

## 2. Agile Principles

IT 4406 – Agile Software Development

**Level II - Semester 4**

# Intended Learning Outcomes

- At the end of this lesson, you will be able to;
  - Describe Agile principles
  - Explain how Agile principles differ from traditional, plan-driven, sequential product development

# List of sub topics

2.1 Introduction

2.2 Variability & Uncertainty

2.3 Prediction and Adaptation

2.4 Validated Learning

2.5 Work in Progress (WIP)

2.6 Progress

2.7 Performance

2.8 Comparison Summary of Plan-Driven and Agile Principles

# 2.1 Introduction

- Agile principles
  - Agile principles make organizations robust & antifragile
  - Embracing Agile principles makes the development process and organization robust and at times antifragile to the disorder of uncertain events, avoiding harm and reaping benefits of uncertainty
  - These principles are organized into several categories

# 2.1 Introduction

## Principles



Copyright © 2014, Kenneth S. Rubin and Innolution, LLC. All Rights Reserved.

## 2.2 Variability and Uncertainty

- Four Principles under Variability and Uncertainty
  1. Embrace helpful variability.
  2. Employ iterative and incremental development.
  3. Leverage variability through inspection, adaptation, and transparency.
  4. Reduce all forms of uncertainty simultaneously.

## 2.2.1 Embrace helpful variability

Plan-driven development	Agile Development
<ul style="list-style-type: none"> <li>• Treat product development like manufacturing—they shun variability</li> <li>• and encourage conformance to a defined process.</li> <li>• The problem is that product development is not at all like product manufacturing. In manufacturing our goal is to take a fixed set of requirements and follow a sequential set of well-understood steps to manufacture a finished product that is the every time</li> </ul>	<ul style="list-style-type: none"> <li>• In product development, the goal is to create the unique single instance of the product not to manufacture the product</li> <li>• Some amount of variability is necessary to produce a different product each time</li> </ul> <div data-bbox="975 1006 1787 1192"> <pre> graph LR     Input[Same input] --&gt; Process[A defined process]     subgraph Process         direction LR         A[ ] --&gt; B{ }         B --&gt; C[ ]         B --&gt; D[ ]         C --&gt; E[ ]         D --&gt; E     end     Process --&gt; Output[Same output]             </pre> <p>The diagram illustrates a 'Defined Process' flow. It starts with a box labeled 'Same input', followed by a large box labeled 'A defined process' which contains a flowchart. The flowchart begins with a rectangular box, leading to a diamond-shaped decision node. From the decision node, the flow splits into two parallel paths, each ending in a rectangular box. These two paths then merge into a single path that leads to a final rectangular box. The process concludes with an arrow pointing to a box labeled 'Same output'.</p> </div> <p data-bbox="1207 1213 1555 1249">Fig. 1 Defined Process</p>

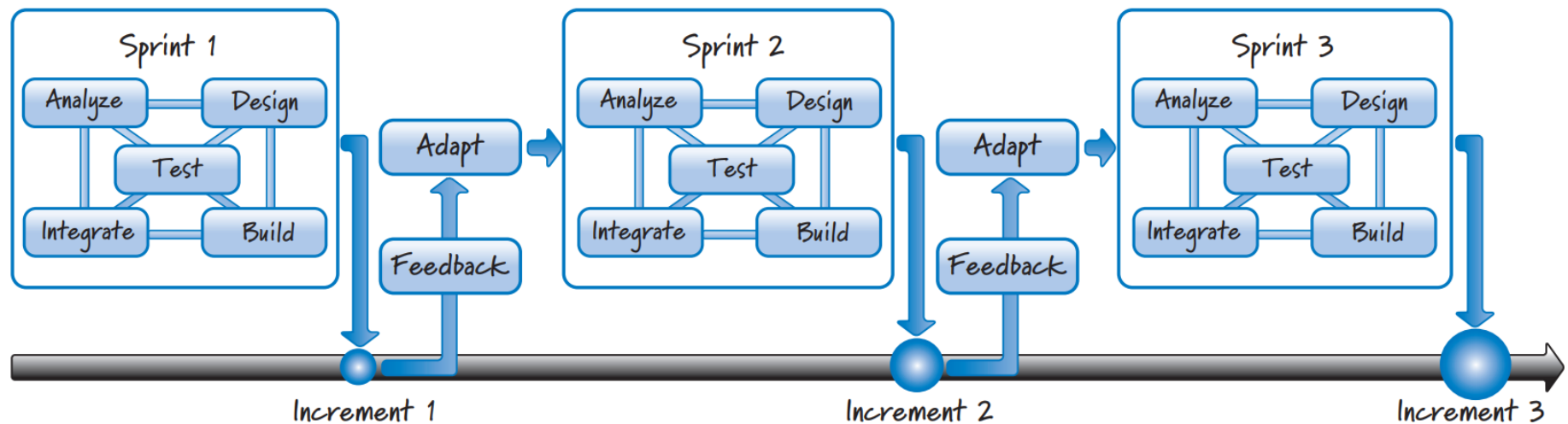
## 2.2.2 Employ iterative and incremental development

Plan-driven development	Agile Development
<ul style="list-style-type: none"><li>Assumes that we will get things right up front</li><li>and that most or all of the product pieces will come together late in the effort</li></ul>	<p><b>Iterative development</b> is a planned rework strategy</p> <ul style="list-style-type: none"><li>use multiple passes to improve what we are building so that we can converge on a good solution. start by creating a prototype to acquire important knowledge about a poorly known piece of the product</li><li>an excellent way to improve the product as it is being developed.</li></ul> <p><b>incremental development</b></p> <ul style="list-style-type: none"><li>“Build some of it before you build all of it.” break the product into smaller pieces so that we can build some of it, learn how each piece is to survive in the environment</li><li>The biggest drawback to incremental development is that by building in pieces, we risk missing the big picture (ex. we see the trees but not the forest))</li></ul>



## 2.2.2 Employ iterative and incremental development

- Usage of “iterative and incremental development” in Scrum



- Sprints : series of time boxed iterations
- During each sprint we perform all of the activities necessary to create a working product increment

## 2.2.2 Employ iterative and incremental development

- No need to work on a phase at a time; have to work on a feature at a time.
- By the end of a sprint we have created a valuable product increment
- That increment includes or is integrated and tested with any previously developed features; otherwise, it is not considered done
- At the end of the sprint, we can get feedback on the newly completed features within the context of already completed features.
- After receiving feedback on the sprint results, we can choose different features to work on in the next sprint or alter the process we will use to build the next set of features

## 2.2.2 Employ iterative and incremental development

- Sometimes we can schedule rework for a future sprint
- Scrum does not require that we predetermine a set number of iterations
- The continuous stream of feedback
- Will guide us to do the appropriate and economically sensible number of iterations

