Test Scenario Design Models: What are they and why are they your key to Agile Quality success?

Andrea Gormley, Robert Gormley
anniegormley99@gmail.com, hotspur99@hotmail.com

Abstract

Customer satisfaction is considered the most important criterion for successful product delivery by a majority of Agile teams (52%). Yet as Digital Disruption continues to fuel the need for hyper speed IT delivery, many Quality teams are being left behind as they continue to leverage an old-world testing paradigm focused on Functional & Regression Testing built on traditional, linear test case design that is not customer centric. What's more, traditional test case design has become an antipattern for most product teams as it neither serves to validate the many different emerging layers of software architecture nor does it establish a core understanding of what should be built into automated tests.

What if, instead of traditional test case design, Quality teams could focus on building scenario-based outlines of user behaviors in a consistent, ubiquitous narrative to not only connect product functionality to users but also to establish a tight boundary for automated testing of acceptance criteria? Welcome to the world of the Test Scenario Design Model, where user behavior is king and test cases are short, concise, and business language-driven. In this paper, we explore what the Test Scenario Design Model is, how it can help Agile Quality teams accelerate their testing and show how Test Scenario Design Models tie to traceability and Quality metrics to truly measure your product team's Agile health.

Biography

In her role as a Principal Quality Consultant, Andrea works with organizations to master the art of Agile Quality. Her strengths in Lean Agile processes and business transformation allow her to develop practical and pragmatic Quality solutions for product teams to implement and refine. Andrea excels at helping organizations align on Quality goals, implement meaningful behavior-driven Quality processes, and mentoring less experienced resources on Agile Quality.

In his role as Chief Essentialist of Continuous Quality, Robert provides executive level total quality management strategy through efficient and effective testing, leveraging test automation and global service capabilities. Industry experiences include finance, healthcare, retail, security, wearables, IoT, Big Data, AI, and MBL. Robert helps organizations create lean agile processes that allow product teams to deliver business value with built-in quality. Robert uses his Master's in Education to help train and mentor all levels and types of resources in the ways of Total Quality.

1. Introduction

In its most recent survey on the State of Agile, VersionOne states that 52% of their respondents believe that success within an Agile initiative is measured by customer/user satisfaction and that success for an Agile project is measured equally by customer/user satisfaction at 46% (VersionOne 2019, 11). These percentages represent a jump for the customer/user satisfaction measurement as previously only 44% and 28% (respectively) of respondents believed this in 2016. Given the raise in the importance of customer/user satisfaction, most savvy IT folks would expect that Quality strategies within Agile organizations would naturally gravitate toward validating customer/user experience to ensure their satisfaction. Surprisingly, in multiple Quality reports at the end of 2018, 75% of respondents suggested their Quality strategies were focused on Functional and Regression Testing. These Quality strategies typically focus almost entirely on testing at the UI level to ensure proper navigation, functionality, and completion of simple rote tasks but do little to address customer/user satisfaction concerns. What's more, in many of these same organizations focused on Functional and Regression Testing, the key driver for testing is the over reliance on linear, repetitive test cases that don't place the customer/user at the center of the testing effort.

How then does an Agile Quality team ensure both built in quality and meet customer needs and satisfaction? What if, instead of traditional test case design, Quality teams could focus on building scenario-based outlines of user behaviors in a narrative that not only connects product functionality to users but also establishes a tight boundary for automated testing of acceptance criteria? We have been employing the use of Test Scenario Design Models, an approach to test case design based on the work of Cem Kaner's Scenario-based Exploratory Testing, Hans Buwalda's Soap Opera Testing, and ISO 25010's Product Quality and Quality in Use Models, to solve the challenge of placing the customer/user at the center of the Agile testing paradigm. In doing so, we have been able to help Quality teams be more successful in achieving both efficiency and effectiveness in Agile testing while ensuring that products being built deliver on the promise of customer/user satisfaction. The use of Test Scenario Design Models has allowed our teams to create clear traceability to user stories and acceptance criteria, the cornerstones of business value delivery for Agile teams.

