Nimble, not Agile: Creating a flexible quality assurance framework for your company's unique SDLC

Zenzi Ali

zali@clarity-innovations.com

Abstract

Many software testing lifecycle models are designed to correlate with the most popular software development lifecycle models, like Scrum or Kanban. But, there are many ways to build and develop software outside of these standard models. How can teams create a workflow that is conducive to delivering high-quality software in an unorthodox environment? My team uses Basecamp's Shape Up method to deliver software. It provides a great deal of flexibility for developers to ship products, but it can be challenging to inject thorough test procedures into a method that centers developer productivity above team process. I have spent time dissecting the most popular testing methodologies in order to craft my own flexible framework.

Bio

Agility. Advocacy. Efficiency. Affinity. Amazing things happen when engineering and humanity come together. Zenzi understands the power of high-quality engineering products and their significant role in driving the modern world. But she also understands the need for clear communication when developing and implementing this technology so that it can better serve human needs.

Before joining Clarity, Zenzi managed a community of new technologists (providing code and assignment reviews); created, optimized, and automated unit tests; sharpened her communication skills while working as an Executive Assistant; used Agile methodologies to plan, iterate, and execute tasks within cross-functional teams; and implemented testing processes with startups and large organizations alike. She even contributed to The Code for Philly Front End Fellowship, supporting technology to help Philadelphia eliminate food waste.

Zenzi merges her troubleshooting experience and test craftsmanship with her passion for refining processes and collaboration, to ensure that every edtech solution Clarity creates is as streamlined, secure, and useable as possible. It's all part of her desire to help solve human challenges—such as those facing education—through the latest, most efficient means.

Section 1 Introduction

When I joined Clarity Innovations in 2021, I knew that the team was extraordinary and that I could thrive in the environment. I have a tendency to over-analyze and over-prepare and this team was action-oriented and with the experience necessary to make adjustments on the fly. I felt that their practical method of delivering software was the perfect pairing for the analytical way that I have thought about quality and delivery.

The method that my team uses for building software is called Shape Up. This development model, created by BaseCamp (a project management platform), is fast-paced and solution-oriented. Although the method focuses on strategic scoping and thoughtful problem-solving, there is little room or expectation for testing involved. Because of this most of our testing was built around product validation for our customers and did not have the depth of other testing models. There was no expectation the testing would start early or even before a product was ready to deploy.

At the outset, Shape Up does not seem to leave space for quality. QA is shown to be nearly unnecessary in their process and used to explore edge cases only. Box checking and rubberstamping a finished product is not satisfying testing work, in my opinion. It is my belief that quality can be injected into any process. I also do not believe that Shape Up is alone in its view of the tester. Many testing professionals find themselves feeling like second-class citizens on a team and wondering exactly where they fit in and how to advocate for quality.

Luckily, my team is devoted to building high-quality software and has an interest in more thorough testing and exploring techniques. This left me with a large task. How can I inject quality and testing methodology into my engineering team’s well-oiled process? How can I add value and assess risk without becoming a bottleneck to a high-functioning team? When and where can I shift testing left while keeping our business objectives and procedures as the primary focus? How can I create a process that can meet the varied needs of our business, which both creates and maintains a variety of technical products?

These questions and discovering my own pain points with our procedures caused me to do my own research and develop methods to aid in upholding quality throughout our development process. After reading *Shape Up* and observing my team in action, I developed my own techniques for adjusting my testing lifecycle to the project and product being developed. I will share my journey and methodology here.

The goal of this writing is to provide testers with the tools that they need to think deeply about testing and create a maintainable, effective testing routine. I in no way seek to malign or discredit *Shape Up*, Agile, Scrum, or any other software development lifecycle method. In my opinion, the best framework for creating quality is the one that you use repeatedly and consistently! Removing barriers to your quality practice is equally important to adhering to any framework.

This method is perfect for teams that are like mine:

- Solo testers or testers that are part of a small team.
- Testers that work with teams that use an unorthodox software development lifecycle model.
- Teams that build and ship fast.
- Quickly expanding teams that have not had time to focus on Quality Control and quality management.
- Testers that value flexibility in their tooling.
- Teams that manage multiple products with varied contextual needs.

Context often considers project needs, goals, and constraints, but my model also considers organizational "appetite" [more on this later] and internal standards for quality. My method of creating a test strategy allows me to fulfill my intrinsic need to amplify quality within any product I touch while respecting my team's culture and norms. Business and development needs are, of course, of high importance when one is the sole advocate for quality. My method focuses on prioritizing business needs while fulfilling my personal need for self-development and professional fulfillment.

