

# Agile Principles

Dr. Elham Mahmoudzadeh

Isfahan University of Technology

[mahmoudzadeh@iut.ac.ir](mailto:mahmoudzadeh@iut.ac.ir)

2023

# What is Software Engineering?

*IEEE Computer Society Definition:*

- “Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software.”

"مهندسی نرم افزار به کارگیری یک رویکرد سیستماتیک، منظم و قابل سنجش برای توسعه، بهره برداری و نگهداری نرم افزار و مطالعه این رویکردها است. یعنی کاربرد مهندسی در نرم افزار.

# Software engineering(2)

## ■ Doing the right thing

- Software that **users want and need**
- Software that **benefits society**

## ■ Doing the thing right

- Following a **good software process**
- **Developing your programming skills**

کار درست را انجام دادن

- نرم افزاری که کاربران می خواهند و نیاز دارند  
درست انجام دادن کار

- دنبال کردن یک فرآیند نرم افزاری خوب  
- مهارت های برنامه نویسی خود را توسعه دهید

# Who is a software engineer?

- Software engineers are the creative minds behind computers or programs.
- A software engineer is the one who follows
  - A systematic process that leads to understanding the requirements,
  - Working with teams and various professionals
  - Design and create the application software or components or modules
  - Fulfill the specific needs of the users successfully;

مهندسان نرم افزار کیست؟

- مهندسان نرم افزار ذهن خلاقی هستند که پشت کامپیوترها یا برنامه ها قرار دارند.  
یک مهندس نرم افزار کسی است که دنبال می کند

- یک فرآیند سیستماتیک که منجر به درک الامات می شود،  
کار با تیم ها و متخصصان مختلف

- طراحی و ایجاد نرم افزار یا اجزا یا مأذول های کاربردی

- نیازهای خاص کاربران را با موفقیت برآورده کنید.

- چارچوبی برای به کارگیری شیوه های مهندسی نرم افزار با هدف خاص ارائه ابزار های لازم برای توسعه سیستم های فشرده نرم افزار.
- دارای دو قسمت باشد.
- 1. مجموعه ای از قراردادهای مدل سازی شامل یک زبان مدل سازی (نحو و معناشناسی)
- 2. یک فرآیند، که در مورد ترتیب فعالیت ها راهنمایی می کند،
- در مورد ترتیب فعالیت ها راهنمایی می کند،
- 

# Software Development Methodology (SDM)

- A **framework** for applying **software engineering practices** with the specific aim of providing the **necessary means** for developing **software-intensive systems**.
- Have two parts.
  1. A set of **modeling conventions** comprising a **Modeling Language** (**syntax** and **semantics**)
  2. A **Process**, which
    - provides guidance as to the **order of the activities**,
    - specifies **what artifacts** should **be developed** using the **Modeling Language**,
    - **directs the tasks** of **individual developers** and the **team** as a **whole**,
    - offers **criteria** for **monitoring** and **measuring** a project's **products** and **activities**.

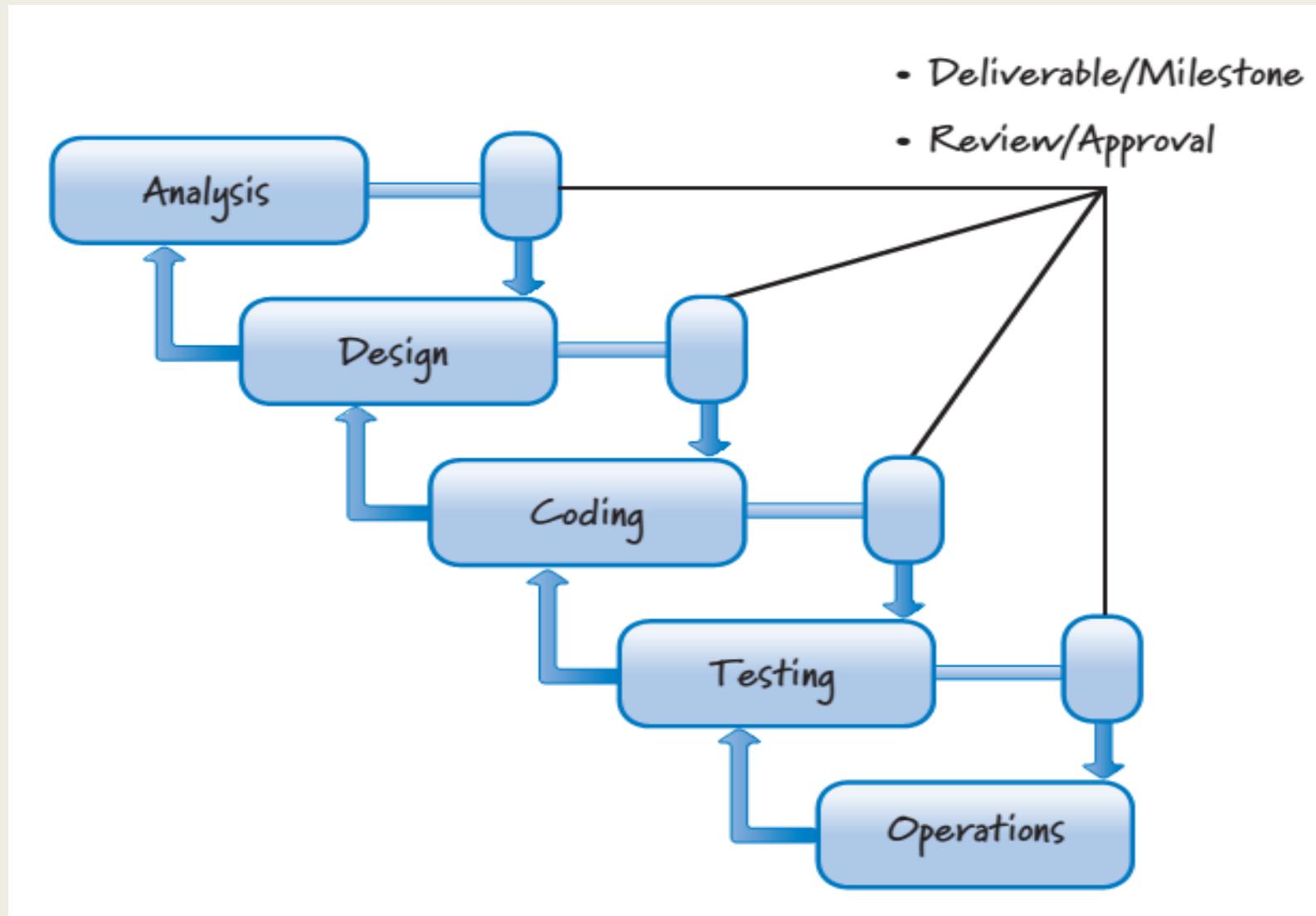
مشخص می کند که چه مصنوعاتی باید با استفاده از زبان مدل سازی توسعه داده شوند،  
■ وظایف توسعه دهنگان فردی و تیم را به عنوان یک کل هدایت می کند،  
■ معیارهایی برای نظارت و اندازه گیری محصولات و فعالیت های یک پروژه ارائه می دهد

# Agile vs. Traditional development process

- The goal of comparing agile principles with traditional development principles is not to make the case that plan-driven, sequential development is bad and that Scrum is good.
- Both are tools in the professional developer's toolkit; there is no such thing as a bad tool, rather just inappropriate times to use that tool.

Scrum and traditional, plan-driven, sequential development are appropriate to use on different classes of problems.

# Plan-driven process (Waterfall)



# Plan-driven process (I)

برای همه ویژگیهایی که ممکن است یک کاربر در محصول نهایی بخواهد از قبل برنامه ریزی و پیشینی کنید و تعیین کنید که چگونه میتوانید آن ویژگیها را بسازید.

■ ایده در اینجا این است که هر چه برنامه ریزی بهتر باشد، درک بهتر و در نتیجه اجرا بهتر است.

■ فرآیندهای متوالی نیز نامیده میشود، زیرا متخصصان به تدریج، تجزیه و تحلیل کامل نیازمندیها را انجام میدهند که پس از آن یک طراحی کامل و سپس کدگذاری/ساخت و سپس آزمایش انجام میشود.

- Plan for and anticipate up front **all of the features** a **user might want in the end product**, and to **determine how best to build those features**.
- The idea here is that **the better the planning, the better the understanding**, and therefore **the better the execution**.
- Also called **sequential processes** because practitioners perform, **in sequence**, a **complete requirements analysis** followed by a **complete design** followed in turn by **coding/building** and then **testing**.

اگر از آن برای مشکلاتی استفاده کنید که به خوبی تعریف شده، قابل پیش بینی هستند و بعید است که تغییر قابل توجهی داشته باشند، به خوبی کار می کند.

■ مشکل این است که بیشتر تلاشهای توسعه محصول غیرقابل پیش‌بینی هستند، به خصوص در ابتدا.

نابراین، در حالی که یک فرآیند برنامه محور، تصور یک رویکرد منظم، پاسخگو و قابل اندازه گیری را ایجاد می کند، این تصور می تواند منجر به احساس امنیت کاذب شود. از این گذشته، توسعه یک محصول به ندرت طبق برنامه پیش می رود.

## Plan-driven process (II)

- Works well if you are applying it to **problems** that are well defined, predictable, and unlikely to undergo any significant change.
- The problem is that **most product development efforts** are anything **but predictable**, especially **at the beginning**.
- So, while a plan-driven process gives the impression of an **orderly**, **accountable**, and **measurable approach**, that impression can lead to **a false sense of security**.

After all, developing a product rarely goes as planned.

# Plan-driven process (III)

- آن را درک کنید، طراحی کنید، آن را آزمایش کنید، آن را اجرا کنید، همه آن ها بر اساس یک برنامه خوب تعریف شده و تعیین شده.
- این اعتقاد وجود دارد که باید کار کند. اگر استفاده از یک رویکرد برنامه محور کارساز نباشد، نگرش غالب این است که ما باید کار اشتباهی انجام داده باشیم.
- مطمئن باشید که اگر آنها این کار را بهتر انجام دهند، نتایج آنها بهتر خواهد شد. اما مشکل از اجرا نیست.
- این است که رویکردهای برنامه محور مبتنی بر مجموعه ای از باورها هستند که با عدم قطعیت ذاتی در اکثر تلاش های توسعه محصول مطابقت ندارند.

- Understand it, design it, code it, test it, and deploy it, all according to a well-defined, prescribed plan.
- There is a belief that it should work. If applying a plan-driven approach doesn't work, the prevailing attitude is that we must have done something wrong.
- Sure that if they just do it better, their results will improve. The problem, however, is not with the execution.
- It's that plan-driven approaches are based on a set of beliefs that do not match the uncertainty inherent in most product development efforts.

# Traditional Pros.

- It is supremely logical.
- Think before you build.
- Write it all down.
- Follow a plan.
- Keep everything as organized as possible.

کاملاً منطقی است

- قبل از ساختن فکر کنید.
- همه را یادداشت کنید.
- یک برنامه را دنبال کنید.
- همه چیز را تا حد امکان سازماندهی کنید.

