PREDICTABILITY

OPEN THE DOOR TO ENTERPRISE-LEVEL TRANSFORMATION BY INCREASING PREDICTABILITY

LEADINGAGILE

We typically think of Agility as being nearly synonymous with adaptability, and that a change in mindset or culture to become more adaptable is the right place to begin Agile Transformation. But the truth is...

ONE OF THE MOST COMMON GOALS ORGANIZATIONS HAVE FOR TRANSFORMATION IS ACTUALLY PREDICTABILITY.

Why? Because it brings the organization the business results and outcomes executives want to see to make the business grow and succeed.

Executives make promises to the market about what the organization is capable of doing, and what they need is for the organization to be able to deliver on those promises, predictably.

Yet while executives ask for predictable results and outcomes, Agilists tend to think you can't get predictable while being Agile. But you can. And also, you *should* if you want to pursue organizational Agility. Because once you show leaders how Agile practices are enabling you to achieve better outcomes by solving for the things that keep them up at night—like increasing predictability—it becomes easy to make a strong case for continued Transformation where executives will participate and invest more as time goes on.

Predictability is ultimately about figuring out how to make and meet commitments and do what you say you're going to do so that the organization becomes a reliable, trustworthy partner of the executives so they know with reasonable certainty that what we say we can provide is what we can provide.

Predictability also helps build trust within the organization. If you start an Agile Transformation and it's chaos, it erodes trust. If we instead come in and use Agile methods to stabilize the system and get it predictable right at the beginning, even in the presence of dependencies and teams that aren't perfectly formed, we build a lot of trust with the organization.

While predictability always begins at the team level, creating enterprise-level predictability comes with an entirely different class of problems to overcome, and therefore requires an entirely different strategy to be achieved.

IN THIS PAPER, YOU'LL FIND OUT WHAT IT TAKES TO BECOME MORE PREDICTABLE AT THE TEAM LEVEL, AND THEN ACROSS THE ENTIRE ENTERPRISE.

Then you can build internal team trust and provide tangible results executives are looking for to invest more in and proceed with continued Transformation.

HOW MUCH PREDICTABILITY DO WE NEED?

Different organizations have different needs for predictability. While some will need the most possible, others won't need any at all.

As an example, at LeadingAgile, the software development team has zero need for predictability. That's because we're building an internal tool that enables our internal teams, we have no commitments to the market, and as long as we're making progress, we're in a good place and continue to invest in it.

On the flip side, let's say you're working for a government, fixed-price contract and triple constraints are locked, you're going to have to be extremely predictable in that case. If you're an organization that needs predictability 3-6 months out or a one-year roadmap, this paper is built for you.

THE 3 THINGS YOU NEED FOR TEAM-LEVEL PREDICTABILITY

Complete, Cross-Functional Teams

Making and meeting commitments to become predictable begins at the team level with complete, cross-functional teams. A complete cross-functional team is a group of six to eight people, give or take a few, who have everyone and everything necessary to be able to deliver a working, tested product or increment of software as described by the backlog. If the team has all of those factors in check, Scrum should work as described within the team.

Backlogs

The complete cross-functional teams need to work against a known backlog, which is simply a stack ranked list of things the teams are going to build, including user stories that are estimated.

We also need to have a conversation around the items in the backlog so the team:

- *Gets clarity on what they're being asked to build*
- *Understands acceptance criteria*
- *Has the ability to agree that what's expected to be done is possible to do*

Once we know these three things, the team must then be held accountable to do what they've agreed to do. This also requires that there is a clear definition of "done" so that when the team says it's done, it's really done. Without a solid definition of "done," it's too easy for there to be emerging "shadow" backlog items that don't get done but are necessary to deploy, like defects and unrealized features, for example.

Getting to a Working, Tested Product

Once the team is solid on understanding the size of the backlog and has a stable velocity, then the team can figure out how many sprints it will take to get through the backlog, or how much backlog they can do in a predetermined number of sprints. With all of these factors in place, they can then reliably make and meet commitments to produce working, tested product.

