FORRESTER®

For: Application Development & Delivery Professionals

Navigating The Agile Testing Tool Landscape

by Diego Lo Giudice, July 18, 2013

KEY TAKEAWAYS

Temper Testing Process Rigor With The Need For Continuous Testing

Tools that model and provide a prescriptive, top-down testing process won't work with Agile development. Instead, Agile projects maintain quality via a continuous testing approach: Unit testing, increased automation, test-driven development or user acceptance test-driven development, and exploratory testing replace manual, prescriptive processes.

Consider What Developers Want As Part Of Your Agile Testing Tool Strategy

Gone are the days where testing was only a job for the QA specialists. In Agile, developers are first-class testing citizens; they need testing tools that fully integrate with their development tools, especially their IDE. Empower developers to experiment and find out what works best for their unit, automation, and nonfunctional testing.

Integrate And Test Earlier With Service Virtualization Tools

Early integration and testing is a common challenge in Agile development, especially for large, complex application projects. Service virtualization tools will allow your developers to simulate access to the services -- such as mainframe access and expensive external online services -- they need for testing their code.

Make Sure Your Agile Project Management And Test Management Tools Work Hand In Hand

On Agile projects, testing and development are one integrated process. Make sure that your choice of an Agile test management tool allows both testers and developers to easily share and synchronize development and testing activities from a common backlog. Facilitated collaboration across multiple development tools is a must.

Navigating The Agile Testing Tool Landscape

Revamp Your Testing Tool Strategy For Agile Development

by Diego Lo Giudice
with Jeffrey S. Hammond, Tom Grant, Ph.D., and Rowan Curran

WHY READ THIS REPORT

As your developers shift to Agile practices, they will invariably perform more testing themselves. So where does that leave your quality assurance (QA) professionals? They need to adapt by getting deeply involved in the daily operations of the development team. Advanced practices like test-driven development, increased testing automation, and continuous build and integration make a significant impact on the day-to-day activities of both developers and testers. These shifts in testing practices also change how development teams select testing tools: Developers want tools that easily plug into their integrated development environments (IDEs), while QA and other software professionals prefer tools that offer a higher level of abstraction and are easy to use. This report details how Agile teams are revamping their testing tool strategy to work well in an Agile development context.

Table Of Contents

Notes & Resources

Forrester interviewed 20 vendor and user organizations, including BetterCloud, CA Technologies, FINRA, HomeAway, HP, IBM, Micro Focus (Borland), Microsoft, New Relic, NTT Data, QMetry, QASymphony, Seapine, Silverpop, SmartBear Software, TechTalk, ThoughtWorks, Ultimate Software, and the US Department of the Treasury Financial Management Service.

Related Research Documents

Consistent Performance In Agile Teams Must Include Testing
January 17, 2013

Forrester's Agile Testing Maturity Assessment Tool
January 15, 2013

Five Ways To Streamline Release Management
February 7, 2011

AGILE PRACTICES ARE BREAKING APART TRADITIONAL TESTING ORGANIZATIONS

Agile development changes the practice of software quality assurance in many ways.[1] How? For Agile teams, 1) product owners and/or business analysts, developers, and testers work together on single projects — side by side and colocated whenever possible; 2) testing becomes the responsibility of everyone in the team; and 3) being "done" means that development teams meet established quality goals and complete testing activities for every sprint and release. Agile development puts a larger quality burden on developers and requires that they do more. This starts with unit testing, extends to test generation and automation, and often includes getting more involved in the daily execution of testing activities. These changes aren't just theoretical; data from Forrester's Forrsights Developer Survey, Q1 2013 shows that frequent use of testing tools is already a reality for 30% of developers (see Figure 1).

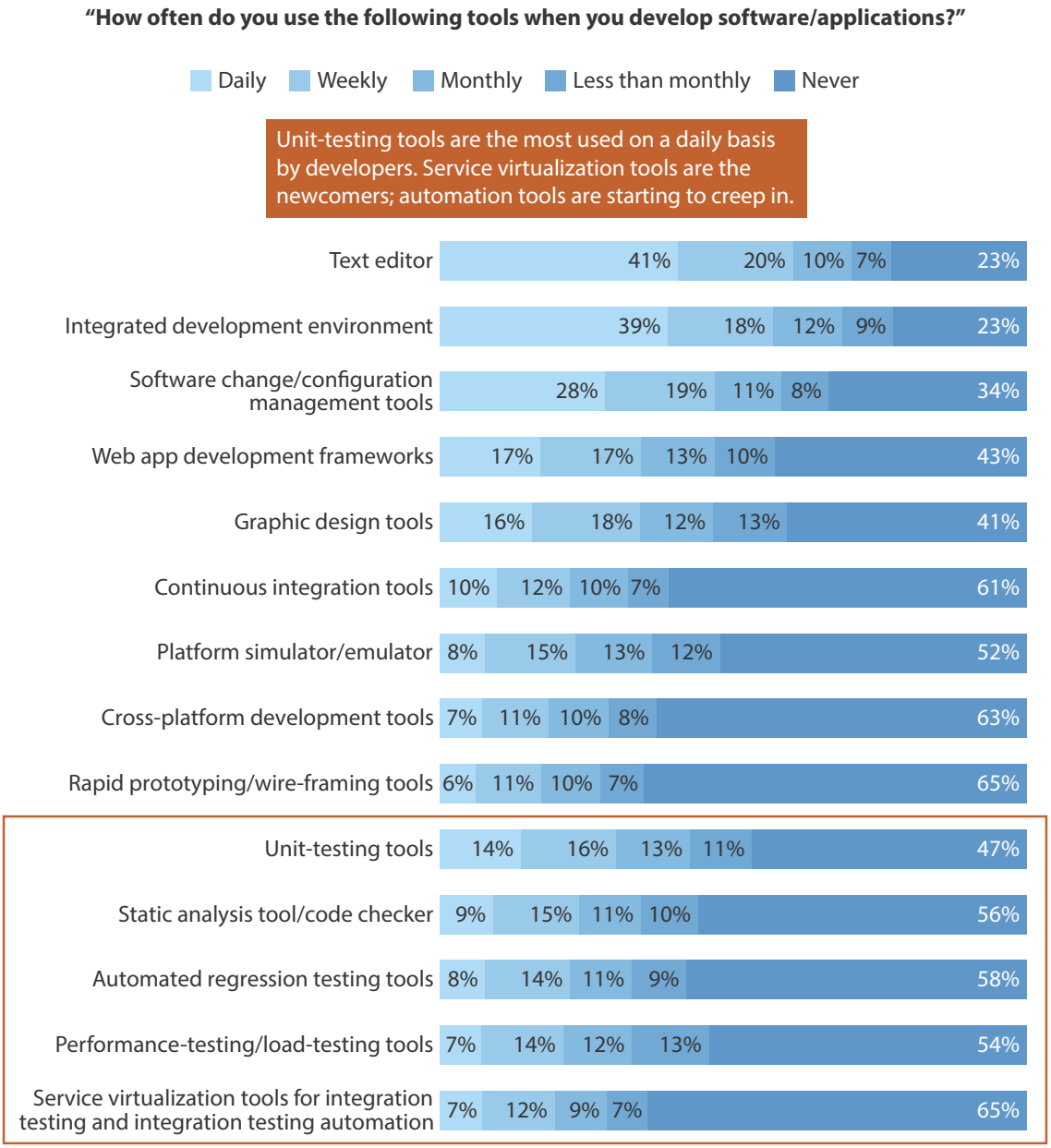
Traditional Testing Practices Struggle To Support Agile Development Teams . . .