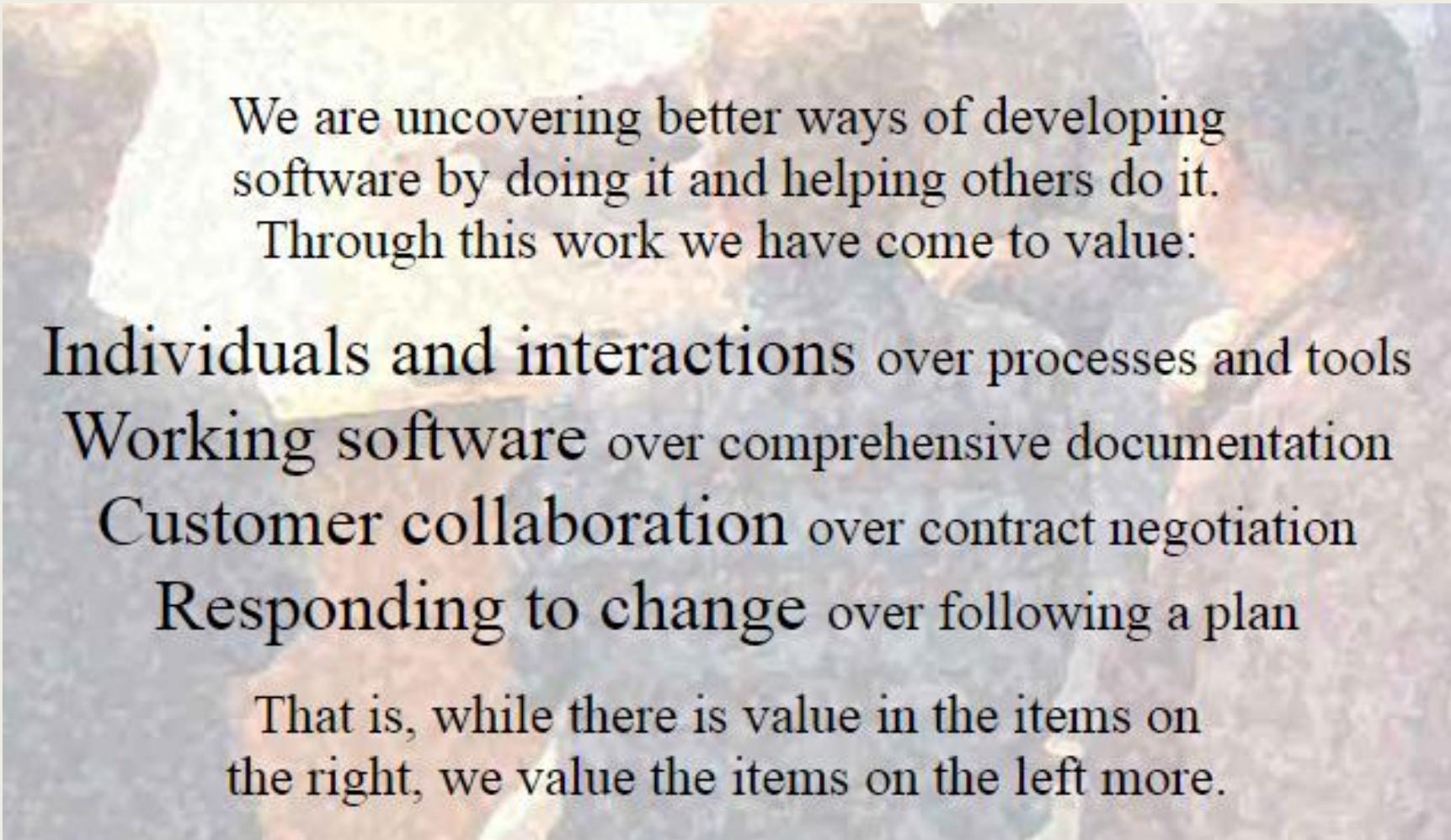
# What's Wrong With Traditional Software Development?

## Humans are involved.

- Creativity is inhibited.
- Written documents have their limitations.
- Bad timing.
- No crystal balls.
- Too much work and no fun.
- Sub-optimized results.

خلافیت مهار می شود.  
اسناد مکتوب محدودیت هایی دارند.  
زمان بندی بد.  
بدون گلوله های کریستالی.  
کار زیاد و بدون تفريح.  
نتایج زیر بهینه

# Manifesto for Agile Software Development



We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

**Individuals and interactions** over processes and tools

**Working software** over comprehensive documentation

**Customer collaboration** over contract negotiation

**Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

# Principles Behind the Agile Manifesto(I)

1. Our **highest priority** is to **satisfy the customer** through **early** and **continuous** delivery of **valuable software**.
2. **Welcome changing requirements**, even **late** in development. Agile processes harness change for the customer's competitive advantage.
3. **Deliver working software** **frequently**, from a couple of weeks to a couple of months, with a **preference** to the **shorter timescale**.
4. **Business people** and **developers** must **work together daily** throughout the project.

بالاترین اولویت ما جلب رضایت مشتری از طریق تحویل زودهنگام و مستمر نرم افزارهای ارزشمند است.  
2. از نیازهای تغییر استقبال کنید، حتی در اواخر توسعه. فرآیندهای چاک تغییر را برای مزیت رقابتی مشتری مهار می کنند.  
3. نرم افزار کار را به طور مکرر، از چند هفته تا چند ماه، با اولویت به مقیاس زمانی کوتاه تر، تحویل دهید.  
4. افراد تجاری و توسعه دهندگان باید روزانه در طول پروژه با یکدیگر همکاری کنند

# Principles Behind the Agile Manifesto(II)

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

5. پروژه ها را حول افراد با انگیزه بسازید. به آنها محیط و حمایت لازم را بدهید و برای انجام کار به آنها اعتماد کنید.

6. کارآمدترین و مؤثرترین روش انتقال اطلاعات به تیم توسعه و درون آن، حضوری است.

7. نرم افزار کار معيار اولیه پیشرفت است.

8. فرآیندهای چاپک توسعه پایدار را ترویج می کنند. حامیان مالی، توسعه دهندگان و کاربران باید بتوانند به طور نامحدود یک سرعت ثابت را حفظ کنند.

# Principles Behind the Agile Manifesto(III)

9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

9. توجه مداوم به برتری فنی و طراحی خوب، چابکی را افزایش می دهد.

10. سادگی – هنر به حداقل رساندن مقدار کار انجام نشده – ضروری است.

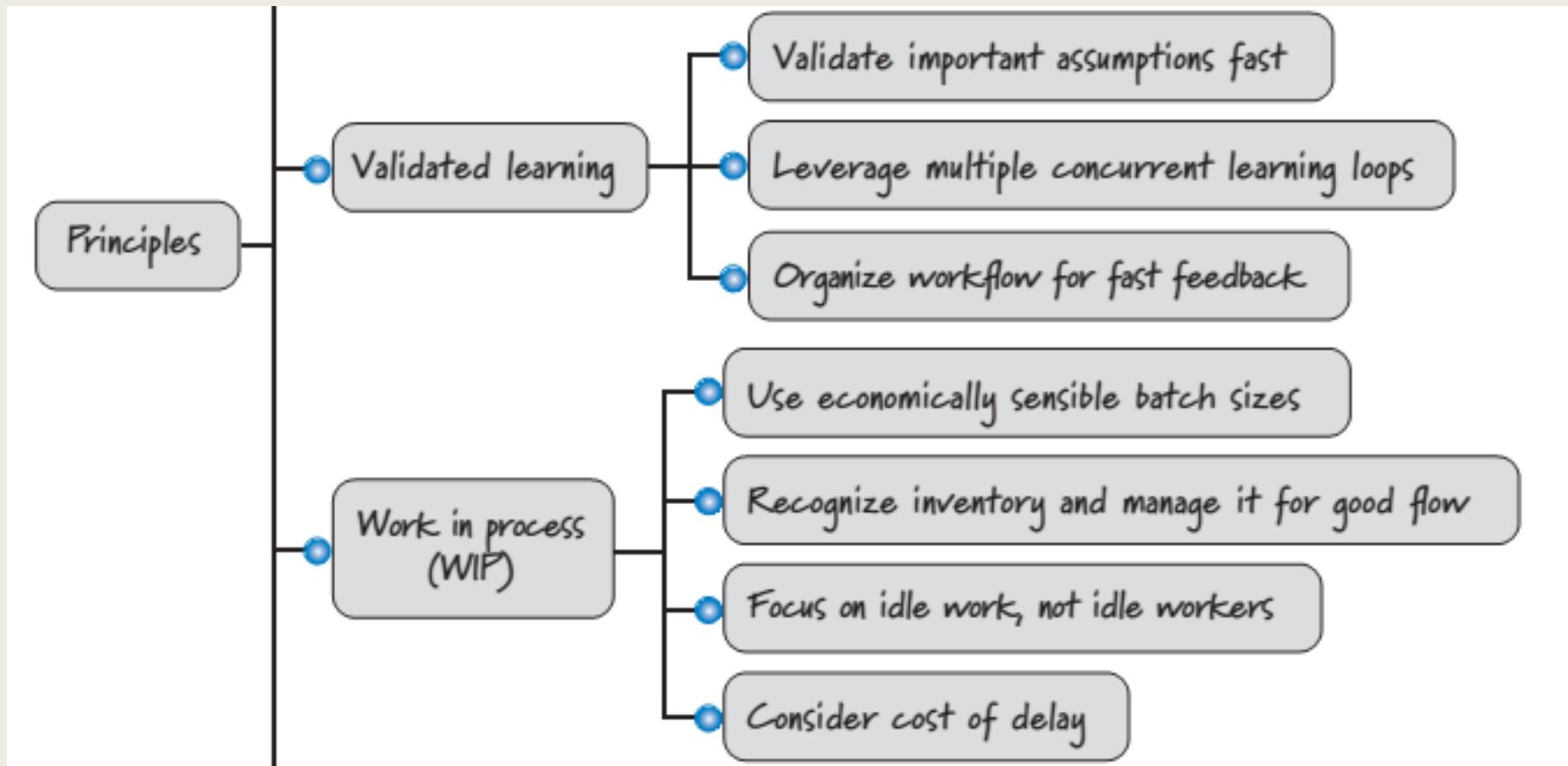
11. بهترین معماری ها، نیازمندی ها و طرح ها از تیم های خودسازمانده پدید می آیند.

12. در فواصل زمانی معین، تیم در مورد چگونگی موثرتر شدن فکر می کند، سپس رفتار خود را بر اساس آن تنظیم و تنظیم می کند.

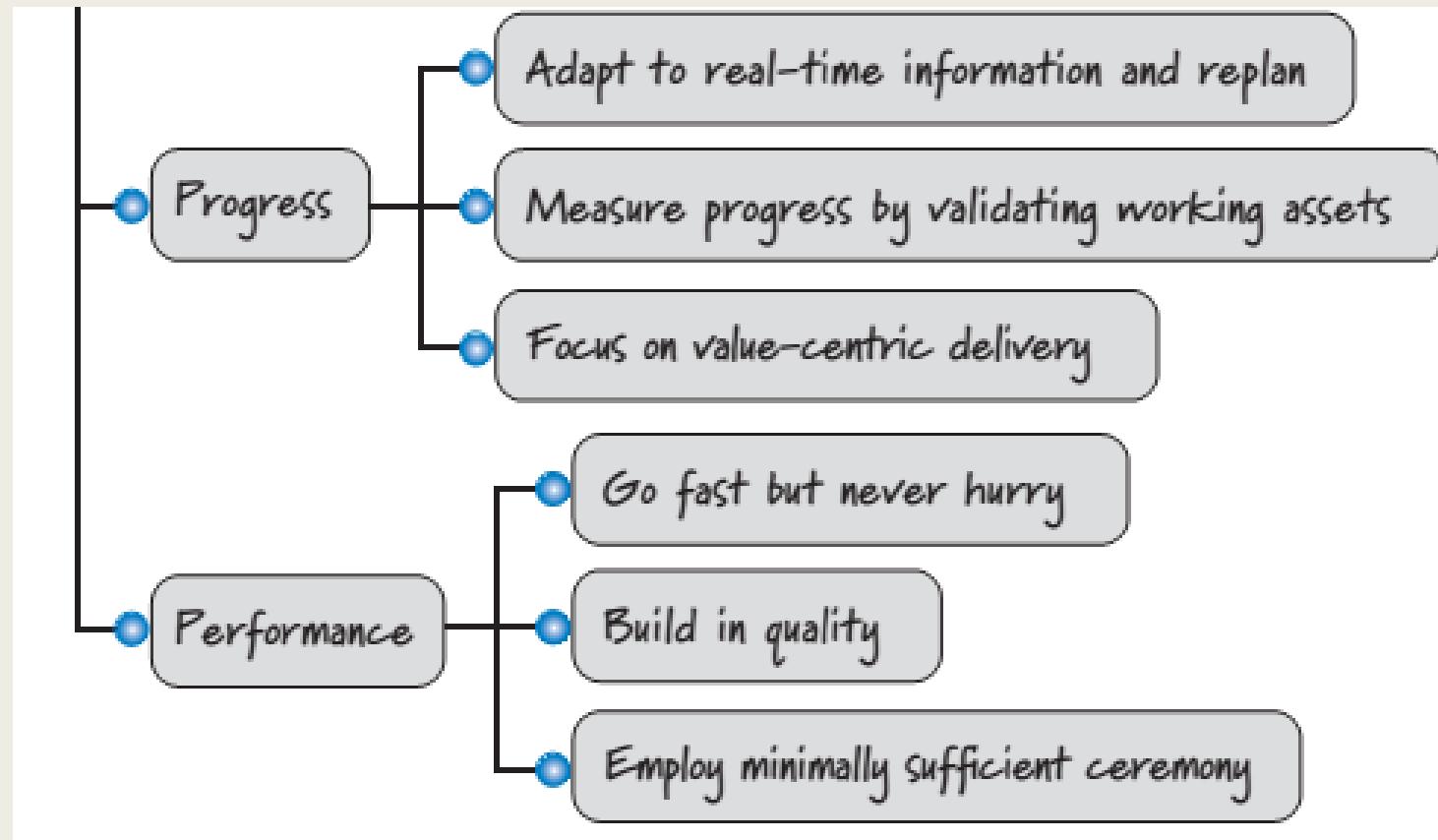
# Categorization of principles (Up)