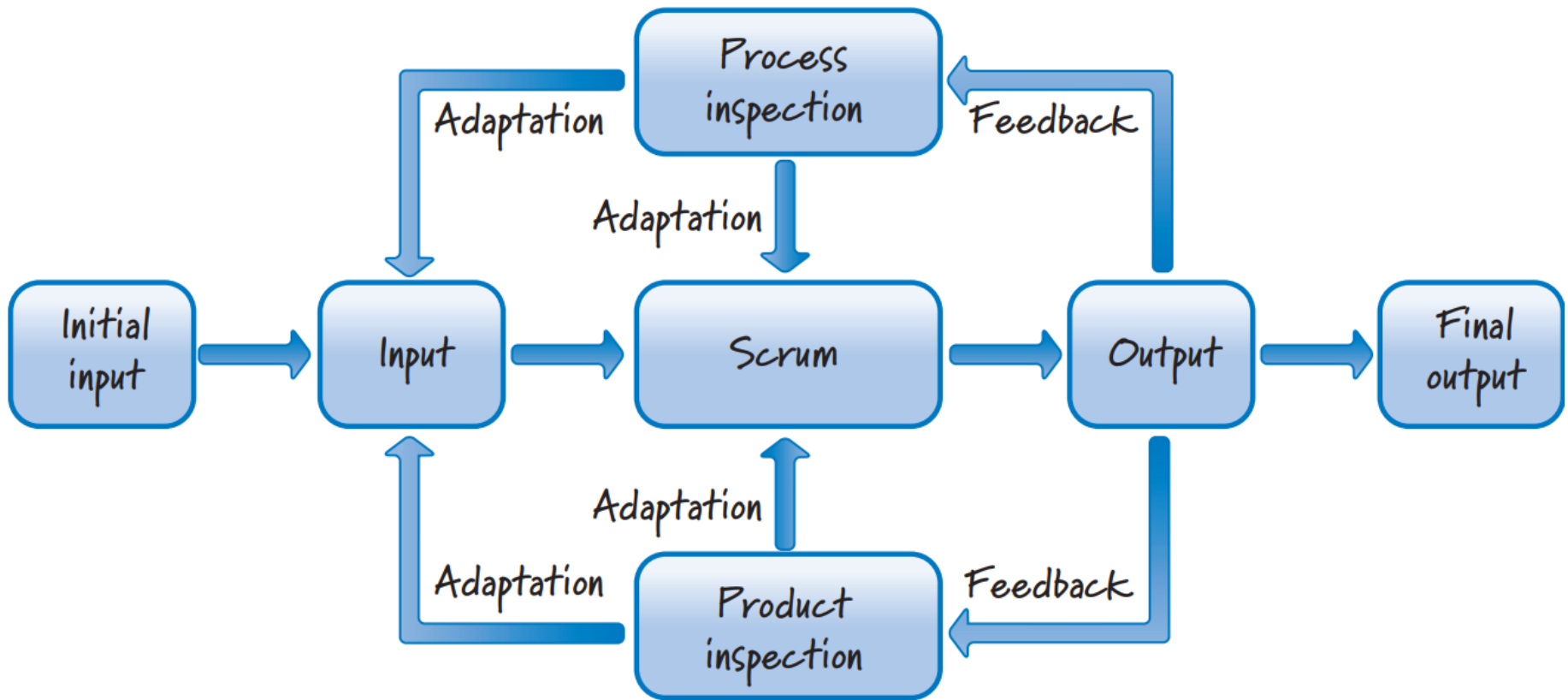
## 2.2.3 Leverage variability through inspection, adaptation, and transparency

Plan-driven development	Agile Development
<ul style="list-style-type: none"><li>• assumes little or no output variability.</li><li>• It follows a well-defined set of steps and uses only small amounts of feedback late in the process.</li></ul>	<ul style="list-style-type: none"><li>• In Scrum, we inspect and adapt not only what we are building but also how we are building it</li><li>• To do this well, we rely on transparency<ul style="list-style-type: none"><li>• All of the information that is important to producing a product must be available to the people involved in creating the product</li></ul></li><li>• Transparency makes inspection possible, which is needed for adaptation</li><li>• It leads to more communication and it establishes trust</li></ul>

## 2.2.3 Leverage variability through inspection, adaptation, and transparency

Dimension	Plan-Driven	Scrum
Degree of process definition	Well-defined set of sequential steps	Complex process that would defy a complete up-front definition
Randomness of output	Little or no output variability	Expect variability because we are not trying to build the same thing over and over
Amount of feedback used	Little and late	Frequent and early

## 2.2.3 Leverage variability through inspection, adaptation, and transparency



## 2.2.4 Reduce all forms of uncertainty simultaneously

- Developing new products is a complex endeavour with a high degree of uncertainty
  - **End uncertainty** (what uncertainty)—uncertainty surrounding the features of the final product
  - **Means uncertainty** (how uncertainty)—uncertainty surrounding the process and technologies used to develop a product
  - There might also be **customer uncertainty** (who uncertainty)

## 2.2.4 Reduce all forms of uncertainty simultaneously

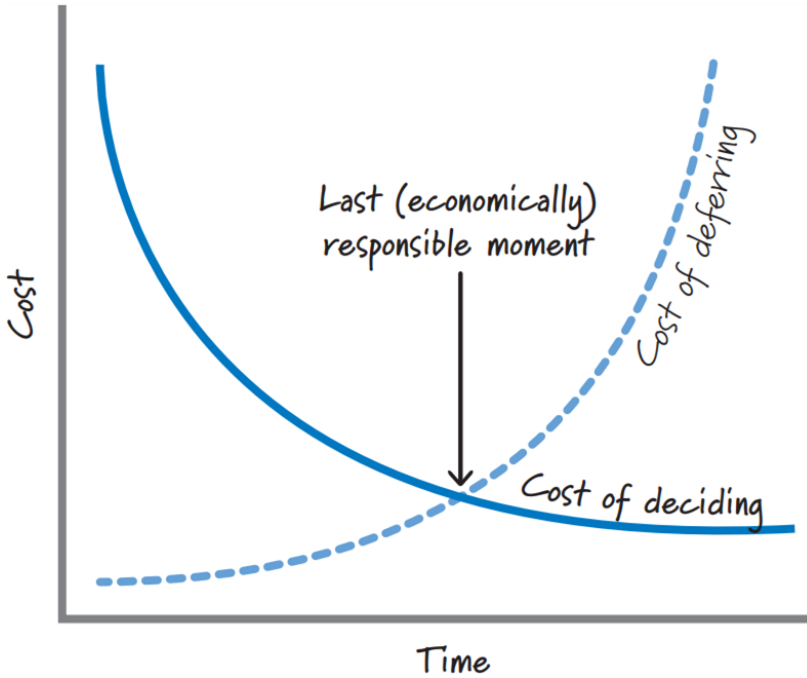
Plan-driven development	Agile Development
<ul style="list-style-type: none"><li>• Focuses first on eliminating all end uncertainty by fully defining up front what is to be built, and only then addressing means uncertainty.</li></ul>	<ul style="list-style-type: none"><li>• Focuses on simultaneously reducing all uncertainties (end, means, customer, and so on).<ul style="list-style-type: none"><li>• Simultaneously addressing multiple types of uncertainty is facilitated by iterative and incremental development and guided by constant inspection, adaptation, and transparency.<ul style="list-style-type: none"><li>• To identify and learn about the unknown unknowns</li></ul></li></ul></li></ul>



## 2.3 Prediction and Adaptation

- When using Scrum, constantly balance the desire for prediction with the need for adaptation
- Five agile principles related Prediction and Adaption
  - Keep options open.
  - Accept that you can't get it right up front.
  - favor an adaptive, exploratory approach.
  - Embrace change in an economically sensible way.
  - Balance predictive up-front work with adaptive just-in-time work