And with all 3 of these Things in place: team, backlog, and working tested software, we can add together the size and velocity of the team, and find out how many sprints it takes for that team to get through the backlog, or how much backlog can be done in a certain predetermined number of sprints. Once you figure this out, you'll be able to make and meet commitments at the team level.

SIZE OF THE BACKLOG + VELOCITY OF TEAM = PREDICTABILITY AT TEAM LEVEL

Size of the Backlog + Velocity of Team = We can start to derive how many sprints it will take to get through the backlog, or how much backlog we can do in predetermined # of sprints = Predictability at team level

OBSTACLES TO PREDICTABILITY

Anti-Patterns

There are many possible ways we can break the potential for predictability. Here are a few examples of common anti-patterns:

Pattern 1: A product owner comes in, the backlog isn't very well defined, the team doesn't know how to plan for it, and can't stabilize velocity, then builds software and accumulates a list of things that don't work. Predictability broken.

Pattern 2: Everything is right, but the product owner comes in at the last minute and gives a sprint at a time.

Either of these scenarios simply won't work in an environment that needs a lot of predictability, like the organization trying to do a trade show in a couple of months, or trying to deliver on a promised time frame to a customer.

To overcome anti-patterns like these, we have to put more effort into grooming the backlog further out and we'll establish more predictability from there.

Dependencies

The reality is, not everyone starts with a team that has everyone and everything necessary for it to deliver a working tested increment of product or software as described by the backlog. But without this, you will immediately have dependencies on your hands. If you have to depend on another team or outside resources to be able to deliver, you're not going to be able to be predictable. Dependencies are the number one killer of Agility.

To overcome dependencies, you have to create the conditions you need to be predictable. You *don't* want to be answering the question "How do I do Scrum in the face of impossible circumstances?" Instead, you need to do the things that are required to be predictable—otherwise, you simply won't get there.

So, for example, to eliminate the dependency of an incomplete team, you either need to design the team, or pull together a team, so it does have everything it needs to enable it to create a stable velocity. Because without that stable velocity, you also can't be predictable.

Backlog Problems

What about a situation where you are mixing a determinate backlog with an indeterminate backlog, like on some very typical Agile teams where the product is built and supported and the build side is predictable. Maybe user stories, estimation, release planning, and stabilizing velocity all work great. But part of the backlog is interrupt-driven.

Anytime you mix a determinate backlog with an indeterminate backlog, you're going to have to deploy buffering strategies between the two, which can get complicated. What teams often do in this situation is, if the break-fix work is somewhat predictable, you can buffer some time to compensate. Some examples of possible ways to buffer:

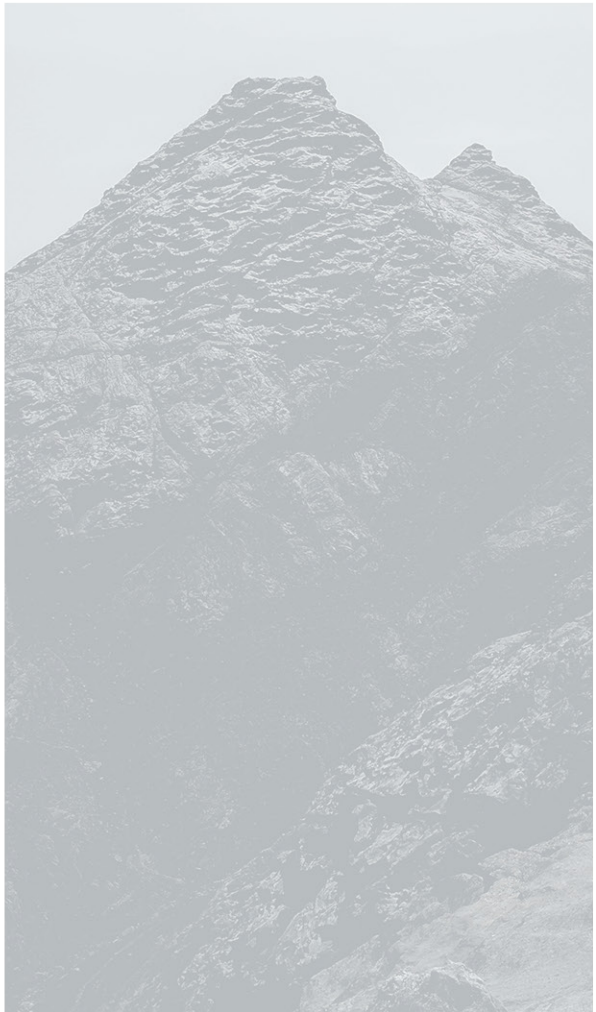
- *Run Scrum for product development, and then throttle work in the process via a Kanban queue*
- *Rotate team members out of the team, or make one team member responsible for triage*

