Advanced Software Engineering

Dr. Elham Mahmoudzadeh

Isfahan University of Technology

mahmoudzadeh@iut.ac.ir

2022

Agile Principles(IV)

Dr. Elham Mahmoudzadeh

Isfahan University of Technology

mahmoudzadeh@iut.ac.ir

2020

Validated Learning

- When we obtain knowledge that confirms or refutes an assumption that we have made.
- Three agile principles related to this topic.
  1. *Validate important assumptions fast .*
  2. *Leverage multiple concurrent learning loops.*
  3. *Organize workflow for fast feedback.*

What is assumption?

- An assumption is a guess, or belief, that is assumed to be true, real, or certain even though we have no validated learning to know that it is true.
- Plan-driven development is much more tolerant of long-lived assumptions than Scrum.
  - *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.*
- Represent a significant development risk.

Validate Important Assumptions Fast

- In Scrum, we try to minimize the number of important assumptions that exist at any time.
- Also don't let important assumptions exist without validation for very long.
  - *The combination of iterative and incremental development along with a focus on low-cost exploration can be used to validate assumptions fast.*
- 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.
- In plan-driven, sequential development, the same bad assumption if validated late might cause a substantial or total failure of the development effort.

Leverage Multiple Concurrent Learning Loops(I)

- 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.

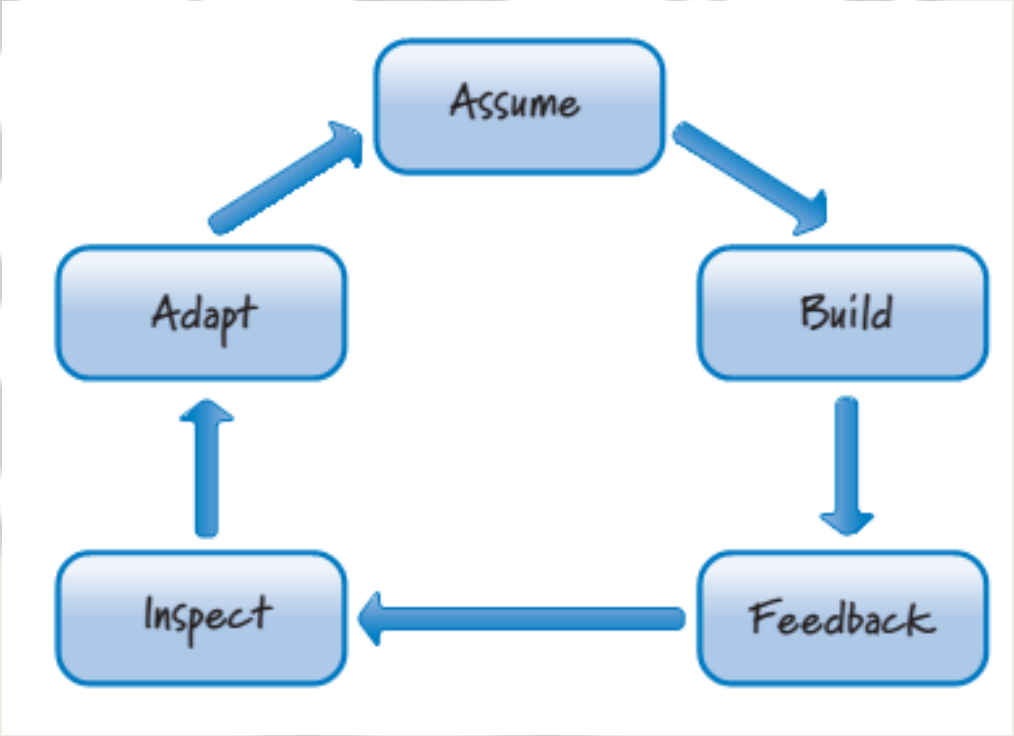
- Late learning provides reduced benefits because there may be insufficient time to leverage the learning or the cost to leverage it might be too high.

might be too high.

