TEST AUTOMATION

eGuide

TECHWELL™

Between the mainstream adoption of agile practices and the growing DevOps movement, you may have noticed the word "continuous" popping up a lot: continuous integration, continuous testing, continuous delivery, continuous deployment.

In today's continuous-everything, app-centric world, test automation isn't a nice-to-have; it's a must-have. To stay competitive in the marketplace, organizations must automate as many tests as possible to release products as quickly as possible. To maintain an edge in the job market, testers must grow their technical skills to fit into this rapid-release, automated environment.

In this Test Automation eGuide

Bringing the Value of Your Test Automation Efforts Front and Center

Once you've adopted test automation, you should determine whether it's actually yielding the expected benefits—and you'll want to keep these benefits visible to stakeholders to reinforce the value. A metrics dashboard aligned with the organization goals and business objectives shows you're on the right track.

Why Your Test Automation Efforts Should Tackle Data First

Automation projects often start by tackling the technical issues, but Linda Hayes says a specific data environment should be established first. If you can't control, define, and predict your data, you won't have the repeatability that makes test automation practical—but it makes sense for manual testing, too.

Selenium: The Open Source, the Myth, the Legend

Many people wonder what it means that Selenium is open source, and further, what the community element of that paradigm brings to the table. This article addresses some of the common misconceptions about that situation, as well as details some of the benefits of the community behind a product like Selenium.

Automation That Learns: Making Your Computer Work for You

It's been suggested that because automation can only do checking, automation cannot learn. But if you're talking about the acquisition of knowledge through experience and study, Jeremy Carey-Dressler believes automation can, in fact, learn—with a tester adding some additional code to capture and analyze more available data.

The Top 5 Test Automation Mistakes

Test automation is a valuable process, but it can be difficult to integrate into your existing test strategy. Hundreds of people weighed in about the biggest mistakes they've made when automating. Here are five of the most popular answers—and advice on how we can stop repeating these mistakes.

The Buzz on Test Automation

Some of the industry's best and brightest share their thoughts on Test Automation and its role in agile, QA, and the shift to continuous integration and delivery.

Additional Resources for Test Automation

Invaluable resources to keep you, your organization, and your practices at the forefront of the Test Automation movement.

Bringing the Value of Your Test Automation Efforts Front and Center

By Michael Sowers

Sometimes, in order to get everyone on board with adopting test automation, you have to build a business case to highlight the potential benefits. Once you've convinced the organization to make that investment, you should determine whether it's actually yielding the predicted benefits—and you'll want to keep these benefits visible to key stakeholders to reinforce the value.

> *How do we frame the quantitative and qualitative benefits of test automation in a way that links to the organizational objectives and business goals?*

We tend to have many metrics in place to track the progress of testing and measure the degree of readiness or risks in our software products. These metrics include test effectiveness, software quality, test status, resources, issues, and so forth, but what about metrics for the test automation platform? How do we frame the quantitative and qualitative benefits of test automation in a way that links to the organizational objectives and business goals? This is an important element of planning and implementing our automation architecture.

One of the approaches that has worked well for organizations I've worked with is a test automation dashboard. A useful dashboard should reflect values that align with the organization and business goals, be agile and adaptable, and contain information that is actionable and meaningful, with just enough information that it's still easy to use and maintain.

It's important to design the dashboard at the beginning of your automation project in order to ensure that the approaches and tools you choose will yield the measures and metrics you need. Five dimensions will usually cover the range of information that the team and key stakeholders require, but of course you can trim or expand these to meet your needs:

Capabilities include the automation platform features, interface capabilities, integrations, types of systems under tests that are supported, and scripting languages supported.

Usage includes the number of users, teams, and projects; the frequency of use of a given automation feature; the rate of automation adoption; and the frequency of use of test suites.

Benefits are metrics such as rate of unit test, story test, and nonfunctional automation; ease of learning and use of the automation platform; scripting or execution efficiency; degree of automation suite code coverage; speed and efficiency of the test automation infrastructure; hours saved versus manual testing; number of additional test cycles achieved per sprint; percent of increased requirements, functional or nonfunctional testing, or structural coverage achieved; and how much earlier defects are found.