If your break-fix work isn't predictable and your backlog is completely indeterminate, that becomes a different problem. Because in that situation, you won't be able to make and meet commitments. And then it's not about how to do Scrum right so you can be predictable, it's about how to stabilize your product so you aren't in that situation in the first place.

What you are trying to get to is a place where you are not dealing with an indeterminate backlog on the backside.

PREDICTABILITY AT SCALE

Team-level predictability is essential for enterprise-level predictability. But team-level Agility is insufficient for enterprise-level Agility. While the same fundamental concepts apply at scale, they are slightly different to match the specific problems we face at scale.



THE 3 THINGS AT SCALE: STRUCTURE, GOVERNANCE, AND METRICS

Even though this all boils down to the same 3 Things concepts from the team-level; teams, backlogs, and working tested software—as we start to scale, there are slightly different classes of problems and obstacles to predictability, so the focus of each shifts slightly too.

STRUCTURE

Instead of focusing specifically on teams, at scale, we must focus on the structure of the teams we have, and the way multiple teams relate to one another.

GOVERNANCE

Instead of the complete focus on backlogs we use for the team-level, we focus on governance: the way the rest of the organization operates such that we're making the best decisions on what to build, and how to build them with our teams—the flow of work across the organization which supports our ability to accomplish business goals over time.

METRICS

We still need to be able to produce working tested increments of software at scale, but more specifically, we also need to measure value across multiple teams using metrics.

At scale, the obstacles become less about managing and eliminating dependencies, and more about the theory of constraints, and things like bottleneck problems in the system.

If you go back to some early thinkers on the topic of Scrum at scale, and LeSS, the challenge is that they tend to glaze over the fact that most organizations are projectized and approve dollars for projects, and have a lot of dependencies and misalignment within the organization.

Team Level Agility is Insufficient for Enterprise Level Agility

Most organizations are projectized and approve dollars for projects, and then deploy people into projects, and the teams end up having a lot of dependencies. And dependencies kill Agility.

If you are a well-formed organization, and you don't have a lot of dependencies and you are doing team-, visibility-, or value-based project funding, those early methodologies like LeSS and Scrum at Scale are spot on.

But what do you do in a system that has uneven investment at the team level?

The mental model that supports how this happens is that executives and governance bring an initiative to the table; "We want to do X, and get X dollars because then we get more on the backside when we complete the project." We then take those dollars and assign them to teams that we've made by slicing people and teams into little bits to make them fit into the project metaphor to execute on a project.

The reason people want to move people around instead of forming the right kinds of teams and building the right kinds of backlogs is that they want to deploy people into projects.

BUT THE ULTIMATE GOAL IS TO BRING PROJECTS TO TEAMS, NOT TEAMS TO PROJECTS.

So what if we broke the project into pieces and sent those components to the right team to do the parts they are best suited for instead?

But when we break up work to distribute among teams in parts, we end up with teams that have way too much on their plate, and teams that have very little to do and are starved.

