Advanced Software Engineering

Dr. Elham Mahmoudzadeh

Isfahan University of Technology

mahmoudzadeh@iut.ac.ir

2022

Agile Principles(IV)

Dr. Elham Mahmoudzadeh

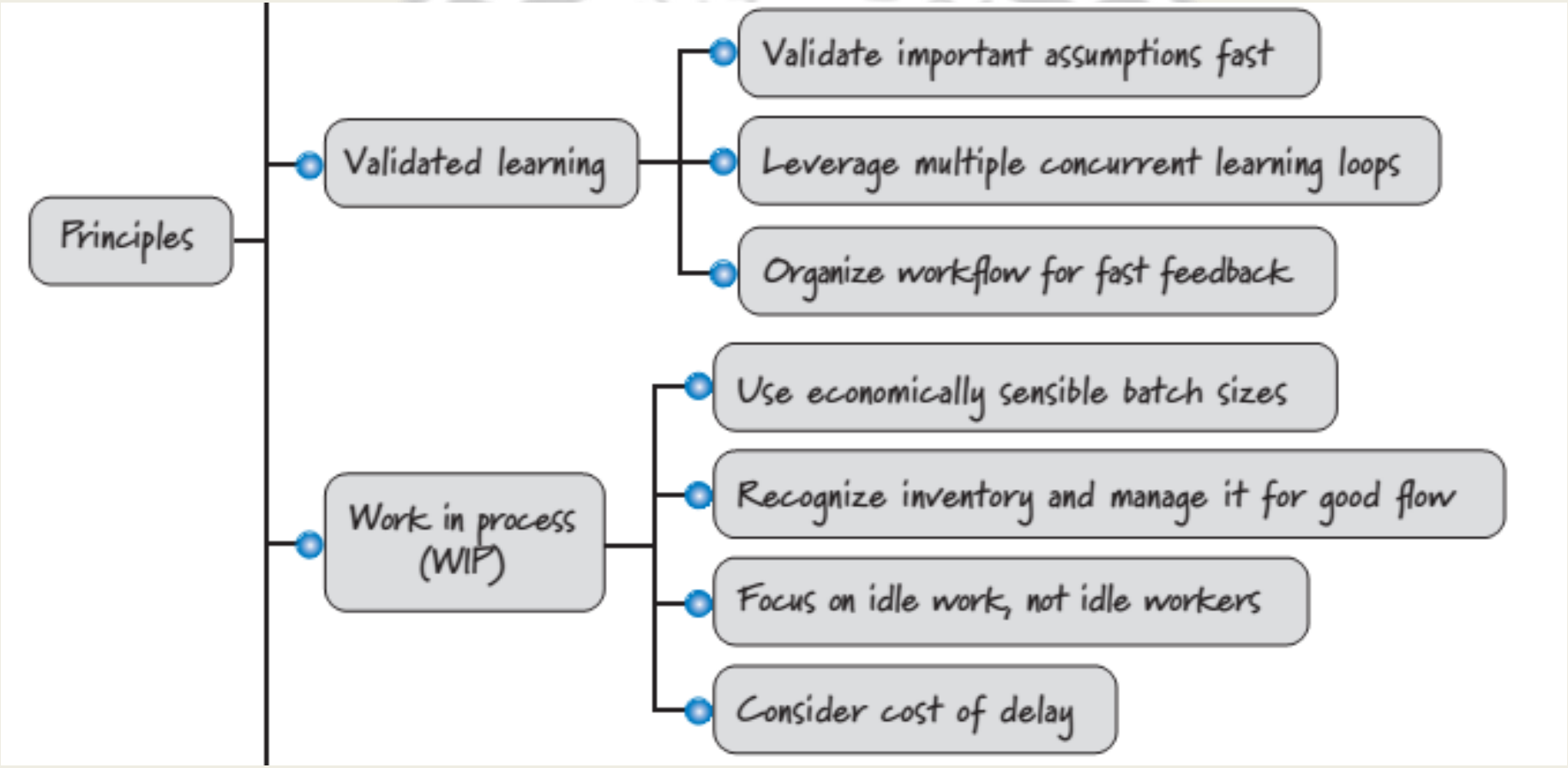
Isfahan University of Technology

mahmoudzadeh@iut.ac.ir

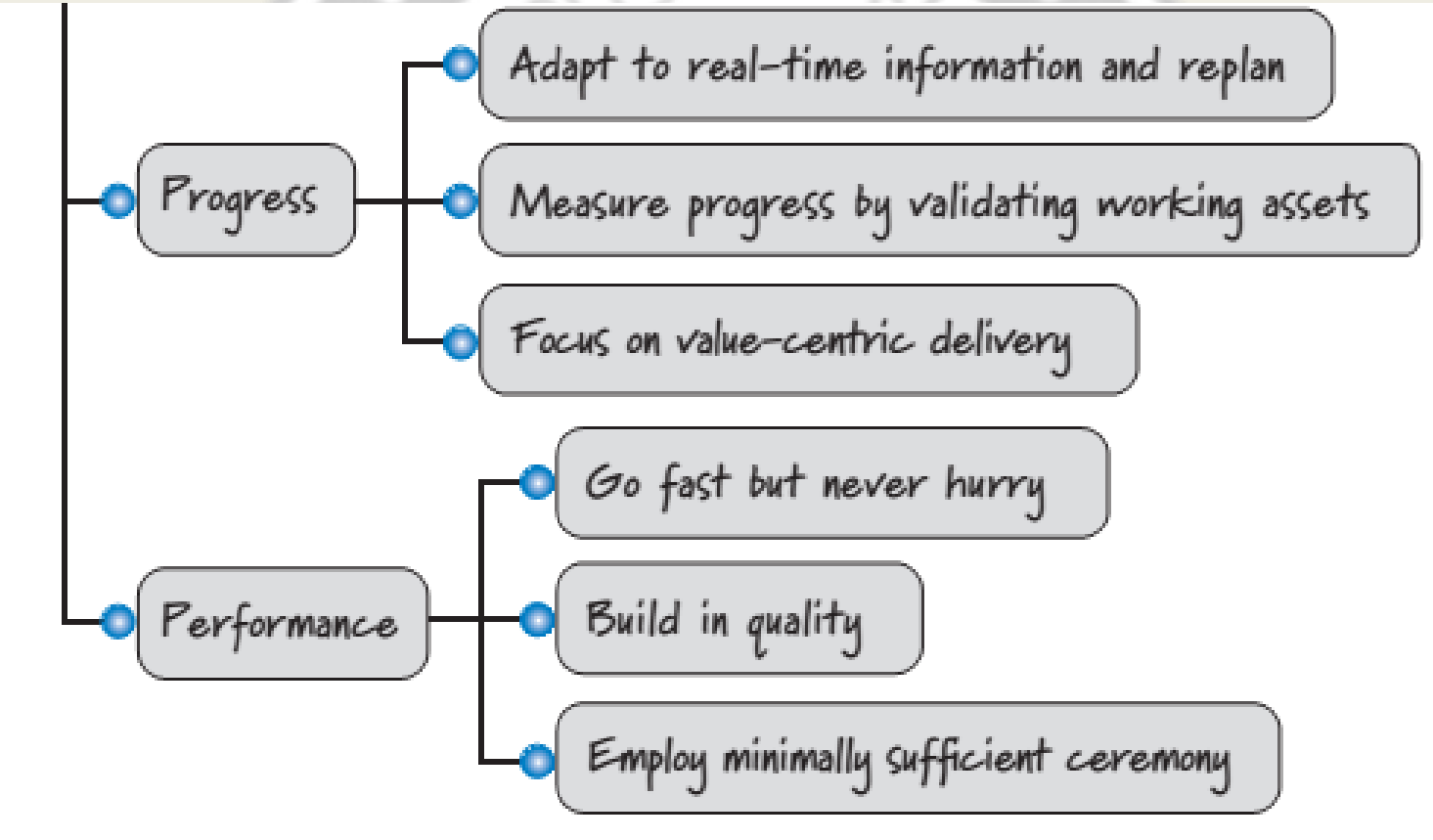
2020

Categorization of principles (Up)

Categorization of principles (Middle)



Categorization of principles (Bottom)



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.*

-
وقتی کی دانشی رو کسب میکنیم باید اون دانشه بررسی بشه و تایید و رد بشه
سه تا اصول داره:
1- ما مفروضات مهم رو زود بفهمیم که درستن یا نه
2- یادگیری حلقه ها و بستری که توی agile مطرح میشه --> حلقه های سریع و کوتاهی که به صورت همزمان هم ممکنه اتفاق بیوفت و این باعث میشه ما زودتر به اون ولیدیشن و غیرش برسیم
3- نحوی انجام کار رو تنظیم بکنیم که زود فیدبک سریع رو به دست بیاریم تا بتونیم رویکرد ادامه کار رو بچینیم

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.

