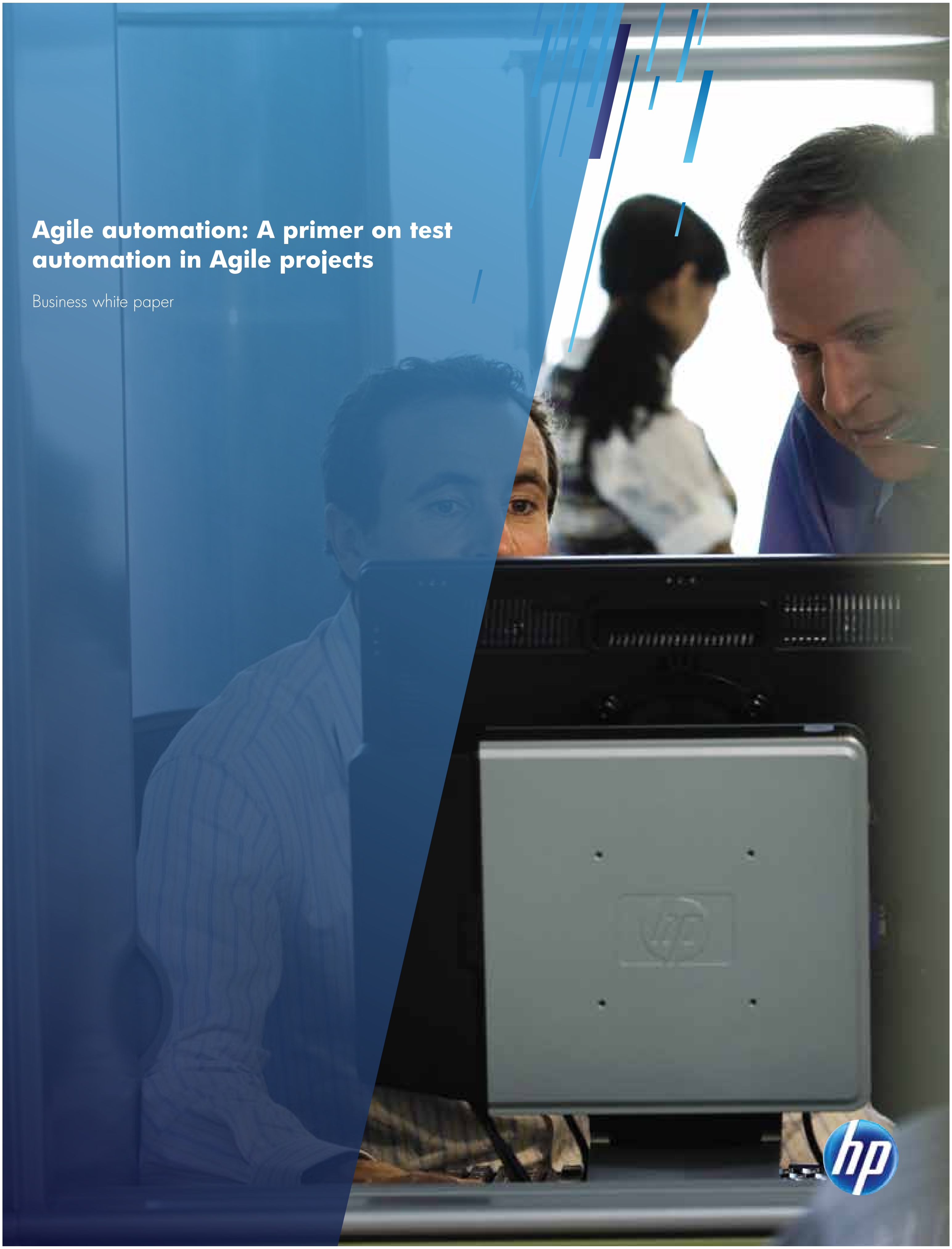
Agile automation: A primer on test automation in Agile projects

Business white paper



Here's a methodology riddle: What is the one aim of Agile, without which a project becomes little better off than it would've been with traditional Waterfall?