2. What are Test Scenario Design Models?

How do you define Test Scenario Design Models? They are scenario-based outlines of user behaviors in a consistent, ubiquitous narrative that allow you to focus on testing at every layer, not just the UI. In this section, we'll go through the details of what Test Scenario Design Models are and the history behind them.

2.1 Scenario-based Testing

Cem Kaner once defined a scenario as "a hypothetical story used to help a person think through a complex problem or system" (Kaner 2003, 1). Hans Buwalda expanded the scenario definition by saying that when used in the context of Soap Opera testing, the scenario is "based on real life, exaggerated, and condensed" (Buwalda 2004, 31). When we put the two definitions together, we have a new definition (a workflow) that places the user at the center of the testing effort while emphasizing the end-to-end nature of the testing approach. However, if you read carefully into both definitions, the biggest short-coming in both approaches is highlighted: the system under test must be mostly complete, a criterion that most Agile product teams could not meet until the end of a project. Additionally, the problem with most in-flight Agile Quality efforts is that the original testing strategy typically focuses entirely on

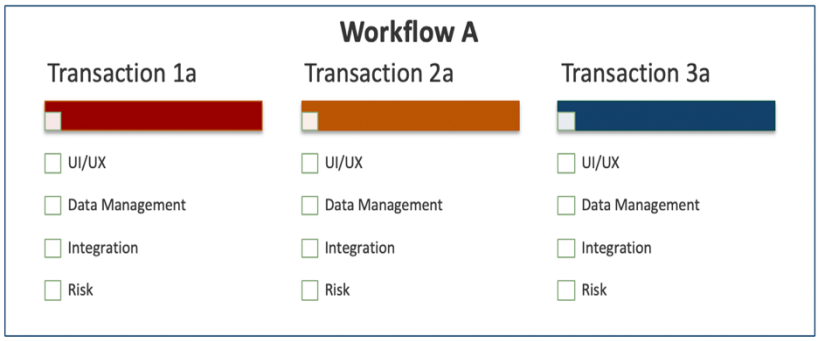

Figure 1: End-to-end nature of scenario-based test design

Functional and Regression testing, an approach that does not look at scenario-based test design and execution.

2.2 ISO 25010's Product and Quality in Use Models

In order to circumvent the shortcomings of the end-to-end testing needing a nearly complete system in order to test, we've brought in ISO 25010's Product Quality and Quality in Use models. The Product Quality model of ISO 25010 emphasizes testing at a feature and functional level to assure the static properties of software and dynamic properties of the computer system (ISO 25010, 2011). By focusing on features and functions, the Product Quality model allows testers to focus on continuously developed increments of software during in-flight Sprints. The Quality in Use model of ISO 25010 emphasizes both the context of software use and satisfaction of users to measure the level of quality maturity in the system under test. In Figure 1, you can see how scenario-based tests run horizontally from one transaction to the next to test a complete workflow. Whereas in Figure 2, you can see that scenario-based tests tied into ISO 25010 run both horizontally and vertically to carve out transactions as discreet modules because you are testing at both feature and functional levels while considering how the modules come together as a whole workflow. Note how, in most cases, user stories and their accompanying acceptance criteria are typically written in stand-alone fashion like the transactions in Figures 1 and 2. The main point of combining scenario-based test design with ISO 25010 is to enable a product team to always be working toward a releasable product that delivers end-to-end functionality by always testing for it. A product team that delivers code in isolation will always deliver poor quality (lack of integration) and a product team that delivers code in end-to-end workflows will likely never deliver code with any sort of speed.

