Test Design Automation:

Driving Testing at the Speed of Agile

WRITTEN BY GEDEON HOMBREBUENO, PRODUCT MARKETING MANAGER, CA TECHNOLOGIES

CONTENTS

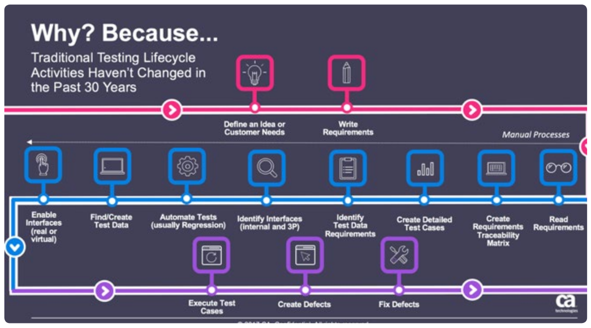
The days of yearly product, feature, or code releases are long gone. And while monthly releases were common in previous years, today, bi-weekly releases are gaining traction — though there are a few over-achievers. By moving to AWS, Amazon bursts out releasing code every 11.7 seconds. Etsy steadily follows a 50+ deployments per day cadence. And Netflix releases thousands of times daily. Needless to say, quality at speed at these companies is producing real results in terms of stellar revenue growth, dramatic cost savings, and enviable customer loyalty. This is only possible because teams have fundamentally changed the way they think about software quality to enable them to test at the speed of agile.

Testing at the speed of agile drives quality at speed. But getting there can be a challenge for many organizations. These challenges start all the way to the left of the Software Development Lifecycle (SDLC) with 64% of total defect cost traced to poorly defined or incomplete requirements. We've seen this happen time and again when there's a disconnect between requirements and the final output.

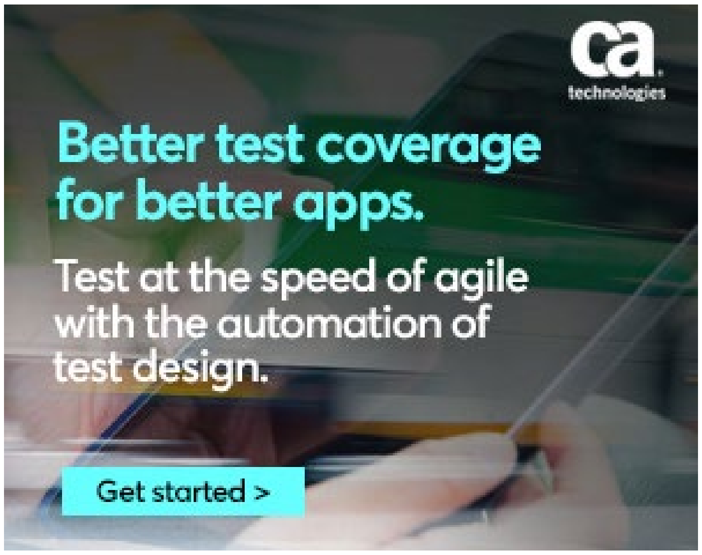
This is particularly important because of the trickling effect of this issue. If you don't get your requirements right, costly rework and defects are sure to happen, affecting the bottom-line and possibly the brand. On the flip side, get your requirements right, and you'll see that 64% defect rate drop significantly, resulting in better quality outcomes — at speed.

THE CURRENT STATE OF TESTING: WHY LEGACY TESTING HASN'T CHANGED IN 30 YEARS

It's hard to believe that after 30 years, 70% of testing is still performed manually. A major bottleneck in the SDLC, legacy testing remains a barrier to speed and quality — unable to keep up with today's agile, continuous testing model. Here's what I mean.