# Categorization of principles (Middle)



# Categorization of principles (Bottom)



# Variability and Uncertainty

- To **create innovative solutions.**
- Four related principles

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.***

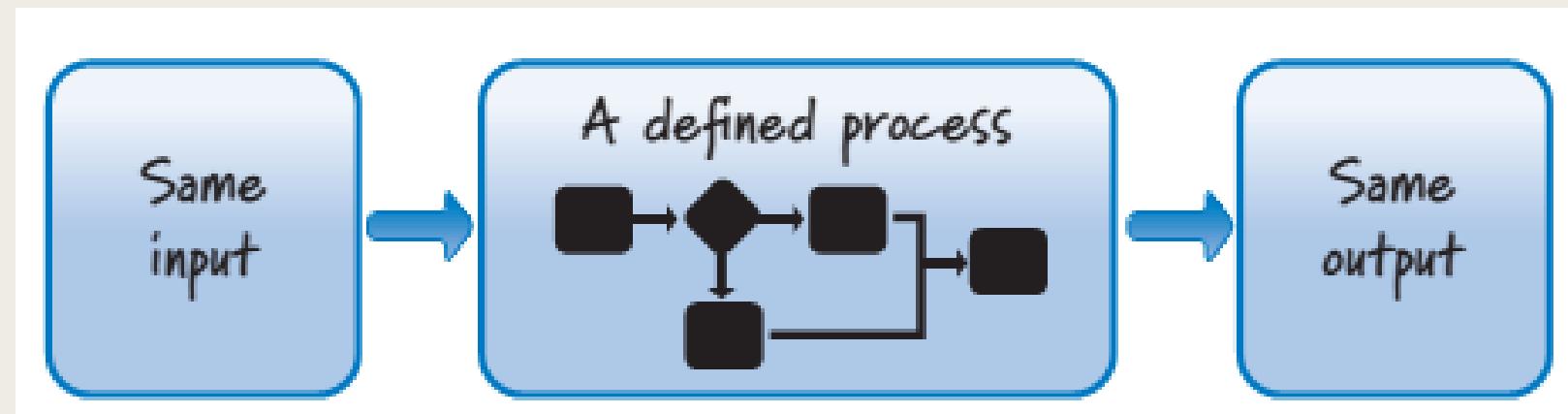
تغییرپذیری و عدم قطعیت  
■ ایجاد راه حل های نوآورانه.  
■ چهار اصل مرتبط  
1. تنوع مفید را بپذیرید.  
2. توسعه تکراری و افزایشی را به کار بگیرید.  
3. تغییرپذیری را از طریق بازرگانی، انطباق و شفافیت افزایش دهید.  
4. همه اشکال عدم قطعیت را به طور همزمان کاهش دهید.

فرآیندهای برنامه محور با توسعه محصول مانند تولید رفتار می کنند. در تولید، هدف ما این است که مجموعه ای ثابت از الزامات را در نظر بگیریم و مجموعه ای متوالی از مراحل کاملاً درک شده را برای تولید محصول نهایی که هر بار یکسان است (در محدوده واریانس تعریف شده) دنبال کنیم.

- در توسعه محصول، هدف ایجاد یک نمونه منحصر به فرد از محصول است، نه تولید محصول. این نمونه واحد مشابه یک دستور العمل منحصر به فرد است

# Embrace Helpful Variability (I)

- Plan-driven processes treat product development like 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 same (within a defined variance range) every time.
- In product development, the goal is to create the unique single instance of the product, not to manufacture the product. This single instance is analogous to a unique recipe.



# Embrace Helpful Variability (II)

- We don't want to create the same recipe twice;
- Instead, we want to create a unique recipe for a new product. Some amount of variability is necessary to produce a different product each time.
- In fact, every feature we build within a product is different from every other feature within that product, so we need variability even at this level.

ما نمی خواهیم یک دستور غذا را دو بار ایجاد کنیم.

■ در عوض، ما می خواهیم یک دستور العمل منحصر به فرد برای ک محصول جدید ایجاد کنیم. مقداری تنوع برای تولید هر بار محصول متفاوت ضروری است.

■ در واقع، هر ویژگی که در یک محصول ایجاد می کنیم با هر ویژگی دیگری در آن محصول متفاوت است، بنابراین ما حتی در این سطح به تنوع نیاز داریم.

# Employ Iterative and Incremental Development (I)

- Plan-driven, sequential development assumes that we will get things right up front and that most or all of the product pieces will come together late in the effort.
- Scrum, on the other hand, is based on iterative and incremental development.

بکارگیری توسعه تکراری و افزایشی (I)  
■ توسعه پی در پی برنامه محور فرض می کند که ما همه چیز را به درستی انجام خواهیم داد و اکثر یا همه قطعات محصول در اواخر تلاش گرد هم می آیند.  
■ اسکرام، از سوی دیگر، مبتنی بر توسعه تکراری و افزایشی است.

■ تصدیق می کند که ما احتمالاً قبل از اینکه کارها را درست انجام دهیم اشتباه خواهیم کرد و قبل از اینکه آنها را به خوبی انجام دهیم آنها را ضعیف انجام خواهیم داد.

■ یک استراتژی مجدد برنامه ریزی شده است. ما از چندین پاس برای بهبود چیزی که می سازیم استفاده می کنیم تا بتوانیم روی یک راه حل خوب همگرا شویم.

■ توسعه تکراری یک راه عالی برای بهبود محصول در حال توسعه است.

■ بزرگترین نقطه ضعف این است که در صورت عدم قطعیت، تعیین (برنامه ریزی) تعداد پاس های بهبودی لازم است.

# Iterative Development (I)

- Acknowledges that we will probably get things wrong before we get them right and that we will do things poorly before we do them well (Goldberg and Rubin 1995).
- Is a planned rework strategy. We use multiple passes to improve what we are building so that we can converge on a good solution.
- Iterative development is an excellent way to improve the product as it is being developed.
- The biggest downside is that in the presence of uncertainty it can be difficult up front to determine (plan) how many improvement passes will be necessary.

# Iterative Development (II)

- For example, we might start by creating a prototype to acquire important knowledge about a poorly known piece of the product. Then we might create a revised version that is somewhat better, which might in turn be followed by a pretty good version.
- In the course of writing this book, for example, I wrote and rewrote each of the chapters several times as I received feedback and as my understanding of how I wanted to communicate a topic improved.

به عنوان مثال، ممکن است با ایجاد یک نمونه اولیه برای کسب دانش مهم در مورد یک قطعه ضعیف از محصول شروع کنیم. سپس ممکن است یک نسخه اصلاح شده ایجاد کنیم که تا حدودی بهتر است، که به نوبه خود ممکن است یک نسخه بسیار خوب به دنبال داشته باشد.

■ در طول نوشتمن این کتاب، به عنوان مثال، من هر یک از فصل‌ها را چندین بار نوشتتم و بازنویسی کردم که بازخورد دریافت می‌کردم و درکم از اینکه چگونه می‌خواهم با یک موضوع ارتباط برقرار کنم بهتر شد.

- بر اساس اصل قدیمی "قبل از ساختن همه آن مقداری از آن را بسازید." ما از داشتن یک سبک بزرگ و بیگ بنگ در پایان توسعه اجتناب می کنیم.
- در عوض، محصول را به قطعات کوچکتر تقسیم میکنیم تا بتوانیم قادری از آن را بسازیم، یاد بگیریم که هر قطعه چگونه در محیطی که باید وجود داشته باشد، زندگ بماند، بر اساس آنچه یاد میگیریم تطبیق دهیم و سپس تعداد بیشتری از آن را بسازیم.

# Incremental Development (I)

- Based on the age-old principle of "Build some of it before you build all of it." We avoid having one large, big-bang-style at the end of development.
- Instead, we break the product into smaller pieces so that we can build some of it, learn how each piece is to survive in the environment in which it must exist, adapt based on what we learn, and then build more of it.
- Incremental development slices the system functionality into increments (portions).
- Incremental development gives us important information that allows us to adapt our development effort and to change how we proceed.
- The biggest drawback to incremental development is that by building in pieces, we risk missing the big picture (we see the trees but not the forest).

توسعه افزایشی عملکرد سیستم را به چند قسمت (بخش) تقسیم می کند.

- توسعه تدریجی اطلاعات مهمی به ما می دهد که به ما امکان می دهد تلاش های توسعه خود را تطبیق دهیم و نحوه ادامه کار را تغییر دهیم.
- بزرگترین اشکال توسعه تدریجی این است که با تکه تکه سازی، خطر از دست دادن تصویر بزرگ را داریم (درختان را می بینیم اما جنگل را نه)

# Incremental Development

- For example, while writing this book, I wrote a chapter at a time and sent each chapter out for review as it was completed, rather than trying to receive feedback on the entire book at once.
- This gave me the opportunity to incorporate that feedback into future chapters, adjusting my tone, style, or delivery as needed.
- It also gave me the opportunity to learn incrementally and apply what I learned from earlier chapters to later chapters.

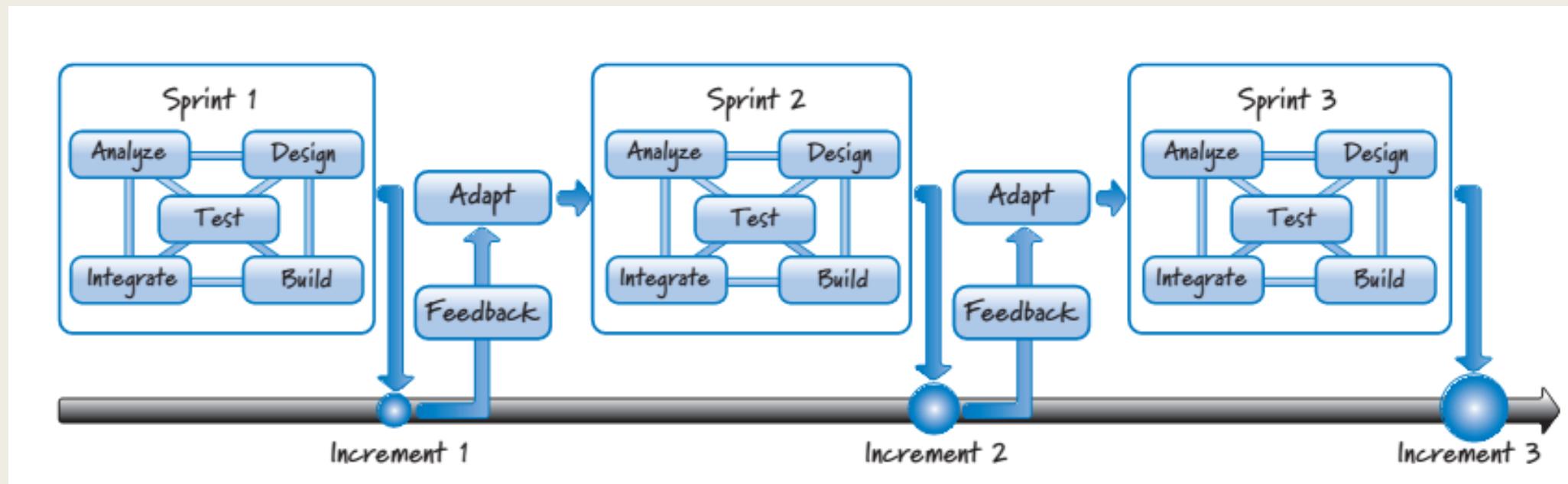
# Iterative and Incremental Development in Scrum (I)

- Scrum leverages the benefits of both iterative and incremental development, while negating the disadvantages of using them individually.
- Scrum does this by using both ideas in an adaptive series of time boxed iterations called sprints.

اسکرام از مزایای توسعه تکراری و افزایشی بهره می برد، در حالی که معایب استفاده از آنها را به صورت جداگانه نمی کند.

■ اسکرام این کار را با استفاده از هر دو ایده در یک سری تطبیقی از تکرارهای جعبه زمانی به نام اسپرینت انجام می دهد.

# Iterative and Incremental Development in Scrum (II)



# Iterative and Incremental Development in Scrum (III)

- During each sprint we perform all of the activities necessary to create a working product increment (some of the product, not all of it).
- This all-at-once approach has the benefit of quickly validating the assumptions that are made when developing product features.
- For example, we make some design decisions, create some code based on those decisions, and then test the design and code—all in the same sprint. By doing all of the related work within one sprint, we are able to quickly rework features, thus achieving the benefits of iterative development.