Challenges capture the constraints or issues with the tools or automation platform.

Investment encompasses the costs (in time or money), such as defects in the automation platform, licenses, platform maintenance, training and retraining, operation (including hosting, servers, etc.), the time to build, maintain, and execute automated tests, and the time to investigate defects in the automation test infrastructure or test suites.

> *A useful dashboard should reflect values that align with the organization and business goals, be agile and adaptable, and contain information that is actionable and meaningful.*

The dimensions and associated metrics that are applicable can be displayed in a graphical dashboard format, with simple or more complex measures associated with each "dial." I find trend measures more useful than absolutes.

The critical element of any dashboard is ensuring that it's aligned with the organization's goals and business objectives, and having just enough measures in place to guide the continual refinement of the automation platform. {end}

Why Your Test Automation Efforts Should Tackle Data First

By Linda Hayes

There is no question that automation poses technical challenges; you have to get your tool and application to play nice together before automation is even possible. But the most difficult obstacle to success is more mundane: It's the data. If you can't control, define, and predict the state of the data, you won't have the repeatability that makes automation practical.

> *If you can't control, define, and predict the state of the data, you won't have the repeatability that makes automation practical.*

It starts with the data in the test environment. For a hospital, this includes which rooms are available, what doctors are on staff, and which medications are in inventory. For an airline, this means knowing what routes are flown, which flights are scheduled, and how many seats are available for each trip. You have to control the state of the data when you start execution because your test must know what to expect.

Next comes the data you provide during the test. As you execute, you will create or transform the data: admit patients, book passengers, or make payments. Defining this data is an integral part of developing your test cases. For a manual test, you can adapt on the fly by looking for available rooms or experimenting with different flights, but an automated test must know the values in advance. Trying to adapt during automated execution is theoretically possible, but usually impractical; it leads to complex logic that introduces ambiguity and instability.

Finally, you must be able to predict the result. This is harder than it sounds. You can't get away with saying "verify that the value is

correct" like you may be able to in a manual test; you have to specify precisely what the correct response is. Manual testers have so much tribal knowledge that they are used to winging it, and reducing experience to exact values takes extra thought and effort.

Too often, automation projects start by tackling the technical issues first, and only once execution becomes possible do they start to trip over the issues with the data. In my experience, the data issues should come first because they take longer to resolve and may involve other departments and resources. Subject matter experts, database administrators, network engineers, and hardware support teams may have to be orchestrated before a robust and repeatable solution can emerge. Waiting until you are starting the test cycle is too late.

To those who argue that the technical issues have to be resolved before the data issues matter, I say that having a comprehensive data strategy enables both manual and automated testing. While manual testers can and do adapt on the fly, the truth is that this takes time. The majority of most manual test effort is spent searching for or creating the conditions required for the test. If everyone can rely on a controlled, predictable data environment, then not only will manual testing be far more efficient, but the transition to automation will be significantly less painful.

In the end, if you have resolved the data challenges but find the technical issues are insurmountable, you have still gained productivity. But if you resolve the technical issues first and then cannot address the data issues, you have lost time and effort. It may not be as fun or as satisfying as slaying technical dragons, but addressing the data leads to ultimate success. **{end}**

> *In my experience, the data issues should come first because they take longer to resolve and may involve other departments and resources.*

Selenium: The Open Source, the Myth, the Legend

By Brian Van Stone

When it comes time to select a tool for test automation, most organizations will create a list of candidates and whittle it down based on various measures: cost, features, alignment to existing standards, suitability for the software development lifecycle, and more. With the wide-scale adoption that Selenium has received in organizations of varying sizes and across many industries, the tool has found its way onto most company's radars.

> *I'm not quite sure on the origin of the myth that open source software is not worthy of the enterprise, but the IT community is certainly coming around on this one.*

But from one place to another, this can raise some very interesting questions. Many wonder what it means to them that Selenium is open source, and further, what the community element of that paradigm brings to the table. Here, I address some of the common misconceptions of this situation, as well as talk about some of the real benefits of the community behind a product like Selenium.