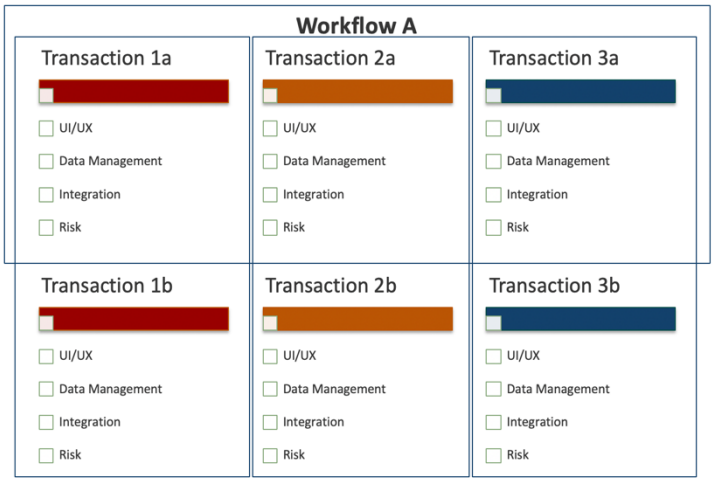

Figure 2: End-to-end scenario-based testing with ISO 25010

2.3 Test Scenario Design Model Details

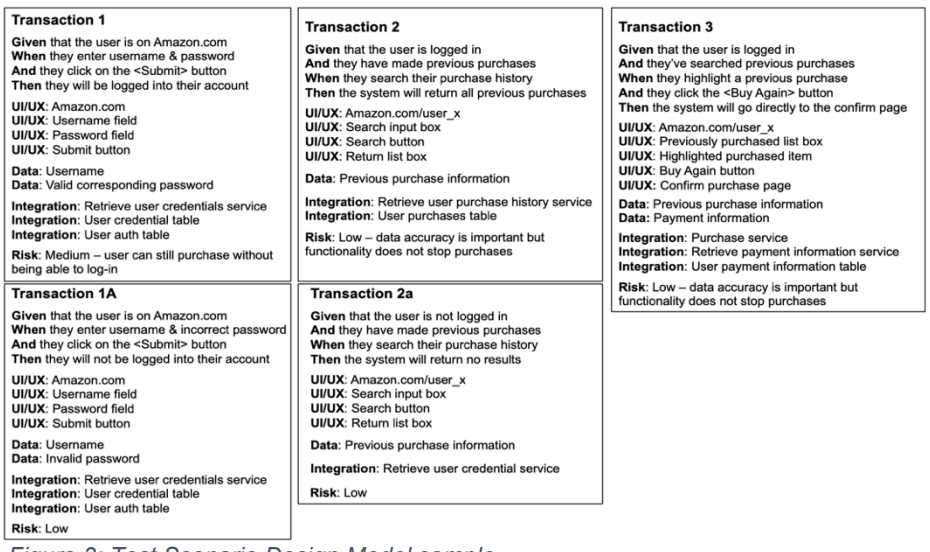

Figure 3: Test Scenario Design Model sample

As you can see in Figures 1 and 2, a Test Scenario Design Model focuses on creating discreet transactions that can stand-alone but are part of a larger workflow. Note that each transaction is broken down into 4 specific parts: the UI/UX, Data Management, Integration, and Risk. The UI/UX component of a transaction should reflect the behavior of the user against the system and the response of the system to that behavior. Note, we aren't talking about the UI exclusively as focusing only on the UI simply validates that the system meets requirements but ignores fit for use.

Data Management is a key component of a Test Scenario Design Model as most modern systems simply store, manager, or distribute data. As such, making sure that you are validating both system and inherent data characteristics as well as semantic and syntactical attributes of data is becoming increasingly important. Properly analyzing data requirements allows you to develop the best test data management strategy while also identifying the complete data set required to test both your transaction and workflow. Integration points are important to understand as the basis of a Test Scenario Design Model is an end-to-end workflow. Ensuring that integrations/hand-offs from one transaction (and in some cases one system to another) to the next are tested successfully is vital to testing workflows. The areas where defects tend to cluster most in modern systems are at points of integration. Emphasizing this testing is important for improving your teams defect detection efficiency. The final component of a Test Scenario Design Model is Risk. Because testing exhaustively is impossible (ISQTB, 2011), understanding where risk lies at a transactional level is important for not only understanding where to test but also how much to test. Working in Sprints predicates that QA teams have to be able to follow a more risk-based testing approach in order to meet testing goals but also to ensure team velocity.

