

Five Keys to Agile Test Automation for Government Programs

Table of Contents

SEI WEBINAR SERIES Keeping you informed of the latest solutions.....	3
Carnegie Mellon University.....	3
Five Keys to Agile Test Automation for Government Programs.....	4
Notices	7
Overview	8
How is testing different in Agile software development?	9
Testing is development, development is testing.....	10
Early and repeatable testing is Agile key to quality.....	11
Agile “Inter-twingles” Development and Test	13
Agile iteration relies on more testing earlier.....	15
No Automation? The Backblob is going to get you!	16
DoD Acquisition requirements are unique	19
Left-shift with Agile Testing	21
Poll 1.....	24
Who should develop automated tests?.....	27
Some factors to consider	28
OEM or government engineering or test staff?.....	29
Poll 2.....	31
How can my program successfully adopt Agile/Automated testing?	34
Automated Testing = Structure and Strategy Change	35
Ensuring Test Automation Success	38
Program-Wide Test Asset Management System	40

REMEMBER: No Automation? The Backblob’s gonna getcha!	43
What kind of tool chain do I need to support automated testing?.....	44
REMEMBER: No Automation? The Backblob’s gonna getcha!	45
Program-Wide Test Asset Management System	46
REMEMBER: No Automation? The Backblob’s gonna getcha!	47
What kind of tool chain do I need to support automated testing?.....	51
Testing scope, lanes, focus, tooling	52
Tools for Test Automation	55
Tools for Test Automation	57
Test Automation Reference Architecture.....	59
Poll 3.....	66
What kind of testing should/should not be automated for Agile software development?.....	69
Testing scope, focus, automation	70
Avoid test automation gotchas.....	72
Poll 4.....	76
Summary	79
No magic!	80
Reasons Automated Test Investment is Often Delayed in Agile Adoption	81
Remember: Automated Testing = Structure and Strategy Change	85
Start early, keep at It!	86
SEI WEBINAR SERIES Keeping you informed of the latest solutions.....	94

SEI WEBINAR SERIES | Keeping you informed of the latest solutions



Carnegie Mellon University

Carnegie Mellon University

This video and all related information and materials (“materials”) are owned by Carnegie Mellon University. These materials are provided on an “as-is” “as available” basis without any warranties and solely for your personal viewing and use.

You agree that Carnegie Mellon is not liable with respect to any materials received by you as a result of viewing the video, or using referenced websites, and/or for any consequences or the use by you of such materials.

By viewing, downloading, and/or using this video and related materials, you agree that you have read and agree to our terms of use (www.sei.cmu.edu/legal/).

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

© 2017 Carnegie Mellon University.

Five Keys to Agile Test Automation for Government Programs

Five Keys to Agile Test Automation for Government Programs

Robert Binder and Suzanne Miller

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213



Software Engineering Institute | Carnegie Mellon University

Five keys to effective Agile test automation for Government programs
June 28, 2017
©2017 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

**001 Presenter: And hello from the campus of Carnegie Mellon University in Pittsburgh, Pennsylvania. We welcome you to the Software Engineering Institute's webinar series. Our presentation today is Five Keys to Agile Test Automation for Government Programs. Depending on your location, we wish you a good morning, a good afternoon, or good evening. My name is Shane McGraw. I'll be your moderator for today's presentation, and I'd like to thank you for attending.

We want to make today as interactive as possible, so we will address questions throughout the presentation and again at the end of the presentation. You can submit those questions to our event staff at any time through the Chat tab or the

Ask a Question tab on your webinar consoles. We also ask a few polling questions throughout today's presentation and they will appear as a popup window on your screen. In fact, the first polling question we want to ask is: How did you hear of today's event? And that will be on your screen now.

Another three tabs I'd like to point out are the Download Materials, Twitter, and Survey tabs. The Download Materials tab is a PDF copy of the presentation slides there now, along with other related work from the Software Engineering Institute. For those of you using Twitter, be sure to follow @sei_news, and use the hashtag #seiwebinar.

And now I'd like to introduce our presenters for today. SuZ Miller is a principal researcher at the SEI. Her current research is focused on synthesizing effective technology transition and management practices from research and industry into effective techniques for use in the governance of programs, adopting or contemplating adoption of Agile or lean methods.

Robert Binder is an SEI senior engineer responsible for client engagements and applied research related to architecture, assurance, and automated testing. Prior to joining the SEI, he was a founder of two consulting businesses and a test automation startup. He's developed hundreds of applications, app systems, and advanced automated

testing solutions. Bob holds a bachelor of arts in political science in government, a master's in finance, and a master's degree in electrical engineering. Bob, SuZ, welcome. SuZ, all yours. Take it away.

Presenter: Thanks very much Shane. So we're very pleased to be her today. Bob and I have been working on helping organizations in the government space understand what they're getting into if they want to start looking at issues related to automated testing, and this is part of our ongoing research in the Agile and government group in looking at adoption barriers for Agile for customers in the acquisition community who are trying to do things with Agile. So this is part of an ongoing series, and we're very pleased that Bob is able to help us with this, because he knows more about this stuff than pretty much anybody at the SEI does, so.

Presenter: Happy to be here.

Presenter: So we're glad that he's here. Today we're going to talk about five keys. Gonna send that forward.

Notices

Notices

Copyright 2017 Carnegie Mellon University. All Rights Reserved.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM17-0395



**002 You've got the remote right now. Oh, there we go.

Overview

Overview



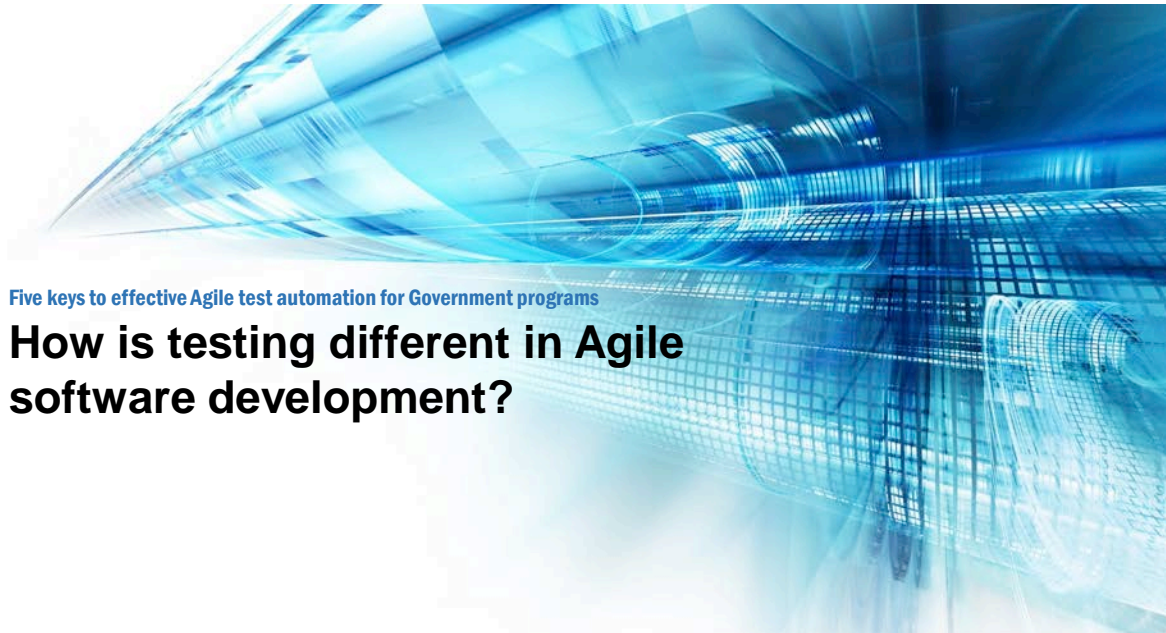
- 1) How is testing different in Agile software development?
- 2) What kind of testing should/should not be automated for Agile software development?
- 3) Who should develop automated tests?
- 4) What kind of tool chain do I need to support automated testing?
- 5) What are some ways that DoD acquisition programs can successfully adopt Agile/Automated testing?



**003 How is testing different in Agile software development? There's some key things you need to know there. What kind of testing should or should not be automated for Agile software development? Who should do that development? It's not always as clear as you might think it should be. The thing everybody talks about, and we will as well, is what kind of tool chain do you need to support automated testing, but then, beyond that, what are some ways that DoD acquisition programs can successfully adopt Agile and automated testing?

So with that, I'm going to let Bob start us off and talk about how is this different.

How is testing different in Agile software development?



Five keys to effective Agile test automation for Government programs

How is testing different in Agile software development?

 Software Engineering Institute | Carnegie Mellon University

Five keys to effective Agile test automation for Government programs
June 29, 2017
© 2017 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

**004 Presenter: Well thanks, SuZ. I think that's a great segue into this.

Testing is development, development is testing

How is testing different in Agile software development?

Testing is development, development is testing

Phased

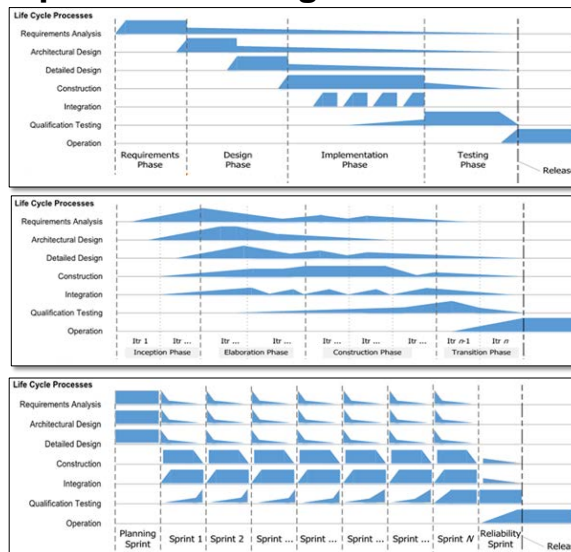
- Design, Code, Test, Test, Test
- Release

Incremental

- Design a little, code a little, test a little...
- Test, Test, Test
- Release

Agile

- Test, fail, code, test, pass ...
- Test, fail, code, test, pass ...
- Test, fail, code, test, pass ...
- Release



Source: IEEE Standard 1633P, *Recommended Practices for Software Reliability Engineering*

**005 First of all, testing is development and development is testing, or at least the two things are put together in an interesting way in Agile practices. Compared to phased approaches to software testing where those things are distinct, where you have a design, code, test, test, and test, and then followed by release, or even in the incremental model, where we might do, as it has been called, the "design a little, code a little, test a little," and then followed by a focus on successively different larger scope testing, leading up to release.

In most Agile practices, this gets sliced down much more finely, and where-- we start out by doing a test and we expect the first test fails, usually. We then implement something that achieves that achieves that function, do the test

again, and then we expect it to pass. That cycle gets repeated many, many times, and very frequently that's the primary difference. That all leads up to release. So you can see the strategies for testing are very much different in terms of where their focus is.

Early and repeatable testing is Agile key to quality

How is testing different in Agile software development?

Early and repeatable testing is Agile key to quality

Agile Quality Practices

- **Voice of the customer**
- **Commitment management**
- **Definition of Done**
- **Demos**
- **Retrospectives**
- **Test Driven Development**
- **Exploratory Testing**
- **Living Tests**
- **Find and fix within sprint**

Test Driven Development

- **TDD – Test Driven**
- **BDD – Behavior Driven**
- **ATDD – Acceptance Test Driven**

Typical Agile Testing Tool Chain

- Component/API testing: Junit, Nunit...
- BDD/ATDD: Cucumber, SpecFlow...
- GUI testing: Selenium, Ranorex...
- Continuous Integration: Jenkins, Hudson ...
- DevOps: Chef, Docker, Puppet...

**006 And that focus really is getting to an early and repeatable testing as a key to quality. There are lots of Agile practices that drive quality, and Agile's underlying pharmacy on this is a little bit different than other ideas that were oriented towards these more structured and standalone type of practices in other lifecycle approaches.

One in particular I want to mention is test-driven development. In test-driven

development, the idea there is, again, test is development; development is test. The two things are very closely related as the system is produced. There's several flavors of this that are commonly used. One is called test-driven development, and that is usually applied at the programming or developer level. Behavior-driven development is another way of looking at how to translate user stories and features into test, and acceptance-test-driven development is primarily done at the user interface level of a system.