Traditional testing practices were designed to optimize the operation of large, centralized testing groups using a testing center of excellence (TCOE) model. But this shared-services approach is breaking down at Agile organizations because it can't support the rapid delivery rates of Agile development teams. For example, the US Department of the Treasury's Financial Management Service had to completely isolate a major Agile project from the regular IT department's governance processes, including the TCOE. The team used a new testing and development process leveraging a behavior-driven development (BDD) approach with great success.[2] The development team selected and used Cucumber, an open source testing tool, because it supported the new approach better than traditional testing tools. The end result was a higher degree of automation and speed in testing that would not have been possible had the team been forced to comply with TCOE governance and processes.[3] Forrester sees the Agile team bypassing the TCOE with increasing frequency. Why can't traditional testing keep up with the velocity of Agile teams?

- **Large volumes of manual test activities slow down delivery.** Manual testing is the oldest and still most common approach to testing software within TCOEs. Test professionals develop test cases that cover as much functionality as possible, which makes the velocity problem worse. There's no way around the fact that manual testing is time-consuming and resource-intensive. Even throwing a phalanx of manual testers at the problem doesn't work — manual testing simply can't keep up with daily builds, continuous integration, and the functional and nonfunctional testing cadence of Agile delivery teams.

- **Teams put off testing until the end of projects, squeezing it in the process.** Another anti-pattern frequently found in traditional testing approaches is that teams only start testing once they have developed and integrated the system — partly due to the expense and time required for manual testing. Unfortunately, projects are often behind schedule, so teams compress and sacrifice the activities left at the end. As a result, testing time gets sacrificed to make up for delays in other processes, compromising quality.

Figure 1 Three In 10 Developers Unit-Test At Least Weekly



Base: 2,038 software developers in North America, Europe, and Asia
(percentages may not total 100 because of rounding)

Source: Forrsights Developer Survey, Q1 2013

94241 Source: Forrester Research, Inc.

- **Late-breaking defects can derail projects.** The longer a defect sits in code unfixed, the longer it will take a developer to fix it, with dire consequences for project deadlines. Developers move on to new code and new problems and lose the context of the features they've delivered (can you remember what you ate for lunch last Monday?). When they have to come back to a defect that they wrote weeks or months ago, it takes time to re-engage with the context of the code.

- **Teams build up too much technical debt.** One sure-fire killer of on-time delivery is finding out late in the development cycle that your application has major quality problems. Late discovery of defects lead to high rates of rework and waste. It's even worse if the quality issues are systemic, like architecture design issues, or if someone discovers that basic user functionality is missing. The earlier testing starts, especially system testing and user acceptance testing (UAT), the earlier systemic risks will surface.

. . . And The Problem Can't Be Fixed With Minor Organizational Tweaks

As TCOEs struggle to respond to the mismatch in processes that an increase of delivery velocity creates, they find that making minor changes to process or organization structure is ineffective. Why?

- **TCOEs don't necessarily reduce QA costs in an Agile environment.** When testing teams are separated from development, it's typical for testers to try to find as many bugs as possible — once developers write the code. Developers, while responsible for fixing the bugs, only see the results of poor attention to quality in retrospect, when the consequences of their actions are harder and more costly to fix. The net result is that TCOEs keep costs low through labor outsourcing and less overall activity, but also by shifting costs back upstream into the development cycle through higher levels of scrap and rework. Tools that are designed to help testers document bugs and help developers reproduce and fix those bugs are useful, but do little to reduce the systemic issues that result in high scrap and rework costs.

  > "We have no central team for regression testing, as we believe that it takes ownership of quality away from developers. We want developers to feel that quality is part of their job." (Robert Sellers, director of QA, Silverpop).

- **Testers don't adequately understand rapidly changing backlogs.** When a company centralizes its test execution activities, testing schedules can't keep up with the rapid course corrections that characterize Agile development teams. When user stories change, Agile teams often reprioritize them in the backlog and don't pay a lot of attention to developing the types of formal, detailed requirements that traditional TCOEs use as inputs to develop test cases. Traditional front-loaded test management (TM) and planning processes are not designed to support rapidly changing priorities and the short cycles required by Agile. This problem is compounded when TM tools don't link into the Agile project management requirements backlog.

- **TCOEs don't integrate well with modern continuous delivery practices.** Segregating testers from developers makes it hard to integrate their work into a continuous delivery pipeline. Fast-moving teams don't build code and then hand it off to a testing organization; they build code, deploy the application, execute it, and immediately observe the results. This is especially true for development teams that employ multivariate testing, which is common in web application and mobile development.[4] These teams make a change, deploy it to a subset of servers, compare the results from each execution branch, and then decide if the change is successful. Teams that employ blue/green deployment (or red/black if you're Netflix) replace system testing and UAT environments with multiple production environments, and they're always expanding one environment while bringing another one down.[5]

These process and organization mismatches indicate that the TCOE model is outdated and offers a diminishing return on investment for Agile organizations. Accordingly, over the next three to five years, increased adoption of Agile practices will speed the breakdown of the TCOE model, even in organizations using a hybrid of waterfall, iterative, and Agile processes.[6] The breakdown of the TCOE as we know it will further accelerate as development organizations devote more resources to mobile application development, where Agile development practices are a must for success.[7] If you want to get ahead of this trend, you'll need to reorganize your testing processes and tools to make them an integral part of your development teams — and your developers will need to hammer out an agreement with the business sponsors they support on ensuring system quality. If you're ready to take the necessary steps to update your testing practices to support Agile development, read on — the rest of this report is about the tools and processes that Agile teams are using to form the next-generation, Agile testing landscape.

The Agile Testing Landscape Remixes Existing Test Processes With New Ones

What elements of traditional test practices will carry over into an Agile context? First of all, the role of the QA tester remains crucial — but as an integral part of the Agile team.[8] We also think that it's important to have a strong QA practice leader, even if day-to-day reporting relationships become less significant. A forward-looking practice leader can smooth the introduction of the new practices that are part of an Agile testing approach. These include:

- **Constant collaboration between development team members.** User stories on the backlog become shared artifacts for collaboration, coordination, and reporting between product owners, QA testers, app-dev professionals, and other stakeholders. Product owners prioritize user stories and collaborate with QA testers to decide on the user acceptance criteria for each user story. In addition, testers and developers collaborate to ensure that release and sprint backlog estimates include all testing activities. Testers track testing data and generate analytics for coverage and quality reports. Some Agile project management tools already provide basic TM capabilities to organize test cases in functional groupings with high-level reporting on testing activities. For more advanced TM, many integrate with existing TM tools (see Figure 2).