در طول هر دوی سرعت، ما تمام فعالیتهای لازم برای ایجاد یک افزایش محصول فعل (بعضی از محصول، نه همه آن) را انجام میدهیم.

■ این رویکرد یکباره دارای مزیت اعتبارسنجی سریع مفروضاتی است که هنگام توسعه ویژگی های محصول ایجاد می شود.

■ برای مثال، ما برخی تصمیمات طراحی میگیریم، کدی را بر اساس آن تصمیمهای ایجاد میکنیم، و سپس طرح و کد را آزمایش میکنیم—همه در یک اسپرینت. با انجام تمام کارهای مربوطه در یک اسپرینت، میتوانیم ویژگیها را به سرعت دوباره کار کنیم، بنابراین به مزایای توسعه تکراری دست یابیم.

# Iterative and Incremental in Scrum (IV)

- In Scrum, we don't work on a phase at a time; we work on a **feature** at a time.
- So, by the **end of a sprint** we have created a **valuable product increment** (some but not all of the product features). 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**. This helps us **view the product** from **more of a big-picture perspective** than we might otherwise have.

■ در اسکرام، ما روی یک فاز کار نمی کنیم. ما در یک زمان روی یک ویژگی کار می کنیم.  
■ بنابراین، در پایان یک اسپرینت، ما یک افزایش محصول با ارزش (بعضی اما نه همه ویژگی های محصول) ایجاد کرده ایم. این افزایش شامل یا یکپارچه شده و با هر ویژگی قبلاً توسعه یافته آزمایش شده است. در غیر این صورت انجام شده تلقی نمی شود.  
■ در پایان اسپرینت، میتوانیم در مورد ویژگیهای جدید تکمیل شده در چارچوب ویژگیهای از قبل تکمیل شده، بازخورد دریافت کنیم. این به ما کمک می کند تا محصول را از منظر تصویری بزرگتر از آنچه در غیر این صورت می دیدیم بینیم.

# Iterative and Incremental Scrum (V)

- ما در مورد نتایج اسپرینت بازخورد دریافت می کنیم، که به ما امکان می دهد تا خود را تطبیق دهیم.
- میتوانیم ویژگیهای مختلفی را برای کار در سرعت بعدی انتخاب کنیم یا فرآیندی را که برای ساخت مجموعه بعدی از ویژگیها استفاده خواهیم کرد، تغییر دهیم.
- در برخی موارد، ممکن است یاد بگیریم که افزایش، اگرچه از نظر فنی با این صورتحساب مطابقت دارد، آنقدر که میتوانست خوب نیست. زمانی که این اتفاق افتاد، میتوانیم به عنوان بخشی از تعهد خود به توسعه مکرر و بهبود مستمر، کار مجدد را برای یک اسپرینت آینده برنامه ریزی کنیم.
- این به غلبه بر این مسئله کمک می کند که از قبل دقیقاً به تعداد پاس های بهبود نیاز نداریم غلبه کنیم.
- اسکرام نیازی ندارد که تعداد مجموعه ای از تکرارها را از قبل تعیین کنیم.
- جریان پیوسته بازخورد ما را راهنمایی می کند تا در حین توسعه تدریجی محصول، تعداد تکرارهای مناسب و معقول از نظر اقتصادی را انجام دهیم.

- We receive feedback on the sprint results, which allows us to adapt.
- 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.
- In some cases, we might learn that the increment, though it technically fits the bill, isn't as good as it could be. When that happens, we can schedule rework for a future sprint as part of our commitment to iterative development and continuous improvement.
- This helps overcome the issue of not knowing up front exactly how many improvement passes we will need.
- 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 while developing the product incrementally.

# Leverage Variability through Inspection, Adaptation, and Transparency

اهرم تغییرپذیری از طریق بازرگاری، سازگاری و شفافیت

■ یک فرآیند توسعه متوالی و مبتنی بر برنامه، تغییرات خروجی کمی یا بدون تغییر را در نظر می‌گیرد. مجموعه‌ای از مراحل کاملاً تعریف شده را دنبال می‌کند و در اواخر فرآیند فقط از مقادیر کمی بازخورد استفاده می‌کند.

- A plan-driven, sequential development process assumes little or no output variability. It follows a well-defined set of steps and uses only small amounts of feedback late in the process.
- Scrum embraces the fact that in product development, some level of variability is required in order to build something new.
- Scrum also assumes that the process necessary to create the product is complex and therefore would defy a complete up-front definition. Furthermore, it generates early and frequent feedback to ensure that the right product is built and that the product is built right.

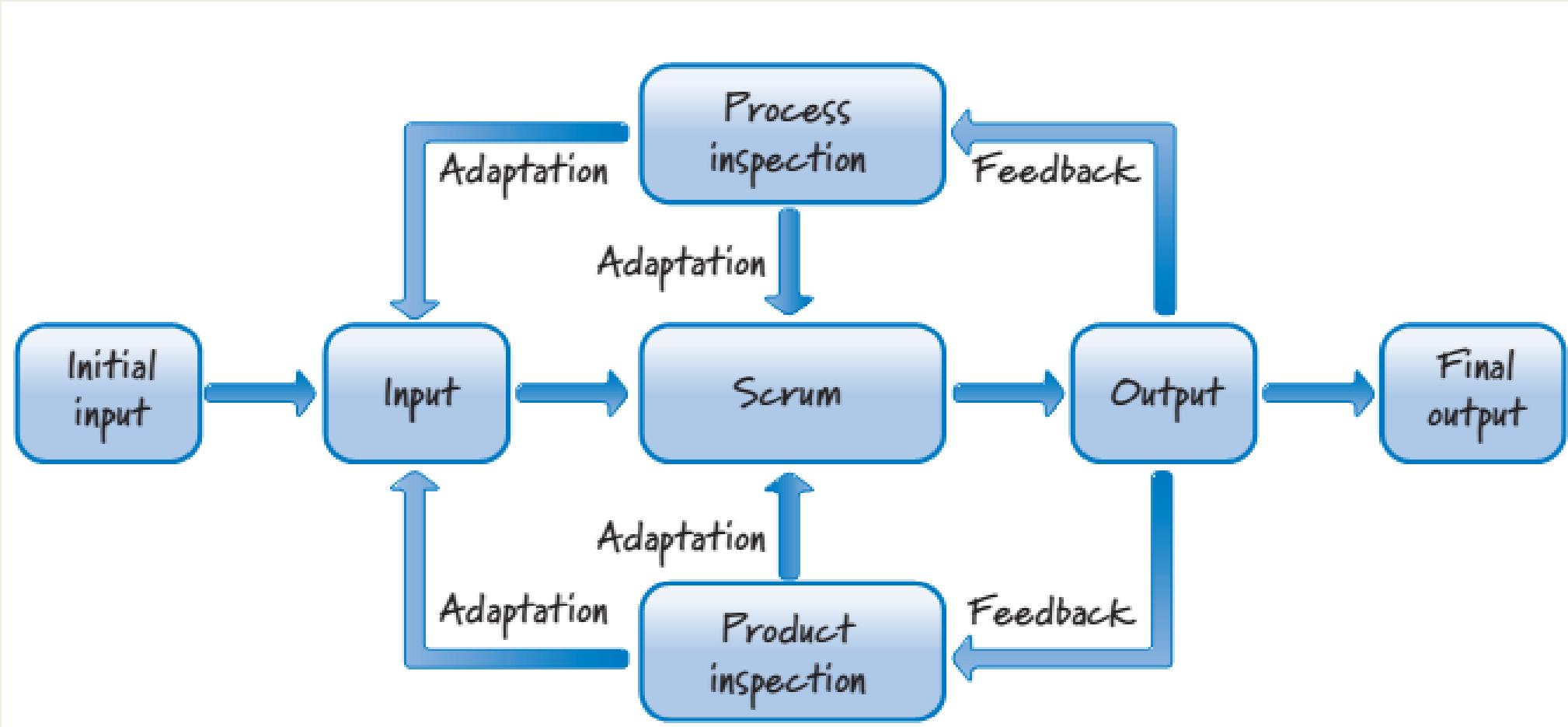
■ اسکرام این واقعیت را پذیرفته است که در توسعه محصول، مقداری از تنوع برای ساختن چیزی جدید مورد نیاز است.

■ اسکرام همچنین فرض می‌کند که فرآیند لازم برای ایجاد محصول پیچیده است و بنابراین با یک تعریف اولیه کامل مخالفت می‌کند. علاوه بر این، بازخورد اولیه و مکرر را ایجاد می‌کند تا اطمینان حاصل شود که محصول مناسب ساخته شده است و محصول درست ساخته شده است

# Comparison

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

# Scrum Process model



# Inspect, Adapt and Transparency

- Are heart of Scrum.
- In Scrum, we inspect and adapt not only what we are building but also how we are building it

بازرسی، تطبیق و شفافیت

■ قلب اسکرام هستند.

■ در اسکرام، مانه تنها آنچه را که می سازیم، بلکه نحوه ساخت آن را نیز بررسی و تطبیق می دهیم

- تمام اطلاعاتی که برای تولید یک محصول مهم است باید در دسترس افراد دخیل در ایجاد محصول باشد.
- شفافیت بازرسی را ممکن میسازد، که برای سازگاری لازم است.
- شفافیت همچنین به همه افراد ذینفع اجازه می دهد تا آنچه را که اتفاق می افتد مشاهده و درک کنند.
- منجر به ارتباطات بیشتر و ایجاد اعتماد (هم در فرآیند و هم در بین اعضای تیم) می شود.

# Transparency

- All of the **information** that is **important** to producing a product must be **available** to the people **involved** in **creating** the **product**.
- Transparency makes **inspection** possible, which is **needed** for **adaptation**.
- Transparency also **allows** **everyone** concerned to **observe** and **understand** what is happening.
- It leads to **more communication** and it **establishes trust** (both in the **process** and among **team members**).

# Reduce All Forms of Uncertainty Simultaneously (I)

- Developing new products is a complex endeavor with a high degree of uncertainty.
- Three categories of uncertainty.
  1. *End uncertainty (what uncertainty)—uncertainty surrounding the features of the final product.*
  2. *Means uncertainty (how uncertainty)—uncertainty surrounding the process and technologies used to develop a product.*
  3. *Customer uncertainty (who uncertainty) —who the actual customers of their products will be.*
- This uncertainty must be addressed or they might build brilliant products for the wrong markets.

کاهش همه اشکال عدم قطعیت به طور همزمان (I)

- توسعه محصولات جدید یک تلاش پیچیده با درجه بالایی از عدم اطمینان است.
- سه دسته عدم قطعیت.

1. پایان دادن به عدم قطعیت (چه عدم قطعیت) - عدم اطمینان پیرامون ویژگی های محصول نهایی.

2. به معنی عدم قطعیت (چگونه عدم قطعیت) - عدم اطمینان در مورد فرآیند و فناوری های مورد استفاده برای توسعه یک محصول.

3. عدم اطمینان مشتری (چه عدم اطمینان) - مشتریان واقعی محصولات آنها چه کسانی خواهند بود.

■ این عدم قطعیت باید برطرف شود و گرنه ممکن است محصولات درخشنده برای بازارهای اشتباہ بسازند.

# Reduce All Forms of Uncertainty Simultaneously (II)

- Traditional, sequential development processes focus first on eliminating all end uncertainty by fully defining up front what is to be built, and only then addressing means uncertainty.
- This simplistic, linear approach to uncertainty reduction is ill suited to the complex domain of product development, where our actions and the environment in which we operate mutually constrain one another.

رآیندهای توسعه سنتی و متوالی ابتدا بر حذف تمام عدم قطعیت های نهایی با تعریف کامل آنچه قرار است ساخته شود، تمرکز می کنند، و تنها پس از آن پرداختن به معنای عدم قطعیت است.  
■ این رویکرد ساده و خطی برای کاهش عدم قطعیت برای حوزه پیچیده توسعه محصول مناسب نیست، جایی که اقدامات ما و محیطی که در آن عمل می کنیم، یکدیگر را محدود می کنند.

# Reduce All Forms of Uncertainty Simultaneously (III), An Example

- We decide to build a feature (our action).
- We then show that feature to a customer, who, once he sees it, changes his mind about what he really wants, or realizes that he did not adequately convey the details of the feature (our action elicits a response from the environment).
- We make design changes based on the feedback (the environment's reaction influences us to take another unforeseen action).