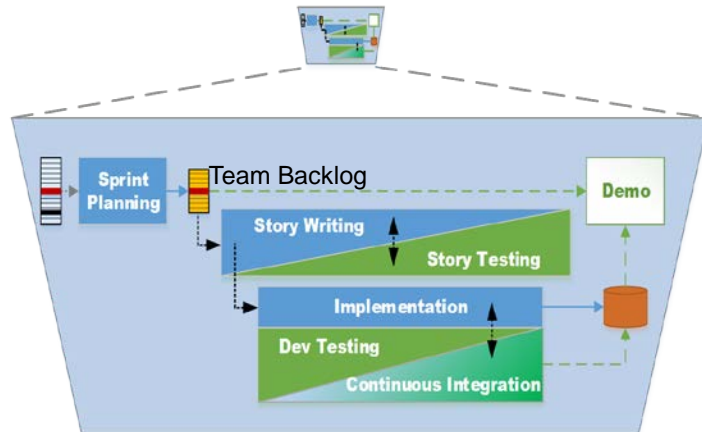
So, there are tool chains that are associated with this. We've got a list here of some of the common ones. We'll talk a little more about this later on and some of the specific possibilities. These are just a very small sample of the literally hundreds of testing tools that are available to support this kind of activity.

Agile “Inter-twingles” Development and Test

How is testing different in Agile software development?

Agile “Inter-twingles” Development and Test

- Development and test are not separate or standalone
- Tests are added to a repeatable test suite
- Test suite is repeated for each change
- Demos *not* a replacement for testing



**007 So Agile can be said to "inter-twingle" development and test.

Presenter: Which is my favorite term. I love that term.

Presenter: It's one of your favorite terms, and what does that mean? Well, the blue trapezoid on the screen is kind of to suggest how Agile is a little bit different from the classic V model of software development, and in this you can see that the activities of development really kind of coincide with the activity of testing, and the two things are interleaved very closely; it's really hard to pull them apart. So they're not standalone and separate. Tests are added to a repeatable test suite, and I think this is key for understanding how automation factors into Agile development. And

then each time there's a change in the system, which happens an awful lot in a frequent way, that test suite is repeated.

Demos, by the way, are a key quality practice in Agile development, but they're certainly not a replacement for-- and serve a different role in testing-- even in Agile testing.

As an example of this, you can think about what happens in many large development organizations. For example, in Google, where they have tens of thousands of people checking in code very frequently, on an hourly basis, perhaps even more frequently than that, the entire target systems get rebuilt every time that happens.

Presenter: You can't do that if you're running all of the tests that are in the regression suite manually.

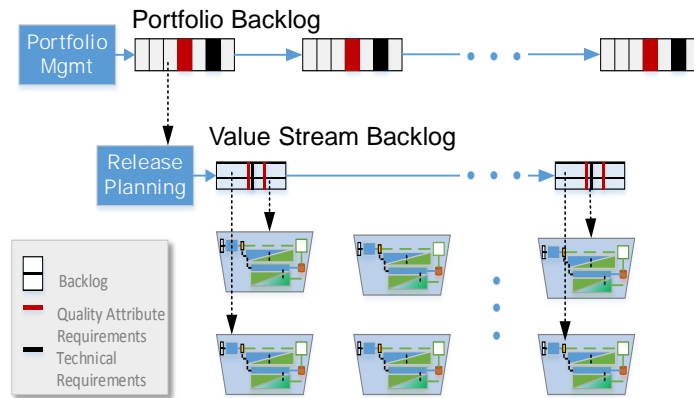
Presenter: That's right. And so you need to have something that's an overlay that accomplishes that for all the participants in software development automatically. It really would not be possible without the test automation system that they developed there, which is really quite impressive.

Agile iteration relies on more testing earlier

How is testing different in Agile software development?

Agile iteration relies on more testing earlier

- Testing used for rapid refinement of loosely understood requirements, architecture, and design
- Requirements and implementation are finely sliced
- Each slice is tested immediately and repetitively



**008 Another aspect of this is that we focused on testing earlier, and so not only do these iterations happen more rapidly within a sprint, but there are, of course, many sprints within a development of a release, and so testing becomes a way for developers to refine the kind of notional definitions of their requirements, features, user stories, etcetera, to actually cast that into working code and then confirm that that code works according to the starting point, and those two things-- the idea is to narrow that down as much as possible.

Presenter: Fast feedback.

Presenter: That's right. Yeah, rapid feedback. And again, one of kind of the key principles of Agile development is that these small slices

are tested rapidly and immediately so that there's no kind of delaying that activity until some later stage. Does this remind you of any particular stories of how that unfolds and circumstances you're familiar with?

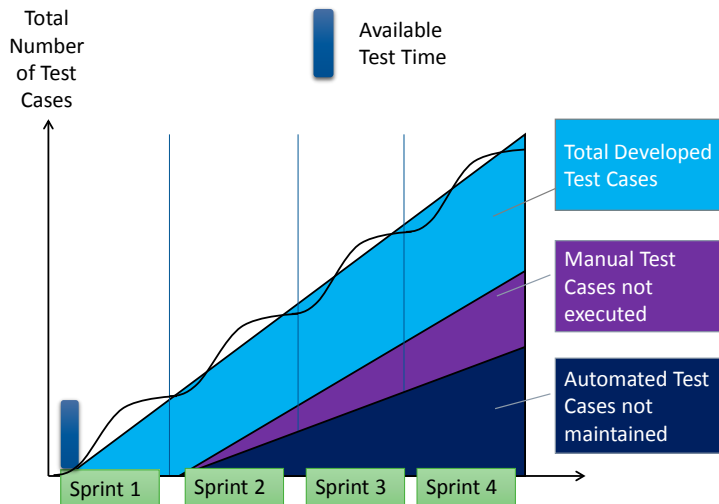
Presenter: Well, we see lots of this in large systems that have adopted Agile and that are trying to move in this direction, but they've got some unique requirements and some constraints that we have to deal with, and I think we've got some things coming up on that I'll talk about.

No Automation? The Backblob is going to get you!

How is testing different in Agile software development?

No Automation? *The Backblob is going to get you!*

- The extent of manual testing is limited to the capacity of testers
- The extent of automated testing is limited to the capacity of test scripters
- Total number of tests increases as project progresses
- Typically, only the newest features are tested



**009 Presenter: Okay, great.

Presenter: After the back-blob.

Presenter: After the back-blob, yeah. And so we-- some organizations that I've worked with in

the past and others have attempted to test this kind of Agile testing in a rapid manner without the support of automation, and so here's-- that usually doesn't work out too well, and there's some reasons for that. One is that the extent of testing that you can do-- and this is where tester interacts directly with the system under test and manual testing. That's limited to the amount of hours that you have of people who can do that kind of stuff. Then that typically is fixed or it doesn't change much once a project gets in motion. The same can be said pretty much about the ability of people who are maintaining or developing automated tests to make changes and create new tests.

Presenter: Testing has always been a constrained resource, in every environment, whether it's big bang approach or incremental. I mean, I don't know any environment where they say, "Oh, we've got more testers than we know what to do with." That's just not what happens.

Presenter: That certainly is true, yes. So there's always a constraint on it, and in Agile development, as we've noted, we want to do lots of tests and run them very frequently. So the total number of tests increases as we move along, and what tends to happen is that the tests that were done previously, if they're not automated, get crowded out. So that's what we kind of refer to as the testing back-blob, and it sort of works like this.

So, let's suppose that you had a certain amount of available test time at the start of a sprint, and in sprint one you use that time to develop a certain number of test cases. Well, I think just simply you can see that that goes on for however many sprints that you have, and you end up with a certain number of test cases. So let's say if you had the ability to do about a hundred, you'd end up, after four sprints, with four hundred test cases. Now, each of those takes a certain amount of time to do, and if you're doing this manually, how many do you have time to do? Well, including the development, you might say that I have time to do a hundred tests, but that means you don't have time to repeat all those tests that you did previously. They get kind of crowded out, and the same thing happens, as there are often many changes during a cycle of sprints-- some of those automated test cases don't get changed or revised to reflect the most current version of the system.

So you end up with a large collection of tests that appeared to me-- it brought to mind the image of the 1957 B science fiction movie, "The Blob", which was a large, pulsating piece of protoplasm that consumed all of Los Angeles, and some projects that I've worked in previously, it seemed like this was sort of what was going on as the tests were produced rapidly but really couldn't be used.

So without automation, that's kind of a situation that you can run into.

DoD Acquisition requirements are unique

How is testing different in Agile software development?

DoD Acquisition requirements are unique

DoD requires large programs to plan for and undergo independent Operational Test & Evaluation

- Planned and executed by a different organization than the Program Office acquiring the software system
- Requires a very early Test and Evaluation Strategy more compatible with a “big bang” delivery than the incremental delivery typical in Agile

USAF guidance (AF99-103) now aligns independent testing with a more incremental approach

- Integrated testing and integrated test teams are a specific strategy called out
- Incremental testing is specifically discussed and encouraged prior to full operational testing of a deployed capability



**010 Presenter: So let's talk for a minute about some of the things that get us in the DoD in particular.

Presenter: Right.

Presenter: So our DoD acquisition requires large programs to plan a different kind of testing than what we've been talking about. It's called operational test and evaluation. If you're large enough, you're going to have to submit a separate plan for that to a separate organization that is not the one that manages the program. So this is an independent, very independent-- almost, you might say, isolated-- organization, and they have a budget, constrained budget just like everybody else, and they've got a lot of programs that want their testing services and need their testing services, and they are

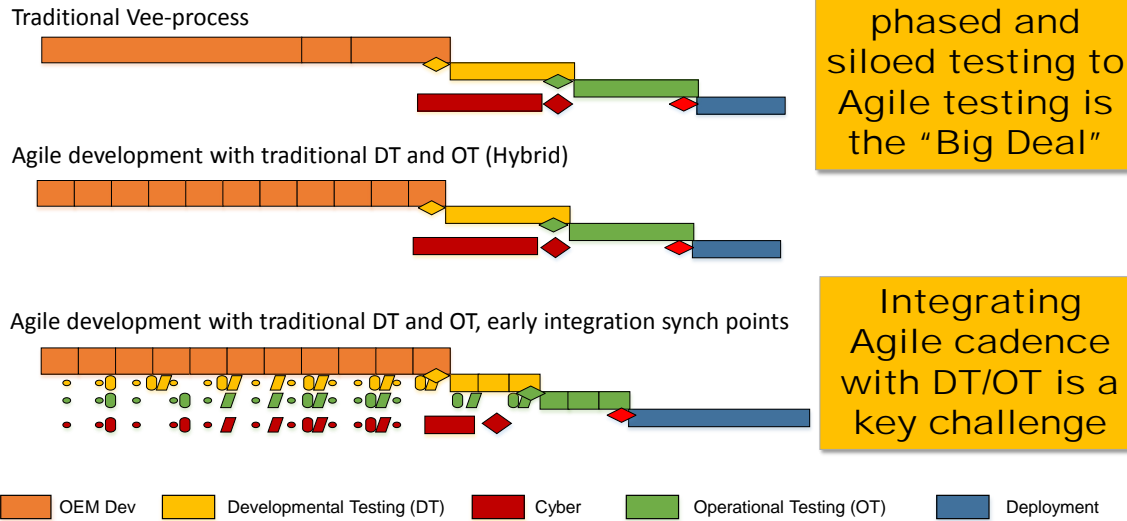
accustomed to a big-bang delivery kind of item so that your program sort of gets paid attention to in one block. That's not very incremental, and that is a budgeting and cultural issue that is one that DoD organizations are going to run into.

We do see some success with the OT&E, as we call them, folks moving to a more incremental strategy but it's not ubiquitous yet, and it's going to come sooner though than later because of this new guidance that came out. This Air Force 99-103 is a guidance document that does align the independent testing with the incremental approach. So this is very recent, in the last month, and this I think is going to make a big change in how organizations in the DoD acquisition programs interact with operational tests and evaluation. They specifically talk about incremental testing, they talk about integrated test teams, so this could be a sea change in the way that this independent testing occurs, and this is going to-- we talk about Shift Left-- we'll talk a little bit about that later. This actually provides a potential opportunity for moving even the operational test over to the left. So that's the most positive thing we've seen recently in terms of practices that are likely to change an acquisition to accommodate some of this. What does that mean though? It means they're going to need all that automated testing too, because as they go incremental, the back-blob issues that is going to be coming up is going to come up for them, just

like it does for everybody else. So, even more importance for automation.