> "Rally clients push user stories into QMetry and associate these with test cases and testing suites to gather detailed information on requirements coverage and passed and failed tests and decide what further test cases to build. Developers can use activity streams and remote linking to get timely, contextual information on associated test cases in their JIRA instance." (Devang Mehta, head of marketing and business development, QMetry)

- **A need for speed that demands a different approach to test automation.** The app-dev leaders we spoke with warn that buying expensive script recording tools does not guarantee greater levels of automation. Instead, they recommend focusing less on the tools employed and more on who uses scripting and automation tools and how. Agile teams that have achieved higher levels of automation have shifted responsibility for automation into the hands of developers and focused more time on the automated test architecture and effective reuse practices. New tools facilitate this shift by providing tight IDE integration. Another key difference is the emphasis on creating tests as part of the overall delivery process. Devs may not like it at first, but they quickly get used to writing test cases when it's a required step in their sprints.

> "Tools matter, but what matters most is using them in the right way. Automation can become a maintenance nightmare if you don't pay attention to the right granularity of acceptance criteria, modularity of test cases, and design for reusability." (Christian Hassa, managing partner, TechTalk)

- **A test suite architecture that keeps automation costs under control.** Maintaining automated tests becomes harder and more expensive as the number of automated tests increases. It's even worse if there is poor traceability between requirements, test harnesses, test cases, and automated code. Poor architecture and monolithic test batteries are often at the heart of the problem, but tools have some responsibility for promoting effective test execution suites.

> "Tools don't always support modular test creation, help spot opportunities for reuse, or identify duplications. Linking test cases or scenarios to actual tested code is key for traceability, but keeping the two types of artifacts separate is crucial to avoid ripple effects during change." (Chad Wathington, managing director, ThoughtWorks Studios)

- **Incorporating whatever works, including developer-selected tools.** If an organization has a TCOE, the budget for testing tools naturally flows there. This creates a problem for Agile teams that want to do more testing. They don't have budgets for expensive testing tools, so they often adopt open source tools as a necessity. Even when budget exists, developers often don't want to spend time dealing with their purchasing department and don't have the time to learn an overly complex tool with features they might not need — so they end up working with tools that they can easily find and which help them do a good-enough job. As a result, Agile development teams are the primary entry point for open source testing tools, many of which are designed

specifically for developers. A forward-looking QA practice leader will embrace this reality, as it leads to the right endpoint — greater developer responsibility for quality code. The opposite approach is like trying to stem the tide.

> "We use JMeter for performance testing if the scale is not too large; tests are run quarterly. As an open source project, JMeter scales cheaply, it's easy to stand up, and it works inside our developers' local environment." (Carl Shaulis, senior engineering manager, HomeAway)

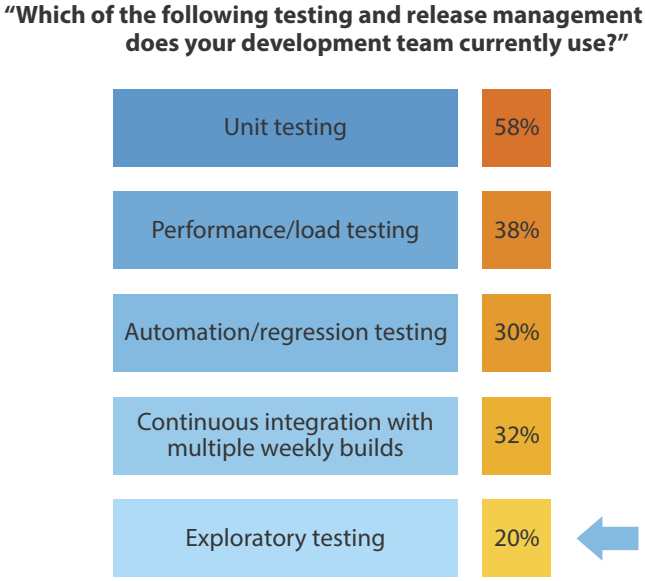
Another aspect of developer selection of testing tools is that developers generally want to test in the programming languages they develop code in, execute tests in the browser or IDE where they debug, and use their existing task management system. They also want to spend as little time as possible creating and maintaining scripts. Developers focus more on the immediate technical aspects of testing: Does it compile? Does it match the user story? Are there unit-level defects? So focus QA professionals on the extended aspect of testing, including exploratory testing, performance testing, and UAT activities.

- **Focus QA professionals on leading testing, not just managing it.** As part of an Agile team, QA professionals will no longer be in charge of enforcing testing activities against a testing plan that's fixed in advance. This would be useless, as plans need to be iterative and reviewed at each sprint; it would be hard due to iterative and frequent course changes; and it would be counterproductive to the new culture of self-management. Instead, QA professionals will need to lead testing activities and help teams make important decisions on quality goals, help product owners with acceptance criteria, make sure that developers aren't "done" until they've tested, and layer on qualitative manual testing to supplement automated testing.

> "We have a QA lead, product manager, and developer manager in each project. The QA lead starts with an open source mind-mapping tool and works with the team to brainstorm and capture testing ideas and organize them in a meaningful way. After setting the strategy for coverage, automation, and the key functionality to test, we pull these into Rally in the form of 'charters' for later session-based testing." (Carl Shaulis, senior engineering manager, HomeAway)

- **Earlier and more frequent UAT.** Business stakeholders need to be involved in Agile development and provide frequent feedback. How frequent depends largely on their availability. The fact is that UAT for Agile is less ceremonial than in traditional approaches, where it is run at the end of long development/testing cycles. Agile teams perform UAT early and continuously cover the incremental implementation of functionality.