The Myths

Open source software is not enterprise-worthy

Enterprise-quality open source software has existed for a very long time. I'm a big fan of the Apache Software Foundation (ASF), and you don't need to look any further to find dozens of examples of high-quality open source software that have become competitive in the industry—and, in a handful of cases, industry standard. ASF was releasing tools like Tomcat and Ant as early as 2000.

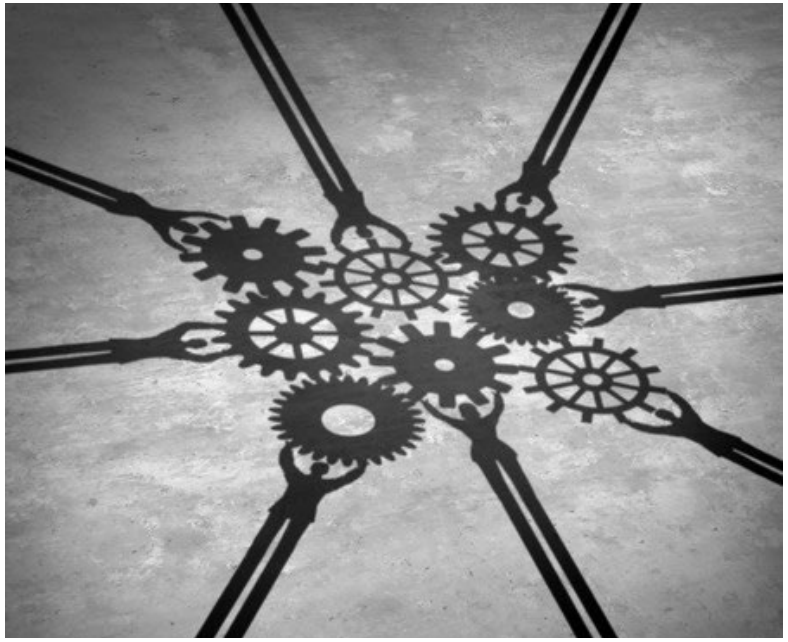
I'm not quite sure on the origin of the myth that open source software is not worthy of the enterprise, but the IT community is certainly coming around on this one.

Open source software is slow to release updates and fixes

While I won't vehemently shout "False!" at anyone who says this to me, I will debate the idea quite heavily. The hard part of talking about this misconception is that it sounds feasible. A community of developers who have other full-time jobs and aren't getting paid to develop the software? Of course people assume that they'll release slower and be less responsive!

Luckily, this is not the reality. While the scenario described here is entirely possible, the same can happen with proprietary software. Also, the developers behind Selenium and similarly successful open source projects have these challenges much higher on their list of priorities than you might guess at first. Couple this with the fact that you can see the source, find the bug, fix it, build it, use it, and share it with the community, and now you're firing on all cylinders. If a bug is problematic enough to frustrate you, it's frustrating plenty of others. Call enough attention to it and the community will respond in force.

> *A large community of driven people will generate good ideas. This isn't a maybe or a likelihood; it happens.*



Open source software compromises security
This one I don't have a problem shouting about. I'll grant that in organizations with lots of proprietary information it makes sense to have blanket policies about protection of source code. That being said, I have never heard an argument approaching legitimate claims that supported this point.

Security is about a lot more than software. It's about arsenals of software, and systems designed to protect that software and your information. I'm not sure what angle people want to take on this, or if this rumor is simply an artifact of a fear of the unknown, but let's be quite clear: An implementation of Selenium, or some other reputable open source software, does not inherently compromise the security of your organization or its information.

The Reality

With these open source myths out of the way, let's turn the conversation toward some of the real benefits of the community behind Selenium. While some can already be inferred from the content above, there are a few notable ones that deserve some special attention.

A strong community drives rapid feedback
The developers of Selenium are users of the product, and vice versa. This is incredibly powerful. When I've been in a position in the past to have sales engineers demo a product to me that seems useful to IT organizations, I've often asked, "Do you use the tool in house?" The question seems pretty innocent, but it has some strong implications. Do you trust the software enough to use it? Do you believe in the power of the tool enough to want to leverage it?

History has shown us that people build better things, whatever they may be, when they are strongly invested in the success of their product and when they believe in it. Open source software does this for us, seemingly for free as a byproduct of the process.

