The Amplio Series

The Path to Effective Lean-Agile Teams

# Amplio Development

Al Shalloway
Success Engineering

**Includes BLAST**
Small scale of up to 9 teams

# TABLE OF CONTENTS

*Please make suggestions via comments.*

*You can also start a dialog on any topic in the free to join* Amplio Community of Practice

# Quick Bookmarks

These are merely chapters I've found I want to be able to jump to quickly. They are here for convenience.

# Preface

This is the first book in a series of books on Amplio. Amplio is Latin for “improve.” It has a double entendre for Success Engineering. The first is to improve how a team, development group, or entire organization improves its effectiveness. The second is how it itself improves. I have been at the forefront of many methods – XP, Scrum, Lean, Kanban, Flow, SAFe, and others. So much so that I have been accused of being a serial adopter. But this is only because I start with something and then transcend it. And move to something else. This has been because of a reluctance of mine to stick with approaches that needed improvement but that the creator didn’t want to do. I won’t need to abandon Amplio because it improves as we learn.

Amplio is a fully fleshed-out approach – at least in my mind ;). I am building new IP for it since the PMI acquired what I had done at Net Objectives. I am currently finishing three “books” related to Amplio Development:

1. Amplio Development: The Path to Effective Lean-Agile Teams. This book focuses on the development part of the value stream
2. Being an Effective Value Coach. This is a book designed to help coaches be more effective with any approach.
3. Amplio Team. A subset of Amplio Development focuses on just a single team. This will not be a separate book but rather a subset of Amplio Development

After these are completed, I intend to codify how Amplio is used at other parts of the value stream. In particular, these include strategy, budgeting, portfolio management, and product management. A field guide on improving SAFe is also in the works. I can already deliver these approaches in onsite training and coaching if desired.

# The Personas for Amplio Development

This workbook is for people who want to learn how to work better. There is a companion book, Being an Effective Value Coach, that I suggest reading with this one. The companion book is designed to convey to associates and/or clients.

There are several personas for Amplio Development. This book is designed for all of them in different ways.

**Primary Persona – Top Level Consultants and/or Coaches**

The book is written for these people. This book also has a subset called Amplio Team that this book prepares these folks to teach. This subset of Amplio Development has not been identified as yet. When it is this subset will be tagged as described in the next section.

**People who want a quick start to Agile.**
I've written a section called Amplio Team that a small team can use to get started on an Agile method.

**People struggling with Scrum, The Amplio Scrum Guide.** Amplio is in no way based on Scrum. But a large group of Scrum adopters appear to fall into this category. I've been in the education business for decades. And one thing I've learned is that you educate people not by presenting something new on its own, but by taking people from where they are to where they'd like to go. Or, more accurately, on the path so they can continue learning on their own. This is ironic because one of the problems with Scrum is that it doesn't do this. It just says "do this" and provides no guidance here.

An integrated subset of this book is intended for these folks. We call it "The Amplio Scrum Guide" because it will result in changes to Scrum Guide Scrum and the creators of that request that we not call it Scrum.

## How to read this book

All readers should start with Part I. After that, they should feel free to read any sections that look most interesting to them. Throughout this book there are "Amplio Sidebars." These are topics that go deeper into presented areas or that provide a reflection on the approach. They can be skipped if they don't look interesting.

## Where To Go For More

This book is used for Success Engineering's Amplio Development Master Class. It is the foundation for Amplio at the team level. You can learn more about what Success Engineering offers by looking at our Learning Journeys.

Please contact me at al.shalloway@successEngineering.works to explore different ways to use this.

The Amplio Community of Practice is a free-to-join community that presents concepts of Amplio on a regular basis.

# PART I: WHAT IS AMPLIO?

Amplio is an approach to help organizations to increase their ability to create more value for their stakeholders in less time.  It's organized around the value stream and is based on the theories of Flow, Lean, The Theory of Constraints, and Human Centered Design.

Amplio's name is a double-entendre. As a verb, it means to "improve."  We must improve how we work, and Amplio itself must continue to evolve and improve.

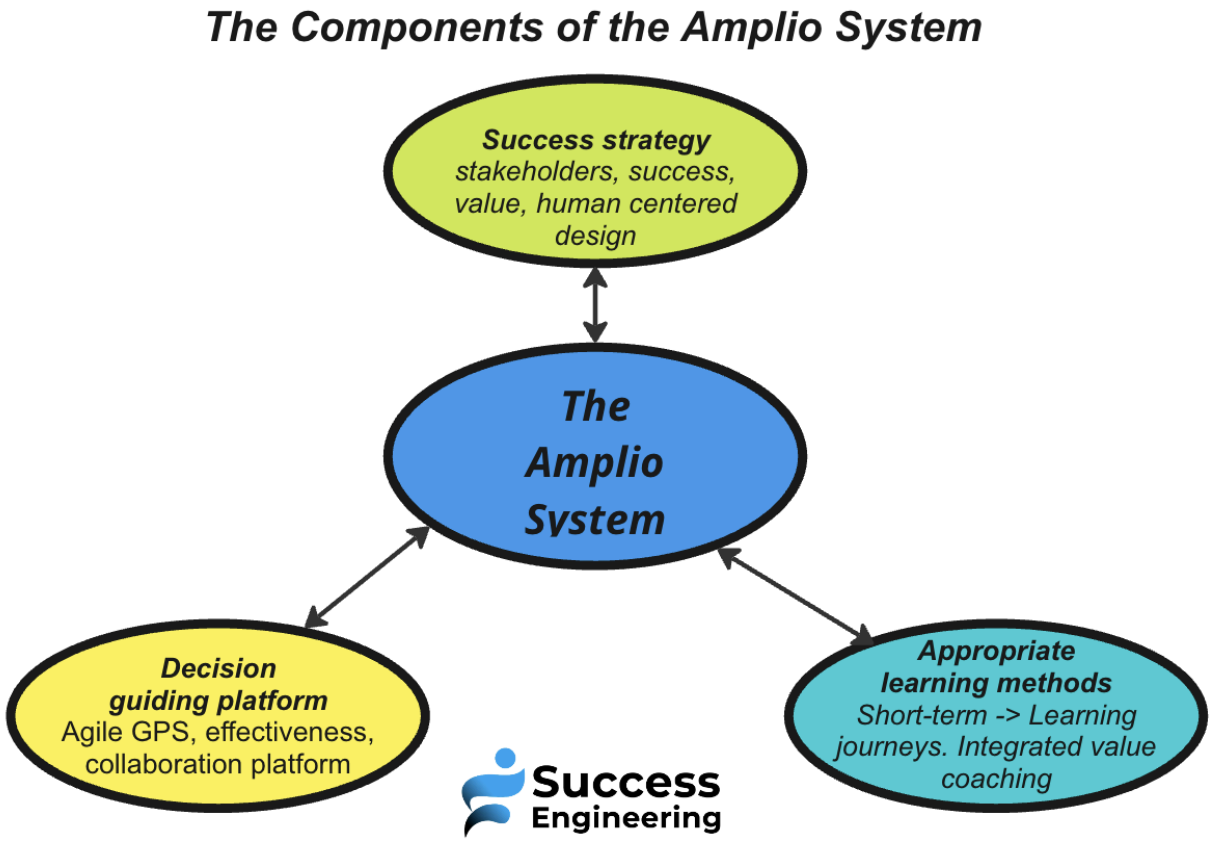
# Amplio Takes Advantage of What's Been Learned in the More than Two Decades Since the Agile Manifesto

In the more than two decades since the Agile Manifesto, there have been major advances in understanding knowledge work, Human Centered Design, what it takes to be successful, how to train more effectively, and the use of virtual boards for distributed companies. While frameworks typically don't provide more than a set of values and practices, it's possible to combine this new knowledge into a holistic system that enables quick, tailored, start that begin and maintain our learning journey that guides improvement.

# Amplio Is a Complete System

Amplio takes advantage of this deeper understanding by providing a complete system. It's focused on helping organizations achieve success. This cannot be done on a repeatable basis merely by telling organizations what to do. It must be supported by a decision-guiding platform as well as a support system for consulting, training, and coaching.

The components are shown in the following diagram.




All of these components are essential. A framework by itself does not provide a sufficient foundation for change. Having old-school training methods increases the cost of necessary education that also impacts any adoptions.

Let's explore each of these in more depth:

## 1. Success strategy: *Stakeholders, success, value, human centered design*

Our focus needs to be on the success of the organization as defined by its stakeholders. This is achieved by creating the value that is important to them. Amplio incorporates human centered design since it is essential to focus on the stakeholders not the system used to create value.

Amplio's product management is driven by Human Centered Design, the Startup Way, and The Theory of Constraints idea of full-kitting. The focus needs to be on both what the customer needs as well as how

what's being built fits into the current and future market. Amplio uses Human Centered Design to help discover what should be built and concepts from The Startup Way to challenge our assumptions and provide guidance into how to define the small steps we build. Full kitting means to attend to the fact that we're not just trying to deliver software, but deliver value and this requires more than just building things at the team.

## 2. A decision-guiding platform

Agile is about self-organization and management. But Agile's popular frameworks tell you what to do. Amplio believes people have access to a great amount of dormant knowledge and can make good decisions if provided with proper guidance. Amplio does this by integrating decision templates for the key decisions that need to be made. For example, how should we plan and how should we organize our talent?

These templates also provide documentation of sorts for the decisions that have been made.

Experience has shown that more poor decisions were made because people weren't provided proper guidance than was due to the complexity of the situation. By eliminating easy mistakes, more focus can be placed on solving more complex issues.

Amplio is a decision-guiding platform designed to allow additional decision templates to be added to it and unneeded templates to be removed. This keeps it from being simplistic or too complicated.

### *Amplio can have you start in ways appropriate for you*

You always need to see where you are. But how to start varies based on your situation. Starting either too slowly or with too much of a jump can doom a transition to Agile. The speed of change depends both on what needs to happen and where it is being applied.

Amplio takes advantage of people's dormant knowledge by relating its concepts to the experience most anyone in knowledge work (Agile based or not) has. While Amplio may make suggestions on what to do, it also explains why so that people can improve their understanding.

Amplio is built on Flow, Lean, the Theory of Constraints, and Human Centered Design. These are all based on systems thinking and first principles. Systems thinking tells us to look at the whole and that most of our problems are from the system, not the individual. It is easier to create a safe space when management focuses on improving systems and not people. The reader does not need to be versed in these theories as this book will explain what's needed as they are used. However, the interested reader can look in the appendix for more information.

First principles describe how the world works. First principles stand on their own. They are not defined but are discovered through observation and relentless evaluation. Violating them has consequences, typically creating waste or lost opportunities. They can be used to provide guidance as to what individuals, teams, and organizations should do or avoid.

Even before COVID, the reality was that teams were distributed. And when they weren't product management was. People need to be able to collaborate across time zones.  We must have tools that help us to work together both remotely and asynchronously - that is, at different times.

Collaboration tools, however, serve another purpose - a basis for learning.

Learning something requires first an awareness that it exists. Then a description of what it is. Very often we have some dormant knowledge about it. Including it on a template reminds us we should consider this. It can also provide a reference to where we can learn more when we find it might be of value.

Collaboration tools therefore can act as initial steps to learning new things and a key into a vast knowledge base without being cumbersome.

# 3. Appropriate learning methods

## Two types of training are needed - each for a different situation:

1. intense half/full day training just before an adoption
2. incremental training over several weeks or even months offered as a learning journey

Intense training is useful if you need to learn a lot before starting something new. Otherwise, most of it is forgotten quickly. Back-of-the-room training, now in vogue, often helps, especially with an entertainment factor, but doesn't provide for in-workplace learning as incremental learning over time does. By integrating modern training methods into Amplio more effective training is possible at significantly lower costs.

## How to Convey the Practices and Theory of Amplio

Knowing what to do is not enough. You must know how to convey concepts to people. This includes how people learn and how to teach complex concepts.

It's also important to know how to talk to people so they don't resist new ideas.

Amplio has a companion book Being an Effective Value Coach. This book discusses how people learn and how to talk to them in a way to increase their understanding.

### Amplio Sidebar: The Relationship Between Difficulty and Motivation

How difficult a change in how to work is often highly correlated with the motivation required to make the change. Large changes also create resistance at times because people may be afraid that the change won't be successful and therefore they will worry about their job, even their self-worth.

Yes, we need changed behavior if we want to become more effective. But how we change behavior depends both on the people involved and the constraints of the situation they are in.

Sometimes a predefined set of decisions like Scrum, which forces the team to make a discrete jump, is useful. Sometimes a transition suggested by the Kanban Method is better. But we should not choose our approach based on just one or two practices. This would be like choosing a recipe to follow because of one ingredient. We need to recognize that a decision is being made here and we should choose what we do based on our need, not based on the brand some guru created years ago.

Furthermore, some teams may need one decision and others may need to do something else. If we believe that teams need to self-manage then it doesn't make any sense to tell teams in a group they must follow one approach. If self-management is a requirement for teams to be Agile then any approach that predefines a practice without providing a way to change it could be said to not be Agile.

## Amplio Sidebar: People are needed to solve problems, but tools are useful.

The value of tools is often dismissed because they don’t solve problems, people do. But if you don't have the tools to solve problems things are more difficult than they need to be. In knowledge work, these are tools to help make better decisions.

In people’s busy days, they often forget to look at things they know they should.

They are empowered by having their attention brought to what needs to be looked at.

Templates to help them check what to look at can be very useful. Not checklists to follow but checklists to remind us and to help us make decisions

A corollary to Edgar Schein’s “We dont think and talk about what we see, we see what we are able to think and talk about.” is “we only make decisions based on what we are thinking of at the moment.

Decision-guiding tools can be a great assistance to us.

# Amplio Addresses Issues of Adoption and Rate of Learning

Besides being a multifaceted system as described in the earlier section, Amplio addresses three issues of adoption and learning. These are:

1. How much to change at the start
2. How much of a tailored start do you want
3. How to disseminate information.

All of these are critical.

## How Much to Change the Organization's Approach at the Start

Towards the end of the book Amplio will discuss how to start. Sometimes it's important to start where you are and do very little change up front. Sometimes it's important to make a jump if the people doing the change have an appetite for it. Many factors are involved including where the organization is, their culture, the urgency of the change, the difficulty of the needed change, and more. Agile adoptions that don't attend to these factors incorporate a risk in the way they start.

## How Much of a Tailored Start Do You Want

Organizations also need to see how much of a tailored start is needed. Some organizations like to do what most others are doing. This provides a degree of comfort. Some recognize that without a fit-for-purpose start resistance may manifest itself.  The decisions to be made here parallel those for how much change to make.

## How to disseminate information

Just as important as how to start is how to keep going. In the Agile space we see two different extremes. Scrum believes in having a "purposefully incomplete" framework that requires people to figure out what's needed. The reason for this is because we don't want to tell people what to do, we presumably have no choice. We can only provide so much guidance without being prescriptive. Of course, Scrum folks ignore that the general approach they are providing is prescriptive. It's like when I joke I didn't prescribe anything to my son when I told him he could play any linebacker position he wanted at any Ivy league school.

The other extreme is to provide everything, as SAFe does. This tends to overload people. This results in them implementing it in stages. Unfortunately, the training covers everything so people pay for more training than they need and have to get a refresher when it is needed.

Frameworks designed in either way require more training and coaching than would otherwise be needed.

The issue that's being ignored is that you want to provide information when it is needed.

Scrum provides too little at the start in an attempt to stay simple. But we don't want a simple system, we want to avoid overloading people with too much information. We can accomplish this by providing a core amount of training and then providing the rest when it is needed.

## We need to shift from a push education system focused on certification to a pull knowledge system focused on providing us insights on how to solve our problems.

In A Timeless Way of Building, Christopher Alexander presents a method where you can have a full decomposition of a complex system that incorporates the relationships between its parts. Amplio is a pattern language where each component is a solution to a particular problem in a context. The language is structured along the lines of a generic value stream, making it easy to find where its components are.

This provides two ways of expansion.

The first is that you can add or remove patterns as appropriate. For example, if your company is regulated by the FDA, a “regulated by FDA” pattern can be added. If an existing problem shows up in a new context you can add a pattern for that. You can also drop patterns if they aren’t needed.

The second is that each pattern contains multiple implementations. Whereas Scrum requires planning via timeboxing (sprints) Amplio can take that approach or choose from others. As we grow options we can add implementations to the pattern.

Thus, we can expand scope while still having it easy to find due to its value stream structure. And, we can expand depth by increasing options in each pattern.

This structure enables people to start with just what they need when they start and then get information when they need it in the future. This enables Amplio to be as rich as we can make it. Every month it is growing in capability. Yet, due to the ability to start with just what you need at the start and to easily find what you need later, it doesn’t get more complicated.

*The trick is not to keep the system simple to avoid overloading people, the trick is to enable people to get the information they need without being overloaded.*

In other words, it’s not the size of the system, it’s the rate at which it can be queried for information.

This means that a user of the system has a two step process to find anything.

Step 1: where are we?

Step 2: which context are we in?

Then they can see our options.

# Amplio Is a Decision-Guiding Platform

## Amplio As An Expert System - How I Created Amplio by Al Shalloway

I created Amplio on the basis of almost two decades of analysis starting in 2004. As CEO of Net Objectives, I observed a dozen top consultants while having conversations with scores of others. By reflecting on their engagements as well as my own, I discerned what successful and unsuccessful engagements had in common.

The challenge was that each of these consultants operated in a different manner. But I was convinced that, although the way they worked varied, what they were trying to achieve had a great deal in common. What I was looking for was not what they were doing but what was the objective of what they were doing. In other words, while two of my best consultants interacted with executives differently, they

were both trying to determine what needed to be created to achieve the greatest value for the organization's customers.

The bottom line was that experts were attempting to accomplish the same objectives but doing so in different ways. Over time I identified two sets of objectives. One was for teams and teams of teams. The other was for an entire organization. Perhaps surprisingly, I found that although companies were in different situations, most companies needed to end up doing the same thing. The exceptions were when there was some sort of government regulation or teams had to interact with non-Agile teams such as those building hardware.

To make these objectives more concrete, I defined each of them in terms of the capability required to achieve it. A capability incorporates both what to do and a means for doing it. I then define each capability in terms of a pattern. That is "a solution to a recurring problem within a context." Patterns provide a targeted set of solutions that adopters can pick from, with sufficient flexibility and guidance to adjust execution based on context.

Most of these capabilities can be defined by a single pattern. For example, how to define the role of the product owner. Sometimes, however, these capabilities are more complicated, and there are many directions to go. For example, should you do iteration (sprint) planning, flow planning, or big-room planning? In these more complicated decisions, you often need to work in a two-step manner. The first is deciding on a general approach, e.g., sprint planning, which specifies a pattern to use. The implementation still needs to be decided on and a pattern can define that. Fortunately, there are only three cases where a capability must first decide on the approach to take and then on the specific pattern to use. The other two are team structures and which type of artifact to use to document Agile requirements.

## Amplio As a Decision-Guiding Platform

Since a capability provides solutions to a problem, we can organize them as a set of capabilities we need to have to be effective, or in terms of the problems they solve. This book organizes them in the former manner so that people can see how Amplio offers a full solution for achieving business agility. A collaboration board that presents many of these challenges from the perspective of the problems they help solve is being constructed here.

A key concept in Amplio is that how to achieve something must consider the context you're in. Almost all capabilities have several ways of being achieved. We use the construct of an Alexandrian pattern. In A Pattern Language, Christopher Alexander states "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice." In a nutshell, each pattern provides a solution to a problem by attending to the context the problem is showing up in and the issues involved to decide on an optimal solution.

I have modified his format slightly because of the different domains he and I are in. But conceptually the intent of the definition is the same.

In a nutshell, each pattern has:

- a name
- a statement of the problem being solved and the context within which it occurs
- the most relevant forces present and how they relate to context
- a general solution
- the intention of the solution
- related patterns

- sometimes potential implementations and / or risks

This enables patterns to provide the problem and solution at a conceptual level with multiple implementations being available. This is useful since we only have a few dozen capabilities to manage but literally thousands of potential implementations.

The patterns listed in this book comprise what is called a pattern language. These work together to create an overall solution for an organization to create value. That is, a collection of patterns that work together to provide an almost complete solution for the people it is intended for. Each of these capabilities includes guidance on whether to use them and if so, how to implement them.

While this sounds more difficult than just picking a preset method such as Scrum, it can be as simple as you want it to be. For example, if you want something as simple as Scrum, just make all the decisions the way Scrum has done. It's like having a radio station with presets, only you can change them when you want to.

On the other hand, if you're in a situation where Scrum doesn't work well, you can tune it slightly to make it more fit for purpose. You can even go all the way and make it fit you completely. How far you go as a preset solution or tuning it for your needs is up to you.

Whichever case you choose, Amplio provides you with two great advantages:

1. You can adjust your practices as needed
2. You don't need to re-invent anything because Amplio both brings up the issue and provides you with a potential solution.

It does this without being overly complicated by presenting information and decisions only when it is useful.

Such a platform is valuable because it supplements the skills, knowledge, and motivations of the people looking to improve their work methods. And it focuses on improving in such a way that business outcomes are enhanced and accelerated by more easily applying the practices and using the platform properly.

By using first principles, which are universal, decision-guiding platforms integrate what we know will apply conceptually with what is needed in our current situation.This differs from traditional frameworks where many of their practices have been chosen based more on the belief systems of the framework's creators which can limit where they apply.

The way we describe the patterns used in Amplio are modeled after Christopher Alexander's approach where each pattern has:

- A name to identify it.
- The **context** in which this problem is occurring
- Objective (the **What**)
- ***Why*** this is important
- The problem to overcome
- Different ways to implement the solution (the ***How)*** that can be chosen from.
- Forces or issues. These must be attended to to determine the best way to implement the solution.
- Symptoms of not solving the problem effectively

An example decision in the Agile space would be how are we going to plan? Choices can be:

- flow
- timeboxed

- big room planning

Each of these can be thought of as a pattern.

Approaches need to provide structure and solutions while maintaining flexibility. It's possible to make a framework more flexible by substituting patterns for the fixed practices of frameworks. A decision-guiding platform provides a way of presenting what needs to be considered without the rigidness of frameworks. This allows the platform to evolve as new concepts become available.

Patterns can provide this needed flexibility because they present a set of solutions that adopters can pick from. This implies that the platform must include a method for selecting the appropriate pattern. Amplio Development provides this with its factors for effective value streams, which is presented later in this book.

## The Advantages of a Decision-Guiding Platform

Many people adopt frameworks because they are well-defined and quick to start with. But they do not always fit the situation within which they are used.

Many teams have decided to create their process, but this is often expensive and ripe with risk unless they have some deeply experienced people involved.

Amplio Development being a decision-guiding platform has the advantage of providing a well-defined approach that is fit for purpose. It provides a set that can be used to start with and tailored to the team(s) using it. Teams can adjust them as they learn.

### *Decision-guiding Platforms Can Change Over Time*

Another advantage of decision-guiding platforms is that they can evolve over time. We can add or remove new patterns as we learn new concepts. This is easier to do with patterns because we are talking about the objectives of the pattern and don't have to worry about the multiple ways it might be implemented.

## How to use the concept of a decision-guiding platform if you're already using a framework.

Each pattern in Amplio represents what is needed to happen. It does not specify *how* it is going to be implemented. Frameworks sometimes specify specific practices (e.g., use timeboxing for planning) and sometimes just specify a general approach (e.g., daily huddle). Either way, the requirement of the framework may not be appropriate for the team involved.

First, consider each prescribed practice as one way to implement a capability. Get clear on what this is. Now you can look for a different implementation that meets the capabilities objectives. This will be discussed more in the Know how to Select A More Appropriate Practice section.

## The Advantages of a Pattern Language

The most common one is to adopt a framework or well-defined approach. Frameworks are popular because they clarify how people are to work together. It is easy to design a framework to specify this. Unfortunately, there is no universal framework that can work for everyone.

Scrum is the most popular team framework. But most teams have challenges with it because Scrum was designed for early adopters in a cross-functional team, building a new product and being allowed to self-manage themselves. This, unfortunately, doesn't describe that many teams using it.

Many teams have decided to create their process, but this is often expensive and ripe with risk unless they have some deeply experienced people involved.

Amplio Development being a pattern language has the advantage of providing a well-defined approach that is fit for purpose. It provides a set that can be used to start with and tailored to the team(s) using it. Teams can adjust them as they learn.

### *Pattern Languages Can Change Over Time*

Another advantage of pattern languages is that they can evolve over time. We can add or remove new patterns as we learn new concepts. This is easier to do with patterns because we are talking about the objectives of the pattern and don't have to worry about the multiple ways it might be implemented.

## Amplio Sidebar: The Challenge with Popular Frameworks

Frameworks are a two-edged sword. On one hand, they provide a quick start and a common focus. But the quick starts are predefined and therefore often not as suited to who is adopting it as they could be. And, while the focus on the framework can help align people in different parts of the organization, it often tends to have people focus more on the framework than on the real job to be done – creating value for the customers.

A disadvantage of the current popular frameworks such as Scrum and SAFe is that they are mostly based on values and practices. While SAFe alludes to first principles, its practices are not truly based on them. Frameworks are essentially defined by their practices and many of their practices have been chosen based more on the belief systems of the framework's creators than on being first principles.

This has organizations following preset practices. Except for a few simple, admittedly useful, practices, no practices work everywhere. *This puts the cart in front of the horse.* Instead, we must first look to see where we are and then see which practices are appropriate. The situation within which they are being applied is critical. *This context must also include the skills, knowledge, and motivations of the people adopting them.* Not doing so means people may not have the skills, knowledge, or motivation to adopt the practices and framework properly.

The danger of this is reflected in Edgar Schein's maxim - "We don't think and talk about what we see, we see what we are able to think and talk about." This means that people will see what frameworks focus on. But these may not be appropriate for them.

A decision-guiding platform can avoid this dilemma by providing us with a focus on what's needed, options to make things fit for purpose, and a way to get a quick start that can also be used for ongoing learning.

# PART II: AMPLIO FOUNDATIONS

# Attend to Value Streams.

## What are value streams?

Value streams refer to the steps required from start to finish to accomplish something. There are many different types of value streams, but all have concepts in common. Value streams shift the focus from the people doing the work to the work being done, both its sequence and timing. There are many different types of value streams and different people characterize them differently For our purposes, we'll classify them as:

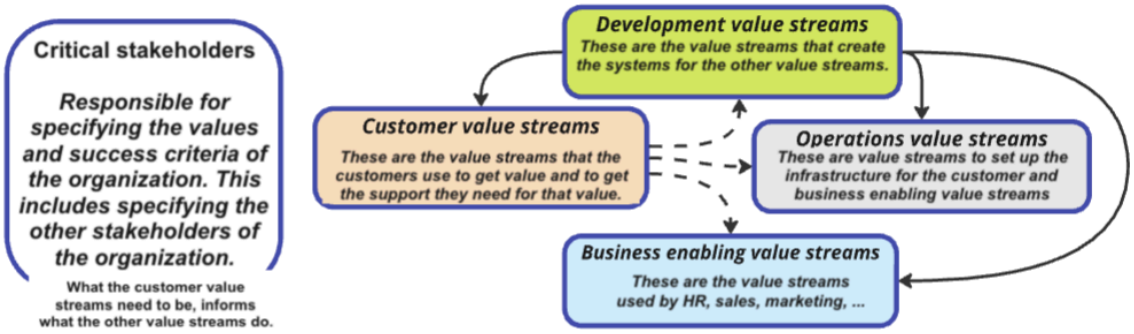
1. customer value streams
2. operational value streams
3. business-enabling value streams
4. development value stream networks

**Customer value streams** are the value streams of the customer, that is, the actions they and others associated with them, take for them to get value. Innovation is primarily about improving these value streams. These include using the product/service, getting support for the product/service, and setting up to get (or getting) the product/service.

**Operational value streams** are those workflows for marketing, selling, deploying the product/service, and supporting customers.

**Business-enabling value streams** are the workflows for the internal process of the company. These activities include buying and maintaining a server network, stocking shelves, HR, and Legal.

**Development value streams** are the value streams creating the ones just mentioned. However, they are not linear as the others are and are better thought of as value stream networks. But the concepts and what to look for in them are very similar.




## The Benefits of Attending to Value Streams

"Attending to value streams" means:

- being aware of them
- noticing how they interact with each other
- when making a change, seeing how it will affect the overall productivity in the value stream

Consider how Apple changed how many of us listen to music. The iPod and its supporting music services changed how people listened to music. By creating products and the systems people used, customer value streams changed. Breakthrough innovation usually comes from providing value to the customer by attending to how *they* work and designing our systems to influence that positively.

In most organizations, people are busy, but the work often moves at a snail's pace because it continually stops and starts. By attending to the value streams, we can see what is happening and how to improve it. This also highlights system constraints across all the value streams mentioned.

Attending to value streams guides us in improving our development work while lowering its costs.

This book is primarily about the development value stream network in which the team is embedded. We will learn to work on smaller items, focus on removing delays, avoid overloading people in the value streams, and get people to be in one value stream to avoid multitasking. Flow, Lean, and the Theory of Constraints all attend to the value stream for several reasons. Mapping value stream networks:

1. Provides visibility on what is being worked on and how
2. Illuminates the delays in value add. These delays also create waste in the process
3. Identifies the constraints in the system

## The Different Benefits of Value Stream Mapping

Mapping value streams is getting to be in vogue. But there are many ways to do it. These include:

1. A high-level view to get a general sense of what's going on and to teach what flow is. This does not involve breaking the workflow down into small steps as much as it shows who the work flows to.
2. Detailed mapping with steps and times. This can be done in days or even weeks. It has two purposes, and the time it takes to do this will depend on that. One purpose is to learn where the problems are and what's causing them. The other is to have people collaborate with each other and see the cause of their problems.
3. Use a generic value stream map which shows what good value streams look like and denote where the company is being effective and where they aren't. This is a kind of Pareto analysis of the value streams. It gets everyone on the same picture while identifying where the challenges are.

# The generic value stream

I have been doing value stream mapping on and off for almost two decades. I say "on-and-off" because I have found value stream mapping can be expensive in time and cost. A little history of my use of value stream mapping helps explain the different options.

### *The history of my use of value stream mapping*

In 2006, in my first Lean-Agile Development workshop I taught people how to do lightweight value stream mapping to see where their real problems were. I then showed them how to use "Five-Whys" to understand what was causing it. Asking five whys is merely asking why something happened, and then why that happened, etc., until you get to a cause of what's happening (in knowledge work there is sometimes more than one "root" cause). I detail this in Using Value Stream Mapping and 'Five Whys' to Get to Root Cause for an example of this.

It was extremely successful. One hour of analysis and identification of the true problem and a 30 minute conversation on how to solve it saved 20% of the work a 100 person person development group was doing.

I, to say the least, was hooked on value stream mapping. For another 8 years I used it at virtually every client I consulted with. Several of the breakthroughs I had in creating new practices were heralded with value stream mapping.. I had great success for several years. However, as Agile became more widespread in organizations, the time required to do it well was usually not made available. I found that value stream mapping to be less and less useful. That at best just gave me an indication of where problems were.

As a consultant I haven't always been given the time to map things properly. I needed to find a faster method that gave me similar results.

### *Dawn of a Pareto approach to value stream mapping - the generic value stream*

I found a few techniques not related to value stream mapping - most notably cafe conversations. However, while these created great collaboration, they didn't provide the insights in terms of the value stream which I knew to be useful if not essential.

Around 2018, I created a new technique - the generic value stream.

The generic value stream is a simple concept. Its purpose is to:

- identify, at a high level, what the challenges an organization is facing
- do this in a way that increases a common vision of what's going on
- do this with many people in an efficient manner
- provide an holistic view of what's happening
- demonstrate why local optimizations are often counterproductive

I had felt this was possible because I had worked at many companies and had seen patterns with what good value streams looked like. How you'd get there would vary, but the end states were surprisingly similar. If a company had special issues such as outside regulatory issues and working with parallel hardware development, these constraints could just be added to the target goal.

I found that if I presented this generic value stream (good strategy, portfolio management, product management, …) in a stepwise manner and had people discuss each stage, it was possible to get agreement that these steps would be useful where the organization was not doing them well.

While the end result is not as effective as a full mapping of the value stream, it can be done at a fraction of the cost while giving a high level understanding of where an organization's problems are. If further detail is needed, a subset of the value stream can be mapped.

I am in the process of creating a virtual collaboration board to help my Amplio Consultant Educators do this "generic value stream mapping." You can learn more about how to do this here.

## An Inherent Problem.

Most organizations are structured in hierarchies to manage the work. But the work flows across the organization. Management is rewarded for the efficiency of these hierarchies instead of attending to the delivery of value. This has adverse effects.

## Managing within a hierarchy

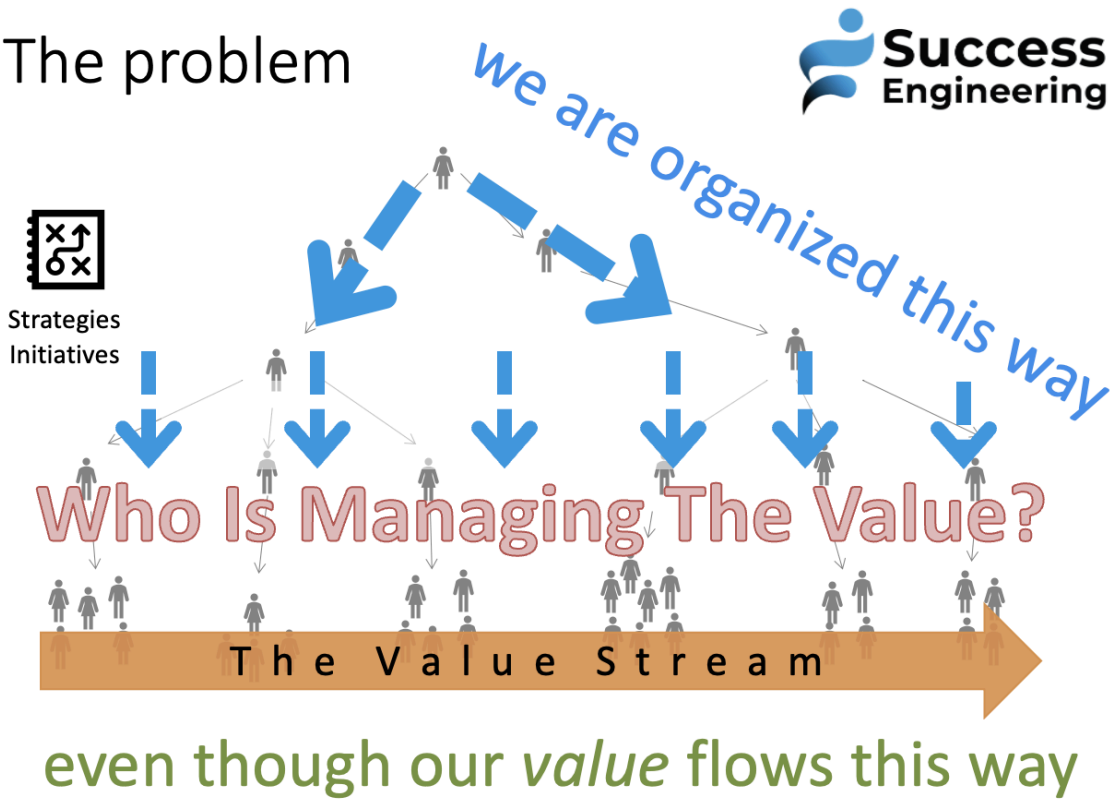
Hierarchies tend to have those responsible for the hierarchy see if the people under them are:

1. working on the right things
2. working well
3. fully occupied

This seems so natural that we don't even think about it. However, it also means that the focus is on the silo, not across the workflow. This results in delays when handoffs are made between the silos. It is common for silos to prioritize work within the silos over cooperation between them. In fact, along with the hierarchical structure are reward and compensation systems that set these silos up to compete, instead of cooperating.

Reflect for a moment on projects you've seen in the past. Let's look at the people doing the work and the work itself (when it is being worked on and when it is waiting to be worked on). See if what you notice matches the following table. Feel free to add to it.

| What's happening with the people | What's happening with the workflow |
| --- | --- |
| ● people are busy<br>● they are multitasking mostly because they are both interrupted and must wait for others<br>● they lose focus on their work, context switch, and become frustrated<br>● people are measured on how well they are doing on their tasks, not on how much overall value is being done | ● the work starts and stops<br>● work items spend time in queues<br>● work often goes forward and then is handed back from where it came from<br>● delays in the workflow tend to cause extra work |




When we make these observations, we often see that no one is managing the work itself but are instead managing the people. Work is seen where it is, but the time it spends waiting is not usually seen.

When we look at the people doing the work, we focus on local steps in the workflow. This is the most significant difference between waterfall and Flow/Lean. In waterfall, we try to maximize the efficiency of the work at each step. The presupposition is that we can do this. Flow and Lean have us look at how long it takes to go from concept to consumption. A project that would take one month if all the people working on it worked on it alone can easily take six months to accomplish when they are busy working on many things.

## The actual cost of multitasking.

Many people point to multitasking as a significant problem. It is a significant one, but not the key one. A significant cause of multitasking is people working in multiple value streams. People are being interrupted and are interrupting others. Correlated with multitasking is work waiting in queues. While multitasking may cause a 10-20% drop in efficiency, work waiting in queues can create unplanned work in the form of bugs; working on the wrong things, rework, etc., is much larger.

These delays of waiting for others also cause delays in feedback. This means errors occur but we don't know that right away. And this means people are working on misinformation. People also often get tired of waiting and move forward when someone is not available and work without a good understanding of what's going on. People are also having to context-switch. They can be working on something, have to hand it over, experience significant delays, and after weeks or months need to look at it again.

## Focus on delays – not eliminating waste

A common Lean mantra is "eliminate waste." The problem is that the mantra comes from looking at manufacturing. In manufacturing, you can see waste. A car is being built, and errors are visible. Planning

can be considered waste because you already know what to do. In knowledge work, the situation is different. First, you cannot see everything. You often don't know when you have an error or not. Also, overdesign and planning are wasteful, however, things like design and planning ensure we are working on the right things are not wasteful. Overdesign and planning are, but these are still necessary functions. On the other hand, re-doing requirements, working from old requirements, building unneeded features, fixing bugs, overbuilding frameworks, duplicating components, and delaying integration testing *are* wasteful.

These wasteful activities create delays in getting the information and using it or when making an error and detecting it. This is yet another reason for quick feedback and reduced delays. The mantra needs to change from "eliminate waste" to "eliminate delays in the workflow."

## It's not bottom-up or top-down, it's attending to the value streams

Many people recognize that hierarchies can be problematic. Many Agile proponents suggest starting at the team. Most are now recognizing the getting executive to buy into Agile adoptions and suggest that a top-down bottom-up approach is required.

While top-down bottom-up is better than just bottom-up, this is still a focus on hierarchies. It often is a "starting at the team with the guidance of management approach" – not a true value-oriented perspective.

A direct focus on value streams is essential. Otherwise we're working on improving local optimizations.

# Dealing With Complexity

*"Complexity is the most insidious enemy of execution. If the environment is complex, the temptation is to mirror the complexity internally. If it is fast changing, the temptation is to match it with the pace of internal change. In fact, if an organization is to cope it needs to create as much internal predictability as it can and to make things simple." Stephen Bungay, The Art of Action: How Leaders Close the Gaps between Plans, Actions and Results (2022)*

*ADD GOLDRATT QUOTES*

## A Fundamental Belief that Separates the Agile Community

Many in the Agile space believe it is impossible to have a well-defined approach to effectively solve complex problems. Underneath this is the idea that cause and effect cannot be seen in complex situations except in hindsight. It results in an approach where we look for impediments and then use "inspect and adapt" to remove them. We try experiments and see what happens.

Systems thinking provides an alternative view. It first tells us that most of our challenges are due to the system. When we look across organizations adopting Agile, we see patterns of challenge. Examples are opening too many stories, having difficulty getting management involvement, and not being able to decompose requirements, …. We can look at other companies that have hit and overcome challenges similar to ours to get an understanding of what's happening.

Systems thinking also tells us that systems are defined by the behavior of the system as a whole, not its components of it. Decomposing a system into different domains, such as simple, complicated, complex, or more, does not really make sense. Instead, we should focus on the relationships present, how understandable they are, and how they interact with each other.

This alone enables a high degree of predictability if we look to see patterns of behavior that other companies are having. We can take advantage of the mantra "if you keep doing what you've been doing, you'll keep getting what you've been getting" by modifying it to "if you keep doing what others are doing that is giving unsuccessful behavior, expect to keep getting the unsuccessful behavior that others are getting."

Dr. Eli Goldratt takes this further. He introduces the notion of "inherent simplicity" when he states:

*"The first and most profound obstacle is that people believe that reality is complex, and therefore they are looking for sophisticated explanations for complicated solutions. Do you understand how devastating this is?"*

*"Inherent Simplicity. In a nutshell, it is at the foundation of all modern science as put by Newton: 'nature is exceedingly simple and harmonious with itself.'"*

*"If we dive deep enough, we'll find that there are very few elements at the base—the root causes— which through cause-and-effect connections are governing the whole system."*

Amplio integrates systems thinking with inherent simplicity in the domain of knowledge work - which is always complex. This enables us to get a good sense of what is happening even when a complex relationship is present. As we try things, we either improve or become aware of what we don't know.

While attending to challenges in our workflow, an understanding of first principles provides us with a model of understanding and enables us to take actions that either provide us with improvement or a deeper understanding of what is happening. Furthermore, this "model" enables us to create new practices consistent with the model and for which we expect them to be helpful. Instead of running experiments, we come from established theory and improve our workflow based on it. Of course, we must validate that improvement has occurred. When it doesn't, it is almost always the case that we've learned something new in our situation.

More waste occurs due to not considering known factors than due to factors obscured to complexity.

Waste is often a result of a small event triggering a problem that is not detected quickly. This small error often cascades into a series of other errors resulting in significant waste. This is an example of non-linearity - a small event creates a large result. By creating visibility and isolating events from each other we can be reasonably confident of avoiding these.

The following sections give an overview of such a model. This model is used to drive improvement and deter the risks of complexity. This integrated approach means that when you learn how to improve your workflows, you also learn how to deal with the risks of complexity.

Some relevant quotes by Dr. Eli Goldratt from The Choice.

> *From his daughter, Efrat – "The hardest thing to do is to struggle to find an answer to a problem when we believe that there is a high chance that it doesn't have an answer; it is so easy to give up. That is why Father recommends starting with the conviction that a better solution exists for sure."*
>
> *A comfort zone has less to do with control and more to do with knowledge.".*

## Five areas of complexity in knowledge work

We can take advantage of that knowledge work, while complex, is not complex in the same way a nuclear reactor or the weather is. In knowledge work we can think of the following areas of complexity:

1. How we should be doing our work.
2. How people in an organization will react to proposed changes.
3. DIscovering the product to build.
4. The market our product is in.
5. What happens outside of our company.

We must accept that the outside world will provide us with surprises, even black swan events. We cannot predict these or even see them coming. Discovering what will create value for customers is also a somewhat emergent process. And although there are some universal truths about people, surprises will always occur. However, we have plenty of insights into how to build products or create new services. We can mostly predict what will happen when we change our methods of working, and when we don't, we learn something that is obscured. So while perfect predictability does not exist, we can move forward predictably well. We will discuss this more when we talk about Eli Goldratt's "inherent simplicity" and Amplio's factors for effective value streams.

The key here is that the way we need to be able to deal with the first three areas of complexity mentioned is to have the ability to pivot. And a solid workflow method can provide that to us. Amplio's approach to systems thinking and complexity is to strive to improve workflow and enable quick pivoting for when we get surprises.

# Dealing with Complexity in Knowledge Work

***Abstract:*** *Systems thinking tells us that the relationships between the components of the system are more important than the components themselves. The system is the behavior resulting from these relationships – not the sum of the parts. A system is more complex the more these relationships are so interrelated that we can't see or understand them. Most systems with people are complex to some extent if only because improving it involves getting people enrolled in change.*

*Complexity is not the only characteristic of a system, however. There are relationships that are complicated (discernible but not obvious at first glance), coupled (meaning changing one relationship affects another), and susceptible to nonlinear events (a small change in one place creates a significant change in another).*

*Looking at systems this way enables us to take advantage of what we know about them even if some aspects are impossible to see at the beginning. Amplio does not attempt to create an ontology of systems in general, but rather focuses on knowledge work. This enables us to take a three-prong approach. The first is to use the factors for effective value streams discussed earlier in this book to provide insights into what is happening. The second is to create tight feedback loops at all levels. The third is to make changes based on the factors for effective value streams. This results in an improvement or reveals a relationship we either hadn't seen or understood. The result is improvement and learning.*

*Navigating complexity means making good decisions, not necessarily being able to make perfect predictions. The key is to avoid the waste of nonlinear events while being able to respond to events beyond our control.*

*Amplio's approach recognizes that there are things we know, things we don't know, and things we don't know we don't know. We know all of these are embedded in our work. Our approach is not to worry about what we don't and possibly can't know but rather to be able to deal with the consequences of this reality.*

*As stated earlier in the book, there are four areas to contend with:*

***1. Our workflow.*** *(In our control) Here we can guide ourselves reasonably effectively. We improve, or we learn.*

***2. Enrolling people to make changes to have an effective workflow.*** *These may be people we work with directly or others in the organization.*

***3. The product to build.*** *(Partially in our control). By attending to the customer journey and the values of our stakeholders, we can understand our customers' needs and how we must innovate for them. At times we'll have to run safe-to-fail experiments, but that should not be the standard approach.*

***4. The market we are in.*** *Markets are always shifting. We must be able to adjust as needed.*

***5. Forces outside of us.*** *(Not in our control). Having an effective workflow (#1) and seeing customer needs (#2) enables us to pivot when Black Swan events occur. This creates a competitive advantage.*

Understanding the first helps us align others on the second and pivot when we need to with the others.

---

*Paraphrasing Neo and Morpheus:*

*Neo - What are you trying to tell me, that I can understand complexity?*

*Morpheus - No Neo. I'm trying to tell you that when you're ready you won't have to.*

Many people in the Agile community have embraced complexity. This has had a disempowering effect. Complexity theories tend to dismiss systems-thinkers as being naïve. However, systems thinking does attend to complexity, just differently.

Dr. Eli Goldratt, the creator of the Theory of Constraints, says, in The Choice:

“The first and most profound obstacle is that people believe that reality is complex, and therefore they are looking for sophisticated explanations for complicated solutions. Do you understand how devastating this is?”

“If we dive deep enough, we’ll find that there are very few elements at the base—the root causes—which through cause-and-effect connections are governing the whole system.”

This is especially true in knowledge work.

In Normal Accidents: Living With High-Risk Technologies, Charles Perrow laid out a deep understanding of complex systems and how they will create unpredictable behavior. He classified two types of forces interacting with each other: complexity and coupling. How these interact is shown in figure 1.

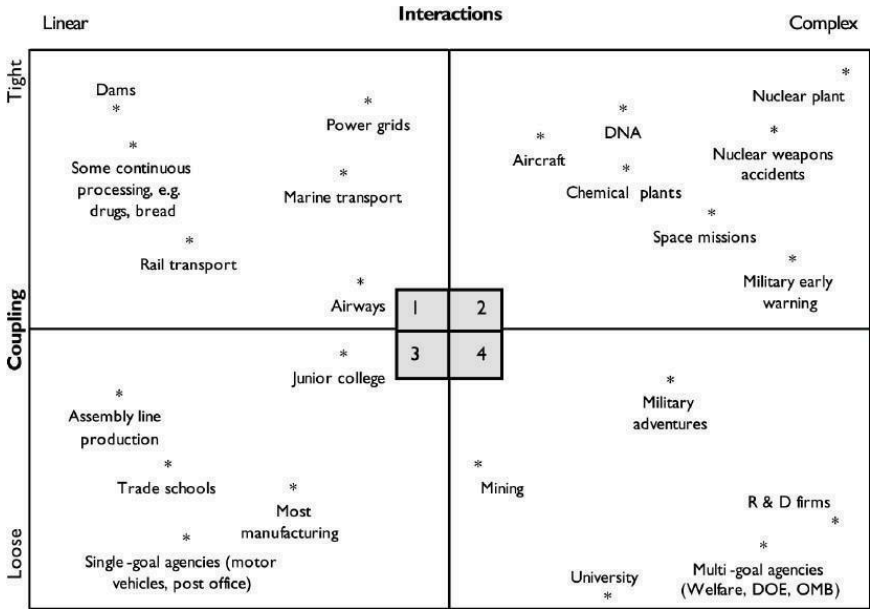



Figure 1: Charles Perrow’s Interactions Vs. Coupling Table from Normal Accidents.

It is worth noting that when people think of complex systems, they think of Three Mile Island, the Titanic, etc. Those systems would be in the upper right-hand quadrant. But knowledge work (he calls it “R&D Firms”) is in the bottom right. In this quadrant, we can avoid system failures that are impossible to avoid in complex and tightly coupled systems. Dr. Perrow discusses this here:

> Systems with interactive complexity (cells 2 and 4) will produce unexpected interactions among multiple failures. But while these are troublesome and unwanted, they need not bring about accidents—that is, damage to a subsystem or the system. Accidents will be avoided if the system is also loosely coupled (cell 4, universities, and R&D units) because loose coupling gives time, resources, and alternative paths to cope with the disturbance and limits its impact. But to make use of these advantages of loose coupling, those at the point of disturbance must be free to interpret the situation and take corrective action. Since the disturbances are generally (not always) likely to be experienced first by operators (which include first-line supervisors and other on-duty personnel such as technicians and maintenance), this means the system should be decentralized.

Perrow, Charles. Normal Accidents (p. 331). Princeton University Press. Kindle Edition.

To state this in another way, we can avoid disasters if we can avoid coupling and dampen what might become a nonlinear event. Disasters happen when a simple event becomes nonlinear because things are coupled, and there is no visibility of what is happening. Complexity is the culprit, primarily because it hides that this is happening.

In knowledge work, it is essential to realize that much of what is going on can be explained through fundamental theories of Flow, Lean, and the Theory of Constraints. However, many relationships are obscured by the complexity of the system. Much of the waste in knowledge work is due to this obfuscation.

We can improve our way of working by creating visibility, adding feedback, & decoupling events to avoid a cascade of errors. While often complex, we can see the relationships between people's actions when they are doing knowledge work. We can't assume a predictable path forward because we can never be sure how people will react. But exploring the causality that is present creates an opportunity for improvement.

Examining this causality is essential for another reason. When you disavow cause and effect because the system is "complex," you also weaken the likelihood that people will want to change. Why should they if it's all a guess? Exploring what cause and effect there is reveals other relationships that can make a difference. Learning and improvement is an emergent process.

The factors for effective value streams is one way of looking at the cause and effect in the system. When one does an informal analysis of challenges Agile teams and organizations are having, I believe the causes of their challenges are (in order of both frequency and impact):

- lack of knowledge of first principles
- lack of systems thinking
- the above results in a lack of attention to the value stream
- nonlinear events due to slow feedback, lack of visibility, and coupling

A nonlinear event (also called a "chaotic event") is when a small action causes a large output. On a graph, we would see what starts as a line and then jumps up at some point – non-line-> nonlinear. "The straw that broke the camel's back" is an example. In knowledge work, this is often the impact of a slight misunderstanding of a requirement.

Complexity does cause problems, but mostly because when something unexpected happens, the above failures make it difficult for a team or organization to respond. We should consider dealing with internal complexity and the fact that teams and organizations are embedded in other complex systems. We can reduce most of the waste caused by internal complexity by attending to the above. While we can't predict what will happen in the market or the world, we can respond quickly when needed by being effective internally.

We can mitigate complexity by recognizing that much of the waste present is due to nonlinear events. That is, minor errors create big waste. But we can dampen this exponential waste. Virtually all disasters due to complexity are caused by these four factors – complexity, nonlinear events, coupling, and lack of visibility. Think of disasters caused by complexity as gunpowder blowing up. Gunpowder consists of a fuel (charcoal), an oxidizer (saltpeter or niter), and a stabilizer (sulfur) to allow for a constant reaction. Complexity represents the fuel. If you don't provide the oxidizer and stabilizer, you won't get an explosion.

You don't need to manage complexity; you manage what makes complexity risky. Use feedback to mitigate the risk of nonlinear events. Avoid coupling when possible and always make it visible.

The claim is not that the world is not complex; the approach to complexity can be what stops us more than the complexity itself. Remember that our systems are more about the relationships between the components than the components themselves. In complex systems (such as knowledge work), some of these relationships are poorly seen or misunderstood. But if we attend to first principles, use feedback to create visibility, and de-couple our work, we can avoid the waste complexity causes. We can also improve our approach in an emergent way. Make predictions based on first principles and the factors for effective value streams. These will either be reasonably accurate or will clarify a relationship we either weren't aware of or which was misunderstood.

We do not need to let complexity impede us.

## Amplio and Complexity

Amplio has been designed to take advantage of what's known about how knowledge work is done. This is very important. Amplio takes advantage of the context and constraints knowledge has.

Before going into how Amplio deals with complexity, it's worth reading some insights from Dr. Eli Goldratt's The Choice.