Figure 2 Developers Are Also Participating In Exploratory Testing Activities



Base: 698 professional software developers, internal IT developers, game developers, and consultants in North America, Europe, and Asia (multiple responses accepted)

Source: Forrsights Developer Survey, Q1 2013

94241 Source: Forrester Research, Inc.

THE TOOLS LANDSCAPE REFLECTS THE ASCENDENCY OF AGILE TEST PRACTICES

The set of testing practices detailed above are changing the landscape of testing tools as new tools and classes of tools emerge to meet the needs of Agile teams. Changes include: 1) the increasing importance of service or test virtualization tools (these are not so new, but are becoming increasingly important as smaller, more numerous Agile teams interact with each other, making early integration testing more mandatory); 2) the substantial growth and proliferation of open source testing tools; 3) the adaptation of traditional TM tools to Agile practices; 4) the growing presence of low-cost, cloud-based TM tools; and 5) the fact that established test data management (TDM), performance, load, and security tools are lagging behind while product teams figure out what Agile testing means to their traditional customers (see Figure 3). As development teams adopt Agile testing practices, they affect the use of testing tools in several ways:

- **Unit-testing tools gain in importance.** Many of the app-dev and QA leaders we interviewed for this report require the building of automated unit tests, which help build quality into code from the very beginning of development. When teams develop unit tests for all functions and

methods, the tests help quickly identify changes — and, most importantly, regressions — that introduce system faults. Unit tests are also the bricks that firms use to build the walls of more comprehensive functional automation and regression testing suites. Teams further extend the concept of unit testing with test-driven development (TDD). TDD works when developers iterate over a very simple cycle: write a test that fails; write code to make it pass; and refactor. Developers commonly use frameworks like JUnit, NUnit, and xUnit.net to build unit tests and drive TDD. While it sounds great in practice, TDD can be a challenge to implement, as it requires a substantial mind shift for developers. Tools don't do much to help make the cultural transition to TDD simpler, although they do aid in its execution.

> "We leave the choice to teams as to where they should use TDD. Each team has a different philosophical approach; some teams do pairing, some don't. If you force TDD on developers, it won't work. We find that projects that are more API-driven work better with TDD than UI-driven ones." (Bjorn Freeman-Benson, VP of engineering, New Relic)

- **BDD and UAT design tools further improve code quality.** BDD is an extension of TDD, but it focuses explicitly on testing code behavior. QA professionals and/or business analysts write test scenarios as declarative English-language statements that everyone on the team understands; developers then write automation code for each scenario. Most of the tools that currently support BDD are open source, including the most popular choice, Cucumber with the Gherkin declarative language. Other options include Spec Flow (also based on the Gherkin language, but available for .NET development) and JBehave (an extension to JUnit). User acceptance test-driven development (UATDD) achieves the same goal of BDD, but does so by providing a keyword-driven language to specify acceptance criteria to test the correct behavior of code.[9] Open source FitNesse (based on Fit) supports UATDD, while Twist, a commercial tool from ThoughtWorks, supports both BDD and UATDD.[10]

> "We have more than 1,000 Cucumber test scenarios and 2,000 unit tests. Our scenarios are written by product owners in collaboration with business analysts, developers, and testers. Being written in the English-like Gherkin Cucumber language, they are a clear description of business requirements that anyone in the business can understand with no cumbersome implementation details." (Steven Kennedy, program manager, US Department of the Treasury Financial Management Service)

- **Automated regression testing tools speed up release checking and smoke-testing.** To achieve higher levels of automation beyond unit testing, Agile teams pay a lot of attention to verifying each release that they build, usually multiple times a week. For example, for automated web app testing, they make sure that teams use a model-view-controller architecture so that they can define and drive tests at the underlying API and process/activity layers going beyond the graphical user interface. Many of the teams we spoke with mentioned using the open source framework Selenium for automated web app testing. Some teams use it to augment traditional

commercial tools; others use it to replace commercial tools.[11] Other open source tools in this area include Canoo WebTest, Sahi, Tomato, and Watir.

> "We have more than 250,000 functional test cases, 244,000 of which are automated. We have more than 6,000 manual regression tests, but they're not run for every build. We use Selenium because it has great multibrowser support, on top of which we have implemented our own automation framework to simplify browser testing for QA folks. There is also a big push to go underneath the UI for even more automation." (Stephen Reid, VP of software engineering, Ultimate Software)

- **Exploratory testing tools support tests that teams can't automate.** Teams can't automate all functional testing. Business and technically skilled testers use ad hoc exploratory approaches to test edge cases that are either too expensive or too hard to automate. An increasing number of vendors provide support for exploratory testing in their functional testing tools. They allow testers to record their manual testing and then share the full context of their results with developers at a later date. HP, IBM, Micro Focus (Borland), and Microsoft offer varying degrees of support for exploratory testing, as do newer competitors like Atlassian, QASymphony, QMetry, and SmartBear Software.

- **Performance and load-testing tools support projects from the very beginning.** Finding performance issues late in the life cycle of a project makes them harder to fix and can result in major architectural rework (see Figure 4). Teams must set performance goals up front, during test strategy and planning.[12] Performance and load testing might *require* that project teams include specialized testers with performance skills. If the application landscape is too complex, the team might execute end-to-end performance and load testing in a later hardening sprint or might delegate to a parallel specialized testing lab (part of the TCOE).

  Various commercial performance testing tools exist; some vendors are starting to specifically position their tools for Agile load testing by abstracting performance assessment to the application level, where developers can more easily intervene.[13] When no strong performance strategy exists, developers take the easy (and cheap) way out and tend to grab open source tools like Apache JMeter, Benerator, and OpenSTA (which also supports security testing). JUnitPerf provides developers with a framework for instrumenting their unit tests with load performance methods that allow the recording of execution times under varying loads.[14]

- **Service and test virtualization tools enable early and automated integration testing.** Early integration testing is harder in Agile, as teams must deliver working software in early development phases. But the likelihood of dependencies on unfinished components is high in early sprints, which complicates both integration and testing. This is especially true for large projects that have multiple teams working in parallel. But, as IBM's Walker Royce suggests, early integration testing is mandatory for affordable high quality.[15] One way to solve this issue is to

have the development and system test environments provide access to a simulation of the web services, protocols, and systems that your application must interact with in a way that resembles production environments as much as possible.[16]

Commercial service virtualization options include CA Lisa, HP Services, IBM Green Hat, and Parasoft Virtualize. But be warned: These are expensive tools, so you'll need to work with your team to develop a compelling business case.[17] Alternatively, we see some development teams turning to much simpler and lower-cost tools with much less functionality. These alternatives focus on mocking service interfaces. Open source mocking tools are readily available; examples include Cgreen, Mocha, Mockito, and Rhino Mocks.

- **Static and dynamic analysis testing tools improve technical quality.** Given the evolutionary design approach that Agile teams take, it's not uncommon for architecture to take a back seat to refactoring. Static and dynamic analysis tools can help ensure a basic level of security and architectural design by checking for common programming errors that affect software performance, stability, adaptability, and security. These tools scan code and collect useful technical information on cyclomatic complexity, API dependencies, and code inconsistencies; they also instrument code at runtime, making it easier to diagnose defects based on complex runtime interactions. In our research, we found that some development teams also use these tools during retrospectives to identify opportunities for improvement.

  Tools in this space include those from Cast, Coverity, and Klocwork. Another option is adding a collaborative peer code and documentation review tool, like the one offered by SmartBear Software.[18] The teams we spoke with also mentioned a few open source options, including Nikto and Rips (for security testing of malicious back-door code).