The early discovery of issues

Where sequential methods leave application validation to a mass effort at the end stages of the project, Agile concentrates teams on developing potentially releasable code for each sprint. This means that sprint by sprint, the work of the project team is put through a proper quality vetting, with the aim of uncovering problems and changes early enough to be addressable without the usual last-minute general panic.

So crucial is this aim that most Agile teams understand quality to be the responsibility not of a specific individual or department, but of the entire team. Indeed, this all-hands quality concept is important, because properly validating an application as it develops poses a unique set of challenges. Foremost among these is the role and use of automated testing.

Note: This paper is first in a series on effective testing in Agile projects.

Can't live with it, can't live without it

Ask a QA Director whose team has recently engaged in Agile projects how test automation is being leveraged, and you're likely to see her close her eyes and wish herself somewhere else. It's an understandable response. Since automated testing requires an investment in designing and developing the framework and scripts, the payout is usually highest when applied to mature (read: stable) applications. Under Waterfall methods, the majority of the application is "baked" before arriving in the tester's hands. But Agile projects, with their embrace of regular change and ever-evolving features, are an automated tester's nightmare.

And yet, the hard truth is that without automated testing, Agile delivery isn't possible. Unlike unit testing, which focuses almost exclusively on the functionality within the current sprint, the challenges of system tests expand with each sprint. System testing must encompass not only the new functionality of the current sprint, but also features from previous sprints—as well as regression testing any legacy functionality. Automated testing is the only way to ensure complete coverage of both legacy and new functionality, performance and security, all in sprint-time.

Put another way, Agile without test automation is essentially Waterfall by another name. Developers may create code and run unit tests on a regular sprint cadence, but proper system validation is deferred to later and later phases; there simply isn't time to manually test the accumulating functionality from sprint to sprint. Inevitably the team encounters surprise problems late in the game, and must incur budget and/or deadline overruns to address them. Projects working in this mode may give off the appearance of speed—but only until just before the end.

Where to start

First things first: incorporating test automation requires that teams have some standing criteria for deciding what is to be automated. No project, whatever its methodology, will automate every last test. Manual testing will always play a vital and necessary role as well.

In general, test automation makes sense for:

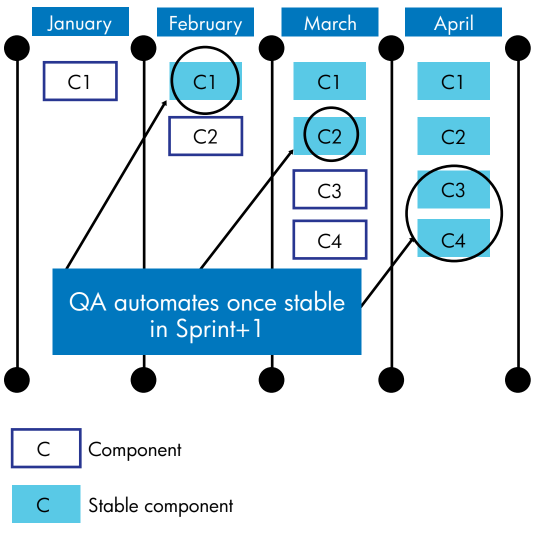
- functionality which is critical to the business or supports many concurrent users;
- functionality that has been assigned specific service-level agreements (SLA);
- functionality which touches multiple areas of the application (or other applications);
- tasks which are either repetitive or error prone, such as data loading, system configuration, etc;
- testing that involves multiple hardware or software configurations.

Establishing the test automation criteria upfront helps to the team to make consistent, pragmatic decisions about automation, and will reduce the cycles and decision analysis which are unfavorable to Agile's sprint-time.

Speed and (not or) quality

As a general rule of thumb, we advise customers that automated test creation lag by no more than one sprint. In other words, the exit criteria for a second sprint includes the agreed upon automation for the functionality of the first sprint, and so on.