3. Why use Test Scenario Design Models?

Back in the day (you know you're old when you say that!), we used to proudly talk about the number of test cases we had written; the larger the number, the better we felt we were doing our jobs. That all changed when we became part of an Agile product team because Agile favors "working software over comprehensive documentation (Agile Manifesto, 2001)" and everyone knows having thousands of manually written test cases has little value for an Agile product team. There are those who would argue (Robert used to be one of them) that writing detailed test cases helps in automating the test cases. However, we now argue that the increased technical skillset of Quality Engineers means they require less mundane details (read UI); rather they value a complete story that helps them test the different layers of software systems. In addition to testing the different layers, automation that leverages Test Scenario Design Models focuses on creating modular, stand-alone tests at a transaction level, thus making it easier to script and much more reusable. The stand-alone, transactional nature of the automated tests allow you to exponentially expand test coverage by simply juxaposing variations of transactions one after the other in order to create better coverage of all scenarios. Traditional, linear tests have a limitation in that they have a singular intent (not parameterized, modular, or mutable) which limits reusability.

Test Scenario Design Models are built on fundamental testing principles. We've already noted that exhaustive testing is impossible (Testing Principle 2 according to ISQTB) so explicitly writing out every permutation of test case is a wasted effort. We also note that testing early (Testing Principle 3 according to ISQTB) or shifting left improves testing efficiency and effectiveness while reducing overall cost. Analyzing and planning test strategies for individual user stories while maintaining the focus on the whole picture allows you to build more comprehensive tests during pre-testing. Capers Jones, in his book *The Economics of Software Quality*, notes that pre-test activities are 25% more likely to find defects compared to the act of testing itself (Jones 2012, 5). What's more, we know that testing is context sensitive (Testing Principle 6 according to ISQTB), that is to say, the more we know about what we are testing, the better we are at finding flaws in the system. Test Scenario Design Models allow you to focus testing at the Product Quality level but also emphasizes that end-to-end testing should always be the goal to ensure customer satisfaction is delivered. Understanding the end-to-end behavior of the customer is truly aligning your testing to the context of why the system is being built.

One final point on why using Test Scenario Design Models is key to Agile Quality success is that they are easy to create. We like to use the Gherkin language to help us provide a consistent business language that is easy to write, read, and maintain. Leveraging Gherkin allows you to both connect your models back to the user story and develop fairly detailed traceability to the acceptance criteria. Those teams practicing Behavior Driven Development can leverage the Gherkin syntax to feed into their Feature Files of creating automated tests through tools like Cucumber, SpecFlow, and JBehave. We've seen how creating Test Scenario Design Models has reduced test case creation by over 25% or more depending on how long you've used them. In addition to taking less time, we've seen a reduction in the number of tests written by as much as 8 to 1. As more and more organizations embark on the journey to DevOps, Quality teams are going to be put under considerable pressure to reduce test execution time and parallelizing the

test execution will only save so much time. It's going to be up to the Quality teams to achieve a dramatic reduction in the number of test cases in order to support quick deployment times key to DevOps.

4. How to Build Test Scenario Design Models?

When building Test Scenario Design Models, start by doing the following:

- List possible users, analyze their interests and objectives
- List "system events" and understand how the system handles them
- List "special events" and what accommodations the system should make for them
- Work beginning to end and break scenarios down into "transactions" tied to UI/UX, data, integrations, and risk

It's important to note, per Cem Kaner, that your scenario is credible. Kaner writes "it not only could happen in the real world; stakeholders would believe something like it probably will happen (Kaner 2003, 2)". Buwalda similarly notes that his Soap Opera tests were written to come up with "stories based on the most extreme examples that had happened, or that could happen in practice (Buwalda 2004, 32)". Avoiding the "what if" game (when testers sit and think about all the "what ifs") of test case design is the key of building the right level detail into the Test Scenario Design Models.