- **Security and TDM tools struggle for acceptance.** TDM is an area where developer adoption of tools lags. Few teams recognize that data is the "glue" that holds quality deliveries together and that they need to deal with it early on in testing. Copying data over from production is not an option in many industries; privacy regulations prevent it. Yet teams need the sort of high-quality test data that production systems hold. TDM tools can make processing production data within regulations possible, by providing acceptable masking and transfer of the data on demand.

  Vendors like IBM and Informatica offer TDM products, but they have yet to really hit their mark with Agile teams. One reason: They are designed for use by data professionals and a TCOE instead of Agile developers. There are also some up-and-coming alternatives in this tool category, like those from Camouflage and Grid-Tools , but for the Agile teams we spoke with, it doesn't seem like any one tool has really cracked the nut of the problem yet.[19]

Figure 3 The Landscape Of Testing Tools Is Becoming More Crowded And Dynamic

Testing activity	Description	Recommended Agile criteria	Vendors/tools
Quality/test management	Tools for managing testing strategy, plans, test cases, testing processes, exploratory testing, bug tracking, reporting, etc.	• Supports team collaboration • Incremental planning capabilities • Supports self-managing teams • Supports continuous integration • Change management integration • Supports traceability from user stories to UAT criteria • Integrates into developer IDEs • Integrates with Agile project management tools	**Commercial:** Altassian, HP Quality Center, Microsoft Visual Studio Test Professional, IBM Rational Quality Manager, Oracle, Micro Focus (Borland) Silk Central, Parasoft Test, QMetry, QASymphony qtest, Seapine Software, SmartBear Software, SmarteSoft, ThoughtWorks GO*, Tricentis, Zephyr **Open source:** Tarantula (specific to Agile), Bugzilla Testopia, FitNesse, IST, Limus, QA Manager
Unit testing	Frameworks for developers supporting various programming languages; open source software is almost always the first choice of developers	• Developer IDE integration • Supports test automation in the programming languages and platforms developers are using	**Commercial:** Typemock, Parasoft C/C++ test, AgitarOne, Micro Focus (Borland) Silk **Open source:** xUnit.net, JUnit, NUnit.org, Jmock.org, Cgreen
TDD/BDD/ UATDD	Specification languages for driving development from test cases; good tools enable test design intent to be clear and persistent through iterations and releases	• The tool and/or language helps describe the tester's design intent • The tool and/or language helps perpetuate the developer's design intent	**Commercial:** ThoughtWorks Twist, EggPlant **Open source:** Fit/FitNesse, Cucumber, JBehave, SpecFlow (Gherkin for .NET)

Test automation (functional and regression)	Frameworks or tools for automating functional tests	• The tool helps achieve high high levels of:
		  • API automation
		  • Process/activity automation
		• Provides features to maintain automation
		• Supports efficient and modular test design
			Commercial: HP UFT, IBM Rational Test Workbench, Original Software, TestDriver, Ranorex UI, TAF, Seapine QA Wizard Pro, OpKey, Watcher, Test Complete, Robot, SmartBear Test Complete, Micro Focus/Borland Silk Test, Sahi PRO, SOASTA
			Open source: Selenium, Watir, Sahi, Sauce Labs, Canoo

*Agile project management tools are often stretched for UAT and TM.

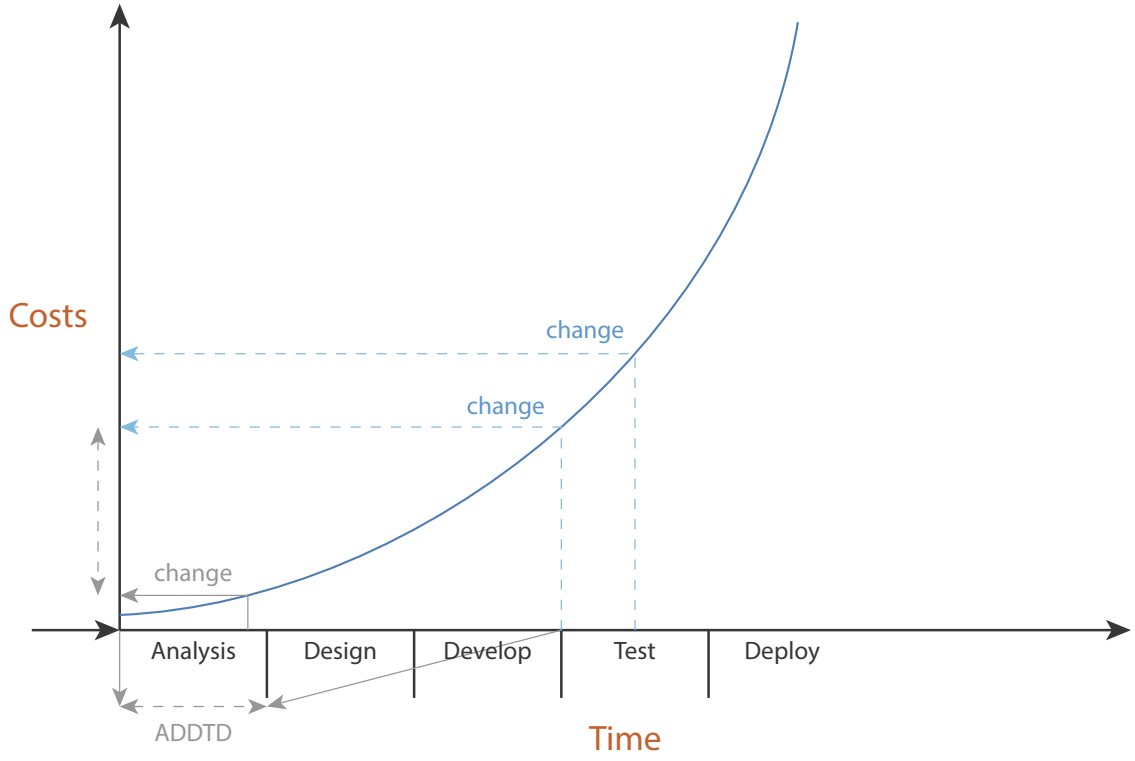
94241 Source: Forrester Research, Inc.

Figure 3 The Landscape Of Testing Tools Is Becoming More Crowded And Dynamic (Cont.)

