

Holistic Testing: Weave Quality Into Your Product



**HOLISTIC
TESTING**

By Janet Gregory and Lisa Crispin

<https://agiletestingfellow.com>

Copyright 2023, Janet Gregory and Lisa Crispin

Holistic Testing

Contents

Introduction	3
What is quality	3
Quality definitions.....	3
Approaches to quality	4
Final thoughts on quality	5
Overview of the Holistic Testing Model	6
Stages of the Holistic Testing Model	8
Creating a quality and testing strategy	10
Whole-team approach to quality.....	10
Working towards quality objectives.....	10
Building in quality using the left side of the Holistic Testing Model	12
Checking what you built using the right side of the Holistic Testing Model	23
Getting the whole team involved.....	30
Assessing quality	30
Measuring product quality.....	30
Measuring quality practices	31
Quality practices assessment.....	31
How leadership and culture impact quality	33
Culture.....	33
Leadership and communication.....	33
Support for quality	34
Summary	35
Resources for further learning	36
Our Holistic Testing Courses.....	36
Our YouTube channel and websites	36
Resources referenced in this mini-book.....	36

Introduction

Holistic testing is a comprehensive term to encompass testing throughout the development cycle from the start of an idea, through to customer interaction with a delivered feature. All members of a delivery team think about testing and quality from the beginning and throughout the cycle. They engage in all kinds of testing, including how to instrument their code to provide information about how it's really behaving in production.

This mini book is about the Holistic Testing Model and how to develop a quality and testing strategy for the whole team. By whole team, we include all specialities. UI/UX designers, product owners, programmers, testers, database experts, business analysts, as well as site reliability and platform engineers. Some organizations may have specialties in separate teams rather than embedded on cross-functional teams – for example, security, performance or load testers. In our model, they collaborate closely however the teams are organized.

Many thanks to fellow practitioners who reviewed our drafts and gave us valuable feedback: Gáspár Nagy, Bertold Kolics, Henrique Mergulhão, Lisi Hocke, and Nolan MacAfee.

What is quality

In the software world, we talk a lot about quality. Business leaders say they want the best quality product – though they often fail to understand how investing in quality pays off. Customers have their own views of what quality means to them, which may be surprising to the business. Delivery teams are concerned about code correctness and the many types of testing activities.

In this mini book, we use the term quality management rather than quality assurance. Information from testing can help improve the quality but cannot 'assure' the quality is there.

There are many contexts, and each may need a different way of looking at quality. However, in each case, quality needs to be built into the product from the beginning, with the customers part of the process.

Quality definitions

There are many definitions for 'quality', and the most popular definition seems to be the one from [Jerry Weinberg](#) -"Quality is value to some person." This may be too simplistic and understates some of the dimensions that we should be thinking about to create a good quality strategy.

[W. Edwards Deming](#) defined good quality as: "a predictable degree of uniformity and dependability with a quality standard suited to the customer." Dan Ashby wrote a post about [Crosby's 4 absolutes of quality](#) showing how they are still applicable but need to be adjusted to fit into a software context. And David A. Garvin talks about [five dimensions of quality](#).

Holistic Testing

People talk about quality as if there is only one kind – all encompassing, but it’s not. For example, product quality is about the quality of the finished product – what our customers experience. When teams try to measure quality, it is usually about how well they adhere to their processes, which is really something different.

Approaches to quality

Product quality can be looked at from many perspectives. The customer’s perspective is only one – and different customers have different needs and wants and is from an external viewpoint. There are other stakeholders who also have an interest in a product’s quality. There are many lenses in how people view quality.

Process quality

In the development cycle, the emphasis is on the practices followed to build the product. The focus is on defect prevention and limiting rework, and this is often where many teams spend their time building it right and measure the quality of the process. Many teams target testing in this perspective.

Some processes that support quality are: TDD (test-driven development), coding for maintainability, peer reviews, continuous integration, exploratory testing or even conforming to the “Definition of Done”. Teams measure process quality, the technology-facing quality that the team owns, the practices that help them build quality in. They aim to answer the question: “Are we building it right?”

Product-based quality

People often assume that using good ingredients should make a good product. If quality is viewed as an inherent characteristic of goods, higher quality means better performance, and enhanced features – things that may increase the cost of the product. This thinking may be slightly flawed. Using the metaphor of cooking, an ordinary cook with good ingredients may not have a good outcome, but a great chef with ordinary ingredients may produce something magical. We need to understand what our product is and how it is put together.

Some processes that teams can do to help support product quality are: ATDD (acceptance test-driven development) or BDD (behaviour-driven development). Testing quality attributes such as security, performance or reliability is another example. The question to be answered is: “Does it work as expected (or desired)?”

User-based quality

The user-based perspective is what is most often used to talk about quality, and it is highly subjective and highly personal.

It assumes that consumers possess sufficient information to evaluate product quality. If they do not, they will rely on other cues when making that assessment. Let’s consider a cup of coffee. Some folks prefer a simple black medium roast coffee (Figure 1a) to a well-made cappuccino (Figure 1b), but others take the cappuccino every time. Who is your consumer? Who is evaluating

Holistic Testing

your product's quality? Do you need to satisfy most consumers? Or target a specific group and satisfy that group only?



Figure 1a: simple black coffee



Figure 1b: cappuccino

Processes that support user-based quality include user experience (UX) designers working with customers to learn what they want, ATDD, testing using customer personas, A/B testing, accessibility testing, observability and exploring analytics of production usage. The question to be answered, "Is this what our consumers want?"

Final thoughts on quality

We really like this phrase "quality is woven into the fabric." It's a great visual (Figure 2) that goes along with our more common phrase of "building quality in."

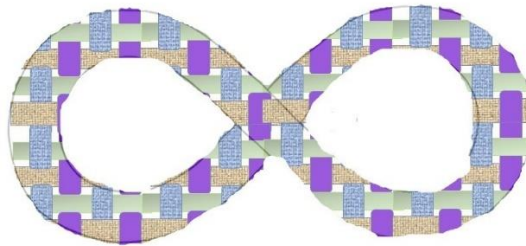


Figure 2: Quality is woven into the fabric

We often think about quality as costing something... extra processes, extra work. But, a lack of quality may be far more expensive, if we don't consider both tangible and intangible costs. There is loss of opportunity, or erosion of trust and reputation. Once lost, these are hard to get back. Start by analyzing and understanding your risks. That is where your most valuable testing begins. Think about how you define quality. If you want to explore more, please read some of the articles mentioned earlier.

In the next section on Quality Strategy, we show how testing supports the quality discussion for both product and process quality. There is a close relationship between testing and quality, but they are not the same.

Overview of the Holistic Testing Model

Software development happens in a continuous cycle. The DevOps loop in Figure 3 is a popular illustration of this infinite cycle. It has one problem – it shows “test” as a separate phase in this continuous loop.

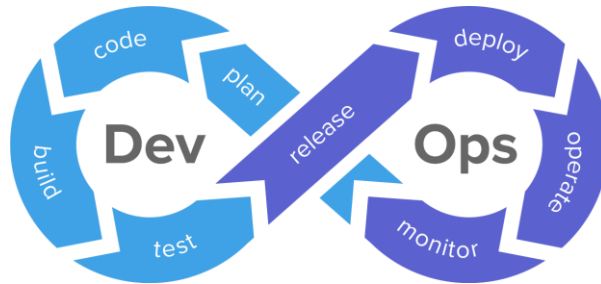


Figure 3: Typical DevOps loop

In modern software development, testing is not a phase. Testing is one of the many activities in software development, along with coding, planning, design, architecture, operations, and more. Janet crafted her own depiction of the development lifecycle, shown in Figure 4. The stages are not only different from those specified in the DevOps loop, but most importantly, there is no segregated testing.

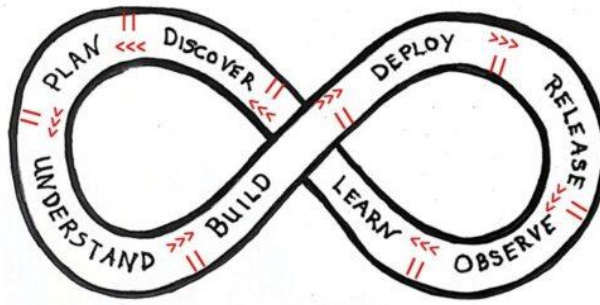


Figure 4: Janet's version of the development loop

This loop provides the basis for the Holistic Testing Model. The pauses (shown as ||) are a simple way to illustrate the iterative process of agile software development. For a variety of reasons, some development stages may go slowly, some may go quickly. Sometimes a team needs to pause and reflect on what is happening. They might decide to go back a stage or two. For example, as they're building a new feature, they may realize some information is missing, or even that the feature idea is flawed. It's time to go back to the planning or discovery stage. Maybe they deploy a new change to a test environment and see unexpected behavior and go back to building or understanding to fix whatever is needed.

You may ask “Where is the testing?” It's in every stage of the development cycle. Taking the idea of Dan Ashby's continuous testing loop and his [“we test here and here and here”](#) diagram, we added examples of the types of testing activities that might happen throughout the cycle to the

Holistic Testing

development loop. That's the Holistic Testing Model. The examples in Figure 5 are not meant to be an exhaustive list of all testing activities that can be performed.