در اسکرام، قبل از پرداختن به نوع بعدی، خود را با پرداختن کامل به یک نوع عدم قطعیت محدود نمی کنیم.  
■ در عوض، مابه طور همزمان بر کاهش همه عدم قطعیت ها (هدف، وسیله، مشتری و غیره) تمرکز می کنیم.

■ البته، در هر مقطع زمانی ممکن است بیشتر بر یک نوع عدم قطعیت تمرکز کنیم تا نوع دیگر. پرداختن به انواع مختلف عدم قطعیت به طور همزمان توسعه تکراری و تدریجی تسهیل میشود و با بازرسی، انطباق و شفافیت مداوم هدایت میشود.

■ امکان کاوش و کاوش فرصتطلبانه در محیط خود را برای شناسایی و یادگیری ناشناخته های ناشناخته (چیز هایی که هنوز نمیدانیم و نمیدانیم) به هنگام ظهور آنها را میدهد.

# Reduce All Forms of Uncertainty Simultaneously (IV)

- In Scrum, we do not constrain ourselves by fully addressing one type of uncertainty before we address the next type.
- Instead, we focus on simultaneously reducing all uncertainties (end, means, customer, and so on).
- Of course, at any point in time we might focus more on one type of uncertainty than another. Simultaneously addressing multiple types of uncertainty is facilitated by iterative and incremental development and guided by constant inspection, adaptation, and transparency.
- Allows to opportunistically probe and explore our environment to identify and learn about the unknown unknowns (the things that we don't yet know that we don't know) as they emerge.

# Agile Principles(III)

Dr. Elham Mahmoudzadeh

Isfahan University of Technology

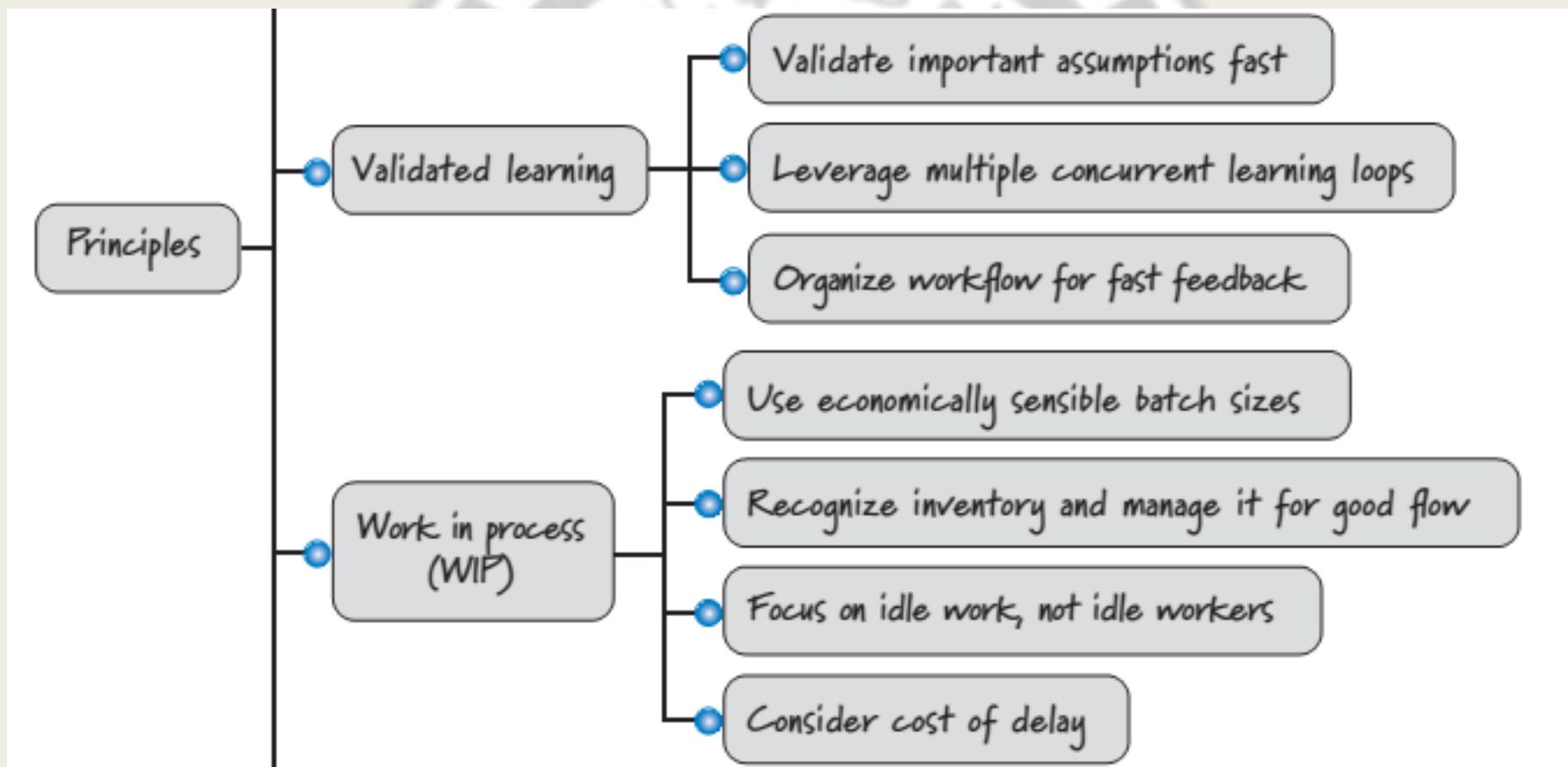
[mahmoudzadeh@iut.ac.ir](mailto:mahmoudzadeh@iut.ac.ir)

2022

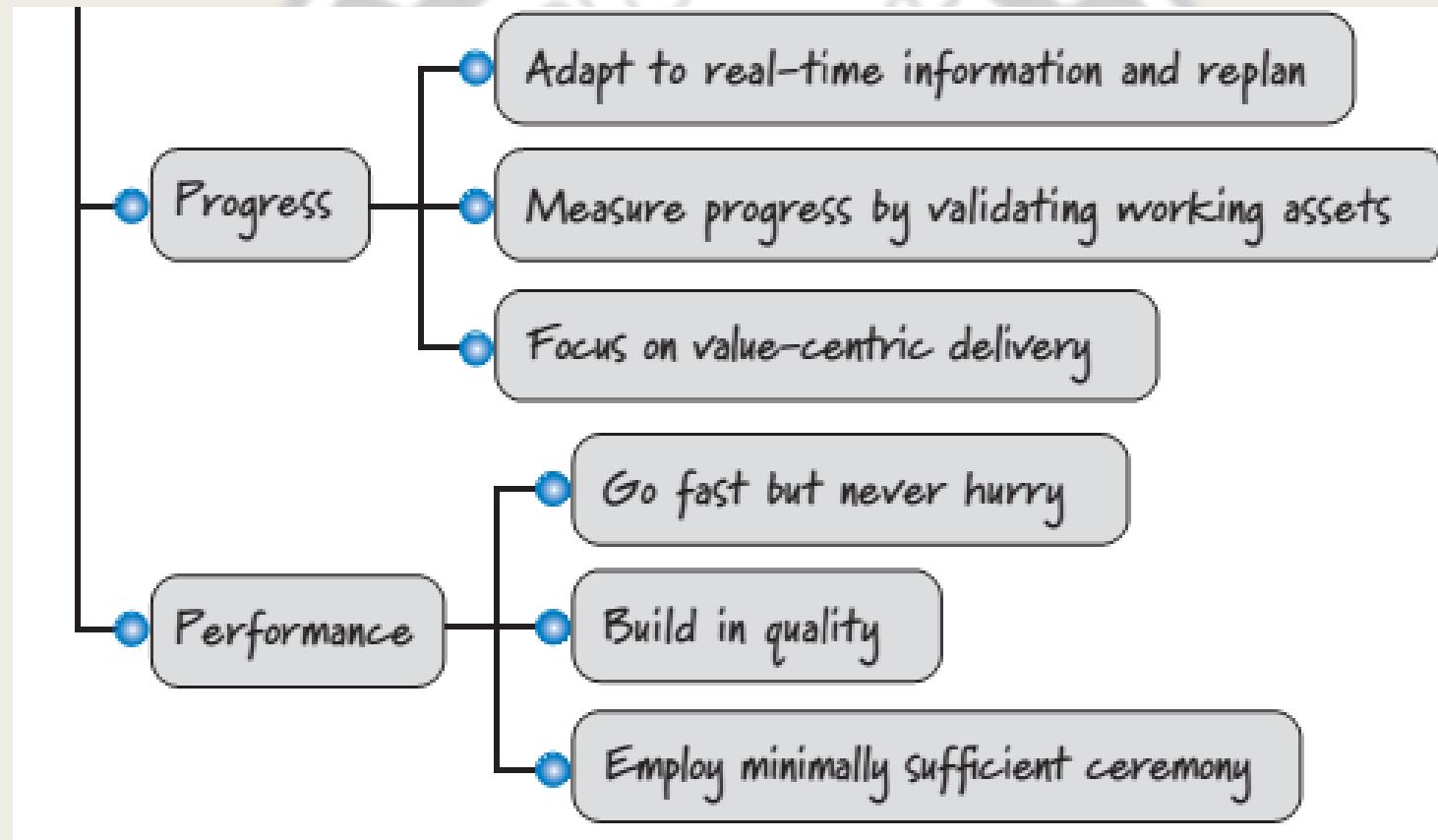
# Categorization of principles (Up)



# Categorization of principles (Middle)



# Categorization of principles (Bottom)



در اسکرام، ما دائماً میل به پیش بینی را با نیاز به انطباق متعادل می کنیم.

بنج اصل مرتبه

گزینه ها را باز نگه دارید

بپذیرید که نمی توانید آن را از قبل دریافت کنید.

از یک رویکرد تطبیقی و اکتشافی حمایت کنید.

تغییر را به رو شی معقول اقتصادی بپذیرید.

کار پیش‌بینی‌کننده را با کار بهموقع تطبیقی متعادل کنید.

# Prediction and Adaptation

- In Scrum, we are constantly balancing the desire for prediction with the need for adaptation.
- Five related principles
  1. *Keep options open.*
  2. *Accept* that you *can't get it right up front.*
  3. *Favor an adaptive, exploratory approach.*
  4. *Embrace change in an economically sensible way.*
  5. *Balance predictive up-front work with adaptive just-in-time work.*

# Keep Options Open (I)

توسعه متوالی مبتنی بر برنامه مستلزم آن است که تصمیمات مهم در زمینه هایی مانند الزامات یا طراحی در مراحل مربوطه اتخاذ، بررسی و تایید شوند.

علاوه، این تصمیمات باید قبل از انتقال به مرحله بعدی گرفته شوند، حتی اگر این تصمیمات بر اساس دانش محدود باشد.

- Plan-driven, sequential development requires that important decisions in areas like requirements or design be made, reviewed, and approved within their respective phases.
- Furthermore, these decisions must be made before we can transit to the next phase, even if those decisions are based on limited knowledge.