Why? Because...
Traditional Testing Lifecycle Activities Haven't Changed in the Past 30 Years
Define an Idea or Customer Needs
Write Requirements
Manual Processes
Read Requirements
Create Requirements Traceability Matrix
Create Detailed Test Cases
Identify Test Data Requirements
Identify Interfaces (internal and 3P)
Automate Tests (usually Regression)
Find/Create Test Data
Enable Interfaces (real or virtual)
Execute Test Cases
Create Defects
Fix Defects
ca technologies

Ideas and customer needs are discussed with visual aids such as diagrams on a whiteboard or piece of paper. These ideas are converted into written, text-based requirements. What follows is a series of manual steps — arguably, the most tedious, labor intensive part of the testing lifecycle. This includes:


ca technologies
Better test coverage for better apps.
Test at the speed of agile with the automation of test design.
Get started >

Test at the speed of Agile.

Automatically create tests with maximum coverage right from your requirements.

Only CA offers a Test Design Automation strategy that lets you do shift left testing into the design phase. Drive quality at speed today.

Explore >

ca technologies

- Reading requirements; understanding user stories and acceptance criteria.
- Creating a requirements traceability matrix, providing visibility on the desired coverage against those requirements.
- Designing detailed test cases; outlining expected results.
- Identifying test data requirements satisfying the test cases you just created.
- Identifying internal and third-party interfaces needed to run those tests against the test data.
- Automating tests. This is challenging because test automation technology hasn't evolved at the pace of application development technology. We still rely on test automation approaches like OCR, object-based recognition, or image-based recognition, and they tend to be brittle, with those scripts breaking easily when application changes or new builds occur — leaving you to manually resynch objects. And depending on how you've implemented your framework, you have to rework the actual scripts and so on. The answer to that? Automate only the tests that are not expected to break: the regression testing suite.
- Provisioning test data and enabling real or virtual interfaces.

Only after all these laborious (and typically manual) steps are performed can you finally start test case execution.

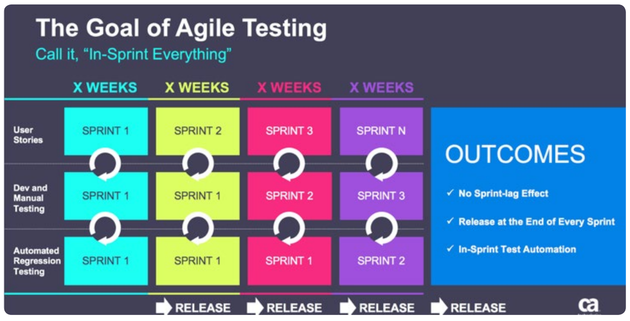
Testing practitioners know that these mundane tasks take a lot of time, effort, and grief before getting to the tail end of the process, which includes executing those test cases and creating and fixing defects. And that's where the real value is for the agile team: that quick, immediate feedback on the code that was just checked in or deployed to the environment, and not just feedback on application regression.

And so, many organizations today that want to go agile and speed up their application delivery and release processes find that traditional testing no longer fits their need for speed and quality.

WHAT IS TEST DESIGN AUTOMATION?

As more and more organizations adopt test-driven, agile development methods, they gravitate towards test design automation — practices and technology that help test teams automatically generate reusable test assets like test cases, test data, and test automation scripts right from clearly defined and complete requirements. QA rises above the challenge and goes from being a bottleneck to becoming a driver for better and faster application development and delivery. In the end, test automation and development within the same sprint becomes a reality.

You'll know that you're on the right track toward agile testing when you're able to do full-on "in-sprint everything." Take the current sprint you're in right now with your development team. A good litmus test is that you're not only doing the automated test execution of your regression testing scripts for that sprint but you're also automatically generating your test automation scripts for the new features being coded in that same sprint. This means within that sprint, you're doing the following:



1. Automatically generating your Selenium code, VB script code, or any language your test automation tool works with depending on the application you're testing. This includes your Gherkin feature files if you're into behavior-driven development (BDD) or acceptance test-driven development (ATDD), as you can automatically generate those from your models.
2. Hooking up those scripts with a continuous integration tool like Jenkins or what have you to kick off those scripts in your favorite test automation engine

Why does this matter? Imagine the remarkable cutback in test cycles by reducing — if not essentially eliminating — script maintenance because now everything is derived from a model. In effect, you can tie everything that's related to your acceptance criteria, including your user stories and requirements, and get them tested automatically right within the sprint. Further, in true "in-sprint everything" fashion, if a change occurs during the sprint (like when new code is introduced days before the end of the sprint to fix a defect or better code gets written as a result of refactoring), it won't derail you in any way because all you have to do is make those changes on the models. This allows you to easily regenerate your testing artifacts including all your tests, your data, and your automated scripts, and still get those going within the sprint hooked up within your CI pipeline.