4.1 Getting Started with a Narrative

When we teach people to create Test Scenario Design Models, we start by getting them to write a short story about an experience most people have had before; buying something online at Amazon. We give them a few simple instructions:

- Avoid focusing on details like buttons, instead focus on user behavior
- Identify what event is the starting point and what event is the end point of your story
- Call out key data points

Here's an example of what we see:
Merrick received an e-mail from Amazon about the latest iPhone accessory. He was hooked, he had to have it. He clicked on the link provided and landed on the page for the accessory. He chose his color (red) and initiated the checkout process. Unfortunately, Merrick couldn't remember his password to log-in so he had to initiate the reset password process. He did that and received an e-mail which gave him his temporary password. He reset it, logged-in, and went to buy his accessory. Before he could check out, Merrick had to add credit card information since he hadn't saved it the time before. He entered his credit card info and clicked buy to confirm his order. Apparently, everyone who received the bulk e-mail rushed to buy the accessory and since Merrick had to reset his password, the red-colored accessory was out of stock!

4.2 Breaking It Down to Transactions

Once we have stories created, we ask our students to break down the stories into discreet transactions. For the example above, here's how it looks broken down by transaction:

- Merrick received an e-mail from Amazon about the latest iPhone accessory
- He clicked on the link provided and landed on the page for the accessory
- He chose his color (red) and initiated the checkout process
- Unfortunately, Merrick couldn't remember his password to log-in so he had to initiate the reset password process
- He did that and received an e-mail which gave him his temporary password
- He reset his password
- Logged in
- Navigated to his accessory
- He entered his credit card info and clicked buy to confirm his order
- The accessory was out of stock

4.3 Using Gherkin

Once we've broken down the story in to discreet transactions, we ask our students to focus on flushing out the details for each transaction using the Gherkin syntax of Given, When, Then. For the example we are using, the Gherkin would look like this for the first discreet transaction:

- **Given** Merrick has received an email from Amazon with a link to an iPhone accessory
- **And** he has clicked on the link provided in the email
- **And** he has been taken to the Amazon page for the iPhone accessory
- **When** he adds the Red accessory to his cart
- **And** clicks on the <Complete Order> button
- **Then** Amazon will initiate the checkout process

4.4 Layering on UI/UX

With the Gherkin created, we have our students focus on layering the UI/UX underneath it to guide testers on test execution. Note: the guidance shouldn't be detailed steps, they should be more focused on providing identifiers or behaviors to be used to imitate the user. For the example above, here's what the UI/UX guidance looks like:

- UI/UX: Email
- UI/UX: Amazon iPhone accessory page
- UI/UX: <Add to Cart> button
- UI/UX: <Complete Order> button
- UI/UX: Login or Proceed as Guest page

4.5 Adding the Data Layer

The next step to creating a Test Scenario Design Model is to identify, at a high level, the data required to complete a transaction. When you work through the data, it's important to make sure you consider the inherent (what the data looks like when it is in its raw, newly created form) and systematic (what the data looks like after it has been handled by a system) nature of the data. It's also important to include considerations for data syntax (structure) and data semantics (form) as you add the data layer to your Test Scenario Design Model. If we continue with our previous examples, you would add the data layer in the following way:

- Data: Temporary password (syntax & semantics)
- Data: Username (syntax & semantics)
- Data: Valid, corresponding password (syntax & semantics)
- Data: Credit card information (syntax & semantics)

When you are identifying the data needed for your Test Scenario Design Model, avoid falling into the "find every possible combination" trap. The modified condition/decision coverage (MC/DC) testing approach shows that even with testing each entry and exit point, the number of test cases needed for complete code coverage is finite. Here's an example we use to illustrate this point with our students. We ask our students the following question: if you are testing login functionality with just username and password, how many test cases do you need to test the login functionality completely? Before you (our readers) answer the question, let us give you some of the answers we've seen in practice at a number of different clients. We've seen up to 100+ test cases used to test login functionality. When we ask why these clients have so many test cases for validating login functionality, we always hear the same answer; "we make sure we test every combination possible". When we tell them the correct number of test cases is 4, they do not believe us and ask us to prove it. Here's the mathematical proof for why there are only 4 test cases:

- Test case 1: Username = T, Password = T, Result = Pass
- Test case 2: Username = T, Password = F, Result = Fail
- Test case 3: Username = F, Password = T, Result = Fail
- Test case 4: Username = F, Password = F, Result = Fail

Note that every combination of "business rule" applied to this proof is simply repeating one of the above 4 test criteria. For instance, one case we get all the time is "what if the username has a special character in

it?" If your username field accepts special characters, it invokes test case criterion #1 and if the username field does not accept special characters, it invokes test case criteria #3 or #4.