In the rest of the document, I will provide insights gained by using my experience in managing my internal test procedure. I will explain how Shape Up works, the challenges the method produces, and how my team chooses to meet those challenges. I will outline how I align my test activities with Shape Up's software development lifecycle. I will also outline some ideas on how to advocate for the team buying in your new testing procedures. Moreover, I will include some visual aids that review phases of software testing and how they relate to software development. Finally, I will review scenarios of different types of teams and address how I might approach creating a maintainable, complete, and flexible process within those structures.

Section 2 Analyse the Environment and Create Goals

A plan of action was not drafted until my team's processes were analyzed and understood. My company is relatively small, but engineering makes up nearly 1/3 of our employees. We are an ed-tech firm that performs consulting work for K through 12 educators. Although our area of expertise is specific, our products and projects range in type and scope. Some products arrive "complete" and are in production but in need of updating or scaling. Others are developed in-house for our clients from inception to completion. We use the technology that suits their needs and user base. This can range from Flutter mobile development to Drupal web applications or any tech stack that the client has used previously. This means that our team must be highly adaptable and prepared to support a wide range of projects that we do not necessarily own.

Testing falls within the domain of our development team. Our testing team is small; a team of not quite two. I am the sole full-time tester with our senior quality assurance professional of the team dividing time equally between testing and DevOps. Our project work can start anywhere within the software development lifecycle but, no matter where we start or what the status of the project may be, we use Basecamp's product development model, Shape Up. Shape Up is a development model that is not well known but has worked for Basecamp for over a decade.

Basecamp's approach features a six-week build cycle that is shaped by senior project owners. There are no sprints, kanbans, or backlogs. The goal of this method is to speed development by keeping requirements relatively vague and focusing on solutions to a rigidly scoped problem. When it comes to Shape Up, the key is proper scoping and an understanding of the problem you would like to solve. This helps engineers tame complexity for faster development and separate concerns for better strategic planning. "Projects are defined at the right level of abstraction: concrete enough that the teams know what to do, yet abstract enough that they have room to work out the interesting details themselves." (Friedman 18)

The Shape Up model operates in three distinct phases: Shaping, Betting, and Building. During **shaping** a product owner focuses on how to deliver a solution for a whole product or feature set. The designs and ideas are rough, but the goal is to solve a problem. In this phase, shapers and trackers cooperate to create shaping documents for a pitch (a meeting where shaper(s) advocate for their project). The goal is to define the scope of this project, which should last no longer than its six-week cycle. This phase should include extensive discussions involving the project goals (what need or problem will this project address), possible rabbit holes, clear parameters, and project constraints. The project shaping document provides a development team with enough information to create without constraint. Setting the project parameters and defining "done", also known as breadboarding, is accomplished by deciding on the size of stakeholder appetite, and how much time, energy, and resources are we willing to invest to solve this problem.

Key staff must decide if a sufficient solution, even a rough one, is created within the time outlined. This process is done in isolation, or with only staff with the necessary expertise for scoping and shaping.

After shaping, the **betting** begins. Betting is the stage where shapers share a pitch with the larger group–placing their "cards on the table". They discuss the details of what they have shaped and the team "bets" on which project they will be working on for this time. If a shaper's project is not chosen, there is no backlog. The theory is that an important idea will show up again in the future. If a project needs refinement or if a solution is inadequate, the shaper can pitch at a later time. Betting is also when additional context to the project is added. Is this an existing project or a new one? Is research and development needed? Is this product already in production? Has it reached a maintenance phase or is it nearing retirement? All of these questions serve to add context to designing a project and betting on the most impactful and necessary project for this round of development.

Finally, **building**, the final phase, occurs. The building phase starts with a kick-off meeting between stakeholders and the development team, as well as any other required team members. At that time, stakeholders hand the project to our development team with the goal of developing slices of complete work that can be integrated into a larger system if necessary. Developers are expected to be assigned projects, not tasks. There are no Kanban boards or Issue tickets to assign. Developers should have a complete scenario or issue that they are programming a solution for and they should be programming just enough to complete the next building step.