In the end, there's no sprint-lag effect. You're in a position to have a shippable product at the culmination of every sprint, thus achieving in-sprint everything. This model-based testing approach allows you to have your test cases and test automation scripts ready to go on day one of the sprint, just waiting for the application code to be available for testing.

What's clear is without applying test design automation practices and technology, none of this would be possible. By enabling you to automate the above-mentioned testing activities, test design automation shortens testing time significantly and helps you avoid costly defects that stem from poor requirements. You're not only building better quality apps; you're also delivering them to market faster at lower costs. The question then becomes: In what ways does test design automation drive agile testing?

KEY USE CASES FOR TEST DESIGN AUTOMATION

Increasingly, organizations apply test design automation practices around the following four key use cases.

1. **Requirements Engineering**

Today, They Model Their Visions and Requirements…
Using Diagrams, Defining Expected Behavior, Rules, & So On
DEVELOPERS
EXECUTIVES
PROJECT MANAGERS
TESTERS
PRODUCT OWNERS

Business analysts, developers, and testers usually get together to flesh out the details around a requirement. They tend to start with a whiteboard where they draw their initial vision for the new feature or change to be implemented. But the next step is usually where it all breaks down. Time and again, we see amazing information gets captured clearly and completely through diagrams on a whiteboard only to get translated to an inherently ambiguous medium called "the written language." Defining requirements in this text-based manner is inefficient. They tend to be error-prone, and the lack of intuitive support for traceability makes this approach even more risky. Not to mention, it is just plain hard to organize, communicate, and collaborate on a text-based document or tool. Grooming and optimizing those requirements can be a challenge, as it is hard to find the information across paragraphs, bullet lists, tables, etc. And when changes happen, dealing with the impact of those changes across all that text, whether manually or with some tooling, can be even more daunting.

At the heart of requirements engineering is this concept of modeling, where that original requirement conveyed on a whiteboard does not get converted into traditional text-based requirements. These diagrams represent requirements as mathematically precise visual flows, adding accuracy to requirements engineering and reducing requirements ambiguities while supporting better collaboration and communication across key stakeholders.

2. **Automatic Test Generation**

One way to boost testing speed and velocity is having the ability to automatically create test cases linked to the right data and expected results right from your requirements models. Test design automation technology can get you there. By automating the automation, it allows you to generate every automated test needed to exhaustively test an application. This active automation approach enables the automated generation of the test cases and automated test scripts themselves. In other words, it automates all phases and elements of the testing process. Further, when used in conjunction with test data management technology, the data required for relevant test cases is automatically generated and used when running those tests.

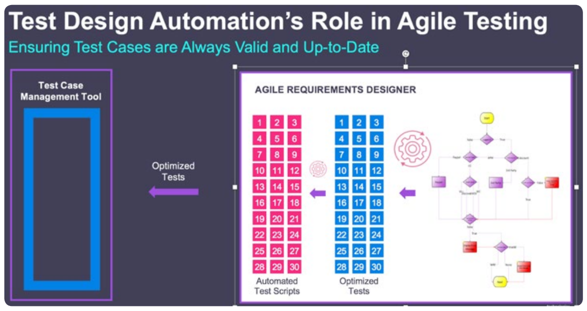
3. **Test Case Migration and Optimization**

Test case migration and optimization allow you to create the smallest number of test cases for maximum coverage. So, you end up with just the right tests needed to deliver better apps, faster. And since most organizations use various test case management or lifecycle management tools for their test case maintenance and management, integration with these various management systems and tools such as CA Agile Central, Micro Focus ALM, Atlassian JIRA, and more is imperative. This integration allows test teams to easily import test cases, remove duplicates, and export optimized test cases back to their existing management system.