4.5 Integrations, Integrations, Integrations

Once the data layer has been added to the Test Scenario Design Model, add integration areas for your transaction. As modern systems have become increasingly complex, the number of integrations within software systems has dramatically spiked, as has the importance of testing these integrations. When you add integrations, be sure to focus on sources of data, data locations, services, and security attributes or protocols. For the example above, the integrations would look like this:

- Integration: Bulk email
- Integration: Content manager
- Integration: Authentication service
- Integration: Add product to cart service
- Integration: Credit card validation service
- Integration: Checkout service fulfillment

If you find yourself identifying upstream and downstream dependencies, you are on the right track!

4.6 Don't Forget Risk

The final piece of a Test Scenario Design Model is risk. Risk, used here, is focused on prioritizing transactions based on business value. This definition of risk is sometimes difficult for tester to understand because they equate risk to severity or likelihood of failure. When we ask our students what level of Risk they would attribute to the login transaction, the most common (and by far the most immediate) response is high. When we ask "why" they gave that response, our students say login functionality is critical for a website. When we say that in the Amazon example the Login transaction is low risk, we get a lot of incredulous looks. We remind our students that Amazon's core business is to sell something to its customers. Whether or not those customers are signed on or not does not make any difference to Amazon because they can complete a purchase as a guest. Therefore, the login transaction does not represent a significant risk to Amazon's business if it does not work. At this point, we remind our students about the importance of context to testing especially as it applies to user-centric design.

As you look to assign risk to your transactions, do not lose sight of the fact that integrations should play an important part in determining your risk designation. Transactions that feature in many of your Test Scenario Design Models should always have a higher risk designation. If we refer back to the example of Amazon, the payment transaction would be high not only because the business value is high but also because every business critical scenario for Amazon would leverage that transaction.

4.7 Putting It All Together

If we put everything together from our previous sections, our Test Scenario Design Model would look something like this:

Transaction 1

Given that Merrick has received an email
When he clicks on the embedded link
Then the default browser should launch and the iPhone accessory page should load

UI/UX: Email
UI/UX: Embedded link
UI/UX: iPhone accessory page

Data: None

Integration: Default browser
Integration: Accessory page

Risk: Medium – if email doesn't work, the user could go directly to Amazon and find the accessory by searching

Transaction 2

Given that Merrick is on the accessory page
And he's selected the color red for the accessory
When he clicks on the <Buy> button
Then Amazon loads the Login/Guest Purchase page

UI/UX: iPhone accessory page
UI/UX: Color option drop down box
UI/UX: Buy button
UI/UX: Login/Guest Purchase page

Data: Red

Integration: Order management service
Integration: Login/Guest Purchase page

Risk: High – being able to add items to cart and proceeding to the purchase page is critical

Transaction 3

Given that Merrick is on the Login/Guest Purchase page
When he clicks on the <Reset Password> link
Then Amazon loads the Reset Password page

UI/UX: Login/Guest Purchase page
UI/UX: Reset Password link
UI/UX: Reset Password page

Data: None

Integration: Reset password page

Risk: Medium – Being able to reset a password is important but does not stop a user from completing a purchase

Note that this Test Scenario Design Model is not complete, we've only included three of the transactions as a sample. The full Test Scenario Design Model is actually 7 transaction. We've boxed up the transactions like a card which can be helpful if you are running Kanban (each box could be a Kanban card) or if you like using a physical Agile board (treat each box as a testing task to move along the board until it is done) but it's not necessary to do this.