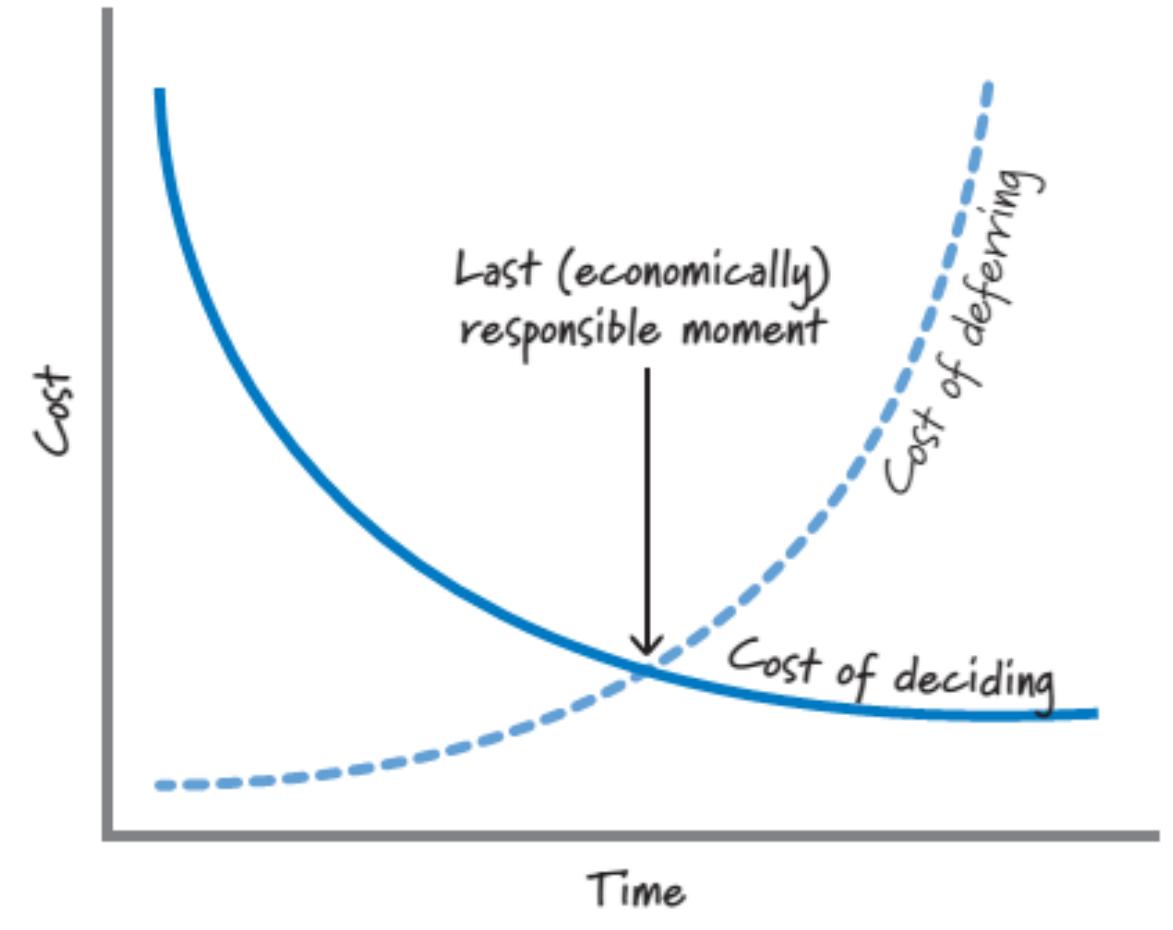
اسکرام مدعی است که ما هرگز نباید تصمیم زودهنگام بگیریم فقط به این دلیل که یک فرآیند عمومی حکم می کند که اکنون زمان تعیین شده برای تصمیم گیری است.

■ در اسکرام، ما از استراتژی باز نگه داشتن گزینه هایمان حمایت می کنیم.  
■ اغلب از این اصل به عنوان آخرین لحظه مسئولیت پذیر (LRM) یاد می شود، به این معنی که تعهد را به تأخیر می اندازیم و تا آخرین لحظه مسئولیت تصمیمات مهم و غیرقابل برگشت نمی گیریم. LRM زمانی است که هزینه عدم تصمیم گیری بیشتر از هزینه تصمیم گیری می شود. در آن لحظه ما تصمیم می گیریم.

# Keep Options Open (II)

- Scrum contends that we should never make a premature decision just because a generic process would dictate that now is the appointed time to make one.
- In Scrum, we favor a strategy of keeping our options open.
- Often this principle is referred to as the last responsible moment (LRM), meaning that we delay commitment and do not make important and irreversible decisions until the last responsible moment. LRM is when the cost of not making a decision becomes greater than the cost of making a decision. At that moment, we make the decision.

# Keep Options Open (III)



# Keep Options Open (IV)

ر اولین روز تلاش برای توسعه محصول، ما کمترین اطلاعات را در مورد کاری که انجام می دهیم داریم.  
در هر روز بعدی از تلاش برای توسعه، کمی بیشتر یاد می کیریم.  
بسیاری از ما ترجیح می دهیم صبر کنیم تا اطلاعات بیشتری بدست آوریم تا بتوانیم تصمیم آگاهانه تری بگیریم.

- On the first day of a product development effort, we have the least information about what we are doing.
- On each subsequent day of the development effort, we learn a little more.
- Most of us would prefer to wait until we have more information so that we can make a more informed decision.

# Keep Options Open (V)

هنگام برخورد با تصمیمات مهم یا غیرقابل برگشت، اگر خیلی زود تصمیم بگیریم و اشتباه کنیم، در قسمت تصاعدی منحنی هزینه تصمیم گیری خواهیم بود.  
■ همانطور که درک بهتری از تصمیم به دست می آوریم، هزینه تصمیم گیری کاهش می یابد (احتمال تصمیم گیری بد به دلیل افزایش اطمینان بازار یا فنی کاهش می یابد). به همین دلیل است که قبل از تصمیم گیری باید منتظر بمانیم تا اطلاعات بهتری داشته باشیم

- When dealing with important or irreversible decisions, if we decide too early and are wrong, we will be on the exponential part of the cost-of-deciding curve.
- As we acquire a better understanding regarding the decision, the cost of deciding declines (the likelihood of making a bad decision declines because of increasing market or technical certainty). That's why we should wait until we have better information before committing to a decision.

# Accept That You Can't Get It Right Up Front(I)

- Plan-driven processes not only mandate full requirements and a complete plan; they also assume that we can “get it right” up front.
- 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.
- What's worse is that when the requirements do change, we have to modify the baseline requirements and plans to match the current reality.

بپذیرید که نمی توانید آن را به درستی انجام دهید (I)

■ فرآیندهای برنامه محور نه تنها الزامات کامل و یک برنامه کامل را الزامی می کنند. آنها همچنین فرض میکنند که ما میتوانیم آن را «درست» پیش ببریم.

■ واقعیت این است که خیلی بعید است که بتوانیم همه الزامات یا برنامه های دقیق بر اساس آن الزامات را از قبل اصلاح کنیم.

■ بدتر از آن این است که وقتی الزامات تغییر می کنند، باید الزامات و برنامه های پایه را اصلاح کنیم تا با واقعیت فعلی مطابقت داشته باشد.

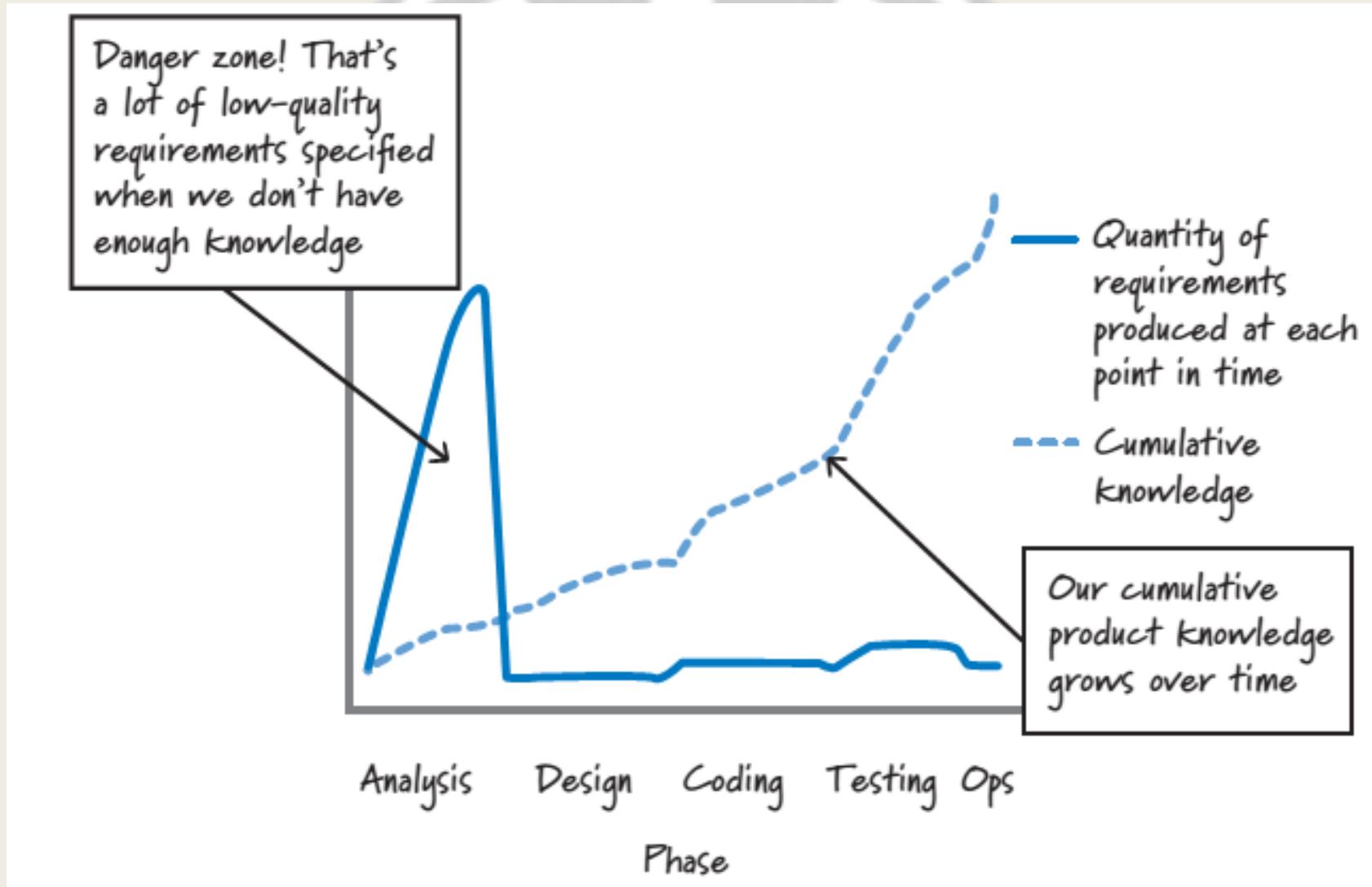
# Accept That You Can't Get It Right Up Front(II)

- In Scrum, we **acknowledge** that we **can't get all of the requirements** or the **plans** **right up front**.
- In fact, we believe that **trying to do so** could **be dangerous** because we are likely **missing important knowledge**, leading to the creation of a **large quantity of low-quality requirement**.

در اسکرام، ما تصدیق میکنیم که نمیتوانیم همه الزامات یا برنامهها را از قبل دریافت کنیم.  
در واقع، ما معتقدیم که تلاش برای انجام این کار میتواند خطرناک باشد، زیرا احتمالاً دانش مهمی را از دست میدهیم،  
که منجر به ایجاد مقدار زیادی از نیازهای با کیفیت پایین میشود.

# Plan-driven requirements acquisition relative to product knowledge

کسب الزامات مبتنی بر برنامه نسبت به دانش محصول



# Accept That You Can't Get It Right Up Front(IV)

با اسکرام، ما هنوز برخی از الزامات و برنامهها را از قبل تولید میکنیم، اما به اندازه کافی، و با این فرض که جزئیات آن نیازمندیها و برنامهها را با کسب اطلاعات بیشتر در مورد محصولی که در حال ساخت هستیم، تکمیل میکنیم. به هر حال، حتی اگر فکر کنیم که چه چیزی را بسازیم و چگونه کار را برای ساختن آن پیش‌بیش سازماندهی کنیم، به محض اینکه محصولات اولیه افزایشی خود را تحت تأثیر محیطی قرار دهیم، متوجه خواهیم شد که کجا اشتباہ می‌کنیم. آنها باید وجود داشته باشند. در آن نقطه، تمام واقعیت‌های ناخوشایند آنچه واقعاً مورد نیاز است، ما را به ایجاد تغییرات سوق می‌دهد.

- 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** we are building.
- After all, even if we think we're 100% certain about what to build and how to organize up front the work to build it, we will learn where we are wrong as soon as we subject our early incremental deliverables to the environment in which they must exist. At that point all of the inconvenient realities of what is really needed will drive us to make changes.