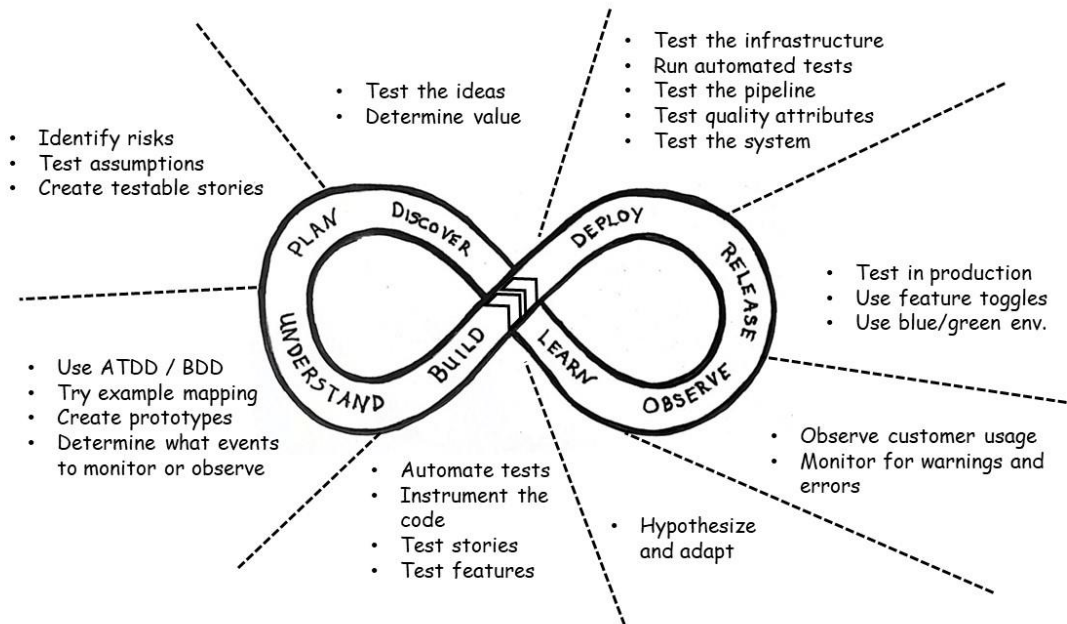


Figure 5: Holistic Testing Model with examples

The left side of the loop shows testing done early in the cycle. It creates shared understanding that guides development and prevents defects. The right side of the loop is about testing what has been built, and finding defects that escaped the build process. It's also for learning from customer usage and gathering data to help plan the next changes.

A graphic that we use in our *Holistic Testing: Strategies for Agile Teams* course shows how testing fits into a software development lifecycle (SDLC). Figure 6 shows how the stages in the Holistic Testing Model would map to that graphic.

Holistic Testing

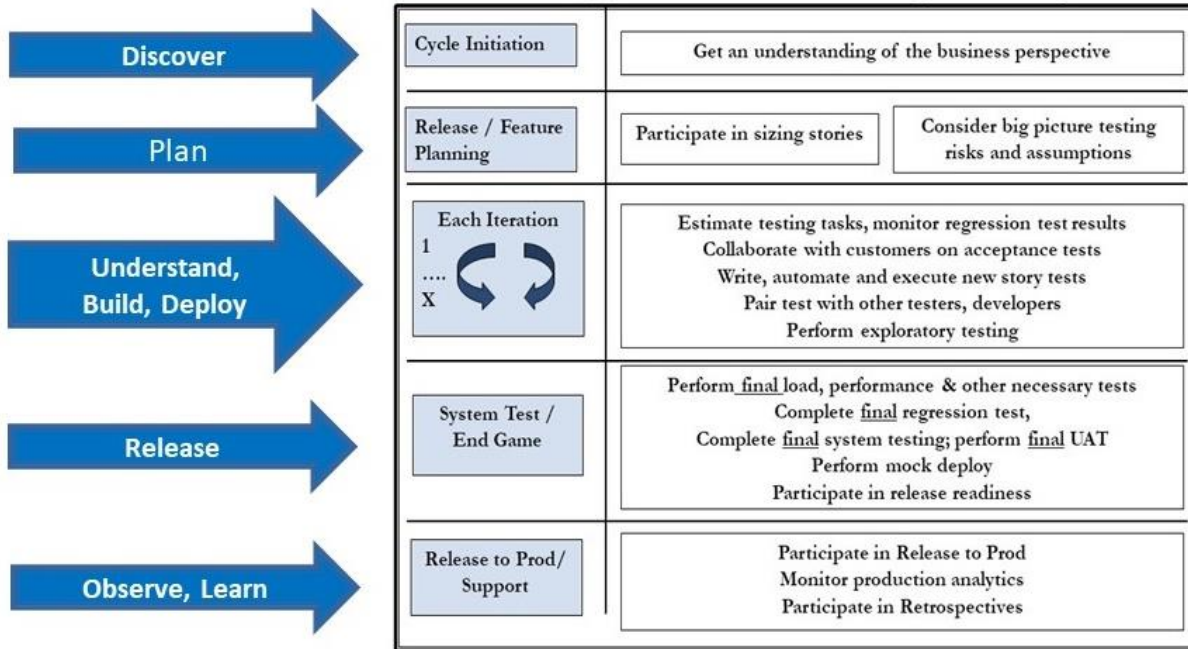


Figure 6: Mapping stages of the Holistic Testing Model to how testing fits in an agile SDLC

Stages of the Holistic Testing Model

In this section, we explain the stages at a high level. The sections on creating a quality and testing strategy go into more detail about the types of techniques you may choose to include in your strategy.

Discover: This is a time to look for value to the business to understand the business perspective. What new changes will help customers solve their problems? Activities here might include product managers talking with customers to see if a feature worth pursuing. It's a time to test the feature ideas for business value. Delivery team members may challenge ideas, point out risks, and raise questions to help the business to make decisions. UX designers may do some research in the field to help as well.

Plan: During planning, teams identify risks to determine which quality attributes will be important to the feature. Features are sliced into smaller stories during story readiness workshops or backlog refinement sessions. This is time to advocate for testable stories and flush out hidden assumptions. Teams may start looking at high level acceptance tests and start to get a big picture understanding of the stories. Organizations with multiple teams may organize a workshop to discuss dependencies between teams and how they will be managed.

Understand: Having planned features, stories, and other changes at a high level in the planning stage, the goal in this stage is to build shared understanding of the story details. This starts the iterative cycle of working on and delivering user stories, a process which will continue in the building and deploying stages to follow. Practices like acceptance test-driven development or

Holistic Testing

behaviour-driven development are part of understanding the stories. Teams may involve user experience experts to learn about customer needs. Programmers may help by creating low-fidelity prototypes for fast, iterative feedback.

Conversations around the different quality attributes, such as performance, reliability and security, consider what data or tools they may need for testing those attributes. This is the time to plan what data and events to log for later monitoring and observability.

Build: This is where the team implements user stories and features – perhaps practicing test-driven development (TDD) and using fast feedback principles to test as soon as possible. Small testable stories make these practices easier to apply. Ensemble work, pairing, code reviews and test reviews take place. The examples created earlier can be used to create executable tests that guide development. These automated tests can become part of your regression suite to give fast feedback to detect regression failures. The team also builds instrumentation into the code to capture log data and events for monitoring and observability, as they discussed in the previous stage.

Teams perform exploratory testing to uncover what they didn't think about when they were developing. Understanding the version control system and workflows help determine what testing needs to be done to manage technical risk.

Deploy: Many teams still struggle to have stable test environments. As organizations transition to virtual and cloud infrastructures, teams can spin up test environments on demand. This can simplify testing some of the quality attributes, such as load or performance testing. Deploying to a production-like environment is a common practice for testing the system. Today's technology enables teams to configure different test environments in different ways to test different quality attributes.

Release: There are many different mechanisms for releasing to the customer these days. Release strategies can provide safe ways to test in production environments. New changes can be hidden from users while the team tests without impacting them. As changes are turned on for a subset of users, the team can monitor activity and watch for problems. Once the team feels confident about the changes in production, they can release them to all users.

Observe: The team instruments their code to capture events that enable them to detect and understand any problems quickly (observability). It's a way to watch how the customers use the product and allow the team to respond accordingly. Teams can also add telemetry to the code that lets them monitor system behavior and send alerts and warnings when problems occur.

Learn: As teams observe how customers use their product, they can hypothesize about how to improve it. Testing is a big part of this learning. Resulting ideas and experiments feed into the next discovery sessions.

Each stage in the infinite loop is not meant to be entered through gates, but flows naturally – fast or slow, depending on the team and on the story or feature they are building. The next section goes into more detail about how to apply test strategies to each stage, and how to build an overall quality strategy for your team.

Creating a quality and testing strategy

Whole-team approach to quality

The whole team needs to take responsibility for quality. This is a phrase we use frequently to counter the common misperception that testers or the “quality team” own quality. It also should not be only the software delivery team that owns quality. People at every level of the organization have a different part to play. Figure 7 shows some places where people contribute to quality.

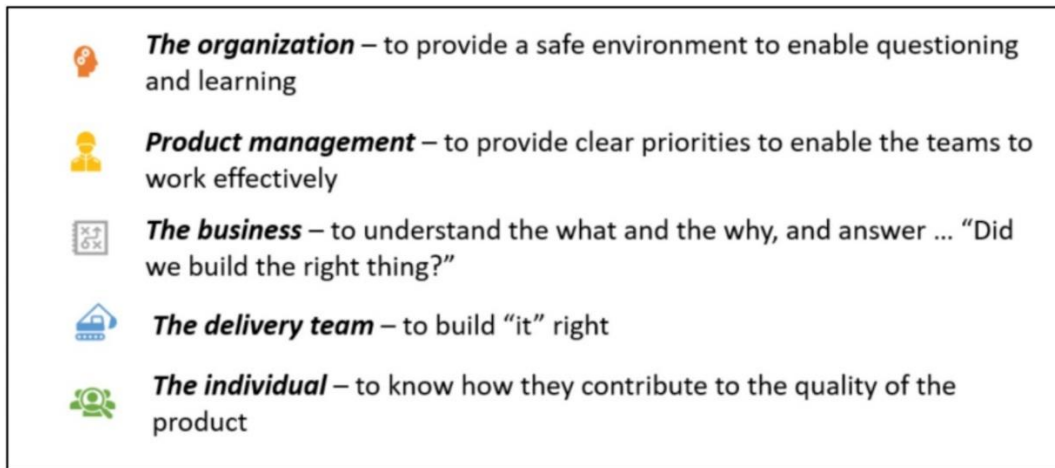


Figure 7: Who owns quality?

It starts at the organization level. By having psychological safety in place, where the ‘blame game’ is not played, teams feel that they can experiment without getting punished for failures.

Many organizations have a “Center of Excellence” of some kind that mandates processes that teams should follow. These can be useful if they focus on offering advice and coaching which promotes learning instead of prescribing a process. Teams that don’t see the value in a process will find a work-around or else blindly follows something they don’t understand and don’t get the expected result. We’ve seen much more success when teams start small, evaluate, and adapt, getting better with each small step forward. Even when experiments aren’t as successful as you had hoped, or they fail altogether, the cost of learning is low because they are small.