Before we end this section, we want to highlight a few keys points from the Test Scenario Design Model we've included here as our example. The first point to highlight is how the **Given** statement for Transaction 2 and Transaction 3 are the **Then** statements from Transaction 1 and Transaction 2 respectively. Note that a link is created between the transactions but that the link is not creating a dependency as each transaction could be run independent of the linked transaction. For instance, Transaction 3 can be run at any point in time as long as the user has something in their cart and they proceed to checkout. You can see that there are also links in other areas of the transaction like in the UI/UX and Integration details but that these links are also independent of each other when it comes to test execution. This point is important because one of the features of Test Scenario Design Models is that they are built in a modular way so that you can easily juxtaposition individual transactions in order to create more scenarios and expand system coverage quickly.

The second point to highlight is the language and syntax used to create each transaction. The Gherkin language gives each transaction description a consistent and easy to follow narrative. That narrative provides a tester with a clear test objective and a concise criterion by which they can determine if the Test Scenario passes or fails. The syntax is succinct but comprehensive, it gives the tester enough information to guide their exploration of the system but also provides enough flexibility for the tester to use their analytical skills and experience to identify defects.

The third point to highlight in this example is that fact that not every transaction has to have details at every layer. In this example, Transaction 1 and Transaction 3 have no data details because no data is necessary as an input for the transaction to be completed. Avoid the compulsion to add details to every layer for the sake of "completing" the transaction details because Test Scenario Design Models emphasize the minimal viable product principle.

Final, the hardest part of teaching people how to create Test Scenario Design Models is getting them to break from the old paradigm of testing that emphasizes quantity over quality. You do not need endless numbers of test steps to figure out how to test a system, you simply need high quality guard rails that keep you on the right path. We often get pushback here from people who ask "how can you completely test the system if you don't explicitly call out what to test in detailed test cases?" It's a fair question especially for organizations that rely heavily on a traditional testing approach where much of the testing is conducted manually. However, according to recent studies like the one done by Dimensional Research, over 87% of QA teams say they have both executive and financial support to implement and use test automation (Dimensional Research 2018, 2). Test automation is an important factor here because the right strategy around test automation eliminates the need to worry about "completely" testing a system. Take for example a team that is leveraging data models and iteration factories in the test automation they write. By virtue of leveraging a data model and iterating data through it, an automated test can literally execute hundreds of permutations of data inputs in a single test run. Making use of Test Scenario Design Models does not mean you move away from good fundamental testing strategies that combine the right balance of people, process, and tools. Test Scenario Design Models enable QA teams to focus less time on meaningless tasks like writing thousands of manual test cases so that they can focus more time on high value, user-centric test execution and test automation.

5. Test Scenario Design Model Metrics

As we worked to help organizations implement Test Scenario Design Models, one thing has become clear to us: traditional "testing" metrics are not going to work as a reporting mechanism. This fact was made clear to us when our clients asked us to provide statistics like how many test cases had we created and how many test cases were being run every day. Since the emphasis of Test Scenario Design Models was to reduce the number of test cases being written while increasing the coverage of critical business

scenarios, we could not report on simple test case counts. We started to move the reporting conversation away from numbers of test cases and toward operational readiness by business function. By directing the conversation to readiness, we were able to talk more directly about quality measurements instead of testing statistics. When we tied in test automation numbers, the product owners became excited about the fact that not only were business critical scenarios being tested in an end-to-end manner but that a small but powerful set of business-value driven test cases were being run on at least a daily basis. We continuously pushed organizations to emphasize customer satisfaction as the most important measure of success and since POs were satisfied, traditional testing metrics became a thing of the past.

5.1 Defect Removal Efficiency

As the need to focus on traditional testing metrics faded, we were able to start focusing on quality metrics like defect detection efficiency, defect removal efficiency, volatility index, and quality throughput. One of the most important metrics you can report on while leverage Test Scenario Design Models is defect removal efficiency (DRE). “The DRE metric measures the percentage of bugs or defects found and removed prior to delivery of the software (Jones 2011, 1).” Why we find this metric important is because it is focused on removal of defects, not just the detection of them. The one variation on this metric that we always add for Agile teams is to measure not on production escapes but by escapes into User Acceptance Testing. By measuring against UAT, we are able to see how well our quality strategy is performing in removing defects before our customer uses the software in two-week increments. The timing of the data allows us to pivot our quality strategy to focus on the activities that help remove the most defects on a frequent basis.