And if we try to balance this by looking at the backlog to see if there's more work we can give to the lower value teams so they aren't starved, we still end up with work in the backlog that never really gets done by the higher value teams since they're consistently overloaded to begin with.

So we may be able to solve the problem of underutilized starved teams, but the teams loaded up with low-value tasks will have more to do, while the high-value teams remain constrained.

This is a tricky situation because it's not a problem of prioritization, or dependency management, or even a making- and- meeting- commitments problem. It is that you will simply never get high enough in the queue to get the attention of the constrained teams that are producing all the work that has the highest business value.

The strategies that we tend to try to employ to fix this situation include:

- *Balancing at the portfolio level*
- *Making investment decisions that are aware of the capacity of each team*

So the hypothesis you can form at the portfolio level in this situation is: We have constrained capacity on high-value teams. We have tons of capacity on the lower level teams. What we have to do is balance our portfolio to not overload high-value teams. How do we do this? Creating balance at the Portfolio level with a multi-tiered strategy.

CREATING BALANCE AT THE PORTFOLIO LEVEL

Multi-Tiered Portfolio Strategy

To be predictable across the enterprise, what you need is an ability to look at the portfolio of work that's coming in organizationally and make some guesses and assumptions and establish constraints that assert that the organization, based on the predictable, reliable delivery and performance capacity of individual Scrum teams.

Using a multi-tier strategy, we can start making bets about what the organization is capable of delivering, which enables us to be more predictable.

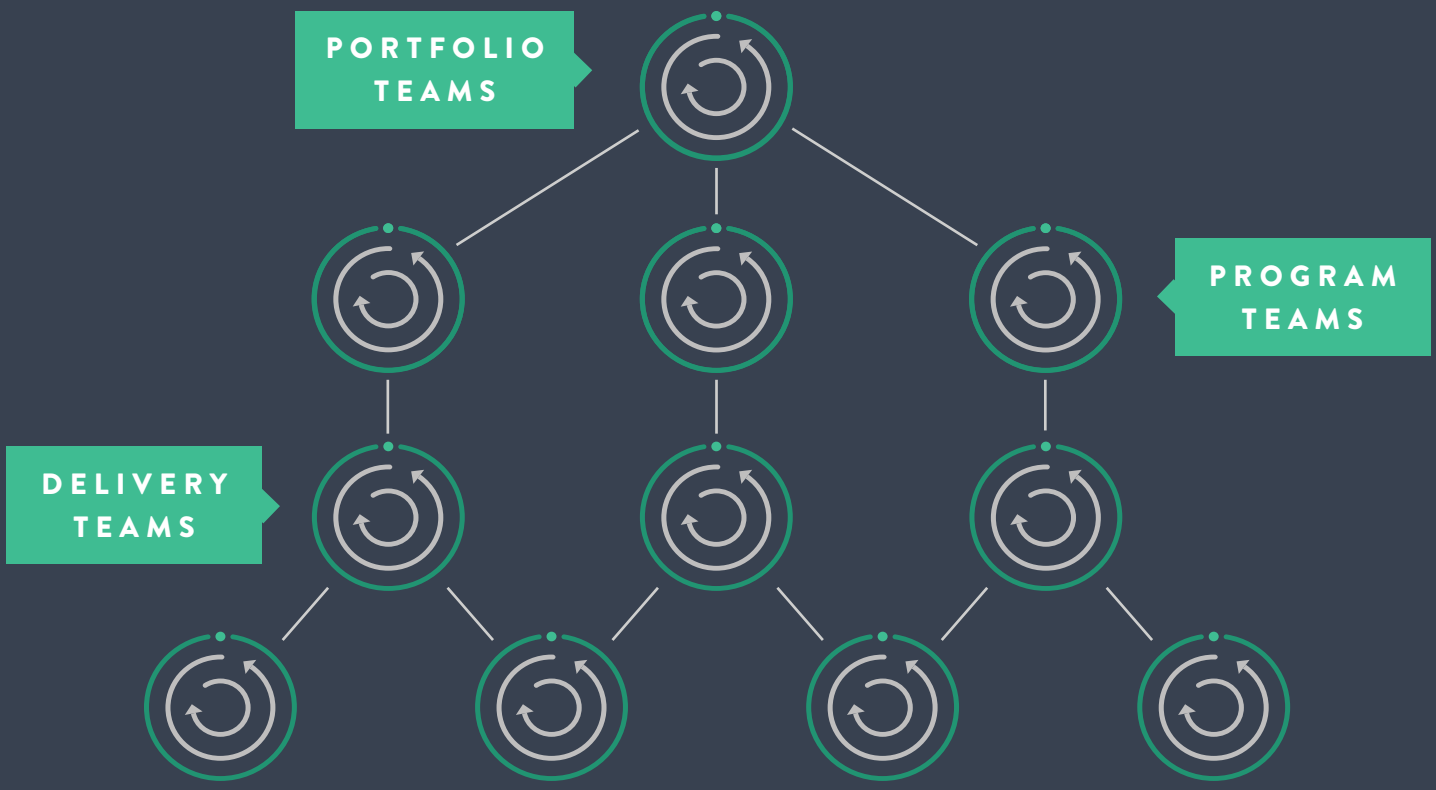
We start with a business problem or opportunity we need to do something about—Epics. Then we have Features, which are the capabilities we intend to build, and Stories, which are what show up in the team backlogs and turn into User Stories. Epics will get broken down into Features, and Features will get broken down into User Stories.