Working towards quality objectives

An organization’s leadership should define the level of quality they need or want. This needs to be something specific and clear that everyone can understand. Examples of quality goals are a certain net promoter score based on customer surveys, and service level objectives such as 99.999% availability. The product teams then define how to achieve that level. They may start with a basic definition of *Release Done*. Of course, there are exceptions. Not every team can do

Holistic Testing

exactly what they want if they need to work in conjunction with other teams. They need to coordinate, manage dependencies, and find common processes. Perhaps they have a common definition of *Feature Done*, and each team has their own definition of *Story Done*.

Product management folks need to coordinate their efforts so that the most important pieces of work get done in a timely manner. Each software team needs to have the right people and resources to do their work. A team that's held to unrealistic deadlines and lacks essential skills and tools is not likely to focus on building quality into their software. From a business perspective, features are the highest priority, but teams also may need time to manage technical debt – deficiencies in internal quality that make code hard to maintain, such as code not covered by automated regression tests - that is slowing them down. This backlog of rework will drag down the team, make it harder to build new features, and erodes product quality.

Business stakeholders, whether they are product owners, business analysts, sales, or marketing, need to understand the purpose of each new feature, and the problem it will solve for the business or its customers. They need ways to communicate this clearly to the delivery teams. The delivery teams can then concentrate on the most effective technical implementation. They own their development processes and practices, so they can build quality in from the beginning. There are many ways business stakeholders and delivery teams can collaborate and build shared understanding to make sure the right thing gets built. For example, frameworks like story mapping and example mapping help people in a wide range of business and technical roles build shared understanding. Iteration demos are a way teams can get fast feedback from stakeholders to know if they've met the business needs.

Other folks that may want input into the quality are governance – they care about regulatory conformance, or perhaps security. Look around your organization to understand how people in different roles or goals may perceive quality. Start that conversation if you haven't already.

Every individual in an organization should be able to articulate how they contribute to the quality of their products. The next section is about techniques and approaches to help a team build a quality and testing strategy.

Holistic Testing

Building in quality using the left side of the Holistic Testing Model

The left side of the Holistic Testing Model (Figure 8) is about building quality into the process and the product. The right side is testing to see that teams got it right and adapted if they didn't. The model balances testing early with testing after code is built.

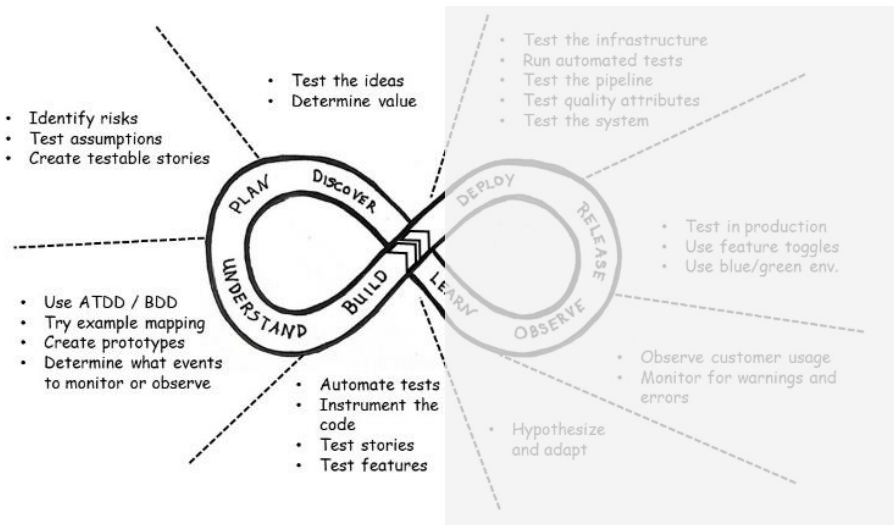


Figure 8: Left side of the Holistic Testing Model

It's not uncommon for a delivery team to discuss a story during the iteration planning meeting, give an estimate, and start working on it. They do a great job of coding and testing the story using good practices and delivers it to the product owner for acceptance. The product owner rejects it with a puzzled comment: "This isn't what I asked for." So frustrating! The delivery team understood the story one way – the product owner understood it a totally different way. They failed to share the same view of the outcomes.

When teams work together with product folks and others in the discovery and planning activities, they start to build shared understanding. Once the high-level picture is established, it's time to encapsulate that shared understanding in artifacts that help build the right thing. In our experience, there are a few factors to succeed with this effort.

Using the Holistic Testing Model, we cover specific practices to use in different stages of the lifecycle to help plan a team's quality and testing strategy.

We start at the beginning and extract the left top part of the infinite loop to expand upon testing early – in discovery and planning – Figure 9.

Holistic Testing

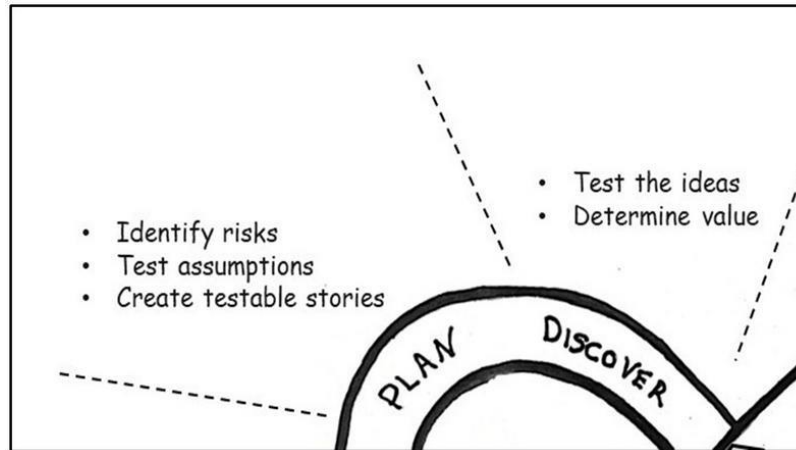


Figure 9: Discover and planning in the Holistic Testing Model

Discover

There are many techniques to bring diverse team members together to talk about testing and find ways to deliver better outcomes for our customers. Because every organization and team have different contexts, think about your own team's test ecosystem as you read.

Product managers engage in discovery activities to determine what value a particular feature offers to their customers and the organization, and what that might look like. They test the idea to see if it is worth pursuing, and if it fits the company's goals and vision. Techniques like [Impact Mapping](#) can help organizations with this. Delivery team members may also work with customers, sit in on sales calls, and other activities to better understand user needs. They contribute their ideas and questions during discovery sessions.

Consider your company's context. A new start-up without existing customers needs different ways to research what potential users value. Established companies can analyze historical data and interview users. Getting programmers, testers, and other specialists involved along with product people and designers can lead to better feature ideas.

Plan

In the planning stage, the delivery team can show their strength asking questions, and offering suggestions based on their own experiences, system understanding, and domain knowledge.

Visual tools like mind maps can help to visualize how big the feature might be. What does it consist of? Keeping the implementation details out of this discussion helps focus the discussion on business value. Once these aspects of a feature are visible, teams and the product manager can make decisions about what is immediately needed, and what might be pushed out to a later release, or not needed at all. A team can test this artifact by questioning some of the assumptions behind the ideas.

Once the big picture is known, teams plan in more detail. Identifying risks is a big part of the value added here. Lisa has a blog post about [techniques to help teams](#) brainstorm about risks.

Holistic Testing

Testing early includes testing assumptions. We all have biases and make assumptions based on our own experiences.

Janet wasted about an hour when working with a new client based on an incorrect assumption. Fortunately, she realized her error and sent her client a quick email to check that assumption. She could have wasted a lot more without that simple test. If you think you have a different impression of a feature or story, ask the question. Be brave.

In planning, teams break features into stories – smaller bits that are easier to digest. However, often these stories are not testable. This leads to stories that need to wait for other stories to be complete before they can be tested. This results in delays, bottlenecks, slow feedback.

Story mapping is one technique to help break features into user stories. Another is using flow diagrams to visualize the flow, make assumptions visible, get a shared understanding and help break features into testable stories. The example flow diagram in Figure 10 shows how a team could address many complexities in a feature. First identify the core – a slice through the application (or as we like to call it, the steel thread). Often this is the happy path. As complexities (new stories), are identified, many times the programmers push back. They may be used to splitting stories based on components, building one complete piece at a time.

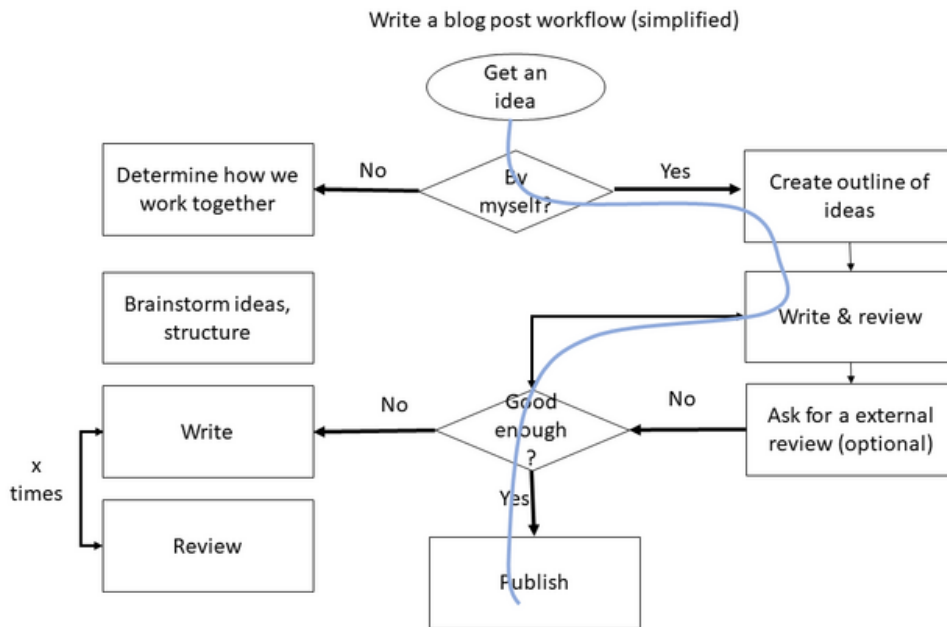


Figure 10: Flow diagram for writing a blog post

If you hear comments such as, “But that’s not how we code. We always do the complete configuration first, but now you are asking us to go add to that file for every story,” repeat the

Holistic Testing

reason for visualizing the flow. The goal is to get testable stories, and to get faster feedback from testers, product owners or other team members.