Left-shift with Agile Testing

Left-shift with Agile Testing



**011 Presenter: Yeah, right. So let's take a little bit closer look at this idea of left-shift. In this, we've got kind of a notional layout, the timeline of a typical phased or V process, where that's the orange development work followed by developmental testing and operational testing and then finally release.

Presenter: And I want to point out cyber you mention separately in here, and in most of our federal, not just DoD organizations, cyber testing is also its own thread, its own organization, and figuring out how to integrate that into the cycle-- people are working on it, but that is not a problem that's solved yet, and so that

can cause some perturbation in your content for incremental testing.

Presenter: Right, and so those two things-- if they're done on separate tracks, as is probably the most common, then there are plenty of opportunities for potential conflicts, and there's also kind of an interesting dilemma that arises when we attempt to do Agile development.

So here's Agile development in that earlier part of the lifecycle, paired with sort of traditional DT and OT, and you can see that that really doesn't-- that model--

Presenter: It doesn't give us the feedback.

Presenter: Right. You don't get the early feedback and you really have the same kind of situation that you would have in the traditional one, and what we're trying to work towards and assist some of our clients in doing is to begin to take pieces of those three kinds of later-stage testing and achieve those earlier on in the lifecycle.

Presenter: And I do have an instance of a case where this is happening in a sustainment organization in the Air Force, where air worthiness is another thread that you can end up with if you're working with airplanes, for example, and their typical air-worthiness recertification, whenever a change happens to something that touches the airplane, takes months. This organization had

a need for fairly frequent changes to the Apple operating system-- they're using iPads for maintenance-- and so they had to have changes happen quickly, and so they were actually able to get the OT&E folks to allow them to actually automate what was manual testing and approve those tests as being, "If we pass these tests, then we're good on the air worthiness, except for the new thing, whatever the new thing is, then we have to test that traditionally." So that to me was a huge breakthrough, and it represents that bottom line in ways that I hadn't seen before. So I got at least one that's moved in this direction.

Presenter: That's great.


Presenter: So we're looking for more. If anybody out there in the audience has other examples, send them me, because I collect them.

Presenter: Right, and this is something that we're very much interested in and working on in a number of different ways, and so kind of watch this space. We hope to have some more concrete guidance on how to deal with these issues later on.

Poll 1

How is testing different in Agile software development?

Poll 1




A: What software life cycle do you use?

- A. Phased, “waterfall”
- B. Incremental
- C. Evolutionary, “Agile”
- D. Hybrid

B: Is a different life cycle:

- A. Being evaluated
- B. Being piloted
- C. Active rollout
- D. No change anticipated

 Software Engineering Institute | Carnegie Mellon University

Five keys to effective Agile test automation for Government programs
June 29, 2017
© 2017 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

12

**012 So we're now at our first poll.

Presenter: So, as I mentioned, we've got back-to-back polling questions here. Before I hit the launch, just a quick comment, if you guys to respond to, from Richard, asking: How do you respond to the assertion that TDD is the enemy of architecture?

Presenter: It can-- I think the sense of the question as I take it is that TDD presumes that an architecture isn't necessary to get started. That's not necessary. You'd like to have some definition of architecture and an approach laid out in advance, and certainly some of the scaled Agile methods that we use and follow and that others do provides a means for achieving that.

Presenter: We're not seeing a conflict between architecture and TDD in the places where it's being used. Primarily these are large programs, so it's not a single team. I think this kind of a statement may be something you would see more with a much smaller kind of project, but by the time I get to the team level in some of these large systems, there is architecture. There are architectural runways, and I'm doing test-driven development within the umbrella of an architecture that's existing and evolving. Now, acceptance-test-driven development is essentially something you would need to do in concert with architecture because you're really looking at the requirements and architecture level there. The TDD as we've been talking about it, which is a much lower level development task down at the unit test level essentially, I don't see and I haven't seen any conflicts in how it's used in combination with a good architectural strategy.

Presenter: In combination, and that's the key to it. TDD as practiced sometimes can be problematic for a lot of reasons, but that's a story for another webinar.

Presenter: There you go.

Presenter: So I'm going to close out the first question, which was: What software lifecycle do you use? And now I'm going to post the next one, which is a different lifecycle. So that will be on your screen now. Take about 15 seconds, and I'll give it back to you guys.

Presenter: So the themes from this part of the webinar-- development is test, test is development. I mean, think about it that way and a lot of things fall right into place, and don't let the test back-blob get you. That's a big thing. And also don't-- there are constraints from a DoD acquisition viewpoint, but that doesn't mean that you can't work within them, and people are working around them and through them. So don't think that just because you're in a DoD acquisition, "I can't do any of this stuff." Any other themes that you'd want to bring out?

Presenter: Just that the understanding of how Agile achieves quality-- it's taken me a while to appreciate sort of how different it is, and I would say that the Agile approach to quality assurance really is a system, and it relies on all these various practices which--

Presenter: Not just test.

Presenter: Not just test, which we alluded to earlier. So all of those fit together in a particular way.

Presenter: Just to wrap up our results, we got 33 percent hybrid, 33 percent evolutionary, 18 percent incremental, 16 percent phased/waterfall. Then for the second question, which was, "Is a different lifecycle being evaluated?", that was 21 percent; being piloted, 19 percent; active rollout, 21 percent; no change anticipated, 40 percent.

Presenter: Okay, so we still have a lot of people that are in the No Change Anticipated, although the way we worded that, they could be using Agile with no change.

Presenter: That's true. That's right. So we don't know.

Presenter: All right, we have to go back and get retrained on survey question-writing.

Presenter: It was a poll, not a survey.

Presenter: Oh, right, right, right. That's right, that's right.

Who should develop automated tests?



**013 Presenter: So let's move along and try to answer-- that was sort of the What question. Let's try to take a crack at who should be involved in automated test development.

Some factors to consider

Who should develop automated tests?

Some factors to consider

Test automation is a specialty: the software development engineer in test (SDET)

SDET skill set versus functional tester skill set

- Writing automated tests IS writing software
- Not all functional testers come from a software development background

Organic or Outsource?

- For some types of testing, there are consulting groups that will convert manual to automated tests
- Works best with well understood systems with well-documented tests and sufficient subject matter experts to answer the myriad questions that will come up



**014 So I'm going to let SuZ kind of go at this part.

Presenter: So this is one of the things in an acquisition that we really have to look at, because the people that have been in what we would call traditional testing organizations don't necessarily come from a software development background. Many of them come from an operations background, they come from the system knowledge, domain knowledge viewpoint, not from the technical details of how a software program is constructed, and now we just said test is development, so we have potential conflicts there, and that's not to say everybody's in that boat, but there's a subset of folks that really aren't equipped in terms of how they've been trained in how to do this kind of development and thinking of it as a development.

So there is this different set of skills, and you've got to figure out: Do you have the right skills within your test group? Do you need to supplement that? And if you do need to supplement that, can you do it with people that are inside the organization, or do you need to outsource it? There are companies that will come in and take a legacy system and they will write the automated tests from the legacy manual tests for you. You have to have well documented tests. I mean, there's a whole lot of caveats to that, but it can happen, so you have to make those decisions. And then the other decision is: How am I going to deal with-- go ahead and go to the next one, please.

OEM or government engineering or test staff?

Who should develop automated tests?

OEM or government engineering or test staff?

OEMs with active Agile practices typically use local automated testing for component testing and local integration testing

- Contracting to obtain those tests as part of software delivery increases the test base for regression testing
 - Be sure to ask for the test environment as well, so the OEM tests can be run by others!

DT&E (Development Test & Engineering) government staff can interact with Agile OEMs via Iteration, System, and Release Demos and the activities that lead up to them

- Leveraging automated tests from early activities can build confidence in the accumulation of evidence related to the system's robustness



**015 --With my OEMs or contractors? The ones that have

active Agile practices often are using automated testing. From the government side, can I actually leverage that? Now, I can't leverage that if I didn't ask for that material to be delivered to me as part of the contract, right? But that-- so contracting comes into play here, which most people probably did not expect to see anything about contracting in an automated test webinar, but it is a factor in these kinds of environments.

The other thing is that we do have opportunities through these demos-- this is one of the things demos can do-- is give the government folks that are in system test and development test, even in OT&E, to actually see what's happening and provide feedback or get feedback that is going to allow them to do a better job of their testing, however they're doing it. So even if you don't have a hook in to get tests from your OEM, the demo process is one of the processes where you can invite participation, and should invite participation, from the test community.

Presenter: Right, yeah, and that's a pretty low-friction way to at least get started on this, this earlier shift-left integration. That doesn't require any additional training, tooling, just presence at one of these demo events.


Presenter: And not to say that that's trivial. I mean, I've been parts of programs where if you're not there

in person, you can't get through a firewall to get to the demo, and there's those kinds of constraints as well, but it is a much lower barrier to entry than actually going in and getting all the tests from the OEM.

Poll 2


Who should develop automated tests?

Poll 2



Who is responsible for automating tests in your project?

- A. We don't automate
- B. Non-technical testers or QA staff
- C. Application developers
- D. Test engineers
- E. Software development engineer in test (SDET)

 Software Engineering Institute | Carnegie Mellon University

Five keys to effective Agile test automation for Government programs
June 29, 2017
© 2017 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

16

**016 Presenter: All right, so we've got our second poll now.

Presenter: You'll see that on your screen now. The question is: Who is responsible for automating tests in your project? And maybe while we're looking at that one, we had another comment-- let me pull that up-- from Jean-Paul asking: How do you maintain the independence of team testing or include this team into the same team Agile development?

Presenter: Say that one again.

Presenter: How do you maintain the independence of team testing, or include this team into the same team for Agile development?

Presenter: Oh, I see. So I can talk about some examples of how we see this happening in government settings. I have one example where the DT&E organization, the testers in that organization, are all assigned to different Agile teams that are working, and there's a part of their time where they are actually working in the teams, doing the kind of test automation, the kind of local testing, but they're also using that time to actually build what we would call the independent test cases that are going to be used as things go to release and go a little bit farther into integration. So they are-- they're buying into the concept of, "Independence does not mean isolation," and they're buying into the fact that they learn a lot about the system to build better tests by being involved at that local level on the teams. I have another case that's a larger application where we have layers of testing that are going on, and the layer that is at the OEM level-- so the contractor level-- they have TIMs, Technical Interchange Meetings, with the testers on the government side that come out of these different layers on a regular basis. So in that case I don't have the government testers in bed, if you will, with the team, but they are frequently interacting with those teams and they get to provide feedback, and they also do attend

the demos. Everyone's invited to the demos. So those are a couple of different strategies that we've seen in government settings for doing that.

Presenter: Okay, and we'll wrap up our polling question with some results. Twenty-one percent, we don't automate; 6 percent, nontechnical testers; 23 percent application developers; 42 percent test engineers; 8 percent software development engineer.

Presenter: Okay, so I-- wow, only 21 percent are saying, "We don't automate." Okay, so from this poll, if you're in that 21 percent, you're a little bit behind the curve, if you didn't hear that, because-- so I don't know where our population is in terms of government versus non-government, but I would say if I were to poll a pure government audience, it would be higher than that, that this is at the beginning of the curve, not in the middle of the curve, for the government audience. But this is also just very informal support for: People are doing this. You got to get onboard.

Presenter: Yeah.

How can my program successfully adopt Agile/Automated testing?



Five keys to effective Agile test automation for Government programs

How can my program successfully adopt Agile/Automated testing?



