WHITE PAPER

5 MUST-DO'S FOR SOFTWARE QUALITY WITH SPEED

Overview

In March 2015, Magenic commissioned Forrester Research to evaluate the current state of application development approaches, as well as the challenges, opportunities, and importance of quality in Agile development. In the report on that evaluation ("Leverage Agile to Meet Customer Demands and Software Quality Needs"), a recommendation is made to "Apply 5 must-do's to get your testing in order." These must-do's have been further broken down and analyzed to show how to best implement them into your organization.

Andy Tinkham
Magenic QAT Practice Lead

Magenic® // Software to move forward.™

Fundamentally, testing focuses on providing information to the team and stakeholders so that better decisions can be made. For many test teams, quality information is provided in the form of defect reports and count-based metrics (X number of test cases, Y number of defects, and so on). For these teams, test planning involves the (time-consuming) creation of detailed documentation. As development and deployment accelerates, these approaches no longer fit with the goals of the organization and the team's need to have information to make smart, rapid decisions. Test teams may begin to become increasingly marginalized as the broader dev team strives to work-around the perceived roadblocks caused by testing, possibly going so far as to try to completely replace human testing by creating a suite of automated scripts.

In March 2015, Magenic commissioned Forrester Research to evaluate the current state of application development approaches, as well as the challenges, opportunities, and importance of quality in Agile development. In the report on that evaluation ("Leverage Agile to Meet Customer Demands and Software Quality Needs"), a recommendation is made to "Apply 5 must-do's to get your testing in order." (p. 10). The "must-do's" identified refer back to another Forrester paper ("Five Must-Do's For Testing Quality at Speed", written by Diego Lo Giudice in January 2015). These must-do's are:

- Organize testing in a lean way
- Use shift-left and shift-right testing
- Build a practice for testing skills
- Reduce manual testing in favor of automation
- Automate test data and environments provisioning

The testing industry is changing – at this point, we have no doubt about this. We do not see this as the death of testing – far from it! – but the testing role and the activities around testing are evolving. Here are some ways you can bring Forrester's Must-Do's into your organization.

Target testing activities to provide the most value

One of the main principles of lean development is to maximize the value created in a process through the reduction of wasteful activities that consume time and resources but do not provide a corresponding increase in value. Testing is a key area where waste can be found. Test suites grow from release to release (and build to build) without the individual tests being evaluated to determine if the information that test could provide is of value to the team and uniquely provided by that test. Multiple tests cover the same functionality. A too-common example of this is testing valid logins as separate tests in a regression suite. Almost definitely, the regression test includes tests that require logging into the application under test as a precursor action to get to the functionality being tested. It is unlikely that a basic valid login test is going to detect a failure that a test using that same login on the way to something else is going to miss, and yet teams will keep that basic login test, taking up time that the tester could have used for more valuable actions. By selecting activities in each cycle that best provide the information that the team most needs at that point to understand the current state of the application and the implications of releasing in that state, wasteful actions can be removed from the process, streamlining testing and keeping the team moving forward at a rapid pace.

Utilize risk-based test prioritization

When prioritizing tests by risk, the team looks at several risk sources: the likelihood and impact of functionality failing in production, the costs of the team not having information when it is needed (including rework and switching costs when tasks need to be revisited), and the opportunity costs of not running other tasks (since testing time is always finite and doing one task may mean that another no longer fits in the available time).

While planning tasks to address the greatest risks first focuses the test team and adds significant value to the process, it is equally important to call out the tasks which were considered but subsequently rejected. These tasks may have been rejected as not a high enough risk level to merit the action or may simply not have fit in the allotted time due the presence of higher risk tasks. Reviewing them with the team and stakeholders may bring new perspectives that highlight otherwise missed risks or make a case for additional time and resources to be allocated to the testing effort. Having these conversations early in an iteration also ensures that the entire team is aligned in their thinking of what will and what will not be tested.

"While planning tasks to address the greatest risks first focuses the test team and adds significant value to the process, it is equally important to call out the tasks which were considered but subsequently rejected."

Cross-team fertilization of ideas

Too often, teams become isolated both internally and externally. Teams may form "silos" based on role, or only interact with members of their own team. This can lead to a stagnation of ideas – a lack of new ideas coming in means that the existing ideas already in place never get challenged. As a result, practices may no longer align with the goals and values of the team, but yet remain due to long-established tradition. To combat this problem, facilitate interactions with people outside the team. This can be encouraging members of different teams in the same organization to regularly talk about what their team is doing, establishing community user groups to bring people from different organizations together to talk about their challenges and solutions, bringing in outside experts to speak or assist with projects, or sending team members to conferences and training. All of these approaches bring new ideas to a team, providing a crucial stream of information the team can use as they continue to evaluate value, determine their actions, and have retrospectives to review what worked well and what can be improved.