5.2 Defect Detection Efficiency

Understanding how well your quality strategy is performing at defect detection is important for measuring the success of Test Scenario Design Models. That is why we typically like to combine our DRE metric with defect detection efficiency (DDE). The DDE is “the number of defects detected during a phase/stage that are injected during that same phase divided by the total number of defects injected during that phase (http://softwaretestingfundamentals.com/defect-detection-efficiency/).” That is to say “the ultimate target value for Defect Detection Efficiency is 100% which means that all defects injected during a phase are detected during that same phase and none are transmitted to subsequent phases (IBID).” Providing metrics that show defects injected, detected, and removed by phase depicts the level of quality maturity that you are operating with especially as you transition into leveraging Test Scenario Design Models. One additional metric that we use is a pareto analysis of defect detection activities such as static code analysis, unit testing, functional testing, regression testing, user story grooming, etc. The Pareto diagram of the top defect detection activities allows product teams to rapidly pivot to activities that are helping the team identify defects as quickly as possible.

6. Conclusion

As we have guided our clients in their transition to Agile, we have most prevalently seen the challengebreaking from the old testing paradigm that favors quantity of test cases over the quality of test cases. We have sought to bring forth a way where QA teams can focus on building scenario-based outlines of user behaviors in a global narrative to connect product functionality to user-centric designs in order to test We believe that Test Scenario Design Models is that way forward. Test Scenario Design Models allow you to build succinct but comprehensive end-to-end scenarios that test systems and software based on business value priority.

Reference:

Adaptavist. *Future of Automation Report*, https://www.adaptavist.com/atlassian-apps/future-of-automation-report/, 2018.

Black, Rex. *Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing*, Wiley, 2009.

Beck, Kent. *Test-Driven Development*, Addison-Wesley, Boston, MA 2002.

Buwalda, Hans. *Better Software*, www.stickyminds.com, February 2004.

Capgemini. *World Quality Report*, www.worldqualityreport.com, 2018.

Dimensional Research. *Testing Trends for 2018: A Survey of Development and Testing Professionals, Feb. 2018.*

International Organization for Standards. ISO 16085, https://www.iso.org/standard/40723.html.

International Organization for Standards. ISO 25010, https://www.iso.org/standard/35733.html.

International Organization for Standards. ISO 25012, https://www.iso.org/standard/35736.html.

International Organization for Standards. ISO 25030, https://www.iso.org/standard/35755.html.

International Organization for Standards. ISO 25040, https://www.iso.org/standard/35765.html.

International Software Testing Qualifications Board. https://www.istqb.org/, 2018

Jones, Capers, Bonsignour, Olivier. *The Economics of Software Quality*, Addison-Wesley, Upper Saddle River, NJ 2012.

Jones, Capers. *Software Defect Origins and Removal Methods*, www.namcook.com, 2013.

Jones, Capers. *Software Defect Removal Efficiency*, https://pdfs.semanticscholar.org/4e1b/c72a0324664e46e1404c6dc5ce3ed266020d.pdf, 2011.

Kan, Stephen H. *Metrics and Models in Software Quality Engineering*, Addison-Wesley, Upper Saddle River, NJ 2005.

Kaner, Cem. *An Introduction to Scenario Testing*, http://www.testingeducation.org/BBST/testdesign/Kaner_ScenarioIntroVer5.pdf, April 2013.

PractiTest. *State of Testing Report 2018*, http://qablog.practitest.com/wp-content/uploads, 2018.

SmartBear. *The State of Testing 2018,* https://smartbear.com/resources/ebooks/state-of-testing-report-2018, 2018.

VersionOne. *The 12th Annual State of Agile Report*, https://explore.versionone.com/state-of-agile, 2018.

VersionOne. *The 13th Annual State of Agile Report*, https://explore.versionone.com/state-of-agile, 2019.