Software Engineering Institute | Carnegie Mellon University

Five keys to effective Agile test automation for Government programs
June 23, 2017
© 2017 Carnegie Mellon University

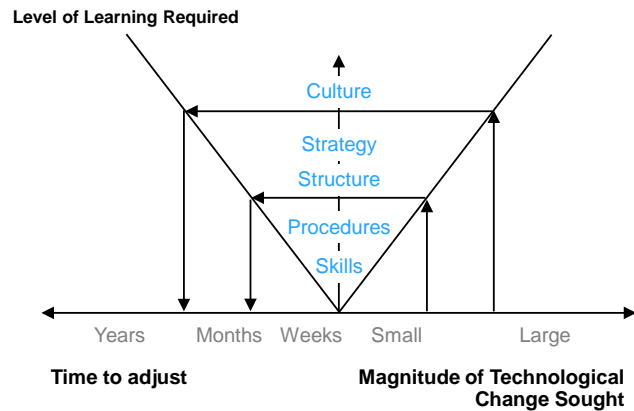
[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

**017 All right. So now let's go to kind of our third question. What we want to try to address here are some of the considerations that are going to drive whether or not you're successfully adopting Agile and/or automated testing within your program.

Automated Testing = Structure and Strategy Change

Automated Testing = Structure and Strategy Change

- Scope of change is similar to automating a manual business or operational process
- Requires many behavioral changes
- Not plug-and-play
- Learning curve



**018 Presenter: So this is a model that Bob has seen a lot now, and some of you have seen it--

Presenter: It's a great model.

Presenter: --If you have worked with the SEI. We talk about this a lot. This comes from Paul Adler years ago, but it's talking about how big is the change that you are contemplating. So the question here is: How big is the change if I'm going to Agile and automation-based testing? And I could argue that we are at least looking at structure changes. We've already talked about things related to how the testers interact. We've talked about the lifecycle differences. Those are structural changes. But there's also a strategy change in terms of moving from big bang to incremental. That

in itself is a strategy change. It means I have to change the way my resources are applied, change the way I phase my budget, right? So these are all strategic kinds of questions.

Some would argue that it's also a cultural change, and if you are in an organization where independence equals isolation, I would argue, yeah, you're moving into culture there as well. But a lot of times you're not dealing as much with culture as you are with the strategy and structure. Now, the thing about this model is that I have to deal with the stuff below too. Right? "Oh great, strategy. All right, we'll just our strategic plan and we'll change the structures," but you've also got to change the skills and the procedures that go along with this. So basically this is not a small change, and we are not-- part of our series Agile in Government, our theme is "Go in with your eyes open", and this is one of the places you need to go in with your eyes open.

So a couple things, and you can think of this-- it's like automating some other manual business or operational process. I have to understand what's being done now, I have to figure out what the structural changes are, and I've got to deal with the actual automation itself, the skills, abilities, etcetera, that go along with that. The behavioral changes just in terms of I may have test engineers that have never written code. Some of them decide they're going to learn

that; some of them decide they're not, and they're going to have to go do something different. So there's a lot of things that could be different in terms of the behavior. This is not a plug-and-play technology. This is not just about putting a new app on your cell phone and automatically being able to do the new things.

Presenter: Yeah, it's not like installing a virus detector on your PC or laptop. It's not that kind of automation.

Presenter: Right. Right, exactly. So these are all things to pay attention to, and of course the other thing that this diagram highlights is that you're not talking about weeks, in most cases, to do this, unless it's something very small. You're probably talking about months and possibly into years depending on how big your legacy system is that you're evolving into this. So don't expect this to be an overnight change, or else you will be disappointed.

Oh, and the learning curve. I forgot about that one, but we've already talked about that.

Ensuring Test Automation Success

- Treat development and operationalization of your test automation system like any other critical operational system
 - Agile development of test infrastructure & assets
 - Rollout support: deconfliction, training, socialization, funding of maintenance
- Grow or hire Software Development Engineers in Test (SDETs)
- The payoff of automation is repeatability and consistency
- Automation doesn't eliminate the need for manual testing (and testers)
- Training and support for tools is critical



**019 So some things that you want to be sure you're dealing with. Treat it like any other critical operational system. It is a system. You are building a system. Test is development. You have infrastructure and assets. You need to have rollout support. You need to have deconfliction across different organizations, training, all these things, funding of the maintenance-- don't like the back-blob get you. All those things are the same as if you're doing the system itself. And this idea of Software Development Engineers and Test-- SDETs? Is that how you say it?

Presenter: SDET, yeah. It's a Microsoft-ism originally, but now it's pretty well understood, I'd say, in the industry at large.

Presenter: It's a way to understand that I still have-- one of the things I've learned about software engineers is you tell them they're now a software tester and some of them feel like they've lost some status.

Presenter: A lot. Yeah.

Presenter: I think this is one of the ways of getting through that and saying, "No, you're still a software development engineer, but you're developing the test system and the test assets, and that's just as important as developing the system itself." If I can't verify that the system is working, I can't deploy it, or I shouldn't. And so it is a very important role.

What's our payoff for automation? Consistency, repeatability, and the thing that's not on here is just capacity. Without automation and all this incremental building and development, I don't have capacity for it in almost any environment, so we need all of those payoffs from automation. But it doesn't completely eliminate the need for manual testing and testers that do certain kinds of testing. Exploratory testing is something that maybe in ten years machine learning will be able to take up the role of exploratory testers, but not yet.

Presenter: Certainly not today.

Presenter: So there are certain things, and we'll talk a little more about that, that you need to still do

manually. So don't eliminate-- just because someone doesn't transition over into an SDET does not mean that they are not still useful for the testing activities.

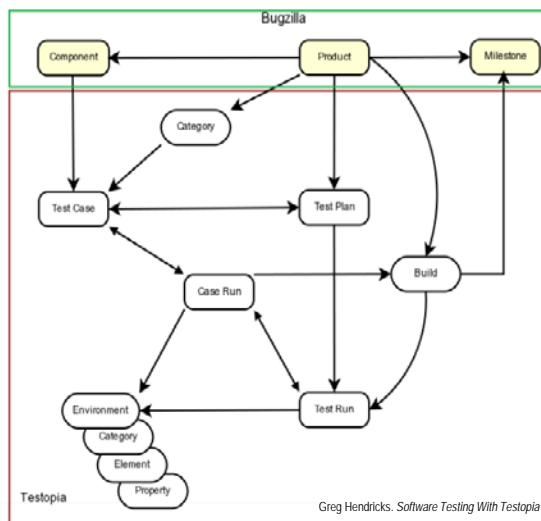
Presenter: Absolutely correct.

Presenter: All right, we've said "training" a lot. We're going to keep saying it, and the support for the tools, there's a whole-- there's licensing, there's configuration management-- there's all the things that go along with managing software. You're managing a software system. So get used to it.

Program-Wide Test Asset Management System

Program-Wide Test Asset Management System

- Team-wide repository, test meta-data
- Tool chain interfaces
 - Test harness(es)
 - Bug tracking
 - Requirements
 - Version control
 - Continuous integration
- Part of all leading Application Lifecycle Management (ALM) systems
- Several open source systems
- *Can track status of **all** test activity*



**020 All right, why don't you talk about the asset management system, because that's a really big part of this.

Presenter: Kind of picking up on that theme of test asset management, there are tools which are sort of not all that well known that I refer to and others do as Test Asset Management Systems, and that basically means that-- what they do is they provide a database of metadata about the testing activity.

So you can see a picture here on the left that kind of suggests how one of those-- this is actually a diagram from Testopia, which is an open source tool that's paired up with another open source tool that's very popular called Bugzilla, and you can see there that the items in the diagram basically are about the testing process, the things that get created in test, what their status is, what you've done with them, not necessarily the test cases or the automated scripts themselves. So this is information about testing and where it stands. This can be very useful and is particularly important in trying to support some of that collaboration, or at least shift left from the original development organization up to development test organization, operational test organizations.

So these kinds of systems typically have interfaces to other parts of the tool chain. We'll talk about those later. There are some of the sort of main ones. They are part of all of the leading application lifecycle management systems, and those are, of course, the systems from large

vendors like IBM, HP or Microsoft, and others as well.

Presenter: This is like VersionOne, and the CollabNet and Rally and some of those other-- in the Agile space--

Presenter: For example, in the Agile space, those would be part of them, but you'd also see these as part of the tooling suite. For example, HP has one that's popular called Quality Center, and there are many others.

There are several open source systems. They have their pros and cons. Of course all these systems do. The key thing about this is that they give support for looking at and tracking the status of all test activity in sort of a single dashboard or place. So it gives you kind of a focus for that.

Presenter: And you can imagine, if I've got these layers of developer testing, development test and evaluation-- sort of the government side of initial testing-- and then the operational, I have a lot of activity going on potentially. So this kind of a system is going to make that activity a little bit easier to manage.

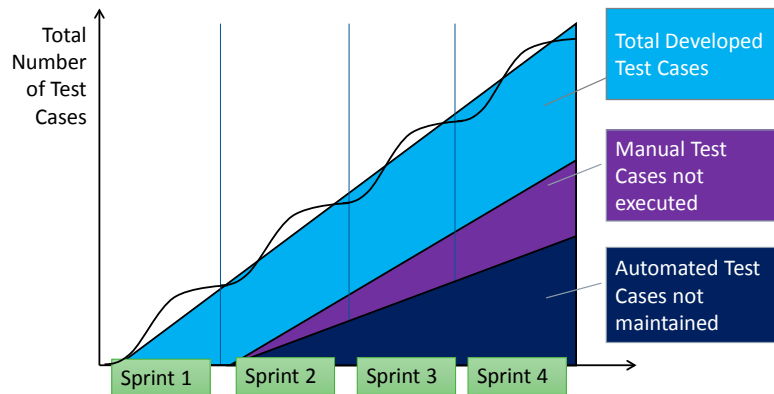
Presenter: Right, and so it also provides a kind of common way of defining test requirements such that if there's a test requirement that's really important to developmental testers, for example, you can start to call that out early on and perhaps

achieve part of it, at least, during some of the earlier phases. So there's an opportunity for sharing and reuse of test assets or at least concepts, information, definitions-- lots of ways that this can facilitate cooperation.

REMEMBER: No Automation? The Backblob's gonna getcha!

REMEMBER: No Automation? *The Backblob's gonna getcha!*

- The extent of manual testing is limited to the capacity of testers
- The extent of automated testing is limited to the capacity of test scripters
- Total number of tests increases as project progresses
- Typically, only the newest features are tested



**021 And then kind of remember, in terms of our subject here, is: What does it take for success? Don't let the back-blob get you.

What kind of tool chain do I need to support automated testing?



Five keys to effective Agile test automation for Government programs

What kind of tool chain do I need to support automated testing?



Software Engineering Institute | Carnegie Mellon University

Five keys to effective Agile test automation for Government programs
June 23, 2017
©2017 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

**022 All right. So we're going to go now to the next topic.

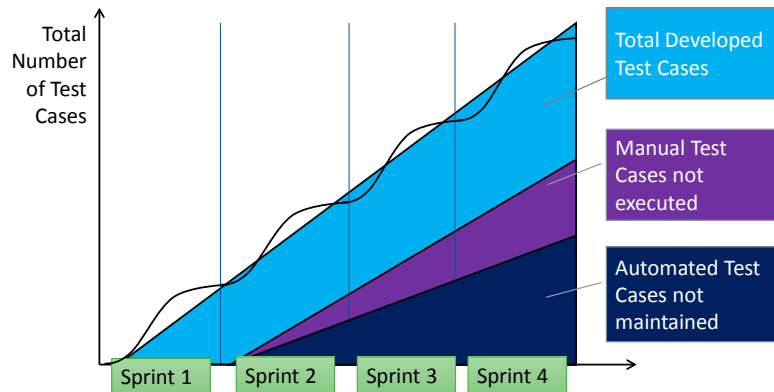
Presenter: Before we go there, do we have any questions?

Presenter: We do have a good question here that just came in. It said: Should SDETs--

REMEMBER: No Automation? The Backblob's gonna getcha!