> "The first and most profound obstacle is that people believe that reality is complex, and therefore they are looking for sophisticated explanations for complicated solutions. Do you understand how devastating this is?"
>
> "When I left physics and started to deal with organizations, I was astonished to see that the attitude of most people is that the more sophisticated something is, the more respectable it is. This ridiculous fascination with sophistication also causes people to altogether avoid using their brain power. You see, since complicated solutions never work, people tell themselves that they don't know enough. That a lot of detailed knowledge is needed before one can even attempt to understand an environment."
>
> "The key for thinking like a true scientist is the acceptance that any real-life situation, no matter how complex it initially looks, once understood, is actually embarrassingly simple. Moreover, if the situation is based on human interactions, you probably already have enough knowledge to begin with."
>
> "Inherent Simplicity. In a nutshell, it is at the foundation of all modern science as put by Newton: 'nature is exceedingly simple and harmonious with itself.'"
>
> If we dive deep enough, we'll find that there are very few elements at the base - the root causes - which through cause-and-effect connections are governing the whole system."
>
> From his daughter, Efrat -"The hardest thing to do is to struggle to find an answer to a problem when we believe that there is a high chance that it doesn't have an answer; it is so easy to give up. That is why Father recommends starting with the conviction that a better solution exists for sure."
>
> "The difficulty is that if I'm not sure, really sure that a second effect does exist, I might stay inside the box; it is always safer to stay within the comfortable boundaries of a box than to jump out into the unknown. Since the other effect is not within that box, I will not find it. I'll give up searching and remain stuck with a tautology (circular logic)."
>
> "A comfort zone has less to do with control and more to do with knowledge."

These insights create awareness of how people are going to react to any approach to complexity.

1. Amplio only concerns itself with knowledge work, not all types of work as shown in Figure 2.

2. Amplio views that knowledge work has aspects of simple, complicated, complex, and chaos in it. Systems thinking tells us we can't decompose into parts and still understand the system. What varies is how much of each of these is present.
3. Amplio pays particular attention to nonlinearity, the cause of most rework and waste in knowledge work.
4. Amplio is based on Inherent Simplicity, the concept that what appears to be complex can be viewed as the result of just a few concepts. Amplio has identified these with its factors for effective value streams.
5. Amplio suggests taking actions based on the Factors for effective value streams knowing that it may be ineffective due to complex relationships that aren't seen or understood. But when that happens, learning will take place and better prepare us for future improvement.
6. Amplio has its approach to complexity integrated into its approach for general improvement. This means you don't need to take additional training or add a new model to use it. In a nutshell, Amplio has us work on the cause and effect we can see to improve our methods and understanding. When we don't get the response expected we improve our understanding by seeing some relationship we hadn't seen before.

We want to keep Edgar Schein's mantra "We don't think and talk about what we see, we see what we are able to think and talk about." Amplio keeps us focused on actions that usually make a difference. We want to learn more about where these work and where they don't. This lets us act on those areas we have control over (our organization) and prepare for events outside of our control – that is, enabling us to pivot to black swan events.

Many executives don't appreciate the difference between complicated (many interactions but all can be understood), and complex (many interactions can't be understood, and that true understanding must emerge. I find using VUCA – volatility, uncertainty, complexity, and ambiguity to be effective in doing this.

Instead of focusing on understanding what domain we're in we presume all domains have a combination of simple, complicated, complex, and susceptible to nonlinear events. Doing this enables us to see how to eliminate much of the waste that occurs in knowledge work. In addition, by enabling quick pivoting, Amplio facilitates responding to Black Swan events that we, by definition, can't foresee.

This has a track record of success.

This focus is illustrated in Figure 2.

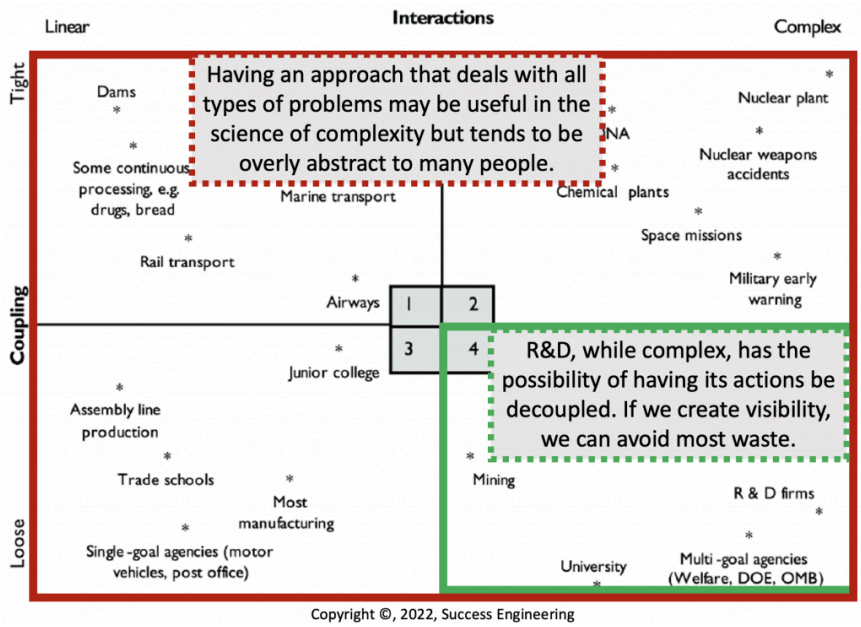



Figure 2: Charles Perrow's Interactions Vs. Coupling Table from Normal Accidents with comments added.

## Examples of potentially nonlinear events

Many small errors can each turn into big waste.

- a slight misunderstanding in requirements that leads to building the wrong thing
- a one-off coding error that causes intermittent problems
- a tight schedule and one person's schedule changes and then is not available when they need to be and it causes other problems
- doing analysis on too many items, then they don't all get started in a sprint, they get started on the next sprint, but no one took advantage of what was learned

## A Sidenote on Why It's Helpful That Amplio's Approach Inherently Deals With Complexity

While this chapter explains how Amplio deals with complexity, it is important to note that this is integrated into how Amplio works. That is, quick feedback, decoupling, and being guided by the Factors for effective value streams, the driver for Amplio also deals with the challenges that complexity in knowledge work presents to us. This is by design - that is, the normal moment-by-moment working of Amplio inherently deals with the complexity of knowledge work.

This is important for two reasons. First, the risks of complexity can strike at any time. The way teams work must deal with it at all times. Second, by integrating it into Amplio teams have to learn less to be effective. While additional training may be useful, it's not required to deal with complexity at the start.

## A Sidenote on The Factors for Effective Value Streams and Inherent Simplicity

The relationship between first principles and the factors for effective value streams should be clear. Small items, short delays, visibility, etc., contribute to being effective. I collected a few of these as actions to

achieve and principles to follow. After reading Dr. Eli Goldratt's The Choice, I realized these factors could be used to look at the entire problem of what's working or not in a value stream. That is, the factors could be slightly tailored to be consistent with Dr. Goldratt's concept of inherent simplicity. One aspect that should be clear is how they all work together, that is, are in harmony with each other.

This expansion of the factors for effectiveness enables them to be our eyes for what we need to attend to be effective. We talk about them so that we can see what is essential.

## Getting Executives to Understand the Value of Quick Flow.

Here's a way to get executives to understand the value of focusing on the most important work in order to shorten delivery time and take into account the cost of delay. Let's pretend we have an enhancement to a very important product to your company that took six months to build but would have taken only two months if the people working on it had focused on it. Imagine that there is also a significant cost of delay by not getting this enhancement released in two months. Imagine you asked one of your top executives: "Would it have been worth paying 20% more to get this enhancement out the door in two months instead of six?" He'd almost certainly say "sure", if there is a clear cost of delay. But now ask yourself, how would you get to do it in two months instead of six? You'd have to cut down delays between steps. Doing so might require adding a little slack to the system, but other than that, removing delay is more likely to reduce unplanned work. In other words, focusing on the most important work and removing delays in its workflow, doesn't increase the work required> It reduces it since a focus on maximizing value flow reduces the amount of waste produced.

This shows another shift that's needed: stop focusing on the utilization of people and focus on removing delays in the workflow.

### What Analyzing the Inherent Problem Teaches Us

- We must shift our focus from people doing the work to work being done.
- We shouldn't manage hierarchies; we should manage the value stream.
- We shouldn't try to go faster; we should focus on eliminating delays by having fewer and smaller queues.
- To align people around the value to be realized.
- To shift from focusing on people's utilization to removing workflow delays.
- To stop striving for local optimizations but attend to the throughput of the value stream.

### Why this shift is necessary

The shifts above will enable us to focus directly on creating value for our customers. This is more effective and empowers people doing the work.

### Consultant's Corner

**Value streams are not business processes.**

There is a difference between value streams and business processes. While there are some similarities, the differences are significant. Especially when going into an organization that thinks they understand their value streams because they have documented their business processes.

One significant difference is that business processes are what you are suggesting to be done. Value streams are what is happening.

There is much value being lost by conflating these two concepts. This post points out some of the differences. One difference is focus—a shift from process to flow of creating value. Value streams are based on lean thinking - business processes stand alone. The context of lean thinking creates a focus on time and value. This reflects the importance of attending to Edgar Schein's observation that "We don't think and talk about what we see, we see what we are able to think and talk about."

Here are some specific differences:

1. Value stream mapping can be used to untangle value streams - that is, not have them cross each other. This occurs when people are in the same value stream
2. Value stream mapping can be used to improve the value-creation structure of an organization
3. Value stream mapping pays particular attention to the delays in workflow and feedback
4. Value stream mapping shows what is. You can also create a future value stream map to create guidance for improvement.
5. Value stream mapping is useful to identify the constraint in the workflow
6. Value streams have an integrated way of identifying if a change will be an improvement to the workflow (i.e., factors for effective value streams)

It may be that business processes can be modified to include this, but an approach that inherently does it is more useful. We have seen too much of what results from “intentionally incomplete” - not many people fill in what should be present.

Additional Resource: Why Leaders Should Focus on Value Streams

# First Principles, Mental Models, and Values.

*Experience [alone] teaches nothing. In fact, there is no experience to record without theory… Without theory, there is no learning… And that is their downfall. People copy examples, and then they wonder what is the trouble. They look at examples, and without theory, they learn nothing.* W. Edwards Deming

## First Principles (Flow, Lean, Theory of Constraints)

First principles refer to the fundamental concepts and basic assumptions from which all other knowledge or understanding is derived. They are the foundational building blocks upon which a system of knowledge or a particular field of study is constructed.

First principles are principles that cannot be decomposed into other principles but stand on their own.

Here are some first principles for knowledge work:

1. **Multi-tasking** causes **delays** in workflow which causes waste
2. **Delays** in getting feedback and in taking advantage of lessons learned cause waste by not taking corrective action quickly.
3. Most of our challenges are **caused by the system**.
4. People **working beyond capacity cause delays** that create additional work to be done (waste).
5. Working on things of **lesser value to critical stakeholders is waste**.
6. We can complete a subset of a large item faster than the full item. **Smaller is typically faster**.
7. We **only see what we think and talk about**.
8. We **can't manage what we don't see**.
9. **Local improvements** don't necessarily result in **global improvements**.
10. **Handoffs** risk a loss of knowledge and information.

Each of these first principles may be self-evident for you. Feel free to skip the Amplio Sidebars that follow that go into greater depth on each of them.

For the reader who wants to learn more about first principles, read First Principles: The Building Blocks of True Knowledge.

### *Elaborating on the first principles*

First principles are used to provide a basic understanding of what is happening and why. Thinking from a first principle standpoint means trying to understand the most fundamental truths in a particular situation and reasoning up from there. You test your conclusion on what you think are fundamental truths. In knowledge work, these would validate to see if you're creating delays in your workflow or in getting feedback (amongst others).

I will not try to prove these based on my evidence but on yours.

First principles do not emanate from anyone, but rather are out there. The ones listed here are not *the* first principles. They are the ones I've noticed and put in Amplio.

### *1. First Principle: **Delays** in workflow, **delays** in getting feedback, and **delays** in taking advantage of lessons learned cause waste.*

Let's break this down into each of its three statements:

1. Delays in workflow ... cause waste.

Consider a time you were working on something and needed someone's help and had to wait for it. You likely stopped working on the thing you needed help for and picked up something else while you were waiting. This is an example of multitasking. Multitasking and delays in workflow go together.

But let's just consider the cost of this work. Say you come back to it 2 days later. What you were doing is now out of your short-term memory. If this involves programming this can be a big cost. This delay literally caused extra work (waste).

But let's consider another case. You write up requirements on a product backlog. You finally get to review the product backlog. You likely don't remember exactly what was written. I remember a time I came back to a product backlog, had no idea what something meant, asked myself "What idiot wrote this" and then remembered *I* was the idiot! :) The extra research or rewriting causes waste.

2. ... delays in getting feedback... cause waste. When you've made a mistake and you don't realize it until later waste results. The cost of fixing it increases as the time to get and use the feedback increases.

3. ... delays in taking advantage of lessons learned cause waste. Not taking advantage of what we've learned means we're not working as effectively as we could.

Summary:

There might be exceptions every now and then but I believe your own experience can provide evidence for the correctness of this principle.

### *2. First principle: **Local** improvements don't necessarily result in global improvements.*

Russ Ackoff explains this with a car metaphor - if you take the best parts of the best cars and put them together you have a pile of junk. "Carness" results from how the parts of a car work together.

A problem we often see is teams attempting to improve how they work. This then requires coordinating the teams.

It works better when teams are aligned and don't go for their own optimization.

This was very self-evident in the case study Coordinating teams with backlog management (video. Will add writeup soon).

While the teams were doing Scrum well before I got there, each team was working in its own way. By introducing some concepts of flow they created alignment. Instead of focusing on their own optimization, they focused on the optimization of the whole.

## Scrum Sidebar: Understanding Why Scrum Works by Looking at First Principles

Why Scrum works when it is followed is clear when one looks at the first principles.

Working in small increments by using timeboxing, removing delays by using a pull system to manage workload, and having cross functional teams aligns well with first principles. It's why it's useful to consider Scrum a partial manifestation of Lean-Thinking. It's not a full implementation because it doesn't explicitly state these principles and therefore does not have a theory to accompany its practices.

Scrum work by increments (batches) and not with a flow method. This is not necessarily a problem since incremental is sometimes better than flow.

*3. First Principle: We can complete a **subset** of a large item faster than the full item. **Smaller** is typically faster.*

Working on smaller items of value provides us quicker feedback. These smaller items often involve fewer capabilities as well making creating them easier to manage.

"Often reducing batch size is all it takes to bring a system back into control" - Eli Goldratt

*4. First Principle: We only see what we think and talk about.*

Our mind filters out most of what is around us. We won't notice things unless they are in our consciousness. This happens when we think and talk about them. It underscores the importance of Edgar Schein's mantra "we do not think and talk about what we see; we see what we are about to think and talk about."

This is one reason sometimes you're pondering on a problem and someone says something related to what you are working on but you hadn't seen it - and then you do.

*5. First Principle: We can't manage what we don't see.*

This should be self-evident. It underscores the importance of not setting up boundaries about what's possible. This is one of the damages that immutability of a framework causes. People tend not to talk about things they aren't supposed to do.

*6. First Principle: People working beyond capacity cause delays that create additional, otherwise unnecessary, work to be done (waste).*

Working beyond capacity causes waste in many ways. It typically causes multitasking which lowers our efficiency. It also delays getting feedback which causes waste. Not to mention that all of this delays the delivery of value.

*7. First Principle: Handoffs risk a loss of knowledge and information.*

When one person has been working directly on something there is a good chance that not all of that knowledge will be transferred. It's a risk, not a certainty, as the person they are handing it off to may have some other insights. There is also the chance of miscommunication. But ironically, sometimes miscommunications create insights that might otherwise not have happened. The point here is to attend to the risks of handoffs.

*8. First Principle: People learn faster when they can relate their own experiences to what's being conveyed.*

Learning from scratch takes a long time. Bue we can often learn from our own experiences.

Sidebar: Using waterfall to explain first principles.

1. Delays in workflow, delays in getting feedback, and delays in taking advantage of lessons learned cause waste.

Look at the delay from start until even a feature is done in waterfall.

2. Local improvements don't necessarily result in global improvements.

Waterfall is all about improving the siloes.

3. We can complete a subset of a large item faster than the full item. Smaller is typically faster.

Waterfall has us do big initiatives because it takes so long - which makes it take longer.

4. We only see what we think and talk about.

People are only looking at their silo.

5. We can't manage what we don't see.

No one is looking at the whole.

6. People working beyond capacity cause delays that create additional work to be done (waste).

The stress is on productivity so scheduling is at 100-120%

7. Handoffs risk a loss of knowledge and information.

Huge handoffs between siloes.

8. People learn faster when they can relate their own experiences to what's being conveyed.

Use this to explain things to people.

## Making First Principles Less Abstract

Although first principles are very powerful, they are also abstract to many people. For this reason we want to take these first principles and build guidance on how to use them more deeply.

While some of these insights are universal, many are applicable only to specific contexts. It is therefore important to understand what context you are in when applying first principles.

## Underlying Mental Models and Values

Our mental models and values filter what we notice. This is often called ‘cognitive bias.’ We must continue questioning our  mental models and ensure they are helping us.

- Take a systems thinking point of view.
- Assume you are working in a complex system and attend to the symptoms of complexity (lack of visibility and potentially nonlinear events).
- Have an attitude of doing things just in time.
- Focus on the delivery of value to the customer.
- Continuously improve your mental models by using both double and triple loop learning.
- Have a positive attitude towards management. If they are not cooperating, ask yourself what they do not see and provide that to them. *Notice how this requires improving their understanding.*
- Respect people.
- Those close to the work usually have the best understanding of what needs to be done.
- Have those close to the work make decisions, but provide them with the information they need to make good decisions
- Attend to and mitigate risk.
- Use metrics to see if you are progressing in the right direction.
- Recognize that work is incomplete until the customer receives value from it.
- Make learning a habit.

# Guidance

There are four main dimensions to effective workflows:

1. what is being worked on
2. how the teams are organized
3. how the work is being done
4. how much work is being done

## Success and Value

- Get clarity on what success means for you and your organization
- Focus on value creation for your clients, staff, organization, and society
- Focus on delivering high value quickly that is easily consumable
- Attend to the quality of the product

## Attend to the effectiveness and efficiency of your workflow

- Use pull, manage queues, and focus on finishing
- Work on small items of deliverable value
- Have short feedback cycles at all levels of the work
- Organize your people to eliminate delays and handoffs in the workflow
- Start with a fit-for-purpose starting point

## Leadership and Management

- Have leadership provide clarity on the why of the organization
- Have management work to create a great working environment for the people of the organization
- have those close to the work make decisions and provide them with the information they need to make good decisions

## Overall Guidance

- Understand the customers' value streams
- Design a customer journey that improves their operational value stream
- Align your efforts around improving your development value stream
    - Focus on value delivery with high quality. Then see how your team works within that context.
    - Recognize that those parts of an item to be released that are not finished represent incomplete work. The more incomplete work that is present, the more risk is present.
    - Look for ways to work with others.
    - Work on small items of value.
    - Minimize delays in the workflow by managing queues, focusing on finishing, and avoiding handoffs.
    - Organize people into teams to reduce handoffs within a team and between teams.
    - View handbacks as symptoms of misunderstanding or discovering a prior step was not done correctly.
    - Attend to the constraints in the system. Don't try to improve everything at once.

- Focus on global improvement and be aware of local improvements that don't address global improvements.
- After completing something, look to finish something else before starting something new.

## A Note About Amplio's Focus on Workflow and Systems

Theories of Flow and Lean suggests that we must attend to the workflow across the organization, not just the people creating the value. This provides a holistic perspective that can shorten the time to market. The Theory of Constraints is also consistent with attending to the workflow. Edwards Deming (the person the Japanese credit with bringing quality to Japanese manufacturing) provided evidence that 96% of errors were due to the systems people used – not so much the people themselves. While this evidence was for manufacturing systems, it also holds in knowledge work.

The focus on workflow instead of people is not a degradation of the importance of people. It is just the opposite. It is based on our trust in people, so we don't need to manage them. Instead, we want to give them an effective, efficient, sustainable environment.

## Why Amplio uses work in process and not work-in-progress

We should always use intention-revealing names, phrases, or names of things that describe what they are referring to. There are currently two phrases for WIP. One is "work in progress," whereas the more accurate is "work in process." This difference is not academic; the usage of 'progress' can lead to bad practices.

Progress means "forward or onward movement toward a destination." But WIP refers to work that has started but hasn't been completed. Work may be blocked, that is, not progressing at all. Work in progress (in English) does not include blocked work. But it is WIP. This has led many teams new to Kanban not to include items that are blocked (not progressing) toward their WIP limits. This is not effective.

"In process" means "of, relating to, or being goods in manufacture as distinguished from raw materials or finished products." English tells us that something blocked is not in progress but is in process.

It's important to have our words mean what is inferred by their standard definitions.

It is interesting to note that Scrumban (the first book on Kanban) uses process as does Don Reinertsen's work.

# Learning

## Single and Double Loop Learning

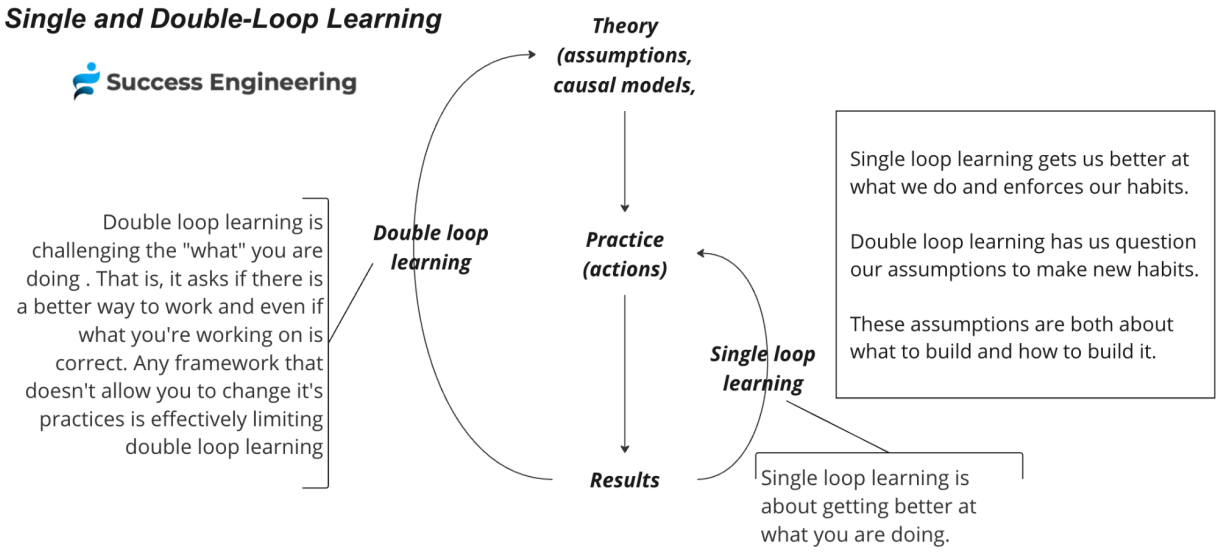
In "Double Loop Learning in Organizations (HBR, 1977)" Chris Argyris clarified that there are two levels to learning, which he described as single loop learning and double loop learning. Here are his definitions:

- **single loop learning**: Learning that changes strategies of action (i.e., the how) …in ways that leave the values of a theory of action unchanged (i.e., the why)
- **double loop learning**: Learning that results in a change in the values of theory-in-use (i.e., the why) as well as in its strategies and assumptions (i.e., the how).

In other words, single loop learning focuses on how you are trying to solve a problem but that doesn't change the theory on which you are solving it. double loop learning has you change the theory (the why) and any assumptions underneath it.

Amplio's meaning "improve" (in Latin) applies both to its intention of helping people using it to improve their methods as well as to Amplio improving itself.

Even the first principles it espouses are up for questioning. We must always keep in mind George Box's maxim "All models are wrong, but some are useful." While we use Amplio's theories and those of Flow, Lean, and the Theory of Constraints on which it is based, we must remember that these are just *theories* and that they can be improved.

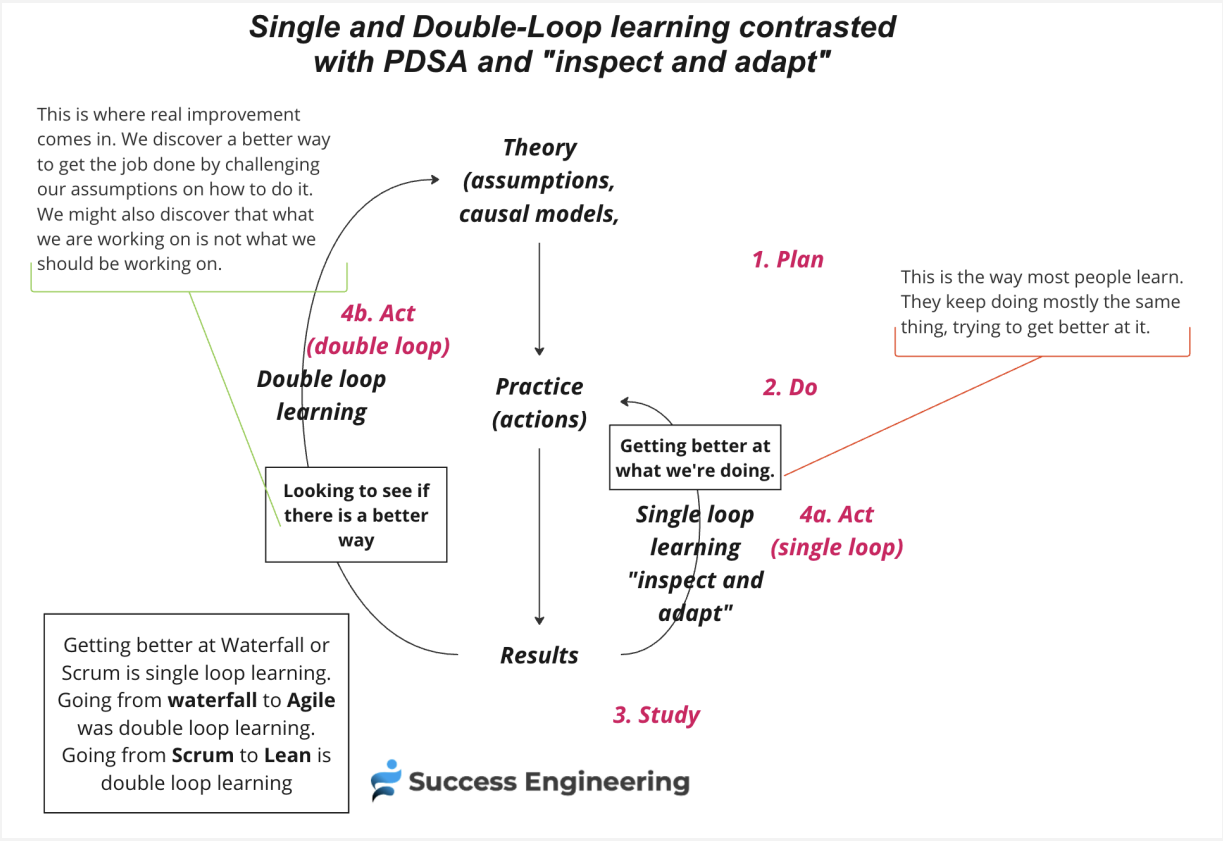



single loop and double loop learning compared graphically.

### Scrum Sidebar: The Difference Between PDSA and Inspect and Adapt Illustrated with double loop Learning

Amplio uses double loop learning both on what is being built and how.
Scrum, with its immutability clause, effectively limits double loop learning to what is being

built.




## Triple loop learning

We can take this one step further and learn how to do double loop learning. This is called triple loop learning.

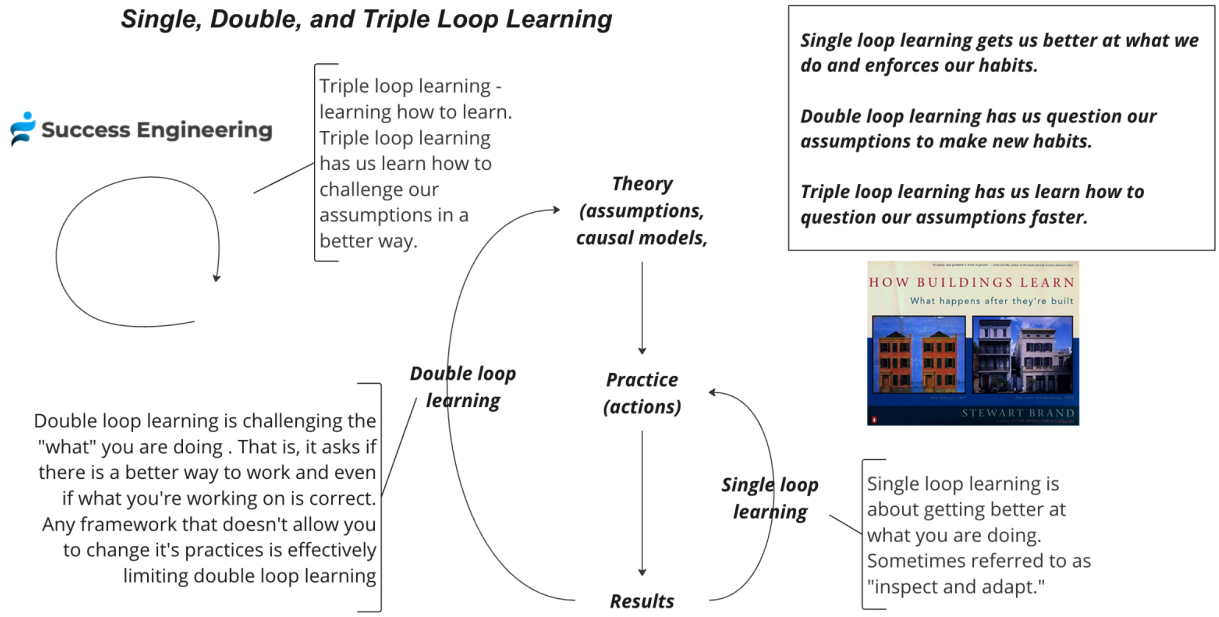



Figure: triple loop learning compared graphically.

## Continuous error detection is a quick path to prevention

Detecting errors and impediments is important. But what's better is preventing them. Early detection of errors with quick action to fix them leads to the prevention of these errors in the future. When one considers that most errors are caused by the system, this means that we can prevent a significant number of errors when we focus on early detection and correction.

This requires having a solid model of how things work, however. Fortunately, Flow, Lean, and the Theory of Constraints provide that.

It is tempting to be very specific at the front and leave the rest to the adopters. But we can't be.

We need to prepare for the journey. Scrum is very specific on how to start. Makes it simple to understand. Scrum leaves how to improve up to its adopters. Makes it difficult to master.

The quicker you put in a correction, the quicker you prevent defects. Early detection is the same as defect prevention. If you move detection and correction up a certain amount of time, you prevent the errors over that time.

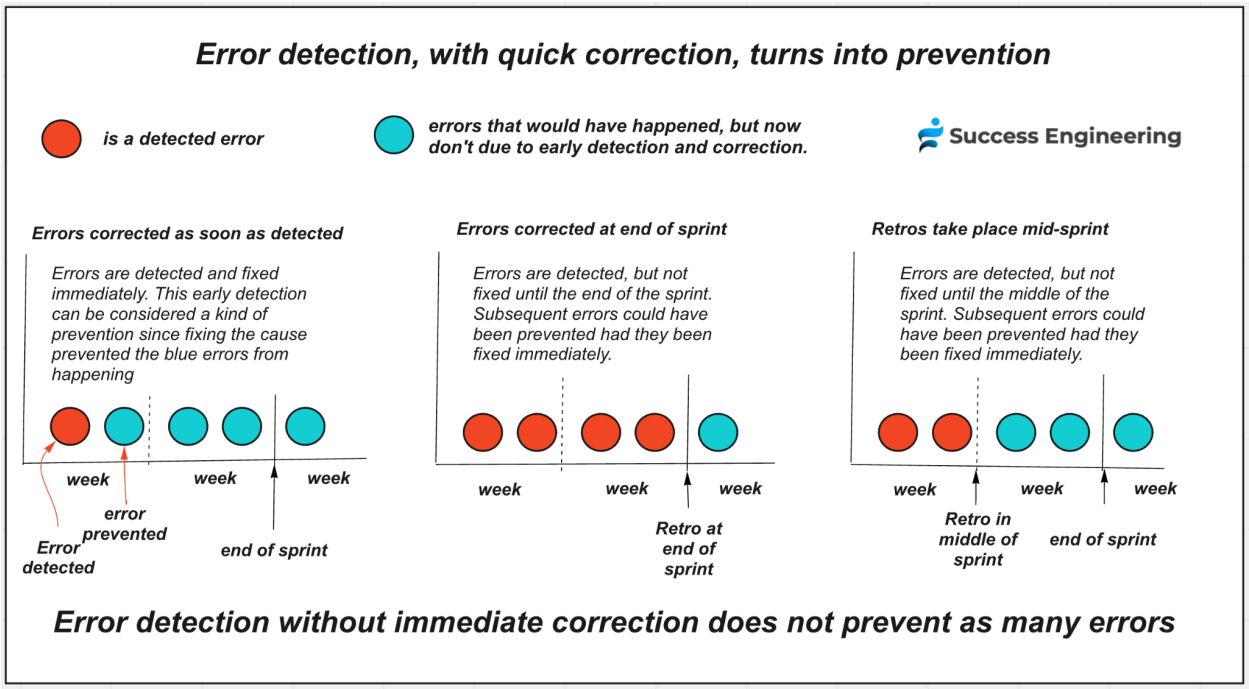
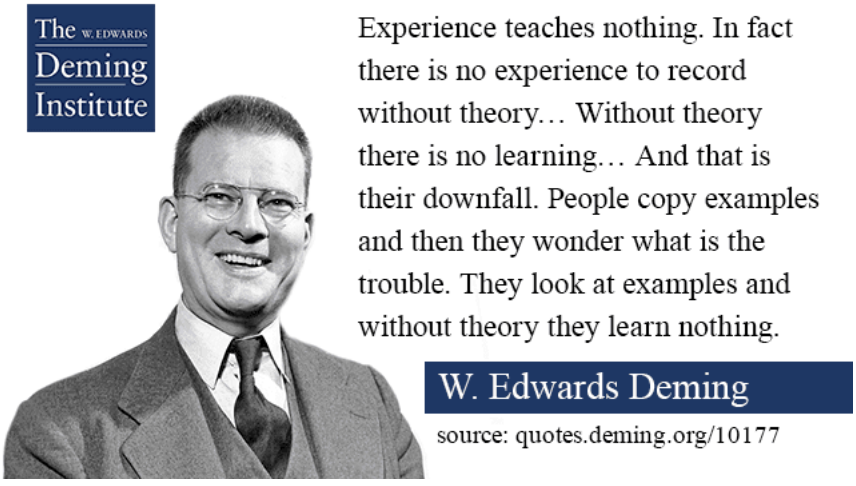



Figure: Error detection, with quick correction, turns into prevention.

## Ongoing Learning

Two popular learning methods are Deming's Plan-Do-Study-Act (PDSA) and John Boyd's Observe-Orient-Decide-Act (OODA). Neither one is inherently better than the other. But the practicalities of each have me favor the OODA loop. It is worth understanding both since each highlights different ways of learning.

A key to both is this observation from Deming:




The essence of this is that when we learn we must not only improve our practices but must improve our understanding on which they are built.

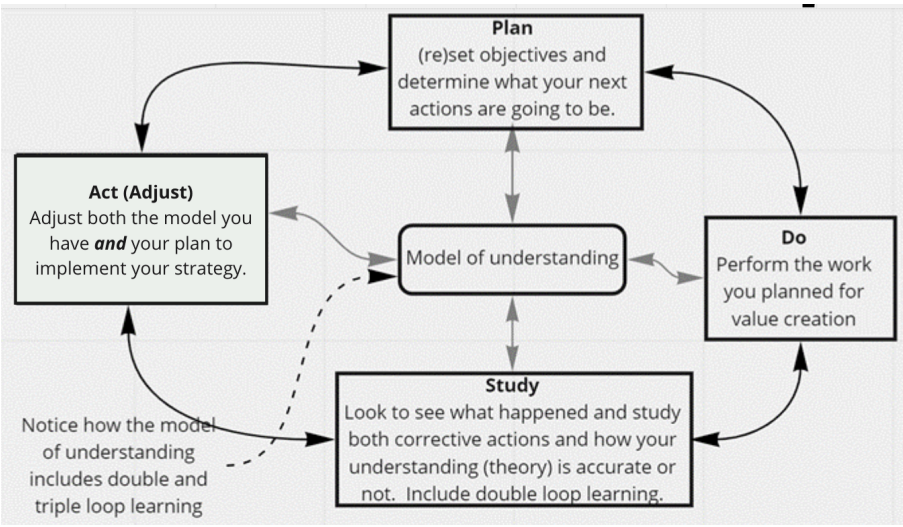
# Plan-Do-Study-Act

Plan-do-study-act means to:

- **Plan** what you are going to do based on your understanding of what's happening
- Then **do** it.
- **Study** the results. This includes questioning whether your understanding of what's happening is accurate.
- **Adjust** both our model of understanding and our strategy if needed.

This is shown in the following figure.

The key in PDSA is that we're using our understanding of the situation to guide us while using the experience we get to improve our understanding. It's a classic example of using double loop learning.




Plan-Do-Study-Act

## Plan-Do-Study-Act Vs Inspect and Adapt

PDSA is an example of double loop learning. Each cycle includes questioning if we can improve our approach. Inspect and adapt has us look to see how we must respond to what happened. But it doesn't have us question our approach.

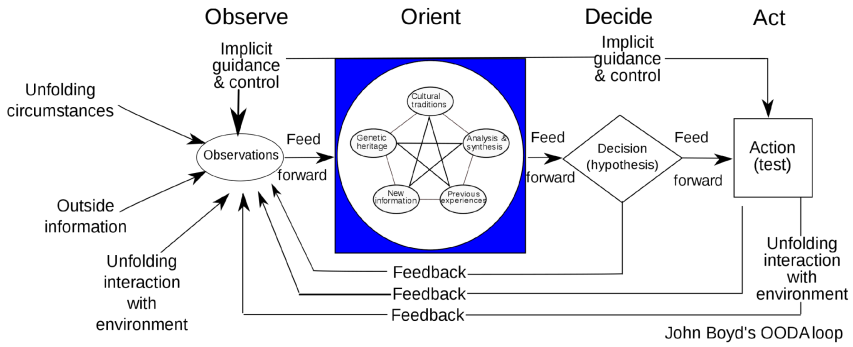
There is a big philosophical difference here. PDSA reflects our belief that much of what we do has a clear cause and effect. Inspect and adapt has us act as if we can't understand this causality. We're not suggesting that the work we do is deterministic, but rather that many of our actions have somewhat predictable consequences.

Lean takes a scientific approach. It believes you can understand the effects that your actions have. Lean suggests that one should consider how they are working to be the best way they know-how. In this regard, their method of working is a hypothesis – "this is the best way to do our work." We make improvements to how we work by suggesting a new hypothesis and seeing what happens. That is, we see how our actions affected our results. In Kanban, we focus on managing work in process levels. Our process hypothesis typically includes a set of limits of different types of work plus service level agreements. We adjust these to maximize the value delivered to the customer.

This is a marked difference between it and Scrum. While Scrum suggests single loop learning, its lack of first principles requires you to stick with single loop learning. From the Scrum Guide “Scrum is free and offered in this Guide. The Scrum framework outlined herein, is immutable. While implementing only parts of Scrum is possible, the result is not Scrum. Scrum exists only in its entirety and functions well as a container for other techniques, methodologies, and practices.”

In the Agile space, you can think of “Inspect and Adapt” as single loop learning in order to improve the process you are doing. For example, a sprint retrospective would have you look at how you could improve what you did in the last sprint. double loop learning would have you look at whether you should even be doing sprints. In other words, single loop learning is questioning how to do your process better while double loop learning questions the assumptions on which your process is based.

## Observe-Orient-Decide-Act.




The OODA Loop

The OODA loop was created by Colonel John Boyd. Some people mistakenly believe it is focused on how to engage in fighter combat. It is much more sophisticated than that. But this is one of its advantages. It can be used with any cadence or cycle length. This includes the second by second cadence of air combat and the month-by-month cadence of organizational strategy. The steps are similar even as the time frame varies.

**Observe.** We must observe both what’s happening outside of us and observe our own biases and understanding.

**Orient.** We ask ourselves how our situation relates to our understanding. In particular, we’re recognizing that our biases affect our observations.

**Decide.** Now that we’re observed where we are and contrasted with where we’d like to be, we can make decisions on what actions we need to take.

**Act.** The actions are now taken.

**Loop.** We repeat the process by seeing what effect our actions have had. We use this to see if our model needs to be updated.

References: Revisiting John Boyd and the OODA Loop in Our Time of Transformation.

## PDSA Vs OODA

If you are already familiar with PDSA there is no immediate need to switch to OODA. But there is a lesson in OODA that is worth considering when doing PDSA. PDSA has the sense of a particular planning

cadence. This, of course, can vary. But PDSA was never intended as a continuous process – although it can be. OODA can be thought of as a variable interval control. OODA can be used at different cycle speeds, so to speak.

Reflection in knowledge work needs to be both continuous and iterative.

## When do you use double loop, triple loop, PDSA, and/or OODA?

The most salient aspect of Lean is that it is about learning. While it's known for eliminating waste, building quality in, just-in-time amongst other things, its total focus is on learning. The question of this section is therefore answered with "as much as possible." This doesn't mean it's *always* possible. If we challenged our assumptions all the time we'd likely not get any work done.

So when do we do all of this?

Two timings are useful.

It's important to set up a regular frequency for doing a retrospective and see what's going on. It's suggested not to be less often than weekly.

That said, anytime you seem to be struggling or things aren't working like you think they should, you should have a special meeting to discuss what's happening.

## Why I Include Comparisons To Scrum In This And Other Books

Amplio is not based on Scrum. In my mind, Scrum's foundation of empirical process control and Complex Adaptive Systems leaves Scrum adopters without a strong understdanding of how to solve their challenges. While Scrum can work, it requires more effort and motivation in most situations than what would be required if they had a deeper understanding of what it takes to do knowledge work.

It justifies this by postulating that "simple" is the way to go. However, there is a difference between how simple something is built and how simple it is to use. While it is true creators of frameworks must avoid overloading adopters, approaches that provide a way to learn in stages can avoid this while providing necessary insights to make work easier.

Many teams partially abandon Scrum after struggling with it for a while because they don't see a way out and have pressure from management to get things done. Ken Schwaber, a co-creator of Scrum defines ScrumBut as "Scrum has exposed a dysfunction that is contributing to the problem but is too hard to fix. A ScrumBut retains the problem while modifying Scrum to make it invisible so that the dysfunction is no longer a thorn in the side of the team".

There are patterns to the ScrumButs one sees. Unfortunately, instead of the designers of Scrum making Scrum simpler to adopt, not requiring reinventing many well-known practices, or providing a way to tell if a change to a practice will be an improvement, they say Scrum is immutable and if you change it, you're not doing Scrum. This lack of guidance (and in my mind responsibility for one's own approach) leaves many Scrum adopters on their own.

Amplio can provide a bridge for many Scrum adopters struggling with it to an easier, more effective approach. When faced with an impediment they can't solve, Amplio's decision-guiding platform provides insights into alternative practices while its factors for effective value streams can tell us if we're avoiding a problem or helping solve it.

People learn better when you teach them from where they are. They do not make big jumps in concepts very effectively. I do not want people to "follow" Amplio or me. I, therefore, provide them with an understanding of what they know and then guide them on how to improve it. This seems like a reasonable approach to me.

## Scrum Sidebar: Scrum's requiring certain approaches limits where it can be used.

Scrum has several constraints on approaches that constrain where it can be effective. Adopting Scrum means you're assuming these constraints are not a problem. However, it shouldn't be used in the following situations.

- when you can't create cross-functional teams
- when you can't plan ahead at least a week or when you can't avoid being interrupted a significant amount of the time
- when you are working on a fixed cost, date, and scope project

Note that Amplio Scrum *can* be used in these situations.

## Scrum Sidebar: Scrum Makes Decisions For You

Scrum makes many decisions for you that may or may not be appropriate. For example, to do Scrum you must have cross-functional teams, do iteration planning, and have specific roles. Sometimes these decisions are appropriate - often they are not. The difficulty, of course, is making them at the front creates risk. Even when they are fine at the start, things change and then Scrum's immutability comes into play. While Scrum proponents claim you can change them if you like, this ignores the reality that when teams have been told to do Scrum and have hired a Scrum Master, most feel pressured to keep true to Scrum.

Amplio doesn't find it acceptable that something is "simple to understand but difficult to master." Although building a product may be difficult to master, the approach should be simple to understand and relatively easy to master. Chess is designed to be difficult, product development shouldn't be.

While you do need to learn the rules of the game, you need to make sure you're playing the right game. And you always want to understand why the game's rules are there. This contrasts with many Scrum proponents saying, "Scrum is like chess; follow the rules first, then you can change things." The problem, of course, is chess is a game. See this sidebar, Scrum is like chess, for more.

Amplio does not require any particular practice to be used. The theory presented in Amplio allows you to improve on the practices provided without the danger of making your workflow worse.

Some framework proponents claim "You must follow until you understand." This is both wasteful and carries a big risk that the start is not appropriate and people give up or go down

a bad path. When this happens these same proponents blame the adopters saying they didn't do what we said. This is like "buy low sell high" advice where if you don't do it it's your fault.

It's true you need some experience before understanding. But chances are good you have the experience needed. It just may be in doing things wrong. This is enough, however, when how to do things properly is pointed out, for you to understand what's being suggested. This is called "dormant knowledge."

Amplio is designed to take advantage of this knowledge.

Amplio doesn't focus on being "purposefully incomplete." It focuses on being "purposefully sufficient." It focuses on providing adopters the information they need, when they need it, without overloading them.

## The Relationship Between Motivation, Difficulty, Effectiveness, and Taking Responsibility

I hear many consultants of other methods explain away failures of their approach with the statement "well they just weren't motivated enough." This, of course, is just circular logic. "Why'd they fail?" They weren't motivated enough." "You mean they failed because they weren't motivated enough?" "Yes." There is no evidence. It's just being explained away. Worse, this justifies not taking action to improve the approach that just failed since it's not its fault, it's the fault of the people using it.

As I mentioned in the section on The Amplio Attitude, Amplio takes a "Sense of responsibility and commitment." This means that when Amplio is misused, I look to see why and am committed to improving it.

It is important to be aware of the relationship between motivation, difficulty, and Effectiveness. While the target is effectiveness, we should look for the lowest amount of energy required for the highest amount of effectiveness. Fit-for-purpose approaches often take less energy to achieve effectiveness than approaches that are mandated by a particular framework. Not being fit for purpose is correlated with requiring more energy and more motivation. While people may be up for it, this often results in a less effective approach. And even when people do put in the extra effort to achieve a reasonable result, it means this extra energy and motivation has been used when it might have been better served elsewhere.

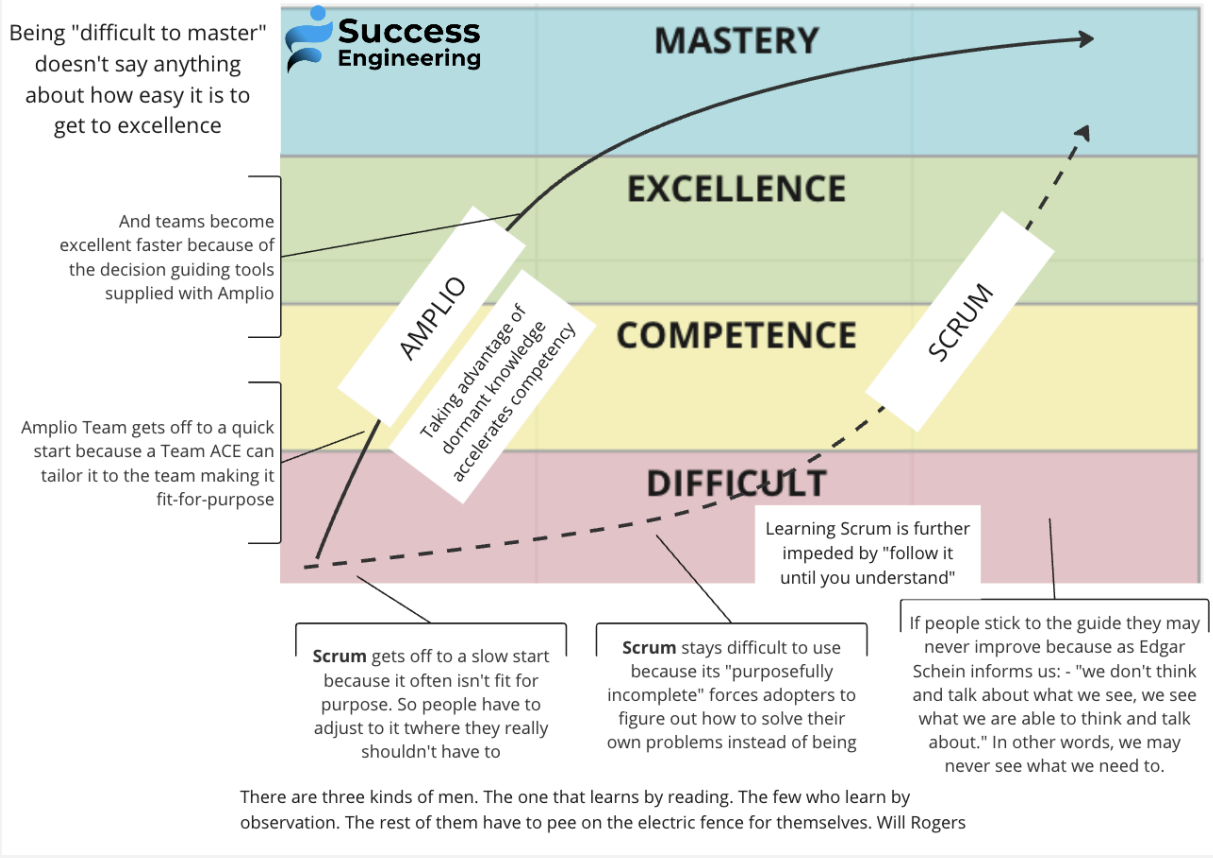
Amplio's flexibility is intended to lower the amount of motivation required to achieve great results by enabling it to be fit for purpose.

### Scrum Sidebar: Beware of Scrum's Immutability

Choosing to do Scrum means choosing to use its prescribed approaches. Unfortunately, Scrum doesn't explain why and where these work. In fact, they acknowledge you don't understand this with the statement "follow until you understand." Scrum's immutability means you'll always get the same start with it.

Unfortunately, these practices may not be fit for purpose. This means they will take more energy and motivation to achieve comparable results. It may be that you're not motivated to

do Scrum well but are motivated to use a fit-for-purpose approach.




## Taking Charge in Complex Adaptive Systems or Stop Throwing the Baby Out With the Bathwater

You hear many in the Agile space saying that the fact we're in a complex adaptive system means predictability is not possible. That we can only improve our approach through "fail fast" experiments. This is both simplistic and a gross generalization.

This article discusses:

- what we can and can't predict about future actions
- an alternative viewpoint on cause and effect
- why we should look at relationships not domains
- how we can move forward or at least learn as we change how we work
- the path to understanding is the same path to agility
- what we know causes waste

### What we can and can't predict

There are many aspects of product development. These include:

1. discovering what product must be built
2. how long this will take and how to schedule for it
3. how we communicate with each other
4. how we do our work to keep workload below capacity and achieve quick feedback
5. how teams should be organized
6. how individual teams should work
7. how teams interact with each other
8. the role of management

While the first three of these are usually beyond our ability to predict them, the last 5 are of a different nature and we should explore what we can do here before discarding any abilities we might have here.

## An alternative viewpoint on cause and effect

“The first and most profound obstacle is that people believe that reality is complex, and therefore they are looking for sophisticated explanations for complicated solutions. Do you understand how devastating this is?” Dr. Eli Goldratt

"If we dive deep enough we’ll find that there are very few elements at the base - the root causes - which through cause and effect connections are governing the whole system.” Dr. Eli Goldratt

It is important to realize that Dr. Goldratt is not saying that we can see all of this cause and effect. He is just noting that it is there. Some we can see – for example overloading people with work causes delays in the workflow which causes multi-tasking and waste. Some we must discover.

We still have to try things. Run experiments at times.

But these shouldn’t be like “let’s try this and see what happens” very often.

Instead, it should be more to gain improvement in our workflows or achieve a better understanding of them which will lead to improvement.

Deming said (paraphrased) “Theory without practice is useless. Practice without theory is expensive.” Much of Agile has become very expensive. It is harder to guide ourselves and learn. Immutable methods work against this.

The idea of having problems and then solving them is not as good as taking a perspective that avoids the problem in the first place.

Approaches that require you to hit problems and then solve them, may provide an inexpensive start but have a high total cost of ownership.

It reflects Kant’s precursor to Deming’s comment - “Experience without theory is blind, but theory without experience is mere intellectual play.”