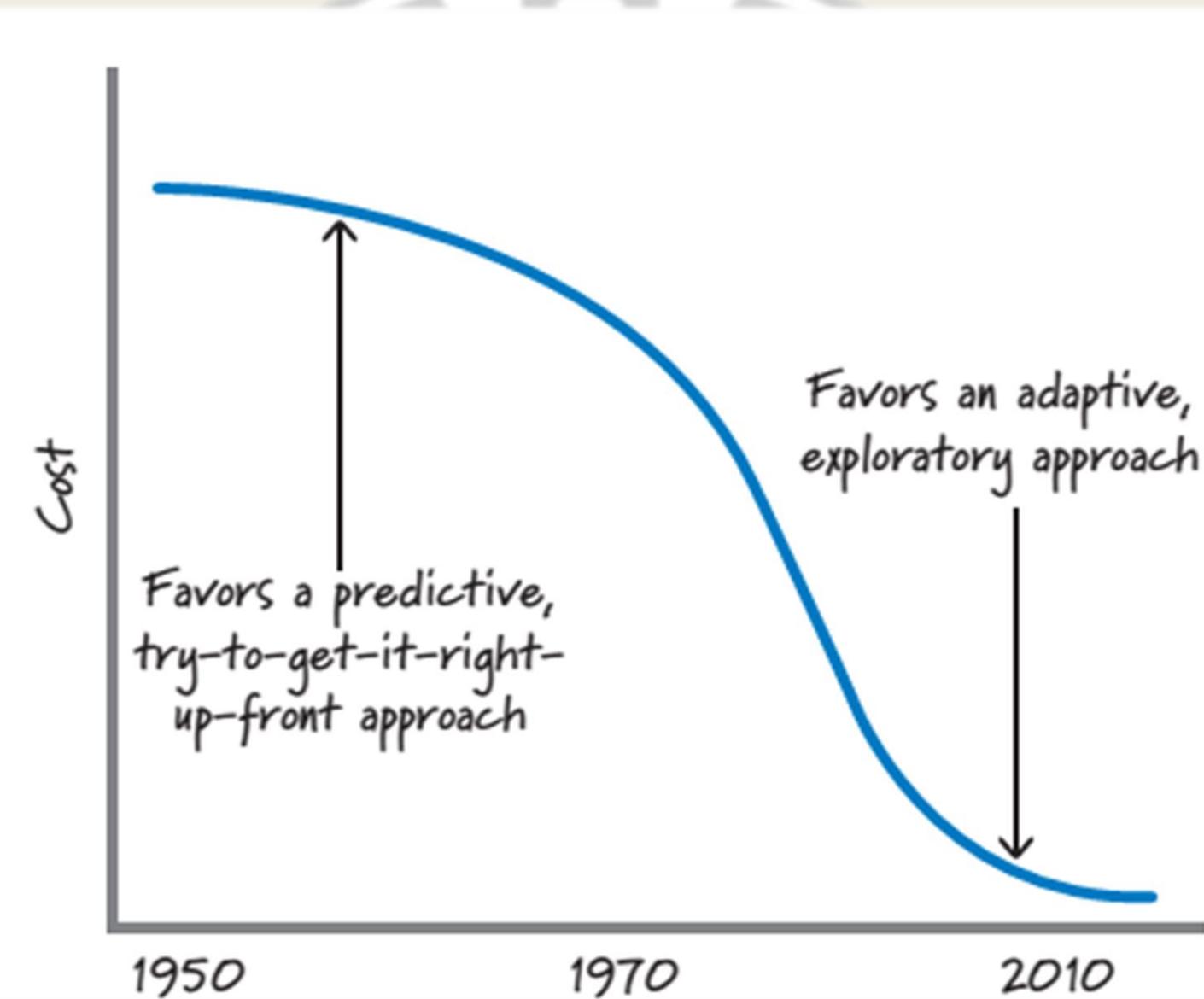
# Favor an Adaptive, Exploratory Approach (I)

از یک رویکرد تطبیقی و اکتشافی حمایت کنید (I)

- کاوش به موقعی اطلاق میشود که ما انتخاب میکنیم با انجام برخی فعالیتها، مانند ساختن نمونه اولیه، انجام مطالعه یا انجام آزمایش، دانش کسب کنیم. به عبارت دیگر، زمانی که با عدم قطعیت مواجه می شویم، اطلاعات را با کاوش خریداری می کنیم.
- فرآیندهای متوالی و مبتنی بر برنامه بر استفاده (یا بهره برداری) از آنچه در حال حاضر شناخته شده است و پیش بینی آنچه ناشناخته است تمرکز می کنند.
- اسکرام از رویکرد انطباقی تر، آزمون و خطاب مبتنی بر استفاده مناسب از کاوش استفاده می کند.
- ابزارها و فناوری های ما به طور قابل توجهی بر هزینه اکتشاف تأثیر می گذارد.

- **Exploration** refers to **times** when **we choose to gain knowledge by doing some activity**, such as **building a prototype**, **performing a study**, or **conducting an experiment**. In other words, when **faced with uncertainty**, we **buy information by exploring**.
- Plan-driven, sequential processes focus on **using** (or **exploiting**) **what is currently known** and **predicting what isn't known**.
- Scrum favors a **more adaptive, trial-and error approach** based on appropriate use of **exploration**.
- Our **tools** and **technologies** significantly **influence the cost of exploration**.

# Historical cost of exploration



# Favor an Adaptive, Exploratory Approach(III)

- Tools and technologies have gotten better and the cost of exploring has come way down.
- In fact, nowadays, it's often cheaper to adapt to user feedback based on building something fast than it is to invest in trying to get everything right up front.

ابزارها و فن آوری ها بهتر شده اند و هزینه کاوش بسیار پایین آمده است.  
■ در واقع، امروزه، تطبیق با بازخورد کاربران بر اساس ساختن سریع چیزی،  
اغلب ارزانتر از سرمایه‌گذاری در تلاش برای پیشبرد همه چیز است.

# Favor an Adaptive, Exploratory Approach(IV)

- In Scrum, if we have enough knowledge to make an informed, reasonable step forward with our solution, we advance.
- When faced with uncertainty, rather than trying to predict it away, we use low-cost exploration to buy relevant information that we can then use to make an informed, reasonable step forward with our solution.
- The feedback from our action will help us determine if and when we need further exploration.

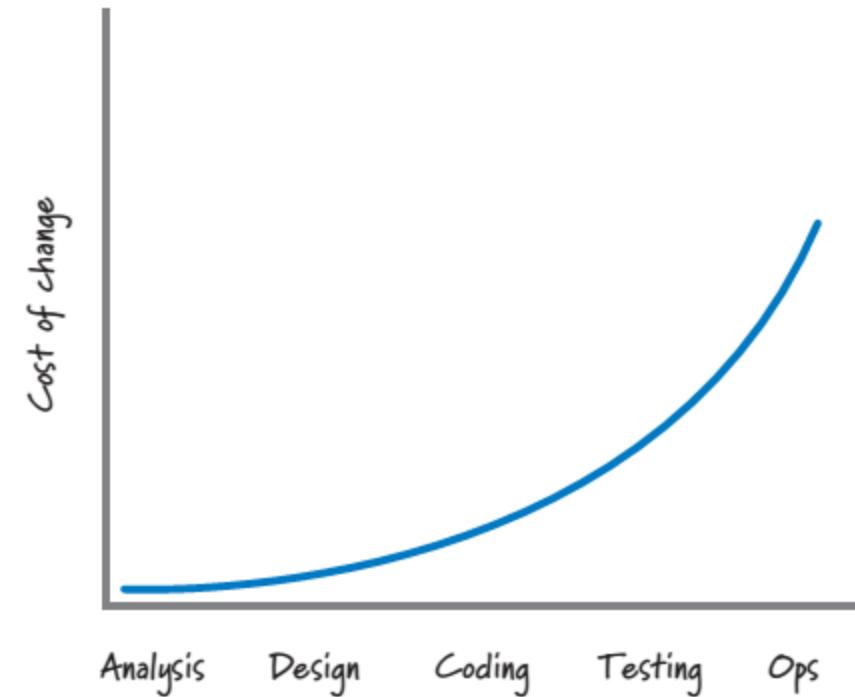
■ در اسکرام، اگر داشت کافی برای برداشتن یک گام آگاهانه و منطقی با راه حل خود داشته باشیم، پیشرفت می کنیم.  
■ هنگامی که با عدم قطعیت مواجه می شویم، به جای تلاش برای پیش بینی آن، از کاوش کم هزینه برای خرید اطلاعات مرتبط استفاده می کنیم که سپس می توانیم از آنها برای برداشتن یک گام آگاهانه و معقول به جلو با راه حل خود استفاده کنیم.  
■ بازخورد حاصل از اقدام ما به ما کمک می کند تا تعیین کنیم که آیا و چه زمانی به کاوش بیشتر نیاز داریم یا خیر.

# Embrace Change in an Economically Sensible Way(I)

پذیرش تغییر به رو شی معقول اقتصادی (I)  
هنگامی که از توسعه متوالی استفاده می شود، همانطور که همه ما آموخته ایم، تغییرات به طور قابل توجهی دیرتر از زمان اولیه گران تر است.

■ برای جلوگیری از تغییرات دیرهنگام، فرآیندهای متوالی به دنبال کنترل دقیق و به حداقل رساندن هرگونه الزامات یا طراحی های در حال تغییر با بهبود دقت پیش بینی ها در مورد آنچه که سیستم باید انجام دهد یا اینکه چگونه قرار است آن را انجام دهد، هستند.

- When using sequential development, change, as we have all learned, is substantially more expensive late than it is early on.
- To avoid late changes, sequential processes 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.

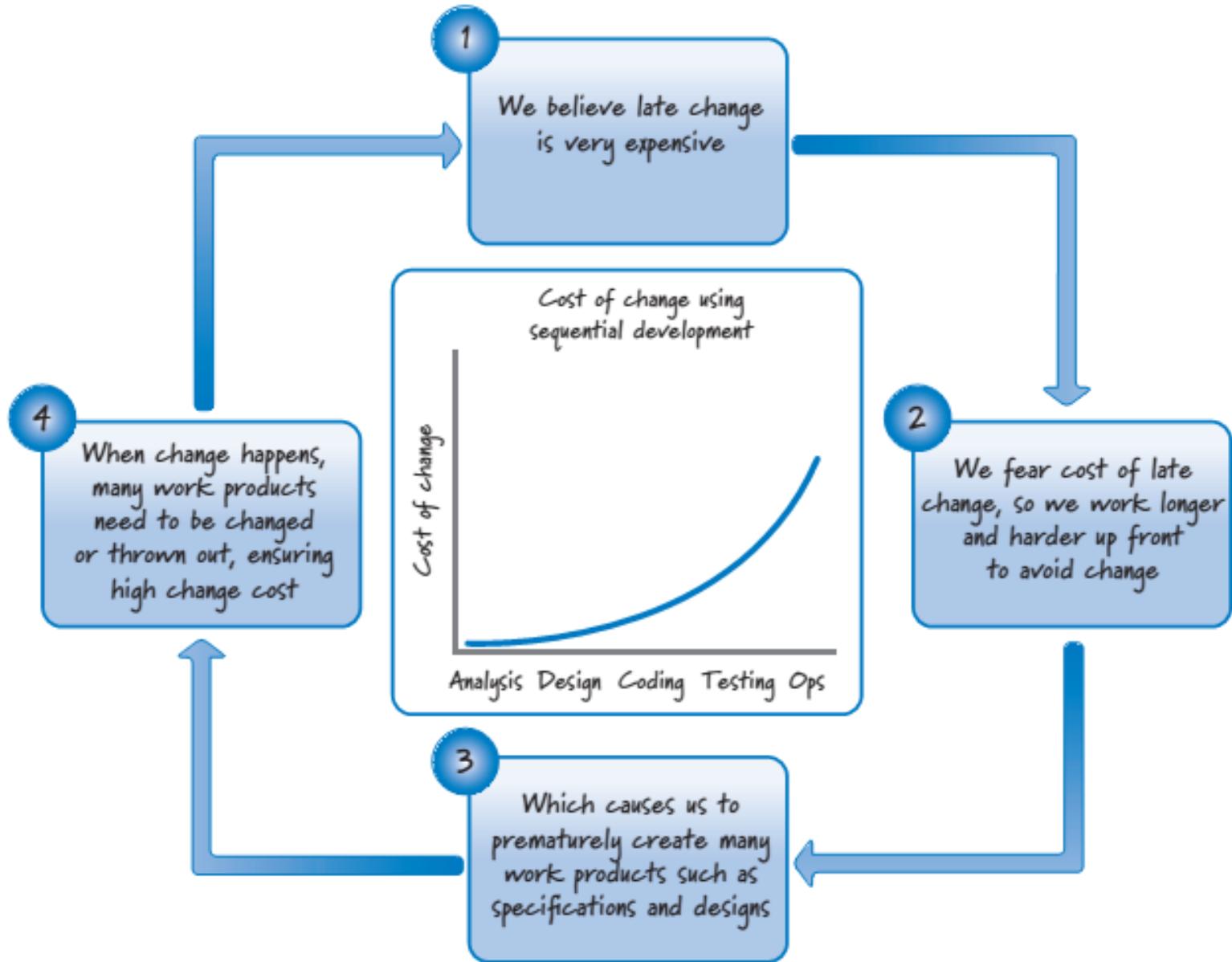


# Embrace Change in an Economically Sensible Way(II)

متأسفانه، پیش بینی بیش از حد در مراحل اولیه فعالیت اغلب اثر معکوس دارد.