Leverage Multiple Concurrent Learning Loops(II)

- In Scrum, we understand that constant learning is a key to our success.
- When using Scrum, we identify and exploit feedback loops to increase learning.
- A recurring pattern in this style of product development is to
  - *make an assumption (or set a goal),*
  - *build something (perform some activities),*
  - *get feedback on what we built,*
  - *use that feedback to inspect what we did relative to what we assumed.*
  - *make adaptations to the product, process, and/or our beliefs based on what we learned.*

Leaning loop pattern



Leverage Multiple Concurrent Learning Loops(III)

- Scrum leverages several predefined learning loops.

- For example, the daily scrum is a daily loop and the sprint review is an iteration-level loop.

Leverage Multiple Concurrent Learning Loops(V)

- The Scrum framework is also flexible enough to embrace many other learning loops.

- For example, although not specified by Scrum, technical practice feedback loops, such as pair programming (feedback in seconds) and test-driven development (feedback in minutes), are frequently used with Scrum development.

Organize Workflow for Fast Feedback(I)

- Being tolerant of long-lived assumptions also makes plan-driven processes tolerant of late learning, so fast feedback is not a focus.
- With Scrum, we strive for fast feedback, because it is critical for helping truncate wrong paths sooner and is vital for quickly uncovering and exploiting time-sensitive, emergent opportunities.

Organize Workflow for Fast Feedback(II)

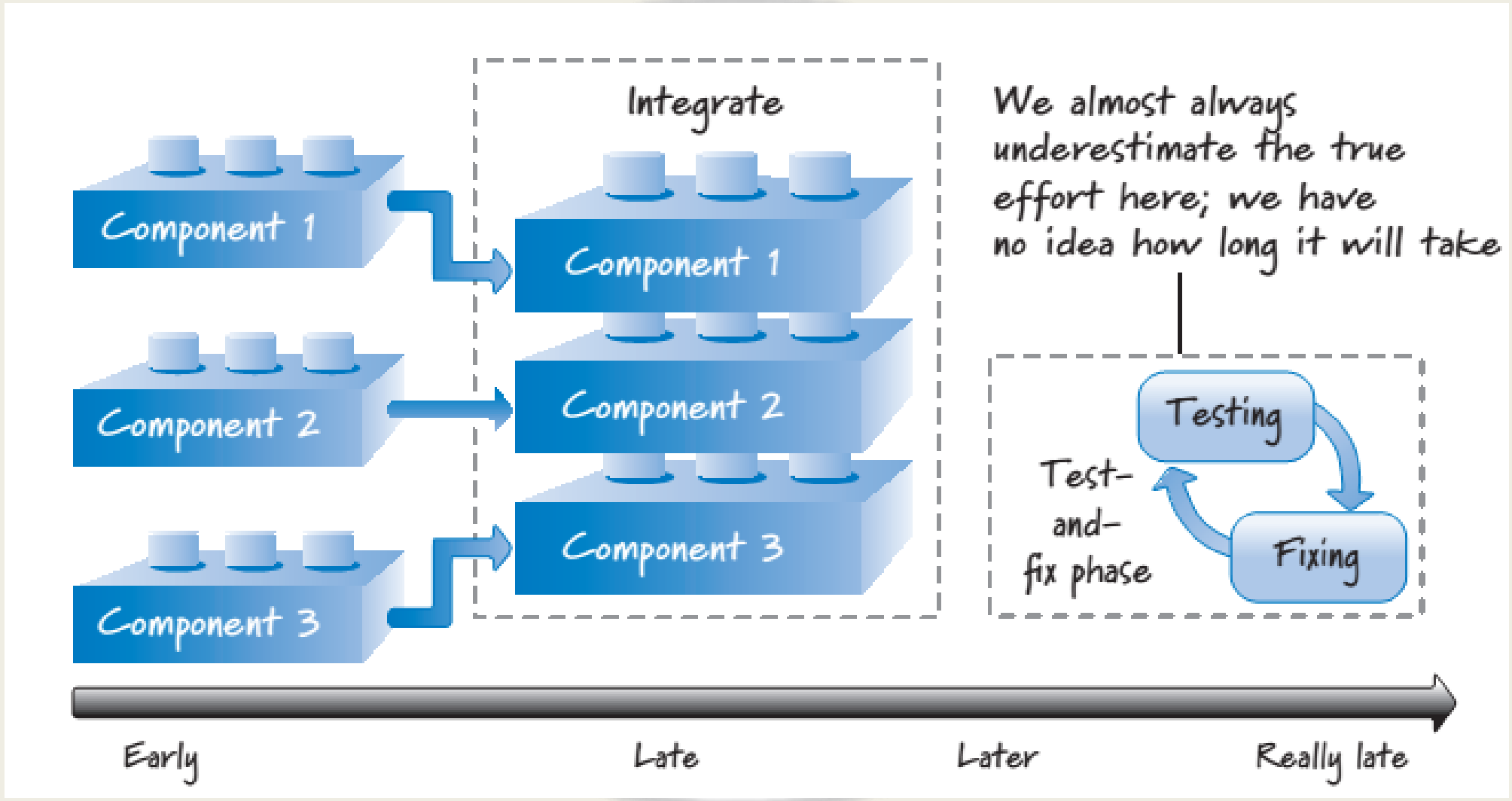
- In a plan-driven development effort, every activity is planned to occur at an appointed time based on the well-defined phase sequence.
- This approach assumes that earlier activities can be completed without the feedback generated by later activities.
- As a result, there might be a long period of time between doing something and getting feedback on what we did (hence closing the learning loop).