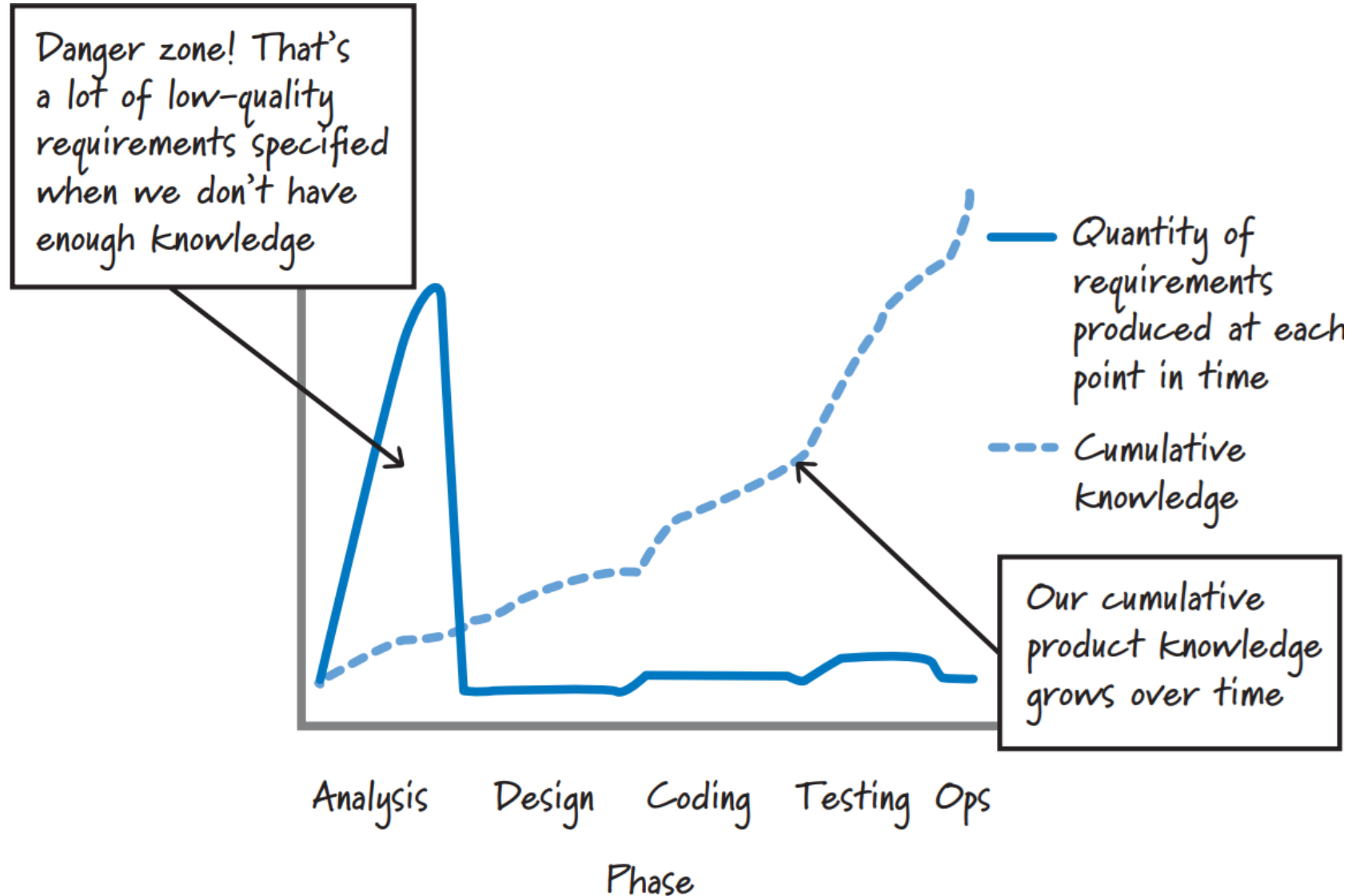
## 2.3.1 Keep Options Open

Plan-driven development	Agile Development
<ul style="list-style-type: none"><li>requires that important decisions in areas like requirements or design be made, reviewed, and approved within their respective phases</li></ul>  <p>The graph illustrates the trade-off between the cost of making a decision early versus the cost of deferring it. The 'Cost of deciding' (solid line) decreases as time progresses, while the 'Cost of deferring' (dashed line) increases. The intersection of these two curves marks the 'Last (economically) responsible moment', after which the cost of deferring becomes significantly higher than the cost of deciding.</p>	<ul style="list-style-type: none"><li>But Scrum favours a strategy of keeping options open<ul style="list-style-type: none"><li>Referred to as the last responsible moment (LRM) - we delay commitment and do not make important and irreversible decisions until the last responsible moment</li><li>When dealing with important or irreversible decisions, if we decide too early and are wrong</li></ul></li></ul>

## 2.3.2 Accept That You Can't Get It Right Up Front

Plan-driven development	Agile Development
<ul style="list-style-type: none"><li>• Mandates full requirements and a complete plan; they also assume that we can “get it right” up front.</li><li>• The reality is that it is very unlikely that we can get all of the requirements, or the detailed plans based on those requirements, correct up front.</li><li>• What's worse is that when the requirements do change, we have to modify the baseline requirements and plans to match the current reality</li></ul>	<ul style="list-style-type: none"><li>• Acknowledge that we can't get all of the requirements or the plans right up front</li><li>• With Scrum, we still produce some requirements and plans up front, but just sufficiently, and with the assumption that we will fill in the details of those requirements and plans as we learn more about the product</li></ul> <div data-bbox="838 811 1468 1233"></div> <div data-bbox="1518 859 1765 1188">Image on next slide</div>

## 2.3.2 Accept That You Can't Get It Right Up Front



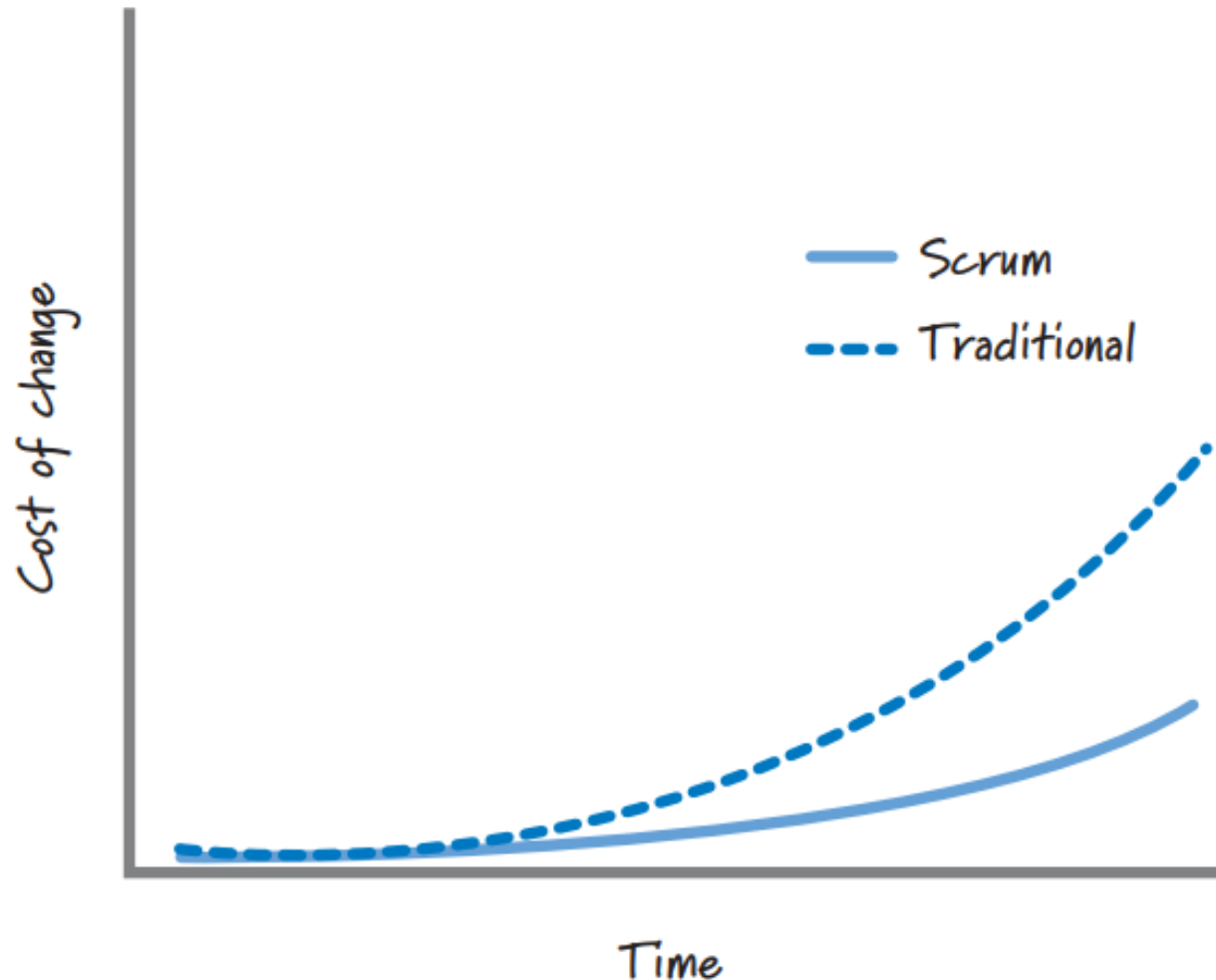
## 2.3.3 Favor an Adaptive, Exploratory Approach

Plan-driven development	Agile Development
<ul style="list-style-type: none"><li>• Focuses on using (or exploiting) what is currently known and predicting what isn't known.</li></ul>	<ul style="list-style-type: none"><li>• Scrum favours a more adaptive, trial-and error approach based on appropriate use of exploration<ul style="list-style-type: none"><li>• Exploration refers to times when we choose to gain knowledge by doing some activity, such as building a prototype, creating a proof of concept, performing a study, or conducting an experiment. i.e. when faced with uncertainty, we buy information by exploring.</li><li>• Feedback from our action will help us determine if and when we need further exploration.</li></ul></li></ul>

## 2.3.4 Embrace change in an economically sensible way

- we assume that change is the norm.
- Goal is to keep the cost-of-change curve flat (Fig 1) for as long as possible making it economically sensible to embrace even late change
- Can achieve that goal by managing the amount of work in process and the
- Flow of that work so that the cost of change when using Scrum is less affected by time than it is with sequential projects