REMEMBER: No Automation? *The Backblob's gonna getcha!*

- The extent of manual testing is limited to the capacity of testers
- The extent of automated testing is limited to the capacity of test scripters
- Total number of tests increases as project progresses
- Typically, only the newest features are tested

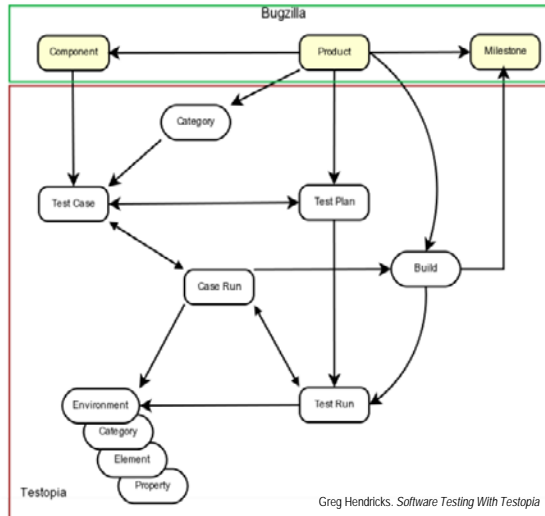


**021 I think I said it right there--

Program-Wide Test Asset Management System

Program-Wide Test Asset Management System

- Team-wide repository, test meta-data
- Tool chain interfaces
 - Test harness(es)
 - Bug tracking
 - Requirements
 - Version control
 - Continuous integration
- Part of all leading Application Lifecycle Management (ALM) systems
- Several open source systems
- *Can track status of **all** test activity*

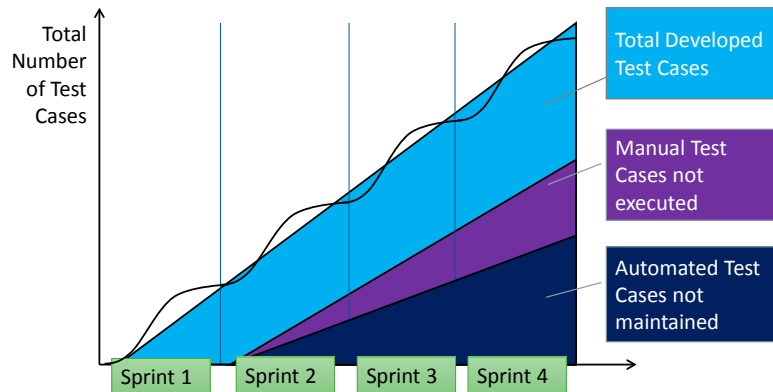


**020 Who develop test automation be the same person who develops the code?

REMEMBER: No Automation? The Backblob's gonna getcha!

REMEMBER: No Automation? *The Backblob's gonna getcha!*

- The extent of manual testing is limited to the capacity of testers
- The extent of automated testing is limited to the capacity of test scripters
- Total number of tests increases as project progresses
- Typically, only the newest features are tested



**021 Presenter: I think there are pros and cons on that. They can be. Typically an SDET is somebody who focuses primarily on producing test automation. Oftentimes in larger project teams, the SDETs will have a close working relationship with the developers, where the developers in a sense are their customers, where they are looking-- the developers may be looking to the SDETs to create and establish an environment and perhaps some specialized type of tooling that's application-specific.

Presenter: I mean, the case I've seen that's closest to this is where I develop code and I develop tests but I don't develop the tests for my code; I develop the test for Bob's code. So I'm still not developing-- it's not that fox in the henhouse where, "Oh, I know how the code works, so I'll

write the test that makes it pass."
But it's more that sometimes-- I think in terms of-- I know some people that are just really, really good algorithm developers in terms of bringing mathematical algorithms into code, and you want them doing that, and they also are very good at writing automated tests, but you don't want them to write the ones for the algorithms that they put in. That's really the boundary that I've seen.

Presenter: Yeah, and oftentimes SDETs kind of live in that middle world between unit-level testing and system testing, where system testing is driven by subject matter experts and unit-level testing is driven by developers. They're the ones that have the technological chops to actually grab hold of those parts of the system and drive them through a particular test plan automatically when that might be technically challenging for others to do who don't really have that job of integration and the sort of technical support to do that.

Presenter: And I think the place where you're going to see the temptation for that to be-- where you have the code and the SDET in the same person is in really small teams, where you don't have a large team, and in that setting, one of the things I've seen is just using a different team; "I'm the SDET for your team, you're the SDET for my team" is another way of dealing with that sort of resource-constrained environment

when you're looking at that kind of an issue.

Presenter: Yeah, this has been a classic debate within the testing community for probably 50 years.

Presenter: At least.

Presenter: As to whether or not you should write your own test code, the philosophies on that have varied, and so I think-- I'll just say it depends.

Presenter: There you go. There you go.

Presenter: How's that for a great answer?

Presenter: So one more relevant to this section, if you don't mind. It says: Where the unit-test integration and system testing is automated, are organizations seeing significantly shorter integration and test phases and lower defect densities?

Presenter: I think other things being equal, yes.

Presenter: The thing I'll say is not-- I would say in my experience, not necessarily lower defect density initially, because you now also have the defects related to the tests. So at the beginning, I've seen a mix of maybe not so much defect density in the code, but I also have to deal with maintaining and making sure that the tests give me the right information. So there's a defect potential in the

automated tests as well. I think the thing that I can say with a lot of confidence is that automation is freeing resources to do more analysis, and so when I do have defects and when I do have issues and risks coming up, the people that normally would be taken up with trying to fix a lot of little defects, they're freed up because that stuff is coming through faster; it's coming through when it's easy to fix. When I only have one to two weeks of code that I've put into a code base, I don't have as many opportunities for defects as when I'm putting a month or three months or six months or a year's worth of code in. So it's a different cadence, maybe?

Presenter: Cadence-- yeah, sure.

Presenter: A different cadence, and the size of effort related to actually dealing with the defects seems to be less, is what I'm seeing.

Presenter: Yeah, it's a very interesting question, and it's hard to give a general answer. This is-- there are a lot of factors that determine what drives defect density. Test is certainly a big one. It's not the only one though.

Presenter: Right, right.

Presenter: All right, so let's move on to the tool chain question.

What kind of tool chain do I need to support automated testing?



Five keys to effective Agile test automation for Government programs

What kind of tool chain do I need to support automated testing?



Software Engineering Institute | Carnegie Mellon University

Five keys to effective Agile test automation for Government programs
June 23, 2017
© 2017 Carnegie Mellon University

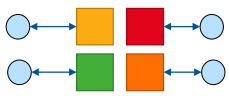
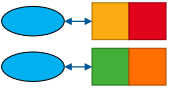

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

**022 What kind of tool chain do I need to support automated testing?

Testing scope, lanes, focus, tooling

What kind of tool chain do I need?

Testing scope, lanes, focus, tooling

Scope		OEM/Dev	PO/FAT	DTO	OTO	Focus	Example Tool
Unit, CSCI		✓				Functions	<ul style="list-style-type: none"> JUnit SonarCube
Component, Subsystem		✓	✓			Features, User stories	<ul style="list-style-type: none"> Selenium SOAP UI
System, SoS			✓	✓	✓	Use cases, Performance, Mission threads,	<ul style="list-style-type: none"> Jmeter

**023 Presenter: And there's people on here that have been waiting for half an hour already just to-- because they wanted to hear about tools. We've been talking about all this other stuff, so now they get their fix.

Presenter: Now we're getting into the real interesting stuff, depending on your perspective. So let's just try to do a little level-setting here in terms of testing. Testing, by the way, is not a monolithic activity. There are many different kinds of testing, many that depend on certain factors for many purposes, and that also drives variation in the kinds of tools that make sense and the kinds of tools that are available.

So first of all you have testing at the so-called unit level, or what's

sometimes called the Computer Software Configuration Item level, and that's a focus on a small part of the system and its functionality. If you start to put those together in some useful way, that gives us integration testing of components and subsystems, and then at the system external level, we have an entire system and perhaps its interoperation with other systems-- so-called system of system testing-- we then have a different kind of testing problem.

Presenter: And that testing problem-- I like the fact that you put the black boxes in there, because that's really what you're getting, is when you're doing system of systems, you may know the interfaces and that's about it. You really don't have a lot of information. So that kind of testing is a really complex type of testing.

Presenter: It definitely has its own challenges. We'll mention a few minutes some of the ways in which automation can help with that.

The other one is lanes or sort of who does what and when. Here's a very rough guide to that. Of course there are many possible exceptions to this, but essentially kind of the smaller-scope testing unit and component tends to be done by development. Development also will typically do some kind of system testing. Often times the main focus of that will be on the program office and their so-called factory acceptance testing.

And then system testing for developmental test organizations and operational test organizations typically focus on the external factors and integration with external systems. So focus is on functions, computable types of things; others on features and user stories. Use cases, performance, and mission threads are at the sort of highest level.

Here are some example tools. We'll talk a little bit about these later. Since I have called these out, I do want to say that these are just examples and we're not endorsing anything. These are some of the more popular tools in these Categories. The point here is that there are many forms and purposes of testing and they each have their own kind of automated support.

Tools for Test Automation

What kind of tool chain do I need?

Tools for Test Automation

- There are *hundreds* of COTS, FOSS, and GOTS software testing tools



**024 So what kind of tools are there out there? Well, there are hundreds-- commercial, off-the-shelf, free open source and government off-the-shelf software testing tools. It really is quite a lot, and I think, SuZ, you'd commented previously on how this is different from times that you looked at this in other circumstances.

Presenter: So, yeah, I was telling Bob that the first time that I actually looked at automated tools was back in 1988 when I worked at Lockheed, and not very many vendors were actually competing for our business, and we didn't buy anything at that point because what you could get out of automated testing, it required a huge investment and you really weren't getting a lot at that point, and the automation was-- we talk

about some of the complexity of managing the assets and things; it was actually a lot worse at that point. I've observed that since Agile came into play as a dominant force in commercial industry, that's when I saw a real uptake in the number and the quality of automated testing tools, because it just is such a necessity in Agile to be able to have that fast feedback, and fast is not a day many times, it's minutes, and you can't do that without automation. So that's a big difference. We talk about a lot of things are the same as they used to be back in the '80s. This is one of the things that's really different, is what you can achieve and what's available to you in this setting.

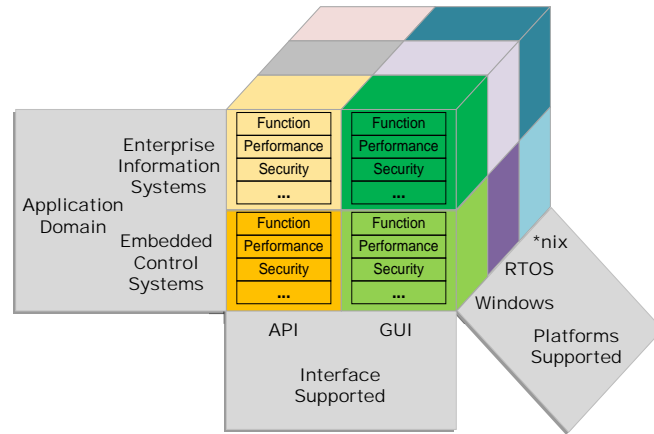
Presenter: Right, yeah, and so it's changed really very substantially, and if you've even been looking at test automation, say, ten years ago or five years ago, I think you might be surprised if you kind of take another look. There are lots of new capabilities and lots of stuff that really, frankly, is quite impressive. Things that I personally hands-on struggled with for months for years in trying to automate in previous situations are now essentially a drop-in that you can get for free and really are quite impressive.

Tools for Test Automation

What kind of tool chain do I need?

Tools for Test Automation

- There are *hundreds* of COTS, FOSS, and GOTS software testing tools
- Each tool is specialized for a certain kind of testing
- Each tool is specialized for a tool stack, target stack, and target interface



**025 One of the reasons that there are so many tools is that there are so many ecological niches, you might say, for the different kinds of tools. So you can sort this by application domain, whether you're supporting doing testing of a large enterprise transaction processing system, or whether you're doing some kind of embedded control or communication system. And then you can look at what is the main interface that you're trying to drive during test, whether it's a programmatic one or whether you're interacting with a user interface.

And then the platform that the tool runs on and the kind of application that it tests its platform are also variables in all of this. So this is kind of a Rubik's Cube, and that's how you get the hundreds of possible testing

tools, because each one is specialized.
You can slice this further
according to what sort of test
purpose it supports, whether that's
function, performance or security
testing, and there are other
specializations as well.