Small, testable stories are the key to being able to complete the stories in a timely way. Most teams realize very quickly the importance of working with the whole team to slice the stories into thin slices through the functionality. Testable stories help to remove integration issues.

Visualization activities like mind mapping and flow diagrams are structured ways to explore requirements. The key is getting people with diverse viewpoints and skill sets talking, drawing, and experimenting together.

Structured conversations

Start conversations about quality by discussing what quality means for the company, the product, the team. What level of quality do you need, and then what kinds of testing are needed to support that level of quality? Include a wide range of specialists in these conversations, to get diverse perspectives.

Conversations among people with diverse perspectives

In an [episode of the “Making Tech Better podcast”](#), Lou Downe referred to Melvin Conway’s work, and included this striking thought: “The quality of software is directly related to the quality of conversations we have.” Lou said collaboration is a privilege. Teams need to find ways to have conversations that often circumvent their organizational structures.








For example, a delivery team may use different terminology than people in other parts of the organization such as marketing, data, and design teams. A collaborative discussion helps to ensure they all have a shared understanding about how a particular feature is valuable to the customer, what it should look like, and how it should behave. A safe space to talk with people in other parts of the organization is important.

Structuring the conversations

Once the team has committed to enabling cross-team collaboration, they can take advantage of the many frameworks and tools that enable them to get the most value from these conversations.

It’s important to ask questions. The “[7 Product Dimensions](#)” from Ellen Gottesdiener and Mary Gorman in Figure 11, is a great tool to think of questions related to different quality attributes – along the lines of, “Where will customers be when they use the product? What interface or device will they use? Do we have test data? How many concurrent users do we need to support?” Asking these types of questions help to minimize the “unknown unknowns” (unexpected or unforeseeable conditions).

Holistic Testing

						
User	Interface	Action	Data	Control	Environment	Quality Attribute
Users interact with the product	The product connects to users, systems, and devices	The product provides capabilities for users	The product includes a repository of data and useful information	The product enforces constraints	The product conforms to physical properties and technology platforms	The product has certain properties that qualify its operation and development

Discover To Deliver: Agile Product Planning and Analysis, Gottesdiener and Gorman
 © EBG Consulting Inc., 2012
 With permission for non-commercial use. Not for reproduction or distribution.

Figure 11: 7 Product Dimensions

Teams should also discuss risks. There will be some they can mitigate through testing and others that they can't. What events should be logged in the new workflow, to allow identifying and responding to issues once the feature is in production? What alerts and dashboards would be helpful? The answers add to shared understanding of the technical implementation.

These conversations may also produce low-fidelity prototypes or other visuals that can be used to explain the desired outcomes to other team members and stakeholders outside the team. In our experience, teams are amazed at how much information they were each missing when they tried such a visual exercise. Share the outcomes with others so everyone starts from the same level of knowledge when you have your team-wide planning discussion.

Context diagrams

Context diagrams are a tried-and-true way to understand how your application interfaces with others – humans, machines, APIs, or even other systems. Start by identifying the external entities with which the system interacts. Next, determine the flow of information between your system or application and those entities, looking to see which way the information flows – one direction, or is it bi-directional. For example, Figure 12, shows a context diagram for school administration where the focus is on bussing children to and from school. You can easily see the relationships and can ask questions to determine what is missing.

Holistic Testing

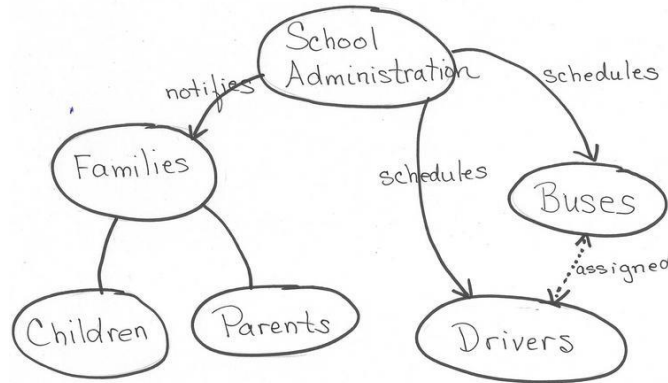


Figure 12: Context diagram for bussing children to school

Dependency Mapping

Another practice that is useful during planning is to visualize dependencies using dependency mapping. One of the biggest areas that cause agile teams to slow down, is not recognizing dependencies early. Often a team doesn't realize until they start to code or test, that they need another piece.

To start thinking about dependencies, choose the core piece of your product and draw it on a whiteboard or flip chart (virtual if needed). Then, represent all the pieces of the system that make that core piece work. One way is to use circles, overlapping each other like a Venn diagram to show touch points. You can use circle size and color coding to represent impact of and ownership. Visualizing the dependencies often reveals bottlenecks that need to be addressed and helps identify the key people who can help. Figure 13 below shows an example of one type of dependency map.

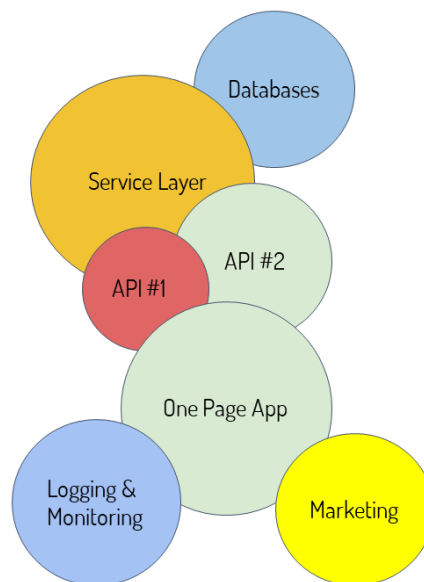


Figure 13: One version of a dependency map

Holistic Testing

You can also look at individual features to see what impacts they may have on other parts of the system. If you can identify that a particular feature touches more than one team, that is a signal that you need to work with those other teams early to mitigate the risk.

Summary for Planning

Not all the techniques listed in this section will be useful for every feature. Experiment to find the ones that work for you and make them part of your quality strategy. Some will work best for complex features – where the team has no previous experience and may have to do a lot of exploring and experimenting. Others for complicated features – a few people on the team have solved the problems before or the solutions are documented. If you are working on a feature that your team considers simple, something everyone understands and has done before, you may choose to only work with the 7 Product Dimensions or a portion of them.

Understand (at the story level)

Using a holistic process of getting a cross-section of perspectives together to build shared understanding and produce artifacts that let teams know what to code is a proven way to avoid re-work and waste due to stories getting rejected by the product owner or customer. Teams with strong shared understanding of what they're going to deliver enjoy shorter cycle time. Figure 14 shows a few of the techniques used to help get that shared understanding of user stories.

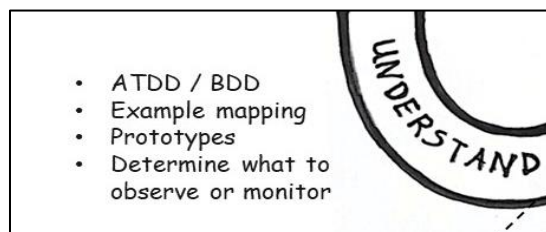


Figure 14: The Understand stage of the Holistic Testing Model

Power of Three

One practice that forms a great foundation for shared understanding is what we call Power of Three, or as George Dinwiddie dubbed it, Three Amigos. Before each story planning workshop, a product person, a programmer, and a tester may meet to go over the proposed stories. These days, with such complex systems, this session often needs more participants, maybe up to four to six – a designer, a data expert, a marketing specialist, an operations specialist – whomever has valuable insight to the stories being discussed.

Example mapping

Keep these discussions quick and focused with a framework such as [example mapping](#) by Matt Wynne (Figure 15). It is a great framework to elicit concrete examples of behavior to illustrate business rules for shared understanding across the team. The key is agreeing on the purpose of each story, its main value to customers, and specifying the business rules, with each rule

Holistic Testing

illustrated with concrete examples. The business rules and concrete examples that define how a capability should work can be turned into business-facing tests that guide development.

As shared understanding grows, the conversation may also uncover missing stories or ones that are too large.

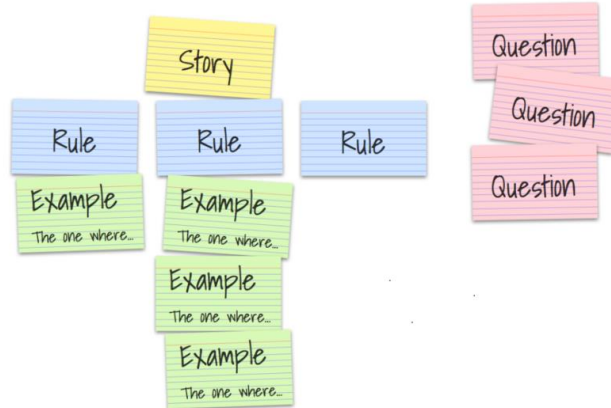


Figure 15: Example Mapping by Matt Wynne

Guiding development with examples

Figure 16 shows a simplified view of guiding development with examples. It starts with a conversation, and exploring examples before coding, testing, and automation happen.

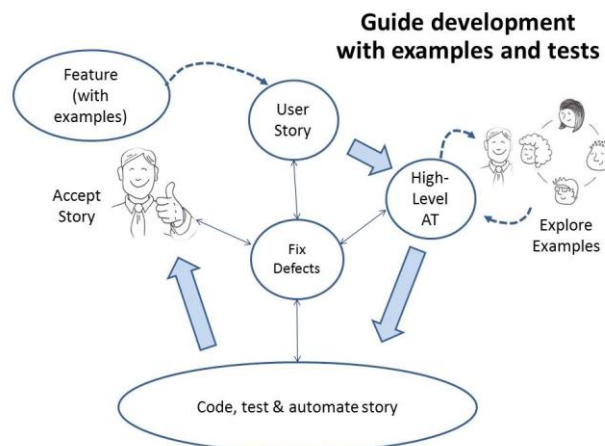


Figure 16: Simplified view of guiding development with examples

Working together, programmers, testers and product owners can turn the business rules and examples from workshops into executable tests, using a domain-specific language that people outside the technology team can also understand and review. These tests form the basis for acceptance-test or behavior-driven development, where the business-facing tests guide development. They help to build the right thing the first time we try.