-
ما مفروضات مهم رو باید زود بفهمیم که درستن یا نه
این فرضیات گمان هایی هستن ک از اول کار داریم و نمی دونیم درستن یا نه
توی متدلوژی های سنتی این بحث است که ما مقاوم باشیم در برابر تغییرات و تحمل پذیری خیلی نسبت به اون فرضیات tolerant است ینی فرض میکنیم یک فرضیات مهمی رو از اول کار داریم و متحد به اون فرضیات هستیم و خیلی اعتقادی به این نداریم که اون فرضیات باید عوض بشن و روی همون فرضیات بیس کار چیده میشه و می ره جلو در حالی که اعتبار سنجی اون فرضیات انتهای کار باشه که این کار رو سخت میکنه و در واقعیت این کار یک ریسکه که هر چقدر میزان فرضیات بدون اعتبارسنجی زیاد باشه ریسک بالا می بره ما توی اسکرام سعی میکنیم این فرضیات مهم رو مقدارش کم بکنیم و در حد ممکن زودتر اعتبارسنجیشون انجام بشه ینی تا جایی که میشه زود فیدبک رو از مشتری بگیریم

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.

-
پس فرضیات مهم و دانش مهمی که داریم زود فیدبک رو بگیریم و مطمئن بشیم که درسته و بعد گام های بعدی رو برداریم
در بدترین حالت هم اگر این فرضیات رو داریم و فیدبک رو نگیریم از مشتری نگیریم و یک sprint بریم جلو و اشتباه کرده باشیم زمان خیلی کمی رو از دست می دیم و هزینه کمتری داریم پرداخت میکنیم چون اخر اون sprint اینو می فهمیم که زود هم این اتفاق می افته و در بدترین حالت اینه که زمان یک sprint رو از دست دادیم ولی توی متدلوژی سنتی این اتفاق خیلی بده ینی اگر فرضیات ما اشتباه باشه بخاطر این که ما دیر فیدبک از مشتری میگیریم میزان هزینه خیلی زیاد میشه و ممکنه یکسری عواقب و نقص های جبران ناپذیری توی سیستم قرار بگیره

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.

-
اولین: ما اطلاعات مهم و فرضیات و دانش های مهمون زود از مشتری فیدبک بگیریم و مطمئن بشیم که درسته
2- یادگیری حلقه ها رو توی اصول Agile داریم --> میزان دانشی که از اون کار نرم افزار مشتری بهمون میده زیاده و اگر این یادگیری دیر اتفاق بیوفته مزایای فیدبک گرفتن از مشتری کمتر خواهد بود حالا اگر ما بستر رو جوری فراهم بکنیم که این یادگیری زود و مداوم اتفاق بیوفته و حتی ثابت باشه این می تونه یک فاکتور مهمی باشه توی این که ما هزینه های کمتری بدیم و هم زودتر با زمان کمتر و effort کمتری بتونیم به نظر مشتریمون نزدیک تر بشیم