## 2.3.4 Embrace change in an economically sensible way

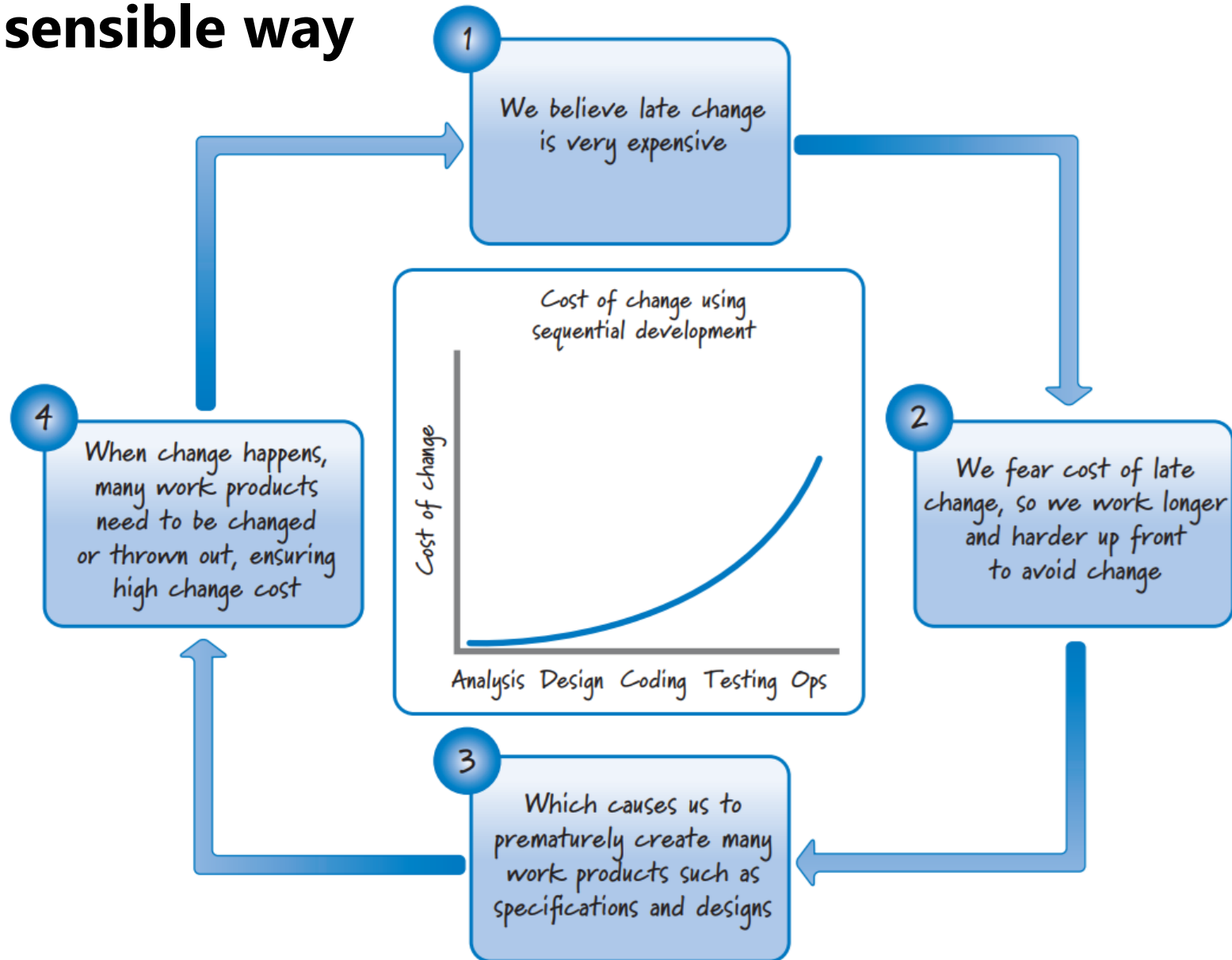


## 2.3.4 Embrace change in an economically sensible way

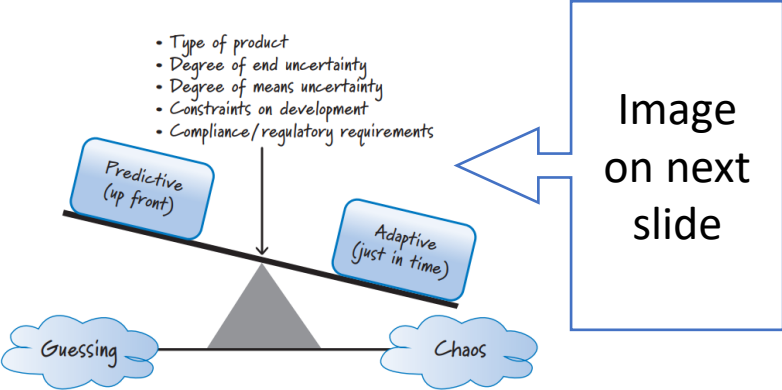
Plan-driven development	Agile Development
<ul style="list-style-type: none"><li>• As we have all learned, is substantially more expensive late than it is early on</li><li>• Seek to carefully control and minimize any changing requirements or designs by improving the accuracy of the predictions about what the system needs to do or how it is supposed to do it.</li></ul>	<ul style="list-style-type: none"><li>• We produce many work products in a just-in-time fashion, avoiding the creation of potentially unnecessary artifacts.</li><li>• When developing with Scrum, there does come a time when the cost of change will no longer be proportional to the size of the request, but this point in time the inflection point on the Scrum curve occurs later</li></ul>



## 2.3.4 Embrace change in an economically sensible way

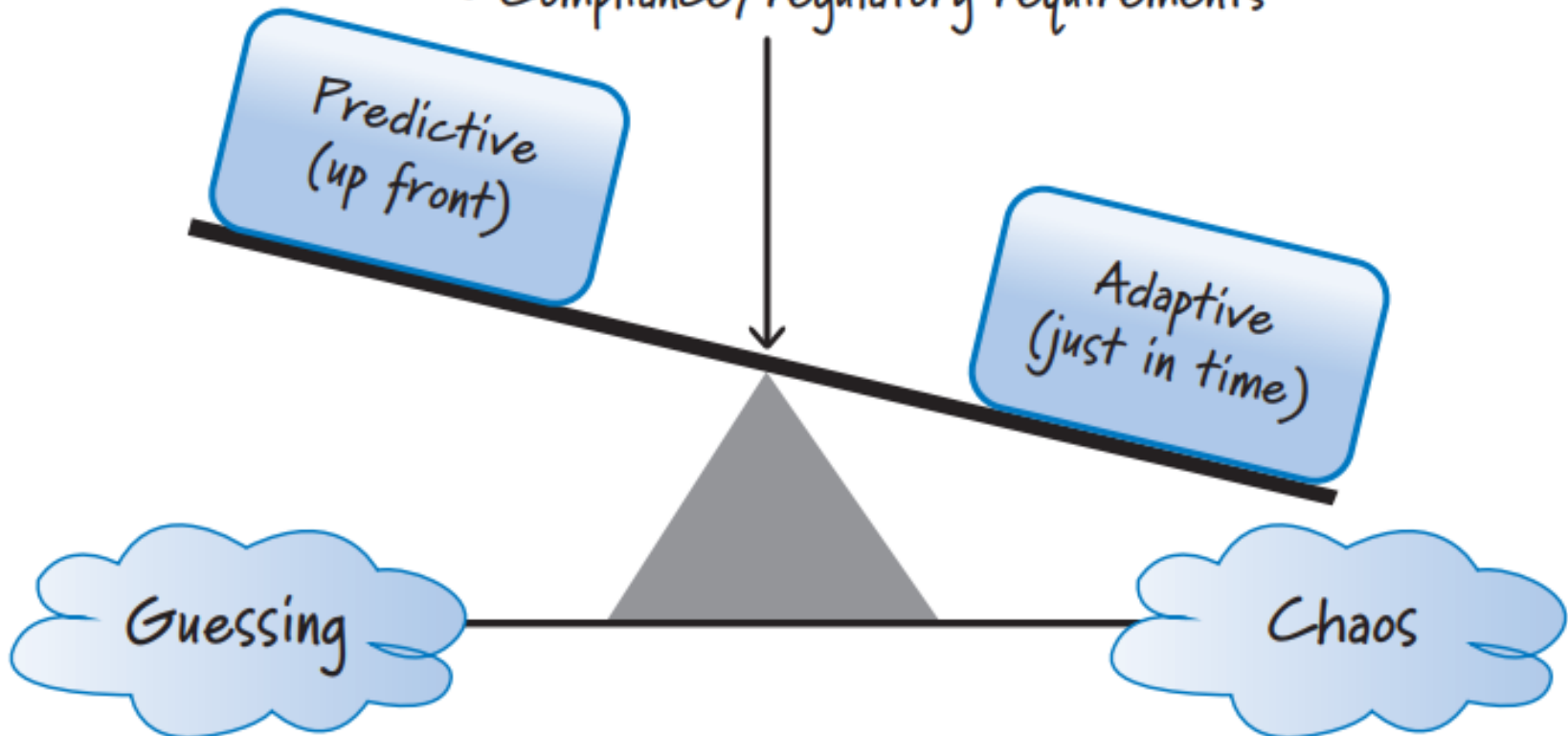


## 2.3.5 Balance predictive up-front work with adaptive just-in-time work

Plan-driven development	Agile Development
<ul style="list-style-type: none"><li>Detailed up-front requirements and planning are critical and should be completed before moving on to later stages.</li></ul> 	<ul style="list-style-type: none"><li>Up-front work should be helpful without being excessive.</li><li>Scrum, acknowledges that it is not possible to get requirements and plans precisely right up front.</li><li>Scrum is about finding balance between predictive up-front work and adaptive just-in-time work</li></ul>