It's also hard to get management support forof something they don’t understand and intuitively know that its supporters don’t either.

Goldratt also said, “A comfort zone has less to do with control and more to do with knowledge.”

Theory is underrated in much of the Agile space.

## Why we should look at relationships, not domains

We need a finer more useful way of talking about complexity than just what domain we're in. There is always some degree of complexity wherever we are – especially in knowledge work.

Dr. Russell Ackoff gives us insights into how to look at complex systems. He suggests systems are more defined by the relationships between its components rather than the components themselves. Consider a car. If you take the best components of the best cars and put them together you don't have a car. You have a pile of junk.

To relate this to knowledge work – optimizing teams does not provide us with an optimized value stream.

When we look at the relationships between the components of a system, we see that some are clear, some complicated, some complex, some coupled, and some non-linear (meaning a small change can result in a big impact). By attending to these relationships, we can discern the "cause and effect connections [that] are governing the whole system."

## How we can move forward or at least learn as we change how we work

To move forward we must embrace first principles. Here are a few:

1. Delays in workflow, delays in getting feedback, and delays in advantage of lessons learned cause waste.
2. Local improvements don't necessarily result in global improvements.
3. We can complete a subset of a large item faster than the full item. Smaller is typically faster.
4. We only see what we think and talk about.
5. We can't manage what we don't see.
6. People working beyond capacity cause delays that create additional work to be done (waste).
7. Handoffs risk a loss of knowledge and information.
8. People doing the work often have more insights into how it should be done than those managing them.

We can use these to determine if a change to our practices will be an improvement or not.

When considering a new practice or selecting from options, we can ask the following questions:

1. Will this give us greater clarity on what value is to our stakeholders?
2. Will this help us work on the most valuable items to achieve success for our stakeholders?
3. Will this improve the rate at which we get actionable feedback
4. Will this help us build the right product in the right way?
5. Will this help us avoid overloading people with work?
6. Will this lower waiting time in our workflow?
7. Will this improve the ecosystem within which people work?
8. Does this make our work and how we are working more visible?
9. Does this help us learn faster?
10. Does this help motivate everyone?

While these are good questions to ask, we must remember that these are not all of the relationships present. We can't see some relationships and some others are too complex to understand. We must get feedback to ensure we are moving in the right direction.

## The path to understanding is the same as the path to agility

Feedback, feedback, feedback.

Feedback on how we're working.

Feedback on what we're building.

Feedback on what we've built.

By shortening feedback loops across the board, we accomplish several things:

1. We cut out the waste that would be created without it.
2. We improve our workflow.
3. We keep the amount of work in process we have to an effective level which lets us pivot as when we need to.

This enables us to control what we can (our work) while responding to what we can't (what the customer wants, the market needs, and our changing world.

We must not confuse the need for emergence with a lack of predictability. In the same way a sailor can sail upwind by sailing in angles to the wind, we can get to where we need to by taking small steps and correcting as we go. This is also not unlike driving in dense fog. Although we may only be able to see a few feet ahead, we can avoid running into things when we pay attention.

## What we know causes waste

While we may not see all of the relationships present in our system, we do see many of them. And many of the ones we see we know will cause waste. These include:

- overloading teams
- working on bigger items than necessary
- planning too far in advance
- forcing people to work a certain way
- having people who aren't familiar with the work tell people who are familiar with the work how to do it
- disrespecting people
- interrupting teams without attending to the cost of doing so
- delaying getting or using feedback

We can predictably improve our way of working by avoiding these actions.

## The bottom line

We must shift our attention to what we do know and not have our actions be impeded because we don't know everything. In the process we can create a model of understanding and continue building on it.

## 3.6 Capability: Know how to select a more effective practice for your situation.

A fundamental assertion of Amplio Development is that there are no universal practices. We've used this to help decide which practice to use for a particular situation. This starting attitude can also be used to adjust the current set of practices we have if later we see we're having challenges with one of them.

The idea is not to merely change practices when there are challenges, but rather to improve the practices we use as our understanding improves.

### *Why this is important*

Using practices that are not well-suited to our situation has many disadvantages. First, we're not working as well as we can. But just as importantly the team will get frustrated and stop trying to improve. The idea of "do this until you understand and then you can change" is disrespectful as well. Understanding should be with doing.

### *Implementation Methods*

Making changes to practices can be done at any time. Whether you are doing timeboxing or flow does not matter. In both cases, you should have scheduled times to reflect on what you are doing. When using iterations there is a tendency to only reflect at the end of the iteration – this often creates a lost opportunity for improvement.

#### How to improve your practices

Here's a process to use when you are having a challenge with a practice:

Step 1: Ask if you are having challenges with the practice because it is being done poorly? If yes, then consider how you can do it better. If not, continue.

Step 2: Is there something outside of the team that is causing us this problem? If yes, then see how to fix that or at least influence the fixing of it. If not, then continue.

Step 3: Is the team in an ecosystem that is causing problems? That is, are people not collocated when they need to be or are required skills missing? Can you improve on this? If yes, do so. If not, continue to see if another practice that works within this ecosystem will work better (see next step).

Step 4: What else can we do that meets the same objective of the practice? If there is something else you can do, then try that. If not, stick with the practice until you learn more.

#### Using Factors for Effective Value Streams to Detect Challenges

The factors for effective value streams can be used to detect errors in our workflow. Consider how each of them helps us in table 1.

## Factors for Effective Value Streams.

### Purpose

Provides us with a way of looking at complex situations where we can make decisions based on a few factors. Making decisions on how to do our work is complicated. We need a guide to help us see which of two or more options is better. Factors for effective value streams indicate how well we are doing and whether a new way of working will be an improvement.

### *Why*

At first glance the interactions of forces in value streams appear chaotic without a clear pattern. However, it is possible to discover the "*very few elements at the base—the root causes— which through cause-and-effect connections are governing the whole system"* that Goldratt refers to in his theory of inherent simplicity.

### *Factors for Effective Value Streams*

**Time for reflection**

Consider when you were introduced to an organization where your immediate reaction was, "Wow, this place is cool. I can see why they get so much done!"

Now consider when you were introduced to an organization where your immediate reactions were, "Wow, this place is horrible. How do they get anything done?"

You are likely reacting to your tacit knowledge –what you know but are not always consciously aware of. Consider what factors you are looking at. For example, are people talking to each other? How busy are they? These factors are likely present in both situations, but in one direction, they are being done well, and in the other, they are not.

Consider how consistent these tacit judgments are in different places. While each may appear different, a set of actions should be taken that make places easier to work in.

The factors for effective value streams are intended to focus attention on what effective value streams attend to. They are highly correlated with an organization's ability to create and deliver value that leads to the organization's success. By attending to these better decisions can be made on how to work than when they are not attended to.

These factors are stated in the form of questions to ask in order to guide effective change.

***1. Have we identified our key stakeholders, and do we know what success is for them?*** If we've not done this then we'll be working on the wrong things while making it difficult to achieve alignment.

***2. Are we working on the most valuable items to achieve success for stakeholders?*** The items we are working on must be known to be of high value to stakeholders. These should provide value in and of themselves or are small slices of functionality that will provide feedback. We'll learn business artifacts that will assist with this later in this book.

***3. Are we getting actionable feedback from all work being done?*** The purpose of having efficient flow is not merely to avoid waste. Efficient flow enables us to get feedback which is essential to create the right products in a cost-effective manner. We need feedback both to validate what we are attempting to build and the way in which we are building it.

"Feedback is the single most powerful concept for successful projects. Methods that use feedback are successful. Those that do not, seem to fail. Feedback helps you get better control of your project, by providing facts about how things are working in practice. Of course, the presumption is that the feedback is early enough to do some good. This is the main need: rapid feedback." Tom Gilb. Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage

***4. Are we building the right product in the right way?*** Is the quality of the product (both its utility and how it's built) high? Are we managing queues on critical paths? Is the quality of our product slowing us down? Are we having to do rework? Are there a lot of handbacks[1] in the workflow?

**5.** ***Are people being overloaded with too much work?*** Avoid having people working on too many things. Don't overschedule people with work. This is usually achieved with pull methods since the people doing the work have the best knowledge of how much can be worked on without overloading them. One should also attend to the presence of handbacks [1] as they will result in delays in completion.

**6.** ***Have we designed and optimized our processes, with strategies to minimize waiting?*** Attending to team topologies is one way to assist here. Consider delays, handoffs, and handbacks [1] as symptoms of a poorly organized team structure. Attention should be given to keeping people in a minimal number of value streams while ensuring their skills are available if they are scarce. Cross-functional teams can assist here when they are practical and achievable.

**7.** ***Is there attention on managing the ecosystem and not micro-managing the people?*** The system people are in causes most of the problems. It must be managed to improve work. People must also be given the proper information so as to make effective decisions. This requires a shift from people to the ecosystem they are in. Look to see that micromanagement is being avoided. See if people are being given the context within which to make good decisions. The focus should be on improving the environment, not the people. We should be able to trust our people.

***8. Is all work in process visible? This includes how it is being done.*** Can people see work coming their way? Is there an intake process? Do we have agreements with each other about how work is being done? Visibility and common understanding lay the foundation for improving our way of working. Having all work be visible enhances collaboration and creates clarity of the bigger picture.

**9.** ***Is there a focus on cost-effective continuous improvement of the ecosystem, process, and product?*** We must improve how we're working (single loop learning) while challenging our assumptions so we can get better practices (double loop learning) while also learning how to learn faster (triple loop learning).

**10.** ***Are leaders, management, teams, and individuals motivated? People are inherently motivated.*** We must take steps to not demotivate people by fear, misunderstanding, or any other reason.

[1] A handback is when one person hands some work to someone else, and then later they hand it back

They are asked in the form of a question to see the direction of what helps (when answered with yes) and what hurts (when answered with no). The selection of these factors is not sacrosanct, and they can be adjusted as you learn and for your context. The factors have the following characteristics:

1. Each factor is apparent and can lead directly to action that will lead to improvement.
2. The set of factors provides us with enough information to make good decisions for almost all challenges faced by teams doing knowledge work.
3. We've identified a minimal number of factors to provide coverage while keeping things as simple to understand as possible.

In other words, are they of use while providing coverage and not being too complicated?

Doing things poorly will slow us down. But consider that other things are taking place as well. For example, not getting quick feedback will create extra work for us since we'll make a mistake and work on the wrong things and the wrong way. Therefore, attending to the factors for effective value streams can enable us to work without delays *and* not create much extra work we don't need to do.

**Key point**

It is worth considering that there are two different types of waste. The first is work we didn't need to do, for example, overplanning. The other is work we created that we now need to do but only because we created it—for example, fixing bugs.

### *The Factors Work Together*

It is essential to notice how the factors work together. As you improve one, it either improves the others or makes it easier to improve them. For example, notice how working on small items will help us achieve quicker feedback. This is consistent with Dr. Eli Goldratt's (creator of the Theory of Constraints) view that work principles are harmonious. He called this "Inherent Simplicity" in his and his daughter's seminal book "The Choice" and contends that complex problems are more straightforward than they look if one knows where to look.

Let's look at a few of the interactions between these factors:

- Small work items are easier to get feedback on
- Pull methods are the best way to keep workload within capacity
- Having work and workflow be visible enables collaboration
- Working in one value stream makes it easier to keep workloads within capacity

There are of course, many many others. The key point is that they usually enhance each other and virtually never work against each other. They are in harmony.

The factors for effective value streams provide a holistic view by attending to different aspects of a system. This makes it very useful.

### *How we use the factors for effective value streams*

These factors can be used in two ways.  First, they will help identify what we're doing causing our problems. The second way it will be used will be to verify if a new practice being considered to replace a current one will be an improvement. If the change works against these factors, we shouldn't try it.

A summary of the first principles and factors for effective value streams is posted on the next page.

## Scrum Sidebar: Eliminating Scrum's Immutability with Factors for Effective Value Streams

When we have a set of factors for effective value streams we no longer need immutability in Scrum.

Scrum's immutability is required because Scrum provides no guidance on whether a change to a practice will be beneficial or not. This is because it's not based on first principles but on its own theory.

When one understands what causes waste in product development they can make changes to the way they work with high confidence that it will be an improvement. And if an improvement doesn't occur they will learn about some relationship between the components of the system they had not yet understood - improvement or learning.

## Summary of "The Mental Models, First Principles, and Guidance of Amplio"

These pictures are here for your reference and can be freely copied.

# Mental Models, First Principles, and Guidance of Amplio

This is a collection of insights drawn from Flow, Lean, and the Theory of Constraints that Amplio uses. They are useful for any approach.

### Mental Models

- The goal is to achieve success by creating value for the critical stakeholders of the company.
- We can understand what's going on in the complex system of knowledge work.
- We must attend to the value streams.
- Systems thinking is essential and is more about the relationships between the components than the components themselves.
- We must continuously question our methods.

### First principles

First principles of knowledge work are like the laws of what makes knowledge work efficient. First principles stand on their own. They are not defined but are discovered through observation and relentless evaluation. Violating them has consequences, typically creating waste and lost opportunities. They can be used to provide guidance as to what individuals, teams, and organizations should do or avoid. Those listed are Amplio's best discernment of the most useful first principles. Suggestions requested.

They are based on Eli Goldratt's theory of inherent simplicity, which suggests, "If we dive deep enough, we'll find that there are very few elements at the base—the root causes—which, through cause-and-effect connections are governing the whole system."

See https://fs.blog/first-principles/ for their definition.

*Deming- "Experience without theory is expensive."*

***Go to successengineering.works/acop to join the free Amplio Community of practice.***

Success Engineering

### Selected First Principles

1. **Multi-tasking** causes **delays** in workflow which causes waste
2. **Delays** in getting feedback and in taking advantage of lessons learned, cause waste, by not taking corrective action quickly.
3. Most of our challenges are **caused by the system**.
4. People **working beyond capacity cause delays** that create additional work to be done (waste).
5. Working on things of **lesser value to critical stakeholders is waste**.
6. We can complete a subset of a large item faster than the full item. **Smaller is typically faster**.
7. We **only see what we think and talk about**.
8. We **can't manage what we don't see**.
9. **Local improvements** don't necessarily result in **global improvements**.
10. **Handoffs** risk a loss of knowledge and information.

### Guidance

1. Use **systems thinking**. Manage the ecosystem, not the people. Don't blame people when things go wrong.
2. Strive for **quick feedback** to eliminate waste.
3. Keep **workload** within capacity by using pull and having a focus on finishing.
4. Focus on delivering the **greatest value** soonest.
5. Identify your **critical stakeholders'** success criteria and any constraints they are imposing.
6. Attend to the **quality** of the product.
7. Innovate by attending to **the customer journey**.
8. Have an **explicit workflow** to improve collaboration.
9. **Alignment** is more effective than coordination.
10. Frequently reflect on **what** you are building, **how** you are building it, and what your **assumptions** are.

# Factors for Effective Value Streams

The first principles and guidance above provide factors for effective value streams. These factors enable us to discern how effective value streams are and can help us determine whether a proposed change to our work will benefit us. They are stated as actions to achieve.

1. **Getting actionable feedback quickly** on your product and on how you are working.
2. Using pull to **keep workload within capacity**.
3. Working on the **most valuable** items to achieve success for your critical stakeholders.
4. Working in **small increments.**
5. **Managing work in process** at all levels to **reduce delays** in your workflow.
6. **Organizing people** to **avoid multi-tasking** and **delays in people being available.**
7. Understanding the **acceptance criteria** before writing any code.
8. Making all work **visible.**
9. Focusing on the **quality of the product**
10. Having **clarity** on how people **work together**.

# Attending to the Factors for Effective Value Streams

1. ***Getting actionable feedback quickly*** on your product and on how you are working.
Knowledge work is complex, and the challenge of complexity is that it obscures what's going on. Using what we know to get quick feedback provides this visibility and enables us to avoid large mistakes by identifying and addressing small mistakes. Most of the factors for effective value streams involve supporting this key factor.

2. Keeping ***workload within capacity***.
When people are overworked, multi-tasking happens. This lowers their efficiency by 20-40%. People also end up waiting for each other, injecting delays into other people's workflows since they must wait for the overworked person to be available. This ***creates*** additional work. Overloading people makes them less efficient, gives them even more work to do, and makes teams less efficient.

3. Working in ***small increments.***
Eli Goldratt (creator of The Theory of Constraints) once observed, "Often reducing batch size is all it takes to bring a system back into control." One reason is that you get actionable feedback more quickly no small increments, which enables you to pivot with less cost. It also lowers the amount of work to do because the size of the work is smaller.

4. ***Managing work in process*** at all levels to ***reduce delays and handoffs*** in your workflow. This works with "keep workload within capacity." This can be accomplished by gating what comes in and focusing on finishing at the story, feature, and release level. When looking for something to work on, first, see if you can help finish something your team is working on.

5. ***Organizing people*** to ***minimize handoffs and delays*** in the workflow. How people are organized significantly impacts handoffs and delays, which cause waste. Cross-functional teams, when applicable, are the best way to handle this, but there are other ways to organize people.

6. Working on the ***most valuable*** items to achieve success for your stakeholders. This will increase your overall value delivery even if you don't improve your efficiency. This is also a great way to manage work in process.

7. Understanding the ***acceptance criteria*** before writing any code. This is part of what is called the Definition of Done. Before creating something, you need to know what it's expected to look like. The acceptance criteria can be in the form of objectives to meet. They may also change as you learn. And they will help avoid writing code that isn't necessary. Perhaps most importantly, the conversation about the acceptance criteria leads to greater alignment.

8. Making all work ***visible*** so people can see what they need to do. It also enables management to see the progress of work and not require continual status reports.

9. ***Clarity*** about how people work together. Having an explicit workflow is very important. This enables people to understand the agreed-upon best ways of working. It's just a clear understanding of what we have agreed to as the best way of working at any point in time. It is fine to "do what you think is right" at any step.

10. Focusing on the ***quality of the product.*** Product quality is important to make stakeholders happy because poor product quality usually takes more work to modify. It also tends to take more work to support. High product quality is important both for increasing value to the customer and to avoid creating waste.

[1] A handback is when one person hands some work to someone else, and then later they hand it back.

These factors can be used to determine if a new practice would work better than the current one.

The process involves examining the suggested practice and determining which factors will improve if the new practice is adopted. Factors usually improve or don't improve but are not mixed. This is because they work together.

*This means that practitioners can use the factors for effective value streams to be reasonably confident of when a new practice will be better than the one currently used.*

scan for offerings

## Key Points

- We can use each of the factors to see how well we are doing
- The factors for effective value streams provide a way of determining if a change in actions will improve the effectiveness of our value stream

## How We've Manifested the Purpose

We started this chapter by stating, "we need a guide to help us see which of two or more options is better." The Factors for effective value streams ask us questions about how well we are doing. Each question points out whether we follow the first principles of knowledge work.

See how the factors for effective value streams can help tell where improvements in the workflow are called for.

| | Factors for effective value streams (needs updating) | Indicators that they are not being followed |
|---|---|---|
| 1 | ***Get actionable feedback quickly*** *on our product and on how we are working.* | Expect small errors to become big problems because we won't find them quickly.. |
| 2 | *Keep* ***workload within capacity****.* | People will be multi-tasking and for there will be delays in the workflow which will create waste. |
| 3 | *Are we working on the* ***most valuable*** *items to achieve success for our stakeholders?* | Not producing as much value as possible because we're spending time on less important items.. |
| 4 | *Work in* ***small increments?*** | If no, then expect to have higher levels of work in process than necessary and a delay in getting feedback on what value we are producing. |
| 5 | *Have we* ***organized our people*** *to* ***minimize handoffs and delays*** *in the workflow.* | *Requires having people from multiple teams to get the work done. This causes a considerable number of handoffs and delays in our workflow. This significantly slows down getting feedback.* |
| 6 | *Are we* ***managing work in process*** *at all levels to* ***reduce delays and handoffs*** *in our workflow?* | Small errors becoming big problems because we won't find them quickly. |
| 7 | *Do we understand the* ***acceptance criteria*** *before we write any code?* | Doing a considerable amount of rework |
| 8 | *Is all work* ***visible?*** | Having delays in the availability of people required to get the job done. Management being misinformed about what's happening. |
| 9 | *Are we focusing on the* ***quality of the product?*** | Having more work to do than necessary. Both because we'll have to rework what we did and quality problems will require more effort. |
| 10 | *Is how we are* ***working together clear?*** | People working at odds with each other, lowering alignment. |

### How to tell if a change is better - choosing a better practice

Look at the current practice you are doing. Consider another practice that meets the same objectives.

Ask your team which practice will meet the objectives better. Sometimes changing one practice requires the adoption of more than one. For example, timeboxing has several objectives:

- limiting work in process over a period of time
- designating a time to provide a review of what's been done to the customers
- designating a time to meet with product owners to see what's next
- designating a time to do a retrospective
- providing a time to do short term planning

If you decide to not do sprints, you must find practices that give you all of the objectives that go with it. Just not having iterations won't do that.

One way to see if a new practice will be better is to consider how it will likely affect the factors for effective value streams. Discuss each one and use the table below to indicate what the team thinks. Then use the conversations that took place to determine if this looks to be an improvement.

| | Questions to ask based on Factors for effective value streams | Y/N |
|---|---|---|
| 1 | Does/will this improve the rate at which we get ***actionable feedback***? | |
| 2 | Does/will this ***keep workload within capacity***? | |
| 3 | Does/will this help us work on the ***most valuable*** items to achieve success for stakeholders? | |
| 4 | Does/will this help us work in ***small increments***? | |
| 5 | Does/will this improve how our people are ***organized to minimize handoffs and delays in the workflow***? | |
| 6 | Does/will this improve our management of our ***work in process at all levels to reduce delays and handoffs in our workflow***? | |
| 7 | Does/will this better ***understand the acceptance criteria?*** | |
| 8 | Does/will this improve the ***visibility of work***? | |
| 9 | Does/will this improve the ***quality of the product***? | |
| 10 | Does/will this make ***how we're working together clear***? | |

### Symptoms of Not Selecting the Appropriate Practice for Your Context

Not attending to the factors for effective value streams has teams continue to use practices that are not well-suited for them. Look for the team not using the practices they agreed to because there is a sense they are not getting value from them.

# Foundations

## The Amplio Attitude

It may sound funny to say an approach has an attitude, but they all do. Approaches reflect their creators' and sponsors' values and commitments in how they create approaches. It is important to see what they are. Amplio's foundation is based on the following (the "we" in this section refers to the creators of Amplio):

**1. Responsibility and integrity**

**2. Systems thinking and empathy**
**3. A belief that we can understand what's happening**
**4. Humility and seeking knowledge**
**5. Pragmatism**

**1. Responsibility and integrity.** Approaches will always be misunderstood and misused. We take responsibility for when this happens. Responsibility is not blame but represents a commitment for something to be effective. When Amplio is misunderstood, its creators look to see how it can be made clearer. And when it's abused, we'll look to see how we can demonstrate that this is happening. This incorporates having integrity - Amplio needs to benefit its users, even at the expense of potentially lowering training and coaching revenue.

**2. Systems thinking and empathy.** Systems thinking tells us most of the errors that take place are due to the system, not so much due to the people in the system. Furthermore, it tells us that we can't separate the two. This means that frameworks must attend to the content of the framework as well as how people will react to it. This has us committed to its successful adoption by those who choose to use it. And not merely scoffing off its misuse as "well they weren't using it properly." "Responsibility and integrity" coupled with "systems thinking" has us look at how the system can be improved so people use it better as a matter of course. Thus, we incorporate empathy, taking the perspective of its users, into account.

**3. A belief that we can understand what's happening to make progress.** While Amplio incorporates Flow, Lean, the Theory of Constraints, and Human Centered Design, this aspect is based on the idea that even in the complex world of knowledge work we can see what's going on. That the real problem is not understanding, but of getting other people to understand. This is based on insights from Dr. Eli Goldratt's The Choice. Most important is "If we dive deep enough we'll find that there are very few elements at the base - the root causes - which through cause and effect connections are governing the whole system." This does not mean we will see all of these connections, but that they are there and that if we work with what we know we will discover what we don't. He also remarks "A comfort zone has less to do with control and more to do with knowledge." The reason management sometimes resists is that no one explained to them how Agile helps in knowledge work. Or even say we can't understand it because of its complexity.

**4. Humility.** Amplio does not profess to be the only way or even the best way. It builds off what is known. It takes its own tact while acknowledging there is a lot to be learned from other approaches. It acknowledges that, as a metaphor to George Box's comment "all models are wrong, some are useful", all frameworks are wrong, some are useful. We strive to continue to improve Amplio. We understand that others can contribute to this. Having humility is not enough. We must not only respect others, but we must attend to how they approach problems. Without this, any approach will be misunderstood. Part of taking responsibility is to recognize that people bring their own conversations to our new ones.

**5. Pragmatism.** It's easy to create a framework that works if people follow it. But that's like saying "buy low and sell high" in financial investments. Amplio is designed to work in the real world - not in the minds of people wishing it would just be followed as prescribed. Amplio provides options and how to select from them so that people can work in a way that is fit for purpose for this situation and leads to improvement at a pace that is ideal for those adopting it.

### Amplio Sidebar: An Expansion on “A belief that we can understand what’s happening to make progress”

Amplio recognizes that knowledge work is complex. A quick view of Amplio’s approach follows. We’ll go into it in more detail later in the book.

Amplio takes charge in knowledge work with the theories of Dr. Edwards Deming, Dr. Russ Ackoff, Dr. Eli Goldratt, and Dr. George Box.

Deming tells us it’s important to have a combination of theory and practice:

“Experience teaches nothing. in fact there is no experience to record without theory... Without theory there is no learning... And that is their downfall. People copy examples and then wonder what is the trouble. They look at examples and without theory they learn nothing.”

But what theories should we be looking at?

Goldratt tells us "If we dive deep enough we’ll find that there are very few elements at the base - the root causes - which through cause and effect connections are governing the whole system.”

How do we find these elements?

Russ Ackoff tells us that systems are defined by the relationships between a system's components and not the components themselves. In knowledge work, we can find these relationships – the “few elements at the base” - by looking for first principles.

First principles describe how the world works from an objective perspective. First principles stand on their own. They are not defined but are discovered through observation and relentless evaluation. Violating them has consequences, typically creating waste or lost opportunities. They can be used to provide guidance as to what individuals, teams, and organizations should do or avoid. Those listed relate to knowledge work.

Once we find these, we can build a set of questions to see if we’re following them or not. And use these to see how well we’re doing.

George Box, of course, tells us “all models are wrong, some are useful.” It’s important to notice that Goldratt didn’t say these few elements at the base could always be seen, understood, and used for perfect predictability. This is consistent with Ackoff’s view of the relationships in systems – some are too complex for us to see. But this doesn’t stop us from acting on what we can see. And when we end up acting improperly because of a relationship we hadn’t understood due to complexity, we now understand it better. Using Deming’s Plan Do-Study-Act approach and double loop learning (challenging our assumptions) we can improve our model of understanding. And, of course, we remember that our model may be more correct now, but it is still just a model.

By integrating the perspectives of these four people we can move forward while creating a better model of understanding of what’s happening. We don’t have to be limited to mere reaction. We don’t have to play “whac-a-mole” with our work. We can learn and look forward instead of just having to see what has happened. This is theory and experience.

# Foundation

## The core needs

We must recognize the need for the following:

1. We want to be able to take advantage of what's been learned so that teams won't be forced to re-invent known practices.
2. We must incorporate theory to explain why things work. This will enable us to adopt practices that work for our situation since few practices are universal.
3. We must provide team leads with the ability and means to convey these ideas to their teams.

The challenge with accomplishing the above is that the number of practices, theories, and ways of conveying ideas is incredibly large and ever-growing. This puts in a fourth requirement: a manageable set of the above must be able to be used to get teams started and have a way to expand it as needed must also be provided. To be candid, this last requirement was the most difficult one to achieve in creating Amplio.

With this combination of experience, theory, and how to talk about them Amplio can be used to create a simple, appropriate starting point while providing for improvement as the team learns.

### What to align around

The essence of Agile is to provide value to customers in small increments while learning in an incremental fashion. For people to work together, they must see both what they are working on and how they are working. The natural candidate for this is attending to the value stream. Value streams refer to the steps required from start to finish to accomplish something. Value streams can be thought of as comprising:

1. What ideas/requests go into the value stream to be created
2. How the people in the value stream are organized
3. How the workflow of the value stream provides realized value for the customer

This decomposition enables us to work on these aspects of our work in very well-defined ways.

### Different parts of the value stream

While this book focuses on the development part of the value stream, Amplio is defined for other parts of the value stream including portfolio and product management. These will be covered in future books and workshops.

### It's not that simple

Product development is not straightforward. While how to improve our work methods can be based on the cause and effect we see, defining the products we want cannot be. Furthermore, no company is separated from the outside world. Black swan events occur more often than we'd like to think. Amplio is designed to enable companies to be able to pivot with little waste when markets change due to events outside of our control.

## The Design of Amplio Development

Amplio Development takes this approach and applies it to the development part of the value stream. It is designed to meet the requirements stated in the previous chapter.

Amplio Development accomplishes this by:

- Starting with a foundation of first principles based on Flow, Lean, and the Theory of Constraints

- Presenting Factors for effective value streams, which can be used to both identify challenges and evaluate which suggestions for improvement will work best
- Providing a set of capabilities required to be effective. For each capability, Amplio explains why it is essential as well as different options for implementing the capability
- Showing how to create a fit-for-purpose quick start.
- Including critical practices useful for most teams doing knowledge work. These are primarily in management requirements, team formation, and improving workflow.

While this may require more knowledge for coaches to understand, Amplio also provides default starting points that can speed up the process of implementing changes at the start.

## The Mental Models of Amplio

This section presents the mental models, or underlying belief system, of Amplio, of which Amplio Development is a subset. These mental models are fundamental to all Amplio curricula, not just Amplio Development.

If you are not using Amplio, this section is still important to read. The issues discussed in this section are based on first principles that underlie all knowledge work – whether they are acknowledged or not. Learning these can help you as a coach using any framework, method, approach, etc.

*This section is included in both Amplio Development: The Path to Lean-Agile Teams and Being an Effective Value Coach. Regardless of your approach or role, understanding the theories of Flow, Lean, and the Theory of Constraints is essential.*

## Systems Thinking

*"Today's problems come from yesterday's 'solutions.'" Peter Senge, The Fifth Discipline (2006)*

In this section, we discuss the importance of systems thinking. Being able to understand the system in which organizational transformation occurs allows you to anticipate the effects of change and helps you understand how to determine where best to start in a particular context. It also provides a more holistic approach where you consider the entire ecosystem and not just one part of the ecosystem.

Systems thinking is the foundation of flow, Lean, and the Theory of Constraints.

Systems thinking begins with the idea that a system is not merely the sum of its components but rather the system reflects the relationship between them. For example, an automobile is not simply a collection of tires, an engine, a transmission, and so forth on a chassis. It involves the relationship between these components. We cannot take the best parts of the best cars and put them together. That would not be an auto; it would be a pile of junk. What makes an auto is how its parts work together.

Systems thinking tells us that a change to one part of a system can affect other parts. And that can affect even more parts of the system. Changes may even have nonlinear impacts when a small change in one part causes big changes to the whole system (think of the straw that broke the camel's back). Sometimes, the interactions between parts of the system are clear and predictable. Sometimes, they are not. When these interactions are too intricate to understand, we call them "complex interactions" and probably call the overall system "complex.".In complex systems, it is often impossible to predict how changes to one aspect of a system will impact the system as a whole.

Systems can be thought of as a combination of known, well-defined relationships, and those that involve complex interactions. Components can also be tightly coupled to each other (changing one affects others) and potentially nonlinear (a small change in one can cause a big change to the system).

Nonlinear events are when a small action results in a significant impact. The “straw that broke the camel’s back” is an iconic example. While we often hear of disasters occurring due to complexity, we have more control over what is happening in knowledge work. This enables us to decouple the effect of complex events and avoid disaster or wasted effort.

Systems thinking also tells us that it is usually the system and not people that cause errors and waste. It’s easy to attribute bad behavior to individuals, but it’s more likely that the system they are in is causing the problems. In fact, systems impact human behavior.

## An important distinction

Creating new products is an emergent process. It requires adaptation, and we must acknowledge the lack of predictability. But this does not mean that our work methods can't be well-defined.

## Essential lessons of systems thinking

- Systems are more about the relationships between their components than the components themselves
- Systems are comprised of other systems and interact with systems outside of themselves
- Local changes to a system cause global, sometimes unpredictable, side effects.
- Most of the challenges we encounter are due to our system.

## Systems thinking in knowledge work.

In systems thinking, we must look at all aspects of knowledge work. This means we look at the people, the customers, the other stakeholders, management, and the workflow. Everything.  One aspect is not more important than the other because all of these are intertwined. This has often been ignored in the Agile community with its focus on people. People are, of course, critical. But the system within which people work affects their behavior. In Amplio, it may appear that we are over-emphasizing production mechanics. But Amplio believes a balance between all these factors must be achieved. Any failure in one factor will likely lower the effectiveness of the system.

## The Importance of Systems Thinking

A Systems Thinking analysis has many implications:

1. Most of the errors are due to the system, not the individual people in the system.
2. Whatever approach you take, you must consider how people will react to it. This is particularly true if people consistently misunderstand the approach, misuse it, or can’t get it to work on a regular basis.
3. Common challenges should be considered a direct result of the way the approach is designed. The approach should be modified to minimize these.
4. Changes made must be done in the context of the whole system.
5. Optimizing a part of the system without attending to how it will affect the entire system, it is likely that a sub-optimization that actually hurts the overall system will result.

Additional Resource: What if Russ Ackoff Gave a Ted Talk

# Systems Thinking, Context, and Patterns

In the realm of problem-solving and decision-making, two essential concepts emerge: system thinking and context. These concepts are intertwined, complementing each other to drive success and they are not the same concept.

**System Thinking**
When we discuss the big picture, system thinking is the powerful model that encourages a holistic perspective. It involves understanding the **interconnectedness and interdependencies within a system**, recognizing that actions and changes in one part can ripple across the entire system. By analyzing the system as a whole, system thinking enables us to grasp the underlying structures, patterns, and feedback loops that influence outcomes.

System thinking provides a valuable lens for problem-solving, as it goes beyond isolated events and focuses on the broader context. It encourages us to consider the various elements within the system and their relationships. This approach **helps uncover hidden causes and unintended consequences**, facilitating more effective and sustainable solutions. It also **promotes collaboration and cross-functional understanding**, as stakeholders gain insights into how their roles and actions impact the larger system.

**Context**
Context refers to the specific circumstances, conditions, and environment in which a system operates. It encompasses the unique set of factors and influences that shape a situation. Understanding the context is crucial for making informed decisions, as it provides the necessary **background and constraints.**

Context helps us navigate the complexities of a particular situation by considering factors such as culture, history, regulations, size and age of an organization, leadership styles, market dynamics, and stakeholder expectations.From an individual level, people can have a context based on how they see the world and how they see themselves. **It ensures that our actions and solutions are relevant, appropriate, and aligned with the specific context in which they will be implemented. It is where we stop the one-size-fits-all solutions by appreciating the context.**

System thinking and context are not mutually exclusive; rather, they are mutually reinforcing. The interplay between the two is vital for achieving optimal outcomes. System thinking offers a framework for understanding the underlying dynamics and relationships within a system. It helps us identify leverage points, where small interventions can have significant impact. By considering the broader system, we can avoid unintended consequences and create more effective, systemic solutions. At the same time, context provides grounding and specificity to our analysis and decision-making. It ensures that our solutions are tailored to the **unique circumstances and constraints of a given situation**. By considering context, we can adapt system thinking principles to address real-world complexities and nuances. To harness the full power of system thinking, we must apply it within the context that it seeks to address. **Without context, system thinking risks becoming an abstract exercise detached from the realities on the ground. Similarly, without system thinking, contextual analysis may overlook systemic interdependencies and fail to address root causes comprehensively.** Therefore, success lies in striking a balance between system thinking and context. By leveraging the insights gained from system thinking while considering the nuances of the specific context, we can develop robust, **contextually appropriate solutions that drive meaningful and sustainable change.**

In conclusion, system thinking and context are not opposing concepts; rather, they are intertwined and mutually beneficial. Embracing system thinking within the context allows us to understand the big picture, identify patterns, and develop systemic solutions. Simultaneously, considering the context

ensures that our solutions are grounded, relevant to the specific circumstances, and aligned with stakeholder needs. By recognizing and leveraging the dynamic interplay between system thinking and context, we can navigate complexities, drive positive change, and achieve success. Recognizing patterns in systems and contexts allows us to be very efficient in driving positive change.

## Why Systems Thinking Is so Essential To Learning Quickly

One of the biggest insights Deming gives us with this systems thinking approach is that the overwhelming majority of the errors are due to the system people are working in and not the people themselves.

> “A bad system will beat a good person every time. “
>
> “85% of the reasons for failure are deficiencies in the systems and process rather than the employee.”

Systems thinking also tells us if we have similar systems we’ll get similar behaviors.

Learning can be accelerated therefore by looking at systems similar to ours and observing what’s happening.

As Will Rogers remarked:

> "There are three kinds of men. The one that learns by reading. The few who learn by observation. The rest of them have to pee on the electric fence for themselves."

The lack of systems thinking in Agile is very costly. People don't look for common cause errors which are what most people face. Instead, they have to learn what others have learned before them. This is a huge waste and should be eliminated.

# PART III: PREPARING FOR THE GPS

# Understand Your Problems Before Offering Solutions.

While it's tempting to act like an expert and offer an immediate solution, real experts never do this. There is, unfortunately, no "one-size-fits-all" solution to any complex problem. They first look to see what the problem is. Then, *why* it's a problem and only then look for a solution. If we use a framework with predetermined practices, we must see how we can substitute for them.

This approach should be undertaken at all scales - from the individual to the organization. At scale, we must attend to an organization's culture and current state before large-scale adoption. Taking a quick look to see what's needed and adjusting for that does not take much additional time and is more than worth the investment. This is not to say that we must make custom solutions for each organization. There are well-established patterns of solutions to this set of common challenges in any event.

Adopting a fit for purpose solution not only creates a better result, it will likely reduce resistance that it would otherwise face. This is not compromise - this is creating effective solutions.

Although the title of this part says "problems," it's worth noting that a better way to think about this is overcoming "challenges." This is not just "Pollyanna" or saying, "opportunities exist in problems." Instead, it's essential to understand what problems are. We often think of them as real things. For example, most people would consider a flashing blue light from a police car behind them a problem. And if you don't need a police car stopping you, it probably is. But the same blue light could be good if you're driving to the hospital and about to run out of gas. What's the difference? It's what you're trying to accomplish at the time. In the first case, you want to go somewhere without interruption (or getting a fine). In the second, you also want to go somewhere but have another problem (or should I say 'challenge') of running out of gas.

This distinction is important because behind every apparent 'problem' is something you're trying to do – and you're having a challenge in getting it done. Thinking of it as a challenge reminds you of what you're trying to accomplish – and you might find there is some other way of achieving that. Having alternatives to achieve your desired result provides options you can pick by attending to your situation.

Having pre-set solutions to problems misses opportunities to understand better ways to deal with what we want to accomplish.

# PART IV: ORGANIZATION PATTERNS

# Structure of the capabilities and Patterns

*Objective*

*Why this is important*

*Capabilities useful to do before this one*

*Capabilities depending on this one*

*Symptoms of not having this capability*

*Anti-Pattern*

*Story about the capability*

## Pattern(s) for this capability

*Context*

*Challenge being solved*

*Forces and how they relate to the context*

*Solution*

*Potential implementations (risks of the implementations)*

*Examples - show diverse when context varies (see if these are too diverse, if so, split out)*

# Success

## Capability SU1: Identify stakeholders and what success means

### *Objective*

Understand who your critical stakeholders are.

There are three different types of stakeholders

1. **Sponsoring stakeholders** (those people funding the development of the story)
2. **Value consuming stakeholders** (could be users, purchasers, those paying for what has been created, …)
3. **Constraining stakeholders** (government agencies, tools, corporate agreements, …)

Value consuming stakeholders may be internal or external to the company.
This distinction is important than just having customers as a heading.

Not all stakeholders can be considered to be customers. For example, the FAA is not a customer of an airline.

### *Why this is important*

Knowing who your sponsoring stakeholders are is important when it comes to decide how to sequence the work.

Knowing the value consuming stakeholders is important when it comes to how to decide what is of value.

Knowing who constraining stakeholders is important for compliance issues - both internal and external.

### *Capabilities useful to do before this one*

none.

### *Capabilities depending on this one*

Any capability that can be affected by the prioritization of what's being worked on depends upon this directly or indirectly.

### *Symptoms of not having this capability*

Inability to sequence the work to be done.

### *Anti-Pattern*

### *Story about the capability*

## Capability SU2: Strategic Pillars

Job to be done: Identify what we want to invest in. These will be the basis for our strategies.

The purpose of this is to align different initiatives across the organization. Without this people will believe that their projects are more important than others.

These are typically unique to a business but similar across organizations in the same business. Initiatives typically derive from these strategic pillars and are then implemented in quick increments of value.

For example, a financial company might focus on:

- Retaining assets (because they make money based on a percentage of assets)
- Improving customer experience
- Lowering costs
- Compliance
- Lowering risk

They might also include one for improving internal processes.

A not-for-profit that provides meals and housing for homeless may focus on:

- Number of meals served
- Number of beds provided
- Amount of  monetary donations received
- Value of non-monetary donations received
- Number of volunteers

**How many do you need?** Typically 4-6 items for measure are good enough. Less than four does not provide enough clarity, while more than six is not necessary (the seventh can provide at most 15% overall value).  These areas of investment are just guides but create focus when decisions need to be made. They become the basis for an organization's initiatives where we can use them to define the value we want to deliver.

## Capability SU3: Business architecture

# Strategy and Budgeting

## Capability ST1: Strategies

## Capability ST2: Agile budgeting

## Capability ST3: Lean Leadership and Management

This capability will be described here.

LinkedInPosts on Management

- /begin rant on anti-management and we can't know what to do because we're in a complex environment/
- It's real simple. Expect troubles from management if you're not doing the following:
- Why are you surprised when management wants full utilization?
- How to talk to management.
- Bad management is not an incurable disease.
- The biggest challenge to good Agile in general and good Scrum in particular, is management pushing work onto the teams.

If you are in Amplio University, see the following sessions on management:

- 25_01_14 Overview of the management role.

add lucid board on management

Capability ST4: Enterprise architecture

Capability ST5: Initiatives

# Product Management

Amplio Development is not designed as just a guide but as a learning method. Each of the capabilities is organized to provide both why an action is required as well as different options to implement it. This is based on the concept that learning both a practice and the principle on which it is based is the best way to learn how to improve. These principles affect many actions, so as they are learned for one practice, this learning can help improve other practices.

## Purpose of this chapter

This chapter prepares the reader for how and why the practices suggested in the rest of the book are organized. There is a “chicken and egg” problem that needs resolving. Most people want a quick starting point. However, using a predetermined way for a team to do their work will likely not result in an approach that is fit for purpose. Amplio Development solves this dilemma by enabling participants to select a quick-start execution from a set of patterns based on a few key issues. This starting point is to be considered the best the team knows how to work at the moment.

As the team works, it can improve its practices by considering other ways of working, evaluating them based on the factors for effective value streams, and trying them out and seeing if they work better.

## Describing the Capabilities and the Practices

It is not enough to have an approach explain what to do, the approach must also provide a way of learning what to do. Although you may be new to Agile methods, if you’ve been doing knowledge work for any length of time you have experience in the first principles – whether you were conscious of them or not. We will describe the capabilities you need to have by first looking to see what objectives must be met. It must also have a way for you to see what happens when you aren't doing what's needed. Therefore, we include symptoms when the objectives aren't being met.

By presenting the principles with a few practices that will meet the objectives, people can learn the principles being used based on their experience.

### Capabilities as what to aim for provides us with a path and warns us of risks.

Unlike other frameworks, Amplio’s capabilities are not presented as practices to follow but as objectives to aim for. Many of the capabilities have virtual collaboration boards that provide guidance on which particular way to implement the capability.

Each capability provides a way to score how well you’re doing. By having a current state and a target state, Amplio provides a set of steps to take you from where you are to where you want to be. Not manifesting each capability incurs a risk which is described as an anti-pattern.

## Organization of the Capabilities

Almost 30 capabilities have been identified as useful to most teams. Teams don’t need to start with all of them. Those capabilities that should be used at the start are identified as “core” capabilities.

Amplio Development's capabilities are organized as follows:

1. Requirements and Artifacts
2. Managing the workflow
3. Quality of products considerations
4. Organizing teams
5. Learning, improving, and pivoting
6. Roles

The book has a chapter for each of these sets of capabilities.

## Scrum Sidebar: Amplio has a different attitude from Scrum.

Amplio doesn't focus on being "purposefully incomplete." It focuses on being "purposefully sufficient." It focuses on providing adopters the information they need, when they need it, without overloading them.

Amplio doesn't find it acceptable that something is "simple to understand but difficult to master." Although building a product may be difficult to master, the approach should be simple to understand and relatively easy to master.

While you do need to learn the rules of the game, you need to make sure you're playing the right game. And you always want to understand why the game's rules are there. This contrasts with many Scrum proponents saying, "Scrum is like chess; follow the rules first, then you can change things." See this sidebar, Scrum is like chess, for more.

Amplio does not require any particular practice to be used. The theory presented in Amplio allows you to improve on the practices provided without the danger of making your workflow worse.

## Scrum Sidebar: What Amplio adds to and removes from Scrum

**What's been added to Scrum**

- A foundation of systems thinking, Flow, Lean, and the Theory of Constraints
- The factors for effective value streams can be used to predict if changing a practice will likely be an improvement or not. We often don't need to run experiments to see if our actions will be an improvement.
- Insights necessary for practitioners to extend the starting set of practices as needed.
- Objectives for each practice include why it's important and -patterns when they are not met
- A recognition that there are no universal practices but that there are first principles
- Options for organizing teams when cross-functional teams are not applicable
- Be able to use a timeboxing or flow model, not merely flow within timeboxing
- A focus on minimum incremental value delivery
- Some core technical practices when software is involved

- DevOps

**What's been removed from Scrum**

- There are no immutable roles, events, artifacts, or rules in Amplio Development. Anything that is consistent with first principles is ok. You select your practices based on what works according to your context and the laws of product development.
- Implicit distrust of management. While not technically part of the Scrum Guide this distrust resides in the Scrum culture
- The idea that understanding only comes after doing. The two can be learned concurrently.

## Scrum Sidebar: Six benefits of Amplio not being immutable

There are several things that may vary with an Agile adoption. If your approach can't handle these variations expect to have confusion and the need for retraining.

1. Follow a particular method or build your own. Amplio has a set of start Scrum and Kanban practices if you want to just start with Scrum or Kanban. Or, just build your own.
2. Starting practices. If these aren't fit for purpose more energy than is needed will need to be spent and the transition may fail. It is better to lower the motivation required than to complain about people not being motivated.
3. Different teams may need different practices. It's important not to force every team to use the same method
4. Things change over time. At the beginning you might want to use Scrum, then Kanban. You can't afford to get trained every time this needs to change.
5. What you thought you needed may change after you get started. Selecting Scrum or Kanban at the start entails making several decisions you may not even realize you're making.
6. Some teams have different constraints than others. For example, the need to coordinate with hardware teams and government regulation. Amplio allows plug-and-playing additional workflow steps. It also allows removing steps when they aren't necessary. This keeps it from getting too complicated.

**Immutability does not make things simpler or less costly.**
I sometimes imagine that the justification for immutability is that it is easier to teach something when you have it. However, using better training methods than the standard half and full day training allows for getting much deeper and more effective training at a lower cost.

Supplying virtual collaboration boards (e.g., Lucid Sparks) to help teams collaborate with each other can also make it easier to make decisions.

Don't buy into "you have to follow until you understand" or "Scrum is like playing chess" (chess needs rules to play a game and have fun, you need guidance to help you work better). Immutability is a poor design. See this sidebar, Scrum is like chess, for more.

The bottom line is immutability is a symptom of a poor design and is only needed because so many people use Scrum improperly.

Get your Scrum Masters to be able to continue what they've done with Scrum but teach them how to go beyond it to what you need.

## Amplio Development Capabilities (Objectives)

Note: How to read this section.

This section is both an overview of what needs to happen and a reference guide. It is suggested that one first read each section's opening paragraphs – e.g., Requirements and Artifacts, Quality of the Product…. Then one can go back and read each objective. Include reading the details of the objective if desired at this time or wait until you come back and read it as a reference.

Each capability is presented as follows:

- Objective
- Why it's important
- Options on how to accomplish it
- symptoms if you don't

While each of the capabilities is presented in its entirety, feel free just to read the objectives when reading them the first time. Later, you can read through the appropriate objectives more carefully when you want to solve challenges.

These capabilities were not decided via theory but by observing successful and less successful teams over almost two decades. Patterns of behavior were observed and recorded. While practices were significantly different, based on the context of the teams, the objectives achieved by successful teams were surprisingly similar.

### A Case Study

People all too often assume they know what their problem is. Sometimes, however, the source of a problem is not where it is showing up.

Read Using Value Stream Mapping and 'Five Whys' to Get to Root Cause for an example of this.

## The Amplio Team Approach, Challenges, and Related Practices Graphically

Figure 1 illustrates the generic Amplio Team approach.

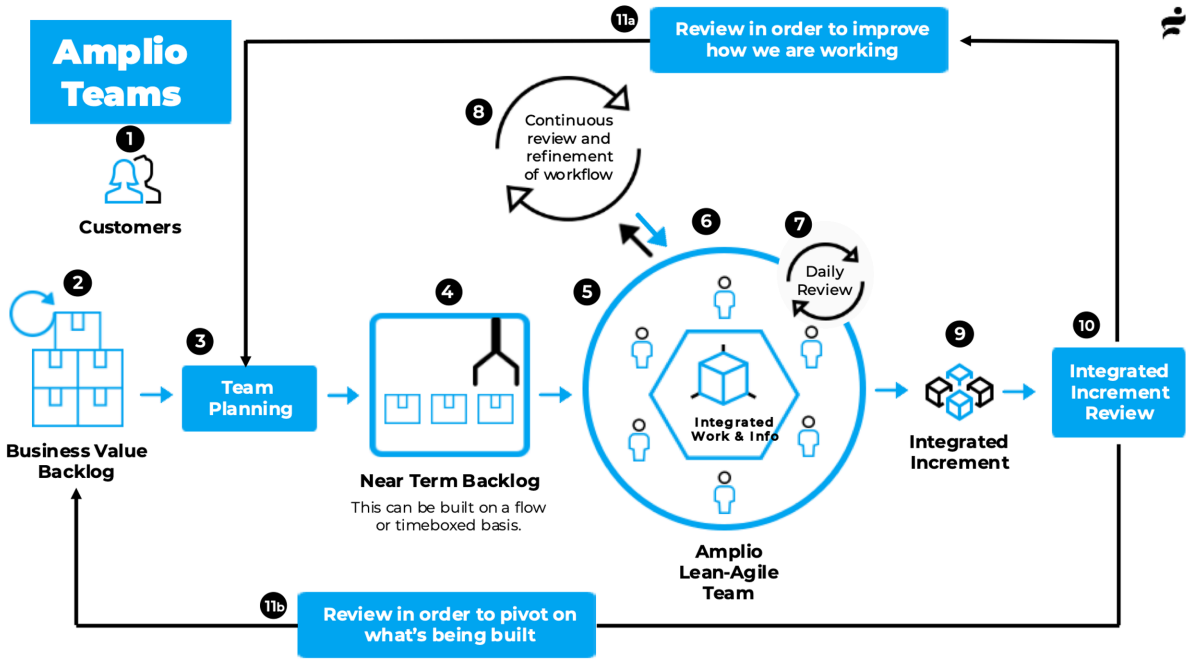



Figure 1. Generic Amplio Team Approach

## The Key Points of Amplio Development

**Value stream management provides valuable insights for a Lean-Agile Team.** By taking a value stream perspective, all the steps and components of Amplio Development fit into place and describe the relationship to each other. Amplio teams take an end-to-end view of their customers' value streams, from taking a concept (idea) through the processes and activities necessary to deliver the intended value. The goal is to discover and remove the constraints and wastes that impede the delivery capabilities of the targeted value stream.

The primary role of an Amplio team coach is to create and sustain a value-delivery-oriented focus, removing the impediments that arise from organizations organized and siloed around business domains or functions.

**1. Use the customer journey to improve the customers' operational value stream.** How customers interact with your services and systems is called the customer journey. The entire workflow of the customer is called the customer's operational value stream. By attending to the customer journey, we can positively influence the customers' operational value streams and provide value to them.

**2. Have a business value backlog focused on our products and services.** This backlog uses minimum viable products (MVPs) when we need to discover if a new product will be valuable and Quickest Valuable Releases (QVRs) when enhancing existing products. Both focus on delivering the highest value in the shortest amount of time. This business focus also provides the opportunity to create a **test plan** based on Acceptance Test-Driven Development or Behavior Driven Development.

**3. Amplio Team Planning.** Amplio team members work together to create a plan for future work. It can be timeboxed planning or flow-based planning. If timeboxed, the iteration length will be no more than two weeks. The result is a sequence of slices for the teams to pull from and build together.