Planning for continuous delivery

Teams also need to think about types of testing needed to help them feel confident to adopt continuous delivery. Lisa has written a blog post to help consider [what types of testing may apply](#) and how to plan by using the agile testing quadrants as a guide.

Looking further along the holistic testing loop, we can see that teams can anticipate putting stories into production sooner, allowing for much faster feedback. Shared understanding should extend beyond feature behavior to other concerns. For example, once we have built a feature, how do we get it into the hands of our customers?

Many agile teams are practicing continuous delivery (CD) or are working towards implementing it. The goal is to get changes and new features into production safely, quickly, and sustainably. The backbone of CD is the deployment pipeline: all the steps a change needs to go through from the time it is committed to the code repository trunk or master, to the time it is deployed to production. There are many questions we can ask about our deployment pipeline to make it better. Note that continuous delivery does not mean teams need to release to customers daily.

One question testers or business stakeholders might ask is, "When we need to fix a critical production problem, how long does it take from when the programmers check in the code for the fix, to the time it's actually in production?" This is a great time to start drawing on a whiteboard or lining up cards on a table with team members who understand the process. Does the 'hot fix' go through the same automated tests as a normal code change? What about exploratory testing? What pieces are automated and what requires human intervention? Do we skip some steps to save time?

Teams transform to a DevOps culture to help them implement CD. The mantra for this cultural shift is, "We build it, we run it." The planning stage is the time to start thinking about how the team will monitor and observe production usage when they release a new change to customers.

What data will be helpful on monitoring dashboards? What misbehaviors should trigger alerts for the team to investigate? If it's not practical to capture all production events to allow for observability, which should be included in a representational sample? Telemetry needs for log data, metrics, events, and traces should be included as appropriate in user stories. Dashboards, alerts, and usage data can be tested along with functionality and quality attributes.

Complex applications present extra planning challenges. For example, coordinate with an external partner to plan how dependencies between the two products will be tested. What contract tests are available for microservices. Find out what data will be available in other parts inside and outside the product to monitor and observe in production.

Holistic Testing

Build

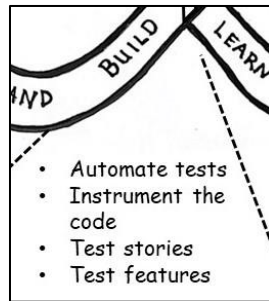


Figure 17: The Build stage of the Holistic Testing Model

Building is where the “magic” happens – at least from the viewpoint of people outside the delivery team. Once the delivery team shares understanding with business stakeholders of the problem to solve, it’s time to start testing and coding. Instrumenting the code and automating the technology- and business-facing tests are all part of the building stage. Figure 17 shows some typical activities during this stage.

A quote people are most likely hear Janet say is:

“Throw away the ‘then’ in ‘code then test’ – replace it with ‘and’. And maybe, also reverse the order to say, ‘test AND code’. Put the test first.”

This came about because teams don’t think about the term “code then test” as a problem. But every time someone says that phrase, it reinforces the misconception that testing comes after coding.

Testing and coding shouldn’t be separated as stand-alone activities. They are part of the same development process. Code is not complete without testing – at least some kind of testing, although some testing activities can be completed without writing any code. An example of this might be when we do an experiment or a simulation to test an idea and the team decides the idea isn’t valuable enough to continue.

Figure 18 shows how ATDD and BDD work during the building process. During planning, a team understands the story, by exploring examples and creating acceptance tests showing the intent and scope of the story. Once they have that, the team can get into more detail about building. Team members, often testers, expand those tests using their own set of tools and heuristics to think of boundary conditions, edge cases and more. They can then pair with programmers to discussion automation. For example, they can decide whether a given set of tests should be created at the unit, API, or workflow level. A [downloadable chapter](#) from *More Agile Testing* on models has a bit more on automation.

Each new automated test is run as the code is written. As each test passes, more code is written until all acceptance tests are passing. Then the team can perform other types of testing needed at the story level, which may be exploratory testing, quality attribute testing, and/or verifying the log data gathered via instrumentation. Once all story-level testing has been completed, the product owner can decide to accept the story.

Holistic Testing

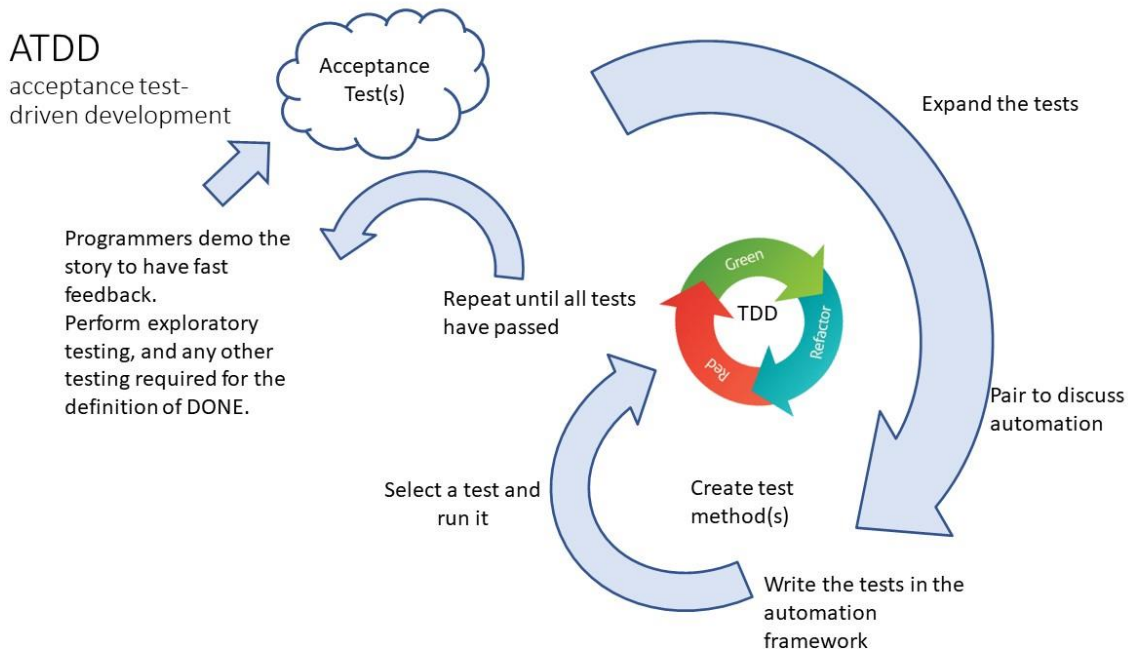


Figure 18: Acceptance test-driven development

TDD is about code design and building testability into the code. As new changes are added, programmers should take care to keep the code testable, maintainable, and operable. For example, remove unused code and any tests associated with it. High-quality code, supported by unit-level regression tests, is the foundation of a quality software product.

BDD and ATDD are about getting shared understanding about what we are going to build. These practices can all be considered testing activities since we are testing ideas and assumptions. And they're testing activities that the whole team can participate in to help make sure they build the right thing, in the right way.

When enough stories that make up a feature are completed, similar testing activities can happen at the feature level. More automated tests might be written at the feature level, such as end-to-end workflow tests. Testing will continue as the new changes make their way into the deploy stage.

Holistic Testing

Checking what you built using the right side of the Holistic Testing Model

When we consider the stages on the left side of the Holistic Testing Model, we focus on uncovering risks, building the right value for customers, and preventing defects in the code. Building testable, maintainable, operable code is the heart of any quality strategy. Experience tells us that unknowns await us, so plan and build for those as well. In the section about testing strategy on the left side, we pointed out that this is the time to decide what data you'll want to capture so that you can quickly identify and understand problems in production and learn how your customers use your application.

This creates a foundation for the part of the testing strategy that applies more to the right side of the Holistic Testing Model (Figure 19). Once there is an artifact to deploy to a test environment, teams can start learning whether they built the right thing, whether new changes behave as expected, whether expectations are met in all important quality attributes.

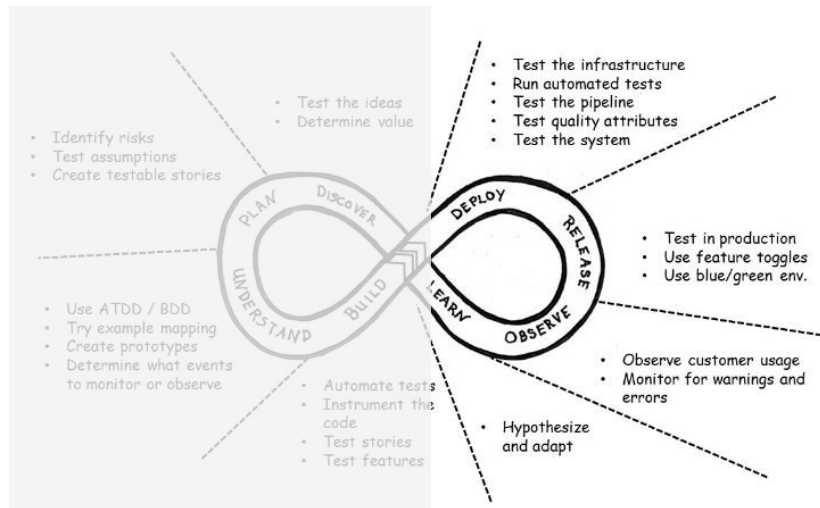


Figure 19: Right side of the holistic testing Model

The following sections for each stage explain techniques to help build the part of your testing strategy that gives your team confidence to release changes to production and learn how customers use those changes.

Deploy

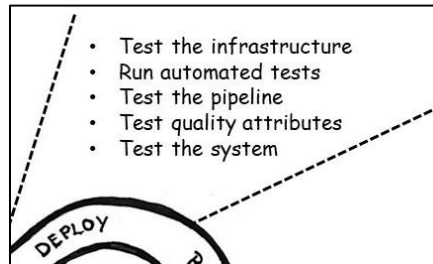


Figure 20: The Deploy stage of the Holistic Testing Model

Holistic Testing

Figure 20 shows the Deploy stage with some of its key activities. Testing activities here depend on good deployment pipelines. Deployment pipeline is a term that is usually reserved for a fully automated process. In this section, we use the term *workflow*, to mean what steps changes go through from the time a team member commits them, to the time they are put into production. The workflow includes “human-centric” steps that are not automated.