Testing activity	Description	Recommended Agile criteria	Vendors/tools
Test/service virtualization	Frameworks or tools for automating nonfunctional tests, API testing	• Enables testing even when pieces of the system are not yet built • Protocol support • Supports performance testing	**Commercial:** HP, IBM Test Virtualization (Green Hat), CA Lisa, Parasoft Virtualize, HP Service Virtualization, IBM Rational Test Virtualization, CrossCheck Cloud Port
Load, performance testing	Tools for testing load and performance (web, services, access, etc.)	• Easy to deploy • Integrates with the IDE • Leverages existing automated scripts • Incorporates real-world networking and public infrastructure • Simulates service failures or reduced quality of service	**Commercial:** HP PC & Load Runner, Microsoft, IBM PTS, SOASTA CloudTest, Parasoft Load Test, BlazeMeter, SmartBear Software, Micro Focus (Borland) Silk Performer, Compuware Gomez and dynaTrace, New Relic, MKS, Neotys **Open source:** Benerator, JMeter, DBMonster, WebLOAD, OpenWebLoad, JUnitPerf, Jailer
Test data management	Tools for creating test data loads, data masking, production data sync, etc.	• Could be disruptive • Usually specialized testers	**Commercial:** Camouflage, Grid-Tools, Oracle, IBM Optim, Informatica **Open source:** Data Generator, JMeter
Technical/ code quality, security	Tools for code quality checking, cyclomatic complexity, defect prevention, static and dynamic code analysis, architectural quality (software performance, stability, adaptability)	How the tools can be integrated or adapted for continuous testing/ integration/development/ testing	**Commercial:** CAST, Compuware, Coverity, Klocwork, Lidra, GrammaTech, Vector Software, Mathworks, Ranorex Spy, Microsoft Visual Studio Lint, Parasoft Insure **Open source:** RIPS, Nikto, OpenSTA, FlawFinder
Exploratory testing			**Commercial:** Microsoft, IBM, HP, Micro Focus (Borland), QASymphony, qTrace, Seapine Defect Scribe

94241 Source: Forrester Research, Inc.

Figure 4 Late Testing Means Higher Costs Of Change



94241 Source: Forrester Research, Inc.

Management Tools Must Support Relentless Automation And Continuous Testing

Most of the tools discussed in the previous section focus on test development and execution. But how does Agile change test management tools? From the interviews we've conducted and many client inquiries on Agile testing, we've observed four different approaches that Agile teams employ: 1) ad hoc tools like Excel, wikis, or mind-mapping; 2) Agile project management tools with integration but minimal test management capabilities; 3) new, low-cost, SaaS-based test management tools; and 4) a corporate standard test management tool that integrates into their current application life-cycle management (ALM) solution and supports prescriptive governance and compliance. We also observe a general movement from ad hoc toward more integrated and mature approaches as organizations scale Agile throughout (see Figure 5).

> "We used Excel for a while. We had multiple tabs of requirements; when tabs were not enough, we used different colors in tests. After five or six sprints — when we got to around 1,000 user stories — we decided that we needed a test management tool and went for QASymphony." (David Hardwick, CTO, BetterCloud).

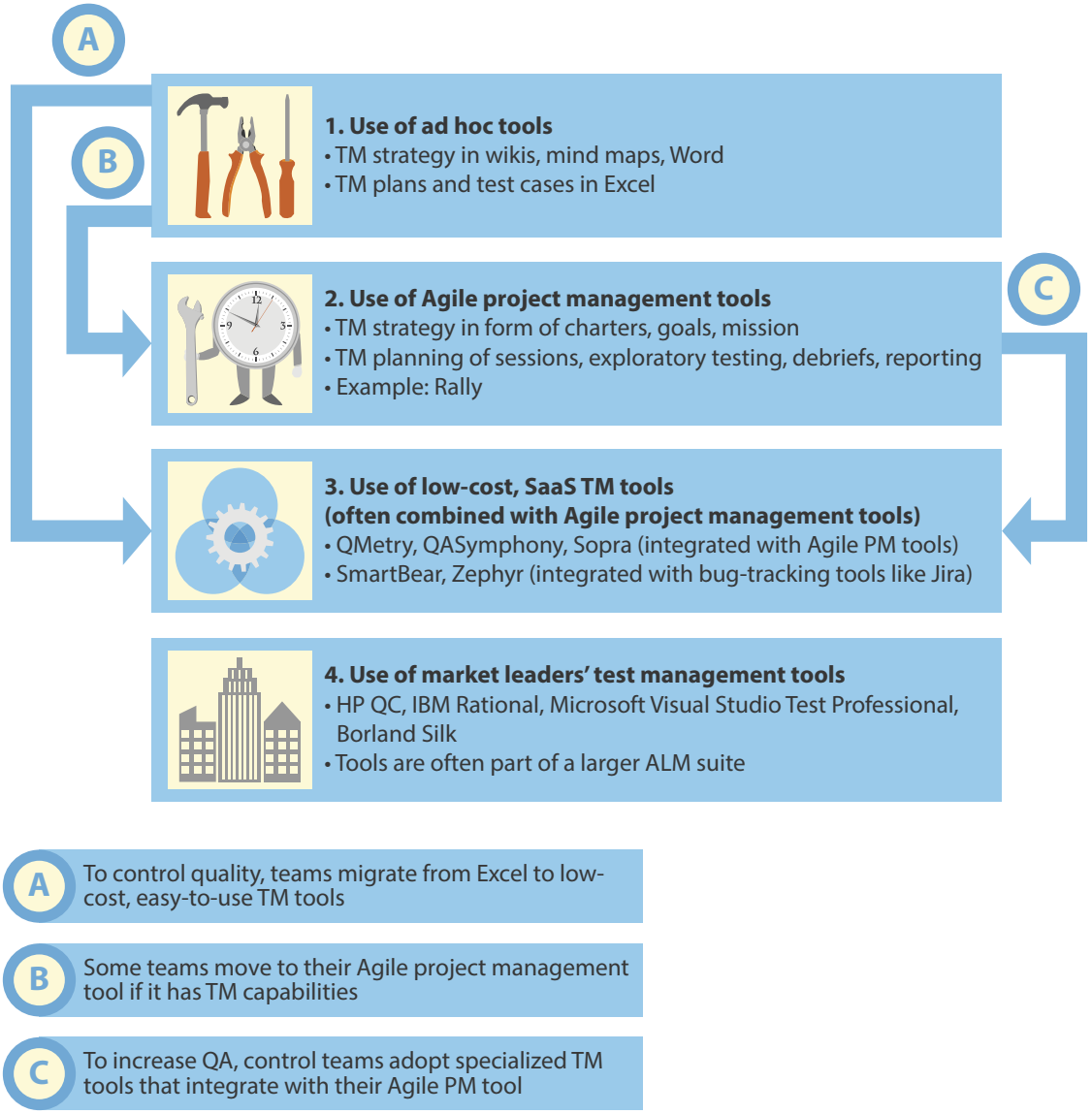
No matter which approach to TM a team employs, tools must support the fast pace of Agile testing. How?