**4. A near-term backlog.** This backlog contains the items the team will work on over the next week or two.

**5. The Amplio Team pulls work from the near-term backlog to improve flow and value delivery.** *Planning, acceptance criteria, development, test, review, and delivery* activities operate at an accelerated pace to create realizable value. The coordinated pull of items from the Product and Iteration Backlogs as capacity becomes available, not pushed as batches of work items into the development streams, reduces waste while facilitating an alignment of teams.

**6. The Amplio Team value-creation structure and workflow.** The Amplio Team is hopefully cross-functional. But cross-functional teams are not always achievable or even desirable if some individuals have specific skills that are costly to replicate.

**7. Daily review.** Short, pre-scheduled meetings to review where we are. The intent is also to enhance a sense of the team. The team walks the board stating, for each appropriate item: 1. Here's what we're working on, 2. Here's where we are blocked, 3. Here's what we're going to work on next

**8. Continuous review and refinement of the backlog.** Members across the team need to coordinate the following: 1. the test plan for non-functional requirements, 2. how to integrate across members, 3. ensure a coordinated workflow takes place, and 4. attending to how the individuals are working together.

**9. Integrated Increment.** These increments can be produced on a cadence basis or as MVPs or QVRs are created

**10. Review.** While it's best if customers frequently review the work being done, each integrated increment must be reviewed when released.

**11a. Review to improve how we are working.** Improving how we are working is a continuous process. In addition, Agile Retrospectives are planned events to have teams evaluate and implement short-term improvement opportunities. Amplio teams also look at how they can deliver a series of QVRs that incrementally improve their customer's value stream delivery capabilities from the Lean-oriented perspective of improving work and information flows. In addition, Amplio incorporates Deming's Plan-Do-Study-Act cycle.

**11b. Review to pivot on what's being built.** Use the feedback from the integrated increment review to adjust the business value backlog.

## Common Challenges of Agile Teams

Figure 4 shows the common challenges of Agile teams.

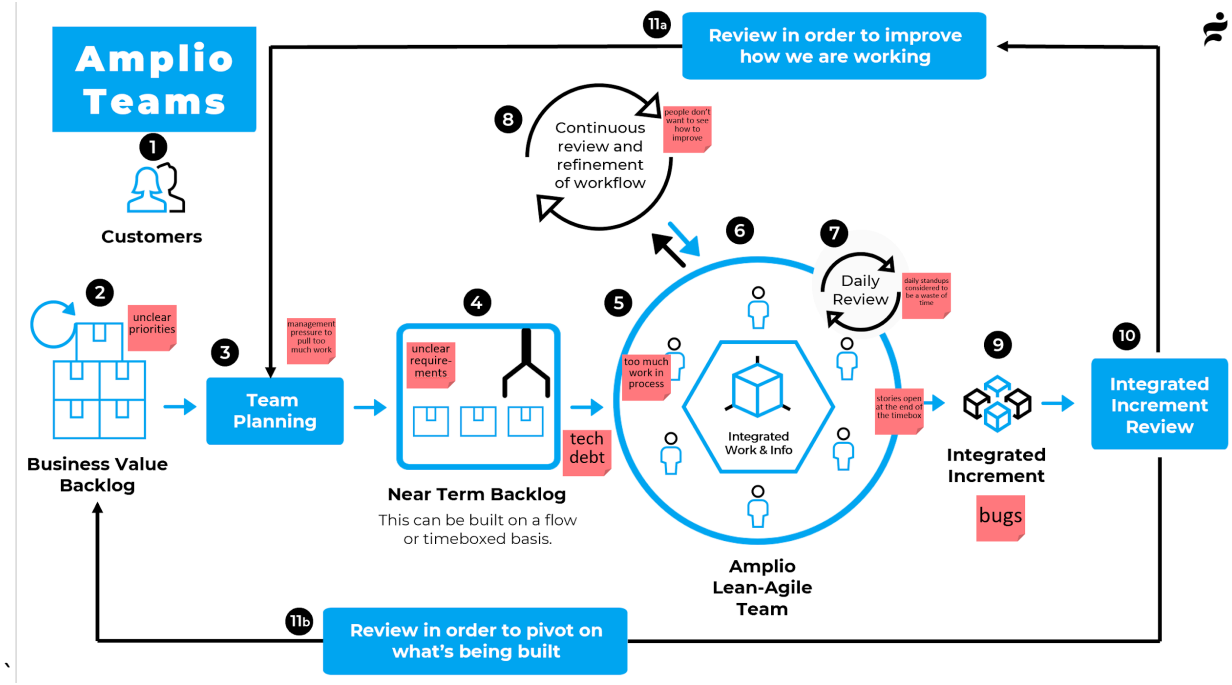



Figure 2. Common challenges of an Agile Team

## Scrum Sidebar: Scrum Vs Amplio-Team: An immutable framework Vs a Decision-Guiding Platform

Both Scrum and Amplio-Team have the same objective – effective, collaborative teams creating value for their customers.

The difference is in their foundation. Amplio extends Scrum by providing principles that can be used to tell what the source of your challenges are as well as whether a change will be useful or not. It also provides a dozen templates to help you make effective decisions based on both these principles and the experiences of others.

While Amplio-Team gives you the opportunity for as easy a start as Scrum, it also gives you the ability to tailor it to your needs. Our goal is not to follow Scrum in the hope it helps us meet our objectives, it's to be clear about our objectives and use whatever methods we can to help us reach them. Adjusting our methods effectively can make it easier and therefore less likely that our teams will fall back into bad practices.

Amplio-Team also provides methods for multiple teams to work together while enabling teams to work in a way that works best for them. It is rare that teams are truly autonomous and cross-functional. Being organized in less-than-ideal ways due to the reality of your

situation can make Scrum difficult to adjust. In fact, Scrum advises against it because it provides no guidance once you modify it.

Amplio doesn't require retraining even if you shift from Scrum to Kanban. Amplio provides the principles and theories underlying both Scrum and Kanban. As your needs change you just make necessary adjustments. It's the difference between hiring a cook who knows a few recipes and hiring one who knows how to use the ingredients they have. This also enables your teams to deal with constraints unique to them such as government regulations.

The bottom line is instead of forcing you to fit into Scrum, Amplio looks at what is the best way to achieve your objectives required to have effective teams. Making things harder because a framework is immutable is not good Agile. Finding the most effective path forward is.

## The Amplio Team Patterns

There is no end to the number of patterns teams can find useful. The ones listed here are those we've found usually essential to be effective. These, therefore, are intended to be a guide to start with. They are categorized as follows.

| Capabilities. VCB means a virtual collaboration board is available to assist in either doing or deciding how to do this. | Amplio Team | Scrum |
|---|---|---|
| 1. Requirements and Artifacts | | Increment |
| 1.1 Identify stakeholders and what success means to them. VCB | | |
| 1.2 Work on small, releasable items of value. VCB | | |
| 1.3 Ensure you're creating the greatest value. VCB | x | |
| 1.4 Have high product quality from the customers' perspective. | | Suggested |
| 1.5 Validate the requirements with examples. | x | |
| 1.6 Attend to the customer journey while creating requirements. | | |
| 1.7 Have definitions of ready (DoR) and definitions of done (DoD) with full kitting VCB | | Just DoD |
| 2. Managing the workflow | | |
| *2.1 Create visibility of work and workflow* | | Suggested |
| 2.2 Manage work in process to remove delays in workflow and to lower risk | | Sprint |
| 2.3 Have a product backlog serve as an intake process | | Specified |
| 2.4 Have a near-term backlog from which you pull your work | | Specified |
| 2.5 Use pull methods to keep workload within capacity | | |
| 2.6 Build in small, vertical, slices. VCB | | |
| 2.7 Agree on classes of service and their corresponding service agreements VCB | | |
| 2.8 Use DevOps when applicable | | |

| | | |
|---|---|---|
| 2.9 Use automated testing to eliminate waste | | |
| 3. Learning, Improving, and Pivoting | | |
| 3.1 Prepare for the start of a project or for the start of building a product | | |
| 3.2 Create value in small steps to get quick feedback | | |
| ***3.3*** *Attend to risk with feedback VCB* | | Sprint review |
| ***3.4*** *Have an agreement on discussing challenges on a frequent basis VCB* | | Daily Scrum |
| **3.5** Frequently step back and reflect VCB | | Retrospective |
| 3.6 Know how to select a more appropriate practice | | Denied |
| 4. Roles | | |
| 4.1 Decide who is the Product Owner in Scrum VCB | | Product Owner |
| 4.2 Developers | | Developers |
| 4.3 Decide on the responsibility of the team coach VCB | | Scrum Master |
| 4.4 The role of management | | |
| 5. Organizing teams and backlogs for effective value streams | | |
| 5.1 Have an effective value Creation Structure VCB | | Cross-functional |
| 5.2 Shared backlogs | | |
| 5.3 Aligning market solutions to teams | | |

Each capability is written up as a pattern with the following structure:

- Name – to identify it
- Objective (the What)
- Why this is important
- The problem to overcome
- Different ways to implement the solution (the How)
- Forces or issues. These must be attended to to determine the best way to implement the solution.

## Scrum sidebar: Purposefully sufficient Vs. Purposefully Incomplete

Amplio identifies 30 capabilities that are useful for most Agile teams.

For many of them, a decision-guiding table is available (DTA). These decision tables don't tell you what to do, but rather what to look at to make good decisions. More waste is created by

not attending to what's been figured out than by missing some things that have been obscured by complexity.

A few other capabilities (more being added) provide a Lucid Sparks board to help track information.

Amplio does not prescribe anything, but rather provides guidance on how to work. Scrum provides little guidance except to make decisions for you (specified).

This specification of a few things while ignoring several necessary things results in having to relearn many well-known methods. As Edgar Schein says "we don't think and talk about what we see, we see what we are able to think and talk about."

Constraints are useful when they focus you on what to work on and prevent you from wasting time on things that aren't useful. But Scrum's constraints, in the form of pre-adoption decisions, constrain you from using better practices.

The immutability of Scrum also discourages innovation and tends to create a "just following mentality." Ironically, what many Scrum aficionados disparage is created by the way Scrum is designed.

If you want an out-of-the-box solution, merely set the Amplio decision tables to what Scrum would have you do. Then, if they don't work so well you can adjust them.

If you have a Team ACE, or take the Amplio Development Master class, or possibly even just read the book on which it is based, you can have your starting approach tuned right up front.

# 1. Requirements and Artifacts

How we organize our requirements is essential. Waterfall didn’t have us just work on things in a (falsely) predictable manner, it had people work on large batches. While MVPs, a la Eric Reis and the Lean Startup, have focused us on smaller items, there are still gaps in how Agile artifacts are created and used. This chapter introduces the Quickest Valuable Release which is an umbrella term for what’s the next value that can be delivered. It encompasses both small increments when challenging assumptions when we create new products and what’s likely larger increments when we extend existing products.

This section starts off with  why we need to go small and conceptually what a Quickest Valuable Release is. It then goes into detail on the following capabilities:

1.1 Identify critical stakeholders and what success means to them as well as what their constraints are.
1.2  Work on small, releasable items of value. Introducing the Quickest Valuable Release
1.3 Ensure you’re creating the greatest value.
1.4 Have high product quality from the customers’ perspective.
1.5 Validate the requirements with examples.

Software Engineering Director,1.6 Attend to the customer journey while creating requirements
1.7 Have definitions of ready (DoR) and definitions of done (DoD) with full kitting.

# Introducing The Quickest Valuable Release (QVR) - an artifact based on Lean thinking.

As we've discussed, Lean is about delivering value quickly by eliminating delays in our workflows. This enables quick feedback to ensure we're working on the right thing to create the most value. Eliminating delays and responding to feedback enables us to eliminate waste. A side effect is also higher quality.

A QVR is:

*The smallest set of work that can be delivered quickly, fully prepared for stakeholder consumption, and provides meaningful, validated value.*

A QVR is not just about building and releasing — it **ensures** that:

- Stakeholders can **immediately understand**, **adopt**, and **gain value** from it.
- The release is **fully kitted**. This means it includes all necessary **training, communication, documentation, onboarding, and support materials** needed to realize the value.

**Key Aspects:**

- **Quick delivery** — accelerate time-to-value.
- **Full-kitted** — packaged so stakeholders can *actually use it* right away.
- **Focused on stakeholder value** — aligned tightly to what matters most.
- **Designed for validation and learning** — closes the feedback loop fast.

*QVRs are not about delivering less, but about delivering value sooner.*

## Why We Want to Go Small

"Often reducing batch size is all it takes to bring a system back into control." Eli Goldratt

"It's now how much we produce. It's how much value is created." Al Shalloway

Consider what we've learned learned from Amplio Foundations:

- ➢ The larger the items we're working on the longer it takes them to get done.
- ➢ We want to deliver value quickly. The slower the delivery the higher our cost of delay.
- ➢ We want to get feedback quickly. The longer the feedback the more waste we're likely to create.

Having smaller items directly impacts value delivery and getting feedback quickly.

However, there are many more reasons for the items we're working on to be small:

- ➢ The larger the item being worked on the greater the number of skills required to build it is likely to be. This may result in more people than can fit on a small team being needed.
- ➢ When we're comparing the value of objects A and B, the larger they become the more likely one part of A will be more important than some part of B and vice versa. This makes it harder to prioritize the two since some parts of each are more important than some parts of the other.
- ➢ The larger the items a team is working on the longer until another item can be injected into the teams workflow without causing an interruption to their work. Interruptions typically create waste.

- ➢ The larger an item is the greater the likelihood that a wider range of skills will be necessary both to build it and to get it properly deployed. This means that as the size of what's being worked on increases, the greater the number of teams involved is likely to increase as well.

The desire to work on small chunks of deliverable value does not mean we want to deliver less. Rather we want to be able to deliver sooner while creating less waste in the process.

## QVRs should be used in both iteration and flow systems

### *Using QVRs in iteration (e.g., sprint) systems*

Iteration systems such as Scrum, SAFe, and Scrum@Scale have a tendency to hold up delivery until the end of their iteration.. While this doesn't need to happen one must take effort that it doesn't as the focus is on the iteration (e.g., Sprint) goal.

### *Using QVRs in flow systems*

Flow based systems typically have us working on the most important QVR(s) at one time and releasing them as quickly as we can. This increases our focus on creating something releasable and not holding it until the end of a span of time.

# Capability: Work on small, releasable items of value

### ***Objective***

Work on small items for which value can be realized.

Have artifacts that can be used to track these items.

### *Why this is important*

Working on small items enables quicker delivery of value. Working on large items not only delays feedback and delays delivery of value, doing so typically involves requiring more people to coordinate.

High-value items can be used to align work to business value. If this is not present people will not coordinate what they are working on to get value out the door quickly.

Focusing on releasable items creates a bigger picture than just completing the story in front of a developer and facilitates cooperation.

### *Symptoms of Not Doing This Well*

Larger than necessary items cause delays, poor feedback, and too much work in process. When this happens, quality goes down and waste goes up.

### *Implementation Methods*

Let's first consider the type of knowledge work being done that has a large impact on making these decisions. When:

- doing new product development, MVPs should be used. This is also an indication to use flow as well.
- enhancing an existing product, we should use Quickest Valuable Releases (QVRs).
- doing mostly maintenance work, requirements are often minimal already and an equivalent to stories is good enough. This is an indicator for using flow unless there is a large backlog to choose from.

The type of work you have to do is one of the key factors in deciding whether to use timeboxing or flow. If the product being built is not well-known, continuous demonstration, validation, and pivoting may be necessary. If we're creating a new product with MVPs, then it is likely that flow should be used. When the speed of feedback needs to be high, recognize that planning an entire timebox will likely be a waste because we will change what we want to work on as we get feedback. Many teams work on small items that come in on a continual basis - in which case timeboxing is not going to work well.

When using timeboxing, work as closely with the customer as possible to get quick feedback. This will eliminate waste and improve product quality. However, the availability of the person who best knows what the customer needs is also a factor. If they are only available on an irregular basis, time-boxing may be the only option.

Organize the development intake around QVRs.

Make stories small.

Many teams that use timeboxing don't do demonstrations of what they are building until the end of the iteration. So, if timeboxing is being used, try to get as much feedback from customers as you can.

*Focus on completing stories, features, and releasable items as quickly as possible to provide feedback on the quality of what's being built.*

Get feedback after each item is accomplished.

### *Scorecard for work on small, releasable items of value.*

1. We just take in work and don't attend to what's releasable or not
2. We work on mostly stories and piece them together to figure out what to release
3. We work on epics and stories and decide what to release as we go.
4. We take in QVRs into our intake queue but work on everything together..
5. Teams work on QVRs and focus on finishing the most important ones.

Exercise
Take a few minutes to reflect on when you worked on larger than necessary items. Did the extra size hurt or help?

## Agile Artifacts.

*Often reducing batch size is all it takes to bring a process into control. Eli Goldratt.*

*There is nothing so useless as doing efficiently that which should not be done at all. Peter Drucker*

Agile, Flow, Lean, and the Theory of Constraints all focus on quickly delivering high-value items. As evident as it is, it's worth pointing out that smaller items are easier to get completed and to the customer more quickly than larger ones. This not only provides value sooner but provides feedback quicker. This has been partially manifested with the concept of small stories that add incrementally to the value being built. But what must also be considered is that the ultimate focus must be on the value realized by the customer. We, therefore, need to consider two sides to the requirements. First, what is the smallest releasable (realizable) value we have and how do we build it in small, end-to-end slices (to get feedback)?

But small is not the only requirement. We must be creating what's of the highest value. This focus can enable us to bind business with development. We identify the highest-value items that can provide value

to our customers that are consistent with our strategies. We then build them quickly to get feedback quickly.

## The Software World Is Not Like the Physical World and What That Means.

Before going into how we should create our artifacts if we're developing software, we should pay attention that software products are considerably different from physical products.

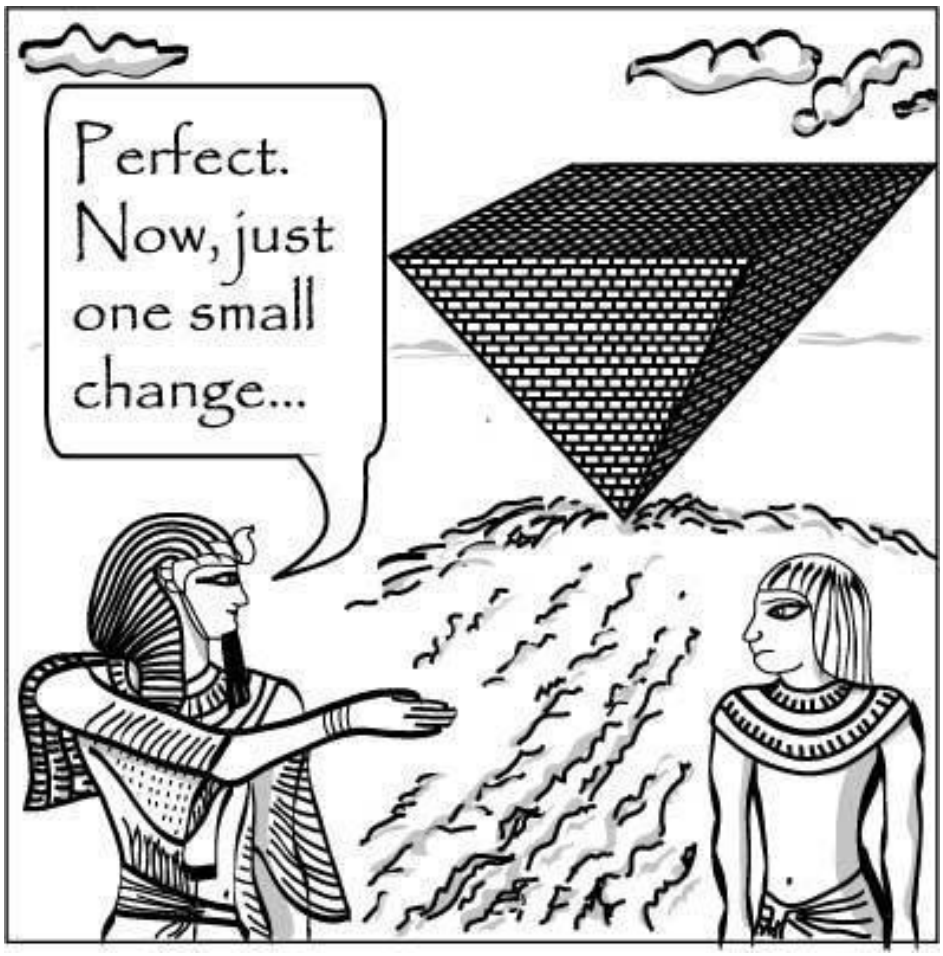
While Lean came from the manufacturing world it is important to recognize that software development is not like manufacturing. In manufacturing you are building the same things (albeit with some variation) repeatedly and try to cut down variation in the process. With product development, you are creating something for the first time and want the variation to try new concepts.

However, software development is not like physical product development either. Even though both are about creating something of value that is new. It is essential to understand these differences and both the risks and opportunities they create.

### Differences Between the Physical World and the Software World

**Visibility**

- In the physical world, you can often see mistakes while they happen. In the software world, you must take an extra step to determine if something is wrong (testing). In other words, getting feedback about how things are going requires an extra step in the virtual world.
- It is easy to see when there is too much work taking place in the physical world. This is not true in the software world.
- The progress being made in the physical world is more apparent than in the software world.




**Dependencies**

- Changing the foundation of something in the physical world often has a significant impact on whatever was depending upon it. In the software world, however, decoupling a foundation from an implementation using it is not costly.
- Dependencies in the physical world are often costly to change. And there is an additional cost if one tries to decouple them. In the software world, dependencies can be decoupled at virtually no cost.

## Delays and Too Much Work in Process (Too Much WIP Causes Delays)

- Delays in the workflow are more readily apparent in the physical world.
- You don't need to move anything (but your fingers) to shift what you are working on in the virtual world. This obscures the real cost of task switching and leads to too many things going on at one time.

## Fixing Errors

- Fixes to an error discovered must be done at the location where the error occurred, but in the virtual world, fixes can often be made remotely.
  The implications of the above
- It is essential to create visibility on the work and workflow when creating software. Especially attending to too much work in process and delays in workflow and feedback.

- Errors can replicate quickly in software if dependencies are not attended to.

## Opportunities in the Virtual World

It's not all bad, however. In the virtual world, there are several opportunities that can't be replicated in the physical world without significant cost.

It is possible to design, build, and deliver software in small increments as fast, or even faster, than building something in its entirety even when you know what to build. As we build a small increment, the next increment to build becomes clearer.

## Incremental Design, Construction, and Delivery

A certain degree of incremental development and delivery is possible in the physical world. For example, housing complexes are built one house at a time. But the increment of value in the physical world is usually greater than in the virtual world. For example, you can't move into the second floor of a house until the first one is built. However, in the virtual world, it is often possible to do the equivalent of this.

Building part of the functionality and using it is not only of value for quick delivery. This means that we can get quicker feedback on value, usability, and design while creating value quickly. In the physical world, you can't move into the bedroom on the second floor before the first one is done. Agile suggests we build in small slices of real value.

While set-based development is often required to try out different options in the physical world, the equivalent of it can often be done by combining working software with prototypes and mocking.

The cost of replicating software (such as software as a service) is significantly lower than in the physical world.

You can track how software is used for virtually no cost.

Remote service, error tracking and updates are possible.

People can team in the software world much easier than in the physical product development world.

## Learning Lean for Software Development and Knowledge Work

Lean is an approach and attitude integrated with an understanding of some key, universal truths, that apply differently in different situations. This is not unlike how gravity applies under, on, and above water, but in different ways.

## A little history

Near the beginning of Agile, the most common artifacts were features and stories. Stories were small bits of functionality retelling a conversation a customer had with a developer. It was often written as "As a <type of user> I want <some goal> so that <some reason>." In these early days, teams would not always look ahead to see what functionality was needed but would just build little bits moving in a direction aligned with the business's strategy. Stories would be added and released when they provided enough value to justify it.

Stories are often grouped into a collection of functionality that can realize value called a feature. For example, one story may be about fast-forwarding on a DVD, with another about reversing. But releasing either doesn't make sense without the other.

As Agile became more widespread and larger projects were worked on, the need for a vision of what was being created arose. These were sometimes called 'epics,' meaning large stories. They contained within them the concept of some product to be built. There has always been some confusion about the relationship between epics, features, and stories. Much of this is because there was no good definition for what a releasable artifact was. Sometimes features could be released (had value on their own) sometimes not. This was even true for stories.

### *The Limiting View of User Stories*

In many ways, Agile has never grown out of its heritage of a team-centric approach working with a customer. Not all functionality is about the user of the system. Function needs to be written for owners and managers of the business, not to mention anyone providing legal or financial support. Putting requirements into the form of 'user' stories is somewhat myopic. I prefer calling our stories "stakeholder" stories. But I acknowledge that may be a bridge too far for some. Whatever term you use it's important to remember that stories are not intended for just the users of the system.

## The difference between discovering if a product is viable and extending an existing product

There is always more unknown when we are creating a product for the first time. Without an established market we can't be sure our innovation will be successful. In many ways, creating a new product is more about discovering if it is even variable.

On the other hand, when we already have a product and have a customer base we can talk to, the risks involved are less. We can often infer from the existing customer base what will be useful by talking to them. Since they are already using the product they can usually tell whether an enhancement will be useful. QVR

These two differences for building products - creating new ones and enhancing existing ones - requires us to have different artifacts for managing the requirements. We will call these Minimum Viable Products and Minimum Value Increments. MVPs will be focused on validating the viability of the product while QVRs will be focused on discovering, implementing and deploying what is of the greatest value.

## Minimum Viable Product (MVP)

Although Eric Ries didn't originate the term MVP in his book The Lean Startup, most people now use his meaning of it. A minimum viable product (MVP) is a development technique used to develop a new product or website with just enough features to satisfy early adopters. Only after considering feedback from the product's initial users is the final, complete set of features designed and developed.

### *Here is how The Lean Startup describes MVPs:*

- Used for developing products for early adopters by focusing on learning what they want
- Geared toward startups
- Designed for the first time a product/service is released
- Usually built by a small team that can already pivot

This is very useful; however, MVPs are not universally applicable. The question is, what do you do in these situations when:

- You are an established company
- You are building enhancements to an existing product/service
- You are in IT and implementing known operating processes
- The teams building it are not aligned and don't work together well.

Of course, in most cases, the details of functionality are rarely known well in advance. But what about the *value* of a product or service? In times of innovation and new products, value is not yet known. It is not clear if the product or service is even viable in the marketplace. This is where the techniques of the MVP are most helpful.

But when a product or service is more established, there should already be a good idea about its viability and the value that an additional feature will provide. You do not need the experimental approach of the MVP. This is the situation the Quickest Valuable Release (QVR) addresses.

Eric Ries defined the Minimum Viable Product (MVP) to determine if a proposed new product was viable. A slice of functionality would be built, shown to, or used by a customer to move us forward in understanding what was needed. At some point, the product's viability would be affirmed or denied.

In the Lean Startup, Eric Ries discusses how to discover if a new product is viable. He suggests starting with just enough functionality to gauge if customers will like it. He calls this a Minimum Viable Product, or MVP. The concept is to build an MVP quickly, get feedback, get the next slice of functionality, and repeat this until the product is ready to release as a new product.

Each MVP provides functionality but may not necessarily be easy to use. Once the viability of the product is established, its usability may be increased. It's important to realize that the MVP is a slice of functionality as shown in figure 1.

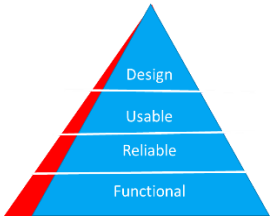



Figure 1: MVP as a slice of functionality.

*Note for users of SAFe. The MVP described in SAFe bears no relationship to that of Eric Ries' MVP. SAFe uses MVP ubiquitously which causes it to lose the value it would have had otherwise.*

In Ries' StartUp Way, he extends the idea from questions about the product to all other aspects of the product. These include assumptions about marketing, support, how the market will perceive the product and anything. The MVP approach should be considered a method of challenging all of the assumptions present.

## Understanding the different types of requirements needed.

But epics, MVPs, features, and stories leave out the most critical need of all – what do you call the artifact that describes the piece of functionality that extends an existing system? This is where the Quickest Valuable Release (QVR) comes in.

## Quickest Valuable Release – QVR.

*Deeper insights are available on the online [Amplio Concepts sheet](#).*

Most work done by established organizations is enhancing new products. This is quite different from discovering if you have a product. The Minimum Marketable Feature put forth by Denne and Cleland-Huang in Software by Numbers: Low-Risk, High-Return Development was about enhancing existing products. SAFe uses the term MMFs and references this book as well but the definition they use is not what Denne and Cleland-Huang propose.

Around 2005 Net Objectives enhanced the notion of the MMF to be called the Minimum Business Increment (MBI). This renaming was done for IT departments that created software but didn't market it. Over time, the MBI was enhanced to include not just what was to be built, but who was to build it as well as any acceptance criteria. We later changed it to Quickest Valuable Release so that it would relate to government IT departments as well.

A QVR is a clear description of the minimum amount of value that can be realized from the people using it. It also details all the steps required for its release and realization.

- **Quickest.** It should be the value that can be released the quickest. The focus on speed is two fold. First, it gets value to the stakeholder. It also enables quicker feedback which can minimize waste. We must, of course, attend to transaction costs in seeing what we can do the quickest.
- **Valuable.** The focus is on delivering value from a business perspective. While it's directed at the customer, it needs to be aligned with the strategy that initiated the product or project.
- **Release.** It represents what is actually released to the stakeholders.

It can be used in many different contexts: business, not-for-profit, government, services, and products. It helps them realize the business value quickly. It is important to be clear that t*he QVR is not a reason to deliver less; it is a focus on quicker delivery of value.*

**Comparing MVPs with QVRs**

While there are similarities with MVPs, they are not the same.

The Quickest Valuable Release (QVR) applies when the value of an enhancement or new product/service is reasonably well-known. It is the smallest piece of functionality that can be delivered that has value to the users of what's being built. It helps the organization focus on realizing that value quickly. Of course, we must still validate that we've gotten it right and that it is of value. But the degree of uncertainty is less than with an MVP.

Here are some characteristics of a QVR.

- Adds value for the customers of the organization.
- Provides valuable feedback that the proper functionality is being built.
- Provides valuable feedback that the functionality is being built in the right way.
- Provides functionality that can be delivered and which can also be validated as useful.
- Enhances the ability of the organization to deliver value in the future
- Contains all the pieces that are required for value realization. This includes any work required by documentation, ops, and marketing.

Both product management and developers typically work together to create a description of a QVR.

QVRs are created by first determining who your target audience is. This target audience may be external or internal. Then, decide on the scenarios for this market for the business objective in question. Focus on the Quickest Valuable Release for the scenarios in question – and that becomes your QVR.

A series of QVRs is often required to achieve the desired functional implementation of an initiative. By building and delivering the QVRs incrementally, you deliver value and get feedback more quickly and this offers you the opportunity to pivot more effectively.

Value for the organization may involve paying down technical debt, achieving steps in agile transformations, improving platforms for a product or anything else that the business considers to be of value. It is up to the business to identify value.

Finally, any adverse effect a QVR may have on existing functionality must be incorporated into the QVR about to be built and not thrown over the fence to those who built the affected code. Determining how one QVR affects another is usually the responsibility of business architects.

| Minimum Viable Product | Success Engineering | Quickest valuable release |
|---|---|---|
| Discover if a new product is viable | Investment reason | Get a return on investment |
| Focus is on learning and challenging assumptions | Focus | Focus is on building, delivery, and consumption |
| Create a new product for early adopters | Customer market | Extending to a new market segment or solidifying offering to existing market segment |
| Have focused interactions with prospective clients | Marketing | Market with existing marketing organization through existing channels |
| Start with a dedicated team | Teams need to create | Likely to require several teams coordinating together unless a product alignment already exists |
| Start with small features and expand | How to implement | Decompose into features that implement only the scope of the QVR |
| Little direct impact on existing products as this is new | Impact on existing offerings | Likely to interact with multiple systems. |
| May not have a viable market. Want to discover this early | Risks | Enhancement may not be as valuable as anticipated. May affect other offerings. |
| May need to create new marketing and support systems | Full kitting | Need to coordinate with all necessary parts of the organization. |
| What will validate that we have value | What to build first | What we're sure will give us the biggest return |

Figure 2. Comparing MVPs with QVRs

## QVRs attend to all stakeholders, not just customers.

QVRs can also focus on value for internal clients. The focus is on business value and sometimes this is value for the organization. It is not always just for an external customer.

This does not just include paying down technical debt or agile transformations. In companies whose products are platform based, a QVR could be an improvement to the platforms it uses. It could be about improving internal processes and/or tools in the organization.

## Advantages of using QVRs

QVRs are typically documented to clarify the work that needs to be done. Here are some advantages of this approach.

- **Provide an early descoping to high value.** By doing this, the organization can focus on manifesting the most important value. Smaller pieces are easier to manage. It is as Eli Goldratt, the creator of the theory of constraints, once said, "Often reducing batch size is all it takes to bring a system back into control." Smaller pieces can be delivered more quickly. And, by focusing on the high-value pieces first, descoping early helps you avoid spending time on items of lesser value.

- **Ensure completeness to realize value.** A QVR contains all the work that is required to realize value. The scope of the QVR includes non-development aspects of value realization, such as user documentation, market support, ops, and others. QVRs create visibility throughout the entire value stream and provide clarity for DevOps as well.
- **Enable the ability to sequence the list of work to be done while attending to shared services that are likely constraints.** This also enables avoiding starting work until you have the capacity to complete it.
- **Provide clarity of what to align around.** All parts of the organization should be working towards defining, implementing, deploying, and allowing for the realization of the most value as defined by the business stakeholders.
- **QVRs ensure that at all levels, the scope is always constrained by a focus on the faster realization of value.** Of course, when QVRs are initially defined, they represent the minimum chunks of business value that can be realized. But then the QVRs are decomposed into features and stories, the scopes of the features and stories are limited to that of the QVR. And this means building only what is needed to realize value. This contrasts markedly with most agile methods of decomposition which start with epics and then pull the most important features out of the epics. While this does limit the scope, the features are often built fully scoped instead of limiting them to a more focused target audience or purpose.
- **QVRs help to manage WIP.** WIP is often thought of as the amount of work being worked on. But if a feature is started, then that entire feature is work in process. Same for an epic. QVRs have an influence on the amount of WIP because teams know they need to focus on finishing all of the stories in a feature and all of the features in a QVR. Plus, the features and stories are smaller since they are just implementing the part of the feature needed for the QVR. QVRs, therefore, minimize WIP by being smaller to begin with, having smaller pieces be decomposed from them (features and stories) and providing a higher view of what to finish.

QVRs are fundamentally different from epics. An epic is simply an agile construct that represents a “big story” without making the connection to value. A QVR is oriented toward business stakeholders and is directly tied to business value. It surfaces the conversations needed to get deeper agreement on if and when something should be built.

QVRs can create a focus for Agile teams by making it clear that these are the items that are to be released. Most teams amass stories and features to be released but it’s better to focus on the actual value intended to deliver. This also enables culling out parts of a feature that aren’t needed for a QVR. The deferred analysis enables the team to release the QVR faster.

Why QVRs Are So Important

Consider these questions:

- How are you deciding which market segment is most important to you?
- How do you decide what it takes to deliver value?
- How do you avoid working on items of lesser value than what can be delivered sooner?
- How do you align teams that are required to deliver value?

Notice how QVRs provide answers for these.

## QVRs must be fully-kitted

The full-kit is a checklist that contains all of the elements that are necessary to complete a task or project. The concept comes from the Theory of Constraints.

Before you would start a paint job you'd make sure you had all of what you needed - paint, drop clothes, brushes, tape, etc. Before you start a project you should make sure you have the full-kit. That is, what you'll need to get it out the door.

Too many projects are started too soon and end up waiting for parts of the organization to do what's needed to get value consumed.

You don't start working until you verify that all the boxes in the full-kit are checked.

This should be part of the Definition of Ready.

- It is not a phase gate.
- It is a way of avoiding starting something too early where it has to wait to get finished later.

It is not enough to build functionality to deliver. We must include all of the functionality and support information required for deployment. This would include, marketing, sales, support, ops, etc.

## Amplio Sidebar: Lead the pack by being more Agile when change happens.

*A case study by Al Shalloway*

Many years ago I was brought into a company that wrote networking software to teach them design patterns and provide some coaching. The company wasn't that large - I remember it having about 50-75 people in development.

After a couple of days of training, the development manager asked me if I would talk to their executive team for an hour about Agile. The teams wanted to be more Agile, but management didn't understand Agile, and they thought I could convey its value to them. I said I'd love to have a chat.

I sat down with all the C-level folks and the director of marketing. I talked to them about being more nimble and delivering more quickly. At some point, the marketing person said that they needed all of the functionality requested and couldn't build things in stages.

I, of course, don't know if this is true or not. It's not wise to discount what a person says because you've heard similar things said that were incorrect. So I asked him to elaborate. He said that their customers did feature comparisons between their software and their competitors and that they needed to win on this feature comparison.

Now, since I didn't know their business, I started asking questions to probe deeper into the reality of the situation.

I know that many times, not all features are used in an application so I asked him if they were. He responded, "They're not all used but they want them there. It gives them a sense of comfort that we're the best."

I asked, "Do they know up front that they don't need all of them, or do they discover later that they don't need all of them?"

He responded, "yes, but they want to have them just in case they need them."

I didn't know his technical background, so I asked him, "You do know that having more features adds complexity to the system and slows down future development?" He said he did.

So the issue now, by his accounting, was that prospective customers wanted them to have all of the features their competitors had, they knew they wouldn't use all of them and that having all of them slowed development down.

I accepted what he said as accurate because he was in a better position to know this than me. But that didn't mean he'd been making correct decisions on this. I saw another possibility he hadn't mentioned. So I asked him:

"Would you rather be known as the company that has all of the features available, but many won't ever be used, or would you rather be known as the company having the features you need and providing new ones that are needed faster?"

He paused and considered this. This option had clearly never been considered. He said he'd rather have the important ones now and be able to create new ones faster.

This is the advantage of being Agile.

## The Bridge Between Business and Development.

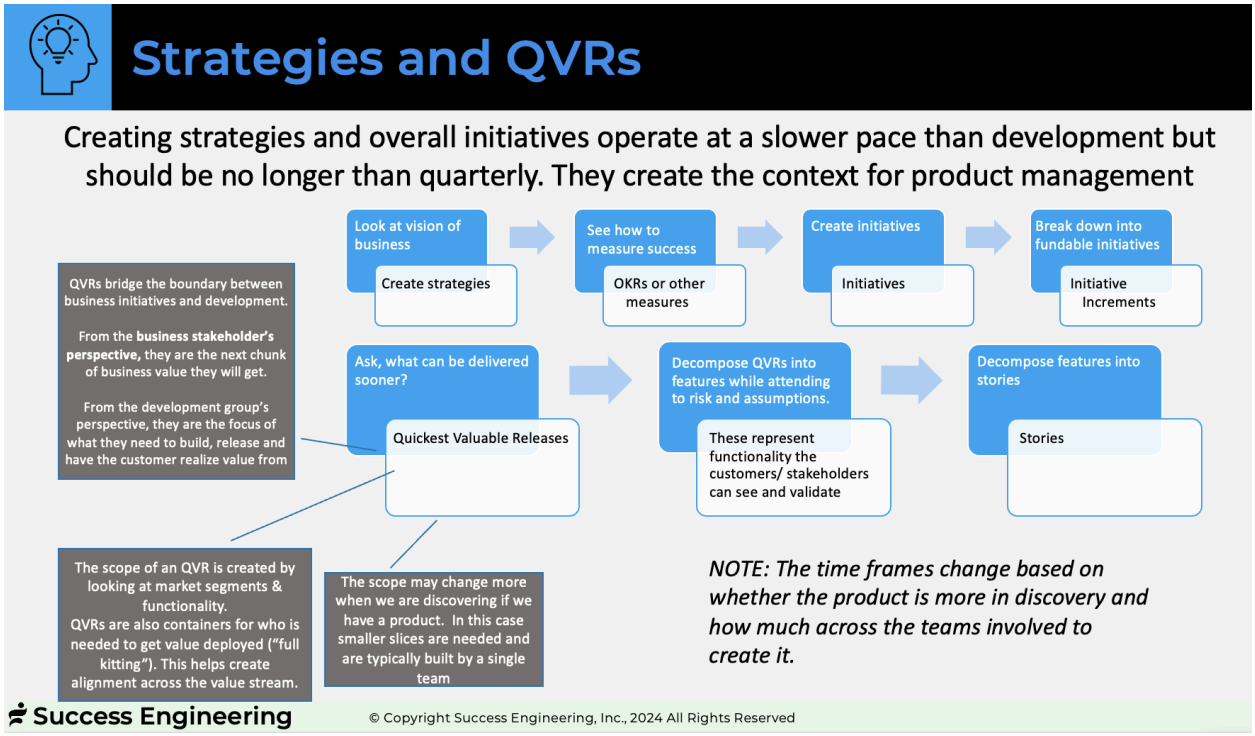



Figure 3. QVRs and MVPs in the hierarchy of artifacts.

## Line of Sight

It is important to see the successive decomposition described in Figure 1 as providing a "line of sight" for every artifact involved. For example, whatever level we're at we can identify the next level up that it came from – all the way to the top.

## Targeting Markets and Increased Alignment.

When writing a QVR, consider the scenarios in which the new capability can be used. You may have scenarios based on the different market segments using the capability. QVRs can be created based on which scenarios and which markets should be targeted for maximum value delivery. This might result in several QVRs.

By creating QVRs that represent focused business value, they can be sequenced in order of importance to the organization. That is, those QVRs that deliver the greatest value soonest can be sequenced as a higher priority than those that don't. This alignment of what is of greatest business value can also be used to align disparate teams in building things in this same order – thereby working together more effectively.

### The requirements for a Quickest Valuable Release

- Stated as a business objective
- What capabilities are needed to implement it
- What is the expected return
- Must be able to calculate the cost of delay for it

MVPs and QVRs are similar in that:

- They use vertical slices of functionality to validate the work
- They are customer centric.
- They are both built in thin slices

## Using MVPs and QVRs Together.

Here is what to do. Use MVPs when you are trying to *discover* if something is of value. MVPs are therefore used for creating new products. QVRs are increments of value to be realized. You should have a good sense of what this value is before even starting the QVR. If you don't, start with an MVP and then move to QVRs.

This approach helps to make things clear by highlighting when we are in one state or another. MVPs when we are in discovery and QVRs when we are focused on realization. See Figure 2.

This helps when thinking about funding. For example, MVPs require short cycles and smaller teams. QVRs work well at scale.

This helps when thinking about funding.

- An MVP should be funded for the value of discovery. After it is built the decision to continue, pivot or stop should be made. It is also possible that an MVP is not intended for a full release but to a limited audience to determine its viability.
- QVRs need to be fully funded. Remember that a QVR is the *minimum* business increment so this may not be much funding.

# Why QVRs result in smaller features and why they should be used in big-room planning.

The use of QVRs enables building only those parts of a feature that are needed for the QVR. This enables them to be built quicker and avoid waste that might ensure if we find we never need that part of the feature. This quicker feedback limits work in process as well.

This is critical in planning events. Consider when a team or group of teams starts planning with features not decomposed from a QVR. Let's say they can pull in enough work for the timebox being used. While this looks good at first, this means they are building more than they need to get a release because the features include scope not in the QVR. But now consider what happens when it appears they won't get enough value pulled into the timebox. At this point, they must thin out the features. This adds confusion to planning events. And the quality of the features is sure to suffer.

## Creating QVRs

There is no one way to create QVRs. However, there is an expectation of what a QVR will contain. QVRs should:

- Meet the organization's definition of ready and have a definition of done
- A statement of who will get value from the QVR
- What capacities (teams) are required to implement it
- What capacities (teams) are required to release / support / market it
- Any architectural issues required to build in
- A description of the features required to manifest its value

QVRs can be built by first looking at the functionality needed and then writing the features to implement them. Or they can be built by identifying the features needed and combining those needed to comprise the QVR. If this later approach is used, it's important to reflect on the features after the QVR is defined and see if any part of them is required for the QVR. This is a useful step because sometimes the features when defined are larger than necessary before the QVR has been clarified.

## Going from Epics to QVRs.

One way to create a QVR is to go through this sequence.

First, make sure you want a QVR, that is, you are creating the requirement for enhancing/extending an existing product. Ask the following questions:

1) What part of the epic do you want to deliver soonest? Consider different subsets by looking at one of the following:
    a. Customer
    b. Stakeholders
    c. Geography
    d. Market segment
    e. Particular customer
2) What do I need to
    a. build
    b. market
    c. support

    d. deploy
3) Must allocate cap ex and op ex (learn more about cap ex and op ex by going here).
4) What are the features in the context of this QVRs

# Quickest Valuable Release Template.

This QVR Template is used to prompt questions when QVRs are being created.

It should be considered a starting point and should be adjusted for the organization using it.

## Description of the QVR

_____

_____

_____

## Benefit Hypothesis

- The anticipated benefit of this QVR
- How will we evaluate if we've achieved this

## Initiative This Came From

## Customer related

- The target market for this
- Who are you building this QVR for?
- Do they have another customer this is for?

## Requirements

Use cases to describe the value

What's needed to meet the organization's definition of done

## Architectural Issues

- Are there any system or application architectural issues to be explored?
- Has the business architect signed off on this?
- What help from architecture will be needed

## What's needed for release / realization

- Development teams needed
- UX
- Documentation
- Marketing
- Support
- Shared Services
- _____

### Ops Issues

- Who will be involved?
- When do they need to get involved?

### Validating you have the smallest QVR

- Is there any subset of the QVR that can be delivered sooner? Consider: geography, market segments, language, anything else you can think of.
- Is the QVR larger than it needs to be because of deployment issues?

### Miscellaneous

- Are there any risks associated with not completing this QVR?
- Are there other QVRs dependent upon completing this one?

## FAQs Relating to MVPs and QVRs

Q: Let's say you validated the MVP and you learnt that it was viable and there is product market fit - then does it evolve into a QVR?

Good question. MVPs evolve into QVRs. In that attached picture I use the scale in between the two to show that there is a gradient. At some point you shift from discovery to improving a product. Validation is needed in both cases. The point to the distinction is to emphasize where your focus needs to be.

## Coaching Tip When People Say They Need It All

When you introduce QVRs to customers and product owners and ask them what part of the larger requirement they could use first, expect them to say "I need it all!" This is to be expected because they have probably had times they only got part of what they asked for. They will be listening to you through the filter of "we've run out of time to get you the rest." Arguing with them about this won't work.

Instead, tell them something like this: "Yes, we're going to get all of it for you. We've found that we can build things better, faster and with higher quality when we understand what's more important. So you giving us this information is not to affect the plan or release, but just to help us. Understanding the primary market and languages and geographies will enable us to make a better product at the end."

Then use the results of this conversation to create a QVR yourself. If they won't have this conversation with you, create the QVR as best you can. Then, when it's been created and you want to build the next one, tell them something like this: "We're continuing to build the product as we said we would. We've got part of it already, however if you want us to release that now." Then tell them what you've got done.

They will likely say something like "you can release that now and continue building the rest?"

At some point you can ask them if there is something else they'd prefer to get instead of the rest of the lesser important aspects of the requirement.

## Lean allocation (attend to CapEx / OpEx)

## Product Management: Strategies -> Initiatives -> QVRs/MVPs

## Product Backlog

## Decomposing QVRs into features & stories

## Capability PM7: Full kitting Capability (support, marketing, *sales, PeopleOps, Legal)*

### *Objective*

Ensure we have all of the players needed to complete this QVR identified.

**Why This is important**

It is easy to build the functionality for a QVR and miss some key component such as document or hardware support.

### *Implementation Methods*

Create a checklist for each general type of QVR that contains a list of the players typically required for release and consumption of the value of the QVR. During the QVR definition process go through this checklist and ensure the required people are available. If you are in a company that does big-room planning, verify the accuracy of the full-kitting during the planning sessions.

### *Symptoms of not having this capability*

When full kitting is not done in a disciplined manner, the function of QVRs will be completed but not be releasable, will not have proper documentation, be able to be supported, or have some other missing function that makes the function unable to be consumed.

### *Anti-pattern*

Wait until the completion of the QVR to see what's needed for deployment and consumption.

## Full Kitting Pattern Organizations

**Context:** More than

**Challenge being solved:** Having functionality be completed but not releasable.

**System of forces and how they relate to context:**

- The number of functions required to get the function consumed
- The disparity of management of these functions
- Whether QVRs are being used (if not this becomes much more difficult

**Solution:**

**Related patterns:**

**Potential implementations (risks of the implementations):**

## 1.3 Capability: Ensure you're creating the greatest value.

*There is nothing so useless as doing efficiently that which should not be done at all. Peter Drucker*

**Objective**

While we are always looking to increase our ability to build more in less time by eliminating wasted effort, at any one point we have a limited capacity. To maximize our value-adding capability we need to work on the most important items.

Value, however, is often in the eyes of the beholder. The question of value must include who the beholder is. We must break from the bias that the user of our system is the main stakeholder in our system. This bias is rooted in the roots of Agile which is more than 2 decades old. Instead, we must look at all stakeholders. This includes those in our company as well as the company using our services. For example, a restaurant management system will have stakeholders in the restaurant (management, legal, food services) as well as in the company building it (marketing, sales, executives, legal). All of these stakeholders will be competing for the constrained capabilities available.

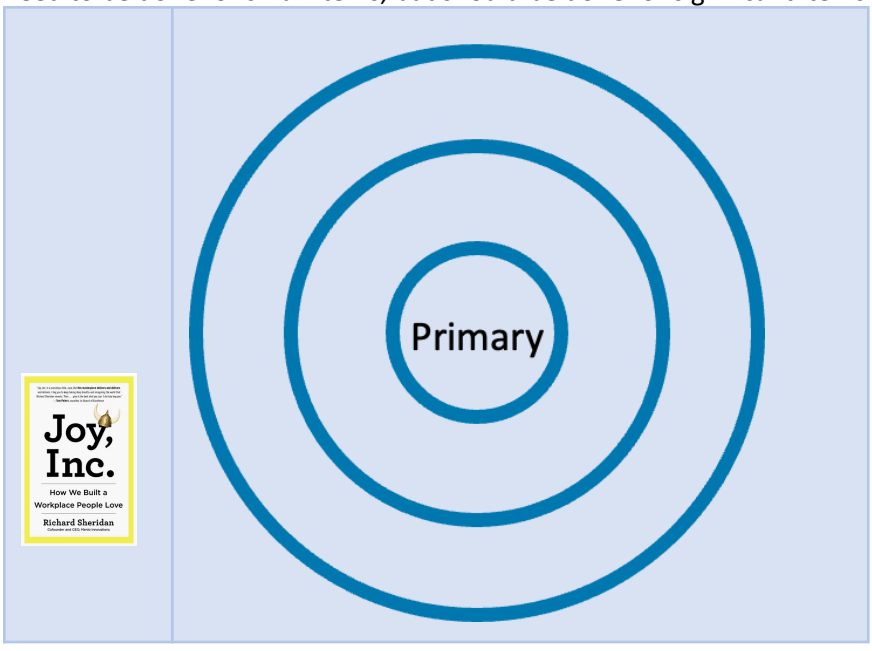
### *Why this is important*

The most significant cost/waste is building the wrong thing. Don't be misled into listening to only some of the stakeholders involved.

### *Implementation Methods*

**High-value items** can be used to align work to business value. If this is not present people will not coordinate what they are working on to get value out the door quickly. Whether you are using timeboxing or flow is not particularly significant except that flow will likely get you quicker feedback.

One method for discovering what is of value is to identify the value of those stakeholders. Joy, Inc, presents the idea of a persona target. It is used to identify the users of a system and how their importance relates to each other. But this can be used to identify stakeholder values as well. This doesn't need to be done for small items, but should be done for significant items of work.



Use the target by placing the role of the stakeholder and what they consider to be of value on the target in the appropriate position. The more important the stakeholder the closer to the center it is on the target.

### *Scorecard for "ensure you're creating the greatest value"*

1. We work on things as they come in. Typically just doing what the loudest voice tells us to do,
2.
3.
4.
5. We have sequenced work to be done based on the values of our stakeholders.

### *Symptoms*

If you aren't building from the stakeholders' perspectives your product will likely not be the most important thing that can be built.

## 1.4 Capability: Have high product quality from the customers' perspective.

"Software is notoriously difficult to use. I've often said 'if a person acted the way most software systems do they'd be punched out within 5 minutes.'" Al Shalloway

Product quality has several aspects:

1. How well does it meet the needs of the stakeholders of the business who are funding the product?
2. How well does it meet the needs of the customers?
3. How well the product is built?

The third one has several aspects to it. These will impact how easy the product is to maintain as well as change. These aspects include:

- Technical architecture
- Business architecture
- Implementation quality

While building the right thing is critical, building it well is also essential. This is especially true when the company has long-term relationships with its customers. In this case, the quality of the product is of value to the customer since updates will be expected.