Understanding and improving the path to production

Visualizing your team’s workflow is a great way to find ways to improve it. Get together on a video call with a virtual whiteboard, or in person with a table and index cards or use a whiteboard. When a change is committed to the source code repository, what steps (human-centric or automated) happen to get that change into production? Use virtual or actual sticky notes to represent all the steps. In the simplified example in Figure 21, automated steps are blue, and human-centric ones are yellow.

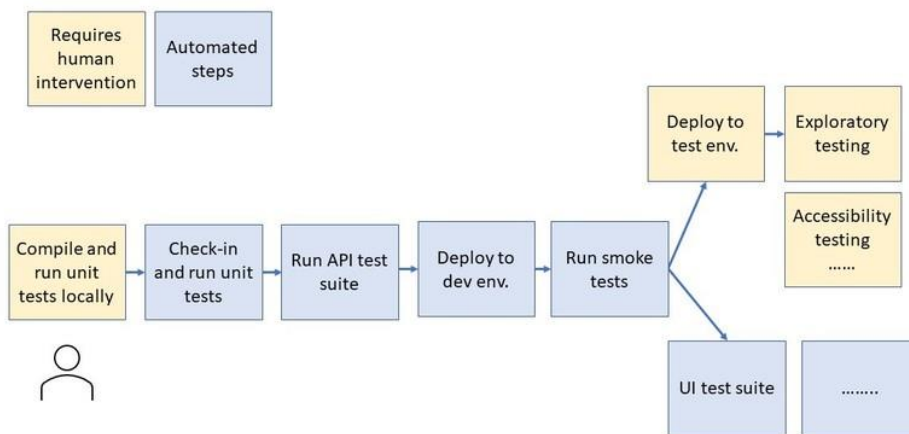


Figure 21: Simplified deployment workflow

Visualizing your workflow (or deployment pipeline) helps you learn about the normal steps each code change goes through in your team's pipeline. Ideally, the whole team or a cross-section of team members with different specialties draw the pipeline together. Even if that doesn't happen, it's valuable to do it for yourself. Ask a programmer or operations specialist on your team to walk you through the process.

Your workflow probably consists of both human-centric and automated steps. For example, when a code change is checked in, the first step might be static code analysis to see if the code meets team standards. The next step might be running automated unit tests. Human-centric steps might include exploratory testing. Deployments to test environments could be automated or require human intervention.

Fast feedback is important, so ask what your team is doing to speed up the automated regression tests and deployments to test environments. Can you do functional, performance and security testing in parallel? What can we do in a virtual environment? If so, what does that require? If your process includes user acceptance testing, do the users have their own environment and is

Holistic Testing

it done in a timely manner? Are there tests being done by testers or developers that are candidates for automation? You can help your team prioritize efforts to speed up those feedback loops.

Visualizing the workflow helps your team identify bottlenecks, opportunities to speed up the feedback cycle, stages that could be automated, and more.

Testing the workflow or deployment pipeline

Another important question is "How has the pipeline been tested?" Remember, in today's world, infrastructure is code. Automated portions of pipelines include configuration files and scripts that determine when and how one step triggers another step, how team members are notified of results, and how artifacts are stored or deployed. Like any software, we need to test to make sure it does what we want.

We may hear statements like "DevOps teams are in charge of pipelines," but don't forget that testing is an integral part of DevOps. Teams benefit when all roles, including testers, engage in improving pipelines to production. Just like our feature code, pipelines require analysis and testing to validate and optimize them. It's one more area where testers can work with other team members to mitigate risk. When a separate team is managing the infrastructure, build relationships and find ways to collaborate to provide effective feedback loops.

Some organization struggle with physical test environments. As you visualize your pipeline, consider how this slows the deployment process. What can you do to speed things up? Consider test data as well as the types of testing performed in each environment.

Automated test coverage

When tests are automated to speed up the deployment pipeline, consider the best way to split up the tests so they run effectively. Ashley Hunsberger has developed a Test Suite Canvas (Figure 22) which we have found to be extremely useful to see what different test suites are used for, their benefits, who takes responsibility for investigating failures and maintaining tests, and more. Ashley has written about how to use this canvas in this [article about feedback in deployment pipelines](#) (there is a larger picture in the article). In our experience, data is one of the most overlooked areas. Ask your team if they know how or where they are getting it from, and how is it managed?

Holistic Testing

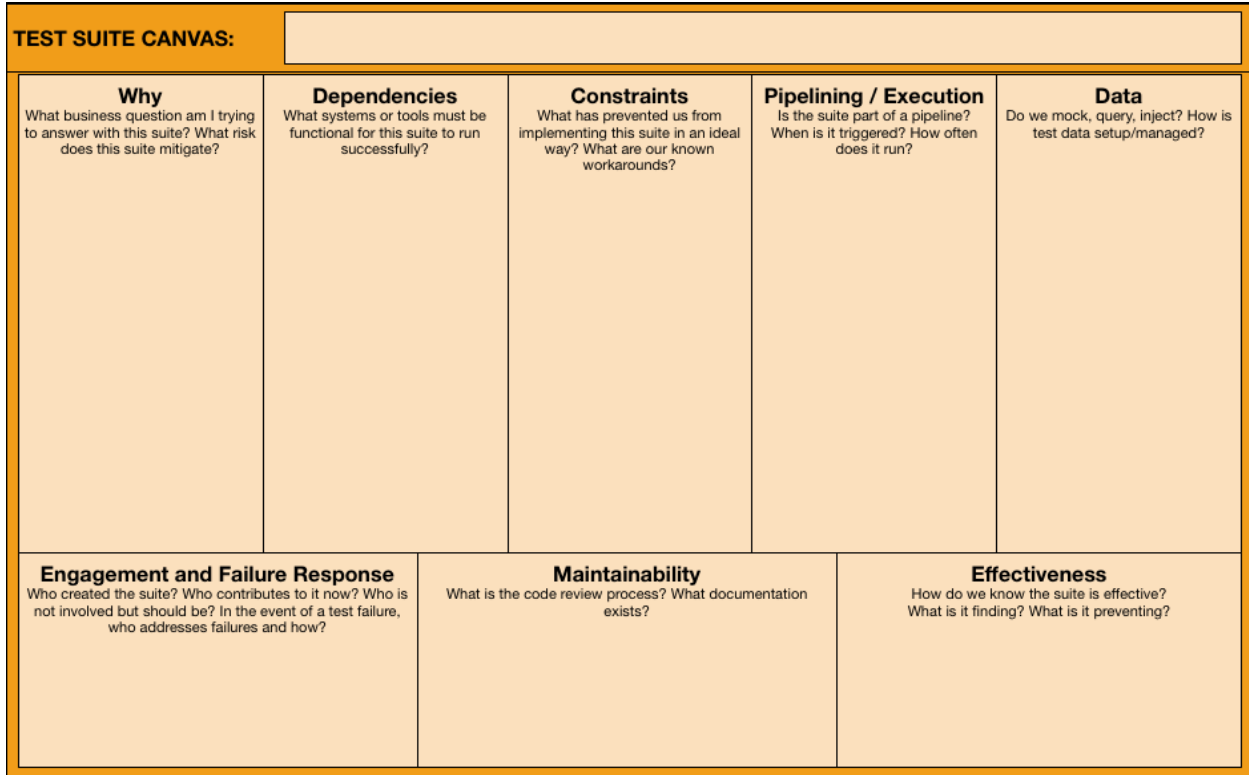


Figure 22: Test Suite Canvas (from Ashley Hunsberger)

Once you know what your automated test suites are, decide in which environment they should run. For example, unit tests should run locally on every developer’s local machine before check-in, and then run automatically on each build after each new commit. They likely won’t run again until the next check-in. A smoke test suite will run on the first deployment to a development environment to make sure that the most important features work correctly, and perhaps to see if someone can log into the system. The smoke test suite might run on any test environment with a new build or run regularly on a staging environment or even on production as a health check if it can run safely. Take care to ensure that each automated test uses reliable test data that isn’t changed by other testing.

Testing quality attributes

During planning, the team identifies risks and quality attributes that need to be tested. During development, programmers design the code for ease of testing. For example, they may put in hooks to help test loading time of a page. However, testing quality attributes often cannot be completed until after the code has been written and deployed to a test environment. For example, accessibility testing can have some automated components, but ultimately, it means a human being making sure they can access the information as intended.

Your team likely has (or will have, as you proceed along your automation journey) other automated test suites including performance, load, and security tests. Human-centric exploratory testing that supplements these automation efforts is often necessary.

Holistic Testing

Your team may also have determined that recoverability is important, so you'll have a playbook to try many different scenarios. Some scenarios may be automated, but many will need human interaction.

There are too many quality attributes to address them all here, and each team's context is different. The important thing is to remember to have a conversation early during planning to determine your constraints and determine what to test with every story and with every feature.

Release

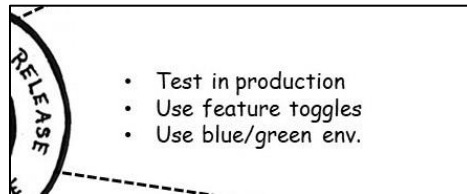


Figure 23: The Release stage of the Holistic Testing Model

Teams practicing continuous delivery or deployment use release strategies to allow deploying changes to production while hiding them from some or all customers. Examples of these activities are shown in Figure 23. Shared understanding of the release strategy is essential! Document it, have an up-to-date runbook, Train everyone, practice on pre-prod and test environments. Try different approaches to see what works.

Testing in production (TiP)

Testing in production is becoming more common, now that it can often be done safely using techniques such as [feature toggles](#), [canary releases](#), and [blue/green deploys](#). If this isn't an option (some business domains, for example, do not permit any of these techniques for security reasons), teams can closely monitor and observe production usage and use that information to guide development and testing.

Getting a diverse group of developers, operations specialists, testers, product people and other practitioners together to brainstorm about the various "what if" scenarios that could happen in production is one of the most important ways to identify risks. These conversations can lead to designing hypotheses to be tested in production.

Some companies practice chaos engineering (a type of TiP) which was pioneered by Netflix. [Wikipedia](#) describes it as the discipline of experimenting on a software system in production to build confidence in the system's capability to withstand turbulent and unexpected conditions. This type of testing in production requires a robust infrastructure and fail safes. Chaos engineering can also be conducted on staging environments, providing useful information with lower risk.