- **Tools must enable a free flow of collaboration.** Agile team members share testing tasks and artifacts at the user story level in the backlog.[20] The backlog governs which testing tasks each team member must complete. There is no fixed testing workflow process, but rather a fast reprioritization of stories and the test cases associated with them. Team members share artifacts; each team member needs to define testing artifacts, execute tests, and have visibility into the overall progress of testing for each sprint and release. Leading ALM vendors like HP, IBM, Micro Focus (Borland), and Microsoft are extending their TM suites by adding their own Agile project management tools or integrating with third-party ones.[21] While there are open source test management tools, we haven't seen much evidence that Agile teams are adopting them yet.[22]

- **TM tools must also integrate into continuous delivery processes.** When Agile teams practice continuous delivery, they declare a build either as qualified for further testing or not qualified (rejected) based on an initial set of "smoke tests."[23] TM tool integration with commonly used build and continuous integration (CI) tools is crucial to facilitating the automation of tests for each build and synchronizing test execution with CI. Integrating TM with CI tools enables the flow of useful data from the TM tool and informs developers when they create a bad build — even if it's compiling correctly. At the same time, passing test results from the software build and integration process informs testers that a build is ready for more extensive automated and manual testing.

- **Developers need to know why a test fails so they can debug and fix problems.** On Agile teams, developers try to fix the defects as soon as they're caught. TM tools need to facilitate rapid debugging by integrating their test results analysis workflow with developer debugging tools so teams can quickly identify and isolate problems. Integration between the TM tool, Agile project management, and bug-tracking tools makes it easier for developers to look in one place to get context for their next development task, whether it's a bug to fix or a new user story from the sprint backlog. Most TM tools integrate with developer favorites like Atlassian JIRA or the free open source Bugzilla. Microsoft offers .NET developers a real-time solution for bug-squashing with IntelliTrace, which allows for recording and tracing the execution histories of production code and recreating the production trace in development to help developers find bugs without setting lots of break points in code.

Figure 5 Four Approaches To Using TM Tools And Three Common Migration Paths



94241 Source: Forrester Research, Inc.

RECOMMENDATIONS

EVALUATE AGILE TESTING TOOLS WITH THE FIVE MUST-HAVES IN MIND

Evaluating testing tools to support your Agile transformation? Assessing the tools you already own? Look for the following capabilities when selecting which ones you'll take forward:

- **Collaboration.** Agile testing tools must support seamless collaboration among Agile team members. Social features like wikis, chat, IM, and discussion streams are table stakes; more important is programmatic interoperability via APIs and open data formats. Example of effective Agile collaboration include being able to easily share artifacts and tasks, like a user story from the sprint backlog, a burn-down chart to see development and testing progress, and quick, universal access to testing specific reports. If test management tools aren't open and interoperable, via a repository, APIs, or other mechanisms, it will seriously impede the flow of changes through your deployment pipeline.

- **A simple user experience and a short learning curve.** Ease of use matters when team members are juggling multiple tasks, including development, testing, and deployment. How easy is a team member's journey as they go through daily testing tasks? How intuitive is the user interface overall, and especially the organization of test plans, test suites, and test case execution? How long does it take team members to learn how to use the test management tools? Are one or two weeks enough, or does it take more? Look for test management tools that organize and store test cases using folders as a simple structure, or that map to user stories and scenarios.

- **Improved levels of automation.** Unless you achieve a high level of test automation, your Agile transformation will sputter, held back by repetitive manual testing. So ask yourself the following questions: Do the tools you are evaluating enable you to create modular, reusable, and well-designed test automation code? Do they support defining and reusing architecture patterns and practices? How do the tools support an increasing level of automation and how will they help maintain large automation test suites as your test cases grow into the thousands and beyond? Does the test automation tool provide features for visualizing redundant test cases? Can test cases be abstracted and reused in multiple places? Do the tools also help automate some of the manual steps of your process and capture successful execution of manual test runs? Can you easily integrate the results of mixed manual and automated testing?

- **Continuity and accuracy of information flow.** In Agile, testing is a continuous process from start to finish, within sprints and within releases. The above characteristics of automation and improved collaboration are key drivers for continuous testing. However, there also needs to be constant synchronization between the artifacts across the different tools, status of activities, and data sharing. Accordingly, all of these activities should revolve

around common artifacts like a change management request and a unified task management system. Anytime a team member has to stop and manually cross-reference information or import artifacts from one tool to another, you've impeded the flow of status and created an opening for the subjective measurement of progress.

- **Instant accessibility.** Agile teams don't have time to engage in long sales cycles to get hold of new tools. They also tend to want to demonstrate value before investing in commercial tools. Make it easy for teams to get access to tools they need, especially during early sprints. Expensive tools with per-user licenses can be a problem at scale, because testing can easily shift from team member to team member as sprints progress. A flexible licensing policy (or an entry-level open source product) can lead to broader adoption. In some cases, you may need to differentiate between team members who have change rights on artifacts versus those who only need read access. Be wary if the vendor recommends an extensive proof of concept to justify purchase — you may get strong pushback from your Agile teams. For specialized testing tools, such as test data management, test virtualization, and performance and load testing, you'll need to assess how long it will take to install and configure them in your preproduction environments. Other factors to consider include how much training is required and whether the tools are offered as SaaS services that you can quickly deploy when needed.

WHAT IT MEANS

ADAPT YOUR TOOL STRATEGY TO SUPPORT TESTING WITHIN AGILE TEAMS

Whether you're inheriting an existing testing tool strategy from past development approaches like waterfall or are just firing up your Agile testing practices and have to build a tool strategy, consider the following:

- **Your tool choice will vary with your level of Agile adoption.** The more your business peers demand rapid software innovation, the more you'll find that a standalone TCOE impedes the flow of rapid change. As you adopt Agile processes, expect an increased need for collaboration, simplicity, automation, continuity, and accessibility to drive development teams toward tools that fulfill these needs.

- **Improved quality starts with effective test management, even for Agile.** Small to medium-size application development and delivery organizations (less than 100 people) often start with no test management tools, thinking that ad hoc tools like Excel will suffice to manage test plans and test cases. If you are in this situation, we recommend adopting a test management tool sooner rather than later, as it will improve your overall quality management process. If you're in a large company with a corporate standard TM tool, make sure that it enables the practices described in this research before just adopting it. If not, look for alternatives that are better suited to Agile development.

- **A range of testing tools are still necessary for Agile development.** Performance, load, security, and test data management are just as important for Agile projects as waterfall or iterative projects. But like functional testing tools, they have to be part of the entire development cycle and capable of frequent use from the very beginning of a project. Designating certain sprints as "hardening sprints" can work in the short run, but easy, frequent test execution is a must-have for a long-term approach.

- **Tools matter, but how you use them matters more.** Tools make a difference, but how you use them will ultimately determine your level of success. Learn about and apply the new Agile testing practices mentioned in this report first, and then determine what support you need from a tool to execute practices effectively or scale them to multiple Agile teams. Where possible, experiment first with open source and then decide if the investment in a commercial tool is warranted. Make sure that the entire team is on board with your selection or risk seeing substitute open source software tools popping up as alternatives.