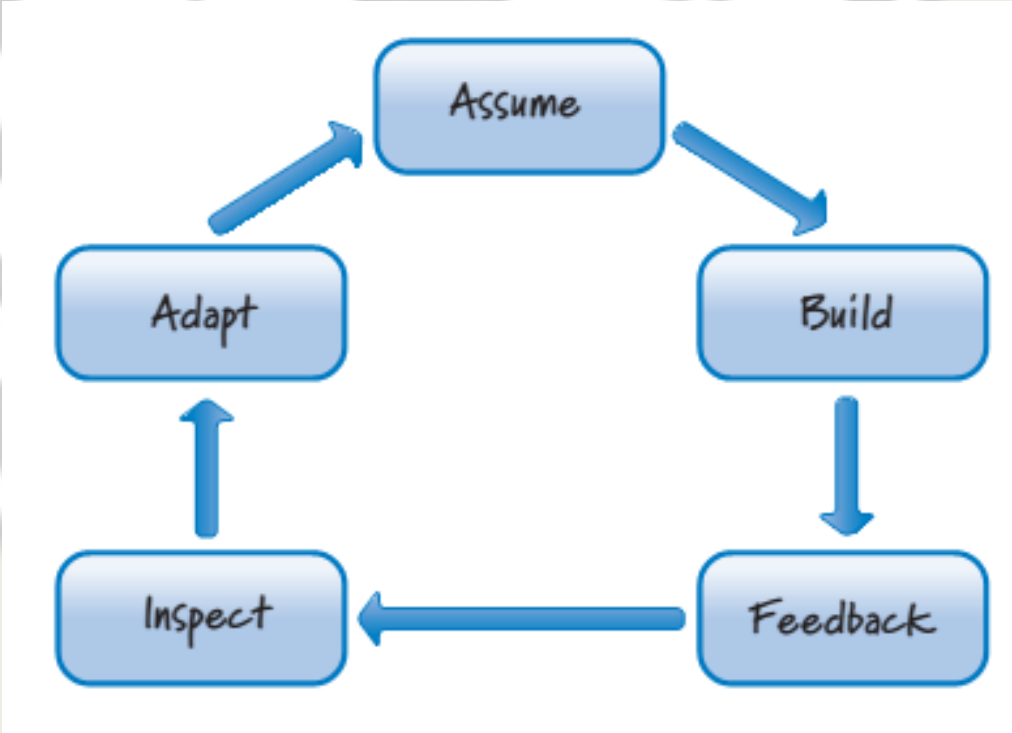
توی اسکرام این یادگیری حلقه ها سریعتر اتفاق میافته و اخر هر لوپی اون فیدبک رو از مشتری میگیریم ینی هم باعث میشه که سریع این فیدبکه رو بگیریم و هم باعث میشه که اون میزان دانشی که از محیط یا مشتری میگیریم زیاد باشه

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



-
این یادگیری لوپ است:
فرض میکنیم یک چیزی رو برمبنای اون فرضیات می سازیم و ارائه میدیم و فیدبک می گیریم و بازبینی های فنی یا inspect انجام میشه و adapt انجام میشه ینی این یادگیری لوپ حتی توی اسکرام روزانه هم اتفاق می افته که این باعث میشه میزان دانشی که می گیریم زیاد باشه و این کار زود اتفاق می افته
حتی اگر یه جای کار هم اشتباه باشه خیلی چیزی از دست نمی دیم اینجا چون فهمیدم زود اشتباهه و مسیر بعدی رو درست میکنیم

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.

-

این یادگیری لوپ --> لوپ های حدس و انجام و ارائه و گرفتن فیدبک و تطبیق این لوپ ها توی اسکرام معنا دار است و هم می تونیم زود بفهمیم مشتری چی می خواد

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.

-

علاوه بر اینکه یادگیری لوپ فرض کردن، ساختن، فیدبک گرفتن، بازبینی فنی و تطبیق اتفاق می افته یادگیری لوپ های دیگری هم وجود داره مثل TDD یا ..

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.

-
تنظیم بکنیم workflow ینی خود کاری که انجام میدیم و اون رویکرد انجام کار رو تنظیم بکنیم که فیدبک ها رو سریع بگیریم
توی پلن دریون اصلا کلا مطرح نیست ینی می گیم فرضیات اوکی است و فیدبک دیر اتفاق می افته ولی توی اسکرام این خیلی مهمه که فیدبک سریع وجود داشته باشه بخاطر اینکه اگر اون فیدبک سریع اتفاق می افته اگر خروجی اون فیدبکه بر این باشه که کاری که انجام شده اشتباه است ما زودتر اون نقاط اشتباه رو متوجه میشیم و باعث میشیم اون خطا توی کل سیستم توسعه پیدا نکنه و اون مسیرهای اشتباه زودتر بریده بشن و ادامه مسیر با یک اطمینان خاطر بیشتری جلو ببریم

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).