Figure 1: The "Sprint+1" Law of Automated Testing



This rule recognizes that rarely is there enough time within a sprint to automate the tests of that sprint. But it also lessens the temptation to let test automation slide to later and later sprints. Furthermore, this "sprint+1" law of automated testing helps to ensure (a) that time for test automation is allocated to each sprint, and (b) that if the team is not able to complete its target automation, the gap in remaining work is treated just as a code gap would be. That is, either the sprint is extended to complete the work, or the outstanding tasks are moved to the backlog for allocation to future sprints.

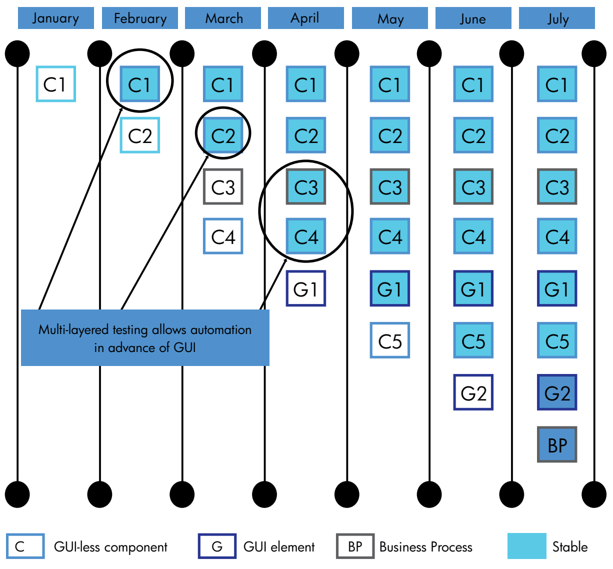
In speaking with customers who are struggling to integrate automated testing into Agile, our first question involves just this use of the backlog. Is the backlog being used to capture test automation tasks that were planned but left incomplete in the sprint? All too often, the answer is no. The upshot of this is that critical work remains "hidden" to the scrum master. Again, think of it in terms of code: what would be the ramification if the backlog wasn't a reliable measure of completed features? The work to automate tests is as necessary to Agile project success as code creation.

The plot thickens

As a final complication, modern applications are not the stovepiped things of several years ago. Today's application is more often a conglomerate of "headless" (that is, GUI-less) services and sub-applications. This means test automation can no longer assume each component will have a GUI element to record against. However, these componentized or "composite" applications also provide automated testers an advantage in Agile. With the right tools, teams are able to start automated testing before the GUI components are stable, or even available.

In early sprints, teams may be engaged primarily at the component or service layer. This means test automation that is centered around API, protocol, and Web service validation. As the application evolves, the team develops test automation for the GUI and business process (that is, cross-application) layers.

Figure 2: Multi-layered Automated Testing in Agile



Having developed automated tests from the service to GUI layers, the team is well positioned in later sprints to create more complex test scenarios which traverse multiple layers of the application stack within the same test—also called multi-layered testing. An example might be a test that fetches data via a web services or API interface to a database, and then uses the data for the rest of the test scenario at the GUI layer. This strategy provides greater test coverage, while uncovering defects that might not otherwise be encountered until production.

Conclusion

While many Agile teams will say quality is a team-wide responsibility, too few recognize that this means test automation is co-equal in importance to code. Successful Agile teams know:

- Time must be explicitly allocated for test automation in each sprint
- Automation should begin as early as possible and should not lag by more than one sprint
- When assessing the results of a sprint, the automation goals should be considered as vital as the development objectives
- Multi-layered testing allows for test automation in advance of GUI stability, and furthers test coverage

To hear about the real-life experiences of how HP R&D organization has applied test automation in their Agile projects, visit YouTube at: **http://www.youtube.com/hewlettpackardvideos#p/search/1/LhdFgoHOUSw**

Shift the way your organization thinks; transition from traditional waterfall software approaches to iterative agile methodologies with HP Software, visit **www.hp.com/go/agile**.

4AA3-2489ENW, Created January 2011; Updated March 2011, Rev. 1