QUALITY

IMPROVING QUALITY THROUGH INTENTIONALLY BUILT AGILE SYSTEMS

LEADINGAGILE

IN THE EARLY DAYS OF LEADINGAGILE, we would go into a prospective client and ask them what they were trying to achieve by "going Agile." Why would they want to adopt Scrum, or SAFe, or whatever other methodology they had in mind? And what we would often hear is that they were looking to get more predictable, improve quality, reduce cost, or get things into market faster.

These goals were a little counterintuitive because most of the Agile stuff that was out in the world at the time was all about emergent and adaptive work environments. But it turns out that's not actually what people were looking for.

Of course, there have been a few companies along the way who were looking for better product fit so that they could meet the demands of and move into emerging markets and companies who wanted to be able to innovate better and create new markets. But those seem to be the outliers—not the norm.

These companies all had one thing in common, though.

They believed that if they do the Agile stuff, they will get the Agile benefits. So, they would try Agile, not get the benefits, and wonder what they had done wrong.

The reality was that, from a dogmatic Agile point of view, they hadn't done anything wrong. They just hadn't been thinking about the problem the right way and had failed to take the necessary steps to create the conditions that Agile needed so that it could thrive and help them achieve their desired business objectives.

Because that's really what it's all about. It's not about daily standups or big room planning if you're doing SAFe. Adopting Scrum, or SAFe, or any other methodology is about understanding what it is about Agile that will help you achieve your desired business goals and going to out and creating the conditions for those elements of Agile to take root and flourish.

In this paper, we're going to explore what it takes to adopt Agile, but through the lens of improving quality.

WE'LL DISCUSS

HOW TO THINK ABOUT QUALITY

THE DIFFERENT KINDS OF QUALITY LARGE COMPANIES CARE ABOUT

THE SET OF CONDITIONS THAT HAVE TO BE PRESENT FOR QUALITY TO IMPROVE

HOW THE RIGHT CONDITIONS IMPACT QUALITY

COMMON ANTI-PATTERNS & FAILURE MODES

SCALING AGILE TO IMPROVE QUALITY AT THE ENTERPRISE LEVEL

RETHINKING QUALITY

Back in the early days of project management, you would hear people talk about the triple constraints: Time, Cost, and Scope. Then people came along and hypothesized that maybe it wasn't so triangular. Perhaps it was more like a square, and there was a fourth constraint: Time, Cost, Scope, and Quality. And, as an industry, we explored that hypothesis for a little while.

What we've come to believe is that quality is an attribute of scope—this is an important nuance. If you look at the way project management has traditionally gotten done, and you're living in this world of triple constraints, you're assuming that quality is level.

You don't want to bad product into the market, so you do all the things you know to do. You write up the requirements documents, come up with resource and cost estimates, work with the teams to decipher their availability, and derive a schedule.

And that all sounds good, but the reality is that there's a ton of uncertainty in the system, time and cost estimates are often wrong, and there's a tremendous amount of pressure to get product into market faster.

So, what tends to happen is that quality becomes the one area that the delivery teams can kind of get away with cheating on a little bit. When an organization doesn't have a balanced system, when it struggles to manage capacity & demand, there has to be an outlet somewhere. And quality suffers as it becomes the lever that gets pulled when the organization is operating in an unsustainable way.

Fast forward a few years, and along comes Agile. Now, if you go back to the Manifesto days, the methodology du jour was Extreme Programming (XP). XP was all about technical excellence and craftsmanship, and so this kind of became the model for how we think about quality.

But, quality in today's world, the world of large, complex enterprise environments, goes beyond technical excellence and craftsmanship and has become a way to measure the

excellence of several different facets of an organization's system of delivery.

DEFINING QUALITY

So, as it pertains to Agile, we tend to think about quality as having five different elements. So, let's explore what those five elements are, and then we're going to talk about how you deal with those elements at various levels of scale.

Intrinsic Quality

This kind of quality is the element that encompasses the traditional craftsmanship aspect of quality. When we're looking at this type of quality, we're looking at things like:

- *Are we building technical debt?*
- *Are we architecting the system well?*
- *Are we making the code base resilient?*
- *Are we doing test-driven development?*
- *Are we doing unit tests?*
- *Is the code easy to change?*
- *Are we making the code where it's easy for the teams to understand it?*

There's this whole class of things that the customer doesn't necessarily see, but that are important to maintaining the overall health of the system.