EPICS
Business Problem/ Opportunity

FEATURES
Capabilities we intend to build

STORIES
Action items in team backlog

MULTI-TIER PORTFOLIO STRATEGY



In the top tier, we have a Kanban based governance model with the **PORTFOLIO TEAMS.** This team determines what's the next most important thing to do to support the business strategy, and pulls in Epics.

PROGRAM TEAMS are in the middle tier where Epics are decomposed into Features, the capabilities we intend to build. Then, the result of the analysis and design at the Program level goes to the next tier down.

DELIVERY TEAMS, or Scrum teams, are the final tier, where Features are then broken down into User Stories.

And once this is accomplished, we must align the portfolio level to the execution ability of the lower level teams. But we have to do this with the knowledge that it's not going to be perfect and there's going to be variation and concerns we can't anticipate.

But those concerns are where the middle tier comes in, the Program level. At the Program (middle tier) level, we take the highest value Epics and decompose them into Features in a way that they are scoped to be able to live within the higher-level of constraint, and then decompose User Stories out of the Features and assigning them to the Delivery Teams with Product Owners, therefore feeding the system to align with the business strategy.

To do this, you can create a Program-level working group that, in real-time, distributes work based on the capacity of the teams, so the teams are getting the right things and as they build, and then there is a steady flow of Features coming from the Program level. And as the Features get delivered, there is a steady flow Epics from the Portfolio level.

CLOSING THOUGHTS

A predictable Agile enterprise demands predictable teams or nothing else works.

You can be predictable and perfect at the team level and the organization can see no predictability and value at the enterprise level because you didn't create structures to make sure the system is being fed in a way that funnels through properly.

Once we have a set of predictable teams, we have to make sure we are feeding those teams from a ready backlog and to be able to make tradeoffs in those backlogs to balance where the constraints are across the teams. Then, as Features are being delivered, we must make tradeoffs at the Feature level to make sure we're delivering the Epic.

Once you get things right at the team level, it's about balancing the portfolio and the decomposition processes to make sure you're exploiting the capacity of the delivery organization in a way that maximizes its ability to create flow.

PREDICTABILITY IS IMPORTANT AND BUILDS TRUST, BOTH WITHIN THE ORGANIZATION AND FOR EXECUTIVES.

Executives want it and they're willing to pay for it. And once you show them how Agile is solving the problems they want to solve, like predictability, and the things you need to do to get it, you can make a strong case for Transformation.

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