This means that existing test cases can be imported from these lifecycle management tools and converted into an unambiguous, active flowchart. Everyone has those old regression testing suites with hundreds or thousands of manual test cases and, for the most part, they don't know whether those tests are still valid. Sound familiar? Imagine if you could optimize that old regression testing suite and ensure that you have only valid tests moving forward and that there is zero maintenance on them as the application changes. You can do that by maintaining the model, which will regenerate all those optimized test cases whenever there are application changes and store them in your current test case management tool.

4. **Achieving Agile Testing**
Following in the footsteps of agile development principles, agile testing involves baking in quality to the product as they are being defined, designed, and delivered. In this use case, testing occurs within the sprint, allowing the various stakeholders to collaborate actively with the development team. So, ongoing feedback happens, turning into executable specifications that guide coding. Both coding and testing are performed incrementally and iteratively within the sprint, building each feature in accordance with quality standards.

This agile testing use case rounds up the previous three use cases leading up to this ultimate goal of testing at the speed of agile. This means employing test design automation practices, providing testers with everything they need to test the most functionality possible at speed, within the sprint.


Test Design Automation's Role in Agile Testing
Ensuring Test Cases are Always Valid and Up-to-Date
Test Case Management Tool
Optimized Tests
AGILE REQUIREMENTS DESIGNER
Automated Test Scripts
Optimized Tests

CRITICAL CAPABILITIES OF TEST DESIGN AUTOMATION TECHNOLOGY

REQUIREMENTS GROOMING

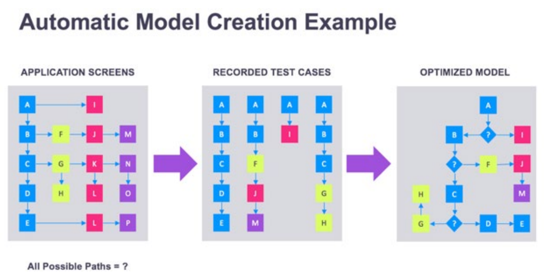
Many organizations today use requirements management or test case management tools to capture and store their requirements, including user stories and associated test cases. But these tools usually lack the capability to model requirements in visual and dynamic flowcharts and they don't have the capability to optimize tests for maximum coverage.

The key idea behind requirements grooming is having the capability to create requirements directly and organically or model the environment in a supported lifecycle management tool and import it. Support for common tools like Microsoft Visio and Business Process Models via BPMN and XML Process Definition Language (XPDL)-compliant formats becomes a must, as well as the ability to import test assets such as user stories and existing test cases from a range of tools, including JIRA, VersionOne, MicroFocus ALM, and CA Agile Central. That is important, as those testing assets can be used as the starting point for grooming a requirement when there is nothing currently documented. While there are a variety of options for requirements grooming, the key point here is that once you have captured your requirements in a flowchart, all the remaining artifacts (specifically, test cases and test scripts) are automatically generated from the flowchart.

AUTOMATIC REQUIREMENTS MODELING

Once you start defining your requirements as models, you are effectively going into the realm of model-based testing. In this case, your requirements and your testing model become one and the same.

Here's an example to show why this approach has advantages. In the traditional testing process, you usually define your requirements by hand in some text-based format. Your testing team then creates test cases based on them. When these requirements change, your testers will need to scour through the change request, analyze the impact of those changes on your existing test cases, and painstakingly carry out those changes across potentially hundreds or thousands of test cases by hand. This slow and error-prone approach simply goes away when you choose to take the model-based testing route.



A model-based testing approach delivers real ROI, but building your initial model can take time and effort. One way to break this barrier to model-based testing adoption is using technology to automatically create models for you. Truly game-changing automatic requirements modeling accelerates model creation, allowing you to quickly and easily generate your initial flow. That is accomplished by capturing the user actions (steps, assertions, data used, etc.) during your regular regression testing cycle. The data captured from those actions is then parsed into a flowchart. With the flowchart, this automatic requirements modeling capability allows you to deduplicate the steps, turning them into one concise model. As you can imagine, this allows you to radically simplify and accelerate the creation of your initial models, opposed to manually creating them by hand. Then, you just inspect the model for accuracy and add to it what you might have missed.