## 2.3.5 Balance predictive up-front work with adaptive just-in-time work

- Type of product
- Degree of end uncertainty
- Degree of means uncertainty
- Constraints on development
- Compliance/regulatory requirements



## 2.3.5 Balance predictive up-front work with adaptive just-in-time work

- The balance point should be set in an economically sensible way to maximize the amount of ongoing adaptation based on fast feedback and minimize the amount of up-front prediction, while still meeting compliance, regulatory, and/or corporate objectives.
  - Balance is achieved is driven in part by the type of product being built, the degree of uncertainty that exists in both what we want to build and how we want to build it and the constraints placed on the development.

## 2.3.5 Balance predictive up-front work with adaptive just-in-time work

- Being overly predictive would require us to make many assumptions in the presence of great uncertainty. Being overly adaptive could cause us to live in a state of constant change, making our work feel inefficient and chaotic.
- To rapidly develop innovative products we need to operate in a space where adaptability is counterbalanced by just enough prediction to keep us from sliding into chaos.
  - ❑ The Scrum framework operates well at this balance point of order and chaos.

## 2.4 Validated learning

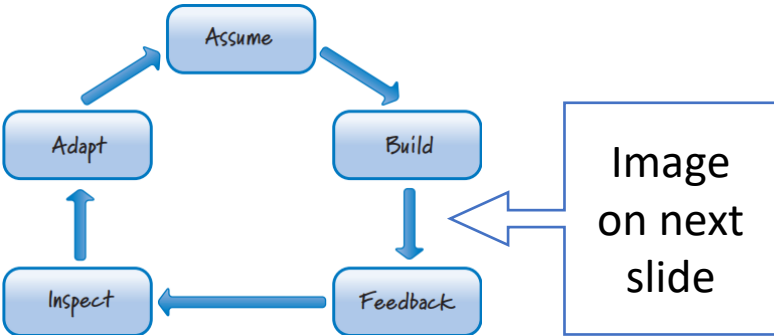
- Can acquire validated learning when we obtain knowledge that confirms or refutes an assumption that we have made.
  - Three agile principles related to this topic:
    - ❑ Validate important assumptions fast.
    - ❑ Leverage multiple concurrent learning loops.
    - ❑ Organize workflow for fast feedback.

## 2.4.1 Validate important assumptions fast

- Assumptions represent a significant development risk.

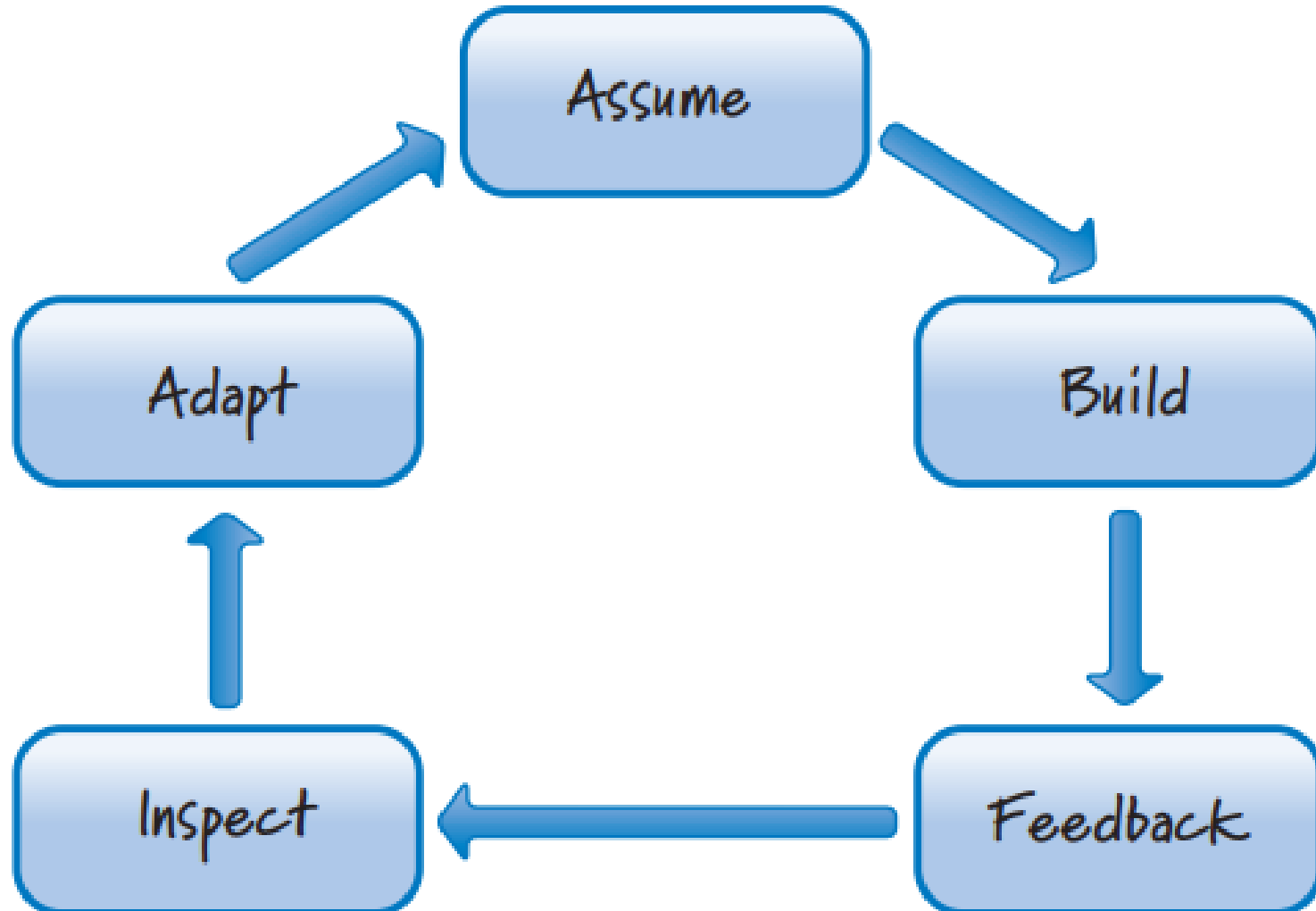
Plan-driven development	Agile Development
<ul style="list-style-type: none"><li>• Much more tolerant of long-lived assumptions than Scrum. Using plan driven development, we produce extensive up-front requirements and plans that likely embed many important assumptions, ones that won't be validated until a much later phase of development.</li></ul>	<ul style="list-style-type: none"><li>• Try to minimize the number of important assumptions that exist at any time.</li><li>• Don't want to let important assumptions exist without validation for very long</li><li>• As a result, if we make a fundamentally bad assumption when using Scrum, we will likely discover our mistake quickly and have a chance to recover from it.</li></ul>

## 2.4.2 Leverage multiple concurrent learning loops

Plan-driven development	Agile Development
<ul style="list-style-type: none"><li>• There is learning that occurs when using sequential development. However, an important form of learning happens only after features have been built, integrated, and tested, which means considerable learning occurs toward the end of the effort.</li><li>• Late learning provides reduce benefits because there may be insufficient time to leverage the learning or the cost to leverage it might be too high</li></ul>  <pre>graph TD; Assume[Assume] --&gt; Build[Build]; Build --&gt; Feedback[Feedback]; Feedback --&gt; Inspect[Inspect]; Inspect --&gt; Adapt[Adapt]; Adapt --&gt; Assume;</pre>	<ul style="list-style-type: none"><li>• Scrum identifies and exploits feedback loops to increase learning</li><li>• Recurring pattern in this style of product development is to make an assumption (or set a goal),<ul style="list-style-type: none"><li>• Build something (perform some activities),</li><li>• Get feedback on what we built,</li><li>• Use that feedback to inspect what we did relative to what we assumed.</li></ul></li><li>• Make adaptations to the product, process, and/or our beliefs based on what we learned</li><li>• Ex: pair programming (feedback in seconds)</li><li>• Test-driven development (feedback in minutes)</li></ul>