SUPPLEMENTAL MATERIAL

Companies Interviewed For This Report

BetterCloud

CA Technologies

FINRA

HomeAway

HP

IBM

MicroFocus (Borland)

Microsoft

New Relic

NTT Data

QASymphony

QMetry

Seapine

Silverpop

SmartBear Software

TechTalk

ThoughtWorks

Ultimate Software

US Department of the Treasury Financial Management Service

ENDNOTES

[1] Agile disrupts everything that developers have learned about testing. From practices to skills to organizational models, companies need to act now to move to the forefront of Agile development. See the January 17, 2013, "Consistent Performance In Agile Teams Must Include Testing" report.

[2] BDD focuses explicitly on testing code behavior. QA professionals and/or business analysts write test scenarios as declarative English-language statements that everybody on the team understands. Developers then write automation code for each scenario.

[3] Cucumber is an open source tool that is becoming quite popular; it enables an extended test-driven development approach called behavior-driven development.

[4] Mobile app development is part of a larger structural change in the way we build applications. We're entering a new age of application development that creates modern, compelling systems of engagement and links them with systems of record and systems of operation. See the January, 17, 2013, "The Future Of Mobile Application Development" report.

[5] Jez Humble and Dave Farley describe one trick that shops use to reduce their risk of a bad deployment in their book, Continuous Delivery. Shops alternate deployment to two different but highly similar production environments (one "blue," the other "green"). See the February 7, 2011, "Five Ways To Streamline Release Management" report.

[6] Clients mix various Agile methods with more traditional waterfall and iterative methodologies. See the April 30, 2012, "Survey Results: How Agile Is Your Organization?" report.

Forrester defines this approach as "Water-Scrum-Fall." See the July 26, 2011, "Water-Scrum-Fall Is The Reality Of Agile For Most Organizations Today" report.

[7] Mobile development presents a completely different set of challenges than any of the enterprise development projects you've delivered in the past. These challenges aren't limited to your development teams. They will permeate all aspects of your software development life cycle, enterprise architecture, and the methodologies you use to develop and deliver mobile applications. See the May 28, 2013, "Mobile Development — Smooth Sailing Or Titanic Voyage?" report.

[8] Forrester has previously reported on how the role of testers and the overall testing organization changes. See the January 17, 2013, "Consistent Performance In Agile Teams Must Include Testing" report. For more on how testing has to change to mature toward Agile testing, see the January 15, 2013, "Forrester's Agile Testing Maturity Assessment Tool" report.

[9] SpecFlow was developed by Austrian vendor TechTalk.

[10] Fit is a tool for enhancing collaboration in software development. FitNesse is a wiki-based integrated development environment for Fit. Twist specifically provides some interesting features to improve maintenance of test cases and code automation.

[11] Commercial test automation tools are: HP Integrated Functional Tester (market leader), Microsoft Visual Studio Test Professional, Micro Focus/Borland Silk Test, IBM Rational Functional Tester, Worksoft (SAP), Ranorex UI, Seapine Software QA Wizard Pro, and SmartBear Software TestComplete. Most of these tools have been on the market for quite some time.

[12] A performance baseline is established during performance testing. Then maximum loads are then specified along the baseline and verified during all regression testing to make sure performance goals are met under the different loads.

[13] The commercial players are HP, the market leader with LoadRunner, IBM with Rational Performance Tester, Microsoft with Visual Studio Ultimate, Borland Silk Performer, and Parasoft Load & Performance testing. Compuware is positioning dynaTrace as a performance testing tool that is more natural for developers to use, providing a higher level of abstraction than more traditional performance-testing tools. Other solutions are New Relic with more of a monitoring bent, or cloud performance testing tools like Neotys with NeoLoad and Cloud Test from SOASTA.

[14] Other open source software tools used by clients include Jailer and Webload.

[15] Walker Royce from IBM demonstrates how integration testing should precede unit testing. Source: Walker Royce, "Measuring Agility and Architectural Integrity," ISCAS, 2011 (http://walkerroyce.com/PDF/Measuring_Agility.pdf).

[16] Duplicating the production environment in the testing environment is becoming more and more cost-prohibitive as production environments become more complex, leading to the need for simulation. Service virtualization tools enable other nonfunctional testing besides integration testing: performance testing against a simulated connection to a mainframe or online service avoiding costs, load-testing simulating connections and scaling up and down, and more. Service virtualization tools enable various aspects of nonfunctional testing besides integration like performance and load testing against a simulated connection to a mainframe or expensive online services. In a test virtualization environment, you can simulate upscaling and downscaling, increasing and decreasing loads, and more.

[17] The business case will leverage the saved costs of early integration versus late extended testing, lower costs for a virtual production environment, lower costs for virtual access instead of expensive real online or mainframe services access, higher precision of web services versus stubs/mockups, and automation versus error-prone manual. The business case is usually favorable in large complex application and middleware integration environments and decent-size Agile transformation programs.

[18] The usual suspects here are the traditional corporate players like HP, IBM, Compuware, Microsoft, and Micro Focus/Borland.

[19] For more information, please check out Grid-Tools. Some vendors offer TDM capabilities in their service virtualization tools — e.g., CA Lisa.

[20] Acceptance criteria, testing charters, test scenarios, automated tests, test outcome, unit tests, test coverage, and others get associated to user stories in backlogs.

[21] Microsoft has built its TM tool with exploratory testing in mind and collaboration is enabled through the TSF integration. HP, IBM, and Micro Focus/Borland started from existing pre-Agile TM tools based on the TCOE model.

[22] Tarantola is the only open source TM tool that our clients mentioned during the interviews.

[23] Source: Martin Fowler, "Continuous Integration," ThoughtWorks, May 1, 2006 (http://martinfowler.com/articles/continuousIntegration.html).

FORRESTER®

About Forrester

A global research and advisory firm, Forrester inspires leaders, informs better decisions, and helps the world's top companies turn the complexity of change into business advantage. Our research-based insight and objective advice enable IT professionals to lead more successfully within IT and extend their impact beyond the traditional IT organization. Tailored to your individual role, our resources allow you to focus on important business issues — margin, speed, growth — first, technology second.

FOR MORE INFORMATION

To find out how Forrester Research can help you be successful every day, please contact the office nearest you, or visit us at www.forrester.com. For a complete list of worldwide locations, visit www.forrester.com/about.

CLIENT SUPPORT

For information on hard-copy or electronic reprints, please contact Client Support at +1 866.367.7378, +1 617.613.5730, or clientsupport@forrester.com. We offer quantity discounts and special pricing for academic and nonprofit institutions.

Forrester Focuses On Application Development & Delivery Professionals

Responsible for leading the development and delivery of applications that support your company's business strategies, you also choose technology and architecture while managing people, skills, practices, and organization to maximize value. Forrester's subject-matter expertise and deep understanding of your role will help you create forward-thinking strategies; weigh opportunity against risk; justify decisions; and optimize your individual, team, and corporate performance.

« **ANDREA DAVIES,** client persona representing Application Development & Delivery Professionals