More concretely, this means that the community behind an open source product is strongly incentivized to provide rapid and con-

tinuous feedback in order to drive long-term improvement in the solution that is delivered. It's actually that simple.

A thousand heads are better than one

There are a lot of smart people in this world. Why not leverage that? A large community of driven people will generate good ideas. This isn't a maybe or a likelihood; it happens. Selenium and other open source projects are what they are today because of their communities.

Thanks to the ingenuity, devotion, and real-world experience of thousands of users, Selenium is constantly progressing toward being a better and more comprehensive browser automation solution. And this isn't just a result of the vision of a handful of people, but rather a reactive and evolutionary process grooming Selenium into the solution that real people, doing real work, want it to be.

Many users, many solutions

While a community of active users can be crucial in growing a product into the solution the market needs, this community is also a massive 24/7 support network for IT professionals using the product. I personally think this is the most obvious and direct benefit of the open source community. This is so powerful that it happens across all of human interaction on the web. Just think about the Stack Exchange network, MSDN, or even something like Pinterest. People with similar goals band together on the web every day to collectively solve challenges or share knowledge.

While it is clear that this trend is not unique to open source communities, open source software development more strongly encourages this type of community interaction than other methods. It's a classic example of not having to reinvent the wheel when someone else has already solved your problem and shared the solution, and now there is more content to draw from than ever before.

Open Source: It Takes a Village

I had the pleasure of attending the Selenium Conference, and I would strongly recommend this conference to anyone interested in

Selenium. My experience was extremely positive, the topics of the talks were interesting and ran the full spectrum between business and technology, and the organizers really do a wonderful job of making all of the attendees feel like they're participating in something great. If you want to experience the power of the Selenium community firsthand, there is no better way.

At the end of the day, open source software can experience all the same problems we're used to with proprietary software, and it isn't the solution for everything—but this isn't because community-driven software is bad. Software development is a creative process, and under any paradigm it requires iterative improvement. The nature of software creation necessitates support and feedback and is aided by the agility to implement this feedback in a reasonable amount of time. A strong community of invested users supports these core tenets of software development in compelling ways.

If you're still skeptical, attend a Selenium conference or try your hand at contributing code or bug reports. I'm sure you'll find this group to be very receptive, and with any luck, the community will grow one user stronger.

Automation That Learns: Making Your Computer Work for You

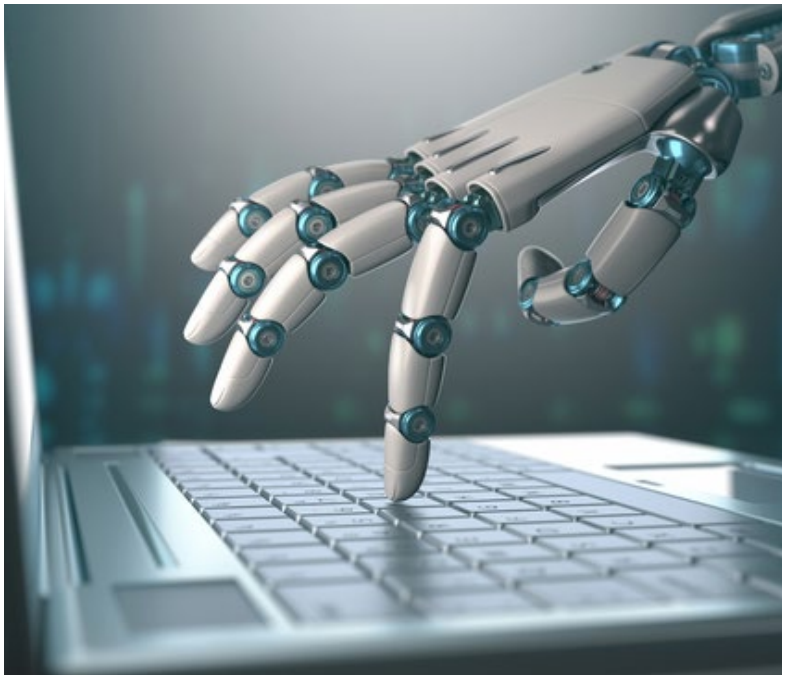
By Jeremy Carey-Dressler

I have heard it suggested that because automation can only do checking, automation cannot learn. By "learning," I will use the dictionary definition, "the acquisition of knowledge or skills through experience, study, or by being taught." If I claim that the act of programming was teaching a machine, and that was how automation learns, it would not be a particularly meaningful claim. Instead, what I am talking about is the acquisition of knowledge through experience and study. I believe automation can, in fact, learn, with a tester adding some additional code around the automation.