### *Objective*

Figuring out what customers want and need is considered to be notoriously difficult. But much of this is because most people pay attention to the system they are building and not the customer's journey. Focusing on the customer's value streams via the customer journeys is how we determine what to build.

Let's face it. A lot of software is difficult to use and built poorly. It's awkward, inconsistent, hard to learn, and hard to use. If a person responded to people the way most software responds to people, they'd be punched out in five minutes. This happens because most developers come from the perspective of what they are building instead of how the customer is going to use it. Or assumptions are made that don't

need to be made. For example, many products work best for right-handed people when with just a little thought they could have been designed for either left or right-handed people.

What's needed is software that functions the way people want it to. Even better when it opens possibilities or people that they hadn't considered. We want the software to improve how people work and live.

Most products, especially software systems, are created from the perspective of the product or the software system being built. But this is the wrong perspective to use. It creates a need for the customer to understand the way the product is designed or how the system behaves. Unfortunately, the way designers think, and the way customers think is often not in sync.

What's needed is to focus on what would be a good value stream for the customer. From this, we can ask "what would the customer journey have to be to enable a good customer journey?" When that is understood we can build the product or system from the customers' perspective, not the developers'.

### *A physical product example.*

I once was an avid photographer. I was using an Olympus camera that had through-the-lens light metering. This meant that I could use two strobes at once and the camera would automatically set the shutter speed. In looking for hand-held camera mounts that would hold the camera and an additional strobe (one strobe would be mounted on the camera itself) I noticed that many mounts would only allow for the handle to be mounted on the left side of the camera. Left-handers, like I was, were left out. In looking at these holders, it was clear they could have been just as well designed to be held by either hand. That users would want to do it differently was clearly not considered. Instead, it appears the thought process was based on the components of the camera mount.

### *Why this is important*

If people have difficulty using your products, they won't appreciate the value that might be in them.

### *Implementation Methods*

This section includes the same directives as the previous implementation section on how to implement and ensure you're creating the correct value. However, it has one more – use the customer journey to design the system.

As mentioned earlier in the book, the customer journey can be used to improve the innovation of products and services. We can do this by taking these steps:

1. Get familiar with the customers' goals and commitments
2. Look at their commitments to see if there are better ways to achieve them
3. Design a customer value stream based on these better ways
4. Look to see what the customer journey would be to enable this customer value stream
5. Design the system to enable that customer journey
6. Demonstrate to stakeholders what's been developed so far

### *Objective tbd - merge with this capability*

We must ensure what we're building is what is needed. This requires feedback.  Showing what we've done is a great way to get feedback. This also drives the development team to build demonstrable product and not build things in unobservable steps.

### *Why this is important*

Understanding what stakeholders need is not always easy. Furthermore, as we demonstrate what has been created, new possibilities often arise. Quick demonstration provides for:

1. ensuring we're headed in the right direction with what we've done
2. an opportunity to pivot to a better path
3. a realization we're not going to get what we had hoped for

### *Implementation Methods*

The main variation is in the cadence of demonstration. At a minimum, a regular cadence of demonstration is essential. Alternatively, if stakeholders are available, demonstrations can take place as soon as something new to demonstrate is available.

It is important that the product being built is demonstrated. A status report does not manifest this capability.

Symptoms of Not Doing This Well

If you aren't building from the stakeholders' perspectives your product may be awkward, even difficult to use. It won't garner the support your organization would like to get from its customers. You won't get viral recommendations from your customers that you'd like to get.

Working on the product and finding out well into its development that it is not what is needed.

## 1.5 Capability: Validate the requirements with examples.

*"You Cannot Learn What You Think You Already Know." Epictetus*

*"It ain't what you don't know that gets you into trouble. It's what you know for sure that just ain't so." Mark Twain*

*"The single biggest problem in communication is the illusion that it has taken place." George Bernard Shaw*

Reflecting on the quotes above provides insights into why we often have such bad requirements. People hearing the requirement think they understand it. But they don't.

### *Objective*

It doesn't make sense to build something if you aren't sure what the requirement is. After you build it the person giving you the requirement will use some acceptance criteria. Asking them what that criteria is before you build something will help avoid building the wrong thing. This acceptance criterion is already in their head or will soon be. They may just not have told it to you. The idea of "test-first" is really getting what the person making the requirement is going to use to see if the right thing was built before you start creating it.

### *Why this is important*

Without this, the creators of the functionality will mostly be guessing, while believing they know what is needed. This will waste time and create a lot of handbacks.

### *Implementation Methods*

This approach can be followed when using either timeboxing or flow. The only difference is the number of examples gathered.

A simple practice to always follow is for people being given a requirement to ask, "how will I know I've done that?" whenever they are asked to create something. This requires the requester to stop and think about it, gaining clarity on the request. It also provides the person about to create the requested

functionality with an opportunity to start their process by having the end in mind. When combined, these two create a mutual understanding of what is required.

How to do requirements by example – an example

Consider a situation where developers are tasked with writing a payroll system. They are given these initial requirements:

- People are paid a set wage when they work Monday to Friday for up to 40 hours a week.
- They are paid time and a half after the first 40 hours
- They are paid twice their salary when they work on holidays or Sundays.

The product owner tells the devs they have to leave for a week but are happy to answer any questions. So the devs ask:

- Where do we find the employee wages?
- What day of the week does a week end?
- Where do we find the hours worked by employees?
- What are the holidays of the company?

Possibly some others. Basically, the devs look at what they know, and what they don't know. They then ask questions about what they don't know. While often considered trite, there is an important truism that we know some things, we don't know some things, and there are some things we don't know, but we don't know that we don't know them. This is the territory for missed requirements. The way to avoid this problem is to include with requirements examples of what it means to satisfy them.

Let's say the product owner gives these three cases:

1. Pay with no overtime
2. Pay with overtime but no holiday or Sundays
3. Pay with overtime and some hours on a holiday.

Let's look at these in tabular form:

| Standard hours | Holiday / Sunday hours | Wage | Pay() |
|---|---|---|---|
| 40 | 0 | $20.00 | |
| 45 | 0 | $20.00 | |
| 48 | 8 | $20.00 | |

It is very important not to proceed until you fill in the rightmost column.

The first two cases are pretty easy and most people get them right.

| Standard hours | Holiday / Sunday hours | Wage | Pay() |
|---|---|---|---|

| | | | |
|---|---|---|---|
| 40 | 0 | $20.00 | $800 |
| 45 | 0 | $20.00 | $950 |
| 48 | 8 | $20.00 | ??? |

Now really, take the time to calculate the third. In running this example most people respond with $1360 or $1120. Actually, there are usually one or two other solutions given, but they are usually the result of math errors. That we get two different answers is interesting on its own. After a brief discussion, it is discovered that the discrepancy is due to whether the "Standard hours" is the total hours worked or just the non-holiday/Sunday hours. If it's the total number of hours worked, then the answer would be:

- 40 * 20 = $800 ---- this is our wages for our first 40 hours
- 8 * 20 * 2 = $320 --- this is our time for working on holidays or Sundays
- TOTAL = $1120

In the other case we'd have

- 40 * 20 = $800 ---- this is our wages for our first 40 hours
- 8 * 20 * 1.5 = $240 --- this is our time for working overtime
- 8 * 20 * 2 = $320 --- this is our time for working on holidays or Sundays
- TOTAL= $1360

But, in fact, the correct answer is $1520. Huh? That's because there's something else afoot. Something else we didn't know, but we didn't know we didn't know it.

Can you figure out why this is? Notice that that's not really important. Now that you know you don't know you can just ask.

Before giving you the answer, let's see if we can make it clear why this problem occurs. Consider a metal-cased pen, a plastic-cased pen, pencil, and a whiteboard marker. Would you consider them the same? Or different? Most people would say they are different. But now consider if you can think of them as the same. Most people say "yes, they are all writing instruments."

People can consider these items the same or consider them different. The question then is how you are thinking of them differently when you consider them the same and how you're thinking of them when they are different? The difference, of course, is the level of abstraction. They are the same conceptually but have different implementations.

Now, consider the requirement of "twice their wage." What is their wage? Is it their standard wage or their overtime wage? How would you know? You wouldn't. And you almost certainly didn't know you didn't know. You just figured it was their standard wage because that's how people talk.

If you asked the product owner, they'd have told you "twice the wage they are being paid." But what is the wage you are talking about? Remember when we talked about pen and pencils as the same, the customer may think of standard and overtime wages as being the same.

Now we can see that the answer to the third case is:

40 * $20 + 8 * $20 *1.5 + 8* $20 * 1.5 * 2 = $800 + $240 + $480= $1520

But also notice that given the answer of $1520 and a little Algebra, we can calculate what the rate we should be using in the 3rd example:

40 * $20 + 8 * $20 *1.5 + 8* x = $1520 OR

$800 + $240 + 8* x = $1520 → 8 * x = $1520- $1040 → 8 * x = $480 which makes x = $60

Wage = $60, the overtime wage of $20 times 1.5 times 2.

The bottom line is, never accept a requirement without asking the question (and getting an answer), "how will I know I've done that?" Notice this takes no extra work since the person asking for the requirement will have to figure this out eventually. You're just asking for it up-front.

How to take this further

I call this "baseline Behavior Driven Development (or BDD)." Dan North created BDD as a way to create clear requirements. BDD suggests using the format "<given> <when> <then>." The <then> is the output, not the action, which is left to the developer. Each line in the tables above includes the "given" and "then." The "when" is when the event that triggers the calculation happens.

### *Symptoms*

You are not doing this effectively if, after building something, the customer says it was the wrong thing.

# Development Workflow

## Capability: Have definition of done (DoD).

### *Objective*

DoDs helps create clarity on what is of value and specifies what's needed to be done to achieve it.

DoDs make explicit the agreements between people providing work to developers and developers saying the work is done.

A DoD is also an agreement between parties stating what it means for a work item to be completed. This may require specific actions while it is being worked on besides just what function the item is supposed to have.

### *Why This is important*

Handoffs often cause loss of information or people not being adequately prepared to do the work being handed off to them. A DoD states what it takes for something to get done. DoDs should include the functionality required and any involvement required by other groups (e.g., marketing). As Dean Leffingwell once said, "don't let something into the sprint without knowing how to get it out of the sprint."

### *Implementation Methods*

There is not a set way to do DoDs. What's important is to have a conversation with everyone who gives or gets things from the team and decide on the agreements between all parties.

#### Levels for the DoD

DoDs can be at the story level up to the QVR level.

Example DoD for a QVR:

- A DoD for a QVR specifies both what the value is and all of the people who need to be involved to create and build it.

Example DoD for a story:

- acceptance criteria has been specified and met
- product owner has signed off on the PBI
- the primary customer has signed off on the PBI

### *Symptoms*

When a DoD isn't present, many handbacks will likely occur between the people handing work off.

Take a few minutes to consider a time when you didn't have an agreement on a Definition of Done. How often did the team think they had completed the item being worked on only to discover that parts of it were missing and wrong? Would a definition of done have helped?

# 1.8 Capability: Have definitions of ready (DoR).

### *Objective*

Using DoRs helps identify items that may be useful to attend to before starting something. It can also be used to identify who what's being built is for. DoRs can be done at any level: QVRs, features, and/or stories.

DoRs make explicit the agreements between people providing work to developers and the developers. A DoR was sometimes used as a stage-gate in the past, so expect to hear some people say they are not a good idea. But in Amplio Development, we only mean them to be an explicit statement of the agreement between parties as to what it means for a work item to be ready. The DoR could be as simple as "it's ready whenever the product owner thinks it's ready." Most often, however, there are some other things the teams will want to see before working on it.

A DoR is needed when handoffs are done, so each party understands their agreements with others.

### *Why This is important*

Handoffs often cause loss of information or people not being adequately prepared to do the work being handed off to them. A DoR creates an agreement about what it takes to get something started. As Dean Leffingwell once said, "don't let something into the sprint without knowing how to get it out of the sprint."

When you let something get started before you have the capacity to build it without delay you slow it and other things down which creates waste.

### *Implementation Methods*

There is not a set way to do DoRs. What's important is to have a conversation with everyone who gives or gets things from the team and decide on the agreements between all parties.

#### Levels for the DoR

DoRs are done at the QVR level.

- People who need to be told that this QVR is being worked on have been notified. Consider Sales, Marketing, Ops.
- DoD has been defined.

### *Symptoms*

When these aren't present, many handbacks will likely occur between the people handing work off.

### *Story about DoR*

Around 2008 I was working with a client that was embracing Agile methods. I had a conversation with a small team about their work. They had been having trouble sometimes with building the wrong thing. I had a little conversation with them about this and if there was a pattern they saw. As they discussed this, they realized the pattern was that they mostly had this problem when their product owner wasn't available, and they figured things out on their own. Looking back, it seems kind of an obvious observation. But they hadn't taken the time to see what was happening. The bottom line is that once they noticed it, they made a definition of ready to not work on anything until the product owner was available and validated what they thought they needed to do.

*Exercise*

Take a few minutes to consider a time when you didn't have an agreement on a Definition of Ready and work was started without having identified some necessary aspects that were required to release it and get value. What was the result?

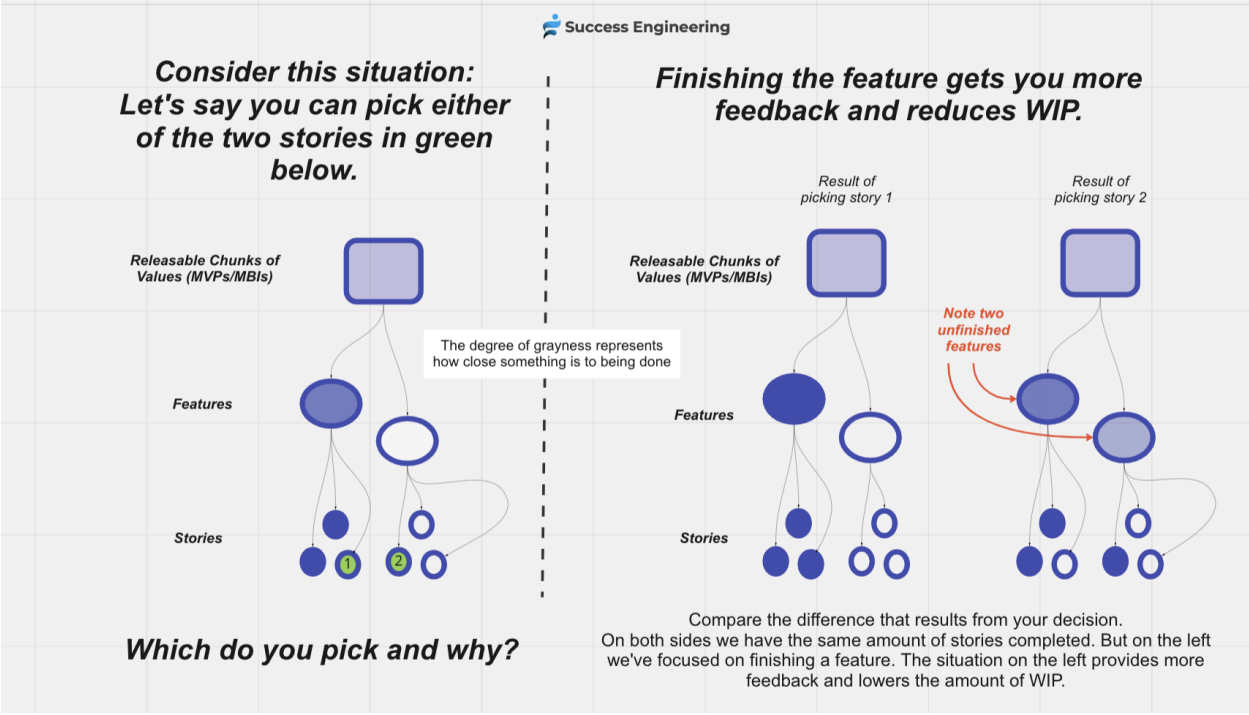
## Capability DW 4: Manage work in process to remove delays in workflow and lower risk.

### Focus on getting feedback while lowering WIP and reducing accumulated risk.

Accumulated risk is the risk inherent in work you are doing for which you haven't gotten feedback on yet. This is caused by incomplete work at all levels.

Accelerating feedback and lowering work in process are two actions that improve flow.

Consider being in the following situation. You have a QVR with two features, each having three stories. Which of the stories (in green) do you pick to do?




How you pick will result in one of the following situations on the right.

On both sides we have the same number of stories completed. But on the left, we've focused on finishing a feature. The situation on the left provides more feedback and lowers the WIP (number of features).

There is another factor to consider, however. After you finish the second story of the QVR being worked on you should check to see if you still need to finish that feature. Don't get trapped by sunk costs.

## A symptom of not focusing on finishing when using incremental development

When a team is using incremental development (e.g., Sprints) they often focus on being completed by the end of the increment and don't always worry about keeping work in process in a manageable state throughout the sprint. When this happens you often see many stories that have been opened but not completed at the end of the increment. If you are using a burn-up or burn-down chart the graph will look a bit like a hockey stick. That is, the burn-up or down will be mostly horizontal with it accelerating towards the end of the increment when there is a mad dash to complete things.

## Create a focus on finishing

WIP limits are a common way to manage work in process. They require significant discipline and are only one of many methods available. An easier way to start and still be effective is to create a focus on finishing. Completion exists at many levels: tasks, stories, features, and QVRs (or whatever artifact is being used to represent the next releasable chunk of value).

- When done with a task, look to finish another task in the same story.
- When done with a story, look to finish another story within the same feature.
- When done with a feature, look to finish a feature within the same QVR. This focus quickens the rate of feedback, which increases both quality and efficiency.

### *Objective*

Work in process (WIP) is not just the amount of work waiting in a queue. It is not even the number of stories or features that have been started yet not completed. WIP is the total of all the MVPs and QVRs that have been started yet not completed. While a story or feature may be complete, it is still a work in process if it hasn't been released. Managing WIP is not done merely by having WIP limits in queues to manage the waiting time in queues.

**Managing WIP is a continual process.** Once you have sequenced your work in the proper order, you can allocate your capacity to the items that are truly most important. Do not start projects that adversely affect more important ones merely to "utilize" your people.

> "*In product development, our greatest waste is not unproductive engineers*, but work products sitting idle in process queues." Don Reinertsen

Managing work in process does not mean just limiting it. It means managing both what and where work is taking place. This includes workload levels as well as where the work is taking place.

Work that isn't completed represents work in process. It is not just those stories that have been started and not completed. It includes all levels of artifacts that have been started but not completed, for example, features and QVRs. These unfinished artifacts represent risk. We may have feedback at the story level, but the feature or QVR has not been validated. We may find that we didn't do everything right when we get the feedback. Therefore, until we get the feedback, we have this risk.

It is also helpful to have work be interruptible. This way, the adverse effects of interruptions can be minimized.

### *Why this is important*

Here are some symptoms that WIP is out of control.

- People have to wait on other people.

- People are being continuously interrupted by other people.
- There are delays in the workflow and in getting feedback.
- There is a high multitasking cost.

A team working on two things simultaneously when they could be focusing on just one will delay the delivery of both. But what is not so clear is that doing this injects delays into both workflows. These delays in workflow and feedback induce more additional work. This creation of new work is why an interruption delays what people are working on by more than the time of the interruption itself.

All of these create waste. In many cases, much more waste than the actual valuable work. This extra work is, of course, unplanned, so cooperation is much more difficult and costly.

While people are busy, if we watch the work being done, we'll see that it is waiting in queues. Teams are more efficient when they are not overloaded. This enables them to get quicker feedback on what they are building, eliminating waste.

Artifacts representing the proper chunk of work to be completed before others are started are critical to avoid having many things in play. Too much work in process (WIP) causes feedback delays and adds unplanned, unnecessary work. It is essential to understand the concept of accumulated risk. This is the risk represented by incomplete work. That is, work started that has not been completed and verified to be what is needed.

We get feedback on whether we are building the right or wrong thing sooner by completing small items. Common artifacts in the Agile space are epics, features, and stories. However, these are not usually deliverable items. Not all of an epic will be delivered since not all of it is needed. Features and stories are often too small to be delivered by themselves.

NOTE: We are not suggesting the "one-piece flow" that Lean Manufacturing emphasizes; rather, we are trying to avoid working on too many things at one time.

#### Overloading people is the real problem.

While many in the Agile space talk about multitasking as a serious problem, overloading people with work is usually a bigger one. Multi-tasking is a symptom of having people overloaded with work. This will cause extra work to be created - not just a degradation in the effectiveness of people due to multitasking. This "extra work" (waste) can be several times the amount of actual work to be done. Not merely 75% of it.

### *Symptoms of Not Doing This Well*

Too much work waits between the different work steps.

Too many things are open at once, with items being completed at the end of the iteration. The graphic will look like a hockey stick if you use burn-up or burn-down charts. That is, few things will be completed until the very end.

When planning and coordination of work are not done well, teams tend to get interrupted a lot.

It is helpful to have work be interruptible. This way, the adverse effect of interruptions can be minimized.

### *Implementation Methods*

For years timeboxing was synonymous with Agile. Find a duration, see how much you can do within that time frame, and then plan that work. This approach can limit the total work in process to no more than what the iteration can hold. The weakness, however, is that there is often little guidance on how to keep work in process low within the iteration.

When using timeboxing, managing the amount of work done during the iteration is vital.

There are several ways to manage work in process:

- manage queues to lower the time between work ending at one step and starting at another.
- don't pull more work than what you have the capacity for
- looking to finish something after you've just completed something instead of starting something new
- don't analyze it until just before it's needed

Queues can also be managed. Larger queues mean more WIP and delays. One of the biggest queues is the iteration backlog. This is one reason that shorter iterations are good. Starting only the necessary amount of work is another way to manage queues. Having a well-defined input queue is essential for this. Without this, work is just given to teams, and the work queue in front of them grows without control.

The team should be focusing on finishing the next increment to be released. When timeboxing, an increment may be ready to be released before the end of the iteration.. Focus on finishing that even if it's not released.

The team is not doing a plan for the iteration (or time of the cadence) when doing flow. The planning is done in real-time. An item is pulled off the product backlog, it's completed, and the next one is pulled off and done. Focusing on finishing stories, features, and QVRs, before starting others will shorten the feedback time for each.

The best way to minimize incomplete work and accumulated risk is to focus on finishing. Managing queues can assist here. An easy way is to have team members see what they can finish next after finishing something. It's all too easy to start something new. But helping someone finish something that has been started will lower the amount of unfinished work.

*Note: You can use methods for flow within the timebox to improve work in process levels.*

### *Scorecard for "Manage work in process to remove delays in workflow and lower risk"*

There are four factors that improve this. They are:

1. Are MVPs and QVRs being used?
2. Is there a focus on not starting work on too many items?
3. Is there a focus on finishing in general?
4. Is there a focus on finishing QVRs?

The score is 1 plus the number of these being attended to.

### *Symptoms of Not Doing This Well*

Besides the obvious "starting too many things," these also are common root causes:

1. running mini-waterfalls in an iteration
2. having stories (at the team) or epics (at the program) that are too big
3. when something gets completed, people start something new instead of helping others finish their work
4. having developers and testers not collaborating
5. having development teams and shared services not collaborating

### *Risks of Not Doing This Well*

- Feedback is delayed resulting in poor quality and rework

- Too much work in process causes multi-tasking which lowers general efficiency
- There is a tendency to interrupt teams because it takes to long to get anything done

### *How This Capability Manifests the First Principles*

Avoids overload.
Speeds up feedback.

## Make work interruptible: A case study.

*Synopsis: It'd be great if Scrum teams were never interrupted, but reality intervenes. While many interruptions may not make good business sense and it'd be better if they didn't happen, many times something comes up beyond the control of the team and interruptions do make sense. A two-part defense against interruptions should be maintained. First, people outside of the team should understand the cost of interruptions and agree to do so only when it makes good business sense to not wait for the next planning event. The second is that the team should manage their work so that the cost of interruptions is minimized. And if interruptions are a way of life, adopting a flow system should be considered.*

All things considered, being equal, it's best not to interrupt work. But all things aren't equal.

I remember a company I was brought into by the CTO with the intention of me talking their Founder-CEO into not interrupting the team so often. They were doing Scrum ok, and I spent some of my time tuning them up. I went into the CEO's office and had a one-hour conversation with him. He was not just the Founder-CEO. He was the creator of the product and knew it better than anyone else.

I talked to him about the cost of the interruptions. As a former tech guy, he was well aware of the impact. We discussed the business need for the interruptions and their technical cost. He agreed that when he could wait until the end of the sprint he would. On average it'd be only a couple of weeks. But if the business value to be gained by securing a new client was big enough, he'd interrupt the team from their normal work.

From a business perspective, this made sense - but it wasn't what the CTO or the team wanted me to do. They were not happy with me.

I was satisfied, however, because it meant decisions that looked at the entire picture were going to be made.

I then had a conversation with the CTO and explained to him that he needed to make the work of his teams be interruptible. That is, instead of relying on not being interrupted for weeks at a time. Instead, he should have people focus on finishing and keeping little in play at any one time.

Not only would this accept the reality of the situation it would speed up feedback and reduce accumulated risk.

### *Setting things up to avoid interruptions TBD*

A dialog between teams and executives should be had to avoid interruptions.
This is just part of the natural healthy dialog between executives / management / development.

But the interruptions were typically coming as surprises where things worked well.

part of the agreement we made with management was for them to see if they could wait what was on average a week to interrupt the team. most of the time they could.

The sense before was that there was no appropriate time for an interruption.

How to use AI instead of being abused because of AI

## Intake process with classes of service

## Planning

## Team backlogs

## Effective domain skills

## Frequent integration

## Dependency Management

## Iterative team development

## Value creation structure at team level (include borrow, share team

This section describes how a few teams can work together. Meeting these objectives is enough for most companies with fewer than 75 people in their development area to be effective.

5.1 Have an effective value Creation Structure DTA

5.2 Shared backlogs

5.3 Aligning market solutions to teams

These capabilities are used to manage the value stream. We want few delays, quick feedback, and quick delivery of value. Each capability contributes to one or more of these goals.

## Description

Value stream management focuses on the work and the workflow that completes it. It focuses on removing delays, having quick feedback, and reducing handbacks. The time from starting to work on a description of value until the customer consumes it includes more than the time it takes to create the value. There is also rework caused by development mistakes, working on the wrong things, miscommunications, relearning, aging requirements, misunderstood integration agreements, and more.

## Understanding the cause of waste in value streams

We need to understand value stream management to understand the best practices that help us manage. Consider how organizations are managed. They usually are in siloes where each silo has a

manager at the top. The manager's job is typically to see that the people under them are 1) working on the right things, 2) working well, and 3) fully occupied. This seems so natural that we don't even think about it. But it also means the focus is on the silo, not across the workflow.

W. Edwards Deming said – "A system must be managed. It will not manage itself. Left to themselves, components become selfish, independent profit centers and thus destroy the system... The secret is a cooperation between the components toward the organization's aim."

The section on value streams discussed the shifts necessary for effective value stream management:

- Shift our focus from people to work being done
- Don't manage hierarchies; manage the value stream
- Don't try to go faster; work on having fewer and smaller queues
- Align people around the value to be realized
- The shift from focusing on people's utilization to removing workflow delays.
- Don't attend to local optimizations but attend to the throughput of the value stream.

We manage the workflow to have an effective and efficient value stream. We will look for handoffs, handbacks, enormous queues, incomplete work, and accumulated risk.

Accumulated risk is the risk associated with incomplete stories, features, MVPs, and QVRs.

## A perspective on managing work in process

There are several causes of too much work in process. How to manage these will be discussed in different places in this book.

These are the main causes:

- Having a poor intake process that makes it hard to see what's happening in the organization and ends up having too many things in play.
- Working without an attention to finishing. This results in more things in play than need to be.
- Management and product owners often interrupt teams without an understanding of the cost of doing this. (The True Cost of Interrupting Teams: An Amplio Mini-Lessons. https://www.youtube.com/watch?v=ns7O_35vRUI&list=PLMgMfRK3eoTf0ENQjRcyn-x-y0cs9WY7l&index=1 )
- Working on larger items than necessary which has them take longer to finish than they need to.

## Capabilities covered in this chapter.

These will help us see how to improve our value-creation structure.
2.1 Create visibility of work and workflow
2.2 Manage work in process to remove delays in workflow and to lower risk
2.3 Have a product backlog serve as an intake process
2.4 Have a near-term backlog from which you pull your work
2.5 Use pull methods to keep workload within capacity
2.6 Build in small, vertical, slices. MB
2.7 Agree on classes of service and their corresponding service agreements DTA
2.8 Use DevOps when applicable
2.9 Use automated testing to eliminate waste

## 2.3 Capability: Have a product backlog serve as an intake process.

### *Objective*

The team needs a place to pull work from when they are ready to do more work.

### *Why this is important*

The product backlog is, of course, a queue. We, therefore, don't want to have it be too big (which will cause delays) nor be too small, which will require last-minute creating requirements. Of course, small is ok if that's how the team wants to create what to work on next.

While teams may be building in small pieces, it is essential to see what context the small pieces fit into. It is also essential to keep work in process visible, so people know what is being worked on. This is much easier to accomplish when all work must go through a product backlog.

When you let more work come into your development pipeline than is being completed this will ultimately overload your pipeline. An intake process can help you control this.

When you sequence, you can better resolve capacity conflicts.  When you have two equally prioritized items you can rationalize that you work on them both. But then, you slow both down.

When you sequence, you can ask if working on the lower sequenced item is worth slowing down the one you consider to be more important.

### Side note on changing priorities

Changing priorities is a symptom of several other problems.

Your work should not be prioritized. It should be sequenced.

In other words, this is the most important item, then this, then this, ...

Otherwise, it's too easy to just say everything is a priority 1.

When you sequence, you can better resolve capacity conflicts. When you have two equally prioritized items you can rationalize that you work on them both. But then, you slow both down.

When you sequence, you can ask if working on the lower sequenced item is worth slowing down the one you consider to be more important.

If you manage work in process, you won't be swapping work in and out either. You'll start working on something and not those further down the list. Changing the order of backlog items is not a problem or wasteful.

Finally, if you use Quickest Valuable Releases (work on the smallest item that can be released) you won't be descoping things since you'll have already "descoped" it before starting to work on it.

Changing priorities is a symptom of:

1. using prioritization instead of sequencing
2. working on too many things
3. coming from stories and features instead of QVRs.

### *Implementation Methods*

When using timeboxing, manage this by having a product backlog that contains what we expect to work on next. It's essential to have enough work pulled that will take the iteration to complete.

When flow is used, one doesn't need to have as much in the backlog to fill the next iteration. Instead, just have the list of increments / QVRs that are to be done.

### *Sequencing the backlog*

It is important to have the items on the backlog be in sequence of importance, not just listed in priority order. When ordered by priority they could have everything be on top. You might tell executives that not everything could be “A” priority and at best they’d lower a couple to “B”.

But only one can be at the top. Sequencing forces executives, marketing folks, etc., to say which is more important.  We found this a good way to get people to say which is more important. And that's an essential conversation to have.

Also, then the stuff at the top is not as important as the stuff at the bottom.

This also enabled us to deflect requests that would slow down more important things with the question - "why would you have us slow this thing at the top for something that isn't as important?"

This requires, of course, using Quickest Valuable Releases in the backlog - so you can talk about items that had deliverable value.

### *Symptoms of Not Doing This Well*

There is no work to be pulled when the team is ready, so significantly more work is in the product backlog than necessary.

The product backlog can operate the same regardless of whether flow or timeboxing is used. However, if flow is used and enhancements to existing products are being created, it is even more critical to use QVRs so that the QVR will create the bigger picture needed.

## 2.4 Capability: Have a near term backlog from which you pull your work

tbd

## 2.5 Capability: Use pull methods to keep workload within capacity.

### *Objective*

Keep workload within capacity. There are two aspects to this. The first is to keep the general amount of work being focused on limited to the capacity of the team. Timeboxing does this by not committing to more work than what will fit in the timebox. Scrum's sprint is an example of this. When using a flow model, artifacts representing releasable value should be used. MVPs and QVRs are good artifacts for this. If a QVR is going to take more than a week or so it should be split up into small pieces, but done within the context of the QVR.

The other aspect is how this identified work is done. We've already stated that it needs to be done in small, end-to-end slices. But there also needs to be a focus on not having too much work in process. It is important to get even small stories and features completed quickly in order to achieve quick feedback. We talk more about this in Focus on getting feedback while lowering WIP and reducing accumulated risk.

How pull is implemented creates the context for many practices.

### *Why this is important*

It is not possible to accurately predict how much time will be taken to do work. Too much work being done creates delays which creates waste. But not having enough work wastes capacity. Pull methods enable us to manage the amount of work in process without having to predict it accurately.

### *Implementation Methods*

Implement pull with timeboxing by planning the amount of work to be done over the next timebox. This is often an easier concept for teams to get and requires less discipline. But it does increase the queue size of the work to be done.

Flow has us pull MVPs, QVRs, and/or maintenance items directly from a queue, get it done, and then pull the next one. This is the most efficient method of pull but requires a bit more discipline. Items pulled must be broken down into small end-to-end pieces to be built.

### *Symptoms of Not Doing This Well*

New QVRs are opened instead of focusing on finishing those already open. New features are opened instead of finishing other features that are already open.

When planning and coordination of work are not done well, teams tend to get interrupted a lot since people get tired of waiting for their work to be done.

### *How This Capability Manifests the First Principles*

3. Avoid overload
6. Maintain qualities that enhance flow.

## 2.6 Capability: Build in small, vertical (end-to-end), slices.

### *Objective*

Regardless of how big the item you're working on is, you want to build it in small slices of functionality. This has a twofold purpose. First, getting feedback to developers quickly so that they make sure they are

building the correct functionality correctly. And providing customers an opportunity to suggest the next step to build.

### *Why this is important*

Agile introduced the notion of using shorter timeframes in the implementation of requirements. Most people consider this to focus on avoiding or correcting developer mistakes. That is, if a misunderstanding occurs, developers may build it, but can quickly correct it if given feedback that a misunderstanding has occurred.

Much of the work we do is fixing problems due to a delay between making an error and detecting it. By having small stories/slices, we start and finish quickly.

### *Implementation Methods*

Sometimes writing small stories can be difficult. One way to write them is to think of a single scenario and function relating to it. Then write the story to implement that. In some complex situations, you need to use Behavior Driven Development so you can use the construct Given-When-Then, but most of the time, writing small stories is not that difficult.

### *Avoiding Waste by Taking Small Steps and Attending to the Customer Journey*

Many Agilists proclaim you must "fail fast" when building products. While some small failures may be unavoidable, building incrementally often allows you to avoid them.

In my product management workshops, I often encountered people who said customers didn't know what they wanted. I agreed, but I asked if they didn't know anything or if it was just that they didn't know everything. When we analyzed what they knew, we came up with an interesting observation – *customers have more certainty about the basic requirements than they do about the 'nice to have' things they want.* We also observed that when customers got some of what they wanted, they had more certainty about what wasn't given to them yet.

***The bottom line is that customers don't know everything, but they do have certainty on some things – and these are often what you have to start with anyway.***

This means that if we build foundational aspects of the requirements and show them to the customer, they can then adjust what they think they want and be pretty much on target with it.

### *A software product example.*

In the 80s, I was building software for people who worked at the front desks of hair salons. Not the small ones you often see, but ones with 20-40 stylists. The front desks at these places were insanely busy. The standard ways of using menu systems designed to navigate the system were too slow and difficult to use. I remember trying to figure out a better way. Instead of thinking about how the system could be better, however, I put myself in the position of a front desk manager. I wanted to go from client to the appointment book to ticket …. And it needed to be contextual. If I went from a client to the appointment book, I wanted to see the client's appointments I was looking at. Imagining what a user meant by asking to go from one place of the system to another was pretty easy if you knew how people worked in the salon.

Using these insights, I came up with the idea of function keys to navigate. While obvious now, these were not in use back then. F3 was appointments, F4 was tickets, F5 was clients, …. The meaning of these never changed, which also allowed us to put a paper strip above them. People who had never used a computer before learned how to navigate around one in a minute or two. Trying to find a faster way to navigate around the system would have been a lot harder than just seeing how people wanted to navigate in the first place.

### *Scorecard for "Build in small vertical (end-to-end) slices"*

The progression of work is:

1. Not aware it should be done this way.
2. Aware but not doing it often.
3. Doing it when easy
4. Always doing it
5. Always doing it and usually validating each slice

### *Symptoms of Not Doing This Well*

Discovering errors in building and/or the requirements late.

Overbuilding the item - if it had been built in stages it would have been discovered not all of it was needed.

### *Risks of Not Doing This Well*

- building more than needed
- taking a long time to complete
- delays in feedback.

### *Exercise*

Take a few minutes to consider a time when you didn't build in small vertical slices so that you could get feedback quickly? What happened? What errors were caught late?

Take a few minutes when the customer journey was not attended to? Did the customers find the product/software easy to use?

<Amplio_Scrum_end>

## 2.9 Capability: Use automated testing to eliminate waste

### *Objective*

Most developers believe they spend a lot of time fixing bugs. However, if you ask a developer to trace their efforts, you'll see something like this:

1) Time spent trying to reproduce the bug
2) Time spent "fixing" the bug
3) Discover it caused another problem
4) Time undoing the fix
5) Time spent again "fixing" the bug
6) Verifying the fix.

The reality is that they spend more time finding the bug than fixing it. This is not semantics. I say this because if the bug were identified immediately, there would be almost no time "fixing" it right after putting it in. This does not even count the added waste from the bug's impact on others.

### *Why this is important*

Not having automated tests creates a tremendous amount of waste. Both for the people who had to correct the errors and those who assumed the code was working.

### *Implementation Methods*

The most effective way to implement automated testing is to use a testing framework.

### *Symptoms of Not Doing This Well*

- Code being fragile.
- Developers believe they spend a lot of time fixing bugs.

# Sprints, Daily Timeboxing, Flow, and Cadence.

Timeboxing is looking out to the future to see how much work you can do in a specified period of time (what we call the timebox). The most common type of timeboxing is the sprint (in Scrum) or an iteration (in XP and other Agile team approaches).

Timeboxing can be done as short as a day. This is called “daily timeboxing.” This creates focus while avoiding over-analyzing. Scrum defines sprints as “one month or less,” but in practice, they must be at least a week in length to allow for sprint planning, a sprint goal, a demon, and a retrospective. In this chapter, we’ll consider sprints to be 1-4 weeks in length.

Flow is when items of value are pulled by the team and done to completion, at which point another item is pulled. Flow may be done with many items in play or only one. This latter approach is called “one-piece-flow,” which is often popular in manufacturing but not recommended as a general rule. When using flow, it’s recommended you timebox the item being worked on. You can add more time if needed, but this gives you a warning that things are taking longer than expected.

Cadence is like the beat of a drum - bump bump bump bump. It is done to coordinate events such as planning, demos and retrospectives. Even if one is doing daily timeboxing a cadence is important so people know when to get together for regularly spaced events.

Teams should usually start with daily timeboxing or sprints. There are situations where a true flow system is ideal but I suggest it only for mature and disciplined teams. When doing a pure flow system you have to create your own timebox on how long you’re going to work on something.

## Sprints or Daily Timeboxing

The decision to use sprints or daily timeboxing as the main driver of your work is an important one. It is often a good decision to make early on as it creates the context for many other practices and will be the basis for how roles cooperate with each other. However, it should not be considered a one-and-done decision. Many times starting with sprints makes the transition to Agile easier. Once basic Agile practices are underway, going to a daily timeboxing model can further the transition. However,  sprints may be better to start with in situations where it will be less disruptive.

In this section, we’ll discuss how to make this decision.

Using sprints  is when a team decides what they will build over a particular length of time. It is the common practice of Scrum and SAFe to use sprints. Scrum tends to use 2-4 week sprints. SAFe uses 2 weeks sprints at the team level within the longer (2-3 month) program increments.

When sprints are used, it typically also defines a cadence of coordination and other events. Cadence means a set period of events. It can be thought of as a drum beat which we work around. Scrum tends to use the end/start of a sprint as a cadence. Kanban allows the cadence to happen an any time, but often coordinates things in the same way.

If you're going to use sprints, it is recommended that you do flow within it. This provides for a straightforward way to manage work in process. This speeds up the completion of individual items speeding up feedback and the delivery of value while lowering the creation of waste.

The choice of using sprints or daily timeboxing is not always a clear one. Many factors are often involved. Some of these factors predispose a team to choose sprints or daily timeboxing. Many times, however, this choice will best be made after going through the required objectives and seeing how to best meet them.

When it's not clear, it's often best to choose the practices that appear best and make a choice after going through the list of roles, events, artifacts, and rules and using the practices chosen. Regardless of which practice is chosen, the focus should always be on value delivered and keeping work in process as low as possible while not impeding workflow.

However, sometimes the choice is clear right up front. These are the factors that may have you decide upfront which approach to take:

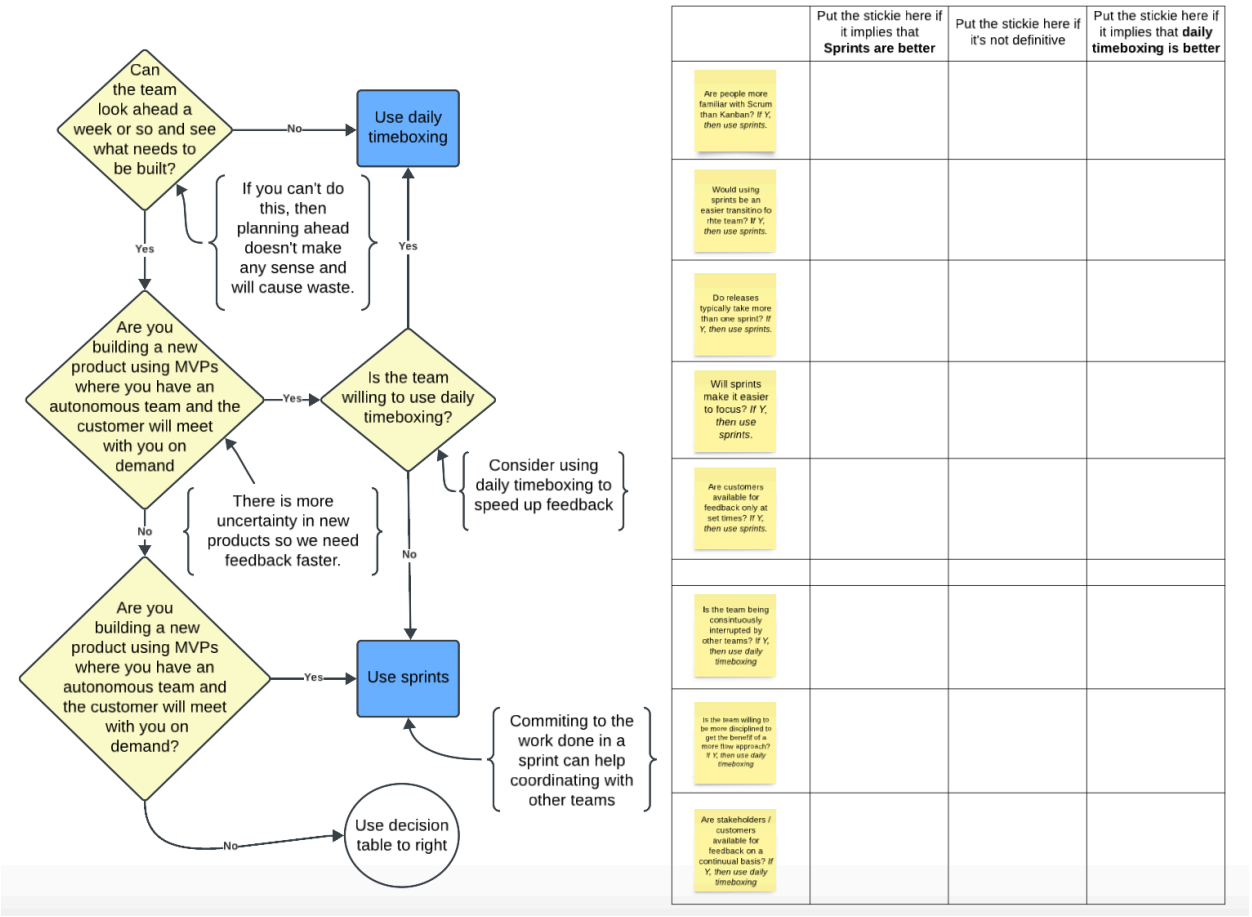
## A few situations that compel us to do one or the other

**1. Do we need to integrate with other teams at a set time and we're not doing CI/CD?**
While you can coordinate with flow, sprints provide set dates for integration. This is not as important when doing CI/CD. *If this is true, it is a compelling reason to use sprints.*

**2. Can the team look ahead a week or so on what needs to be built?**
*If you **can't** do this then it's a compelling reason to do daily timeboxing.*

**3. Are you building a new product using MVPs?**
If you're using MVPs it's important to not plan ahead beyond the MVP which mat take only a couple of days. Doing so would mean you waste time on the work you analyzed but are now not going to do because of the feedback you got when you showed the customer the MVP. *This indicates daily timeboxing will be more efficient but the team must be willing to use it. Otherwise use sprints and bear the extra cost.*

*This is represented graphically in the following picture.*




## If a decision has not been made

There are a number of factors that indicate whether daily timeboxing or sprints would be better but need to be considered with other factors. Use the table to help see which would be more useful.

## Reasons to use sprints

**1. Are people more familiar with Scrum than Kanban?**
Using something people are already familiar with is often a good idea.

**2. Would using sprints be an easier transition for the team?**
Look to see if using sprints is closer to what the team is doing now?

**3. Do releases typically take more than one sprint?**
If releases usually take more than one iteration, then having a short term focus can be helpful.

**4. Will sprints make it easier to focus?**
If it does, use it.

**5. Are customers available for feedback only at set times?**
If you can only get feedback from customers at the end of the sprint, sprints can provide clearer goals on when to be done.

## A sprint anti-pattern.

Some Scrum proponents claim sprints are easier to start with. This is not inherently true and is often only because those saying it don't know how to do daily timeboxing well and therefore can't introduce it smoothly.

## Reasons to use daily timeboxing

**1. Are you continuously being interrupted by other teams?**
If you're a team that gets interrupted a lot, for example shared services, planning may be a waste of time. Of course, you can allocate a certain percentage of your time to the plan and limit your plans for that.

**2. Is your team willing to be more disciplined to get the benefit of a daily timeboxing approach?**
Daily timeboxing takes a little more discipline. Your team must be willing to work that way.

**3. Are stakeholders/customers available for feedback on a continual basis?**
If stakeholders and customers are available on a continual basis then it behooves your team to take advantage of that.

## Daily timeboxing anti-pattern

Daily timeboxing can be problematic if the team can use sprints to coordinate with other teams more easily.

## The difference in how sprints and daily timeboxing show up in work

When sprints are being used, especially if it was selected because the teams were not experienced, there is a tendency for teams to open too many stories at the beginning of the sprint. The focus is on getting the sprint plan accomplished not on managing work in process. While there may be an intention to release part of it early, the commitment is to the sprint, not the early release. This often results in many stories being open throughout the sprint and many get closed towards the end of the sprint. This is not an efficient way of working and delays getting feedback to the end of the sprint.

When daily timeboxing is used the focus is on completing the open stories. This focus tends to prevent too many stories from opening and increases cooperation. Note, however, that this attitude can be used even when sprints are used – and it should be.

## A potential challenge with sprints and management

A common challenge teams face is management asking to insert a new request into the mix. This is very easy for them to justify when sprints are being used because it just adds one more thing to the pile. Working a little harder is expected. When daily timeboxing is used, work is clearly sequenced – we work on items in order of value. It is easy to show management that injecting this new piece of work into the sequence slows everything else done. That doesn't mean management will abide by this, but it does make it clearer that there are consequences. This also sets the stage for making work interruptible, a useful technique.

## This is just a preliminary decision

As you work through this book, you may learn concepts that have you change your decision. There are times when how you work changes as well. For example, you may start working on a new product and decide you want to use flow. After some time, you understand what needs to be built and shift into a time-boxing mode. After it's released, you may be more in maintenance mode and shift back to flow. The critical thing, however, is to stick with the method you chose during the period you chose it for. Also, even when you use sprints, you should use flow within the timebox.

## How Sprints Can Create Waste

*Question to students: When do customers know what they want?*

*Answer: After we show them what they don't want.*

*Question: So when do we want to show them something?*

*Answer: As soon as possible.*

There are tradeoffs in using sprints. It's quick to pick up and useful as a start. Using sprints can be the best approach short and long term. But it has some inherent disadvantages in many situations. These are:

1. In the middle of the sprint, you may discover that the rest of the work to do isn't as important as what you had planned. You can, of course, swap the work, but there is a likelihood that the analysis time you spent on the pushed-out work didn't need to be done.
2. If you find you come to the end of the sprint and haven't started some stories, then you've wasted time analyzing them.
3. If you decide not to carry them over, they may need to be re-analyzed by the time you do get to them.
4. With more stories available to do, people will likely open more stories than needed. This challenge can be overcome with discipline by incorporating flow into the sprint. But if you do that, you don't need to plan the sprint – just take the essential items as they come.

### *If you're ending up having stories not finished at the end of a sprint, shorten the sprint.*

Many Scrum teams face the problem of having stories unopened at the end of a two-week sprint. This represents waste because even if the stories are carried over to the next sprint, the analysis is two weeks old. And they may need to be reanalyzed as well. Analyzing two weeks of stories takes more time as well. You also can't take advantage of what you've learned from previously done stories. Using 1-week sprints will mitigate this problem. Quicker building, quick discovery. Shorter-term meetings. Of course, if getting customers and product owners together is impossible every week, you may need to make sprints longer.

### *If you're fully doing flow in the sprint you will discover you only need the sprint for two things*

Sprints are useful for creating a vision. But a focus on value and using Quickest Valuable Releases (QVRs) is often better. QVRs are the next releasable chunk of value. Focusing on QVRs creates alignment across the team or teams involved.

Cadence can be used with or without sprints to create set times that people come together for retrospectives and reviews. If you have everyone able to meet on demand you can coordinate not with cadence but at the end of a QVR being built.

Since you can release when enough value is present, you can drive from what you are going to release. If you've selected iterations and now have a sense of how to do flow, consider this:

Since you want to release when value is ready, should you focus on the sprint goal or on what you can release first within the sprint? Which will speed up the release?

If you focus on the release and do flow in the sprint- what's the purpose of sprints?

## Scrum Sidebar: Warning. If you are doing Scrum, don't just abandon sprints because you are having challenges with them.

Sprints serve a purpose. Just abandoning them is not a good idea. Flow can provide an alternative set of practices, but you must:

1. Have small batches
2. Focus on completing what you start
3. Manage work in process and accumulated risk

It typically takes more discipline to use daily timeboxing than sprints, but the savings can be significant.

Doing this properly changes "we do Scrum *but* we don't do sprints" to "we do Scrum *except instead* of sprints, we use daily timeboxing."

*A case study where flow works best, and also illustrates the risk of prototyping.*

When you are building something you are absolutely sure of, you can often go off and build it and then come back to have a review by the customer. But this does add risk as even when you are sure you understand, you often won't.

In the mid-90s, before I was consciously doing Agile, I had an important client that wanted a report. I had a history with them that they'd ask for things only to tell me later that I hadn't gotten it right. So when they gave me their requirement I decided to mock it up with their own data.

The client was a large spa, and they were asking for a services report by all their technicians (i.e., hair stylists, masseuses, colorists, …). They wanted both the detail by type of service for the client and also totals by department. It was an involved report but only looked to take a few days to do. So I got confirmation with the mock-up, created the report, and showed them the results feeling very confident I had nailed it.

But I hadn't. They didn't like it and told me what I had done wrong. I admit I was pretty puzzled – they had seen the mockups and I had given them a report based on that. These people were not stupid, nor not caring (they were paying me by the hour). What happened?

On reflection, I realized that seeing a report's results is not the same as running the report yourself. They looked at the mock-up report, but didn't truly understand it.

A few years later when I got into XP and learned about the emergence of requirements, I realized had I done the report in a small stepwise way, getting daily feedback from the customer, I would have saved the time required to mock it up and would have gotten it the first time. I should have created a report with stylist totals and gotten confirmation that was right. Then I could add revenue by department and get confirmation again. I could have repeated this each time, adding a little function, validating it, and then moving forward. This would have worked. And I've done this since and have seen it works..

Sprints give the illusion that we know what to do for a few weeks. It hides the waste caused by not having more feedback with the customer / product owner. While it may not be possible to meet as frequently as useful, at least be aware there is a cost to that.

# Capability DW3: Backlogs

## Shared backlogs

This section discusses the use of a shared backlog. It also illustrates why aligning teams takes less effort than coordinating them.

Multi-team Agile tends to take one of two approaches

1. Scaling Agile from the bottom up is an expensive approach because it requires more coordination than a holistic one that created alignment would. this is often typified by the Scrum-of-Scrum approach.
2. Demanding Agile from the top is inefficient because it demands people work a particular way..

But these are not the only two options. One can align around creating value. If we use QVRs, we can feed multiple teams with a common backlog. This creates alignment because different teams will pull from the same place and automatically work in a coordinated fashion. This automatically has people work together.

Coordination is the cost you pay when your teams aren't aligned.

“The idea of having problems and then solving them is not as good as taking a perspective that avoids the problem in the first place.”