-

توی پلن دریونه این اتفاق نمی افته --> توی این فرض میکنیم همه چیز درست است و می ره جلو در صورتی که ماهیتا توی تولید نرم افزار این اتفاق نمی افته و بدیش این که فیدبک خیلی دیر اتفاق می افته ینی فاصله زمانی دریافت فیدبک نسبت به کاری که انجام شده است خیلی زیاد است و این خودش خیلی زیان اور است

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

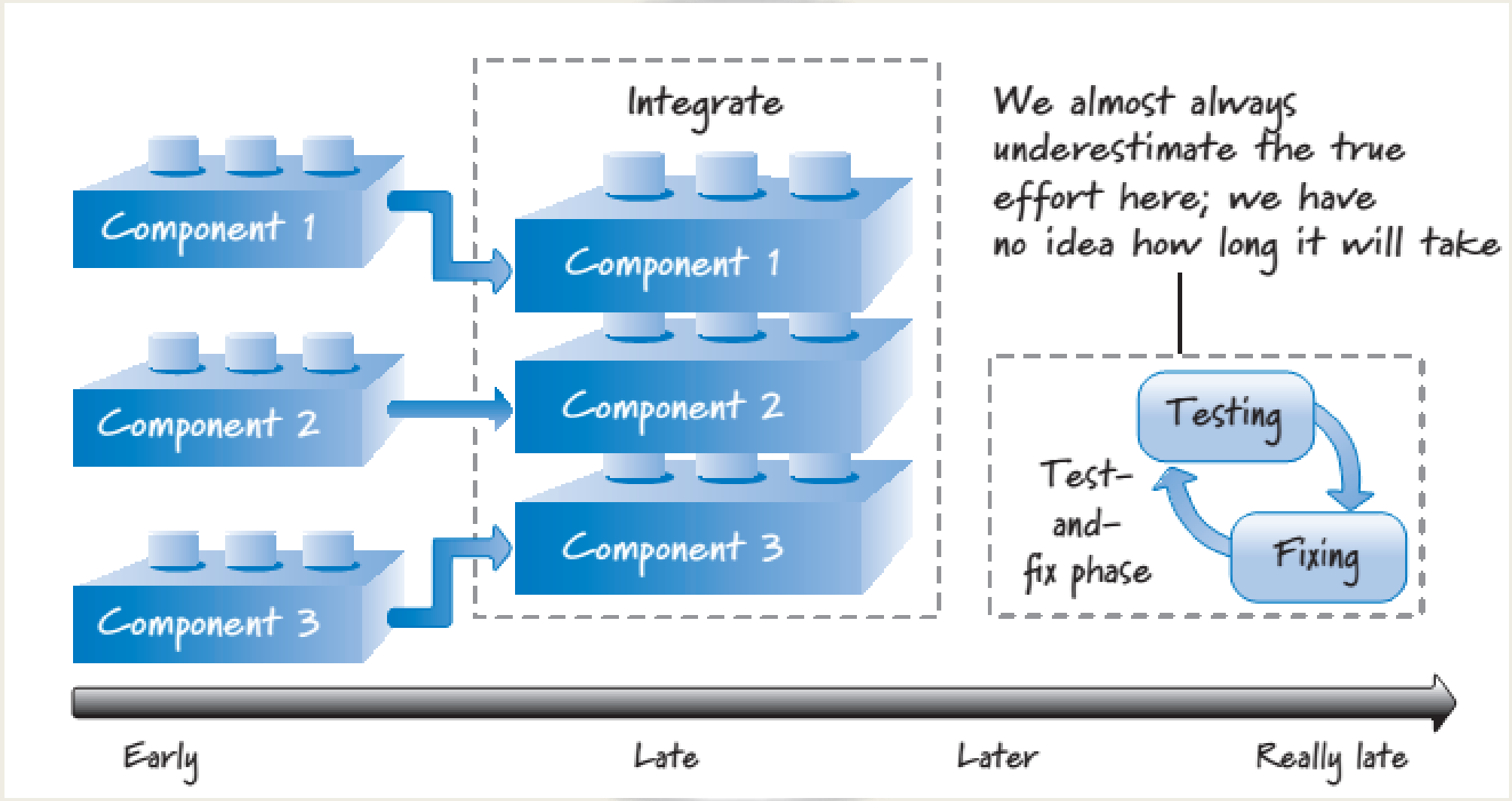
- We are developing three components in parallel. At some time these components have to be integrated and tested before we have a shippable product. Until we try to do the integration, we really don’t know whether we have developed the components correctly. Attempting the integration will provide critical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