Recording Data

One high-level example of a computer learning is simply using a database. A computer can clearly "learn" through the experiences it has, then record that information into a database. For example, a database can log how long a test runs by capturing those run times. Each test execution will be a new row, and over time the computer will "know" about how long a test will take. The automation can then detect if a test is broken by checking a test's execution time compared to the average. For that matter, the test could also notice a trend of a test taking longer and longer and perhaps notify the team that the system is getting slower. Obviously these are not evaluations that replace human intelligence, but they are heuristics a human might care about.

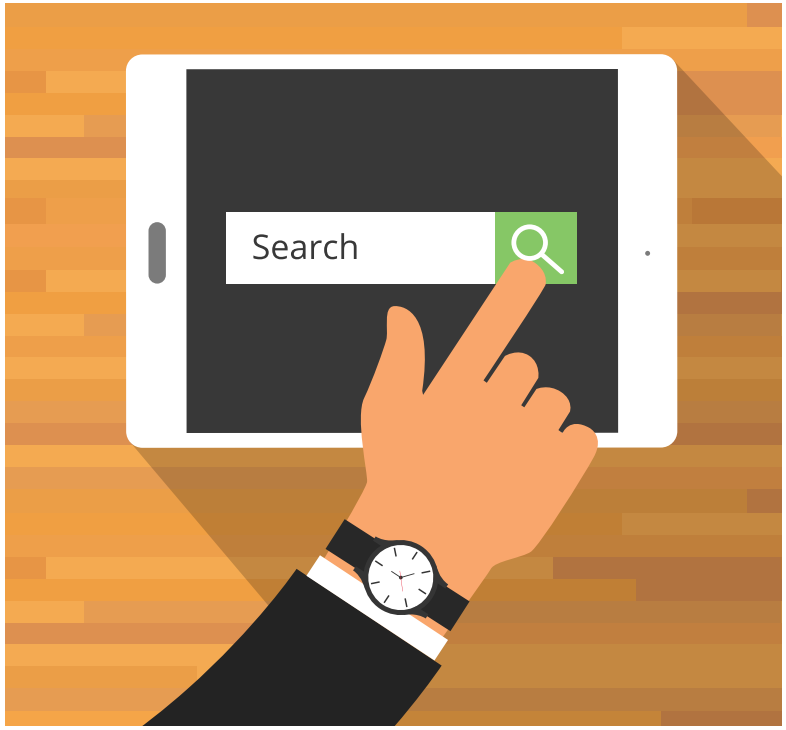
> *I believe automation can, in fact, learn, with a tester adding some additional code around the automation.*



In a real-world example, at my current company we have a load test that runs on a nightly basis. We run light load tests to see if the system has slowed down in any significant ways. Currently a human must review the results, but we plan on adding a formula to decide if the change in performance is too drastic and should be reviewed by a human. The formula would take into account previous runs and decide if the trend is statistically significant or not. Further into the future we want to tie this to builds so that we know which code changes are likely the cause of any slowdowns.

Complex search, such as what Google does, is an area that is difficult to test because of the lack of an oracle. One way to handle this problem is to take search results between runs and see how different the results are—this way you can judge if a new version of search is an improvement or not.

One way to do the validation is by looking at the frequency rate of words you used to search. So, say you search for "Dog" and "Dog" appears twenty-five times in the top ten items that came back from your search. You can store this data, then on the next version of search try "Dog" again. With this, you can create a trend line to demonstrate whether the search is improving. Maybe the next time you search for "Dog" you find the word twenty-four times. While this test might appear like a failure, it doesn't mean the search engine is worse overall. Rather than thinking of each search term as an



> *Given enough searches, you demonstrate a trend showing if the search feature, on average, provided better results to a user.*

individual test that passes or fails, you can consider the entire suite of tests as one large test. That is to say, given enough searches, you demonstrate a trend showing if the search feature, on average, provided better results to a user.