Include testing tasks in estimates and definition of "done"

A common mistake made by teams is to estimate new features based solely on the development work needed to create the code. Iteration planning then is based solely on what the developers can create in the iteration. Testing is either pushed to the next iteration (on the assumption that developers will work right to the iteration end to complete features) or worse left as unpaid technical debt. Developers want to track the work they've accomplished in the iteration the work was done, and so the team defines "done" for their stories at the point when the developer checks in their initial work. The technical debt this model accrues can build very quickly. Some teams achieve success with lag-a-sprint testing, but at the cost of increased switching costs and potential declines in quality when the testing backlog gets so big that testing is skipped. Even on teams that manage their technical debt carefully, planning for features to be ready to release is much more difficult in this model as features have an unknown amount of work remaining when they're marked as "done" – there could be little work if few bugs are found or significant work if major issues are subsequently discovered.

A better way is to bring testers to the front of the process. As features are initially conceived, business analysts, testers, and developers should work together to think about how the functionality should work. These meetings may result in a set of examples, perhaps in the form of an executable spec or they may just result in notes on a whiteboard; the meeting isn't about creating an artifact, it's about building a shared understanding amongst the team members. Then, the team can use this shared understanding as a basis for estimating the full set of tasks needed to complete the story (including development, testing, and any other tasks required), and plan accordingly. By not calling a task done until all these tasks are complete, teams can have a better handle on when the feature is actually shippable instead of having to wait, viewing testers as the bottleneck keeping the team from releasing.

Use automation as a support tool

Automation provides a way to speed up certain activities and has long been a strong draw for testing teams. However, automation does nothing to improve the quality of the testing – if bad tests are automated, the result is generally faster bad testing. Automation also loses bug finding power over a skilled manual tester executing the test. An automated script looks only at the pieces it's been specifically told to consider while a human can recognize unexpected "odd" behavior and immediately follow up on it (or make a note to come back and investigate the behavior further later.

Balancing these drawbacks, however, are the benefits of automation. Many manual testers are treated like robots, instructed to run the same test scripts over and over as each new build is available. In the extreme cases, these testers may be penalized for deviating from their assigned test scripts, but even in less draconian environments, incentives and measurement tend to focus on the easier-to-track aspects of testing (like the number of scripts run) and less on the more nebulous things (like issues found because something looked off). Tasks like this are well suited to being automated; if we can define the test well enough that humans don't need to invest a lot of thought or skill to run the test consistently, we can usually tell a computer to do the same actions. By creating automation to do those tasks, we get two benefits. First, we free up our manual testers and can then use them in ways that take advantage of their ability to think and their skills in testing. This frequently results in increased bug finding, and can lead to better bugs being found prior to release.

"If we can define the test well enough that humans don't need to invest a lot of thought or skill to run the test consistently, we can usually tell a computer to do the same actions."

The second benefit comes from the tighter feedback loop to development as changes are made. Because the tests run more quickly, and can be triggered by anyone on the team, developers can use automated scripts immediately while in the process of making the change, avoiding switching costs they would otherwise incur if they had to wait for someone else to evaluate the impact of the changes. Fast, easily run automation allows developers to get feedback repeatedly as they make changes. These scripts are most frequently unit tests, but that doesn't have to be the case – scripts can execute tests with other focuses or focus on other aspects beyond test execution such as data creation, environment setup, or results analysis.

Production monitoring and testing

No matter how effectively an organization uses its testers and how quickly the team can get feedback, there will still be unexpected events and issues found in production. Complicating matters, better testing prior to production means the issues that do occur are harder to reproduce, and take more analysis to understand. Organizations are dealing with this in two main ways: increasing monitoring in production (to provide better visibility into what led to the unexpected events) and running some portion of their testing against live production systems. Forrester refers to these activities as "shift-right testing". Both techniques can be very powerful when used judiciously. It is possible to overuse them – too many data points to analyze or tests that are destructive in nature or otherwise directly impact real customers can create more problems than they solve – but as modern application development continues its acceleration, these techniques will prove to be more and more essential to getting the team the information they need to maintain their velocity.

It is possible to have rapid development and thorough testing. Reducing waste keeps your testing lean. Bringing testing in earlier in the process and extending testing activities past release through strategic testing and monitoring in production gives you more visibility into the quality of your application. Seeding your team with new ideas keeps teams from getting stuck in a rut of doing things the same way they've always been done, and automation supports your team both through taking the easily repeated tasks off your manual testers to allow them to tackle the deeper challenges and by moving beyond test execution to support data generation and environment setup. While every organization has its differences, we believe the tips here will help many organizations achieve these goals. If you'd like more information or want to tailor these tips specifically to your organization, Magenic would love to talk with you! Our expert testing consultants can help you to get the information you need on time and on pace with your development team.

About Magenic

Founded in 1995 by the same technical minds that still run the company, Magenic delivers software results on the most complex modern application projects.

Visit us at **magenic.com** or call us at **877.277.1044** to learn more or to engage Magenic today.

Magenic® // *Software to move forward.™*