During this stage, there is also scope mapping to confirm that the imagined tasks can be replaced or updated with discovered tasks. The team can continue to evaluate if their appetites match the size of the effort needed for the solution. Sunk costs are examined at every turn. Developers are expected to show progress, not discuss the process. Once our basic solution has been created, the developer stops and moves on. There is no list of future To-Do’s. We have solved our issue and now we provide feedback to the team at the end of the 6 weeks. Is there an additional appetite for a more complex or elegant solution? This can be decided at that time.

After observing my company's software development lifecycle and engaging in software testing, I began to think strategically about how we test, what can be improved, and how I can affect software quality without substantive changes to my team's SDLC, which they were happy with. I also felt that the SDLC was outside of my domain as a tester. Whether my team uses an iterative model or an incremental one, my goal is to be adaptable in my process. One tool that I used to assess our quality management ecosystem was a SWOT analysis. A SWOT analysis is an exercise that helps organizations (or individuals) identify their Strengths, Weaknesses, Opportunities, and Threats. It is essentially a risk assessment exercise. SWOT helps the user differentiate what environmental elements they can control and how to allocate team resources. I time-boxed one hour to review our procedures and another to consider what should be placed in each SWOT section. The analysis matrix looks like this:

Figure 1*: SWOT Analysis of My Team’s Approach to Development and Quality:*

SWOT ANALYSIS

S

STRENGTHS

- *Team open to new testing procedures*
- *Team values testing expertise*
- *Project variety allows for learning new test techniques*
- *Shape-Up features rigid cycles-"done" is defined*
- *Contractual obligations create well-defined UAT parameters*

W

WEAKNESSES

- *Shape Up risk assessment prioritizes “scope bloat” above other quality-related risks (security, performance, UX, etc.)*
- *User feedback is rarely received/explored in this model*
- *Fat Marker sketches are purposely abstract - no detailed wireframes for UI visual testing*

O

OPPORTUNITIES

- *To encourage planning and shift left approach*
- *Model static testing techniques*
- *Kick-off meetings, Shaping documents, and retros can provide opportunities to ask deeper questions for better testing*
- *Create an oasis of quality–lead by example*
- *Ability to show my work (QA is more than processing test cases)*

T

THREATS

- *Restrictions on time spent testing based on client budget, length of the contract, hours allotted, etc.*
- *Shaping is a closed-door, unscheduled process often done alone.*
- *Developer's understanding of QA processes are limited*

Section 3 Reviewing the SDLC and Common Test Activities

The software development life cycle is a methodical approach to planning, building, and shipping software. While the life cycles that are well-known tend to focus on these things independent of test techniques, it is our job to create quality practices that correspond to each phase of the cycle. The most common types of models are sequential, incremental, and iterative. See the key below with key aspects of each type of SDLC. Using an established model increases the likelihood of a successful project by establishing phases, activities, and opportunities for collaboration across job functions.

Our 3 most common SDLCs have corresponding software testing lifecycles (STLCs). The software testing life cycle features test activities and test levels that correspond to the development phases of its development model. For example, the traditional sequential model begins with documentation and planning. One may begin by testing using static testing techniques, reviewing documents for understanding, potential contradiction, or the need for clarification of features/functionality. Shape Up is not a standard technique or life cycle model. Because of this, they have their own approach and goals for QA.

For many years, Basecamp had no QA personnel. Eventually, they decided to hire a single QA professional for their development team of 12. With the developer-to-QA ratio being so high, the team must think differently about quality and who is responsible for it. Traditionally at Basecamp, quality falls on the design team and the developers.

Risk assessment and analysis is the domain of Product Owners and management. In the Basecamp context, QA is done at the end of the build to consider edge cases and approach the project with a tester's mindset. QA is considered valuable and essential, but siloed and separate from development. This approach has been successful for Basecamp but, through trial and error, my team has decided to tailor their approach to allow for more thorough QA testing in each iteration. In section 4, I will discuss how things stood at the beginning of my SWOT analysis (see Figure 1) and the changes I have made.