## 2.4.2 Leverage multiple concurrent learning loops



## 2.4.3 Organize workflow for fast feedback

### Plan-driven development

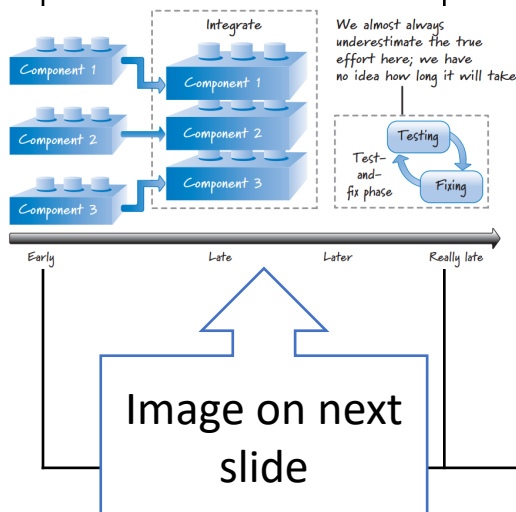
- tolerant of late learning, so fast feedback is not a focus.

### Agile Development

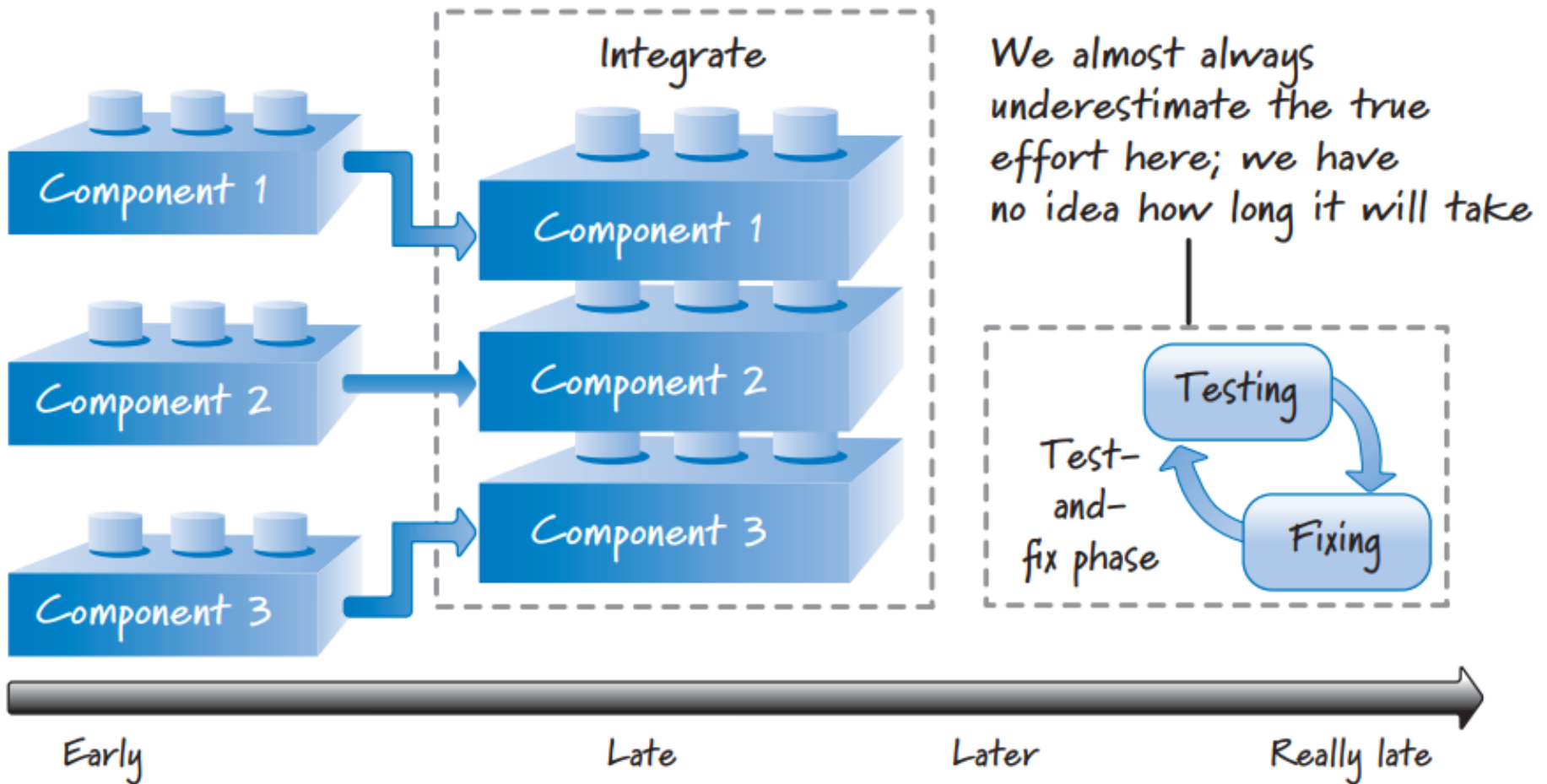
- Organize the flow of work to move through the learning loop in and get to feedback as quickly as possible. In doing so, we ensure that feedback-generating activities occur in close time proximity to the original work.

- Ex: component integration and testing

- At some time these components have to be integrated and tested before making a shippable product. Until we try to do the integration, we really don't know whether we have developed the components correctly. Attempting the integration will provide critical feedback on the component development work.



## 2.4.3 Organize workflow for fast feedback



## 2.5 WORK IN PROCESS (WIP)

- Work that has been started but not yet finished.
- During product development WIP must be recognized and properly managed.
- Agile principles related to this topic:
  - Use economically sensible batch sizes.
  - Recognize inventory and manage it for good flow.
  - Focus on idle work, not idle workers.
  - Consider cost of delay

## 3.5.1 Use economically sensible batch sizes

- Plan-Driven Development  
it is preferable to batch up all of one type of work and perform it in a single phase.

## 3.5.1 Use economically sensible batch sizes

Benefit	Description
Reduced cycle time	Smaller batches yield smaller amounts of work waiting to be processed, which in turn means less time waiting for the work to get done. So, we get things done faster.
Reduced flow variability	Think of a restaurant where small parties come and go (they flow nicely through the restaurant). Now imagine a large tour bus (large batch) unloading and the effect that it has on the flow in the restaurant.
Accelerated feedback	Small batches accelerate fast feedback, making the consequences of a mistake smaller.
Reduced risk	Small batches represent less inventory that is subject to change. Smaller batches are also less likely to fail (there is a greater risk that a failure will occur with ten pieces of work than with five).

Refer to the Ref.1 pg.49

## 3.5.1 Use economically sensible batch sizes

Benefit	Description
Reduced overhead	There is overhead in managing large batches—for example, maintaining a list of 3,000 work items requires more effort than a list of 30.
Increased motivation and urgency	Small batches provide focus and a sense of responsibility. It is much easier to understand the effect of delays and failure when dealing with small versus large batches.
Reduced cost and schedule growth	When we're wrong on big batches, we are wrong in a big way with respect to cost and schedule. When we do things on a small scale, we won't be wrong by much.

## 2.5.2 Recognize Inventory and Manage It for Good Flow

- Keep some inventory on hand but use a healthy dose of just-in-time inventory management
- Requirements are just one form of inventory that exists in product development exists in product development
- Too many requirements, we will likely experience inventory waste when requirements change



## 2.5.3 Focus on Idle Work, Not Idle Workers

- Idle work : we are blocked waiting on another team to do something
- Idle workers : people who have available capacity to do more work because they are not currently 100% utilized
- In plan-driven development : focus more on eliminating the waste
- Of idle workers than on the waste of idle work.
- The idle work (delayed work) grows exponentially once we get into the high levels of utilization.
- Acutely aware that finding the bottlenecks in the flow of work and focusing our efforts on eliminating them is a far more economically sensible activity than trying to keep everyone 100% busy

## 2.5.4 Consider Cost of Delay

- Financial cost associated with delaying work or delaying achievement of a milestone

## 2.5 Progress

- Measure progress by what we have delivered and validated, not by how we are proceeding according to the predefined plan or how far we are into a particular phase or stage of development.
- Three agile principles;
  - Adapt to real-time information and replan.
  - Measure progress by validating working assets.
  - Focus on value-centric delivery.