-
مثال:
چندتا مولفه داریم به طور موازی بحث طراحی و توسعه اینا integrate میشه و بعد از یه زمانی اینا درست میشن و تست میشن و تحویل مشتری داده میشه
تا وقتی که ما توی فاز integration نریم نمی دونیم این مولفه ها دارن درست در ارتباط با همدیگه کار می کنن یا نه و اون فاز integration که فیدبک به ما میده یا integration test که به ما میگه این مولفه ها دارن در ارتباط با هم دیگه درست کار میکنن

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.

-
اگر بخوایم متدلوژی سنتی رو بریم: اول طراحی میکنیم و بعد دولوپ انجام بدیم کامل و بعد وارد فاز integration بشیم و بعد فاز تست
ولی واقعیت اینه که ما تا وقتی که به فاز integration نرسیدیم نمی تونیم بفهمیم بعضی از خطاها رو حتی اگر اینترفیس ها از اول کاملا مشخص باشه ولی پتانسیل این که ما یه جایی از کار رو اشتباه رفته باشیم زیاده


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

توی این مثال سه تا مولفه گرفته
اگر تعداد مولفه ها زیاد باشه به طور سرسام اوری توی بحث integration می تونیم مشکل داشته باشیم

توی پلن دریون ها به طور جداگانه این ها دولوپ میشن و فاز integration تقریبا انتهای کار اتفاق می افته بعد از integration ما تست رو داریم و هر خطایی که اتفاق می افته باید اون خطا فیکس بشه توی سیستم و مجددا تست اتفاق بیوفته
عملا test and fix خیلی دیره ینی ما یکسری زمانی داریم اینجا می ذاریم که قبلا توی پلن دیده نشده خیلی و شاید نمایی بالا بره بسته به اون خطاهایی که اتفاق می افته

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?

-
پس داریم فیدبک رو دیر میگیریم از محیطمون --> این فیدبک می تونه بد باشه ینی زمان دیر گرفتن فیدبک میتونه بد باشه و اگر این فیدبک رو زودتر می گرفتیم توی اون دولوپ کردن مولفه ها می تونستیم تایم و هزینه رو سیو بکنیم

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.

-

توی اسکرام این اتفاق زود می افته ینی ما اون طراحی ها و دانش های اولیه و مهم رو زود فیدبکش رو میگیریم و این کمک میکنه به ما که در ادامه کار میزان کارهای مجددمون رو کم بکنیم

ما یکسری فرضیات رو داریم و برمبنای اون فرضیاته integration قراره انجام بشه و مسیر طراحی برمبنای اون فرضیاته مشخص میشه و هرچی ما این فرضیات رو زودتر اعتبارسنجی بکنیم به ما کمک میکنه که مطمئن بشیم اون مسیر فرضیات درسته و زودتر می ریم جلو

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.

-
اگر ما این رو دیر بفهمیم خطاها گسترش پیدا میکنن و علاوه بر اینکه ما کار زیادتری باید انجام بدیم یک بخشی از زمان رو هم از دست دادیم
ضمن اینکه اگر اون فیدبک دیر به ما برسه ما ادم ها معمولا فراموش میکنیم اون ها رو و هرچی زودتر فرضیات ما در اینده نزدیک مورد بازبینی تایید یا رد قرار بگیرین ادامه کارمون رو منطقی تر می چینیم

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.

-

توی اسکرام این بستر فراهم است که بیایم زود این فیدبک رو بگیریم که این به ما کمک میکنه که هم هزینه کمتر بشه و هم reworking کمتر بشه و هم تایمی که داریم میدیم به صورت قابل توجهی سیو میشه --> پس این فیدبک به ما کمک میکنه که ما توی لوپ های بعدی منطقی تر کار رو انجام بدیم و اگر جاهایی در هنگام انجام کار اشتباه می رفتیم اون اشتباه رو زودتر متوجه بشیم و مانع این بشیم که اون اشتباه توی سیستم گسترش پیدا بکنه و اثرات اقتصادی و اعتباری و حتی روحی و روانی در کار داشته باشیم

Reference

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