So I guess the takeaway from this is
when you start to look at a tool, just
be aware that you really have to know the
answers to these questions in order
to pick the tool that makes sense
for your environment.

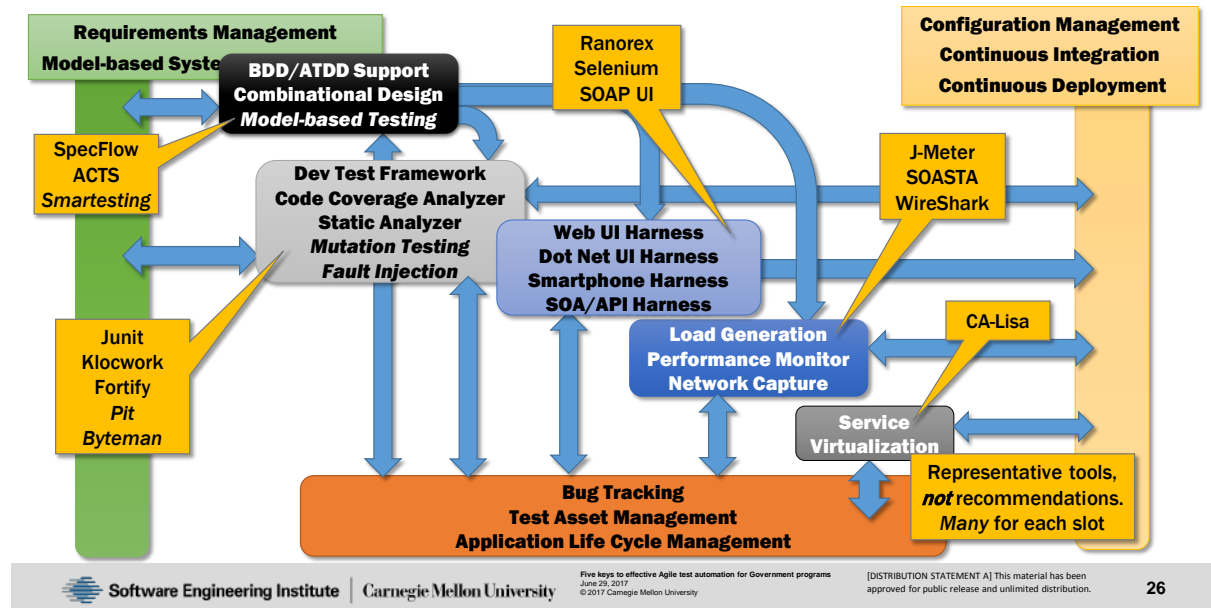
Presenter: As you said, if I'm doing
an enterprise payroll system, that's
going to give me a whole different
set of answers than if I'm doing a
robotics manufacturing line.

Presenter: Absolutely, yes.

Test Automation Reference Architecture

What kind of tool chain do I need?

Test Automation Reference Architecture



**026 So let's now take a look at a reference architecture for the categories of tools and make sense of how these things fit together. So again, I want to emphasize we're going to mention some representative tools, they're not recommendations, and there are many possibilities for each particular slot.

So first of all, let's look at the basic environment for a typical test automation tool chain. This starts with requirements management, model-based systems engineering, or, in the Agile space, something like VersionOne, which manages user stories and all of the derivatives of that. The other part of this that's typically in evidence today is configuration management,

continuous integration, and continuous deployment. Those tools don't do testing per se, but they are absolutely critical for automated testing.

Another basis for this, the data management if you will, bug tracking, test asset management, and application lifecycle management tools, of which there are quite a few.

All right, now we can start to ask the question: What's the test automation tool chain that lives within environment? At the first layer are black box tools and support. In the Agile space, we see tools for behavior-driven development or "BDD" and acceptance-test-driven development or "ATDD." There is some interesting products in this space. Combinational test design is something I always try to mention because it is such a simple test concept but yet it's so powerful. And then there's a specialized kind of test generation called model-based testing, which is a topic for another day.

Presenter: And it kind of goes along with model-based engineering. As model-based engineering becomes more prevalent, model-based testing becomes more possible.

Presenter: That's right. Model-based testing is essentially the automated production of tests, where instead of having a person handcraft them, the tests are produced from a model.

So here are three examples. SpecFlow supports behavior-

driven development for the Windows platform. ACTS is a government, off-the-shelf testing tool provided by NIST at no charge. It does combinational test design. And Smartesting is an example of a model-based testing tool.

Now, let me remark about the blue arrows. The blue arrows indicate that all of these tools have interfaces potentially amongst each other, and whether or not that interface is supported automatically or whether you do that through some kind of knowledge that you transfer among your team or people who might be using these, there is an interface and that's something that's part of the picture of a test automation tool chain.

Then if we look at the kind of gray box, are tools that are sort of oriented towards code level or developer level. We have developer test framework, code coverage analyzers, static analyzers, and then sort of in the advanced category you have something that's called mutation testing, and then fault injection. Examples of this, and probably one of the most well-known ones for developer test framework is something called Junit, and it's very widely used for Java development, and it was one of the original of this particular type of test framework. It's open source, it's free, it's relatively easy to use. It's very popular.

Klocwork is an example of a commercial off-the-shelf tool that

supports coverage analysis. Fortify is a tool which is used for static analysis related to cyber vulnerabilities and is rather popular.

Presenter: Security based, yeah.

Presenter: Pit is a cloud service that does mutation testing for Java. Byteman is one which is also for a Java development environment that does fault injection in an interesting way.

The next group of tools drives the system under test at an intermediate level or system level. So there are many tools that drive web browser tests if that's the user interface. Selenium is the most widely used. Ranorex is popular for the Windows/ dot Net platform. There are many tools for various Unix, Linux platforms as well.

The mobile application development environment (mainly Smartphones) has quite a few interesting testing products in it. This is still challenging due to all the variations in form factor and operating systems, but there are some useful tools and services that support automated testing over all of these variations.

And then for service-oriented architectures and APIs, especially in the last several years, there's been a lot of new stuff that's been released for that, especially if you're doing REST APIs. There's a lot of very good stuff for that. SOAP UI from SmartBear is one example.

To testing systems at scale to make sure that they perform adequately

under load is a critical activity and really can only be done if you have test automation.

Presenter: Even back in the '80s, this was the first place that we were looking at using automated testing, was to be able to create large batches of data that we could pass through and look at performance, and that was hard even then. Today that's really one of the best-implemented areas for automated testing.

Presenter: Yeah, for test automation, this is really I'd say a pretty mature technology. There's some very interesting stuff in terms of generating this through scalable cloud resources. And then network capture.

All right, then one final category that's not as well known is something called service virtualization, and service virtualization is the idea that we generate not just sort of-- if you're a developer, you may be familiar with the idea of a stub. So if you're writing a piece of code and you have a dependency on another one, you might write something that's kind of a temporary placeholder for that other one. Service virtualization is a similar idea, but instead of just a block of code, we're producing a stand-in for another entire system.

Presenter: Gotcha.

Presenter: And so this is very useful when we have interoperability concerns with other systems that we have to work with, but for example, we were really limited in terms of testing because they have production sensitivities that can occur for many reasons.

So this is kind of the landscape, broadly speaking, of the functional categories and some examples of tools that fit into each one of those. And again, I want to say the noted examples are representative, they're not recommendations, and there are many, many, many tools for each of these categories.

Presenter: But I just want to come back to the testing is development. I know I'm harping on that. This is why testing is development, this world. If I'm doing automated testing, this is not just a single, "Oh, I'm just going to develop a little thing." This has got some complexity to it. A lot of decisions-- architectural decisions, implementation-- what am I going to use, where am I going to focus, what am I going to do first? Am I going to do load generation first? Am I going to do-- all of those are system kinds of decisions and so we have to look at it this way if we're going to be productive in this environment.

Presenter: That's right, yeah. And so getting all these things spun up and working, it's part of the reason that we say that this is a substantial effort and should be approached with

that in mind. It certainly is doable, but putting together and integrating all of these is a nontrivial effort.

Presenter: I will give you one little anecdote on the much simpler scale. So this is an organization-- a government organization that adopted Agile very early, 2003 timeframe. No automated testing, and they worked for a year and actually got to where they had a pretty good cadence of delivery and all the rest of it, and after about a year they had to stop the development and put up a test harness just for the beginning levels of their automated testing because it had gotten to where it took two weeks, which was the space of their iteration, to run the regression tests, and they went, "Okay, this is not going to fly." It's the back-blob, right?

Presenter: The back-blob got them.

Presenter: It got them. But they recognized it early enough, and then they took a strategic pause, they called it, for three months, got things--

Presenter: Good idea.

Presenter: Got this going, and then from then on they were managing this very actively. But this is reality that this is a job not to underestimate.

Presenter: Right. Yeah. So let's move on our next poll on this.

Poll 3

What kind of tool chain do I need?

Poll 3



What kinds of testing are automated in your project?

- A. Unit/CSCI testing
- B. User interface functional testing (desktop, mobile, web page)
- C. APIs, web services
- D. Software/Hardware in-the-loop testing
- E. Performance testing
- F. Robustness testing
- G. Other



Software Engineering Institute | Carnegie Mellon University

Five keys to effective Agile test automation for Government programs
June 29, 2017
© 2017 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

27

**027 So Shane?

Presenter: So the poll is up now, and the question is: What kinds of testing are automated in your project? And I think we had another question in the queue while we're waiting for that one. This one was from a little bit earlier which I missed, but from Virginia asking: Is requirement selloff embedded as part of the Agile development test or separate event before fielding?

Presenter: Yeah, I know Bob's looking at me, he's going, "What?" So requirement selloff in the government has a very important meaning, because requirement selloff is the point at which the government is taking ownership of the product that the requirement is related to, and that's really when contractors, in

a contract environment, that's when they really get paid. So very important construct for people is requirement selloff. What we're finding is a variety of approaches to that. So there's something that's often called FQT, which is a full qualification test, or factory qualification test-- I've heard it used both ways-- is the traditional time when requirements are sold off in some settings, and that may not change.

So that could be something that still happens at the end. I will tell you that one of the heuristics about requirement selloff is that there has to be a level of confidence on both the developer and the receiver organization that, "We're really done with this part of the system," that we are not iterating anymore on it. And so you typically will not see requirement signoff in the earliest iterations in integrations. It's going to happen later. But how it happens is very idiosyncratic to the type of organization, to the type of acquisition. Are we kind of a standalone thing, or are we part of a larger system where it has its own requirement selloff practices and we have to be part of those? So, as usual, it's not a single answer. I didn't say "It depends" until now.

Presenter: Right, and we had some comments from the polling question just saying that they wish it was multiple choice because there are some multiple--

Presenter: Oh, okay. Gotcha.

Presenter: Which was a limitation of the system, but yeah.

Presenter: Okay, sorry about that.

Presenter: But to close the results here, we had 44 percent with unit testing, 17 percent user interface functional, 17 percent APIs, web services, 5 percent software/hardware, 10 percent performance testing, and 7 percent other.

Presenter: I'm betting that some of that is because we didn't allow multiple. You'll have to fix that with your-- put a change request into the webcast people.

Presenter: There you go.

Presenter: But I mean, I think it's-- unit CSCI testing is a place that many people start, but I would also say that performance testing is the other place I've seen people start. So that one's probably a little lower because of the way that the question was set up.

Presenter: Right. That's absolutely right.

What kind of testing should/should not be automated for Agile software development?



Five keys to effective Agile test automation for Government programs

What kind of testing should/should not be automated for Agile software development?