“One of the reasons Scrum is so expensive is that you have to hit the problem and then solve it. Why even hit the problem?”

Coordinating Teams with Backlog Management

Tbd (rewrite this, and put capability in standard format).

## 5.3 Capability: Aligning market solutions to teams

### *Objective*

Organize teams to work on one product so that people are primarily in one value stream. Since this is not usually easy to start with, this capability provides a way to move from where an organization is to the structure it should be.

### *Why this is important*

Most organizations have people working in multiple value streams. But just disengaging them is difficult. This capability provides a step-by-step method of going from invisible value streams to autonomous ones.

One can think of value stream management as having three aspects to it:

1. What goes into the start of the value stream
2. How the people in the value stream are organized
3. The workflow in the value stream

We have already talked about the importance of MVPs and QVRs as to what goes into the value stream. How the people are organized (what's called the value-creation structure) interacts significantly with the workflow of the value stream. These two factors (structure and workflow) interact with each other and should improve together.

### *Implementation Methods*

Companies find themselves in varying degrees of autonomous value streams. Here are several going from worst case to best case:

1) Work is **not visible**. Value streams can't even be seen.
2) Work is visible but there is **no collaboration** with the value streams
3) **Stakeholders have been identified**. Value streams try to work on the most important work but there is no collaboration
4) Value streams **coordinate** with each other
5) Work is **planned** to help the value streams work together
6) **Shared backlogs** are used to **align** the work
7) Align teams to market

We will walk through these cases. We will describe what improvements are made at each step and how one step compares to the next. Although we are going through these a step at a time, do not feel you have to. Skipping steps is good and gets better results faster when you are clear about what to do.

#### I. No visibility

This is the worst-case situation and fortunately, not very common. In this case, work comes in and gets delegated to one or more teams to work on. However, no real effort is made in tracking the information. The result is people are working on lots of things but no one is sure what all of the work is. It makes it difficult to coordinate things and people are always being interrupted on an adhoc basis. This is depicted in Figure 4.1

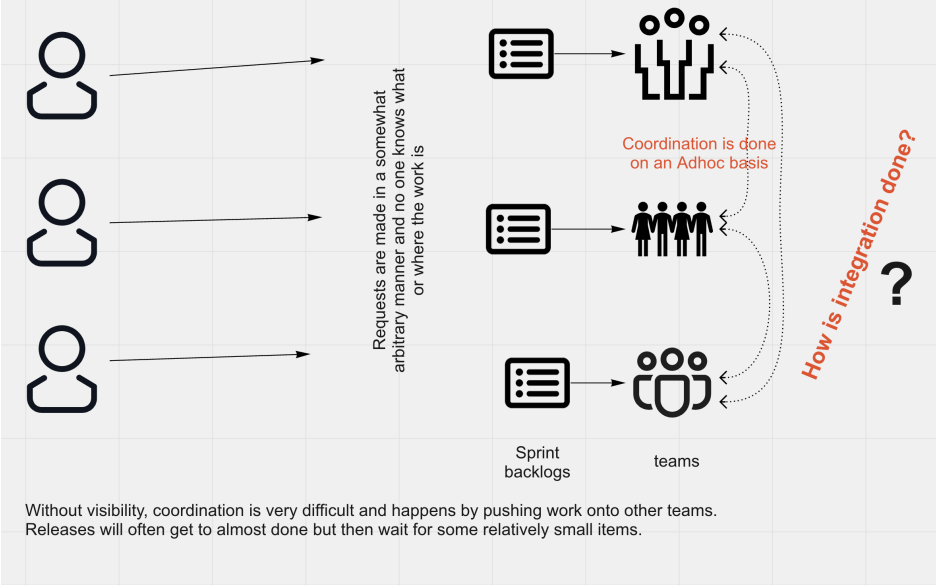



Figure 4.1. Work comes in, is given to teams, and visibility of it is lost

### II. Visibility but no collaboration

In this case, the work is being tracked but everyone is working independently of the other. This is an improvement in that we can at least see the work. But collaboration is done on an Adhoc basis. The main action to move from no visibility to this state is simply having an intake process that all work must go through.

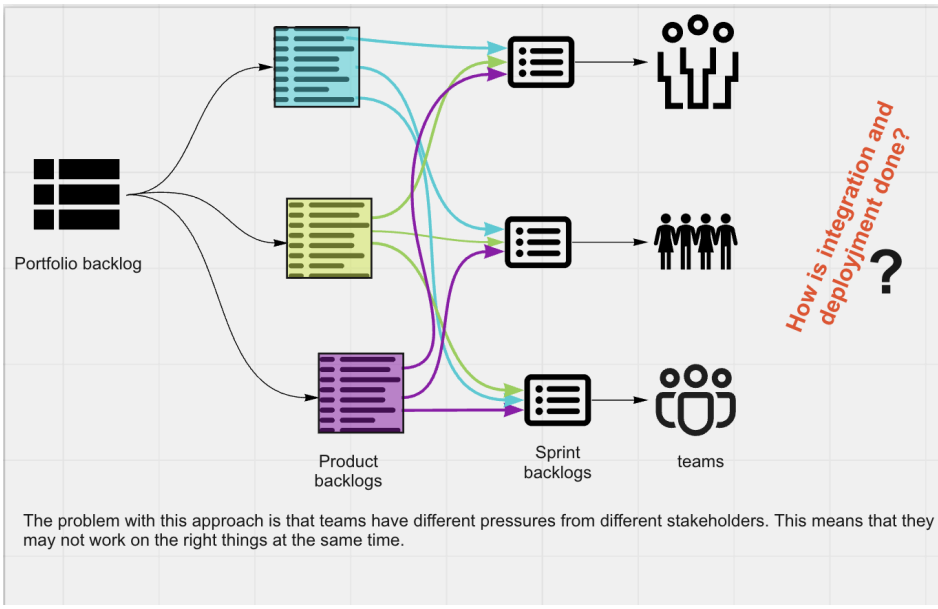



Figure 4.2 Visibility without collaboration.

### III. Stakeholder identified.

An easy next step is to identify the stakeholders and have a general sequence of the work to be done. Everyone is still working independently, but at least there is a general awareness of what's important and what's not. At least some of the time.

This is depicted in Figure 4.3

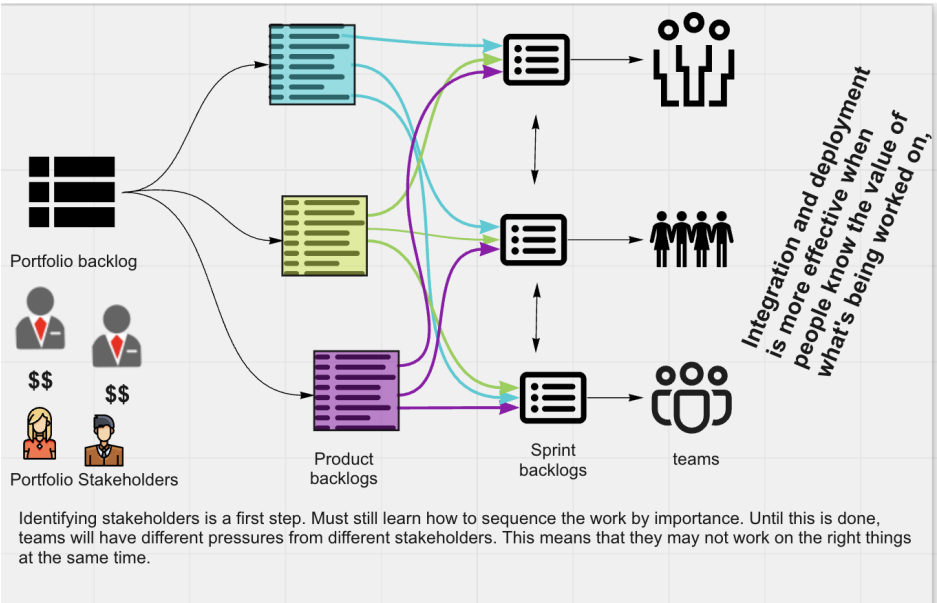



Figure 4.3. Stakeholders identified but little collaboration.

### IV. Value streams coordinate with each other

At this point, it's straightforward to get the teams to start collaborating with each other. This can be done with value managers talking to each other to see what needs to be done. While the teams may plan separately, they are at least working on a coordinated list. The flow looks like what was in the earlier step, but the coordination work makes it have fewer interruptions. This is shown in Figure 4.4.

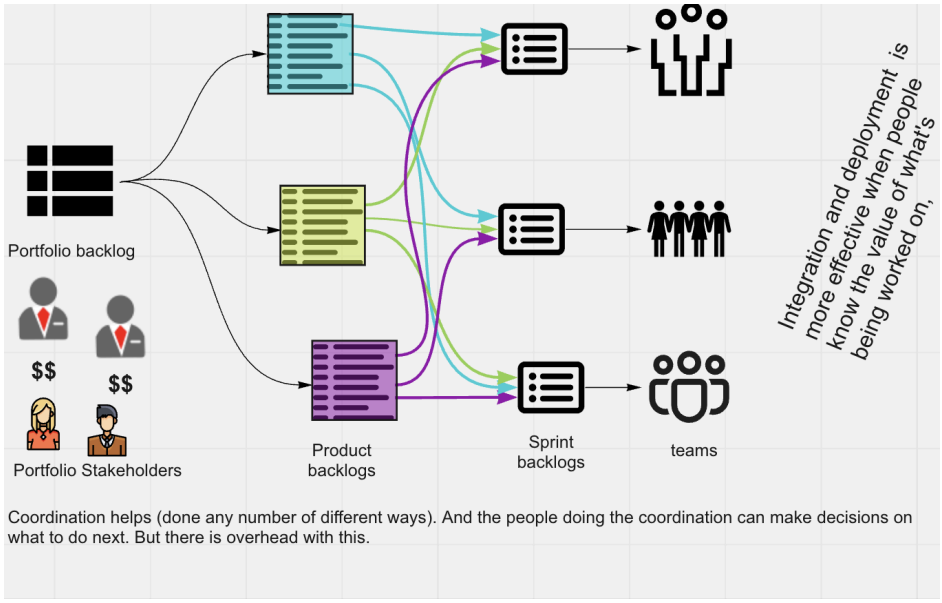



Figure 4.4. Work is visible and coordinated.

### V. Work is planned to help the value streams work together.

In this case, the teams plan together to coordinate the work to be done. Backlogs for the different teams are somewhat sequenced in the same order across the organization.

This is shown in figure 4.5

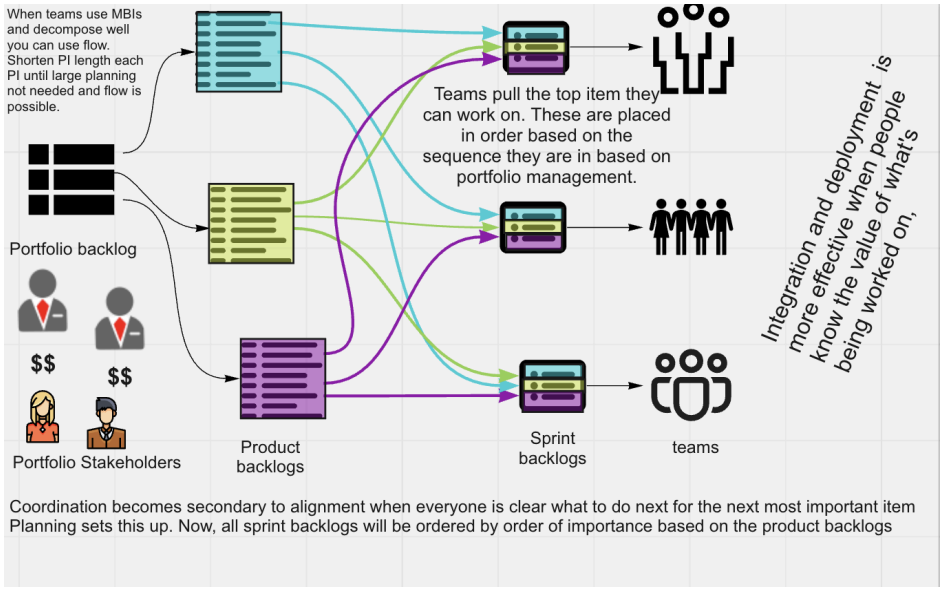



Figure 4.5 Work planned together. Separate backlogs in a coordinated sequence.

### VI. Shared Backlogs

Pulling from a shared backlog can be very useful. When this happens we can be assured that all of the capacity needed to build something will be available when it is pulled. This is shown in Figure 4.6.

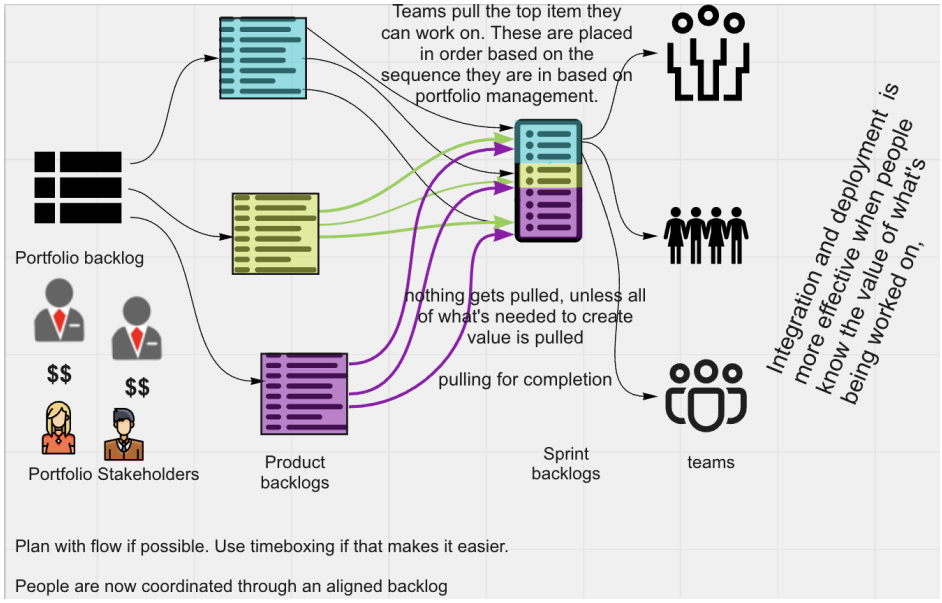



Figure 4.6. Teams pulling from a shared backlog.

### VII. Teams aligned to their markets

In this case, each product/market has people dedicated to building what's needed. And they have all of the capabilities needed as well. This is the ideal situation and is shown in Figure 4.7. Reorganizing to this structure right away is great. But most people don't have the capacity balanced in the way they need to get here from the start. It's often worth looking to see what capabilities are needed and then cross-train to achieve it. This can accelerate your getting here more quickly

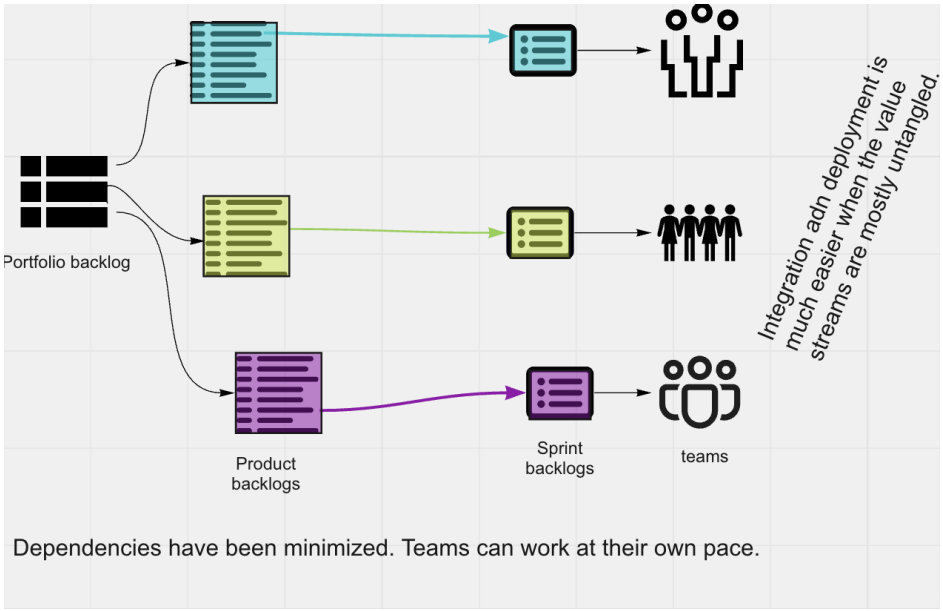



Figure 4.7 Teams aligned to markets.

### VIII. Micro-enterprises

The next level would be each market is separate from each other with its own development group. This has completely independent value streams.

## *Symptoms of Not Doing This Well*

symptoms of not improving here are:

- work does not improve
- work may start being pushed onto teams
- teams interrupt each other
- severe multitasking

Value creation structure at team level (include borrow, share team members, FSTs)

Feedback at all levels

Systems Demo

Daily pivoting and Retrospections

# Shared services, ops, …

Shared services

Other parts of organization

Vendors

Ops

Support

Dev Ops

## Capability: Use DevOps when applicable

### *Objective*

DevOps is a set of practices that combines software development (Dev) and IT operations (Ops). It aims to shorten the systems development life cycle and provide continuous delivery with high software quality.

### *Why this is important*

Value delivery starts with the customer and is not complete until the customer can consume what has been delivered. If the team is not doing the delivery itself, it must work with those that are.

### *Implementation Methods*

DevOps can be implemented in several different degrees. At a minimum, the development team should work with whoever is going to deliver and support the system in the following ways:

1. Ensure ops and support teams have complete visibility on what is being worked on and when they will need to get involved
2. Include the ops team in your work if they believe that would be useful
3. Have one of your team members be with the ops and support teams when the work is handed off

### *Symptoms of Not Doing This Well*

Ops and support not being ready when the work is ready is a symptom of a lack of DevOps cooperation. Surprises by either team are also symptoms of this.

### *How This Capability Manifests the First Principles*

2. Visibility of work
5. Understand the way we're working.
6. Maintain qualities that enhance flow.
7. Organized for efficiency.

# Releases

## MVRs

## Support experience

# Value Stream Management

## Capability VSM1: Create visibility of work and workflow.

### *Objective*

Making work visible helps people see what's being worked on and coming their way. This helps cooperation.

Explicit workflow is nothing more than making the agreements between team members explicit. This does not mean that these agreements are immutable, just that they are the best way we have of working at the moment. This helps cooperation when everyone knows what's expected of everyone.

#### Have people understand how they are working together (personal sidenote)

Some people believe that having explicit policies are destructive and that there is a danger that people will  start following them instead of thinking for themselves.

Having workflow be explicitly stated merely means we've talked about it. It could even be that we don't have an agreement. It does not imply we should devolve into following a process. Instead it keeps alive how teams work together. This understanding keeps a focus on what works by having people talk about it instead of relying on a guide to tell us what to do.

This is a key aspect of Flow, Lean, and the Theory of Constraints, we need to state what we're doing and why. This provides us the ability to change it and see what happens. The better understanding also enables people to join the team with less effort.

This difference underscores a deep division in the mental models of Scrum and Lean. For more on this see Types of Processes by Don Reinertsen written in October 2009.

### *Why this is important*

Both are required for good teamwork.

This visibility is required for both what is to be released and the features and stories that make this up.

Keeping people informed about what is happening:

- Increases collaboration
- Saves time if someone needs help
- Informs people if some problem is going to impact them.

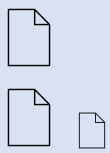
### *Implementation Methods*

Make work visible to enable people to see what others are working on and if their involvement will be needed.

Make workflow visible so that people can see the agreements between the different stages of the work. This is done with an explicitly defined workflow.

#### Implications for the Team Board

Simple Scrum type team boards often look something like this:

| Product Backlog | Sprint Backlog | In Process | Needs Testing | Done |
|---|---|---|---|---|
|  |  |  |  |  |

This provides us with the status of the work but doesn't describe how it moves from left to right. Explicit workflow would add a column for each type of work. This doesn't imply a handoff, just that the story is being worked on differently. It is essential to realize that you don't follow what's on the board; instead, the board reflects what the team considers to be the best way of working.

### The Need for Daily Huddles as an Anti-pattern

The need for daily huddles is often an indication that information is not being shared on an as-needed basis as well as there is no other method for dealing with challenges and other impediments when they come up. This is often due to the newness of these practices to teams. Capability 3.4, having an agreement on handling feedback and challenges while also keeping people informed about what's happening has a section on how to run daily huddles. While mandated as part of Scrum, they are an option in Amplio as long as their objectives are achieved in another way.

#### *Symptoms of Not Doing This Well*

When work is not visible, people will be surprised when it shows up for them to do it. It also makes it harder to trust each other since requests are unclear.

When the workflow is not explicit, people will work differently, and there is no baseline for improvement.

#### *Relating this capability to Scrum*

When you have an explicit workflow on the board, and it is clear what people are working on and what people will be working on next, the board can serve the purpose of a status report. This means that the common status reporting methods of Scrum with three questions should not be needed. Instead, if you do a Daily Scrum, you can focus more on pivoting as necessary. See "5.3 Capability: Have an agreement on handling feedback and challenges while also keeping people informed about what's happening. (AS)". The Scrum Guide removed the three-question approach to the Daily Scrum in the 2020 Scrum Guide to focus more on:

- Inspecting progress toward the Sprint Goal and adapting the Sprint Backlog as necessary, adjusting the upcoming planned work.
- Improving communications, identifying impediments, promoting quick decision-guiding, and consequently eliminating the need for other meetings.

The Daily Scrum is often useful for teams new to Scrum as it enforces discipline in keeping everybody informed.

### How to tell if you have a good team board

A team board should show both the work being done and how it is being done. In particular, it should make clear any explicit workflow agreements members of the team have made. A team board should enable someone not familiar with the team to understand how the team is working by first looking at the board for no more than 15 minutes and then asking questions about it for no more than another 15 minutes. If, after this time, they do not know how the team is working, the work and/or workflow agreements should be expanded on.

The importance of a quality team board is to ensure people know how they are working. It provides the basis for documenting the agreements the teams have. While the board is explicit, it's important to note that the board is not to be followed, rather it is the documentation of what the team's agreements are.

Perceived safety

People following agreed upon across organization agreements

External governance issues

# Roles

## Lean Leadership and Management

## Value Stream Manager

## Business Architect

## Value management office

## Enterprise Architect

## Product Managers

## Product Owner

## Team Lead / Scrum Master

## Team Related Capabilities

# Roles

Now that we know what we have to do, we should specify the responsibilities in the form of roles that are needed. A person does not necessarily fill a role. While sometimes one person will fill a role, very often a role will be split across more than one person or two people will share the responsibilities.

4.1 Decide who is the Value Manager ( Product Owner in Scrum) DTA

4.2 Developers

4.3 Decide on the responsibility of the team coach DTA

4.4 The role of management

## Capability: Decide who is the Value Manager (Product Owner in Scrum).

*Note for users of Scrum: It is not suggested that you change the name of the product owner to value manager as that may cause confusion.*

The **value manager** is the role of deciding what value should be created. Value is based on both the stakeholders and the customers of the company. The Scrum name for this role is “product owner.” Unfortunately, the way the product owner role is defined and typically implemented in Agile is not as effective as it could be. The name change is important to not carry the connotation of the Scrum product owner.

### *Why this is important*

What is built is always an integration of business and customer needs. Having a well-defined backlog to pull from is critical to facilitate the team working on the most important items.

### *Implementation Methods*

This responsibility is independent of whether an iterative or flow-based approach is being used. Options include:

- Have a value manager outside of the development team be responsible. This enables this person to talk to all stakeholders and avoid the team getting conflicting priorities.
- Have the team be responsible. This is often the best option when a technical product is being built and the team understands best what features are important.
- Have the value manager and team be responsible. In this case the product owner is responsible for the objectives as defined by the stakeholder while the team is responsible for the functionality to achieve those objectives.

### *Symptoms of Not Doing This Well*

If you don't have this responsibility well-defined, people will work on different things based on their own priorities. It also enables people outside the team to go to individuals and tell them to do something that's been decided for personal reasons

Very often the value manager becomes a barrier to the team’s ties to the customer. While the value manager has the responsibility for setting priorities, the role should not become a barrier between the team and stakeholders.

References: To see more about what a value manager should be, see Tom Gilb's Value Agile: Agile As It Should Be. To get a better sense of the shortcomings of the product owner, read Mark Schwartz's The Art of Business Value.

> Scrum Sidebar: Why Scrum Teams Often Become Feature Factories
> TBD

## 4.2 Capability: Developers.

Developers are those people who create value in the product or service. They should be a team, not just a collection of people working together. This requires mutual trust and an alignment on what they are creating.

Have all of the capabilities needed to create value quickly in a predictable, sustainable, and high-quality fashion. This does not include dependencies between other teams doing work but relates to when people's capabilities that are shared across several teams are needed, for example, business intelligence or UX. Development teams should have all of the skills required to produce the software being developed. Ideally, this will be for the software being released in its entirety. In larger organizations, it may be just a component or module of what is being built. While it is virtually always best for a team to be cross-functional (meaning they have all the skills necessary to implement the PBIs being built), it is often the case that some capabilities are shared with other teams.

The way these people are organized is discussed in the Organizing Teams and Backlogs for Effective Value Streams section.

### *Why this is important*

In order for people to work together well, they must all identify as working together. Obviously, if you don't have people doing the work, it won't get done.

### *Implementation Methods*

Wherever possible create cross-functional teams. When not possible, see if teams being dependent upon can provide team members needed for the necessary time. If all capabilities cannot be made to exist in the team then manage the dependencies on the shared capabilities via a Kanban board so that the delays created by using people outside of the team can be known and managed.

### *Symptoms of Not Doing This Well*

If you don't have a well-defined team, people will not collaborate well with each other.

## 4.3 Capability: Decide on the Responsibility of the Team Coach.

Being clear about who is responsible for guiding team improvements, managing schedules, and interacting with those that are outside of the team.

### *Why this is important*

Teams must be continuously improving. Some people can do this on their own, others get their heads down in the work. Without clarity on who has the responsibility to be looking for improvement and to lead any of the ceremonies the team has decided on, teams can get into disarray.

Development team members tend to get heads down on their work. Someone needs to be attending to continuous improvement as well as people following their own agreements with each other. Sometimes team members will get stuck but want to push through when an outside observer can notice and help them

### *Implementation Methods*

1. Have a person not doing development work be the coach. In this case, the coach doesn't tell the team what to do but may make suggestions and should educate the team on why.
2. Have a person doing development work be the coach. In this case, the coach doesn't tell the team what to do but may make suggestions and should educate the team on why.
3. The team self-coaches.

1. Have a Team Agility Coach.

Having a Team Agility Coach be responsible for shepherding the team, creating a trustful environment, facilitating team meetings, asking difficult questions, removing impediments, making issues and problems visible, keeping the process moving forward, and socializing Lean-Agile within the greater organization. They will also:

- Protect the team from interruptions to whatever extent possible
- Be a coach to the team in assisting them to get their work done in a more effective and efficient way
- Be responsible for the non-work artifacts that the team produces and uses

2. Have an agreement that the team coaches itself.

Teams can self-coach. It doesn't need to be everyone involved in the self-coaching. If this option is taken it is important that it be validated that it is working

### *Symptoms of Not Doing This Well.*

tbd

## 4.4 The Role of Management.

Management has several roles in Agile. These include:

- continuously improving the environment within which the teams work
- ensuring that teams have the information they need to do their work
- ensuring they have the right resources to get their job done
- helping their people with career and self improvement opportunities
- collaborating with other managers as needed to do the above
- doing whatever's necessary for team success

Management's attitude needs to include recognizing that those doing the work know more about the work being done. While they need to trust their people, they should also not let them go off the rails, so to speak.

## First, ignore all the negatives you hear about management in Agile

I don't buy into many Agile consultants' common complaints about management. In 20+ years of consulting, this has not been my experience. At Net Objectives, we had many managers "get" Agile and lead the improvements. What you must remember is the attitude

Consider this - if consultants ignored you, didn't respect you, talked down to you, called you a chicken, and explicitly barred you from important meetings, all in the name of Agile/Scrum, what would your opinion be of it? Acknowledge that and do something different.

When the people who are pushed outside their comfort zone don't have the experience to judge whether or not the causes and effects underlying our suggestion have merit—causes and effects that are in direct contradiction to the one they assumed—is an explanation enough to cause them to invest the considerable time and efforts needed to launch, monitor and analyze a test?

## The Role of Management in the Agile Space

"A system must be managed. It will not manage itself. Left to themselves, components become selfish, competitive, independent profit centers and thus destroy the system … The secret is cooperation between components toward the aim of the organization." —W. Edwards Deming

Management plays a vital role at all scales. While being a servant leader is important, that must come in the context of the big view of the organization. In essence the purpose of leadership is to facilitate the movement of an organization to provide more value to its customer in a way that is consistent with its strategic vision.

This respects the ability of workers to self-direct and self-organize while creating an effective ecosystem within which they can work. This is accomplished by MIDDLE management looking UP the value stream to see the vision of the organization. They then look DOWN the value stream to see what they need. Management then works with those doing the work to create an environment within which the vision can be manifested. This is called Middle-Up-Down Management.

- Middle-Up-Down Management is described in Toward Middle-Up-Down Management: Accelerating Information Creation, Nonaka (1988). Although Nonaka co-authored the New New Product Development Game (1986), on which Scrum is based, Scrum ignores this foundational aspect of Lean management.
- It balances the imperative to 'process' information in a mature organization with the need to create information in a fast-moving, learning organization
- Middle management becomes the driving force for organizational change to meet the strategy of the business

In any organization, the layer of middle management, those who sit between the topmost strategic level and the level of the "Gemba" (the place where the work is done), can be either the largest barrier to change or a true catalyst for improvement. In many Agile transformations, the role of middle management is not only undefined, they are cast as the antagonist to any productive change. This is counterproductive.

We believe that management, at both the strategic and middle/operational levels, has a key role to play both in transformation and in the ongoing success of the Lean-Agile organization. We subscribe to Nonaka's vision of middle management as the place where strategic direction meets creative response. It is where strategy is shared downward and real-world learning is shared upward.

Middle managers fulfill this role by working with those doing the work. They provide the overall view and can facilitate/coordinate moves beyond an individual team's capability. It's a synthesis of the overall structure and local function.

In our model of transformation, the transformation itself becomes a part of the work to be done. Middle management becomes the focus of getting to done and is the 'owner' of the new forms of leadership and work that it unleashes.

Art of Action: How Leaders Close the Gap Between Plans, Actions, and Results. Short recap of this great book by Stephen Bungay.

Turn the Ship Around: A True Story of Turning Followers Into Leaders. David Marquet creates the leader-leader management paradigm.

Team of Teams: New Rules of Engagement for a Complex World. Gen. Stanley McChrystal. This explains how true servant leadership is creating environments for teams to work well.

### *Why This Is Important*

Managers can see things across the teams that individuals on the teams can't.

### *Implementation Methods*

### *Symptoms of Not Doing This Well*

TBD

Get the right people.

Make sure they know what they need to know to make good decisions.

Reinertsen 777
-Being in the right roles?
-Being in the conversations that empower them?
-Having access to customers, stakeholders?
-Training opportunities
-The right teams (team topologies)
-Long-term career development?

## Amplio Sidebar: Create a way for people to give you anonymous feedback.

*A case study by Al Shalloway*

No matter how sincere you are, people may doubt your integrity. This often has nothing to do with you or even the individual. They may have been in a company where people didn't walk their talk.

When I formed Net Objectives one of the values was to never do something that wasn't in the interest of the client. I had several times where we were asked to do something and I always walked away from them if I couldn't get them on the right track.

Part of the onboarding process included explicitly stating this.

I remember, however, a time when we got a new employee and after he'd been working with us for about six months I found out something he had done at a client that wasn't in their best interest. I called him into my office and asked him why he did it? He said because I had told him to.

I didn't get into an argument with him. I know I hadn't told him. This had been a cardinal rule of mine and I had never broken it. But I recognized that *his* experience was that I had told him to. Miscommunication is common. And what is heard is not always what was said.

I didn't even have to ask him why he didn't challenge me on it though. I had told him when I onboarded him I did not want him to do anything that wasn't in the best interest of the client. I would have hoped he'd have come to me but I understood why he didn't. New employees often feel their job may be in jeopardy asking such questions of the CEO/owner.

This is a perfect example of attending to the system and not requiring people to overcome it.

The need was to create a process that didn't require trust.

It was a straightforward solution.

We had an employee that everyone liked and trusted. He was just that type of guy. I set up a process where if someone thought I was doing something out of integrity then they could talk to him. He'd either explain what was going on or he'd come talk to me - voicing the concern in an anonymous way.

What's interesting is that no one ever used this. From that point on, when there was a question they just came to me. Not surprising, because taking that action *demonstrated* that I was serious.

Lean Leadership and Management

Value Stream Manager

Business Architect

Value management office

Enterprise Architect

Product Managers

Product Owner

Team Lead / Scrum Master

# PART V: Learning, Improving, and Pivoting.

Learning is the essence of both Agile and Lean. Unfortunately, this is not appreciated as much as it should be. Agile has no set method for learning. Lean focuses on continuous learning and an attitude that the system people are in causes most of the errors that take place. Lean also has a considerable number of first principles that have already been discussed earlier in this book.

Its “stop the line” mentality is based on the mental model that the system is causing almost all of the errors that occur. Therefore, when one is found we must fix it immediately or it will cause other errors. While many pay attention to the 4 values and 12 principles of the Manifesto for Agile Software development, its opening words “we are discovering better ways …” are most important.

This is not something to take place daily or weekly or at the end of an iteration. It is something to do continuously. Learning and improving are about what we’re building, how we are building, and what our process is of building. These are the lessons in this chapter:

3.1 Prepare for the start of a project or for the start of building a product
3.2 Create value in small steps to get quick feedback
3.3 Attend to risk with feedback DTA
3.4 Have an agreement on discussing challenges on a frequent basis MB
3.5 Frequently step back and reflect DTA
3.6 Know how to select a more appropriate practice

The key here is “continuously.” Waiting to improve has 3 serious consequences. First is the lost opportunity for improvement. Something noticed early doesn’t get improved for days or weeks. The second is that at the end of an iteration people want to either celebrate or forget what happened. The opportunity for improvement seems lost. Reflect on this for a little while and see if this explains why so many daily huddles and end iteration retrospectives seem stale.  Third, sometimes the opportunity just doesn’t come. Learning is not something to leave for later, but something to do frequently.

There is a lot of evidence that people learn best in small, frequent intervals. This also makes learning safer as we’re taking small, frequent steps. We want to build the habit of learning. A learning organization is one where the people in the organization learn habitually.

One thing we should be looking for is how to shorten the time of delays and reduce the number of handoffs.

There are two salient differences between Lean’s learning model and the popular Agile frameworks’ learning model. Lean calls for continuous learning. It also includes a model based on the first principles discussed earlier in the book.

Continuous learning means detecting challenges immediately and acting on them just as quickly. As we learn we also must update our model of understanding. It’s important to realize that the rate of learning is usually correlated more with the speed of feedback than with the amount of time that has expired.

## Capability LE1: Prepare for the start of a project or the start of building a product.

We want to ensure that we have clarity on what we are about to do, who is involved in working on the project/product, and what success means. Be clear that this is preparation, not the planning itself. That is, we’re not going to figure out what we’re going to do during the project. We’re just going to do the necessary steps to be able to begin it well.

### *Why this is important*

Without taking the proper steps to prepare, we may have the wrong people involved. Possibly worse is that the lack of clarity of purpose will cause challenges and failure. There are three steps to take for preparation. One is to see where you are. The second is to run a premortem to assess the risks of the project. The third is to get the team ready to go forward.

### *Implementation Methods*

#### See Where You Are

There are several ways you can see where you are. These include:

1. Run a premortem
2. generic value stream
3. Use the challenges board
4. Run a cafe conversation.

#### Anticipating what can go wrong: Do a Premortem

Many people are familiar with post-mortems which are an analysis of a project after it has failed. A premortem is intended to identify what can go wrong before the work starts in order to avoid them.

Gary Klein developed the premortem and described it in the 2007 Harvard Business Review. 85 (9): 18–19.

**What is a premortem**

A premortem is a disciplined way of identifying what might go wrong by pretending the project has failed and asking what went wrong and what could have prevented it. In other words, instead of doing a risk analysis of imaging what could go wrong, we imagine the project has been a disaster and look to see what caused it.

For a variety of reasons, imagining the future and working backwards is more effective than looking at where we are and going forward.

Note that this is not your usual risk analysis method. While the goal may be similar, the approach is quite different. Risk analysis has you start where you are and look to the future. This requires more abstract thinking as you try to envision what might go wrong.

Doing risk analysis also requires people having the courage to say things might go wrong. This is especially true if there's not a safe environment. However, when the manager is the one who says things went bad, they are the ones being negative. Now, when someone says what's going wrong they are doing what their manager requested.

Another advantage of the premortem is that it takes advantage of an interesting way the brain works.

Imagining the future as a disaster is not hard. Now, with the end in mind, the brain doesn't have problems coming up with steps as to how it happened.

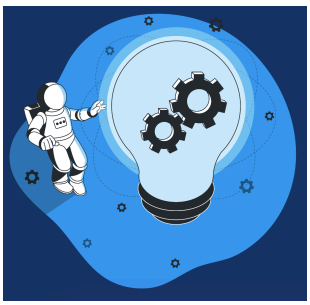
This is why event planners start with the end result and work backwards from that.

This reflects Edgar Schein's maxim, *"we don't think and talk about what we see; we see what we are able to think and talk about."* After imaging what went wrong, we've set the stage for how that can happen.


The benefits of a premortem are:

- Seeing what risks are present
- Creating a safe space to have conversations about the upcoming work. This usually carries over into the work itself.
- Getting people to work together prior to the work also tends to carry over with increased collaboration.

Here's the way you run a premortem.



1. **Set the time for the premortem.** Usually, 30 minutes is all that's needed. If you find you need more you can have a second meeting.
2. **Select a facilitator for the premortem.** This person must act as a facilitator and not a participant. Their role is to guide the process and the conversations. Exceptions can be made at points where everyone has had their say and they feel something very important has been left out.
3. **Imagine your project just ended in a disaster.** The facilitator should give the bad news that the project has ended in disaster.
4. Take 2-3 minutes for each person to write down what happened. This is to be done independently to maximize brainstorming.
5. **Consolidate the lists.** Start with one person. Put their reasons on a whiteboard if everyone is together or a virtual board if anyone is separate. Do this by having one person start, and then have each member add something that is different.
6. Go through the list and speculate on the impact, reversibility, and likelihood of each item.
7. Consider what can be done to reduce the chance of these things happening in the future. *Note this is a kind of learning to learn. We are creating a better environment for future projects/products.*
8. Have someone make a concise list of what might happen and our plans to avoid them.
9. Make a time to revisit / review the list

Don’t allow the premortem to devolve into a risk analysis session.  In other words, don’t ask people to describe any concerns or have the leader ask if “anyone sees any problems?” People who see problems may be quiet in order to avoid looking like a negative person.

Not doing a premortem increases the chances of poor risk analysis.

*I highly recommend Edgar Schein’s latest book is Seeing What Others Don’t.*

### Getting the team set up

Starting projects or creating a product takes a certain degree of planning. We, of course, do not want to over-plan or overthink the process. It could take as little as an hour or two. But avoiding this step can lead to disaster.

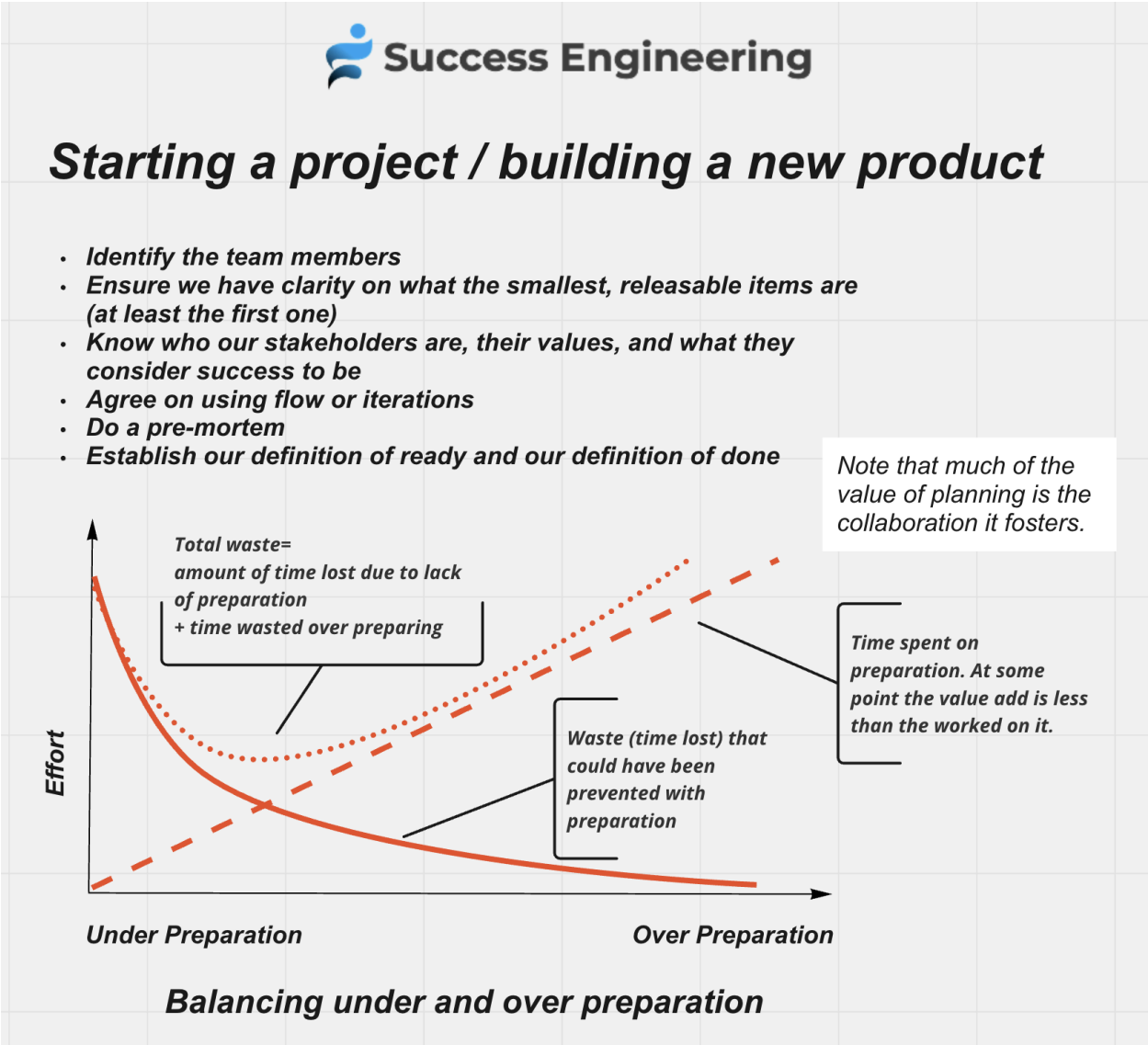
Here are the steps we should take. This is done slightly differently if this is a new or existing team that is taking on something new.

- **Identify the team members** – for existing teams; this may be a mere acknowledgment that we’re using the same team. But a quick reflection on whether new skills, not in the current team, will be required.
- **Be clear if we’re in a discovery (MVP) or investment (QVR) mode.** If QVRs are driving us, ensure we have clarity on what the smallest, releasable items are (at least the first one)
- **Know who our stakeholders are**, their values, and what they would consider success.
- If this is a new team, agree on using **flow or timeboxing**. If this is an existing team, agree on whether to continue using what’s been used
- Do a **premortem**
- Establish our **definition of ready** (DoR) and our d**efinition of done** (DoD)

Premortems typically take 30-60 minutes.

#### *Symptoms of Not Doing This Well*

The symptoms are not doing this or taking too long to do this.




**Some side benefits of the premortem.**

There are some interesting side benefits of running a premortem.

The action of the manager to imagine that things have gone wrong creates a sense of safety to point out bad things that might happen. It has the manager project the negative and then have people answer him. In normal risk analysis it is the people who have to suggest things that will go bad.

By creating this safety in the premortem, there is more permission to discuss what's going wrong in the future.

It also gives you a little training on how to look for problems.

This is quite different from taking a framework and laying it down and pretending things will work out. People often know what can go wrong, so not only providing them space to discuss it, but requesting it, creates more discussion. It encourages people to look at what's really happening, and instead of what they think the framework tells them should happen.

It also encourages agency - that is, taking responsibility for appropriate action.

## Capability LE2: Create value in small steps to get quick feedback.

### *Objective*

Mistakes happen. While we should try to avoid them, we must catch them quickly. Otherwise, more time will be spent on the effect of the mistake.

### *Using Feedback to Manage Unexpected Events*

Unexpected events are a way of life in product development. Learning what will make a difference to customers usually has to be discovered by creating a series of small features, showing them to customers, and getting feedback on their value. This underscores the importance of short feedback cycles. Having them enables the process mentioned above to go more quickly.

As we will see later in the book, much of the unpredictability in knowledge work comes more from people not following good practices. However, external events are unforeseen. To deal with them, we must be in a state that allows effective pivoting.

See the chapter Dealing with Complexity in Knowledge Work for more.

### *Why this is important*

Quick feedback is critical to avoid building the wrong product. In some situations, it is difficult for teams to contact the product owner, so it is essential to set up a cadence for conversations as needed.

### *Implementation Methods*

This is one of the critical factors in deciding whether to use timeboxing or flow. If the product being built is not well known, the continuous demonstration may be necessary for validation and pivoting. If we're creating a new product with MVPs, then it is likely that flow should be used. When feedback speed needs to be high, recognize that planning an entire timebox will likely be a waste because we will change what we want to work on as we get feedback.

The most significant cost/waste is building the wrong thing. That is often hard to prevent, but you at least want to notice when you've done it as soon as possible.

### *Symptoms of Not Doing This Well*

If you’re not getting quick feedback from your customer, you are likely going to do a combination of the following:

- build the wrong functionality
- build more functionality than you need
- miss opportunities to build and create functionality

## Capability LE3: Attend to risk with feedback.

### *Objective*

There are many different types of risk. At the team level, the most important ones to attend are:

1. Risk of misunderstanding a requirement
2. Risk of building the wrong thing
3. Risk of creating bugs

We've already talked about some of these issues. Risk can often be planned for, but the best way to attend to it is to see if you've gone off track continuously. This is why quick feedback is so important.

### *Why this is important*

While it would be great if we could stop making mistakes, the highest cost is the waste emanating from them. This comes from the cost of mistakes propagating, and the time (effort) it takes to fix them. Both of these tend to increase over the time a mistake is made until it is detected (assuming it is). Therefore, detecting errors quickly is critical.

### *Implementation Methods*

Quick feedback loops at all levels are essential. When something is started, it needs to be completed as soon as possible. This is true for all types and sizes of artifacts. Keeping the amount of work in process below our capacity is a way to assist this.

### *Symptoms of Not Doing This Well*

Errors keep happening, and things stay blocked.

## Capability LE4: Have an agreement on handling feedback and challenges while also keeping people informed about what's happening.

### *Objective*

We must continuously look for challenges and be ready to pivot when we encounter them. This must be done at least daily. The more frequently, the better.

### *Why this is important*

When challenges arise, they often are due to the way we're working. Deming suggested that 94% of errors are common cause errors. This means that they will continue happening until we find out why. In any event, the faster an error is detected the less damage it does and the easier it is to correct.

### *Implementation Methods*

There are two common ways to implement this capability. The most common one is the daily huddle (or Daily Scrum in Scrum). The second is having an agreement to bring up issues when they are noticed. The first takes less discipline but has two disadvantages to it. First, it holds any impediments or errors found until the next standup. It also requires a meeting where not everybody is interested in all of the conversation. The second method requires more discipline but is more efficient. We'll discuss both methods.

#### Daily huddles

These shouldn't take more than 15 minutes. The schedule for these should be stable but, unlike the Scrum Guide's mandate to have them at the same time everyday, each day can be varied if that works better for the team. The Agile coach and developers should attend. Although the coach likely won't talk, her observation will enable her to see what she can do to assist people.

The team should consider blocking out an extra 15 minutes in case something comes up that people want to work on. This ensures they have time to at least have a short conversation about it.

**Why the common approach to daily huddles should be avoided**

While no longer in the Scrum Guide, most Daily Scrums are still done by going through the room and having each person answer the following 3 questions:

1. Here's what I worked on since the last standup
2. Here's what's blocking me
3. Here's what I'll be working on next

Notice that #1 and #3 are essentially status. A good team board would make this information available 24/7.

*A problem with these three questions.*

While this seems simple enough, some people will be embarrassed about being blocked - whether it is their fault or not. People from some cultures will actually feel shame.

It's important to notice that using "I" like this puts emphasis on the individuals when we're trying to create a sense of team.

**A More Effective Approach Inspired by Kanban**

Kanban teams often use a daily huddle as well. But, given the greater clarity of the board, it's better to "walk it" instead of focusing on individuals. That is, we review, if necessary, each item that's active along and then go through each "blocked" item. After that, we mention what's coming up. If someone needs help this is a good time to ask for it. *Notice how this shifts the discussion from individuals to the team.* This helps create a team and lowers fear. It creates a greater sense of team as well. This shift also facilitates working on the system.

### Consistency of timing makes sense but it doesn't have to be the same every day

While a set time for Mondays, and a set time for Tuesdays, etc., makes sense, in today's world it doesn't make sense to insist on having it the same time *every* day.

### A more efficient approach for mature teams

An alternative to having daily huddles is to schedule them but only attend them if needed. A common practice is to have an agreement for team members to message each other when there's a problem and resolve it as soon as possible. If you can trust team members to do this, it's often unnecessary to have the daily huddles. It's still recommended to meet at least once a week - just for the social aspect and to ensure all issues have been looked at.

The bottom line is if you have trust and discipline so your team meets the intention of the daily Scrum early you don't need to have it. It saves time and gets you the result faster. Not everyone is required to solve each different problem. People complain about Scrum meetings because they are far from sufficient but are required if management has said to do Scrum.

### A suggestion when you're in a culture of fear

Some standups are ineffective because people are afraid to say what's going on. In this situation, it may be advisable for the coach to talk to people one-on-one and have fewer standups during the week.

### *Symptoms of Not Doing This Well*

You won't have a real team and opportunities to save time will be lost if people on the team don't stay in close touch.

TBD - add visual controls vs information radiators.

## Capability LE5: Frequently Step Back and Reflect.

Have a way of course correcting both what we are building and the way we are working. These go well beyond mentioning any blockages but look to see why they are there as well.

### *Why this is important*

It is important to identify problems and solve them as soon as possible. Waiting until the end of an iteration is risky and wasteful. When challenges occur or mistakes are made, and they aren't corrected quickly it tends to have teams working through their problems instead of solving them.

This is the equivalent to Lean's "stop the line" attitude. When a problem occurs, fix it immediately.

### *How to Implement This*

There are two main ways to implement this. The most common way is to have daily huddles. Standups should be done by focusing on the work. A common approach is to walk the work on the board and ask:

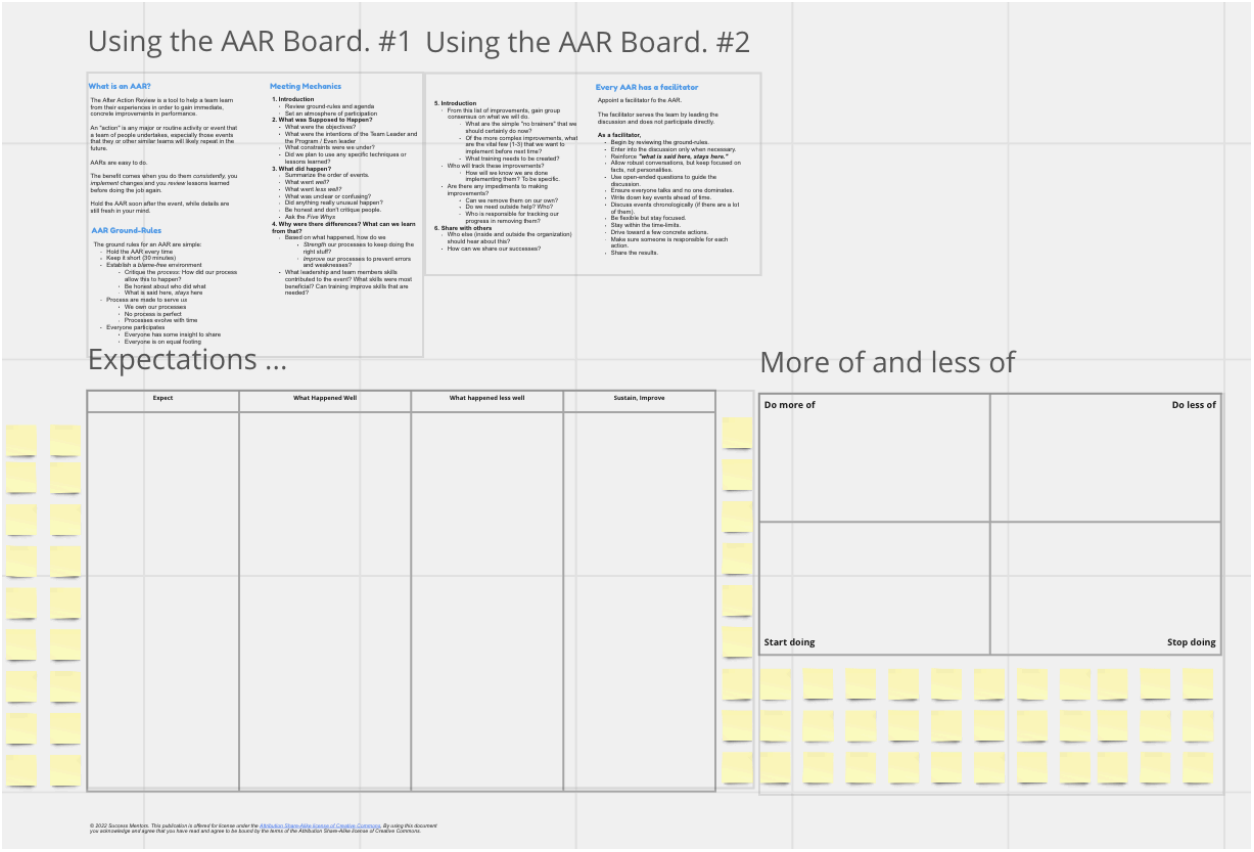
1. This is what we've done - and have a quick statement as to what worked
2. This is what we're going to do next
3. This is what we're having troubles with

Depending on the trouble one runs into, the team needs to look to see not just what the impediment is but what's the cause of it. Is it how work is being done or what work is being done?