## 2.6.1 Adapt to Real-Time Information and Replan

Plan-driven development	Agile Development
<ul style="list-style-type: none"><li>• The plan is the authoritative source on how and when work should occur. As such, conformance to the plan is Expected</li></ul>	<ul style="list-style-type: none"><li>• Goal is to rapidly replan and adapt to the stream of economically important information that is continuously arriving during the development effort</li></ul>

## 2.6.2 Measure progress by validating working assets

Plan-driven development	Agile Development
<ul style="list-style-type: none"><li>• Effort is demonstrated by completing a phase and being permitted to enter the next phase. As a result, if each phase starts and completes as expected, the product development effort might seem to be progressing quite well.</li><li>• In the end, the product we created in full accordance with the plan might deliver far less customer value than anticipated.</li></ul>	<ul style="list-style-type: none"><li>• Measure progress by building working, validated assets that deliver value and that can be used to validate important assumptions.</li><li>• Gives us the feedback to know what the right next step is.</li><li>• It's not about how much work we start; it's all about what customer-valuable work we finish</li></ul>

## 2.6.3 Focus on value-centric delivery

### Plan-driven development

- Focuses on diligently following the process. By its very structure, the integration and delivery of features during sequential development happen at the end of the effort
- With this approach there is a risk that we will run out of resources (time or money) before we deliver all of the important value to our customers.

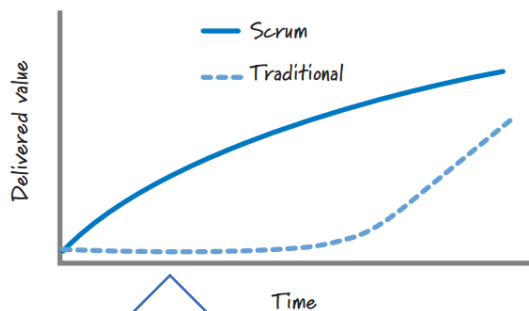
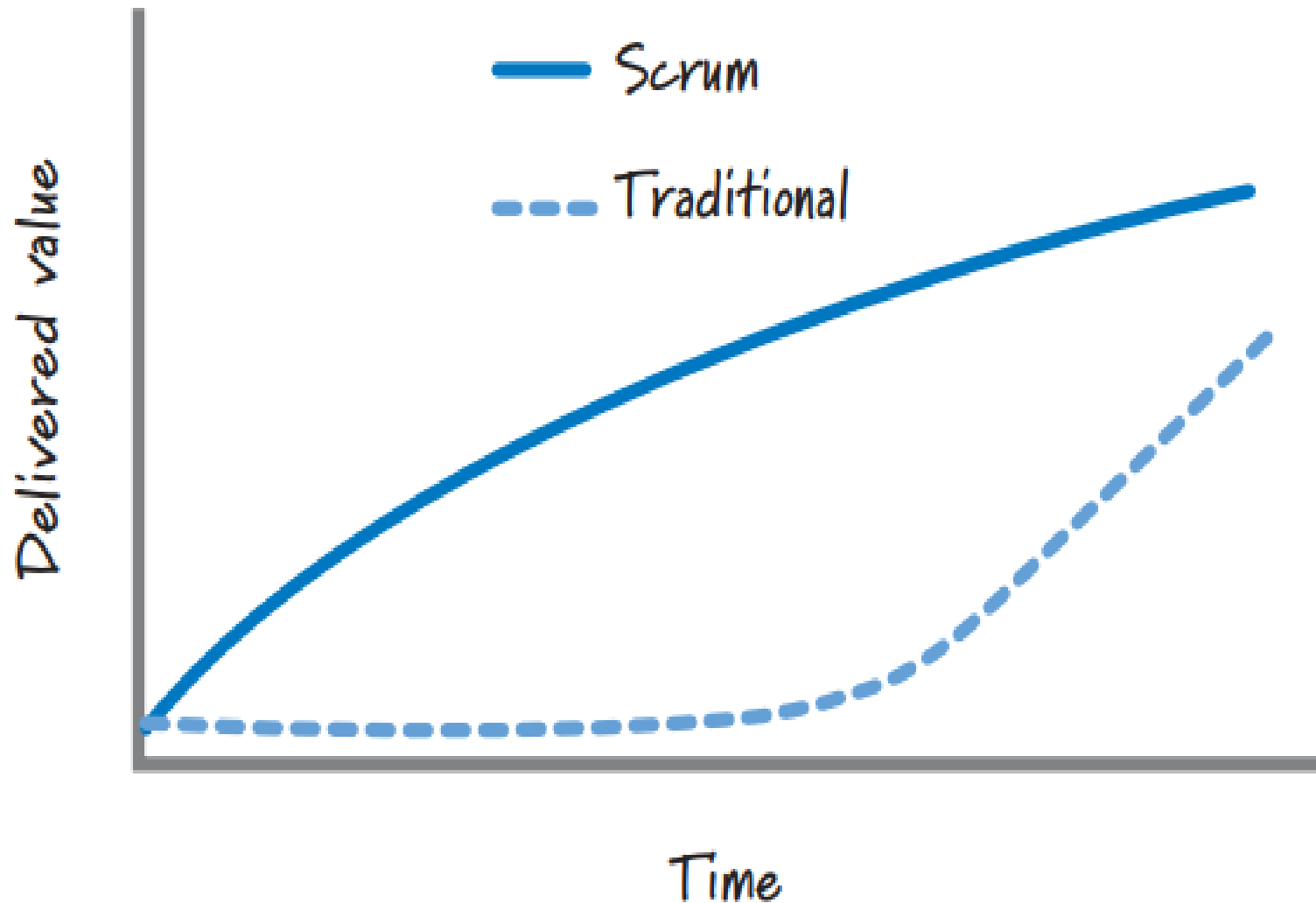


Image on next  
slide

### Agile Development

- A customer-value-centric form of development.
- Based on a prioritized, incremental model of delivery in which the highest-value features are continuously built and delivered in the next iteration. As a result, customers get a continuous flow of high-value features sooner.
- Value is generated by delivering working assets to customers, by validating important assumptions, or by acquiring valuable knowledge
- The intermediate artifacts provide no perceived customer value and are merely a means to an end if they themselves cannot be used to generate important feedback or acquire important knowledge

## 2.6.3 Focus on value-centric delivery



## 2.7 Performance

- Agile Principles related to this
  - Go fast but never hurry.
  - Build in quality.
  - Employ minimally sufficient ceremony.



## 2.7.1 Go fast but never hurry

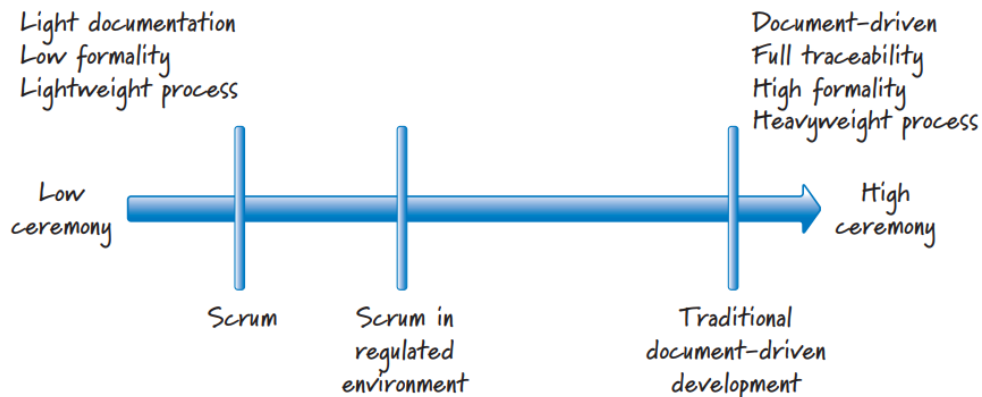
Plan-driven development	Agile Development
<ul style="list-style-type: none"><li>• Plan-driven development believes that if we follow the plan and do things right the first time, we'll avoid costly and time-consuming rework.</li><li>• Moving from step to step quickly is of course desirable, but it isn't a principal goal.</li></ul>	<ul style="list-style-type: none"><li>• One core goal is to be nimble, adaptable, and speedy. By going fast, we deliver fast, we get feedback fast, and we get value into the hands of our customers sooner. Learning and reacting quickly allow us to generate revenue and/or reduce costs sooner.</li></ul>