Software Engineering Institute | Carnegie Mellon University

Five keys to effective Agile test automation for Government programs
June 29, 2017
© 2017 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

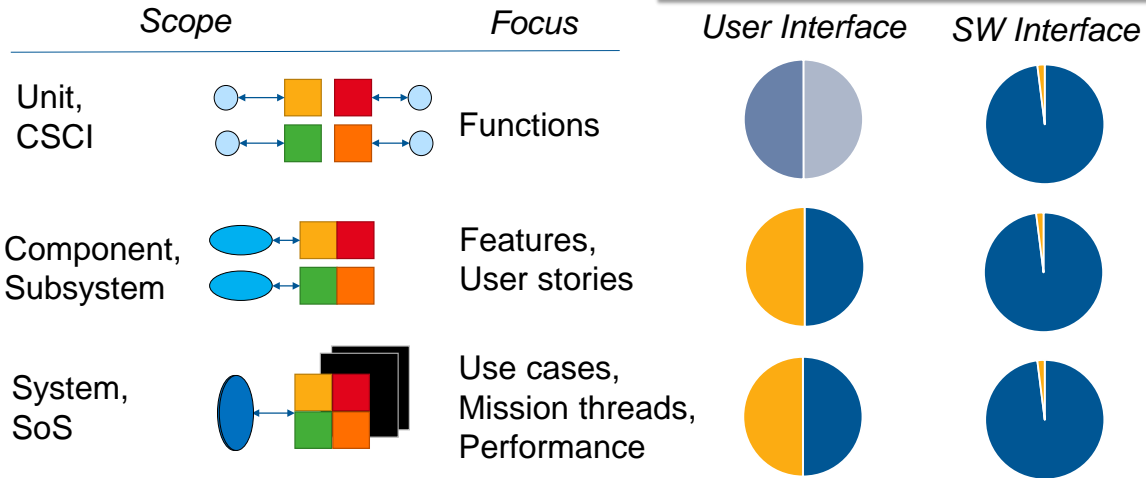
**028 So I think we're on now to the sort of last part of the discussion. Do we have any guidance about shoulds and should-nots for automating development?

Testing scope, focus, automation

What kind of testing should/should not be automated for Agile software development?

Testing scope, focus, automation

Notional proportion of automated and manual test cases



**029 So I'm going to go back and kind of reprise this slide again. Here are-- we're trying to answer the question here: How much of your testing really should be automated? And the answer here is: Let's take a look at what type of testing we're talking about. If we're talking about software components themselves that really don't have a user interface, that pretty much has to be, by definition, almost entirely automated.

Presenter: And it's feasible for it to be automated.

Presenter: And it is feasible. That's a fairly straightforward and well-solved problem. If we're talking about component subsystems, it depends on what type of interface those components or subsystems

have. If it's something where the user interacts with it, then that can be manual and that can also be automated, but we don't want to automate everything necessarily right out of the box. We'll have some suggestions on that consideration later.

If we're talking about a subsystem that has, again, a primarily programmatic interface, again, for the same reason, that should be mostly automated.

If we come and look at the system of systems or systems level, it's pretty much the same story as it is on the subsystem level. Of course some of the considerations here are different, but to the extent that it makes sense to automate some of those repeatable interactions, we should do that, but we should also reserve some effort for interaction that is driven by experienced testers. And again, if the system is one which primarily has technology interfaces as opposed to user interfaces, those can be driven automatically.

Presenter: One of the heuristics I've heard is that if I'm going to write an emulator for it, I might as well automate the testing for it, because I'm already using software that I'm going to know about, I'm going to know some characteristics of, so I can use that to help me with the automation. And that goes to the software interface type of thing.

Presenter: Right. Yep.

Avoid test automation gotchas

What kind of testing should/should not be automated for Agile software development?

Avoid test automation gotchas

- Much more work converting manual to automated testing than expected
- Don't automate tests for unstable interfaces
- Do use exploratory testing
- Do automate tests for APIs, stable user interfaces
- Tools don't automate test design and judgement
- Capture/replay usually results in breakage and/or test script re-recording
- Set expectations for test asset maintenance
- Automate performance testing incrementally
- Follow test automation design patterns



**030 All right. So, here are some gotchas. We're going to suggest that these are the kinds of things that we've seen go wrong in test automation efforts and something maybe to keep in mind is this sort of a risk checklist. I've often seen that the amount of work involved in converting from a manual, procedural approach to testing to an automated one can be surprisingly large, and that's because the persons who are doing the manual testing actually have a substantial amount of domain knowledge, expertise that they've accumulated perhaps over a long period of time that really is not captured in the written form of the test.

Presenter: We call that unconscious competence. There's things that you know that you don't

even realize that you know, and so you don't write them down, because you don't need to.

Presenter: Exactly. But those are--

Presenter: But the computer doesn't know those things.

Presenter: It's teasing those things out and figuring out what they are that can be very time-consuming.

Another area where people often get snagged is in automating tests for unstable interfaces, and unstable means not that they're bad, that they're kind of flaky, but that they're likely to change rapidly, and so you haven't really completely figured it out; they're subject to change. That's the kind of where usually-- not always-- but it usually doesn't make sense to automate tests for those.

Exploratory testing is where you have a kind of testing that is guided by a principle or a general plan of attack, but not by a specific procedure. That's very effective, especially when you have evolving interfaces. And as we've noted, things that have technology interfaces and stable user interfaces are good candidates for automated testing.

And of course tools don't automate test design and judgment. They're not a replacement for a good understanding of the system under test and then deciding how to exercise it.

If you're doing test automation of user interfaces-- this might be a Windows desktop app or a browser-- one mode of doing that is what's called capture replay, and most of the tools make it pretty easy to do this. So you just turn on a recorder, you interact with the system following a plan of some kind, and this gets recorded and can be played back, and that's capture and replay. So that's good for some things, but in terms of using that as a basis to do test automation is not usually a good idea, because if anything at all changes in the user interface, whether it's new functionality or new design or layout, that test is probably going to be broken and will have to be recaptured or redone. So a capture replay is sort of like a sugar high. It feels good for little while but then you get a sort of unpleasant after-effect pretty soon on.

Presenter: You get the hangover.
You get the capture replay hangover.

Presenter: That's right. So avoid that. And the way to do that is through-- there are other strategies that take a little bit more in the way of software architecture for tests that are pretty good at preventing that type of problem.

Also, test assets are not kind of fire-and-forget, one-and-done. They're like other kinds of software-- they need to be maintained, kept up to date in terms of functionality and interfaces.

One of the things we see, that people may defer on performance testing until they get the entire system in place, which makes a lot of sense, but it usually is smarter to do performance testing incrementally, and try to build to it as you go rather than waiting and doing it all at the end.

Presenter: And I'll say here that in cases-- legacy systems, a lot of times you have a baseline of performance and you're trying to either maintain that performance level or increase performance, and if you start your performance testing early, you can find out if you've already hit those boundaries. I mean, that's one of the things about performance testing-- when you make changes, you don't realize a lot of times how you're changing the way the resources are used.

Presenter: That's true.

Presenter: And if I've got an early indication that I've broken some performance boundary, I can deal with it, I have time to deal with it. If I wait until the end to find that out, then it's a much harder recovery.

Presenter: Right. Yeah. So get those things going soon. And then, kind of related to the test asset and all these other things, are to learn about and apply test automation design patterns. That's a longer story, but perhaps someday we'll have a webinar on that.

Presenter: Sure.

Poll 4

What kind of testing should/should not be automated for Agile software development?

Poll 4



What kinds of testing are routinely done in your project?

- A. All manual
- B. Mostly manual, some automated
- C. About same manual and automated
- D. Mostly automated, some manual
- E. All automated

**031 Presenter: Okay, poll number four.

Presenter: Okay, that is launched now and it's asking: What kinds of testing are routinely done in your project? And while they're voting on that one, we had another question, asking: What is the effort estimation techniques used?

Presenter: Oh, for doing test asset design and management?

Presenter: Yeah. Yes, I believe that's--

Presenter: In the Agile settings that I've been working in, they actually use relative estimation, similar to they do in the rest of the project. So they'll use story points to do complexity estimation. I've not

seen-- there's an idea of value points that kind of comes into play for software for understanding the business value. I haven't seen that used in the test community, but I can also imagine where that could be useful. For more traditional automation, I've also seen just traditional, "This is how many hours it's going to take." I saw once place- I don't know if you've even seen this- I've seen one place that had a heuristic of, "If this is my effort for developing this code," then they had a percentage that they applied. "It's going to take this percentage of that to develop the testing automation." I honestly don't remember what the percentage was, but that was their going-in position. I don't know if you've ever seen that before. I've only seen it once.

Presenter: Actually, there are variations on that theme, and I think it's probably not a bad high-level estimator. If you follow what we're saying about Test is Development, Development is Test, you can make the case that about 50 percent of your effort ought to be test. Some people might find that to be excessive, but--

Presenter: Actually there's research that says the opposite, that it's higher than 50 percent by the time you're done.

Presenter: Right. So a high level is not a bad benchmark. Of course every situation is different. The methods for doing estimation I think for tests,

at least in my experience, are not all that different than they are for doing software development, or at least doing test automation. It's a similar kind of problem.

Presenter: And the other thing, when you're getting started-- I mean, we do this in software development as well-- do some prototyping and record your actuals. Figure out what's the difference in writing a test automation script for a user interface task versus a software interface and collect some of that data so that you have it to use in your estimation of the larger system.

Presenter: Yeah.

Presenter: All right, to wrap up our question, we had 18 percent all manual, 45 percent mostly manual, 16 percent about some manual and automated, 18 percent mostly automated, and 2 percent all.

Presenter: Ooh.

Presenter: Oh hey, we want to talk to that 2 percent.

Presenter: I bet they're not in government. But still, I mean, even if you're not-- not all our audience is government. A lot of our government people are saying there's no way that anybody could ever do all automated. We got 2 percent that say they are, so.

Presenter: Well, it depends, and if you have a system with a lot of user

interfaces, with a lot of user interaction, oftentimes there are things that it doesn't make sense to automate. There are some kind of unusual or complicated interactions that don't occur very often and maybe on parts of the system that you're not that interested in, for one reason or another, and perhaps just the effort that it would take to create usable automation is just too much.

Summary



Five keys to effective Agile test automation for Government programs

Summary

 Software Engineering Institute | Carnegie Mellon University

Five keys to effective Agile test automation for Government programs
June 29, 2017
© 2017 Carnegie Mellon University

(DISTRIBUTION STATEMENT A) This material has been approved for public release and unlimited distribution.

**032 All right, so let's try to wrap up.

No magic!

Summary

No Magic!

When done skillfully, test automation can deliver strong benefits to support organizations in delivering high-quality software at a faster pace.

When done badly, it creates more problems than it solves.

Sobejana and Herschmann, "The Eight Essentials When Moving to Automated Software Testing," Gartner Inc., 2016.



**033 I guess first, this is no magic. Test automation is not magic. It's like a lot of things in software engineering; it has high potential for payoff but it also takes a pretty conscientious effort to achieve that result. So here's a quote from the Gartner group on this. I won't read it, but I think the bottom one probably bears repeating. If done badly, it creates more problems than it solves, and I will underscore that point. That's been my experience in test automation. So if you're not sure about how to get going in this, tread lightly, take it a little bit at a time, and build your confidence in it. It's not that you have to be scared about it, but just have an understanding that it can be a challenging effort.

Presenter: Go in with your eyes open.

Presenter: Yeah.

Reasons Automated Test Investment is Often Delayed in Agile Adoption

Summary

Reasons Automated Test Investment is Often Delayed in Agile Adoption

- Testers aren't included in initial adoption strategy
- Difficulties in obtaining required tooling to support
- Some can be bought, but some has to be built from scratch
- Training test personnel in tooling/approach while executing their normal activities
- Hard to pause ongoing work to establish a new capability going vs build it piecemeal
- Organizational challenges related to dev/test structures
- Using automated mechanisms to do things that used to require human governance
- This isn't just "throwing a switch"
- Evolutionary implementation means incremental utility, so monetary benefit won't be seen right away, even though risk reduction might be



**034 SuZ, I'm going to let you do this one.

Presenter: Yeah, so one of the things that we've been-- we've worked with a lot of organizations in the government that are just getting started in their Agile transformation, and when we teach about Agile, we actually encourage people to consider automated testing early. I mean, I hope after seeing all the things we've talked about that people can see why we would recommend that as an early decision. But it is often delayed. It's like I'll teach a group and then come back two months later and they haven't even started

on this. It's like, "You're not talking about this yet." Some of these are reasons that we hear and reasons that we see. If you don't have testers included in your Agile adoption strategy, then they're not going to get a voice and they're not going to necessarily know there's a different way that they could be doing things.