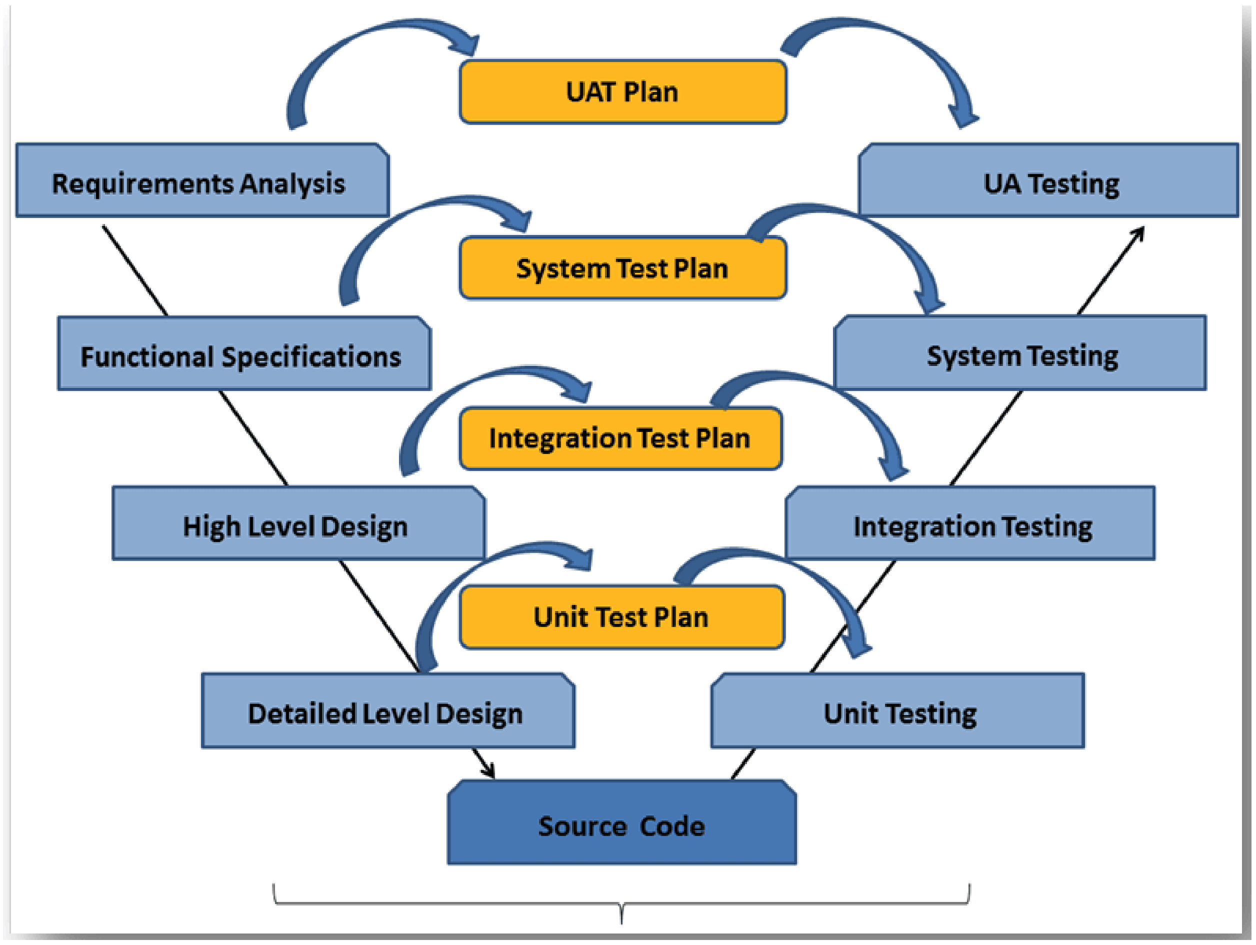
Figure 2: *Comparison of SDLC Models:*

	Sequential (ex: Waterfall, V)	**Incremental (ex: Unified Process)**	**Iterative (ex: Spiral, Prototyping)**	**Shape Up**
Model Attributes	Sequential approach. Begins at the system level and progresses through analysis,	Used to build large, complex systems. Based on the idea of adding new features, or “increments,” to	Initial development work is carried out based on well-stated basic requirements, and successive	Solution-centered approach focused on building a working product or feature to completion by the end of a cycle

	design, coding, **testing**, and support.	an existing system instead of a sequential build	enhancements are added to this base piece of software through iterations until complete.	
Testing Scope	Full system considered	Testing can be performed at each level and increment.	During early iterations, QA focus is on unit and integration testing, while later on, system testing and user acceptance testing.	Little to no QA expected
Test Timeline	Static testing at the beginning, dynamic at the end	Testing at the end of each increment	Testing at the end of each iteration	Must happen within the cycle/scope
Documentation Level	Heavy documentation, documentation is often the basis for testing	Light documentation	System requirements are heavily documented, but features and approaches may evolve as the iterations continue	Documentation after the cycle and often handled by other teams, Siloed collaboration means few details are captured in writing, no backlog means no todo list and light issue tracking
Test Approach	1. Examine requirements 2. Review Designs 3. Integration testing 4. System Testing 5. Acceptance Testing	Regression and integration-focused testing, fast feedback due to shortened development cycles, focus on experience-based testing techniques	Regression-heavy testing	Just edge case testing for QA, focuses on product use and approaching the project with a fresh eye
Automation	Not practical as the entire system should be built for pre-release testing	Automation preferred for faster regression testing	Automation preferred for faster regression testing	NA–not practical for this test approach

