

Top 10 Takeaways from the TISQA Conference: Agile Testing in the Carolinas

*The message at the heart of the 2009 TISQA (Triangle Information Systems Quality Association) conference, “Agile Testing in the Carolinas,” is that when organizations leverage Agile development methodologies, **everyone** becomes responsible for quality software development. Together with major corporations such as IBM, SAS, and BCBSNC; Paragon sent representatives from Development, Support, Professional Services, and QA to the conference and each attendee came back with renewed interest in Agile testing. This article summarizes the top ten ideas gleaned from this year’s TISQA conference.*

Agile software development methodologies have been adopted by a variety of software development teams, including those at some major financial institutions. In 2005, Paragon development teams began by using Agile development methodologies such as pair programming and Test Driven Development (TDD), and now use the scrum process to manage all development efforts (including the associated documentation and testing). Together with major corporations such as IBM, SAS, and BCBSNC; Paragon sent representatives from Development, Support, Professional Services, and QA to the [TISQA](#) (Triangle Information Systems Quality Association) conference, “Agile Testing in the Carolinas.”

The conference offered presentations from Agilists such as Lisa Crispin and Janet Gregory, co-authors of “Agile Testing: A Practical Guide for Testers and Agile Teams” (Addison-Wesley, 2009); Michael Bolton, co-author (with senior author James Bach) of “Rapid Software Testing”, and others. Agile veterans at Paragon, as well as those wanting to learn more about Agile development, benefitted not only from the presentations, but from exchanging ideas and experiences with other Agile practitioners in the Research Triangle area. Paragon attendees sought information that might be of use not only to themselves but to the developers and testers at the various financial organizations that Paragon serves. This article summarizes the top ten ideas gleaned from this year’s TISQA conference by Paragon software engineers Lavanya Reddy, Patrick Quentmeyer, and Tom Walker; and QA analyst, Paula Grange.

1) Enlist the skills of *everyone* on the team, and be prepared to learn new skills

Grange: Agile Testing is a combination of developers, testers, information developers, customers, project managers, and support specialists all collaborating and building **what the customer wants**. We are *all on the same team* with software quality and customer satisfaction as our primary goals. My job, when I’m testing, is not to determine if the product is ready for release, but to give management information about the functionality and usability of the product so *they* can make informed decisions about its release.

Reddy: For Agile team members, a greater range of skills is required. It isn't enough to be just a tester, or just a programmer, or just an analyst, or just a...whatever. Agilists are moving toward a highly iterative and

collaborative approach which requires expanding and generalizing the roles of the specialists typically involved in producing software. Roles and responsibilities overlap and all team members need to be prepared to make meaningful contributions.

2) Accomplish big goals using small teams that concentrate on small stories

Reddy: Smaller teams foster faster feedback, better communication, and enable rapid response to change. Similarly, discrete, well-defined requirements (such as features and user stories) are much easier to estimate and build than larger requirements. They're easier to prioritize and, therefore, to manage.

Quentmeyer: User stories should be small enough to complete in one-half to 4 days. If a story takes longer than four days to complete, it should be divided into smaller stories.

3) Know your user and your user's needs (then get user feedback to validate what you think you know)

Reddy: A user story is a short description of functionality told from a user's perspective. It is a placeholder that is used later to flesh out details. User stories are the basis of all Agile requirements and involve three C's: a *card* on the white board that describes the story, *conversation* with the stake holder and team members to ensure a solid understanding of the requirement, and *confirmation* of that understanding by the stakeholder before development begins.

Grange: User stories are the tool that focuses everyone's attention on what the user needs and what has to be developed to meet those needs.

Quentmeyer: Each story applies to only one user, and those users may vary. For example, other parts of the system may be considered a user of a story. The primary goal of each story is a meaningful purpose. We should ask, "What is it about this function that gives *value* to the customer?" You must be able to define any story with the statement, "As a <user>, I want to <action> so that <business value>."

Walker: One of the primary advantages of using Agile methodologies is that, at the end of every two-week sprint, we get to produce something that we can send to a user and verify "Is this what you wanted?" That is unlike other development methods in which you may find out at the end of the project that the description you *thought* you heard from the user is not actually what they wanted or needed.

4) Make "testing" part of software development, not a separate activity

Reddy: Agile development uses a team in which testers are "embedded" into development and actively participate in all aspects of the project, rather than testing only after the coding is complete.

Walker: Unit testing becomes a *coding* practice, not a *testing* practice.

Quentmeyer: The tester must be an active and contributive participant in the development process that brings a testing mindset to the team, not an individual on the team that tests the product. Testing is not proofreading. It

is active involvement throughout the development process: defining acceptance tests, helping developers with unit tests, defining what the product is NOT supposed to do, and so forth.