Daily huddles can be enhanced or even replaced by impromptu meetings when issues come up. This second way requires a high degree of discipline, however.

#### Tools for Retrospectives

Having virtual boards to do retrospectives with is also useful. The figure below shows one you can get this one and more by joining Success Engineering's free Amplio Community of Practice.




But the main way to get retrospectives to be better is to solve your problems when they arrive. There are two ways to do this. One is to have an agreement among team members to try to solve challenges as soon as they are detected. It often doesn't take the full team to do this. This will both improve the quality of the team's workflow as well as have less to deal with at the end of the sprint.

The second option is to have a short retro in the middle of the sprint. This can be just 15 minutes. If you combine this with the idea of only having daily Scrums when they are needed, you can schedule this somewhere in the middle of the sprint and have it whether a daily Scrum is necessary or not. This can be a quick review of what's going on.

Here are some things to focus on as part of your retrospective that we have noticed make a difference:

1. Identify the most valuable change the team can make and do that.
2. Provide objective analytics to paint a clear story of the impacts. The metrics are for the team. This is the time to use them.
3. Make sure the team provides kudos to each other so that they become aware of each other's unique contribution no matter how small.
4. Whatever actions or experiments that come out of the retrospective must be immediately reflected in the Sprint Goal or as backlog items.
5. Mix it up! Doing the same retrospective style over and over again gets old.
6. Use a retrospective board so that you can see each retrospective compared to previous ones and understand if the team is doing what they said they would do.

The other key thing is to not try to solve too many of the challenges discussed during the retrospection. Typically, just one or two items need to be improved. Then the team will appreciate the progress being made.

### *Symptoms of Not Doing This Well*

Errors keep happening and things stay blocked.

#### Scrum "equivalent"

This capability covers the intention of the sprint retrospective in Scrum.

## Improving Scrum Retrospectives

*This is a shared chapter with Being an Effective Value Coach: Creating Value for Yourself and Others.*

Retrospectives are important but often considered a waste of time.

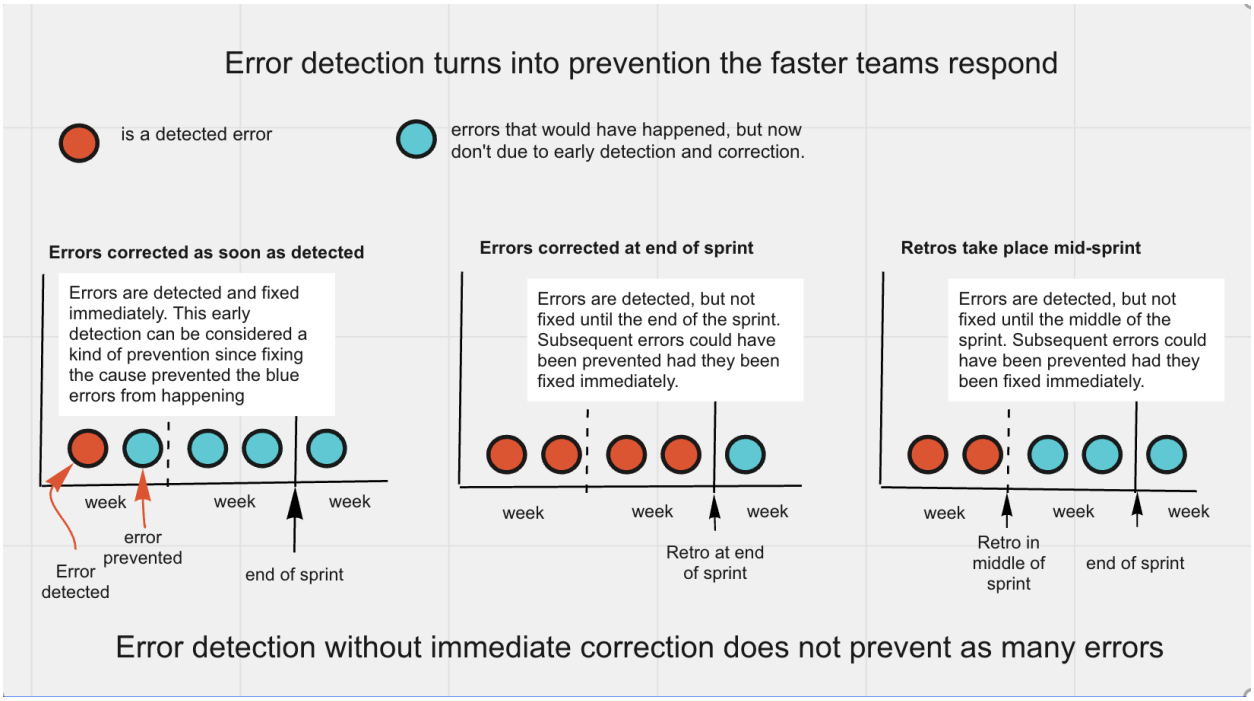
To be effective, we must understand the purpose of a retrospective. There are many:

1. Acknowledge the good that's been done
2. Review the experiment we agreed to and see the impact
3. Detect and discuss impediments
4. Select a few of these to solve
5. Depending on what we have chosen to solve, we follow through by including them in the backlog, implementing new practices immediately, creating a spike for research, and simply taking some committed action
6. Step back and see the bigger picture
7. Team building – solving problems together

They should not be automatically deferred to the end of the sprint. We need to ask "Are they being done at the right time?"

The end of the sprint at first appears a natural choice. But consider that we may have discovered something is wrong much earlier if we held them in the middle of the sprint. By waiting until the end we don't discover potential improvement until it's too late to make a difference in the sprint.

We should also notice that people may be emotionally exhausted from the crunch that takes place at the end of a sprint. A short retrospective in the middle may be more effective. This enables us to detect errors early and prevent them from reoccurring.




## GPS

## Attending to the customer journey

## Coaching

This will pull from the Value Coach Book

# PART VII: OVERARCHING TOPICS

## Capability OC1: Attend to the needs of stakeholders and look to create better methods and outcomes

TBD - talk about the three different types of stakeholders and how you attend to them.

### *Objective*

The customer journey is the path the customer takes using the created system. When developing a product or service, what the customer journey will look like can be a helpful way to visualize what the product/service needs to be. The customer will not appreciate a poor customer journey. It is possible to go one step further, however. Imagine what the customer's operational value stream could look like to improve their current operational value stream. While this can't be imposed on the customer, attending to the customer journey can lead to better operational value streams.

The key thing to remember is – your goal is not to create systems, but to add value to the customer by attending to the customer journey.

### *Why this is important*

Attending to the customer journey and the operational value stream is a kind of UX architecture. When done well, customers will enjoy using the product/service more. More innovation comes from attending to improving how the customer wants to work than by focusing on improving the system itself. Let's consider an innovation that was created by attending to how customers *wanted* to work.

Attending to the customer journey has two advantages:

1. you will focus on the customer more than the system
2. you will see opportunities to innovate on work being done outside of the system (#blog)

**The Sony Walkman**

Perhaps it's an urban legend but I heard this story from someone who knew Akio Morita – the CEO of Sony who came up with the idea of the Sony Walkman. Mr. Morita enjoyed classical music but had to travel a lot. He was unable to bring his music with him. One day he was in his hotel looking down on Central Park and he noticed how some people were jogging and some people walking with boom boxes. He surmised that the people jogging had an interest in music and the people walking with boomboxes had an interest in music. If he noticed someone running with a boombox, given this was New York City, that was likely another matter altogether ☺. He could relate to the dilemma that people could have multiple interests but be unable to meet them both at a single time. The idea of a portable music player hit him.

It is important to be aware that this was not an improvement on a small Sony tape recorder – it has a different purpose. In fact, when he told his engineers that he wanted a small tape player, they kept trying to create smaller tape *recorders.* He had to finally tell them that he'd fire them if they brought him one more *recorder* instead of a *player.* This highlights how developers often focus on the product and not the journey.

### *Implementation Methods*

Quick feedback is necessary. So, if timeboxing is used, the team needs to set up frequent feedback sessions with the customers involved.

We should see how we define the customer journey that can positively impact the customer's operational value stream.

This brings up the following questions on which to base our actions:

1. What would an improvement be for the customer's operational value stream?
2. What customer journey would facilitate that?
3. How can our development value stream attend to this?

### *Symptoms of Not Doing This Well*

When the customer journey is not addressed, the requirements will often not be high quality from the customers' perspective. Using the system will likely appear awkward and not encourage improved operational value streams.

### *How This Capability Manifests the First Principles*

4. Actionable feedback
5. Maintain qualities that enhance flow.

## Capability OC2 How to Estimate - Amplio Team Estimation

*See a presentation and get a copy of a Miro board that you can do Amplio Team Estimation on here.*

People are biased toward underestimating the time it takes to accomplish something. The "planning fallacy" is a term used by psychologists to describe our tendency to underestimate how long it will take to complete a task. The term was first coined in 1977 by psychologists Daniel Kahneman and Amos Tversky.

To combat this, estimating against something you've done in the past is important. Not against what it actually took, but what you estimated it would take. That is, find a job similar to what you would do. See how long you had estimated it to take. Then, see how long it did take. This is likely how long this job will take.

This is called relative estimation.

A more popular type of estimation in the Agile space is called reference estimation. This is where you compare against some standard. Planning Poker uses this approach. One reason Planning Poker is ineffective is that it doesn't combat against the "planning fallacy." The scale is the effort we think something of a certain size should take. But it also has a bias. We need to compare against what we've done and can then see the actual effort taken.

Amplio team estimation takes about 25% of the time that Planning Poker does, is more flexible in situations where expertise on what's being estimated varies and achieves better results. Amplio Team Estimation derives from Steve Bockman's Team Estimation and was influenced by James Grenning's Planning Poker Party (not to be confused with Mike Cohn's Planning Pge of comparing similar things is that you don't need to break each item down into its relative complexity, size, etc.  We use a general sense of effort it will take. Relative sizing is what makes Amplio Team Estimation work so well. And the structure it takes makes it go faster and combats our natural bias to underestimate.

In Amplio Team Estimation you estimate all of our items against each other. This uncovers issues you might not otherwise ses. In Planning Poker, you estimate each item against a standard individually, sizing the items, one at a time.

In Amplio Team Estimation, you first identify the relative sizes of all of the items against each other without assigning a numeric value. You do this by placing similarly sized items in the same column. After you've done this you can add a numeric value. If you include some items you've done in the past, you can get a sense of how long things will really take by presuming you can get the same amount of work done as you did in the past.

My own experience suggests a minimum of 4 times faster than Planning Poker while others have told me they sometimes achieved a 10x rate with Amplio/Team Estimation on which Amplio Team Estimation is based. All with as good or better results. This is the article that first introduced me to Steve Bockman's Team Estimation Game [Team Estimation Game - By Steve Bockman](#).

## Relative estimation

Making estimates based on a value relative to another item being estimated is easier and more accurate. In other words, let's say we want to estimate 5 things: A, B, C, D, and E. It is more effective first to put them in relative order of size, and then add measures of size than it is to say A is a 5, B is a 3, …

When we estimate against a scale, we tend to underestimate it. But estimating whether two things are the same size or one is greater than another does not fall prey to the planning fallacy.

But before we start, we must remember that estimates are really just our best guesses of the effort required based on our current information. We'll have more information later and will refine our estimate then and as we go along.

## There are two levels of estimation.

There is a difference in the need for estimation at the team level and at the product level. At the team level, estimation can often be avoided by just breaking stories down to about a day. But at the product level estimation is more useful, even necessary. I've run estimation sessions with product managers and seen the estimation process make it clear that people had different understandings of what was being built. It's therefore useful to do the estimation process as a way of getting clear about things, even if you don't use the estimates.t

## Step 1: Be clear about what you are estimating

Amplio Team Estimation can be used to estimate anything – size, complexity, value, cost, … Its most common use is in estimating the effort it will take to get the items done. This is useful for planning purposes.

## Amplio Team Estimation gameplay

We'll call the items being estimated "backlog items" (or BIs) because these items mostly end up being on a program or iteration backlog.

The "rules" for Amplio Team Estimation are very simple. Have a board with multiple columns and a blank top row, such as in Figure 1. You don't necessarily need to draw the lines, but keep the BIs neatly in columns if you don't have them.

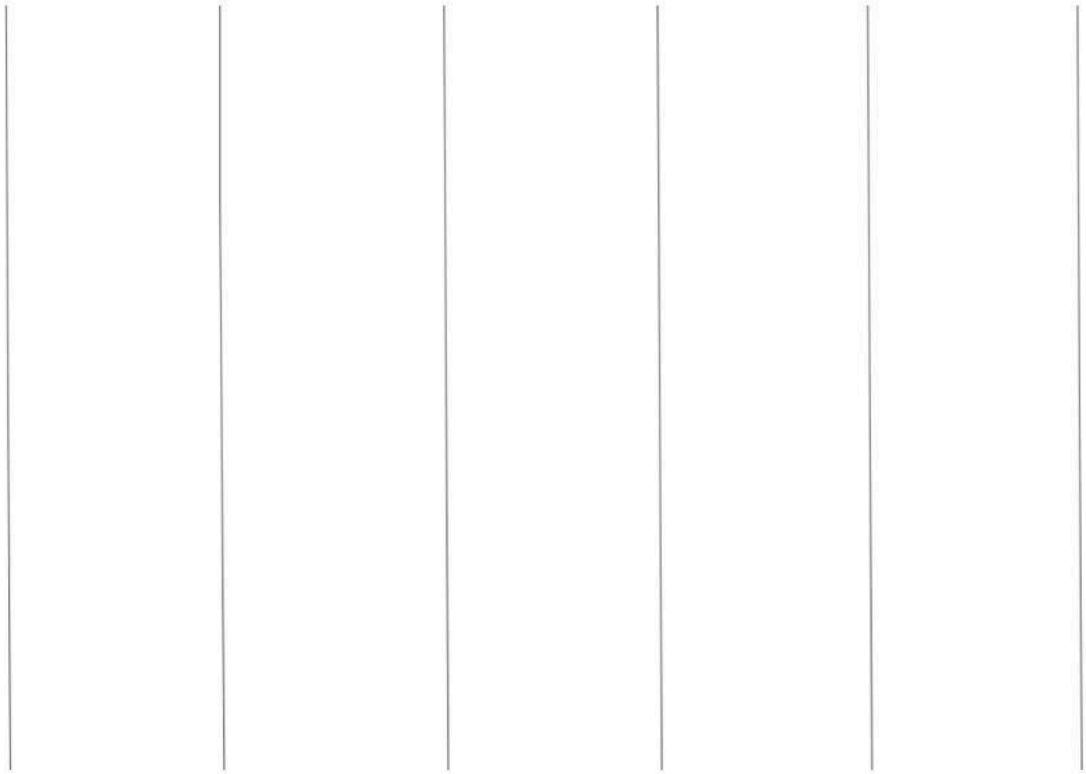


Figure 1: The blank board to start with

## Playing the BIs

- Place your BIs in a pile on the table. Take the top card from this pile and place it on the playing surface at the top of the middle column.
- The next player takes the top card off the pile. She can pass (she may not want to estimate it).
- If she continues, she places it relative to the first card based on size. She can place it to the left if it takes significantly less time. Or she can place it in the same column if she feels it is of the same size. Or to the right if she feels it is larger
- In succession, each player repeats this process with the following extra attentionPlace it in a column that the estimator feels as BI of comparable size.If one does not exist:If the BI is felt to be smaller than BIs on the board, place it just to the left of the leftmost populated column. If the BI is felt to be larger than BIs on the board, place it just to the right of the leftmost populated column. Otherwise, open up a column between the two columns that the BI appears to fit between. Instead of taking a new card, if someone thinks a card has been mis-estimated, they can take it off the board and have a conversation about it. It can then be put in an agreed-upon column or a new one of its own.
- The above steps are repeated until no more cards remain in the pile and no player wishes to move a card

Throughout this time, anyone is free to invoke others in conversation about why they are moving cards and what they think about the size of the BIs. Remember, however, that we are just making estimates, so

conversations to help clarify what the BIs are is useful but getting too hung up on the exact size of the estimates is not worthwhile.

After a few BIs have been played, the board might look like those shown in Figure 2:

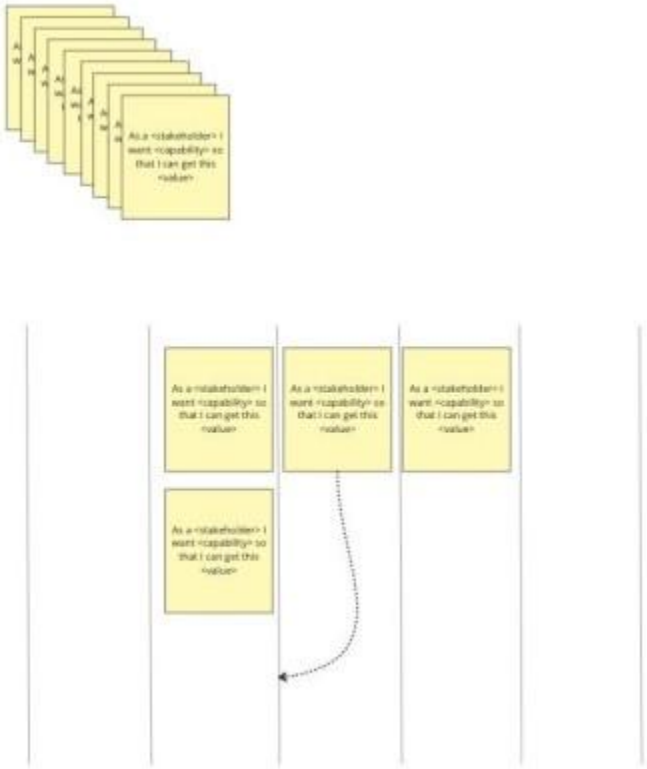


Figure 2. BIs after a few have been played and someone is moving one of the previously played cards.

After all the BIs are played, it may look like what's shown in figure 3.


Figure 3. Possible card layout after all cards have been played

Now is the time to actually add the size of the estimates. We can seed the columns with some past BIs if we like, or we can just guess that a 1 means about a day and go from there. It will normalize itself over time.

We recommend using a modified Fibonacci sequence depending upon the general size of the BIs. For stories, features, slices of MVPs, and QVRs, we suggest using this modified Fibonacci sequence - 0, 1, 2, 3, 5, 8, 13, 20, 40, 100, 200, 400, 800. After 13, the precision of the Fibonacci sequence adds no value and becomes awkward.

For estimating larger items, like QVRs among themselves, you could use something like - 20, 40, 100, 200, 400, 800, 1600, 3200. Larger items need less precision.

We start with the leftmost column and give it a number. Then, work to the right. It may be that we feel a column would fit between two columns where it was significantly larger than the one on the left but still significantly smaller than the one on the right. In this case we can skip a number in the sequence.


Figure 4. Estimated story points to use for each column

After these are agreed upon, these story point sizes should be written on each card.

## Issues related to Amplio Team Estimation - FAQs

**Do Columns need to be used?**

Whether you use columns or not depends upon how you want to do the comparisons. With columns, you can ask, “Is this item significantly smaller or larger than this other item?” If you don’t use columns, you can just list them in approximate order. Usually, it’s worth asking the question. At some point, however, you need to use columns in order to assign a number.

**How do we handle things when we don’t know enough to estimate or have severe disagreements?**

Sometimes, we can’t reach reasonable agreements on estimation either due to unknowns or disagreements on the importance of something. If the item is small, it may not be worth resolving. But when it is large, this challenge points us in the direction of what we need to discover. If resolving this unknown or disagreement is important, it’s worth creating what is known as a “spike.” A spike is a task that aims to discover something or test a theory.

**When should we discuss things?**

We can discuss things as they arise or at the end of the estimation.

## Advantages of Amplio Team Estimation over Planning Poker

**Faster and more fun**

Prior to learning about Team Estimation, we used a technique called Planning Poker. We have found that Team Estimation is faster, easier, and more fun and that people tend not to get bogged down in too many details.

**When only one person can estimate an item**

Sometimes, only one person can estimate something. Notice, however, that they can estimate this item relative to the other items estimated.

**When not everyone can estimate the entire item**

Planning poker requires that everyone can estimate all of the items. In reality, however, two variations sometimes occur:

- Some people are needed to estimate parts of an item. Although there isn't a role of the tester in Scrum, some teams have people who do more testing than development. In this case, those doing the testing may only be able to estimate the testing role. In this case, it may be necessary to estimate the testing effort separately.
- When only certain people have the capability to implement certain items we can have them do the estimates within the context of the other estimated items.

### Use Amplio Team Estimation to Go Beyond Estimating QVRs, Features, and Stories

It should be apparent that Amplio Team Estimation can be used to estimate most anything. Sometimes, you want to sequence items as well. For example, when facing a variety of challenges, teams can use Amplio Team Estimation to order the challenges based on significance. You can use the same techniques, perhaps putting the most significant ones on the left to give them more focus. Whether you size them or not, it can be useful to sequence them. This can open up conversations on how and why you are sizing them this way.

## Case Study Where I Introduced Amplio Team Estimation

I remember a time I was doing a Scrum Tuneup with a group of PhDs who were charged with keeping our nuclear arsenal safe - at least the theory as to how to do it. They didn't really like me there. My initial interviews were supposed to size things up.

But at one point, they talked about how much time they wasted on estimation and how many items could only be estimated by one person. I asked them if they'd give me 5 minutes to show them a better way of estimating. They agreed, and we spent about 15 minutes estimating items. They said in 15 minutes, they got an hour's worth of estimation done. That's where my 4 times comes from. Other people who have used Team Estimation have told me they got a 10-times result.

# Capability OC3: Have an Effective value-creation structure.

tbd - add writeups on cross functional, core, feature, component, and FSTs.

### *Objective*

Have people be organized so that they can work together effectively. How the people who work together are organized is called the "value-creation structure." How they are organized has a big impact on effectiveness. Most important is how focused people are on working as a team. But the way individuals are organized into teams is also important. The greater the physical distance and time zone difference, the harder it is to collaborate.

### *Why This Is Important*

Collaboration is difficult when people are not organized effectively. A result is that synergy between people on the team is lessened. When people are working on multiple projects and/or products, this creates both multitasking and delays.

### *Implementation Methods*

Cross-functional teams are teams where the team members have all of the skills necessary to build what they have committed to. They are designed to eliminate delays, increase collaboration, and create focus on value. While cross-functional teams are ideal, they often can't be achieved due to limited skill sets or requiring multiple teams to build some functionality.

Cross-functional teams are the ideal value creation structure when practical. However, they are hard to achieve and are often not a pragmatic solution for several reasons:

- a few skills are required for multiple teams and only a few people have them - and very often cross-training in these skills is not possible or hiring new folks would give us more capacity than needed at a very high cost
- component teams are needed to reduce redundancy
- in larger organizations, few teams are autonomous and have dependencies on others

Cross-functional teams are effective because they enable having the people who have the skills you need when you need them. But there are other, sometimes more practical ways to achieve this.
*Cross-functional teams need to be a target, not a goal.* As a goal we may be striving for something that is costly and not useful. As a target we can see the difference between where we are and what might be better. We may not be able to achieve them but we may be able to achieve something that has their benefit. It also provides us with information on the risks we have by not having them.

The main goodness of cross-functional teams is needed skills are available when we need them. If we can't, or it's not cost effective to get cross-functional teams, we need to ask if we can get these benefits without them.

The answer is yes. But we need to know the theory of Flow, Lean, and Theory of Constraints to achieve this. And we need to give up a mandate that we need cross-functional teams.

## Scrum Sidebar: The problem with mandating cross-functional teams.

The lack of guidance about what to do when you can't, or it's not pragmatic, to achieve cross-functional teams in Scrum is a serious problem. Many times you see cross-functional teams being formed because the Scrum Guide says to. But this results in redundant code that would be eliminated with component teams. Other times, people can't achieve them but without any theory based on Flow, Lean, or the Theory of Constraints, teams often don't find

solutions. People often don’t recreate obvious solutions on their own. Solutions that merely need to be mentioned for people to be able to adopt them.

## Scrum Sidebar: When cross-functional teams are hard to achieve or not advisable.

Cross-functional teams make it easier for people to work together. It is good to always strive for them. But they are not always the best solution. Most people find themselves in one of the following situations:

1. cross-functional teams already are present or are easy to create
2. cross-functional teams are not currently possible but it would be useful to create them
3. cross-functional teams are not advisable even if they could be achieved

**Let's take some examples to differentiate between cases two and three.**

*Case 1: A few highly skilled people are available that can satisfy the needs of all teams but aren't needed exclusively for any team.*

Ironically, the very first place I taught Scrum was a place where cross-functional teams were not practical. I was working with a team that was creating a tool for another 100 teams. The team I was working with was cross-functional but the other 100 teams were not. Nor should they have been. Each team had access to 10 metallurgists who were PhDs with about 8 years of experience. The metallurgists' time was requested as needed and they were available enough to not slow things down significantly or otherwise cause problems. It would have been exorbitantly expensive to get 90 more folks so a metallurgist could be on every team. Cross-training was not possible. I hold a Masters in mathematics and understand things like Fourier and Laplace transforms. But when I saw what they were doing with these, I not only couldn't understand what they were doing, I didn't even see how it was possible. Non mathematicians could not be cross-trained in what they were doing. Period.

Fortunately, the need for the metallurgists was not excessive. If it were I'd have managed them with kanban boards to help them see who they needed to service next and to let others know about their availability.

*Case 2: Component teams are useful because many teams are working on related products.*

In this case I was working with a company that made products for HR groups in large companies. When we started working with them every team was cross-functional. This was actually problematic. Although their product was essentially the same across their clients, each one had to be tailored to their clients. Each team was focused on the customer and tailored the product to them as needed.

This, of course, resulted in a great deal of duplication - mostly in how information was stored. Each team had their own way. We eventually convinced them that they needed one set of code that worked for all of their clients. That what would be tailored to the client would be the unique implementation required for the client. This led to the formation of a component team that managed the various needs of the database.

Most of the time this worked well. Sometimes a major change was needed and we used the "borrow team member" pattern described below to improve how the two teams worked together.

*Case 3: Limited availability to a few team members who were present when the system was written. Circa 2007*

Before I knew Kanban I had a former client call me to make a proposal. The contract was mine if I could just show them what to do. Unfortunately, I couldn't. The situation was this.

There were 4 teams that were mostly cross-functional. Unfortunately, each required knowledge from one of two team members who were the only ones with deep, intimate knowledge of why the system was built the way it was. Unfortunately, I could not give them any advice on how to handle this situation. Obviously cross-training was advisable, but what to do in the interim? I didn't know.

Now I would have suggested they treat the two individuals as shared services and request them when needed. Since these two people were more senior than the others this would have had the added advantage of getting a higher view of what was going on.

***Case 4: Shared services***

In organizations with more than 50-100 developers there are almost always teams that are required to build and deploy. These can be database analysts (DBAs) or specialized people in ops. I would include architects in this case as well. How do we handle these? Again, it's not difficult to have them manage their requests with Kanban boards.

**When cross-functional teams would not be practical.**

When this case occurs the general approach to take is typically to have as close to a cross-functional team as possible and use Kanban or one of the value-creation structures that follow.

**When cross-functional teams would be useful but don't exist.**

In this case, you don't have cross-functional teams because you don't have the necessary skill set available in the amount needed for each team. We want to cross-train as quickly as possible. But while this is happening, it is useful to use some of the techniques just mentioned to help teams out during this transition.

**Why Scrum's immutability and lack of first principles are so harmful in these situations.**

If teams are aware of Kanban and/or first principles teams can usually figure things out. However, the fact that this breaks Scrum's immutability has two problems. First, the Scrum Master may resist this saying "we've been told to do Scrum but this will break Scrum." This often has teams try to follow Scrum until they can't take it anymore and then just do what's convenient - often leading to problems.

The second problem, the focus on Scrum has them often not see solutions they might otherwise see. Yet another example of Edgar Schein's "we don't think and talk about what we see. We only see what we are able to think and talk about."

These are value creation structure patterns that can be used in different situations.

## Value-Creation Structure Pattern – Borrow Team Member

**Name of Pattern:** Borrow team member

**When to use it:**

When one team has the need for one or more team members full-time from another team for an extended period of time.

**How to implement it:**

The support team members work as if they are on the team they are supporting. They still go to the daily huddles of their real team to keep them apprised of what's happening.

**Forces:**

Comment to write up:

Loaners/borrowers. 🙂 definitely a good way to handle the situation. That said, I would recommend parameters for doing it. Not just in things like keeping on top of what both teams are doing, but also in how frequently the skill set needs to be "borrowed." If it becomes to a regular occurrence, the focus should sh

**A story about how this pattern was created:**

In 2011 I was coaching a company we were working with. They were doing Scrum, but we had also taught them various Lean principles. They were aware that delays and handoffs caused delays and extra work. When people who were required to get a story done were not working together, there was usually a fair amount of rework when integration was done between the teams.

I was showing up at the client every other week. One time I came in, the person running their improvement initiative (he reported to the CTO) asked me about something he had done while I was gone. He said they had one team that required the full-time efforts of two members of another team. They had been working with the supporting team allocating two people full-time to them.

He said he'd been thinking about what I said about the delays in two people working on the same thing but taking days to integrate. He said he had told the two team members to work with the team they were supporting as if they were on it. He added that the members were still considered to be on their original team and had to attend the daily huddles of both teams to ensure any changes they made for the team they were supporting were consistent with their overall home team's efforts.

This worked exceptionally well. There are three salient points to make, however.

First, these two people complained about having to attend two daily huddles. It was a small price to pay, however. It does illustrate that sometimes individual team members don't like doing what's for the benefit of everyone.

Second, the manager based the decision on what he knew about Lean theory, having never seen this technique before. He had created it out of an understanding of Lean theory.

Third, the improvement was immediate. It did not take any learning to get the benefit.

## Value-creation Structure Pattern – Sharing Team Member with Kanban

As previously mentioned, the ideal value-creation structure is a cross-functional team. However, this is not always possible or even advisable. Two cases come to mind (these are actual cases, not fabrications to make a point).

**Case 1:** Specialists whose expertise needs to be shared across multiple teams and who can't be cross-trained, but for which there is enough overall capacity.

The case study I mentioned earlier that referred to metallurgists are an example.

**Case 2:** Three teams were looking to go Agile and wanted to do Scrum. However, two people amongst the teams were needed by all three teams. Essentially, these two team members were part of the team that had originally built the application being extended. Without their past experience, the three teams would struggle. Scrum requires each team to be cross-functional, but since all of the teams needed some of these two, no team could truly do Scrum. In this case, it was possible, and advisable, to cross-train people, but the question remained – what do they do while the cross-training is taking place?

The solution was straightforward. Consider the three teams to be "core teams." That is, each has most of the skills necessary to get the job done but is just missing one or two required skills that the two specialized team members had. Set up a kanban board for the two people the teams need. The three teams put what they need on the board either in a planning-forward manner or when they discover them. In a "planning forward" manner, during the iteration planning of the two teams, they can load up the Kanban board, ensuring they don't overload the two people. The two folks share the board in this case since they had the same necessary skills.

## Value-Creation Structure Capability – Focused Solution Team

TBD can see a writeup from the PMI here.

## Anti-pattern(s)

When you don't have effective value-creation structures, several challenges result:

- Delays occur between the steps
- People have to multi-task
- Integration is delayed, making it take longer time than necessary

## Resources

Expertise And The Shared Services Problem: A Conversation With Don Reinertsen. Another great article debunking the myth that cross-functional teams are needed. Amplio is consistent with this approach.

# BLAST (Basic Lean Agile Solution Team)

BLAST is based on the principle that it is easier to get multiple teams to work together with alignment than with coordination. When teams are aligned on creating value they will understand that the thin slices of work they do in creating value must be aligned with each other to enable quick integration and fast feedback. Without this self-organizing alignment, teams need to be coordinated - adding overhead, cost, and waste to the effort.

The cost of coordination is a significant problem to a scaling Agile from the bottom up approach.

Pushing Agile from the top is inefficient because it demands people work in a particular way (ironically, so does the bottom-up scaling of Scrum).

This is the fallacy of the bottom-up or top-down or both discussion.

What's needed is the alignment created by attending to value streams.

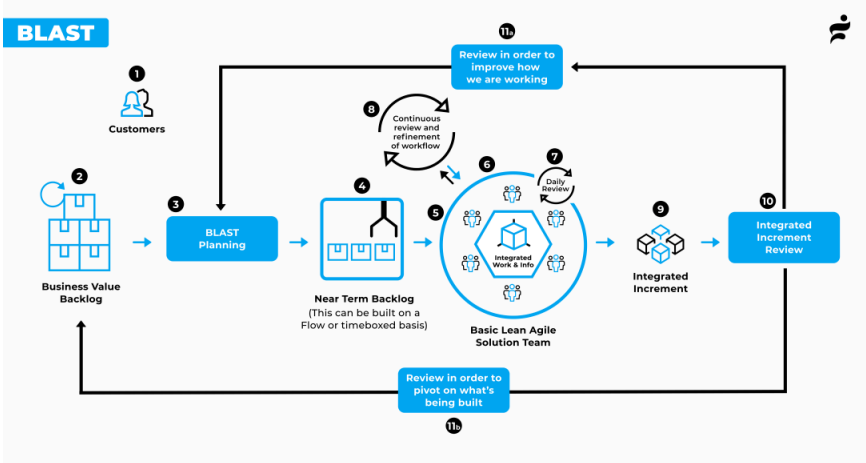
Then everyone is on the same page.

The idea of hitting impediments and then solving them is not as good as taking a perspective that avoids the impediment in the first place.

BLAST is a set of patterns used to coordinate the work of 3-9 teams. Each team can be using any Agile method (e.g., Scrum, Kanban, eXtreme Programming). BLAST provides teams the guidance to work together in either a flow or timeboxed manner. These teams can be from the same part of an organization needing to coordinate to create or enhance a new product or when teams from different parts of an organization need to work together.

BLAST integrates Flow, Lean, and the Theory of Constraints within the perspective of value stream management. Doing so makes coordinating multiple teams straightforward to accomplish. BLAST can be taught as a stand-alone technique but is also presented in Lean-Agile Teams Master Class since few teams work in isolation.

BLAST is not immutable in two senses. First, each pattern comprising BLAST contains multiple solutions and allows for adding others.  As all patterns do, it provides for tailoring its implementations to achieve the desired result for the context at hand. BLAST can also be extended by adding additional patterns.




## The Key Points of BLAST

**Value stream management provides valuable insights for BLAST.** By taking a value stream perspective, all the steps and components of BLAST fit into place and describe the relationship to each other. BLAST teams take an end-to-end view of their customers' value streams, from taking a concept (idea) through the processes and activities necessary to deliver the intended value. The goal is to discover and remove the constraints and wastes that impede the delivery capabilities of the targeted value stream.

The primary role of BLAST-oriented executives, managers, and leads is to create and sustain a value-delivery-oriented focus, removing the impediments that arise from organizations organized and siloed around business domains or functions.

**1. Use the customer journey to improve the customers' operational value stream.** How customers interact with your services and systems is called the customer journey. The entire workflow of the customer is called the customer's operational value stream. By attending to the customer journey, we can positively influence the customers' operational value streams and provide value to them.

**2. Have a business value backlog focused on our products and services.** This backlog uses minimum viable products (MVPs) when we need to discover if a new product will be valuable and Quickest Valuable Releases (QVRs) when enhancing existing products. Both focus on delivering the highest value in the shortest amount of time. This business focus also provides the opportunity to create a **test plan** based on Acceptance Test-Driven Development or Behavior Driven Development.

**3. BLAST Planning.** The BLAST teams work together to create a plan for future work. It can be timeboxed planning or flow-based planning. If timeboxed, the iteration length will be no more than two weeks. The result is a sequence of slices for the teams to pull from and build together.

**4. A near-term backlog.** This backlog contains the items the teams will work on over the next week or two. BLAST allows for teams to either pull from this shared backlog or for each team to have its backlog. In this latter case, it is useful to refine the near-term backlog by splitting items into vertical slices. The appropriate part of these slices can then be given to the teams to work on in a coordinated fashion.

**5. The BLAST pulls work from the near-term backlog to improve flow and delivery of value**. *Planning, acceptance criteria*, *development*, *test*, *review*, *and delivery* activities operate at an accelerated pace to create realizable value. The coordinated pull of items from the Product and Iteration Backlogs as capacity becomes available, not pushed as batches of work items into the development streams, reduces waste while facilitating an alignment of teams.

**6. The BLAST value-creation structure and workflow.** The BLAST consists of 3-9 teams working together. The BLAST itself is cross-functional in building the desired product/service. But each team in the BLAST does not necessarily need to be fully cross-functional. They operate at the same cadence, but each can work in a way that works for them. Each team is responsible for the non-functional requirements for what they build. BLAST retains the notion of having small, autonomous, and competent teams pulling the highest priority work items off a single, definitive product backlog. However, BLAST also incorporates small value-creation structure patterns that are needed when one or a few individuals' skills are required across multiple teams.

BLAST takes advantage of CI/CD and DevOps pipelines. The iterative and incremental development practices of Agile have been supplanted by the highly accelerated pace of Lean-oriented continuous flows exemplified in modern CI/CD and DevOps software delivery pipelines. BLAST simplifies integration by providing teams a way to naturally coordinate their work, enabling quicker, smaller integration steps.

**7. Daily review.** Short, pre-scheduled meetings to review where we are. The intent is also to enhance a sense of the team. Each team reviews their board by stating, for each appropriate item: 1. here's what we're working on, 2. here's where we are blocked, 3. here's what we're going to work on next

**8. Continuous review and refinement of the backlog.** Members across the teams need to coordinate the following: 1. the test plan for non-functional requirements, 2. how to integrate across teams, 3. ensure a coordinated workflow takes place, and 4. attending to how the teams are working together.

**9. Integrated Increment.** These increments can be produced on a cadence basis or as MVPs or QVRs are created.

**10. Review.** While it's best if customers frequently review the work being done, each integrated increment must be reviewed when released.

**11a. Review to improve how we are working.** Improving how we are working is a continuous process. In addition, Agile Retrospectives are a planned event to have teams evaluate and implement short-term improvement opportunities. BLAST teams also look at how they can deliver a series of QVRs that incrementally improve their customer's value stream delivery capabilities from the Lean-oriented perspective of improving work and information flows. In addition, BLAST incorporates Deming's Plan-Do-Study-Act cycle.

**11b. Review to pivot on what's being built.** Use the feedback from the integrated increment review to adjust the business value backlog.

## Key Insights From BLAST

The idea of having problems and then solving them is not as good as taking a perspective that avoids the problem in the first place.

One of the reasons Scrum is so expensive is that you have to hit the problem and then solve it. Why even hit the problem?

Alignment is often present at planning events, but then lost when the work starts.

Theory enables you to look ahead. Inspect and adapt is by definition a waiting to see what happens and then react.

Teams don't need someone looking over their shoulder if people understand what's happening.

But do need it if they don't.

Why can't scrum masters delegate some of their tasks to team members?

That's a scrum super power.

You don't improve a system by improving its parts.

Don't start things you can't finish because if you can't release it you're just increasing the amount of work in process.

Mark Twain - We should be careful to get out of an experience only the wisdom that is in it and stop there lest we be like the cat that sits down on a hot stove lid. She will never sit down on a hot stove lid again and that is well but also she will never sit down on a cold one anymore.

Watch a video from the Amplio Development Masterclass about BLAST.

# How Users of Scaling Scrum approaches (Scrum @Scale, Scrum-of-Scrums, LeSS, and Nexus) can improve their methods with the lessons of BLAST

**The short answer.**

I created BLAST in 2008 at a company doing Scrum well but having difficulty doing integration at the end of their sprints. It looks surprisingly like Scrum@Scale but has several differences. All of the

Scrum-based at scale methods ironically impose Scrum on the teams instead of creating an environment where the teams can choose how they want to work. They can be self-managing with in the context of the bigger picture.

1. it uses Quickest Valuable Releases
2. has all teams align around the QVRs being built instead of trying to coordinate things
3. is based on designing an approach holistically where each team fits inside the bigger context instead of starting with teams and combining them together. This is particularly important because systems are not a collection of pieces as much as they are defined by the relationships between the pieces. By designing it this way each team can use whatever approach suits them best.

**A longer answer**

Basing BLAST on Flow, Lean, the Theory of Constraints, and using the value stream perspective enables holistic thinking. The focus is on the entire group of people involved and does not require fully cross-functional teams. While these are ideal, they are often difficult to achieve. Unfortunately, Scrum@Scale, Scrum-of-Scrums, LeSS, and Nexus don't provide alternatives when cross-functional or feature teams are not achievable. Concepts in BLAST can be used to adjust how the teams work together when ideal team formation is not possible.

It is recommended to users of these methods to consider adding the following to their approach:

1. Attend to the value stream
2. Use Quickest Valuable Releases to create a focus on what value will be delivered next
3. Look to global optimization, not local optimization
4. Use MVPs and QVRs
5. Attend to the customer journey
6. Strive for continuous, not just iterative, learning using the PDSA cycle.
7. Incorporate CI/CD and DevOps Pipelines
8. Allow for flexible value-creation structures
9. Have a role for management to help improve the overall environment within which the BLAST works.

# The Amplio Scrum Guide

*This section only needs to be read if you are using Scrum and want to use Amplio to extend it.*

The Amplio Scrum Guide is not based on Scrum. It is a subset of Amplio designed to help those using Scrum to lower their total cost of ownership. It is called “The Amplio Scrum Guide” because it looks like an extension of Scrum since it is for those using Scrum.

Many companies are enticed by the low cost of starting Scrum - send a person to a two-day workshop or reading the Scrum guide. This ignores its high total cost of ownership which includes it not always being fit for purpose and insufficient theory to enable people to access their dormant knowledge.

Dormant knowledge means something someone already almost knows but which hasn’t hit their consciousness. Often it merely needs to be mentioned and they become aware of it. Sometimes it’s not a particular thing but a relationship between known things.

These quotes tell much of the story:

- Edwards Deming (paraphrased) “Theory without practice is useless. Practice without theory is expensive.”
- Edgar Schein “We don’t think and talk about what we see. We see what we are able to think and talk about.”
- Will Rogers “There are three kinds of men. The one that learns by reading. The few who learn by observation. The rest of them have to pee on the electric fence for themselves."

Reading selected parts of this book will provide:

- a way to create a fit-for-purpose, Scrumlike, set of practices
- a transition path to this more effective approach
- more concepts and practices as your teams mature
- first principles in knowledge work and guides to help you make decisions based on them
- you access to many established practices that are not provided in most Scrum training
- how to use virtual collaboration boards to increase collaboration, even across time zones
- you more effective practices to replace decades old ones that should no longer be used
- a way to trust your own judgment more
- understanding right from the start. Following until you understand is an Agile anti-pattern.
- how to determine whether you should be using a flow or timeboxed system
- how to increase alignment which lowers the cost of coordinating teams.

The overall effect of this will be:

- being more effective
- increased morale
- enabling Scrum Masters to coach 2+ teams and still have time left over to work with management on improving the overall ecosystem

Amplio Scrum’s improvement over Scrum Guide Scrum can be summed up in these three statements:

> “The idea of having problems and then solving them is not as good as taking a perspective that avoids the problem in the first place.”
>
> “One of the reasons Scrum is so expensive is that you have to hit the problem and then solve it. Why even hit the problem?”

> “While Scrum may provide you with an inexpensive start, it has a high total cost of ownership. Amplio *can* provide you with an inexpensive start while providing you more as you need it to keep the total cost of ownership down.”

There are three ways to read Amplio for those doing Scrum:

1. Those who want to get as much as they can. You will see tags “<Amplio_Scrum_Start>” and “Amplio_Scrum_End>” Read all the content between these tags.
2. Those who want to just focus on specific practices that will help. Read sections which are tagged “<Amplio_Scrum_Highlight>” These are usually embedded between the start and end tags.
3. Read the Scrum sidebars. These sidebars correct many misunderstandings and misrepresentations I see in the Scrum community. They provide alternative ways to look at things.

# 21st Century Agile Team Approach <Amplio_Scrum_start>

*I am enthusiastic over humanity's extraordinary and sometimes very timely ingenuity. If you are in a shipwreck and all the boats are gone, a piano top buoyant enough to keep you afloat that comes along makes a fortuitous life preserver. But this is not to say that the best way to design a life preserver is in the form of a piano top. I think that we are clinging to a great many piano tops in accepting yesterday's fortuitous contriving as constituting the only means for solving a given problem. – Buckminster Fuller*

*You never change things by fighting the existing reality. To change something, build a new model that makes the existing model obsolete. – Buckminster Fuller*

Agile as a community working to find better software development methods started in the mid to late 90s. Most teams adopting Agile were innovators and what Geoffrey Moore (Crossing the Chasm) would call early adopters. They were typically organized into autonomous and cross-functional teams.

Among Agile frameworks, Scrum is the most popular. Scrum focuses on identifying impediments but leaves teams to solve them. In the early days of Agile, this was the best we could do. However, times have changed.

Agile, and therefore Scrum to some extent, is used almost everywhere now, most of the time not in places that it was designed for. Scrum was designed for teams that are cross-functional, autonomous, and working on creating new products. That is not where it is being used.

Whereas 20+ years ago we often knew what worked, we didn't know *why* it worked. This had early practitioners adopt frameworks that spelled out how to start but provided little guidance on how to customize the approach to the needs of the teams involved. As organizations adopting Agile have shifted from early adopters to the late majority, and as the adoption of Agile has moved from single teams to an entire organization, the early philosophy of Agile has proven to be insufficient.

The demand for people trained in Agile has created what many sarcastically call the "Agile Industrial Complex," referring to the churning out of certifications by several organizations even though those certified do not really know how to run an Agile project. These organizations have little incentive to provide quality training on evolving methods – and so they don't. Agile is losing its glitter in the face of a 2-decade old approach being taught in an even older teaching style – one exacerbated by the Pandemic, which has had people use in-person training methods for remote training.

Furthermore, the issue of training people in how to convey the ideas they've learned has been ignored. As a result, even those who understand what to do face the challenge of conveying that knowledge to others.

The result of all of this is that Agile has devolved into a difficult situation for many – not unlike a person driving backward, on the wrong side of the road, complaining about how difficult it is. Fortunately, we can now correct this situation.

# PART VI: MISCELLANEOUS TOPICS

This section covers a variety of useful, miscellaneous topics.

# Amplio Team - A Lightweight Agile Approach Based on Getting Feedback Quickly

This lightweight approach gives teams a way to start Lean-Agile in a non-disruptive manner that illustrates the fundamental principles of Lean-Agile. A case study is presented after it to facilitate understanding.

Having a lightweight approach enables a team with no Agile experience to get started. More importantly, it provides a core set of concepts that can be used to continue to improve. But it must be more than intentionally incomplete, it must be intelligently incomplete. That is, its core starting methods must include a way to improve as you learn. Edgar Schein's maxim – "we don't think and talk about what we see, we see what we are able to think and talk about" implores us to use a method that is self-expanding.

Amplio Team is a lightweight approach to getting a team to be more Agile. It is a subset of Amplio Development and is based on the same core principles. These core concepts are sufficient to get started but lead us to richer concepts when needed. This enables us to start quickly while being able to access additional concepts as needed. This avoids overloading people while avoiding stagnation.

The essence of Agile is delivering value quickly by focusing on working on small items while getting quick feedback to both learn what's needed and how to build it. A key to this is avoiding overloading teams. This requires teams to pull their work from a backlog when they have capacity.

Consider how much of your organization's work has been due to:

1. Misunderstanding requirements from the customer or product owner
2. Writing stories that implement the wrong functionality
3. Development errors that were not detected for a long time
4. Taking a long time to fix errors because they were detected well after the time they were made
5. Having a misunderstanding between teammates that results in handbacks and extra work
6. Multitasking

Now consider how getting feedback at all levels of work would help with these. A simple step to achieving this is to see what you could complete that would provide feedback on what's been done instead of starting something new. Taking a set of simple concepts that work together enables you to have a lightweight, fit for purpose, easy to learn solution:

1. Do what it takes to eliminate delays in your workflow; these will slow down work, delay value delivery, and magnify the impact of errors.
2. Organize your people into a cross-functional team. The team should have sufficient domain knowledge and skills to create the product. This includes discovering what it needs to be.
3. Create a product backlog that contains the requirements of what needs to be provided based on what's of value to both customers and other stakeholders
4. Work on small batches of work that provide value, preferably no more than one week's worth. Only pull work from the backlog when you have the capacity to finish it. Too much work creates delays and waste.
5. Build your work in a high-quality manner. If the product is software, then build it in small end-to-end slices. Don't put in extra code to make it changeable but consider how the code may need to change.
6. Get an understanding of how the person specifying the requirements will validate them. At a minimum, when a requirement is given, "how will I know I've done that?"

7. Try to avoid handoffs. When necessary, the people involved should have a conversation about the work. View handbacks as having done an improper handoff and make corrections for the future.
8. Focus on achieving quick feedback by keeping the amount of work in process within the team's capacity. When someone finishes something, have them help finish something that has been started already before starting something new.
9. Finish a feature before starting another feature. Finish function that's part of a release before starting part of another thing that will be released later.
10. Agree when to meet to review how well the way you are working is going. When a challenge comes up, work on solving it immediately.
11. Have all work be visible and the way you're working together be explicit

You can start applying the above to any approach you are taking. The key is to observe when delays and handbacks occur. Notice how these cause extra work. Unplanned work. Wasted work. We want to avoid these delays and handbacks. The first step is seeing them. When visible, many of these delays can be eliminated. Some common delays are the time from when:

- you get a requirement until you use it
- a misunderstanding happens, and it is discovered
- a coding error is made until it is detected
- an agreement is made between teams on how to integrate and when integration starts
- work is started on building a release until it's completed
- work is started on a feature until it is completed
- work is started on a story until it is completed.

Take a few minutes for each and reflect on why these delays create additional work or risk.

## Quick decisions to make before starting.

Before starting, it is a good idea to do the following:

1- get clear on the value you want to create and the stakeholders involved
2- decide on whether to use timeboxing or flow
3- decide how to handle problems that come up – wait for a pre-specified time at the end of the timebox or meet immediately (recommended)
4- take 30 minutes to an hour to do a premortem
5- decide on a definition of done
6- agree on classes of service

All of the above can be achieved in a 2-hour meeting. Recognize that you may change them when you see better alternatives.

## A Case Study

This case study is provided to give a specific example of the above. It was based on **visibility, explicit workflow, a focus on deliverable chunks of value,** and **immediate solving of challenges.** The author was involved in this project.

### Context

A cross-functional team of four people was creating an e-learning workshop. Two were primarily knowledgeable in the content while the other two were responsible for building the actual delivery

material.  One person on the team had a more profound experience than the others. He was considered the subject-matter expert but took no authority over what content to put in. He, and the rest of the team felt better content would be achieved via team consensus.

### *Agreements made among team members:*

1. No one had authority over decisions on how we'd work, but they would be made as a group.
2. Schedule daily discovery and fix sessions. Each was scheduled for an hour, but if there were no issues, we wouldn't hold them. When held, the meetings had two parts, the first a quick session describing the issues (5-10 minutes typically), with the second being a meeting to fix them. Only those needed would attend. This enabled identifying issues and immediately addressing them. This more extended segment would only take place when needed.
3. When a problem was found in how we were working, a quick meeting would occur if it could be solved immediately with appropriate people. Otherwise, it would get discussed in the next daily meeting.
4. All work was to be visible using Kanban-style boards which were used to see where everyone was with their work.
5. We had a weekly cadence but worked in a flow manner so we wouldn't fall prey to having the work fill the time available.
6. We would interact with customers as much as possible.