Extrinsic Quality

This element also encompasses more traditional ways of thinking about quality. When we're talking about Extrinsic Quality, we're talking about does the software meet the Acceptance Criteria, does it perform

the way we expect, does it have bugs or defects in it.

Anything that can escape the Sprint and end up in the backlog, or worse, in production, that impacts the usability of the system of delivery is what we're thinking about when we say Extrinsic Quality.

EXTRINSIC AND INTRINSIC QUALITY *make up the foundation of the conversation that's historically taken place, especially when it comes to Scrum and XP—we want really good code health, we want really good architecture and all that, and we want the software to actually work.*

But **QUALITY** *doesn't end there.*

Fitness Quality

Now, we're getting into the interesting elements. This one is a little more subjective than the first two. Fitness Quality has to do with what someone might call the quality of the implementation. Here's an example of what we mean. Let's say you're working on a Feature and that Feature is a login function. You could have a beautifully written piece of software, fully unit tested, and architected with tons of Intrinsic Quality.

It could work perfectly, with no visible defects, and have a ton of Extrinsic Quality.

But, if it doesn't necessarily do everything a customer might expect a login function to do, like maybe it doesn't give you the ability to do an automated password reset, or it doesn't set password strength. If the build-out of that login function lacks those fine-grained User Stories, a customer might say that's a poor-quality implementation.

We should take note of that because sometimes, when we take big

Features and break them down, we'll make tradeoffs in this area because we can do that without impacting the foundations of Intrinsic and Extrinsic Quality.

Market Expectation Quality

This one is somewhat similar to Fitness Quality. When we think about this type of quality, we're mainly thinking about will the market embrace a particular feature, will people use it? And will it, in turn, get used in the way we expect it to get used.

Systems Quality

The last element of quality is kind of like what a lot of people refer to as "Ilities." Things like scalability, reliability, portability, securability, testability, etc.

Together, all five of these elements make up the cloud of things that we consider when we talk about quality related to Agile.

THE RIGHT CONDITIONS FOR AGILE

If the business is investing in an Agile Transformation, whether it's a full-blown Transformation or just trying to teach people how to do Scrum, there's a business reason behind that decision.

If the business is interested in putting a better-quality product into market, there's a set of conditions that must exist, such that the Agile practices you implement will lead to that particular outcome. You can't just assume that Agile will lead to quality and say that you're going to teach everybody Scrum and simply hope for the best.

There is a specific set of conditions you're going to need to create.

We believe that Agile, in any size organization, starts with something we call "The 3 Things." The 3 Things are teams, backlogs, and working tested software/product. But not just any version of these will suffice. Each of The 3 Things is a very specific construct. For example:

TEAMS

- *Consists of six to eight people*
- *Are encapsulated/have very few dependencies*
- *Able to establish a stable velocity*

BACKLOGS

- *Well-articulated*
- *Estimated & Story Pointed*
- *Clear Acceptance Criteria & Definition of Done*

WORKING TESTED SOFTWARE/PRODUCT

- *An increment of work that is valuable to the business*
- *Continuously Tested and QA'd*
- *Meets the Acceptance Criteria & Definition of Done*

So basically, we have a backlog that flows into the team. That team can produce a working tested increment of product at the end of the Sprint. Because each of The 3 Things gets intentionally constructed in a particular way, we know the size of the backlog, the velocity of the team, and can begin to calculate the scope that the team will be able to deliver in a given amount of time.

HOW DO THE 3 THINGS IMPACT QUALITY?

From a Quality perspective, the first thing we like to talk about is the need for really clear Acceptance Criteria. Because without good Acceptance Criteria, we can't get to a Definition of Done. And if you can't get to a Definition of Done, you won't know what the team is going to be able to test against so that we can get to a Sprint demo and understand that we're done-done at the end of a Sprint.

Now, the idea within Scrum is that as we go through the Sprint, we should be validating the product along the way. We don't want to get in the habit of doing late testing at the end of the Sprint. We want a nice, linear burndown chart.

Ideally, and this may not be the best example for every organization, you would have Stories broken down small enough that you could reasonably get them done in a day. Because, and this is the point of using such an example, what you don't want is the QA people sitting around until the last second.

What you want are the developers, QA people, and the PO all collaborating on the Acceptance Criteria. Then, you can start to imagine a world where the developers and QA people are building a little bit of test cases, they're writing a little bit of code, they're doing a little bit of validation, a little bit of defect fixing, and then they get to a point where the team thinks it's done.