5) Avoid scrummerfall

Quentmeyer: Agilists need to beware of moving from a true sprint to a 2-week waterfall process. (This unproductive combination of scrum and waterfall development is sometimes referred to as *scrummerfall*.) All team members (testers, developers, user assistance developers) should be invested all the time – there is no linear movement from “phases” of coding, testing, and documentation during a sprint. There are no scheduled code freezes or build dates that drive the schedule. Sprint end dates are firm. Unfinished work may be rolled into another sprint, but sprints are not extended so that work can be completed. During each two-week sprint, you *all* work to develop working, tested, well-documented software – and at the end of the sprint, you’re *all* done.

6) Automate regression testing to enable it to keep pace with product development

Quentmeyer: Manual regression testing simply cannot keep pace with development enhancements. As the number of enhancements increases, the number of required regression tests also rises, and more and more time is required to complete the testing manually. Instead, automate acceptance test cases as they are developed and incrementally build a regression test suite. By automating regression testing, you’ll have more time for manual exploratory testing that will root out issues that routine regression testing may not uncover.

7) Focus on “done” and define, up front, what “done” means

Quentmeyer: *Tasks* are not done; *stories* are done, and the story is done only when it passes the acceptance criteria. Testers are crucial to developing the acceptance test (or demo) for a story. Every member of the team who worked on the story is responsible for saying it is done.

8) Bugs: find ‘em and fix ‘em – as part of the software development process

Quentmeyer: Bugs in production are never a good thing, and finding the *source* of a bug in production nearly always takes longer than implementing the fix. If we focus on test driven development, one key benefit is that every time a unit test fails, we have identified a bug before it goes out the door. Fixing that defect immediately is much faster than finding and fixing it in production, because during development you know exactly what code was changed and where that code resides.

Walker: Defect tracking during development can be valuable when you have a distributed development team (particularly when they are in different time zones), but may not be the most efficient way of routinely handling bug fixes during development. A central concept in Agile development is to value interaction over process, and defect tracking is a process. If it’s easier for someone on the team to say, “Hey, I’m having a problem with...” and another team member (who knows what code he’s been working on and what the likely source of the problem is) can fix it in five minutes, then it can literally take longer to log and track a defect than to fix it. The bugs are

addressed earlier (because they are not waiting in a queue), and the team continues to make progress toward their goal of quickly producing working software.

9) Don't rely on statistics to effectively measure software quality

Walker: Statistics may not be a very reliable indicator of software quality or team efficiency. For example, neither the number of defects *found*, nor the number of defects *fixed* by any team member give us very helpful information. If we find and fix 20 defects, we don't know that we've found *all* of the defects in that program—in fact, we don't even know what percentage of the defects we've found, therefore the number really means very little. You can't even use the number of unit tests you've created as an indicator of how well the code has been tested—you can only use them as a measure of the *percentage* of code tested. Even knowing the unit tests exist doesn't necessarily imply that the unit tests will reveal all possible defects.

Quentmeyer: Velocity is another statistic that we need to be careful to use correctly. Velocity is simply a measure of the team's bandwidth (or capacity to produce working software); it's not a tool to track accomplishments. The intent of tracking velocity is to provide a framework for the stories that the team commits to completing each sprint. A traditional indicator of progress in waterfall development might be measuring how many code modules you delivered, with no consideration of whether or not you could actually create a working build. In contrast, Agile development helps the team focus on producing *working software every two weeks*, rather than merely tracking an arbitrary number of tasks completed or modules changed.

10) Embrace change

Being Agile means embracing change: changes in roles, changes in teams, and changes in the development process. Judging by the interest and enthusiasm in Agile development methodologies expressed by Paragon's TISQA conference attendees, change is good.

About TISQA

TISQA (Triangle Information Systems Quality Association) is a non-profit organization dedicated to software quality, based in the Research Triangle Park area (near Raleigh NC) for over 15 years. TISQA promotes and encourages the improvement of information systems quality practices and principles through networking, training, and professional development opportunities. TISQA is a federation chapter of the Quality Assurance Institute (QAI). More information about TISQA can be found on their website: <http://www.tisqa.org>.

About Paragon Application Systems

Paragon Application Systems is a leading global provider of ePayment simulation, configuration and testing software tools to the financial industry. More than 400 financial institutions in over 80 countries use Paragon tools to improve quality and reduce time-to-market. Paragon's broad customer base includes major interchanges, processors, leading software providers, banks and credit unions. Visit Paragon Application Systems at www.paragonedge.com.