### *The net result*

The project proceeded on schedule, within budget, and with the quality we were looking for. Retrospectives on work and the way of working typically happened 2-3 times a week. Everyone felt that they could contribute and there was never an appeal to authority.

The less experienced Agilists on the team learned a lot about Agile. The more experienced people learned how to convey Agile concepts.

## Scrum as Example <Amplio_Scrum_start>

Teams should never buy into a  "just use it as is" attitude.  This is because "as is" may not be fit for purpose for the team. That said, it is often helpful to jump in and start doing something in order to learn. But doing this should not be taken as a commitment to what you started with.

Scrum is sometimes a good, simple start if you already have a cross-functional team. If you start with it, consider it an example of how you can do things. Not that how you are starting is set in stone. In other words, it isn't *the* way but rather *a* way. I call this "Scrum as example." It's a model for starting, but as you learn you can go beyond it.

After starting, the factors for effective value streams can be used to make changes to Scrum's practices safely. Use the factors for effective value streams to validate that the changes look to be improvements in how to work. And then, of course, validate them.

We will learn how to use the factors for effective value streams later in this book. Once you know how to do that, you can start quickly with Scrum and adjust as needed. This requires knowing:

- How to improve when you uncover impediments to delivering value in your organization
- What to do when you have challenges following the practices of Scrum

The power of Scrum as an example is that neither an organization nor any individual has to abandon what they know but can use their knowledge of Scrum as a starting point.

## Contrasting Amplio Team with Scrum – Simpler, Richer, Easier to Master

The biggest difference between Amplio Team and Scrum is that Amplio Team can be slightly tailored at the start and make additional practices available when they are needed. This makes Amplio appear to be just as simple as Scrum at the start but enables it to grow as needed. The tailoring, however, enables it to be more fit for purpose and therefore simpler to adopt.

Amplio is a very rich system. With a connection to hundreds of options that are made available as needed. It can start out as simple as Scrum by providing only a few of its practices – and not going deeply into even those. The trick to being simple is not to have a limited system as much as it is to not overload teams.

Scrum is not so much simple it is minimalistic. This requires people to reinvent practices. Also, because Scrum does not present first principles, or how to change practices for more appropriate ones, teams often struggle when what they need conflicts with Scrum's prescribed practices. Even when there is no conflict, Scrum practitioners have to re-invent the wheels that Amplio provides when needed.

Scrum being immutable is not the problem. It's a symptom of the problem –Scrum isn't based on first principles. Scrum does not provide a method to predict if a change to one of its practices is an improvement. This locks people into it and requires the inspect and adapt routine which is a lot slower than having a solid theory.

**Scrum and Amplio Team are both easy to start.** "Training" on my first Scrum implementation took a couple of hours. Training on Amplio Team doesn't need to take any longer.

Amplio Team is often **more fit for purpose than Scrum** because it allows teams to decide how to achieve their objectives while Scrum requires using some practices that don't work well everywhere.

Amplio Team's core provides guidance on how to continue to **improve by using first principles**. Scrum insists on practitioners following its immutable roles, events, artifacts, and rules and provides no set of first principles to use.

Amplio Team **never suggests following it**. Rather it is organized around easy-to-understand objectives. Practitioners are encouraged to meet the objectives in any way they can that works for the team. Scrum requires many practices to be achieved regardless of how well they fit the team or if there are easier ways to achieve the same desired result.

**Amplio Team heeds Edgar Schein's warning** about not seeing what we talk about by being intelligently incomplete instead of just intentionally incomplete. Its focus on first principles and quickest value facilitates conversations about alternative methods of removing impediments. The book this chapter is a part of - Amplio Development: The Path to Effective Lean-Agile Teams - also provides a reference for what else may be needed.

Amplio Team is focused on continuous learning as opposed to learning at the end of the timebox.

# The Risk of Starting Agile at the Team Level

While it's popular to start Agile at the team level, it runs the risk of making it harder to get Agile to work across the organization. Inter-team dynamics are quite different from intra-team dynamics. Learning to focus on teams often sets up sub-optimizations that are hard to overcome.

It is seductive to think about scaling Agile up from teams to the enterprise. It seems like the correct path to take because you can almost always find a team or two where Agile methods lead to great improvements over Waterfall methods. But what works for a few teams at the local level often obscures the bigger picture: creating enterprise agility. Enterprise agility is the ability for an organization to deliver value quickly when needed. Sadly, I have seen many organizations achieve many successes locally – Team Agility – and move even further away from enterprise agility.

## A common problem

A common problem facing many organizations is having too many projects going on for individual team members. This happens for several reasons:

Some people have certain special skills, domain knowledge, or familiarity with legacy code and so need to be shared across several teams.

When staff is organized by role – with business analysts reporting to one manager and developers to another –teams get formed and reformed as needed. Of course, people are not always available exactly when they are needed. To solve this problem, people are often given two or more projects to work on so they will always have something to do.

When both happen, it is especially bad. The most highly skilled, in-demand people end up being sucked into many teams and many projects, and so suffer the most from multitasking. And everything becomes inefficient because people depend upon them.

Suppose you have 100 people working on an average of five projects each. If the average number of people on each project is 10, then you have 50 projects going on at any one time. Now, you decide to “go Agile” and start a pilot project. You can pull together a cross-functional, self-organizing team that can have those highly specialized people dedicated to the pilot because you want to demonstrate success. You collocate them and give them a dedicated team room. Even if they don’t change anything, our experience would suggest they’d be three times faster and build better code to boot – clearly a successful pilot project. This happens because the team can now focus on one project, and this results in fewer delays which both speeds up the work and raises the quality of the product.

## Early success without a path to scale

That is great for the pilot project. And it is great for specialized people because they don't have to multitask. However, it is not so great for the rest of the organization because these critical people are no longer available. This sets things up for early success without a path to scale. Ironically, you’ve not really demonstrated that you’ve improved the business. In fact, you may have made it worse… only you may not have noticed it.

What will have happened? While the Agile team has had great success, the rest of the organization now has slightly more than five projects to work on and they no longer have access to those key people they had previously relied on but who have now been dedicated to the pilot. Of course, you may not notice this slight degradation because getting things out of those other teams was difficult already. Everyone is working even harder now – but it seems like even less value is coming out.

So, what do you do? Since the Agile pilot project was a success, it seems that Agile is clearly better than the current process. The obvious solution is to create another Agile team. And keep doing this until the entire organization has transitioned to Agile. For a while, you are pleased because each new Agile project is achieving great success. All the while, however, the rest of the organization is slowly getting worse.

Eventually, the Agile teams will start running into the wider organizational impediments that you were trying to overcome in the first place. Your Agile approach may not be making these impediments clear because your focus on the team's success has blinded you to the bigger picture. Even if it does, team-Agile methods do little to help you solve enterprise issues. At some point, you come to realize that merely trying to scale Agile just isn't going to work.

## Agile at Scale Is an Enterprise-wide issue

You conclude that scaling Agile is difficult! Or, you lament that the organization will not solve the real problems they are facing. This last one is true because it is so difficult. Unfortunately, team-centric Agile methods give little insight into what the problems are or how to solve them.

Achieving enterprise agility is an enterprise-wide problem. And it requires an enterprise-wide view. That is what Lean-Agile does. Even if you can only start at a team level, you must look to see how your local changes affect other parts of the organization. This, by the way, is one reason why Kanban can often be a more effective manner to start an Agile transition when you have challenges in creating teams. It enables you to start where you are and directly see the effects of your changes.

## Starting With Teams Teaches Local Optimization

When organizations start with Agile at the team, they focus on optimizing their own work. As Agile spreads across the organizations, teams need to attend to how they can cooperate with other teams to create overall success.

This requires teams to slow down in their work to help the overall effort. Teams, however, often don't like to do this for two reasons. First, members of a team often like to think of themselves as very successful. Performing as well as they can feels good. The second is that members of teams are often compensated for how well the team does. However, these measures often don't take into account how the team contributes to the overall success of the organization.

When teams are successful in improving their own methods, they often are reluctant to deoptimize their own success for the overall success of the organization.

## Another Problem - The Risk of Too Slow or Too Fast a Rate of Change

The reality is that there is no right degree of change to start with. Some companies prefer a small rate of change. Some require a larger one to fit their culture. The risk of change exists both for making too small a change and for too large a change.

This is especially true if making a significant improvement requires many teams to start together.

# Putting it together – Creating a workflow that works for you

We are now ready to create our workflow. Since many people are familiar with Scrum and/or Kanban, two templates are provided as a start. Feel free to ignore the starting "suggestions." They are only there for reference. If you are currently using Scrum or Kanban, you may find them helpful to identify what changes you should make as a start – no need to throw away what you've done and have learned.

It is important now, however, to decide if timeboxing or flow will be your base planning method.

The Amplio Development approach has 30 capabilities, while Scrum has only 17 roles, events, artifacts, and rules. Scrum requires you not to change anything Amplio requires you to think and decide what works. At first glance, Amplio development looks to be more complicated than Scrum. But what is simple has several aspects to it. There are many things to consider. These include:

1. How easy is it to get started?
2. How fit are the practices you start with for your situation?
3. After you get started, how will you learn to improve?
4. How does the approach handle the fact that all systems are incomplete while attending to Edgar Schein's maxim "we don't think and talk about what we see, we see what we are able to think and talk about"?

Scrum has an easy way to get started because it only allows for one set of roles, events, artifacts, and rules. But it doesn't address the other 3 points. Amplio takes the approach of having a fuller system available but to have a simple starting point. This can be accomplished one of two ways:

- Do an analysis of how you should start. Find appropriate implementations of the capabilities and implement those that are appropriate.
- Start as if you are going to start with Scrum. This is as simple as starting with Scrum but then provides for going deeper as needed as well as providing additional concepts as needed.

While reading the "whys" of the capabilities, you learn how good practices reflect first principles. Over time, you should gain some skills in matching practices to achieve alignment with these first principles.

You can also attend to how well your teams are doing. I provide the following table to make a note of this as you go through things. This can be useful to see where improvement is needed. Later in the book, we'll provide a more detailed assessment to create an improvement backlog for your team.

## Three approaches to take

When deciding how to start there are three basic approaches you can take:

1. Start with Scrum
2. Tailor to ensure a fit-for-purpose solution
3. Create a fully customized approach

### Start With Scrum

Amplio can easily accommodate you if you've decided, or been told, to start with Scrum. If your team is already familiar with Scrum, this may be a good choice. Amplio will let you adjust your methods as you learn or have situations where Scrum doesn't work well.;

## Tailor to ensure a fit-for-purpose solution

While many Scrum aficionados often won't admit it, Scrum does not work everywhere in knowledge. Here are a few of the situations where you will need to tailor your approach away from Scrum to some extent:

1. **You can't get cross-functional teams.** Cross-functional teams are usually ideal, but not always. And very often they are not advisable when there are not enough people with specialized skill sets to go around.
2. **You can't plan ahead.** Scrum uses sprints, which are timeboxed, for planning. This requires you to be able to look ahead for at least the length of the sprint. If you're responding to challenges of customers you may not be able to do this. Teams that provide support to other teams often can't use Scrum well either if they don't know when this support is needed.
3. **You do business on the basis of fixed cost, scope, and time contracts.** These are a fact of life for many organizations. Scrum is not well designed for project-based work because it doesn't focus on efficiency as much as it could.

If any of these conditions apply, you need to modify the Scrum approach.

## Create a fully customized approach

While creating a fully customized approach is often risky if you do it on your own, Amplio enables you to do this with little risk. It does this by identifying the capabilities you need to implement as well as provides you with guidance on how to decide which options to use.

| Capabilities | Scrum |
|---|---|
| 1. Requirements and Artifacts | |
| 1.1x Identify stakeholders and what success means to them. DTA | |
| 1.2x Work on small, releasable items of value. DTA | |
| **1.3 Ensure you're creating the greatest value. DTA** | |
| 1.4x Have high product quality from the customers' perspective. | Suggested |
| **1.5 Validate the requirements with examples.** | |
| 1.6x Attend to the customer journey while creating requirements. | |
| 1.7x Have definitions of ready (DoR) and definitions of done (DoD) with full kitting DTA | Just DoD |
| 2. Managing the workflow | |
| 2.1x Create visibility of work and workflow | Suggested |
| 2.2x Manage work in process to remove delays in workflow and to lower risk | |
| 2.3x Have a product backlog serve as an intake process | Specified |
| 2.4x Have a near-term backlog from which you pull your work | Specified |
| 2.5x Use pull methods to keep workload within capacity | Specified |
| 2.6x Build in small, vertical, slices. MB | Implied |
| 2.7x Agree on classes of service and their corresponding service agreements DTA | |
| 2.8x Use DevOps when applicable | |
| **2.9 Use automated testing to eliminate waste** | |

| | |
|---|---|
| 3. Learning, Improving, and Pivoting | |
| 3.1x Prepare for the start of a project or for the start of building a product | |
| **3.2 Create value in small steps to get quick feedback** | |
| 3.3x Attend to risk with feedback DTA | |
| 3.4x Have an agreement on discussing challenges on a frequent basis MB | Specified |
| 3.5x Frequently step back and reflect DTA | Specified |
| 3.6x Know how to select a more appropriate practice | Denied |
| 4. Roles | |
| 4.1x Decide who is the Product Owner in Scrum DTA | Specified |
| 4.2x Developers | Specified |
| 4.3x Decide on the responsibility of the team coach DTA | Specified |
| 4.4x The role of management | |
| 5. Organizing teams and backlogs for effective value streams | |
| 5.1 Have an effective value Creation Structure DTA | Specified |
| 5.2 Shared backlogs | Specified |
| 5.3 Aligning market solutions to teams | |

DTA – a decision-guiding table is available
MB – a Miro board to track information/decisions is available Amplio is a set of capabilities.

# Four factors to consider when starting a new project or initiative.

## Why we need a good transition strategy.

tbd

## Factors to Consider When Starting a New Initiative

There are four very important factors to consider when starting a new initiative:

1. What is the degree of **disruption** the people involved can tolerate
2. How much **change** is ideal
3. How **fit for purpose** do your new practices need to be
4. How people work together

People often confuse disruption with change. They are not the same. When coming up with a new way of working, let's use the term change to refer to how much we are changing how we work. We'll use disruption to mean how much our current methods are adversely affected by these changes. Thinking of change and disruption this way, we see we want change, without disruption.

Changes that improve our workflow are good. While these may require disruption, we prefer to minimize it. We must attend to the fact that the opportunity for disruption changes over time

One must be cautious when one applies a fixed set of practices. Two potential problems occur. The first is that the end result may not be fit for purpose. The second even if it is, it may be more disruptive than it needs to be.

Systems thinking tells us that not having a fit for purpose solution will cause more disruption than having one. Disruption causes resistance.

The idea that less process is always good is misguided. There is good process, and there is bad process. You may want more good process. You definitely don't want any bad process. But process itself is neither good nor bad.
Process that helps people is good. Process that doesn't is bad.

For example, the checklist a pilot uses on a plane before takeoff is a good process. if we think process is bad, we won't find good process. Amplio uses the term "workflow" because it's a less loaded term.

Specifying that we should have explicit agreements amongst ourselves as to how to work is a kind of process. if you focus on improving people in a bad system you likely won't get good results - the system will remain bad, putting down the people

If you impose "improvements" on a system without including people you're likely to make the system worse. Bad systems defeat good people. The role of management is not to manage people, but to help manage improving the environment that people work in. People often don't do this on their own. They often don't want to. They want to do what they consider to be their real job.

When we are about to do some improvement initiative, we should ask:

- How disruptive can we be?
- How disruptive should we be?
- How fit for purpose are these changes?

Not asking these questions is risky.

Having a focus on workflows without a focus on people won't get you a good working environment. Having a focus on people without attending to how they can work together won't get you a good working environment. You need both. This is an example of systems thinking.

Having a great customer focus without a great system means the customer won't get value. Having a great system without a customer focus means the customer won't get value. You need both. This is systems thinking

Is a focus on how people are working together a focus on people or on their workflow?

You can’t separate these two.

## Scrum Sidebar: Scrum’s claim about simplicity and other myths

**Scrum is simple.**
Scrum proponents make several claims about the importance of a simple approach. In its endeavor to create something simple, however, they have actually made it more complicated.

We must notice that “simple” is in the eyes of the beholder. There is a difference between how simply something is designed, and how simple it is to use. *But it must also be*

*appropriate for use.* A radio station that has just one channel is simple to build and simple to use, but it *isn't fit-for-purpose* unless you love that one station.

Scrum may be simple to start with. But Amplio can start just as easily as Scrum - just set the dial to Scrum's decisions.

**Scrum being based on empiricism is a good thing.**
Scrum is based on empiricism, but its lack of theory requires it to be immutable. This lack of theory makes it *harder* to use.
In reality, all approaches incorporate empiricism. Effective ones also use first principles.

**One can only inspect and adapt since you can't get to a root cause.** While this is never stated in the Scrum guide, the Scrum approach does not provide any cause and effect which would be very useful.

**Scrum applies everywhere complex problems are being solved.** Scrum doesn't apply everywhere, as discussed in an earlier Scrum sidebar.

**You need a simple approach to start, and this requires being purposefully incomplete.**
Amplio can be as easy to start as Scrum. Yet it provides considerably more than Scrum.

# Common Myths You May Have to Overcome

## Myth: Fractals apply to behavior in knowledge work

Many people have observed that organizations can be thought of in a fractal manner.

Fractal structures are those that appear similar at different scales. Examples are rivers, broccoli, and trees.

It is often mistakenly inferred that therefore we can look at organizations in a fractal manner - that what works for a team will work for a team of teams.

This ignores that although a team of individuals may appear similar to a team of teams in structure, the dynamics are quite obvious.

The reason is that the relationships between the individuals in a team are quite different than those between teams. Identification, motivation, rewards and more are different. There may be a fractal structure but there is not a fractal relationship. Systems are not defined by their sub-systems. Believing this is true is a systems-thinking anti-pattern.

One way to demonstrate this is to consider physical nature. See the following structures:

- subatomic particles to atomic particles
- atomic particles to atoms
- atoms to molecules
- molecules to cells
- cells to living beings
- living beings on Earth
- Earth in solar system
- solar system in galaxy
- galaxy in universe

There is a fractal nature here.

But the forces are considerably different.

## Myth: You must follow until you understand <Amplio_Scrum_start>

It is a myth to say you must start an adoption following until you understand.

It’s possible to understand before starting.

But you must learn a few of the first principles - and you likely already know these as dormant knowledge.

First principles are the irreducible truths that describe how things work.

Here are a few:

- As a rule, you can complete smaller things faster than larger things.
- Overloading people causes multi-tasking and delays in the workflow
- Multi-tasking and starting and stopping work delays feedback which increases the likelihood of wasted effort

You can tell how well you're doing this by using some of the factors for effective value streams (FFEVS):

#1 Do you have clarity on what success is to key stakeholders?

#2 Are you working on the most valuable items to achieve success for stakeholders?

#3 Are you building them in small steps?

#4 Are you avoiding overloading your people?

#5 Are you getting actionable feedback quickly?

I often hear this harmful mantra from Scrum enthusiasts, so let's see how we can apply these insights to Scrum.

Notice how Scrum is consistent with these:

1. Increments are a manifestation of #1, #2
2. They also have us work on smaller things to release sooner.
3. Timeboxing and only working on what we can do in the timebox can avoid overloading us. This avoids having people be overloaded.
4. Combine not being overloaded with having cross-functional teams (having the skills we need) we can avoid stopping and starting work and get feedback on what we're doing quickly.

You can validate this with your own experience. Since you had this experience even before you started Scrum, you can relate to these at the start – that is, you can understand why it works.

But taking this approach – understanding from the beginning – has other ramifications.

Without this understanding, we fall prey to Eli Goldratt's observation that "A comfort zone has less to do with control and more to do with knowledge" and revert to old habits (as does management).

You also feel better working with understanding. You're more proactive. You learn faster as illustrated in Deming's comment "Theory without experience is useless. Experience without theory is expensive." We need empiricism and theory. This enables you to find solutions quickly.

You can solve your challenges by seeing how you are violating first principles, the principles that guide you to effective value streams.

"We can't solve problems by using the same kind of thinking we used when we created them." - Albert Einstein

<Amplio_Scrum_end>

## Upcoming Myth: Fail fast is a good thing

## Upcoming Myth: You must reorganize to be effective at Agile

## Upcoming Myth: You should not be doing projects

## Upcoming Myth: You can't do fixed scope, cost, and time projects

In a nutshell - create quick feedback, lower waste, reduce risk, get higher quality, talk to clients if the need changes (they usually will take better for same cost).
A combination of flow, lean and Theory of Constraints does this.
You might lose on some projects but generally you'll come ahead.

People who are forced into doing fixed cost, scope, and time projects are not so bad at estimation as Agile folks are. Interesting insight right there. Because Agilists think it can't be done I suppose. Projects are not waterfall. They are constrained contracts. How you address them is up to you The myopia the Agile space has on them is great

## Upcoming Myth: As you speed up quality goes down

# Metrics

# The Amplio Scrum Guide Summary <Amplio_Scrum_start>

Note from Al Shalloway. I may write a separate book on Amplio Scrum. But in the meantime this section services as a guide for people wanting to improve their Scrum. It lays out some new material and provides a table of contents for specific Scrum related materials.

The Scrum sidebars in this document are listed here:



## Additional Scrum Sidebars

### Scrum Sidebar: Scrum is like chess

I hear that Scrum is like chess fairly often. This is used to justify why Scrum has immutable rules and why to say if you don't follow them then you're not doing Scrum. But games in general and chess in particular are not a good metaphor, Here's why:

**Chess is designed to be difficult to master.** Improving how to add value to customers should take as little effort as possible to learn. Many games are designed to be difficult to master so that you can keep playing them and have fun. Some games are designed to be easy to master, but aren't as much fun after a while (tic-tac-toe).

**There is little investment in learning a game to see if you like it.** Trying chess out for a couple of hours to see if you like it is not a big investment. Two people for two hours is 4 person hours. On the other hand, sending someone to a 2-day Scrum class, teaching the team Scrum, and trying it for even just 2 sprints (probably four weeks) is a massive investment. Discovering at that point it's not fun (or rather, effective for the team) is huge. The team has likely already invested their training budget. It's not like they can go to the store and pay $40 for another game to try.

**Chess is about having fun.** While we want to have fun in our work, that's not the point of it. While chess can be said to be more fun the more difficult it is, Scrum is less fun when it's difficult.

**Chess' rules are made up.** Chess' rules were created by someone. Actually, so was Scrum. So maybe the metaphor holds here. But this is not a good thing. Approaches that are based on how the world works (e.g., first principles) have more applicability than those based on some guru's theory of what works.

**The bottom line.** For those who want to explain Scrum's immutability away by comparing Scrum to Chess, I suggest you look at the purposes of the games. Chess is not about creating change. It's about playing as best as you can in a game designed to be difficult to master.

Being Agile means being able to change in the face of reality. I suggest that a "game" designed to be effective in creating change must be able to change itself.

*The one place Scrum is like chess: if you change it it won't work. But this is not a good thing. Better to have a more flexible framework. Making it immutable shuts people's thinking down and often makes it so better solutions are not considered.*

## Making Scrum Lean

This chapter takes many concepts presented in this book and applies them to Scrum. The diagrams here are from earlier chapters. You can read more by taking the links in them.

The Scrum Guide speciously claims Scrum is based on empiricism and Lean thinking. In 2007 I suggested that people doing Scrum considered it a _partial_ implementation of Lean. Both Ken and Jeff vehemently denied it & threw me off the Scrum Development board when I suggested it a second time - Ken saying "this is a Scrum group not a Lean one and I should start my own group if I wanted to talk about Lean."

Not much about the structure or mental models of Scrum has changed in the more than two decades since I first started using it. Lean has a different mental model than Scrum.

*"It is impossible for a man to learn what he thinks he already knows." Epictetus*

That said, if people did Scrum consistent with Lean's mental model, their Scrum would get better.

Lean is first and foremost about creating value for our stakeholders / customers. Lean takes a holistic approach - focusing on optimizing the whole Even if we're looking at just improving the team we want to do that in the context of the whole.

The essence of Lean is:

- create value for stakeholders / customers
- systems thinking
- eliminate waste by removing delays in the workflow and creating quicker feedback
- continuous education
- just in time
- build quality in
- use a pull system across the value stream not just in starting work
- have small batches
- explicit workflow

- have cross-functional teams when possible but flow when you can't
- build a model that explains how things work and continue to improve it with Plan-Do-Study-Act PDSA)

Let's look at how embracing each of these would improve how you do Scrum

**Create value for stakeholders / customers.** The first step here is to know who our stakeholders are, what success means for them, and what value needs to be added to achieve this. This also means that we need to have our eye on the whole value stream. Hence, the mantra “optimize the whole.” We must be aware of the fact that local optimizations often lead to overall poorer performance.

**Systems thinking** tells us many things but the most important one is that systems cause the overwhelming number of errors that occur. This gives us two deep insights – one, that we must improve the system first and not focus so much on the people. We trust our people, we don’t trust our system. The second insight is that we can take advantage of the challenges of others. We don’t have to hit them ourselves.

> *“There are three kinds of men. The one that learns by reading. The few who learn by observation. The rest of them have to pee on the electric fence for themselves.” Will Rogers*

**Eliminate waste by removing delays in the workflow and creating quicker feedback.** One often hears the Lean mantra is “eliminate waste.” In the manufacturing process that is pretty sure. In knowledge work it’s more about eliminating the cause of waste. While we can’t avoid errors we need to identify them quickly and make quick corrections. We do this by eliminating delays in the workflow. This is done by managing work in process throughout the workflow. While Scrum suggests limiting work in process by limiting what can go into a sprint, we need to manage work in process throughout the sprint.

What needs to be appreciated is that you can estimate the correct amount of work that can be done in a sprint, but because you don’t manage work in process throughout the sprint you won’t get it all done. This is because peaks in the work in process amount during the sprint causes delays to feedback and this causes waste. Even though the overall estimate might have been accurate, no one accounted for this unplanned work.

See Focus on getting feedback while lowering WIP and reducing accumulated risk for more.

**Continuous Education** means we strive to detect errors immediately. Not wait for daily meetings or end of sprint retrospections. Early detection can be a kind of prevention. This, coupled with systems thinking means we aim for continuous improvement. It’s part of the reason for “stop the line” in Lean. If something has gone wrong it will continue to go wrong. Short retros weekly can help considerably.

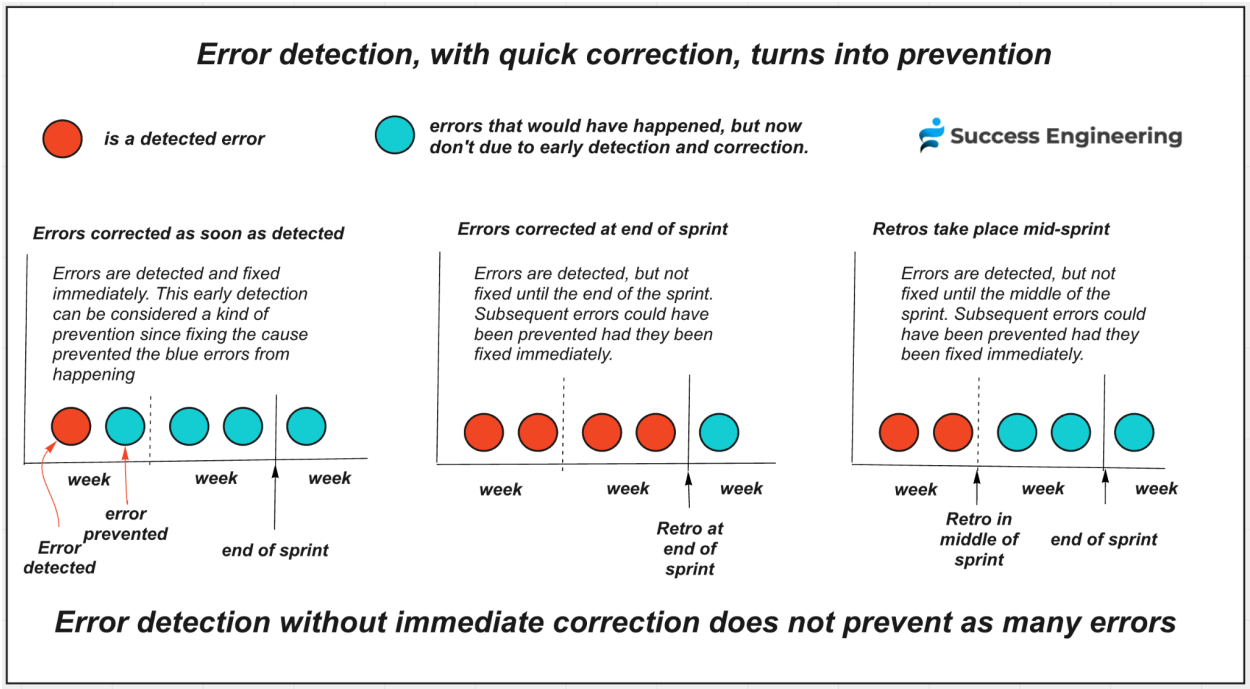



Figure: Error detection, with quick correction, turns into prevention

See Continuous detection is a path to quick prevention for more.

**Build quality in** should tell us to do acceptance test driven development and automated testing.

**Pull systems** should be employed throughout the workflow. Not just in deciding what will be done in the sprint. In fact, sprint planning is a kind of push system like the one waterfall uses only on a smaller scale.

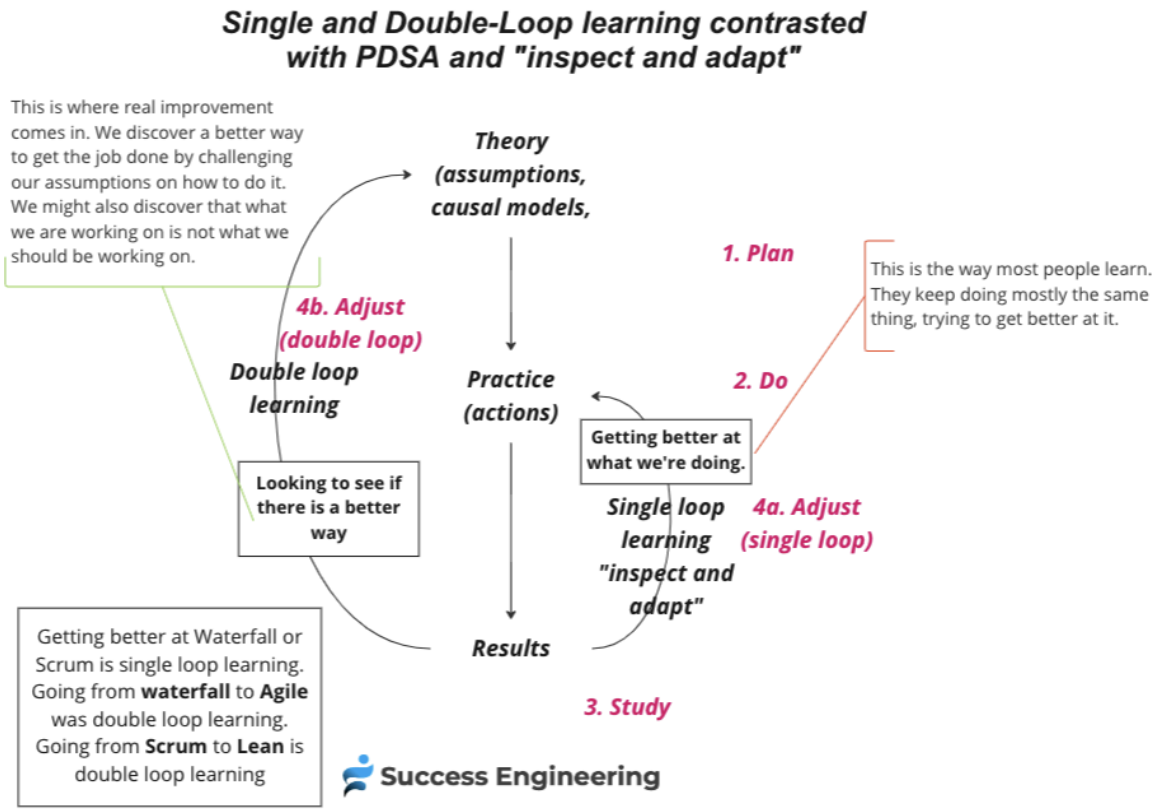
See Create a focus on finishing for more.

**Smaller batches** are critical. Scrum has improved somewhat here in 28 years going from 30 days or less to most teams do two weeks. The better ones use one week. Releasing increments now instead of at the end of the sprint is a bit of Lean embedded in Scrum.

Lean (via Amplio) suggests using a combination of MVPs and QVRs as appropriate. See Agile Artifacts for more.

**Explicit workflow** means to make how you are working be known by everybody. This should be reflected on the board. A Kanban style board works great in Scrum. We don't follow it, rather it reflects how we think is the best way to do our work. See Creating visibility of work and workflow for more.

**Use PDSA.** Scrum is based on empiricism but doesn't suggest creating a model to understand what's happening. Lean actually provides that model. Instead of waiting for impediments to happen, Lean can actually predict what your problems are by looking at the way you're working, how your team is structured and how it interacts with others. "Inspect and adapt" merely takes advantage of what's happening, but doesn't, on its own, improve your understanding. "Plan-Do-Study-Act" (PDSA) has us challenge our assumptions about what's happening. This is called "double loop learning" (see diagram). Scrum has many assumptions in it and doing this may have you go beyond the boundaries of Scrum but into a more effective workflow.




See Single and double loop Learning for more.

Lean reflects Jerry Sternin's observation "It's easier to act your way into a new way of thinking, than think your way into a new way of acting." In other words, instead of trying to motivate people or talk to them about collaboration, getting them to work together in effective ways increases the behavior you want.

**What happens when you're not using Lean in Scrum**

An example of ignoring these practices is that most new Scrum teams open too many stories at the start of a sprint. This causes several problems.

- people aren't looking for opportunities to collaborate
- many stories are completed at the end of a sprint
- we can't take advantage of what might have been learned by stories being completed early

If we were attending to systems thinking we'd be learning by watching others. Scrum's attitude of having people figure things out on their own means teams face trouble at the beginning. Many teams never get through this struggle and start to deviate from Scrum. The fact that this happens so much needs to be recognized as a problem of Scrum, not the people doing it – remember systems cause most of the problems. The Scrum Guide may not change to correct this but there is no reason you can't incorporate Lean into Scrum and fix these errors.

## Why we need first principles

Unfortunately, Scrum does not present first principles. Instead, the creators of Scrum have created their own principles. Instead of incorporating theory into experience, Scrum relies on empiricism (actually, relies on empirical process control) and complex adaptive systems.

Teams often find themselves in a situation where a Scrum practice doesn't seem to be working. Unfortunately, it is difficult for a team that has not been armed with first principles, to tell if the problem is they aren't following the practice properly or if the practice is not the right one to follow.

The fact that there is no set of practices that is universal sets up a dilemma for these teams. They try to make Scrum work but eventually feel they must do something different. Unfortunately, different is not always better and the practice they move to may not solve their problem but merely avoid it. This is the infamous "ScrumBut" defined by Scrum.org - "Scrum has exposed a dysfunction that is contributing to the problem but is too hard to fix. A ScrumBut retains the problem while modifying Scrum to make it invisible so that the dysfunction is no longer a thorn in the side of the team."

The solution here is to understand first principles and find a practice that fixes the problem by following first principles better than the prescribed Scrum practices. Scrum's immutability is a symptom of the lack of first principles. Saying Scrum's practices are immutable and then blaming people for changing them obscures the real responsibility for what's happening.  Without providing a way to change its practices, Scrum lays landmines for its adopters who, when stuck in a difficult situation, and without being given a way to tell if a change is good, has, in their frustration, make a change, which often makes things worse (e.g., abandon timeboxing without following practices from flow). Scrum proponents then declaring "well they weren't doing Scrum" merely abdicates the responsibility that they weren't prepared to remove the impediment they were faced with.

## What Amplio Provides Here and How, If You're Using Scrum, to Avoid This Problem

Amplio provides several approaches to solving this problem in different chapters in this book. They include:

- A list of some key first principles
- The factors for effective value streams which presents first principles in an actionable format
- How to change a practice with confidence it will be an improvement

## Common Problems Scrum Teams Have That Amplio Can Help With

Amplio Scrum can avoid many problems commonly seen in Scrum. Some of them are listed here. Please add a comment if you have seen others and we'll add them to the book.

Systems thinking tells us that the system people are in causes most of the problems seen. Deming made a distinction between these "common" causes and "special causes." When different Scrum teams experience the same challenges the problem is almost always due to the system people are in. While it is easy (but not responsible) to blame the people involved, systems thinking tells us we must improve the system.

There is a metaphor between executives imploring people to just work the plan and Scrum Master imploring people just follow Scrum

## Conflicts Between the Product Owner and Scrum Master

There is a false dichotomy between the roles of the product owner and Scrum Master. Many people believe the roles are in conflict with each other - that POs want more done and SMs need to protect the team.

When viewed this way there will be conflict between these two roles.

But this is the wrong view. The correct view is that both the Product Owner and Scrum Master are committed to the team creating as much value for the customer as possible. They are on the same side - just have different responsibilities.

The Scrum Master can assist their product owner by recognizing this and discussing how the two of them together can help the team create the greatest value.

This requires understanding of some of the theory of flow which tells us that when work levels exceed capacity, multitasking occurs and waste is created. When product owners understand this they won't push too much work on the teams.

The bottom line is that Product Owners and Scrum Masters aren't adversaries. They're allies, collaborating to maximize value.

## Avoiding Scope Creep at the End of a Sprint

Many teams experience sprinting to the end of a sprint, thinking they've made it, only to realize scope creep has hit them. There are a few things missing in Scrum which most teams never figure out that inadvertently facilitate what you are describing. Scope creep means not only that we've identified new things to do but that we aren't able to drop other things being done. Much of this is due to not being able to get execs, product owners, and management to understand the theories of flow which are missing in Scrum. So the cause is a combination of being pushed while not managing the work. Steps that can help:

1. You must manage your work in process throughout the sprint. Many teams open up too many stories at the start of the sprint. While Scrum is based on empiricism, it doesn't seem to apply it to itself. Inspect and adapt focuses on the work but should also focus on Scrum itself (which is disallowed of course because it's immutable and if you DID make a change you'd no longer be doing Scrum). See "Manage work in process to remove delays in workflow and lower risk."
2. Use Quickest Valuable Releases. This is a better focus than sprint goals.
3. Educate execs, POs, and managers in the theories of Flow, Lean, and the ToC.
4. Employ double loop learning on Scrum regardless of its insistence on being immutable.

# Appendices

These appendices are not essential readings to understand and use Amplio. In fact, are intended to be stand-alone articles that address issues that many Agile coaches face in their work.

## Waterfall Is Never the Right Approach

I often hear that there are cases where Waterfall is best.  I don't agree.

Let me first point out, however, that the choice is not between waterfall and Agile. There are other approaches. There are also multiple aspects to consider:

1. Do we know the steps we need to take to accomplish our task?
2. Are all the people and capabilities we need to accomplish the task readily at hand?
3. What is the chance of making an error?

I don't believe adopting waterfall as an approach is *ever* a good choice.  Waterfall comes with the following mental model::

1. we don't need feedback between the steps involved. In knowledge work this is often between analysis, design, code, and validation
2. we can hand work off with little or no cost
3. big batches are ok since they enable us to be more efficient
4. specialized skills working only on their specialty is good
5. we can understand the work to be done before we do it
6. written requirements can specify what we need

### Going through each of these:

**1) We don't need feedback between the steps involved.** In knowledge work, evidence shows we do so that we can correct our misunderstandings. In manufacturing, or set process type work, the feedback is useful to detect when an error has taken place. In both cases not getting feedback creates waste. We understand requirements better when we do analysis.  We understand analysis while in design… Interestingly enough. Royce's original paper said waterfall wouldn't work.  People liked the simplistic model he presented instead of the one with feedback cycles built-in.

**2) We can hand work off with little to no cost.**  When we hand work off, we lose a lot of information. We either have to re-learn that information or, more likely; we go with misunderstandings.

**3) Big batches are ok since they enable us to be more efficient.** This has been proven inaccurate in many industries and is the heart of Lean.  Eli Goldratt said it best –  "Often reducing batch size is all it takes to bring a system back into control."  Waterfall does little to have us work on small batches – a vital thing to bring value to market quickly.

**4) Specialized skills working only on their specialty is good.** Synergy is good. Cross-functional teams, when possible, are good.

**5) We can understand the work before we do it.** I hear a common refrain that "our customers know what they want after we show them what they don't want."  This is because requirements elicitation is a discovery process.

**6) Written requirements can specify what we need.** I'll leave this to people's experiences, and one of my favorite Churchill quotes – "This report, by its very length, defends itself against the risk of being read."

In other words, the waterfall mental model is just not accurate. I would suggest where waterfall worked, people went beyond its mental model and did something else. For those who say, "Well, no one does pure waterfall, " I suggest looking at what they did outside the waterfall that worked.

## What Can We Do?

In knowledge work, we first need to realize that we should not be trying to do Agile. We should be trying to be as agile as possible – increasing our agility as an organization. This means:

- focusing on delivering business value quickly by selecting the most important things of value (this typically means smaller chunks to be delivered)
- having proper team structures to get the job done
- removing delays in workflow by avoiding working on too many things (work on the most important ones without exceeding your capacity)
- shortening feedback cycles is always a good thing to get clarity on where you are and to detect errors quickly

There are more, of course. But I've picked these because these are things we can always do to some extent.

In manufacturing or processes where we understand what is to be done, Lean thinking helps by detecting errors quickly and continuously learning how to do things better.

*One other note. We should remember that waterfall is to Agile as mass manufacturing is to lean manufacturing. In manufacturing, we know all of the steps to do but working in small batches with feedback is still better than big batches with testing at the end.*

# Why Lean-Agile Should Be More Predictable Than Waterfall

Many executives have been led to believe that Agile is inherently less predictable than a waterfall approach. However, when wrapped in Lean thinking, Agile can be more predictable because it enables working directly on the true causes of unpredictability in software development. Waterfall's large projects and stage-gate approach cause delays in feedback, workflow, testing, and integration. These delays inherently create a significant amount of rework (redoing requirements, reworking code that missed requirements, finding bugs, thrashing during integration). This work, of course, is never planned for and therefore results in inherently bad estimation, not to mention that eliminating this extra work improves productivity.

In addition, Lean-Agile methods have us focus on smaller, higher-density pieces of work. This alone helps bring our systems under control observed by Eli Goldratt – "Often reducing batch size is all it takes to bring a system back into control."

The key to having predictable results is eliminating delays, creating unplanned work. This requires smaller batches of work that are appropriately sequenced in importance, properly formed teams (or groups of teams), and solid technical practices. Agile suggests additional methods such as Acceptance Test-Driven Development, automated testing, and continuous integration. These directly improve understanding of requirements (through quick feedback), code quality, and risk reduction.

Although waterfall and Agile approaches have different mental models, some of Agile's methods can be incorporated into a waterfall process. For example, making both projects and planning cycles shorter.

Improving methods will not make an organization Agile, but it may help set the stage for a true transformation.

I've heard it said that "we can't go Agile because we need predictability." This is an understandable sentiment because many Agilists have dismissed predictability as impossible. However, there are two camps in the Agile community – one that is team-centric and another that wraps Agile with Lean thinking (we call this Lean-Agile). A team-centric approach will often not achieve predictability when applied at scale because the team is only one part of the value-add cycle. But a Lean approach is holistic, and although it may not be possible to predict precisely what must be built, it is possible to predict *how much* can be built. With proper selection of the work to be done,

However, more than simple iterations are required to achieve this. Lean-Agile should not be just a change in mental models but must incorporate solid product management and technical practices. Let's consider why projects (regardless of approach) often do not achieve the predictions they make. The most discussed are:

- what the customer wants is not understood
- the time to build something is unclear
- the code is fragile and hard to change

There's an even bigger one lurking here (which I'll discuss a little later) but for now, let's look at these.

**What the customer wants is not understood.** Customers often don't know what they want until they see what they don't want. This implies they should be shown "what they don't want" as soon as possible. This is why it's important to build small segments of work quickly. This also shortens the time between all stages of the work (requirements, design, code, test, integration). This combination of quick feedback with coding and testing being done together enables quick course corrections as both the customer and the development group learns what is truly of value.

Acceptance Test-Driven Development (having customers, developers and testers specify requirements in the form of test specifications) achieves better clarity on what is needed and avoids many misunderstandings altogether. Getting the customer to provide requirements in this precise manner changes the nature of the requirements. Vagueness is no longer allowed, and customers' understanding of what they are looking for improves.

**The time to build something is unclear.** The answer to this is again to go small. Estimating the time it will take to complete a large chunk of work is typically less accurate than estimating the time for smaller things. Some people have ridiculed estimation because they *are* often wildly inaccurate, and managers sometimes abuse the teams for that. But bad estimates are more of a symptom of other issues – unclear requirements, poor code quality, and even fear of management. But avoiding estimation abandons looking at the actual underlying reasons.

**The code is fragile and hard to change.** Poor code quality *will* slow things down, but this does not have to be the path taken. When code is fragile and is difficult to change, it will *not* be possible to accurately predict how long things will take, even for small changes. But that's because of the poorly written code, which is both difficult and risky to change.

Waterfall's approach addresses this issue by doing all coding first and then testing afterward. There is often a perceived need to do this because of the need to batch test. But this doesn't address the issue (poor code quality). Accommodating poor code quality will not improve it but tends to get the code in a slow death spiral that will eventually need to be re-written.

Poor code can be improved with automated acceptance tests. The time test automation saves is only part of its value. It also enables refactoring of the code in safety because defects introduced will be detected quickly. Developers fix these bugs much faster because they are detected while still fresh in their minds. This is a considerable time-saving. Most developers think they spend a lot of time fixing bugs. But on reflection, they see that most of their time is spent *finding* the bug. This is not mere semantics. Bugs detected immediately are fixed much faster than bugs detected weeks later (even when the code hasn't changed).

## The Common Themes

The first common theme here is that *smaller batches of work allow for greater efficiency* by providing quicker feedback, better estimation, and shortening the delays between requirements, code, and testing. This efficiency is due to the fourth issue I earlier mentioned I was going to discuss. This is because delays in feedback, workflow (e.g., waiting for people), and detecting errors *will cause additional work.*

It is common to focus on how to work faster or better. But it is more effective to discover how to avoid creating the need for rework. Although much rework is due to a misunderstanding, having quick feedback significantly lowers the amount of rework required. Consider how much time is spent on redoing requirements, building features of less value than initially thought, fixing bugs, manual testing, thrashing during integration, …. It should be apparent a significant amount of work done in projects was created by the aforementioned delays. The focus must shift from doing things faster to avoiding delays between the work steps and achieving fast feedback.

## Why Waterfall Will Inherently Provide Bad Estimates

Consider the waterfall planning process. Larger chunks of work are planned to be done in separate steps – requirements, design, code, functional test, integration, and system test. This exacerbates the aforementioned issues by increasing delays in feedback between each step and overall feedback achievable at completion. This creates a significant amount of work due to these delays. However, how much of this time is planned for at the start of the project?

*None. Zero. Nada.*

Estimates in waterfall don't consider the work *that derives from delays – work that often takes as much or more as the planned work.* Planning sessions typically don't include the question, "how much time should we set aside for missed or wrong requirements, pitfalls in our design, or all of the extra time it'll take to find the bugs because testing was delayed?" In other words, one reason estimates are bad is that they do not include a significant amount of the work that needs to be done.

## So Why Should Lean-Agile Be More Predictable?

Lean-Agile takes a different approach with requirements, design and coding, and testing. Working in small batches and continuously course-correcting reduces time spent correcting misunderstood requirements. However, technical methods are still needed. Test-first and automated testing also results in better code. Considering writing tests before writing code is an important lesson from design patterns (although not stated explicitly in the iconic Gang of Four Patterns book, both Design Patterns Explained and Essential Skills for the Agile Developer describe this in-depth). And finally, by using smaller batches of work, the time between code and test decreases, thereby lowering the finding time even more.

Another difference exists in the approach to architecture. In waterfall, developers are supposed to design for the stated requirements. However, they have learned that requirements will change and tend

to over-design to accommodate this. On the other hand, Lean-Agile suggests designing to accommodate change. This is known as emergent design.

## An Extra Benefit of Lean-Agile

Building things in small batches results in learning faster about what is needed and enables pivoting as required. Feedback from the customer uncovers the actual value of each feature or story. Furthermore, customers learn that they don't need to ask for everything upfront. Instead, they can adjust their requirements as they see what's being built and better understand their needs.

Waterfall, oddly enough, encourages customers to ask for everything upfront because little justification is required other than "I need it" when initial requirements are stated. Anything specified later, however, must go through a change review board. Customers quickly learn to ask for *whatever they might need* up-front. Lean-Agile teaches customers they only need to specify what they need *soonest* and that they can adjust as they learn more. While a general understanding of the entire system *is* required to enable an estimate for building, only the general size of the work is needed. The details will come later as more is learned. Accurate estimates on the rate of value delivery enable this.

## Management's Role in Lean-Agile

Management's role is to create a great work environment for teams to work to achieve the organization's strategic vision autonomously. They should leave teams to self-organize but need to help create the context within which the teams can work. This is done by getting input from the teams and other parts of the organization (e.g., product management, business intelligence, ops). *Managers do not manage people but rather improve the ecosystem within which people work.*

## Lessons from Lean-Agile Can Improve Waterfall

Although Lean-Agile is as much a shift in mental models as it is a change in how work is being done, it is possible to take some of its lessons and apply them to a waterfall approach and get an immediate benefit. The key is to better work on smaller chunks of value. In other words, two 3-month waterfall projects will almost certainly deliver more value than one 6-month project. The reasons are that delays are removed by having a smaller development cycle and pivoting on requirements is possible after delivering the first chunk of value. A further benefit is that half the entire project's value is delivered in half the time. To learn more about this, see The Business Case for Agility.

## Summation

It is true that many companies that are not hitting their targets using Waterfall methods may not hit them initially when they switch to Lean-Agile methods. But Lean-Agile provides a vehicle for improving the underlying causes of a lack of predictability. Repeated failings in waterfall persist because management often thinks it would work if "people would just follow the plan." But the plan doesn't include things that are certain to happen (extra work due to delays). This makes "following the plan" inherently impossible. Lean-Agile provides the opportunity to get to the root cause of delays that are creating additional work that isn't being planned for. By working on the root causes of the problem instead of the symptoms, Lean Agile enables a path for true improvement.

# Case Studies Involving Value Streams

This page contains case studies involving value streams. Please provide feedback on others you've seen.

- Using the Value Stream to Get to Root Cause With 'Five-Whys'
- Coordinating teams with backlog management

If you are not familiar with value streams, you should read this chapter Attend to value streams first.

# Amplio From the ACEs

This section includes case studies and how to solve particular problems from both myself and other ACEs (Amplio Consultant Educators). These case studies will refer back to what we've learned in this book.

Story carry over at end of the sprint.

How to deal with interruptions.

How to coordinate work when teams are mostly independent but sometimes require working closely together.

List the practices in the book that can stop.

# Appendix: Going Deeper on Flow, Lean, and the Theory of Constraints

This section is online https://successengineering.works/overview-of-flow-lean-and-the-theory-of-constraints/.

# Glossary

Daily huddle. An alternative term for a daily huddle. We use “daily huddle” instead of “daily standup” out of respect for any one with a disability.

| Amplio Core Competencies - Self Study |
| --- |
| |
| **Theory and Learning** |
| Systems thinking and complexity<br><br>**Descriptions of Systems Thinking by Russ Ackoff**<br><br>1. If Russ Ackoff Gave a Ted Talk This is a great talk on systems thinking by one of the best 12 min<br>2. Systems thinking speech by Russ Ackoff. There are many gems in this talk that go way beyond systems thinking. Worth the entire 70 min<br><br>The Choice by Dr. Eli Goldratt. |
| Coming from first principles and factors for effective value streams |
| Attending to Value Streams |
| Using the generic value stream and value stream mapping |
| Understanding practices in terms of first principles |
| Double and triple loop learning |
| |
| **Workflow and practices** |
| Aligning market solutions to teams |
| Attending to the values of the business stakeholders |
| Strategic  pillars |
| Quickest Valuable Releases and full kitting. |
| Agile budgeting and Lean Allocation |
| Understanding how to achieve the benefits of cross-functional teams when you can't get them |
| Middle-Up-Down Management |
| Know how to select a more appropriate practice. |

| |
|---|
| **Coaching / Consulting** |
| Talking to executives from their perspective<br><br>1. Getting executives to understand the value of quick flow.<br>2. Getting Executives to Understand Agile by Talking About Value Streams<br>3. Talking to executives about why they need to focus on the most important value to be delivered |
| How to talk to people to create a transition from where they are to where they would be more effective. |
| Creating an improvement backlog and transition management to achieve it |
| What's the difference between experts and those with less competence? |
| Trim Tabs |
| |
| **Technical practices for non-technical people** |
| Simple test-first methods to increase understanding |
| Avoiding duplication in products |
| Technical Practices to always do (private construction) |