Tester Responsibilities	Requirement reviews, test analysis, test design, system maintenance	Testing incrementally developed features, providing prompt feedback to developers	Collaborate on exit criteria, test planning, bug tracking, test execution, data, and environment prep	Edge case testing, designers and developers are responsible for quality. QA can generate UX tasks that can be added to this cycle or pitched in a later cycle

Figure 3: *Sequential SDLC with Corresponding Test Activities:*



Source: *https://qainsights.com/improving-the-test-process/*

Section 4 Modularize the Test Cycle and Create a Framework

Prior to my team’s SWOT analysis, our testing relied heavily on a checklist approach. The key to finding defects effectively in that approach is having long-term experience with the product. Experience-based testing can be highly subjective. When testing, I found myself reaching out to developers often to confirm functionality because it did not meet my personal expectations. With no test oracle or test management software, communication of this sort happened throughout the testing process. This worked for my team when everyone was based in Portland but, since the pandemic, we have become a remote team with members in every US time zone. We needed test techniques that improved our communication and made the most of what documentation we chose to maintain in the future.

Although Shape Up does not prioritize QA, it does realize the position's importance. I did find the Shape Up method’s testing process to feel isolated and a bit tacked on to development. My remote East Coast position heightened this feeling. I wanted to improve the testing process and feel more integral to the product delivery experience. This required me to decide on my testing principles and discover how I could use those to meet testing and business needs. This can be done via brainstorming, mind mapping, or an additional SWOT analysis.

I decided to brainstorm current blockers and perform a solo retro to assess process failures. This was challenging in a team that does not center metrics. How could I assess my value and decide if incremental changes that I made in the process were impactful? I decided to gather any data that I had access to and that could be compiled quickly. Test completion estimates are very important to my team, so I could not increase the amount of time spent testing by any large amount.