TEST CASE DESIGN AND OPTIMIZATION

The right test case design and optimization approach greatly reduces the number of tests by only generating the minimum number of tests needed to achieve the desired coverage — driving testing efficiencies in the process.

In practice, there are different variations of test coverage. These include (in order of most coverage to least): all possible paths; all pairs (that is, every pair of edges); all in/out edges (every pair of edges where one can follow directly from the other); all edges; all nodes; and probabilistic. But oftentimes, the balance between your testing needs and how much you're willing to compromise for cost determine your desired coverage level. In any case, test case design and optimization can provide metrics that show the amount of coverage you currently have. You can also see if you're over-testing by showing how much each part of your model is being tested more than once. On the flip side, you can see if you're under-testing by showing how much exposure you're potentially dealing when your test coverage is lacking.

TEST AUTOMATION SCRIPT GENERATION

Once you have determined your desired coverage and generated your manual test cases from the model, test automation script generation allows you to generate a suite of automated test scripts. For this, you must define a configuration file for your application, primarily consisting of standard and custom functions for the language you intend to use in your test automation execution engine (Selenium, Appium, VBScript, C#, Java, Ruby, etc.). Each of these functions are available to be referenced from blocks in your model. With each block referencing the functions in your configuration file, as each path through the model is defined based on the coverage technique you selected, the series of function calls concatenated from each block will form a fully executable test automation script that your test automation execution engine will pick up and execute.

You can run these with in test automation frameworks such as Selenium, CA Application Test, Ranorex, etc. Automation makes running and, more importantly, rerunning tests much faster and more efficient.

CHANGE AUTOMATION

Change automation can single-handedly make testing much more cost-efficient by simply having the ability for test cases to be updated automatically when requirements change. If you're still stuck doing the more traditional testing approaches, how do you make changes to the actual test cases when changes in requirements occur? The conventional way is to ignore existing tests in your suite and recreate them all by hand. Some say it's easier that way. Yes, you can do this, of course. But doing so is highly inefficient and potentially costly. Just imagine if a given change only affects 8% of test cases. You're effectively wasting the other 92% of the test cases, which are now being discarded and recreated just as they were. This is utterly unnecessary and on an enterprise-class project involving a large number of tests, the cost of recreating these tests can be huge.

Change automation becomes even more valuable to the business, as it provides the following capabilities in an automated manner:

- Analyze the impact of a change. This means that test cases that are affected by the changes in requirements are automatically detected.
- Heal your test suite. This means that test cases that were detected to have been impacted by a change can be fixed automatically, so they adapt to the new requirements. Or even better, when new tests are generated, they will be confined to paths that haven't already been covered by existing tests.

What if those test cases haven't been automated? If you're part of an agile team, analyzing the impact of that change to the automated scripts — in addition to the manual test cases — would represent a considerable additional effort that won't fit in your current sprint.

The power of change automation is magnified when you're working in an agile environment. Those impacted automated scripts, in this example, would've been automatically healed or regenerated without the need to manually inspect and update scripts. How's that for ROI?

INTEGRATING WITH TEST DATA AND VIRTUAL SERVICES

Going beyond requirements and test case design, test design automation delivers greater business value when it is further associated with test data and virtual services for testing.

Integration with test data management allows you to create and link the test data needed for testing, so when you run a test, relevant test data is generated automatically based on the appropriate parameters.

Integration with virtualized services allows you to access test data beyond the control of the development team. For example, if you are working in a B2B environment, you will need to test against data that is provided by your business partners, simulating this and other sources of data for testing purposes. You're not necessarily testing your business partners' systems; you're testing your own application, which needs data from those partners. That way, you're breaking the dependency on those external teams and not slowing down your development and testing activities.

INTEGRATING WITH BROADER DEVOPS ENVIRONMENT

The right test design automation technology integrates with the majority of testing frameworks. This is particularly important since organizations involved in software development looks for technology that enhances their existing testing practices and tools. This means tests (either manual tests or test scripts) can be exported into CA Agile Central, Micro Focus ALM, and Atlassian JIRA, among others. These integrations go beyond just passing information from one to the other. For example, synchronization with JIRA supports both sync up and sync down — so, once synced, any change in either tools will update the other automatically.