Let's use component integration and testing as an example(I)

- We are developing three components in parallel. At some time these components have to be integrated and tested before we have 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.

Let’s use component integration and testing as an example(II)

- Using sequential development, integration and testing wouldn’t happen until the predetermined downstream phase, where many or all components would be integrated.
- Unfortunately, the idea that we can develop a bunch of components in parallel and then later, in an integration phase, smoothly bring them together into a cohesive whole is unlikely to work out.
- In fact, even with well-conceived interfaces defined before we develop the components, it’s likely that something will go wrong when we integrate them.


Integrate
Component 1
Component 2
Component 3
Component 1
Component 2
Component 3
We almost always underestimate the true effort here; we have no idea how long it will take
Testing
Test-and-fix phase
Fixing
Early
Late
Later
Really late

Let's use component integration and testing as an example(IV)

- Feedback-generating activities that occur a long time after development have unfortunate side effects, such as turning integration into a large test-and-fix phase, because components developed disjointedly from each other frequently don't integrate smoothly.
- How long it will take and how much it will cost to fix the problem can only be guessed at this point?

Organize Workflow for Fast Feedback(VII)

- In Scrum, we organize the flow of work to move through the learning loop 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.
- Fast feedback provides superior economic benefits because errors compound when we delay feedback, resulting in exponentially larger failures.

Let's look back at our component integration example

- When we designed the components, we made important assumptions about how they would integrate. Based on those assumptions, we proceeded down a design path. We do not, at this point, know whether the selected design path is right or wrong. It's just our best guess.

- Once we choose a path, however, we then make many other decisions that are based on that choice. The longer we wait to validate the original design assumption, the greater the number of dependent decisions.

Let’s look again at our component integration example

- If we later determine (via feedback during the integration phase) that the original assumption was wrong, we’ll have a large, compounded mess on our hands.
- Not only will we have many bad decisions that have to be reworked; we’ll also have to do it after a great deal of time has passed.
- Because people’s memories will have faded, they will spend time getting back up to speed on the work they did earlier.

Organize Workflow for Fast Feedback(XI)

- When we factor in the total cost of reworking potentially bad dependent decisions, and the cost of the delay to the product, the economic benefits of fast feedback are very compelling.

- Fast feedback closes the learning loop quickly, allowing us to truncate bad development paths before they can cause serious economic damage.

Reference

1- K. S. Rubin, “Essential Scrum, A Practical guide to the most popular agile process,” 2013.