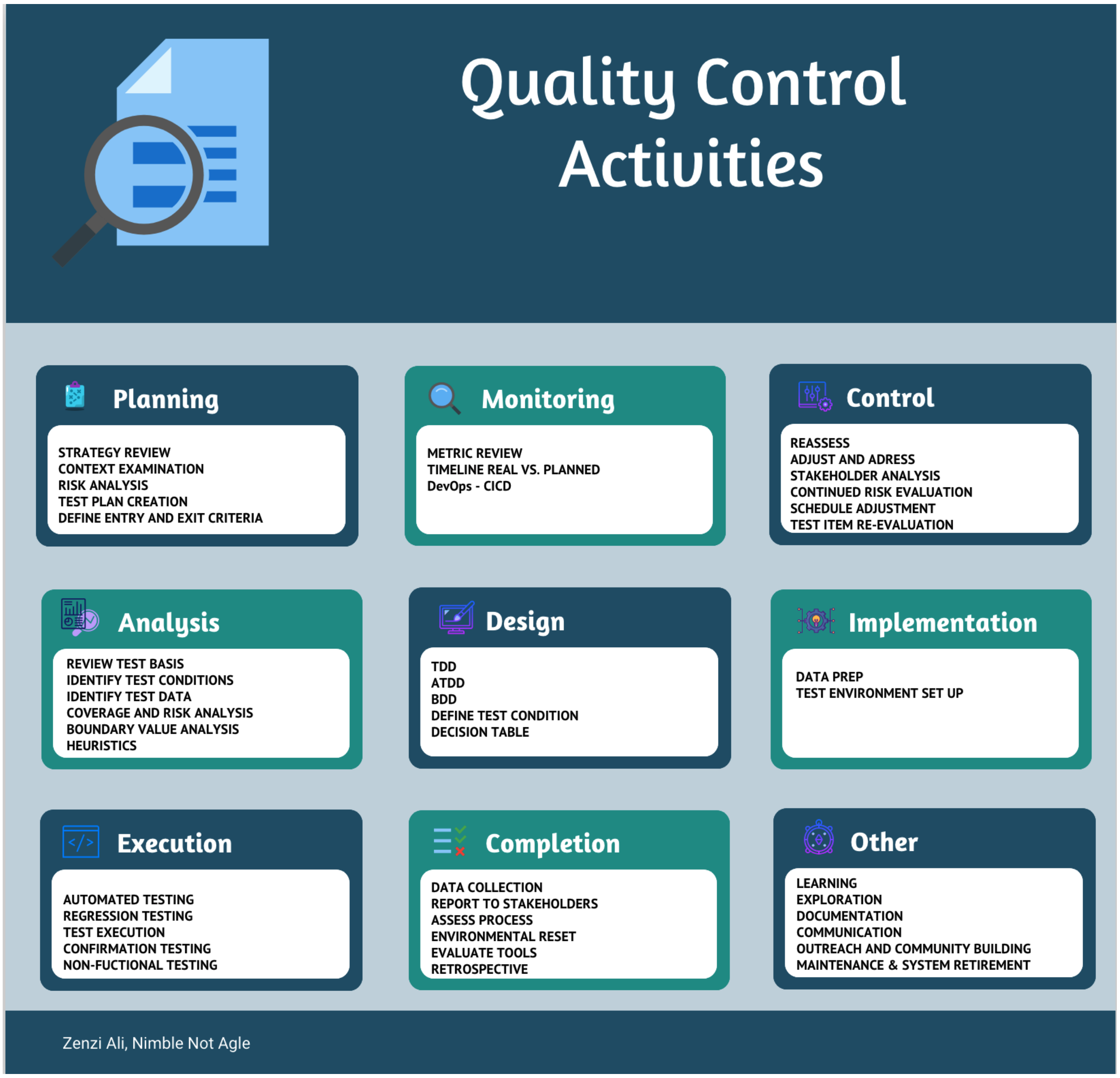
My professional values centered on product ownership, communication, and remote-friendly processes. I wanted to incorporate my professional values into the framework to garner a more personal sense of accomplishment from my work. Because I work in consulting, I’m a few steps removed from product ownership. Owning the process increases the satisfaction that I feel in my work. I also decided to use testing activities as a way to communicate with stakeholders and report early and often to my team without affecting their process. I used the tools that were available to me to complete testing that was not being performed in the past.

After my initial SWOT, we made a series of small process changes that improved testing needs:

- Transition from text document test cases to Test Management Software
- Remove testing discussions from Slack (where they could get lost) to their proper merge request or to a document in a shared drive
- Implemented automation to free QA time from regression testing
- Attending product meetings to observe (understanding context)
- Create Test Plans for new projects
- Re-organize tests to shift to a user story-focused test strategy
- Ensemble testing sessions and test case reviews with Developers

Each activity matches a Weakness or Threat discovered during the SWOT Analysis and/or strategically uses a team Strength to promote quality.

Figure 4: Test Activities by Development Phase:



Quality Control Activities

Planning

STRATEGY REVIEW
CONTEXT EXAMINATION
RISK ANALYSIS
TEST PLAN CREATION
DEFINE ENTRY AND EXIT CRITERIA

Monitoring

METRIC REVIEW
TIMELINE REAL VS. PLANNED
DevOps - CICD

Control

REASSESS
ADJUST AND ADRESS
STAKEHOLDER ANALYSIS
CONTINUED RISK EVALUATION
SCHEDULE ADJUSTMENT
TEST ITEM RE-EVALUATION

Analysis

REVIEW TEST BASIS
IDENTIFY TEST CONDITIONS
IDENTIFY TEST DATA
COVERAGE AND RISK ANALYSIS
BOUNDARY VALUE ANALYSIS
HEURISTICS