When you have the whole team swarming around the Stories like this, you can continuously get Stories done throughout the life of the Sprint. If the developers and QA people stay focused on the Intrinsic & Extrinsic Quality, we can tightly integrate the QA team with the Development Teams, and even the PO to continuously validate.

Why is this so important? Because one of the biggest things that impact a team's ability to move fast, to be Agile, is the ability to make changes without the risk of breaking anything. So, all the teams are doing to ensure that both Intrinsic and Extrinsic quality standards are getting upheld, things like unit tests,

red/green indicators, broken builds, matter. Because focusing on Intrinsic Quality contributes to Extrinsic Quality and increases the speed, confidence, and certainty with which the teams can engage with the product.

It makes the teams, and your organization more Agile.

TEAM-LEVEL ANTI-PATTERNS & FAILURE MODES

So, the idea is that we want cross-functional teams, clear backlog, good acceptance criteria. We want the work to be broken down as small as possible to tightly integrate Dev teams and QA teams as we go. We want to create these feedback loops that are going on throughout the life of the Sprint.

Then, we can walk into the Sprint review with a high degree of confidence that the product is going to work, that it's suitable to its intended purpose, it meets the PO's Acceptance Criteria, and the PO will say that it's ready to go.

WHERE WE SEE THIS START TO GO WRONG IS WHEN:

- *There's no active/ authoritative PO*
- *No ability to write clear Acceptance Criteria*
- *Stories are too vague going into the Sprint*
- *Stories are too big – can't integrate Dev & QA Teams*
- *Teams not writing Unit Tests*
- *Teams not doing Test-First*
- *Teams not getting validation at the end of a Sprint*

Without these fundamental conditions, we have no idea where we're at the end of the Sprint, and we don't know the quality. Or, we get to the end, claim the points, but defects are getting logged, and backlog is building behind us, and it gets out of control. And if the backlog is out of control, then we don't' know where we're at in terms of delivering that product.

SCALING AGILE TO IMPROVE QUALITY

The key to Enterprise Agility is really solid, well-executed team-level Agility. We have to get the practices right at the team-level, in parallel with the execution at higher levels of the organization.

Most organizations that we walk into are rarely working in a single-team Agile environment. They usually have some sort of multi-tiered governance structure that consists of a program and portfolio tier, stacked on top of the delivery teams. So, you tend to have Epics at the portfolio tier, which break down into Features at the program tier, and those break down into User Stories for the delivery team.

In many cases, there's a kind of coordinating layer happening at the program tier. And every methodology, whether it be SAFe, or LeSS, has a different implementation of this core reference architecture. Still, there's always something in the architecture of the organization that takes Features and breaks them down into User Stories for each of the teams.

What the organization has to be able to do, though, is it has to be able to assume that the Acceptance Criteria of the Feature is clear so that the Stories that are getting fed to the delivery tier also have clear Acceptance Criteria.

That same organization must then assume that the working tested product getting produced by the teams meet the Intrinsic and Extrinsic Quality standards of the system of delivery so that it can then go back up to the program tier.

Program Level

When LeadingAgile was first getting started, our approach to program-level governance centered around a structure that we call the Product Owner Team. The PO Team didn't necessarily have specific representation, but certain roles needed to get represented.

Each role could have been one person, a group of people; it didn't matter as long as the following four views were present:

- *Architecture*
- *Product*
- *Project*
- *Analyst/QA*

In some of the early iterations of our program-level governance, the PO Team would run the Features through a Kanban-style queue: Analysis, Design, Build, Test, Deploy. We did this because when you have big things that need to get decomposed into small things, especially in the presence of dependencies, cross-cutting concerns, relationships between requirements, etc. there has to be something that holds it all together.

So, the idea was that Epics would come from the portfolio, they would get decomposed in the program queue through the process of analysis and design—in a continuous way—into Features.

Then, the Features would move, Kanban-style, through the queue and get broken down into User Stories for the delivery teams that would eventually get batched up into some release cycle.

Next, Features and their associated User Stories would move into the build queue. While they're in the build queue, they're sequenced in such a way that they can get to the test queue as quickly as possible because we want throughput across that stack. So, as things are getting built, they're being tested. And, you might loop through that build/test cycle a time or two if there were bugs or defects in the product.

But this is where that model got interesting. At scale, in a multi-team environment with a lot of dependencies, the performance of the individual team is almost irrelevant to the performance of the organization. To some degree, we have to assume the quality is happening at the team-level. We have to assume stable velocity of the delivery teams. All of those types of things are leading indicators that we're going to be successful.