## 2.7.2 Build in quality

Plan-driven development	Agile Development
<ul style="list-style-type: none"><li>• The belief is that through careful, sequential performance of work we get a high-quality product.</li><li>• We can't actually verify this quality until we do the testing of the integrated product, which occurs during a late phase of the process.</li><li>• If testing should indicate that the quality is lacking, we then must enter the costly test-and-fix phase in an attempt to test quality in. Also, because a different team frequently works on each phase, the testing team is often viewed as owning the quality of the result</li></ul>	<ul style="list-style-type: none"><li>• Quality isn't something a testing team "tests in" at the end; it is something that a cross-functional Scrum team owns and continuously builds in and verifies every sprint</li><li>• Each increment of value that is created is completed to a high level of confidence and has the potential to be put into production or shipped to customers</li><li>• The need for any significant late testing to tack on quality is substantially reduced</li></ul>

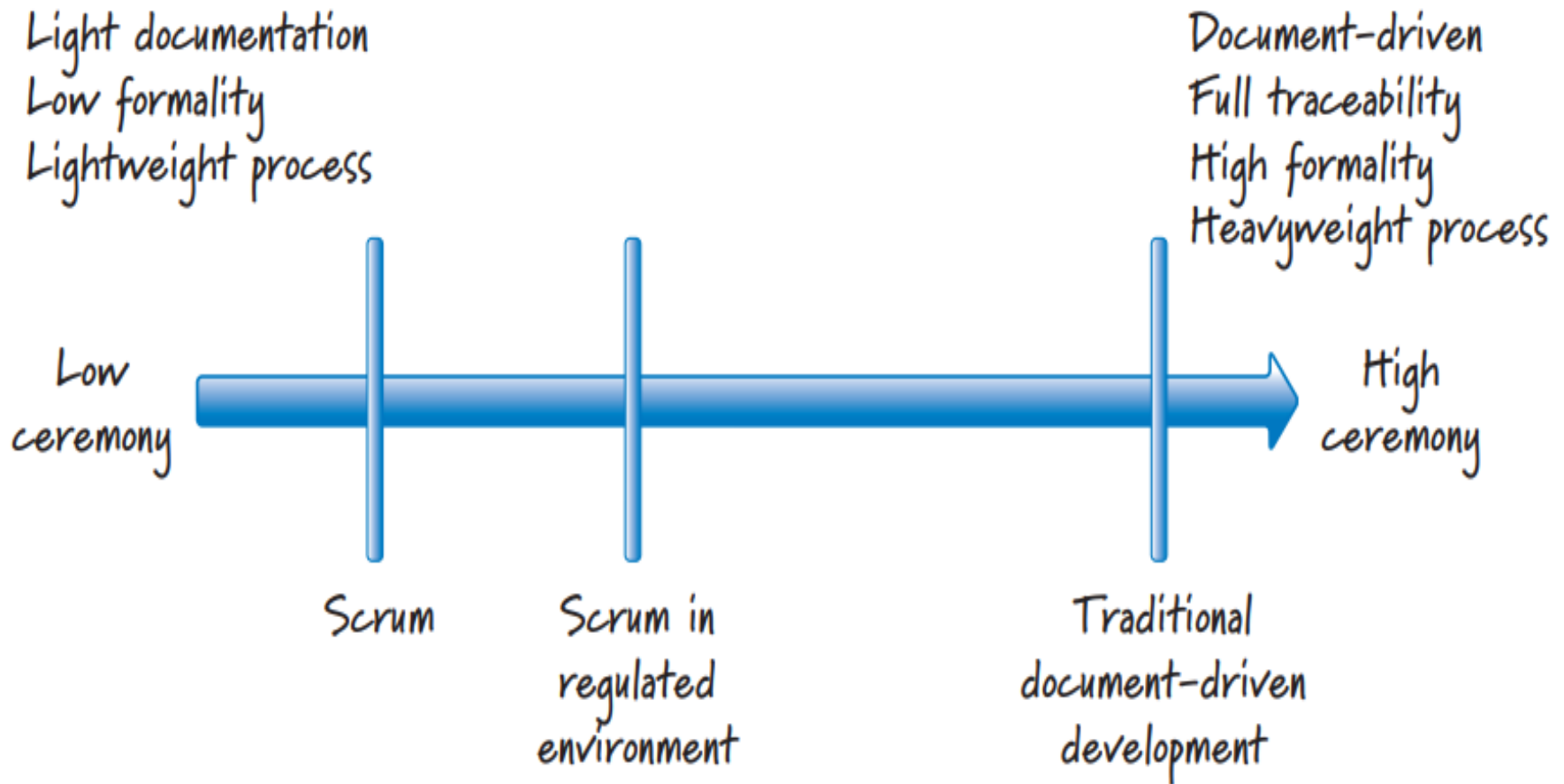
## 2.7.3 Employ minimally sufficient ceremony

Plan-driven development	Agile Development
<ul style="list-style-type: none"><li>Tend to be high-ceremony, document-centric, process-heavy approaches.</li></ul>	<ul style="list-style-type: none"><li>Goal is to eliminate unnecessary formality. Therefore, we set the ceremonial bar at a low level, one that is minimally sufficient or good enough.</li><li>No dead documents</li></ul>



Ceremony  
scale on next  
slide

## 2.7.3 Employ minimally sufficient ceremony



# Summary

Topic	Plan-Driven Principle	Agile Principle
<b>Similarity between development and manufacturing</b>	Both follow a defined process.	Development isn't manufacturing; development creates the recipe for the product.
<b>Process structure</b>	Development is phase-based and sequential.	Development should be iterative and incremental.
<b>Degree of process and product variability</b>	Try to eliminate process and product variability.	Leverage variability through inspection, adaptation, and transparency.
<b>Uncertainty management</b>	Eliminate end uncertainty first, and then means uncertainty.	Reduce uncertainties simultaneously.
<b>Decision making</b>	Make each decision in its proper phase.	Keep options open.
<b>Getting it right the first time</b>	Assumes we have all of the correct information up front to create the requirements and plans.	We can't get it right up front.

Topic	Plan-Driven Principle	Agile Principle
<b>Exploration versus exploitation</b>	Exploit what is currently known and predict what isn't known.	Favor an adaptive, exploratory approach.
<b>Change/emergence</b>	Change is disruptive to plans and expensive, so it should be avoided.	Embrace change in an economically sensible way.
<b>Predictive versus adaptive</b>	The process is highly predictive.	Balance predictive up-front work with adaptive just-in-time work.
<b>Assumptions (unvalidated knowledge)</b>	The process is tolerant of long-lived assumptions.	Validate important assumptions fast.
<b>Feedback</b>	Critical learning occurs on one major analyze-design-code-test loop.	Leverage multiple concurrent learning loops.
<b>Fast feedback</b>	The process is tolerant of late learning.	Organize workflow for fast feedback.

Topic	Plan-Driven Principle	Agile Principle
<b>Batch size (how much work is completed before the next activity can start)</b>	Batches are large, frequently 100%—all before any. Economies of scale should apply.	Use smaller, economically sensible batch sizes.
<b>Inventory/work in process (WIP)</b>	Inventory isn't part of the belief system so is not a focus.	Recognize inventory and manage it to achieve good flow.
<b>People versus work waste</b>	Allocate people to achieve high levels of utilization.	Focus on idle work, not idle workers.
<b>Cost of delay</b>	Cost of delay is rarely considered.	Always consider cost of delay.
<b>Conformance to plan</b>	Conformance is considered a primary means of achieving a good result.	Adapt and replan rather than conform to a plan.

Topic	Plan-Driven Principle	Agile Principle
<b>Progress</b>	Demonstrate progress by progressing through stages or phases.	Measure progress by validating working assets.
<b>Centricity</b>	Process-centric—follow the process.	Value-centric—deliver the value.
<b>Speed</b>	Follow the process; do things right the first time and go fast.	Go fast but never hurry.
<b>When we get high quality</b>	Quality comes at the end, after an extensive test-and-fix phase.	Build quality in from the beginning.
<b>Formality (ceremony)</b>	Formality (well-defined procedures and checkpoints) is important to effective execution.	Employ minimally sufficient ceremony.