Design

TDD
ATDD
BDD
DEFINE TEST CONDITION
DECISION TABLE

Implementation

DATA PREP
TEST ENVIRONMENT SET UP

Execution

AUTOMATED TESTING
REGRESSION TESTING
TEST EXECUTION
CONFIRMATION TESTING
NON-FUCTIONAL TESTING

Completion

DATA COLLECTION
REPORT TO STAKEHOLDERS
ASSESS PROCESS
ENVIRONMENTAL RESET
EVALUATE TOOLS
RETROSPECTIVE

Other

LEARNING
EXPLORATION
DOCUMENTATION
COMMUNICATION
OUTREACH AND COMMUNITY BUILDING
MAINTENANCE & SYSTEM RETIREMENT

Zenzi Ali, Nimble Not Agle

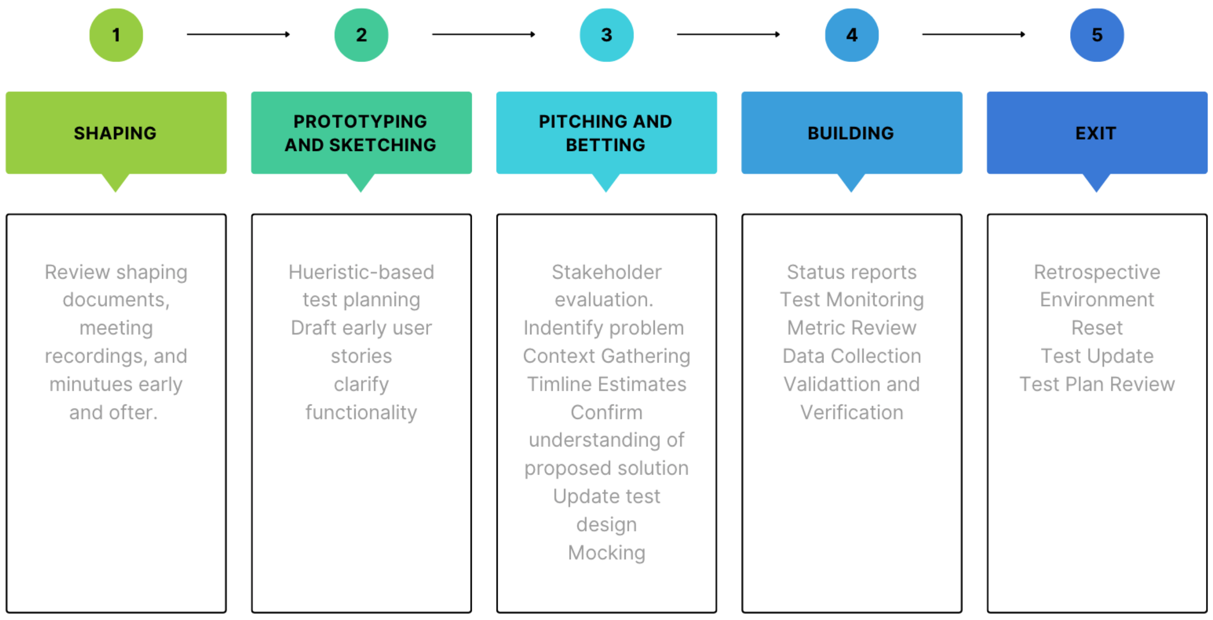
Figure 5 shows the most common testing strategy that I use. I was able to shift testing left because my team does not use the pitching and betting phases as designed by Basecamp. We are consultants, so our clients “pitch” a problem, our client lead performs betting activities, and then the client lead will Shape with a lead engineer.

I have chosen to display the phases linearly, but often the first 4 phases occur concurrently. This is beneficial to my testing process. I can start testing requirements via the documents because I do not need to wait for my team to bet on a project. Another benefit is most of my static testing occurs with no interruption to my team’s process. Shaping and design products are drafted as a part of our business processes. I can use these underutilized assets to my advantage without any extra time commitment from the team.

One area that is not given an official phase is completion. Basecamp completes its projects on time with little ceremony. Feedback is left to review for later pitches if significant negative feedback or product bugs are reported. For my team, there is a formal retro that allows me to solicit feedback from the team.

This is our general process, but it also varies by project. Choosing techniques and activities is easily done by using the ideas in Figure 4 as a starting point. Time is of the essence, so I like to time box creating a one-page test plan. Continuously adding techniques to my toolbox and discussing testing and product development with my co-workers and testing communities keeps my process fluid and nimble for changes that may occur during the building process.

Figure 5: Personalized Shape Up SDLC-STLC diagram



I had many opportunities to propose ideas and discuss changes that I would like to see on our team. I often used weekly one-on-one meetings to discuss ideas for professional development, a not-too-often discussed part of a good test strategy. Because of my team's unorthodox approach to testing, I did not have a basis for common test strategies and spent much of my first year reading books like *Explore It! Reduce Risk and Increase Confidence with Exploratory Testing* by Elisabeth Hendrickson, A*gile Testing Condensed: A Brief Introduction* by Janet Gregory and Lisa Crispin, and *Foundations of Software Testing ISTQB Certification* by D. Graham, R. Black, and E. van Veenendaal. I was encouraged to share our learnings at weekly engineering meetings. My team loves to talk theory, so it was never difficult to gather feedback for testing ideas.