Going beyond testing framework integrations, your test design automation technology must have the capacity to import your existing test cases in a variety of formats — from basic spreadsheets to test cases written in XPDL, as well as tests output by Bender and SoapUI.

The experience must be smooth and simple so you can easily import your existing tests, reformat the resulting requirements model, and produce a suite of optimized test cases before exporting them back into your test case management tool of choice. Moreover, once this has been done once, there's no need to import a second time. You can simply keep refining your requirements and generating test cases.

TEST DESIGN AUTOMATION: REAL RESULTS TODAY AND BEYOND

Testing at the speed of agile entails reducing manual testing effort while delivering higher-quality applications to market faster. Helping you drive agile testing approaches within every sprint, test design automation technology enables you to automatically generate the optimal set of tests right from your requirements modeled as complete and unambiguous flowcharts. Automatically updated when requirements change, these tests allow you to deliver software that reflects changing user needs.

Is it possible to reduce time- and labor-intensive testing tasks to mere slivers of their former selves and still produce a correctly working product? Can test scripts take less time per sprint and less time overall?

For organizations that leverage test design automation practices and technology, the answer is yes. Here are two examples of organizations that have realized real results by taking this approach.

1. A Top 20 global bank started converting their manual script test process to be model-based, allowing them to achieve a 70% reduction in script creation time, going from five days per sprint to one day. But there is more here than just a time advantage. Of excellent value to users is co-monitorial modeling, which reduces the number of scripts needed to execute to get complete coverage. Traceability, from requirements to test cases, is also of immense value since legacy systems often leave no straightforward way to find out if a change has been introduced during a sprint cycle and then fix it in a fast and efficient manner.

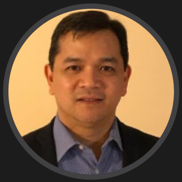
2. A global financial services leader went from doing manual requirements definition and test case design to a model-based testing and test design automation approach. They started building requirements models and creating business process flows. This was followed by automatic test case generation and script creation. The result? The firm dramatically improved their testing efficiency, delivering test cases 10x faster by automating the process in hours as opposed to days or weeks prior to test design automation adoption. Further, they went from 90+ days behind in new automation to full, in-sprint automation with little to no technical debt.

Test design automation can help you break the barriers to testing at the speed of agile, while helping you focus on improving quality, collaboration, speed, and efficiency. More and more organizations that have adopted these practices and technology were able to:

- Improve quality, optimizing testing by getting maximum coverage with the smallest number of tests.
- Boost application delivery speed, reducing manual effort by automating test design and execution processes, data allocation, and change implementation.
- Reduce testing cost, reducing costly rework and detecting defects earlier (where they require less time, resources, and cost to fix).

WHAT'S NEXT FOR TEST DESIGN AUTOMATION?

Keeping up with customer demand and getting ahead of the competition means that you have to continuously deliver high-quality apps and experiences. Organizations are therefore embracing agile testing practices to deliver quality at speed more than ever before. The impact of test design automation in driving testing at the speed of agile is increasingly evolving. We see this advancing even more as organizations take requirements and test case design more seriously as an imperative for a broader set of applications and environments from legacy systems to web-based or mobile applications. In particular, with cloud adoption on the rise, the need for test design automation delivered as a cloud service emerges, as well. Test design automation-as-a-service will allow organizations to get all the benefits of test design automation as described above, amplified by myriad advantages of the SaaS model, particularly in terms of speed, scale, and simplicity.



Written by Gedeon Hombrebueno, Product Marketing Manager, CA Technologies

Gedeon has extensive product marketing and product management experience with enterprise IT solutions in various domains including: DevOps, Cybersecurity, and Systems Management across physical, virtual, and cloud environments. Currently, Gedeon is a Principal Product Marketing Manager at CA Technologies, focusing on bringing CA's Continuous Testing and Delivery solutions to market.

DZone, Inc.
150 Preston Executive Dr. Cary, NC 27513
888.678.0399 919.678.0300