Testing activities still need to be done in test and staging environments, although not all production types of issues can be found in those environments. Testing safely in production, and learning from close study of production use, are key for the complicated, multi-service, distributed systems that so many organizations need today.

Holistic Testing

Observe and Learn

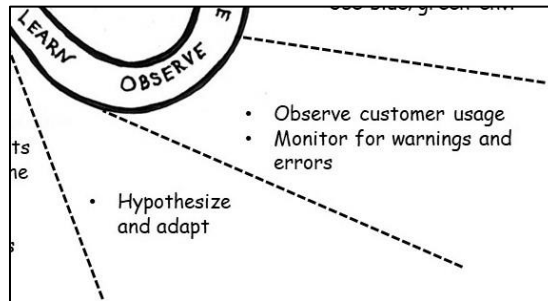


Figure 24: Observe and Learn stages of the Holistic Testing Model

Once a change is in production, we watch for any problems and see how they are being used. We take in all the information we need to address any problems that occur and plan future changes. Figure 24 shows some activities in both the Observe and Learn stages of the Holistic Testing Model.

Monitoring and observability

Monitoring and observability in production are two important ways to care for the product and your customers. Both monitoring and observability use telemetry (measurements for data collection) from application code to understand production system behavior. There are important distinctions between the two.

Monitoring is about looking for behavior that we expect. Teams collect log data, and then use monitoring tools to aggregate it, analyze it, and produce dashboards and alerts. They compare the actual data with what was expected. Monitoring is about predictable failures; it enables a team to see any deviation from expected behavior.

Observability is about the behavior that wasn't expected or couldn't be anticipated. Today's complex systems fail in complex ways. Stuff happens – and teams need to ask questions that they didn't expect to ask. One of the best ways to know if you have observability is to answer this question: "Do you have to add new instrumentation to the code and redeploy it to diagnose a problem?" Practicing observability means being able to diagnose a problem without that step.

We're borrowing a visual (Figure 25) from James Lyndsay with a slight nuance switch to show the difference.

Holistic Testing

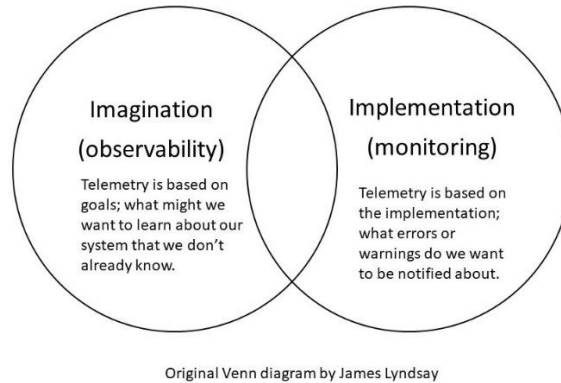


Figure 25: Difference between observability and monitoring

Note: The original Venn diagram is available as a download from [James Lyndsay's website](#). We encourage you to read it.

Teams have used monitoring tools for many years but find that their customers may still feel pain even as their monitoring dashboards show a healthy picture. Hopefully, all the important testing was completed before releasing a change, anticipating what might happen in production. Teams may even have done chaos engineering to explore different types of failures in a controlled way.

From experience, we know teams cannot think of everything, but try to be ready for anything. That is why as much data as possible is captured so that teams can use specially designed tools to analyze it quickly – to diagnose problems. Think of it as an early warning system. That is what we call observability.

Observability is one important tool among many to learn how your customers use your product(s), what pain points they have, and what features they might still need. When teams build software features, they need to build in the telemetry for monitoring, observability, and analytics. All of this enables them to respond quickly to production issues, no matter how many or how few users feel the impact.

Learn

Teams succeed when they work step-by-step towards their goals. Sometimes the team will run into problems and take a step back. Some experiments will fail. Keep the experiments small, so that a failure is good learning without high cost. Find good ways to measure progress in your context, to stay on track towards continuous delivery.

Small experiments are a safe way to try out new ideas without the risk of losing lots of time or wasting money. Work one small step at a time.

One of our go-to resources for testing activities on the right side of the Holistic Testing Model is Katrina Clokie's book [Practical Guide to Testing in DevOps](#). Lisa's website has a [list of additional books, videos, blog posts and more](#) to help testers and others learn about these "right-side" practices. Typically, a team's quality and testing strategy will be strengthened by having testing

Holistic Testing

specialists involved in activities such as configuring continuous integration and deployment pipelines, monitoring, and observing production usage, and managing releases.

Getting the whole team involved

Having a wide variety of specialists collaborate, means building a more effective testing strategy for the activities on the right side of the Holistic Testing Model. Engage developers, testers, database experts, operations specialists, product people and more. This diversity promotes innovative ideas for solving problems and improving the processes.

For example, if a new deployment pipeline tool would improve the reliability of deployments, testers can collaborate with platform engineers and developers to plan, implement, and test it. Marketing specialists may notice anomalies in user behavior via the production analytics tool. Testers are often the first to notice an odd pattern in production usage on the monitoring dashboard.

Assessing quality

There are many ways to measure the quality of our processes, but it's much more difficult to measure the quality of our products. This quote from our second book [More Agile Testing](#), describes how we think about process quality.

“When you work in a company whose leaders understand that focusing on quality is one of the best ways to deliver business value to customers frequently, your team – and company – are likely to succeed. Frequency and consistency of delivery improve over time. Conversely, if a company's focus is on speed first, often quality is sacrificed. Technical debt in the form of fragile, hard-to-change code and slower feedback cycles decreases the team's ability to deliver consistently.”

It is important to recognize the difference between product quality (quality of the product) and process quality (quality of the processes used to build the product). For example, people often want to measure defect counts to talk about product quality because it is simple to do. Defect counts may measure how bad your product is, rather than how good – especially those reported by the customer. For example, if the customer reports five minor defects. Does that mean the product quality is good? What if they reported five high priority defects? As you can see, it depends so be careful how you interpret specific measures.

Measuring product quality

There really is no easy way to measure product quality but we can try. For example, Isabel Evans showed this graph at a conference (Figure 26), showing two different products that had basically the same functional features. However, they were aimed at completely different users. One group did not care about speed and was afraid of making mistakes (not tech-savvy). The other group valued being able to complete tasks quickly and wanted flexibility in their product.

Holistic Testing

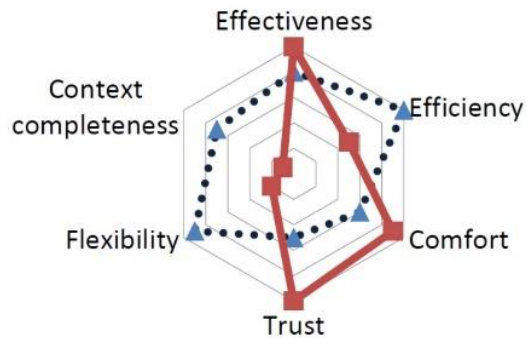


Figure 26: Radar graph comparing functional features

In this case, value was not a price point difference, but an attribute difference. These organizations were in the same product line but not really competitors since they had a completely different customer base. They based their quality measures on value and user-based metrics for individual customer personas. Both provide value to the customer they were targeting.

Organizations tend to use more qualitative measures for product quality like above based on surveys or feedback from customer support. Organizations can also measure things like customer loyalty – how often do customers come back, or how long to they stay customers. For example, if a company offers a product for free, but gets income from add-on services, customer retention is extremely important. However, in industries such as health plans and utilities, where consumers have a hard time switching because lack of any kind of quality, measuring loyalty is probably not a good measure of product quality.

Measuring quality practices

Examples of metrics that are used to gauge process quality are: number and severity of defects in production, deployment frequency, change failure rate, build radiator remains green, no flakey automated tests, static code analysis of codebase is healthy, rework rates are low, and number of defects that get reopened. None of these tell us if the product is valuable to the users, but they satisfy the human need for numbers. They are also easy (fairly easy) quantitative measures to capture but can lead us to incorrect conclusions about product quality.

Quality practices assessment

Selena Delesie and Janet Gregory have written a book [Assessing Quality Practices with QPAM](#) with the idea of assessing quality practices – thinking about the many aspects of quality within four dimensions (Beginning, Unifying, Practicing and Innovating). The three most important aspects of quality listed are:

- feedback loops

Holistic Testing

- culture
- learning and improvement

These aspects are the support for all other aspects and practices such as the development approach, quality and test ownership, testing breadth, the deployment pipeline, and defect management.

Figure 27 is an example using the quality and test ownership aspect. Think about a team who is at the beginning of their journey and perhaps living in a bit of chaos. Their testers may have been taught in the traditional manner that they are responsible for quality and are the gatekeepers. They own testing. On the other spectrum is a high performing team that is innovating. They feel the magic. They've been working together for several years and have learned to work together to build quality in from that first aspect of discover.

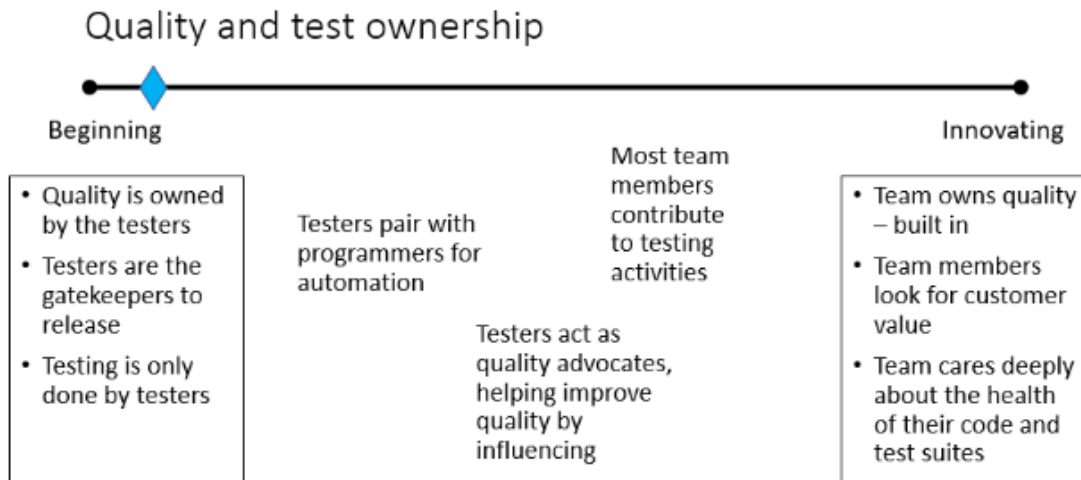


Figure 27: Quality and test ownership spectrum

Of course, there are lots of in-between steps to get from one end to the other, and it is not a linear journey. Some individual practices might race ahead, while others stagnate.