In a previous job, we had a system that would take a screenshot of any failed tests. We would manually look at these for patterns before doing any serious triage because it often was the fastest way. As the number of tests increased, we noticed many fails had similar failing screenshots, but the screenshots often were not exactly the same. There would be five tests that hit HTTP server errors, but there would be nothing the automation could find in common because the tests were executed in different browsers.

One of my brilliant coworkers came up with the idea of using the average luminosity (brightness) of the screenshot and sorting based on that. Luminosity worked because browsers generally render brightness the same way, while font often renders differently. Using the average luminosity, which is just a number, as a sorting method, all similar images would be grouped together and it would become apparent that those tests failed for the same reason, regardless of browser. While we didn't do this, the next step would have been to compare the resulting luminosity value to previous test-run image luminosity. It could, with some additional safety check, then mark the test as "failed like previous run" and thus save having a tester need to reexamine the results.

Learn at Runtime

There are other ways to leverage knowledge around testing. I cowrote a system that automatically did first-level triage on all the automated tests. We noticed there were patterns we found in our triage that occurred over and over again. We had an e-commerce website akin to Amazon that had a variety of interesting code paths. You might use a credit card, or you might use a gift card. You might order in English, or you might order in Portuguese. You might use the search page, the product details page, and cart, but not check out. When we hand-triaged these tests, we would notice a pattern, like "All these tests are failing at search . . . better go test that." It would take thirty minutes to notice the pattern and find the bug. These were things the automation basically knew about each test, and it also knew what the ultimate test result was.

> *We noticed there were patterns we found in our triage that occurred over and over again.*

So we developed a system where the automation would consider all the inputs and all the outputs and determine the most likely cause. Inputs would include things like the name of the test and any parameters in our parameterized tests, like payment method. Outputs would include pages visited, the exception type and message, and page the test failed on. We then would look at all the fails and decide what was common and assign that as the most likely cause. This did not mean a tester didn't have to manually review the results; it just took a lot less time.

Summing It All Up

There is lots of data that automation throws away or ignores, preventing it from learning. If you capture this sort of data when it makes sense, you can analyze it, and perhaps even have the computer check for things that are obviously wrong. Be careful, though—"smart" automation cannot completely replace the need for human

intelligence. There are many other ways you can have your tools "learn" everything from production traffic to what tests are flaky. How much of that you should automate depends on your environment and needs.

Finally, when you find yourself not getting enough value from automation, or you're spending too much time on analysis work a computer could do, it might be best to see if you can get your computer to learn on its own. {end}

The Top 5 Test Automation Mistakes

By Melissa Tondi

I've heard from hundreds of people about the biggest mistakes they've made in their pursuit of test automation success. Here are five of the most popular answers—and recommendations on how we can stop repeating these mistakes.

Automating Everything

Not all tests can or should be automated, so I'm surprised when I hear that teams are still being measured on the number of tests automated versus the overall value the automation is bringing to the team. A better way to approach this is to have a selection process to quickly determine when something should be considered for automation. When you are held to a meaningless metric, your creative license is essentially removed, and you may not be performing the most valuable tests.

Having an Unclear Strategy

If you don't have a clearly defined automation strategy, it will be harder to measure success across teams. Our strategy focuses on automating our build verification tests first, smoke tests second, then a "dealer's choice" of what makes the most sense for the specific solution. The key is that the first two are standard across teams, and more measurable and meaningful.

> *A good tester will utilize all tools and technologies available to ensure their solutions are comprehensive and efficient.*

Not Knowing What to Automate

The goal should be to automate the right tests. In our selection process, we determine the build verification tests and smoke tests first and automatically make them candidates to automate. These are by far the most executed tests throughout a cycle for us, so this aligns well with our strategy. However, we also have clear criteria to ensure there is some thought process around determining whether the ROI makes automation worth it.

Thinking Automation Takes Too Much Time

Don't get caught in the trap of thinking it takes less time to run a test manually than to automate it. Sometimes that's the case, but be realistic with the number of times you run the test. Even though it takes only a few minutes to run it manually once, if you run it many times within a sprint, performing the test manually may exceed the time it takes to automate it. The key is to make sure the tests are meaningful to the goals of the project team and their results are provided quickly.

