defined by how campaign owners searched for eligible clients, whose details may exist in the Salesforce application or two other legacy applications.

The standard approach to testing in a case like this usually involves waiting for complete functionality to be delivered to the QA environment before starting the testing process. Feature-specific and end-to-end tests are designed and executed in multiple cycles to ensure all functional and non-functional requirements are satisfied. That is an effective, time-tested approach for testing this type of application. However, in this case, there was an issue: lack of time.

Figure 2, previous page, walks through the best-case process used by our team of cloud testing experts. During the development phase, the team started with test cases that were Salesforce-configuration-specific and did not have any dependency on the user interface, and gradually moved into testing use cases through multiple steps.

**Step One: Verifying Fields’ Properties**

For any nontrivial Salesforce implementation, the development process for a new feature starts with the creation of custom objects and fields. These fields’ properties might define the number of characters allowed in a text field, or the permissible values in a pick-list need to meet existing criteria. After the development team (or Salesforce configurators, depending on how the team is organized) created the field, the testing team used the configuration in a unit environment to manually create an Excel worksheet to compare against the Salesforce configuration. This helped them ensure all GUI elements matched the criteria established in the requirement.

In the project illustrated in Figure 3, this approach helped the team filter out human errors during configuration of the unit environment and cut out a significant amount of system testing time.

Excel Worksheet Configuration Comparison

Figure 3
Step Two: Verifying Formulas Associated with Fields

The next step was to verify the field-specific formulas. This is where the process got a little tricky. In an ideal world, there must be clear documentation of all rules that are applied to a field. In this scenario, however (and this holds true for most cloud implementations), the application configuration happened in real time during requirement gathering sessions. There was no time to build the comprehensive documentation. So in this case, the testing team needed to understand what a formula does by inspecting it, and then it had to confirm if it could meet the expectations of the business users.

For example, consider the code highlighted below in Figure 4.

It is difficult for a regular testing resource to deduce that the field contains one of the values – West, North, Central, East or South – which represents an individual region and is calculated based on the billing state. This is the biggest value of a cloud specialist tester. The tester can examine the code base and judge its functionality, without much help from the developer or the business analyst. System knowledge, in addition to business knowledge, enables testers to validate the field functionality without requiring a thorough document, which would otherwise be required by any other testing team.

Step Three: Verifying Triggers and Workflows

Formula (and regular) fields are unquestionably the simplest of the building blocks of a functioning application. At the next level are the triggers and workflows used to define the functionalities of the application. Salesforce triggers and workflows are created by using the Apex programming language, which is a variation of Java. A clear understanding of all triggers used in a system helps QA resources start testing early, even before the end-to-end application is installed.

Step Four: Verifying Data Security

Salesforce has a complex multilayer data security model, which not only protects companies from unauthorized external access, but also guards against unauthorized internal access. The security model ensures that only individual users or groups have access to the data that they are authorized to use. The security structure is created after the metadata (table structures and user profiles) is created for a Salesforce application. This can be tested even before a single line of code is written.

Also, different security features, such as the range of IPs that are allowed to access a specific application or log in hours, can be implemented and tested before the actual application is set up.

Step Five: Verifying Web Services

A complex Salesforce implementation rarely works in a stand-alone manner. In most cases, other applications need to communicate with Salesforce in real time to effectively complete the business processes. Different types of Web services can be used to achieve this.

Returns Text Value of “North,” “South,” “East,” “West” and “Central” Based on the Billing State/Province of the Account

```
IF(ISBLANK(BillingState), "None",
```

Figure 4
For example:

- REST API, for interactive integration.
- SOAP API, for XML-based integration.
- Bulk API, for data load.
- Streaming API, for real-time notifications such as stock market updates.

Regardless of the API type used, input/output, data mapping and authentication need to be thoroughly tested. This type of testing requires a good grasp of the technology platform on which the cloud application is implemented.

After completing the aforementioned steps, all field properties, field level security for different profiles, triggers, workflows and integrations all were tested before the code was deployed into the QA environment. Once the code was in a QA environment, the team started testing the end-to-end functionalities and used the cases without worrying about basic configuration or workflow issues. By then, many bugs were already filtered, making the functional testing cycle significantly smoother and shorter. Testers were able to concentrate on testing user stories as against focusing on system configuration testing.

Following this method definitely requires additional planning and training efforts on the part of the testing team. However, the end result often is a satisfied customer. The above example describes a testing approach meant exclusively for Salesforce.com deployments. A similar approach, however, can be followed for any cloud-based CRM system that uses a metadata-driven design approach such as Microsoft Dynamic CRM or Sugar CRM.

**Benefits: QA’s Contribution to Business Value**

With this modified V-model-based, left-shifted testing approach, the team was successful in delivering highly complex functionality, in six weeks of development and four weeks of testing, with zero critical or high-priority bugs in the system. Of the four weeks of QA, one week overlapped with the development cycle, thereby reducing the project duration effectively to nine weeks. The time gained in the development and QA cycles was effectively utilized by business users to review the capabilities in detail and provide productive feedbacks to make key features more user-friendly. The promotional event turned out to be one of the major turning points for the insurance company, enhancing its status as a market leader in targeted segments. That’s the kind of real world impact that makes any tester look forward to the next grinding workday, isn’t it?

**Looking Ahead**

Cloud CRM testing, much like hardware testing or middleware testing, needs to be treated as a separate discipline. Cloud CRM has its own methodology, process, skill and training requirements. Trying to apply the practices of standard functional testing is inappropriate, given the platform’s IT and business demands. Old-world methods are insufficient for achieving the ROI expected from a cloud CRM application.

Cloud-based CRM application testing teams need to enable their members to take up new roles and responsibilities. The shifting testing model mindset and availability of tools and training to team members are the keys to success.

A comprehensive training plan by internal and external certifications can help new resources embrace newfound roles. Our shift-left, V-model-based framework for cloud application testing, as described above, can provide the thinking and wherewithal for testing teams to more effectively and efficiently take cloud-based CRM testing to greater heights.
Footnotes
2 http://aws.amazon.com/what-is-cloud-computing/.

About the Author
Parantap Samajdar is a Project Manager within Cognizant’s Quality Engineering and Assurance Business Unit. He specializes in managing SaaS, PaaS and IaaS solutions testing, and test automation framework design and development projects. Parantap has over 10 years of relevant experience leading complex testing engagements, spanning Salesforce.com, Siebel, WebSphere, mainframe, test automation and performance testing. He currently leads one of the largest Salesforce.com engagements within Cognizant for a major insurance service provider in North America. Parantap holds a bachelor of engineering degree in computer science and engineering from University of Bardhaman, India. He can be reached at Parantap.Samajdar@cognizant.com.

About Cognizant
Cognizant (NASDAQ: CTSH) is a leading provider of information technology, consulting, and business process outsourcing services, dedicated to helping the world’s leading companies build stronger businesses. Headquartered in Teaneck, New Jersey (U.S.), Cognizant combines a passion for client satisfaction, technology innovation, deep industry and business process expertise, and a global, collaborative workforce that embodies the future of work. With over 75 development and delivery centers worldwide and approximately 178,600 employees as of March 31, 2014, Cognizant is a member of the NASDAQ-100, the S&P 500, the Forbes Global 2000, and the Fortune 500 and is ranked among the top performing and fastest growing companies in the world. Visit us online at www.cognizant.com or follow us on Twitter: Cognizant.