Conducting an assessment can be a valuable way for your team to discover where they sit on the quality spectrum, and where they might be able to improve.

Think about what you can do to influence your product quality. When you look at metrics, don't go for the easy ones immediately – keep asking the [hard questions](#) so you start to track the right information. Ask yourself "What problem are you trying to solve?" Start by setting achievable, short-term goals, and find ways to measure whether your improvement experiments are moving you towards those goals.

How leadership and culture impact quality

A healthy culture where people feel safe to experiment, ask questions and point out problems is a prerequisite to an organization's ability to deliver a high-quality product. Its leaders need to nurture that healthy culture.

Culture

In the section on assessing quality, we mentioned the importance of culture in how successful a team can be. An organization's culture sets the foundation for what it can achieve. Working in a space where people feel safe to experiment, explore, connect, create, and be human, enables creative and valuable outcomes. Alternatively, an organization that commands, controls, judges, reprimands, and considers people expendable significantly limits creative output and value delivery. A healthy culture can make a huge difference in organizational success.

There are many kinds of leaders; they include executive leadership, middle management, team leads and individual contributors. In this section, we concentrate on first two – executives and management.

Leadership and communication

Look at the information flow in your organization. Is it top down only – from executives to management to the delivery teams? Is there an easy way for information to flow upwards from the delivery teams and be heard. Think about how changes are determined. Are changes discussed by a few people around the water cooler or in hallways, or is the whole team involved and understand why a change is happening?

Often organizations try to create small improvements within one functional group, and often overlook the bigger issues and create silos within a delivery team. Cultivating trust and pushing decision making to those most in touch with the circumstances allows knowledge workers at all levels to make better decisions.

A model that can help leadership to understand what is happening in delivery teams is the [Cynefin](#) model (Figure 28) developed by Dave Snowden.

Holistic Testing

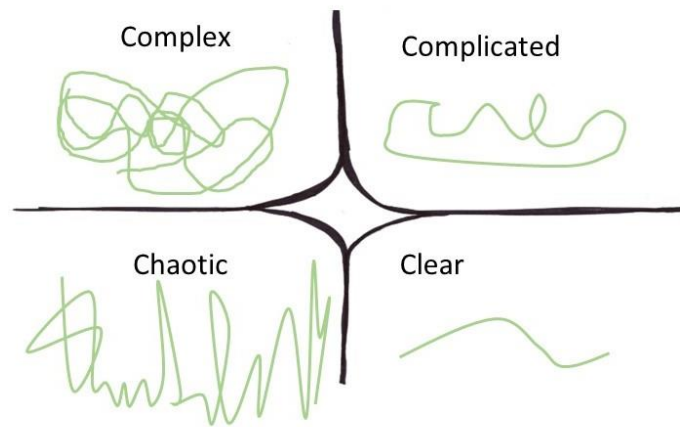


Figure 28: Cynefin model by Dave Snowden

Features are not created equally. Sometimes teams can't give a realistic estimate because the feature is something they have never done before and have no idea how to tackle it – it falls into the complex Cynefin quadrant. They may do experiments to break it down into smaller pieces to develop. Then there are complicated types of features which teams can understand and give reasonable estimates because they understand the work. The simple (clear/obvious) ones are features they know and can easily tell you how long they will take to build. The problem comes into play when leadership has the same expectations for all new work. The chaotic quadrant is about reacting, responding to fires – which teams don't want to be doing because it keeps them from doing their new work.

Leadership can help by creating a vision that is shared with the organization so individuals understand how they contribute to that vision or goal. Software development is an activity that's mired in uncertainty. When organizations communicate freely, even when we are uncertain of the overall situation, people can continue to work effectively.

Support for quality

Not all products are the same and not all delivery teams are the same. Products do not all require the same level of quality or have the same risks. As an organization, you can identify the level of quality required and the risk profile on a large scale, but teams need to understand what it means to them, and their product. They need to understand how much testing might need to be done.

Leaders can help to create that understanding of what is important to the organization and to the different market segments. They can help teams ask better questions to expose the risks and mitigate them. Having the conversation about quality can be difficult, but it is important – at the organization level, at the product / program level, at the team level.

The Leadership team in an organization is the one who sets the ground rules for a culture. They nourish the environment in which teams can grow and improve. For example, creating a no-blame culture so that individuals and teams are free to speak up or try new things.

Holistic Testing

Leaders - create a culture of quality by enabling the teams to take training when needed, and time to learn, and time to build quality into a product. Listen to the teams to understand how they work, and support that fully when it follows the overall vision of the organization. Trust the teams to do what they do well.

Summary

To develop a quality and test strategy, start with a conversation about quality. Only then, can you follow-up with identifying the testing activities that support the level of quality wanted and needed by the organization.

In collaborative, high performing teams, everyone must be able to use the changing customer expectations as the foundation for their work. It is not enough to make sure the product “works as designed”. It needs to provide the value expected by customers and other stakeholders.

Your team can use the Holistic Testing Model to craft a testing strategy that supports the level of quality you and your organization want. It makes the collaboration needed – across roles, across teams – across the organization, highly visible.

When testing, consider all types of testing, not only the ones a tester might be responsible for. Include automation, exploratory testing, or any other type of human-centric testing. Involve the whole team, the product organization, and even the customer to think about quality and testing from a holistic point of view. The Holistic Testing Model helps people understand when the different types of testing might take place.

The highest performing teams and organizations that we have worked with, have taken a holistic point of view to the quality of their product, and understand how testing can support that vision.

Resources for further learning

Our Holistic Testing Courses

- “Holistic Testing: Strategies for Agile Teams” and “Holistic Testing for Continuous Delivery: for a quality DevOps culture” training courses from Agile Testing Fellowship, developed by Janet Gregory and Lisa Crispin: <https://agiletestingfellow.com/training>

Our YouTube channel and websites

- Donkeys & Dragons 15-minute video chats with Janet and Lisa, plus special guests, covering a wide range of topics related to holistic testing, continuous delivery and more: <https://www.youtube.com/@AgileTestingFellowship>
- Our blog posts about holistic testing, testing in DevOps, continuous delivery, leading agile practices and more: <https://agiletestingfellow.com/blog>
- Website for our books and video course, with blog posts and free downloadable content and resources: <https://agiletester.ca>
- Janet’s website and blog: <https://janetgregory.ca>
- Lisa’s website and blog: <https://lisacrispin.com>

Resources referenced in this mini-book

- Jerry Weinberg: https://en.wikipedia.org/wiki/Gerald_Weinberg
- W. Edwards Deming: https://en.wikipedia.org/wiki/W._Edwards_Deming
- “Adapting Crosby’s 4 absolutes of quality into a software context”, Dan Ashby: <https://danashby.co.uk/2019/09/30/adapting-crosbys-4-absolutes-of-quality-into-a-software-context/>
- “What Does “Product Quality” Really Mean”, David Garvin: <https://sloanreview.mit.edu/article/what-does-product-quality-really-mean/>
- “Continuous Testing in DevOps”, Dan Ashby: <https://danashby.co.uk/2016/10/19/continuous-testing-in-devops/>
- Impact Mapping, Gojko Adzic: <https://www.impactmapping.org/>
- “Quick Risk Strategizing”, Lisa Crispin: <https://lisacrispin.com/2022/03/20/quick-risk-strategizing/>
- “Good Services, with Lou Downe”, Making Tech Better podcast: <https://www.madetech.com/podcast/episode-22-lou-downe/>
- 7 Product Dimensions, EBG Consulting, Ellen Gottesdiener and Mary Gorman: <https://www.discovertodelivery.com/image/data/Resources/visuals/DtoD-7-Product-Dimensions.pdf>
- “Introducing Example Mapping”, Matt Wynne: <https://cucumber.io/blog/bdd/example-mapping-introduction/>
- “Applying the Agile Testing Quadrants to Continuous Delivery and DevOps Culture”, Janet Gregory and Lisa Crispin: <https://agiletester.ca/applying-the-agile-testing-quadrants-to-continuous-delivery-and-devops-culture-part-1-working-towards-continuous-delivery/>

Holistic Testing

- “Using Models to Help Plan”, Chapter 8 from *More Agile Testing: Learning Journeys for the Whole Team*: https://agiletester.ca/wp-content/uploads/sites/26/2014/09/Gregory_Chapter_8_Final.pdf
- “The Whole Team Approach: Optimizing Delivery Pipelines via Feedback Loops”, Ashley Hunsberger: <https://www.bizops.com/blog/the-whole-team-approach-optimizing-delivery-pipelines-via-feedback-loops>
- “Feature toggles, AKA feature flags”, Pete Hodgson, <https://martinfowler.com/articles/feature-toggles.html>
- “CanaryRelease”, Danilo Sato, <https://martinfowler.com/bliki/CanaryRelease.html>
- “BlueGreenDeployment”, Martin Fowler, <https://martinfowler.com/bliki/BlueGreenDeployment.html>
- “Chaos Engineering”, Wikipedia, https://en.wikipedia.org/wiki/Chaos_engineering
- “Why Exploration Has a Place in Any Strategy”, James Lyndsay, <https://www.workroom-productions.com/why-exploration-has-a-place-in-any-strategy/>
- *A Practical Guide to Testing in DevOps*, Katrina Clokie: <https://leanpub.com/testingindevops>
- “Observability, Continuous Delivery, DevOps & Related Resources”, <https://lisacrispin.com/observability-continuous-delivery-devops-related-resources/>
- *Assessing Agile Quality Practices with QPAM*, Janet Gregory and Selena Delesie, <https://leanpub.com/qualityassessmentpracticesmodelqpam>
- “Why Our Minds Swap Out Hard Questions for Easy Ones”, MITSloan Management Review, <https://sloanreview.mit.edu/article/why-our-minds-swap-out-hard-questions-for-easy-ones/>
- “Cynefin for Developers”, Liz Keogh, <https://lizkeogh.com/cynefin-for-developers/>