Treating Automation as a Separate Role

I see a lot of organizations use their automation engineers in a matrixed role where they only automate the tests a manual tester instructs them to. On the other hand, I've also seen automation engineers look down on traditional testing activities and refuse to do anything other than automate tests. Somehow, we decided that automation is so specialized that it's more valuable than other kinds of testing. These engineers should still be intimately involved in the product they are testing.

A good tester will utilize all tools and technologies available to ensure their solutions are comprehensive and efficient. Test automation is a valuable process, and if you avoid these common mistakes, you'll be able to take full advantage of its effectiveness.

The Buzz on Test Automation

INSIGHT FROM AROUND THE INDUSTRY

On Automation ROI:

"What we found from customers who have gone the test automation route is, while they have been able to speed up the amount of testing they're doing, decreasing the amount of time it takes to do that testing, they've actually been taking that cost savings and re-investing it into the application."

» Josh Galde

On Agile Test Automation:

"It's very difficult to imagine a project where there is no test automation being done, especially if they're practicing agile because agile embraces test automation right from the get go."

» Kalyan Konda

On Domain Expertise:

"I want people to have thoughts and ideas on how to test. I want my automation people to be able to write their own test cases. I need them to understand the domain to make sure what they're automating is the right thing."

» Mary Thorn

It's very difficult to imagine a project where there is no test automation being done.

On Automating the Right Tests:

"A lot of time is wasted on overly simplistic tests automated just for the sake of automating a test. This is especially true of automated tests that attempt to emulate customer interactions, or GUI automation. Well-designed automated tests with specific purpose can be very helpful in a project, and can also free up a tester's time to test and find ways to improve their product."

» Bj Rollison

On the Human Side of Automation:

"If we divorce the notion of 'automation' from the notions of 'test cases' and 'test scripts,' we can begin to think of automation as a judicious use of technology to help humans do their jobs; in our case, those humans are testers. This broadens our idea of automation to include non-test-case-based tools that will assist humans in performing their testing."

» Paul Grizzaffi

On the Testers' Role in Automation:

"By insisting that real testing activities cannot be automated at all, testers are left out of the continuous testing conversation. That introduces the risk that important improvements in test automation will be shaped more by software engineers than by the testing community."

» Stefan Friese

By insisting that real testing activities cannot be automated at all, testers are left out of the continuous testing conversation.

On Automation and Enterprise IT:

The design of test automation tools may already have reached a boundary where it can no longer cost-efficiently keep pace with change in enterprise IT systems, and where alternative solutions can no longer be ignored.

» Martin Ivison

On Failing the Right Way:

Using automation as an example, we may be able to execute quicker once the initial tests have been written, but that means we fail faster, not that we find the right failures in the code quicker.

» Leanne Howard

Additional Resources

MORE INFORMATION FOR SOFTWARE PROFESSIONALS

STICKYMINDS™
A TECHWELL COMMUNITY

StickyMinds is home to thousands of software testing resources, including articles, *Better Software* magazine articles, conference presentations, and interviews with industry notables.

CLICK HERE

NARROW YOUR SEARCH TO A SPECIFIC TYPE OF CONTENT:

StickyMinds Articles

StickyMinds articles cover a wide range of software testing topics including test automation, test management, test design techniques, agile testing, test process improvement, test tools, and much more. **Click here** to read test automation articles on StickyMinds.

Better Software Magazine Articles

Better Software magazine is a digital quarterly filled with expert analysis, how-to articles, and real-world case studies covering all aspects of software development. **Click here** to join StickyMinds and access test automation articles from *Better Software* magazine.

TechWell Conference Presentations

Couldn't make it to a STAR software testing conference? TechWell conference presentations are available to StickyMinds members soon after a conference ends. **Click here** to join StickyMinds and access test automation conference presentations.

Interviews

Each year, TechWell interviews dozens of software professionals including well-known thought leaders, seasoned practitioners, and respected conference speakers. **Click here** to read, listen to, and watch interviews with test automation experts.