As I mentioned in Section 3, my team does not review metrics often. However, I found them to be useful as I implemented these changes to our software testing process. What I found was that I spent an increased amount of time on test planning and test design. Test analysis led to a more robust risk assessment for our products. We also used the findings of our assessment to increase test coverage of our products. Reading shaping documents, dissecting analytics, and discussing user stories led to updates in our cross-browser and cross-platform testing that have increased reliability for our clients that use older hardware or browser versions.

With increased planning and design time, I noticed execution time was speeded. I was able to provide more accurate estimates for test completion. The use of test management software also allowed for more standardized test reporting (and gave stakeholders the ability to see testing status). I did not see an increase in the number of bugs, but I have noticed an increase in the finding of high-priority bugs much earlier in the testing process. Traceability is also improved as I can more easily add software, operating systems, and hardware versions to test runs. Overall, each step that I have taken to create this framework has led to greater success. Because I discuss and display my work more often, the team has a deeper understanding of the many activities involved in quality management. Executing test cases is a small part of my process.

Section 5 Some Scenarios with Matching Frameworks

The Lone Tester Problem
"I work with a team of developers and I am the only tester on my team. The devs tend to get excited about new ideas and testing is often an afterthought. I often only find out about bugs or new features when they are sent to me for QA. I am super busy and I work on a metaphorical island of testing. To top it off, my team is always asking me about the status of their individual branches."

The Lone Tester Proposed Strategy
The loan tester (LT) needs strategies that keep their team informed so that they can focus on the work. If the team tends to forget to invite them to meetings or feels that their presence is not necessary, request access to meeting minutes or recordings. If the lone tester is commonly surprised with new testing work, a long-term strategy is not practical for them. Daily time boxing must be a part of their framework.

Each day, the lone tester could check for new branches in the team's version control system so that LT could be aware of what their pipeline contains. Reviewing Merge Requests provides the opportunity for preliminary static testing. Automation should be added to LT's toolbox. Setting up automation triggers using Zapier will eliminate busy work. In the mornings, automate a Slack or Teams message that explains where you are in the testing process for each assigned project or feature. "Zaps" can also trigger alerts to your team when you update the status of QA in a Jira ticket or Trello board. If there is a way to automate status reports via messaging, do so. LT can also set up their own testing pipeline Trello board and create a "Zap" that creates a Trello card each time a new project branch is created in version control.

If automation is not available, timebox this, too. I recommend starting with 30-minute intervals for any task and adjusting to fit the day. Thirty minutes is more than enough time to create a quick test strategy or test plan that can be shared with your team. The most important thing that a busy tester can do is to advocate for professional development of some sort. This can take the form of class work, joining special interest groups, ensemble coding or testing, or meeting with product managers to learn more about your product. Professional development and access to product information inform your testing process. Share what you have learned, thoughts, and ideas during regular meetings, retros, or asynchronously.

Team Just Ship It
"My team knows that perfection is the enemy of good. They are creative and aren't afraid to build their parachutes on the way down. This means testing in production is very common. I'll spend a lot of my time waiting for the worst."

The Just Ship It Solution
This QA team should not wait for the worst, they should anticipate it. This means emergency preparedness and risk assessment should be a key part of their framework. Stress testing and forced error testing can alleviate the stress of production surprises. While this is commonly done during the planning and control phases, a team that tests in production must integrate system assessment into all phases of testing as part of the team's new risk strategy.

Additionally, the QA team should be aware of the most common and critical user stories so that, if testing time is short, these are never neglected. Because there will not be a large time commitment to pre-release testing, regular bug bashes must be scheduled and anticipated. This is an excellent opportunity to pair with the developers and assess the most common problem areas. Bugs tend to cluster together, so this can also be a part of risk analysis for future test rounds.

Conclusion

Many software teams feel that their primary goal is to ship software. Of course, that is important. It is the keystone of their business model! But, we must remember that software is more than that. Our development teams are solving human problems using technical expertise. And the testing team is the bridge between customer expectations and our business demands. Upholding quality is a challenging task! However, creating your own system can provide you with a sense of satisfaction and a process for analyzing and reflecting on business impact. Taking a thoughtful approach to this work allows you to promote a culture of quality and nearly any type of team.

References

Shape Up Stop Running in Circles and Ship Work that Matters Singer, Ryan 2019
https://basecamp.com/shapeup