There are some difficulties in obtaining required tooling in some environments. Some places-- open source may have lots of offerings but they don't really have the potential for actually getting to it. So that's one of the things that you have to look at, or budget. I mean, there's a budgeting aspect. "I have to buy licenses," all that kind of stuff. So that's one of the reason that people delay it. Some areas, in some of our very specialized domains, you can't actually get the tooling that you need. You have to develop from scratch, and so that adds a burden to this.

And test personnel, we've said earlier, tend to be a very scarce resource. Just making them available for training for new tooling approaches can be very challenging because they're trying to get through-- they've already got testing underway for the last system, and so getting ready for the next system can be very challenging. So given them the time available to do the training that's needed is a barrier, and even just pausing ongoing work to establish the new capability.

This is-- the example that I gave earlier, that they solved this by saying, "Okay, everybody's going on to automated testing." All the developers, everybody was on that strategic pause, because they couldn't just do it incrementally. They had to figure out how to get it going. And so there's some different decisions there.

There are some organizational challenges related to development and test structures. I said that so politely. There are times when there's political battles between the resources that go into the development shop and the test shop, and when I can have access to a lab, and all those kinds of challenges can make it harder for people to approach the idea of automated testing. And when we use automated mechanisms to do things that used to require human governance-- that example I gave about the sustainment group that got the OT&E folks to say, "Yes, if you pass these tests, you can go forward," they were automating governance. That's a decision that a human used to make, to look, "Can I go forward?" That was a really big deal.

Human governance is what we're accustomed to putting our confidence in, and that shift of saying, "Well you know, since we're doing all this automation, we could just push it out," and this is-- we haven't talked very much about dev-ops, but dev-ops is a place where this is

happening, and bringing test into that is one of the ways that you make that governance able to be dealt with in an automated fashion.

Once people learn-- so there's probably a subset of people that are attending this webinar that are going, "Holy cow, this is going to be too hard. This is not throwing a subject. I thought this was going to be just buy-- 'Here, I'm going to buy the SEI test reference architecture and all will be well,'" and once they find that out, they can't approach it; they've got all these other barriers. So that will happen.

And the monetary benefits-- so when I go to you as a manager and say, "I want to buy this many thousands of dollars in test licenses, test tooling licenses, and I want everybody to have an application lifecycle management tool and I want all my testers to have a test asset management, and I need to train them in all this stuff, and I need to take a subset of the testers and redeploy them as SDETs," and the manager is going to say, "Okay, what do I get for that? What's the ROI? What's the return on investment for that?" And most of the evolutionary implementation that we see means incremental utility. So I see risk reduction very early, but that's not a monetary benefit, and so some people that need the monetary benefit, they can't get approval because that reduction in cost is not going to happen until much later.

So these are all things that you may run into or you may go, "Yep, that's me," and so these are all reasons that we've seen that that delay occurs, and that's part of what we work with people on, is how to get past some of these barriers, because if you never get started, you'll never get finished, and that's-- I know that's a truism, but I mean, it's a truism because it's true.

Presenter: It's true. There's no arguing with it.

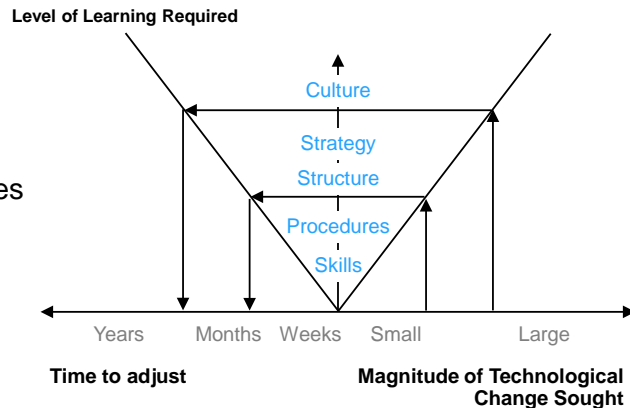
Presenter: And this is enough of a job that you want to get started early. So that's the thing there.

Remember: Automated Testing = Structure and Strategy Change

Summary

Remember: Automated Testing = Structure and Strategy Change

- Scope of change is similar to automating a manual business or operational process
- Requires many behavioral changes
- Not plug-and-play
- Learning curve



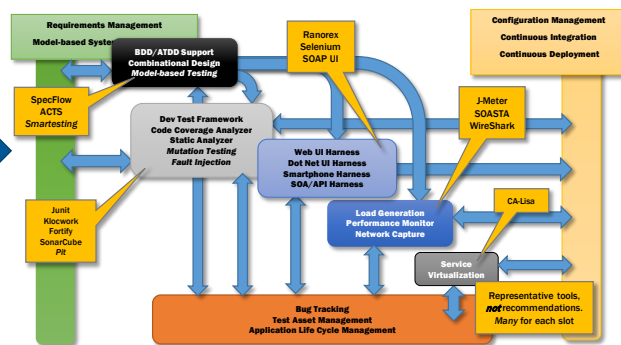
**035 And just a reminder that the scope of change is large enough that this is going to require more than just a, "Okay, let's put it on this year's

Goals and Objectives list, and at the end of the year we'll have automated testing." It's more than that.

Start early, keep at It!

Summary

Start early, keep at It!



Modernizing legacy testing is just like modernizing legacy operations!

**036 So start early, keep at it, and there you go, and you'll get there. You'll get to the reference architecture implemented in your space. You don't want the reference architecture. You want the reference architecture implemented in your space. But this is like modernizing anything else, and so you got to keep going. Final words?

Presenter: I think-- yeah, so what's the tradeoff here? If you don't do this, you leave a lot of opportunities on the table. I think there's pretty uniform agreement of people who have put the effort in to establishing this kind of capability within their development organizations that it

pays off, but we mentioned some of the caveats that you have to pay attention to, to make sure you get the desired result.

With that, the only other comment that I'd make about the reference architecture is: By the way, you do not have to have every one of those; you don't have to check every box in that picture. It depends, again, on your situation. So it's not a "must do all of these", but these are the things that are available.

Presenter: And you have to prioritize your backlog just like you do any other development, right?

Presenter: Certainly. But as SuZ says, if you don't start, you won't finish.

Presenter: Before we get into the final Q&A section with Bob and SuZ, we had a couple questions just asking about the archive of today's presentation. That will be available by no later than tomorrow morning. We'll send out an email to everybody. It's the same URL for registration that was used today. You log in with your email and you'll be able to watch that tomorrow morning.

Presenter: And we're going to be on YouTube at some point, right?

Presenter: Yeah, we'll be on YouTube. It will be on the SEI website as well. We do ask that you fill out the survey today upon exiting today's event, and I'm going to push that survey live now. You can just

minimize it until we close out, but we do ask that you fill out that survey because your feedback is always greatly appreciated. And then let's get into a question from Martin, asking: Is there a way to swag the cost slash manpower for converting a legacy system with no unit test to one with a sufficient number of unit tests? This would need to include some factor of how complex the code would be to break apart so it could be unit-tested. And let me know if you need a repeat.

Presenter: That's a great question. This is actually an issue that I've been working on myself, and I think-- it's hard to provide a swag answer. I think that you might-- one, you might get some very rough guidelines by asking how many functions there are, how many files. So if it's a system composed of something like C or C++, the number of files usually is a rough indicator, and you might ask yourself if a typical file has ten functions in it or ten methods, I want to write one test for each one of those, or do I want to write five, on average?

Presenter: Composite, yeah.

Presenter: And how long will it take me to do those? And so then it's just a matter of running that arithmetic. It's of course a very rough approximation of what it will take. There are so many variables involved in that, it is difficult to give you more than just kind of a general outline for that. SuZ, any ideas?

Presenter: The only one thing I'll say about that kind of a shift is, from a manpower viewpoint, if you are talking about the people that know the system intimately, I'm going to give you a heuristic that that's about half as much as if you're bringing in new people that have to learn the system. It's a much bigger lift, if it's people that know the system, know the domain, know the software, than if you're bringing in new people, and a lot of these legacy systems, you're bringing in new people because it's been sitting around. So it's going to be more work depending on what your situation is. But I agree with Bob's advice about-- you're going to have to get granular with the estimation on this to get any kind of an idea of what you're up against.

Presenter: So there's one other question in the queue and it's from a little bit earlier, and it's about CMMI, or mentions CMMI, so I'm going to look to you, SuZ, and it's from Jean-Paul asking--

Presenter: I have a past with that.

Presenter: Yeah. In this context, how do you apply CMMI for development for the process areas verification, validation, and product integration? Because CMMI requests many information, plans, etcetera, when do you define this? Into the sprint planning, or sprint zero?

Presenter: So, when you look at-- I'm going to sort of go back into CMMI speak for a minute. Look at

the goals level and there's nothing in the goals level that says that you can't provide the utility of verification and validation incrementally, and so that's what you really want to establish, is that all three-- well actually, if you're doing Agile, not just these three areas but also your architecture requirements are also going to be done incrementally. So your system solution is incremental.

So you want to establish in your master plans, which are one of the things that you would want to be delivering-- what's my strategy? And my strategy is going to be incremental and that means these things are going to be at a high level initially and then they're going to get detailed out as we go along. So I haven't seen a problem with applying those process areas in settings with Agile where the implementers understand Agile, and also eventually, if you do benchmark kinds of appraisals, your appraisers have to understand Agile. If they don't understand Agile, then they're not going to understand how these can happen productivity, incrementally, and you're going to have a whole bunch of other issues besides just verification and validation.

Presenter: Okay, next question: Any technical skills you can recommend that will help someone that is not an SDET but obviously needs to use automated testing? Any technical skills you can recommend that will help someone that is not an

SDOT but obviously needs to use automated testing?

Presenter: You might try to learn one of the more popular scripting languages, such as Python. JavaScript is also used in a lot of tools. I would try to learn just the sort of basic software development techniques that are applicable to doing automated test.

Presenter: And the other skill-- I mean, there's a design skill here.

Presenter: Yes there is. Right.

Presenter: So if you're not already kind of in tune to modular design, decoupling-- I mean, there's principles of design that apply to test automation as well, so getting a good course on software design, actually, would probably help you with the test asset design.

Presenter: Yeah, and there are quite a few resources. There's a number of good books on this subject if you do a web search on test design patterns, or test automation design patterns. I think you'll find a number of useful sources.

Presenter: Okay, so you guys got a couple compliments on a great presentation. Another comment said: We'd love to see a follow-up webinar on automated testing for legacy systems, including estimating, choosing what, how much to go back and automate, architecting, etcetera.

Presenter: I told you this was the beginning of a long set of things you're going to be doing.

Presenter: And then a question from Wayne asking: How sensitive are test script development costs to maturity of automated test system implementation? How sensitive or test script development costs to maturity of automated test system implementation?

Presenter: That's a very interesting question, and I think it-- you know, it's like a lot of other things. You can do a kind of quick-and-dirty test generation by doing so-called capture replay, and I've seen this done plenty of times. You can get lots of test code that looks like-- it's impressive because there's a lot of it, but you then find out fairly soon afterwards that it's really unmaintainable and not highly usable. So if your organization has maturity and the insight to good software engineering practices as they apply to doing test automation, I think you're going to come out ahead coming out of the box.

Presenter: And a heuristic I might think about in terms of looking at that sensitivity is if I look at this reference architecture, it sets up an organization of test assets. Right? If I can't-- if I've got a bunch of test assets but I have no organization-- not necessarily this one-- but I can't really see I have this many of this type, this many that do this, this many that do that-- if I have no visible organization, then I think my

test scripting is going to be very sensitive to changes because I'm not going to be mindful of how changing my test tools is going to change my scripting. And so there's some relationship-- maturity has to do with maturity of the design, not just maturity of how old the tools are and how many times I've used them.

Presenter: That's true. Like a lot of these other things, it's kind of hard to generalize; there's so many various factors that come into play.

Presenter: Okay, and then we'll wrap up with this comment from our friend Bill, saying: As Martin Fowler says, daily builds are for wimps. Thanks for covering great ground and the challenges associated with automation in support of Agile practices. So again, thanks everyone for attending today. Our next webinar will be August 9, and the topic will be "Weaving a Fabric of Trust" by the CERT division chief scientist, Dr. Greg Shannon. So be sure to attend that. And lastly, be sure to fill out that survey. Thanks for attending, everyone. Have a great day.

SEI WEBINAR SERIES | Keeping you informed of the latest solutions