But it's only when we begin to test the integration of the Features that we get a sense for whether or not it's all coming together the way we want.

So, that's just one way of doing it. But, governance can get done in a lot of different ways, and it usually depends on the size of the organization. But the pattern is typically a program level and a team level. Sometimes they're separate with dedicated teams for each tier. Sometimes they overlap, and you have teams with shared membership but that have different focuses at different times. In a smaller environment, you might see this all happening within a single team.

Again, each organization is different, but logically, there's some separation in the function of these two facets of governance. And if we're going to think about it logically, you have teams at the lower level who are building the product and whose goal is to package up User Stories as quickly as possible so that we can move Features through the various queues.

So, if you think about that way, you might model that level of quality, that can only be validated once you begin to integrate, at the program level. So, you have Intrinsic and Extrinsic Quality at the team level. Still, the Acceptance Criteria of the Feature implies an Intrinsic and Extrinsic Quality above any individual team—especially if the testing teams are a separate organization, e.g., security or scalability.

So, again, you assume the quality of the delivery teams because that's gotten baked into the Acceptance Criteria of the Feature, and you test the quality as you start to integrate. And what's cool about this is that as long as you're doing the work of breaking down the Epics into smaller batches and sequencing the work across the teams, you begin moving Features more rapidly and feeding them to the testing teams on a more consistent cadence to get faster feedback on performance and scalability.

And maybe you're not deploying or going into release because you don't have enough Features to justify a release. Still, if you're going to start doing a UAT type of thing, you can model that in the Deploy queue instead of an actual release. This way, you're getting Features in front of the client to be validated, which will be better than merely validating User Stories.

Portfolio Level

So, what does quality look like at the portfolio tier look like?

In a lot of organizations, especially large enterprises, it's a gating process.

> **FOR EXAMPLE,** *you might have five gates and you have initiatives, or investment themes, coming in that get broken down into* **EPICS.** *They go into different gates for approval.*

After they've gotten validated at the highest level, they move into the program queue we just talked about.

These Epic-level initiatives are things that we'd like to release in market because each of these initiatives has a literal expected ROI. And we want to get them in market so that we can begin to charge for them, but we don't want to fall into the late validation strategies that we would typically see in a Waterfall, ad hoc environment.

So, as we're breaking down these initiatives, they're moving through the various gates at the portfolio level, we're feeding Features into the program tier and stories into the delivery tier. We're focusing on the Intrinsic and Extrinsic Quality by doing all the testing at the program level and creating faster feedback loops between the organization and the customer on Features instead of Stories.

And as we start to collect a mass of Features, we can begin validating the smaller initiatives in the portfolioand getting product into market to start making sure that the market is

accepting the increments and paying for them in a way that we expect.

And this is where we start getting into enterprise-level delivery Agility.

We want to be able to put integrated sets of Features into market every three to six months. We want to test the market to see if it's willing to buy them from us. We want to know if we're delivering what the market expects and if the market will be able to use it.

If we do that, we haven't committed to giant, 18-month batches; we committed to something like a three-month batch. As we get product into market, as we get feedback, we gain the option to pivot as new information becomes available.

So, the technical stuff we do as we deliver the User Stories at the delivery team level is a leading indicator of the program level quality. Program level quality gives us the ability to test performance and scale and get smaller things in front of the client faster to validate our intermediate progress with the market.

Then our financial initiatives at the portfolio level are being delivered in smaller batches so that we can actually put the things in market and get the ROI we expect.

CLOSING REMARKS

Even though this was a layman's overview that incorporated a lot of different things, but what the main takeaway should be is

HOW THE VARIOUS ASPECTS OF QUALITY FIT INTO AGILE AND HOW YOU CAN BUILD AN AGILE ENTERPRISE THAT IS CAPABLE OF CONSIDERING QUALITY.

Only doing Scrum or SAFe is insufficient.

Daily standups and big room planning are insufficient.

You have to deeply understand how what you're doing, at all levels of the organization, is going to drive quality. And you have to be intentional about how you design your organization so that you can ensure you are creating the right conditions within it so that Agile, and ultimately improved quality, can thrive.

If you want to learn more about how Agile can help improve **QUALITY** *and solve for other business drivers that executive's actually care about, sign up for our* **SIX-PART ON-DEMAND WEBINAR SERIES** *– hosted by our CEO, Mike Cottmeyer.*