- اولاً، میل به حذف تغییرات گران قیمت ما را وادار می کند که در هر مرحله بیش از حد سرمایه گذاری کنیم—انجام کار بیش از حد لازم و عملی.
- دوم، قبل از اینکه این مفروضات را با بازخورد سهامداران خود بر اساس دارایی های کاری خود تأیید کنیم، مجبور هستیم بر اساس مفروضات مهم در مراحل اولیه تصمیم گیری کنیم. در نتیجه، بر اساس این مفروضات، موجودی بزرگی از محصولات کاری تولید می کنیم. بعداً، این موجودی احتمالاً باید تصحیح یا کنار گذاشته شود، زیرا ما مفروضات خود را تأیید می کنیم (یا باطل می کنیم)، یا تغییر رخ می دهد.

- Unfortunately, being **excessively predictive** in **early-activity phases** often has the **opposite effect**.
- First, the **desire to eliminate expensive change** forces us to **overinvest** in each **phase**—doing **more work than is necessary** and **practical**.
- Second, we're **forced** to make **decisions** based on **important assumptions** **early in the process**, before we have **validated** these **assumptions** with feedback from our stakeholders based on our working assets. As a result, we **produce** a large **inventory** of **work products** **based** on **these assumptions**. Later, this **inventory** will likely **have to be corrected** or **discarded** as we **validate** (or **invalidate**) our **assumptions**, or **change happens**.



# Embrace Change in an Economically Sensible Way(IV)

در اسکرام، ما فرض می کنیم که تغییر یک هنجار است.  
ما معتقدیم که نمیتوانیم با کار طولانیتر و سختتر از قبل، عدم اطمینان ذاتی را که در طول توسعه محصول وجود دارد، پیشیبینی کنیم.

بنابراین، ما باید برای پذیرش تغییر آماده باشیم. و هنگامی که این تغییر رخ می دهد، ما می خواهیم که اقتصاد جذاب تر از توسعه سنتی باشد، حتی زمانی که تغییر بعداً در تلاش توسعه محصول اتفاق می افتد.

- In Scrum, we assume that change is the norm.
- We believe that we can't predict away the inherent uncertainty that exists during product development by working longer and harder up front.
- Thus, we must be prepared to embrace change. And when that change occurs, we want the economics to be more appealing than with traditional development, even when the change happens later in the product development effort.

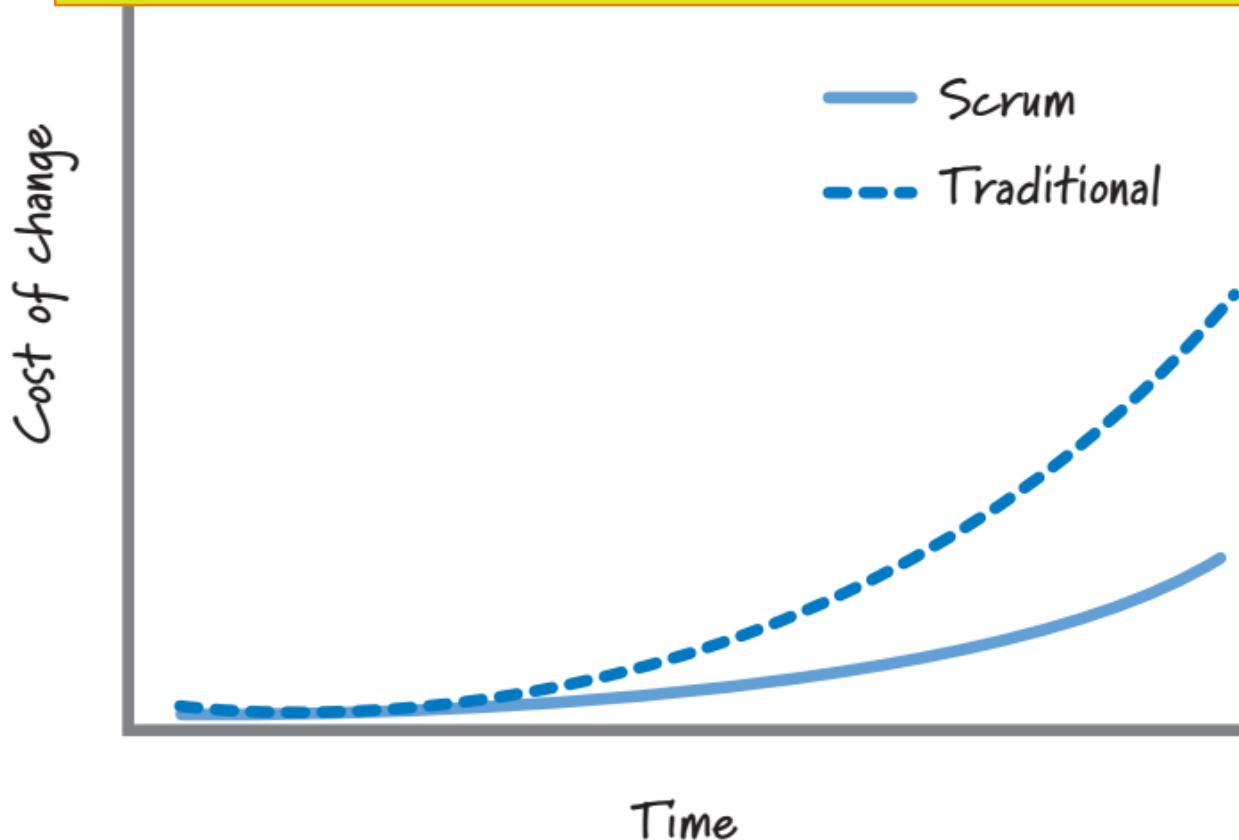
# Embrace Change in an Economically Sensible Way(V)

- Our goal, as possible change.
- We can process using Scrum projects.

Our goal, therefore, is to keep the cost-of-change curve flat for as long as possible—making it economically sensible to embrace even late change.

We 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.

for as long  
e even late



بنابراین، هدف ما این است که منحنی هزینه تغییر اتا زمانی که ممکن است صاف نگه داریم – که ذیرش تغییرات دیرهنگام از نظر اقتصادی منطقی باشد.

ما می توانیم با مدیریت میزان کار در فرآیند و جریان آن کار به آن هدف دست یابیم تا هزینه تغییر در هنگام استفاده از اسکرام کمتر از پروژه های متوالی تحت تأثیر زمان قرار گیرد.

# Embrace Change in an Economically Sensible Way(VI)

پذیرش تغییر به روشهای معقول اقتصادی (VI)

- صرف نظر از کدام رویکرد توسعه محصول استفاده می کنیم، می خواهیم رابطه زیر درست باشد.
  - یک تغییر کوچک در نیازمندیها باید تغییر نسبتاً کوچکی در اجرا و در نتیجه در هزینه ایجاد کند (بدهیهی است که انتظار داریم تغییر بزرگتر هزینه بیشتری داشته باشد).
  - یکی دیگر از ویژگی های مطلوب که ما می خواهیم بدون توجه به زمانی که درخواست تغییر انجام می شود، درست باشد.

- Regardless of which product development approach we use, we want the following relationship to be true.
  - A small change in requirements should yield a proportionally small change in implementation and therefore in cost (obviously we would expect a larger change to cost more).
- Another desirable property that we want it to be true regardless of when the change request is made.

# Embrace Change in an Economically Sensible Way(VII)

با اسکرام، ما بسیاری از محصولات کاری (مانند الزامات دقیق، طرحها و موارد آزمایشی) را بهموقع تولید میکنیم و از ایجاد مصنوعات غیرضروری اجتناب میکنیم.

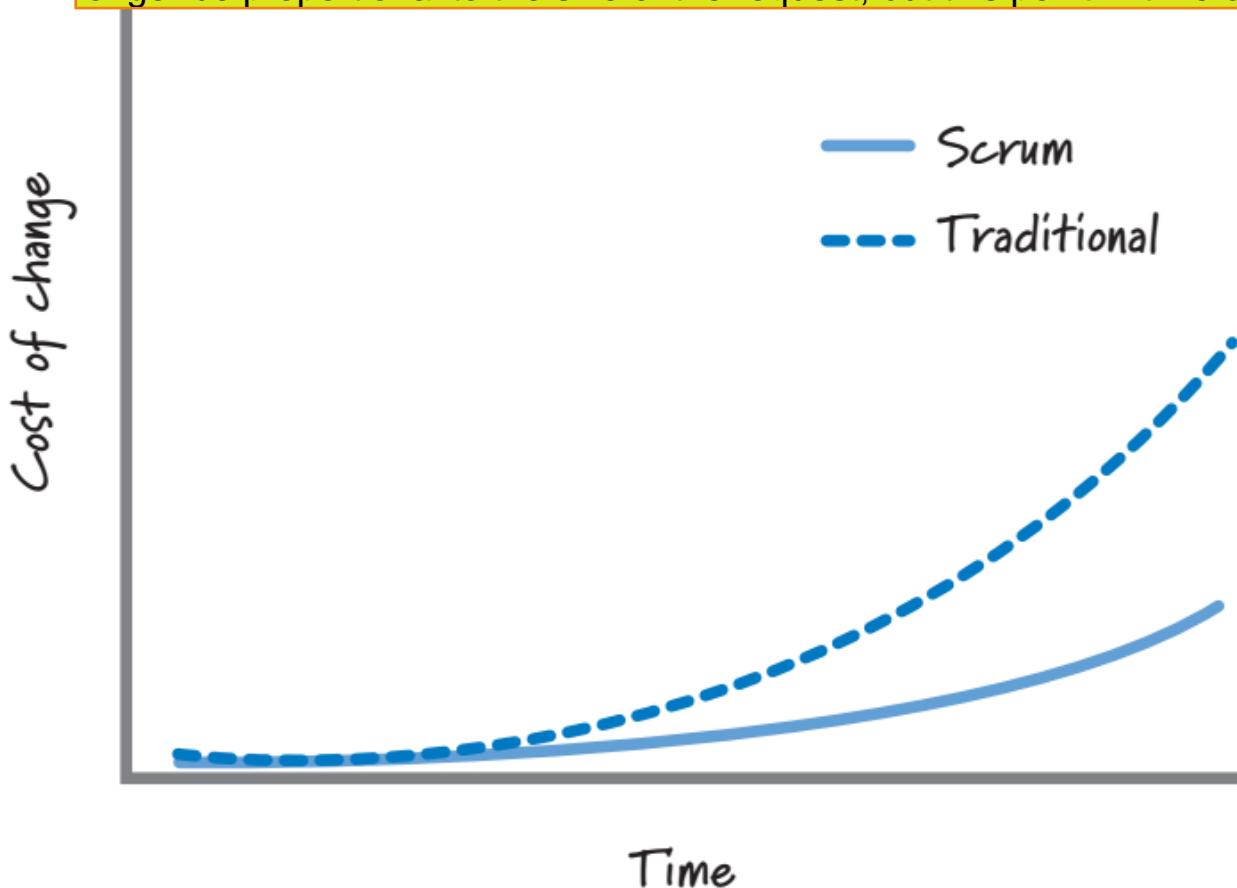
در نتیجه، زمانی که تغییری ایجاد میشود، معمولاً مصنوعات بسیار کمتری وجود دارد یا تصمیمهای محدودکننده بر اساس فرضیاتی که ممکن است کنار گذاشته شوند یا دوباره کار شوند، وجود دارد، بنابراین هزینه متناسبتر با اندازه تغییر درخواستی حفظ میشود.

- With Scrum, we produce many work products (such as detailed requirements, designs, and test cases) in a just-in-time fashion, avoiding the creation of potentially unnecessary artifacts.
- As a result, when a change is made, there are typically far fewer artifacts or constraining decisions based on assumptions that might be discarded or reworked, thus keeping the cost more proportional to the size of the requested change.

# Embrace Change in an Economically Sensible Way(VIII)

- Using sequential development, push for a change inflection point

- 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 occurs later.



Using sequential development, the early creation of artifacts and push for premature decision making ultimately mean that the cost of a change rises rapidly over time as inventory grows. This causes the inflection point on the traditional curve to occur early.

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 occurs later.

Artifacts and

با استفاده از توسعه متوالی، ایجاد اولیه صنوعات و فشار برای تصمیم گیری زودهنگام در نهایت به این معنی است که هزینه یک تغییر به سرعت در طول زمان با افزایش موجودی افزایش می یابد. این باعث می شود که نقطه عطف در منحنی سنتی زودتر رخ دهد.

هنگام توسعه با اسکرام، زمانی فرا می رسد که هزینه تغییر دیگر متناسب با اندازه درخواست نخواهد بود، اما این نقطه زمانی دیرتر اتفاق می افتد.

# Balance Predictive Up-Front Work with Adaptive Just-in-Time Work(I)

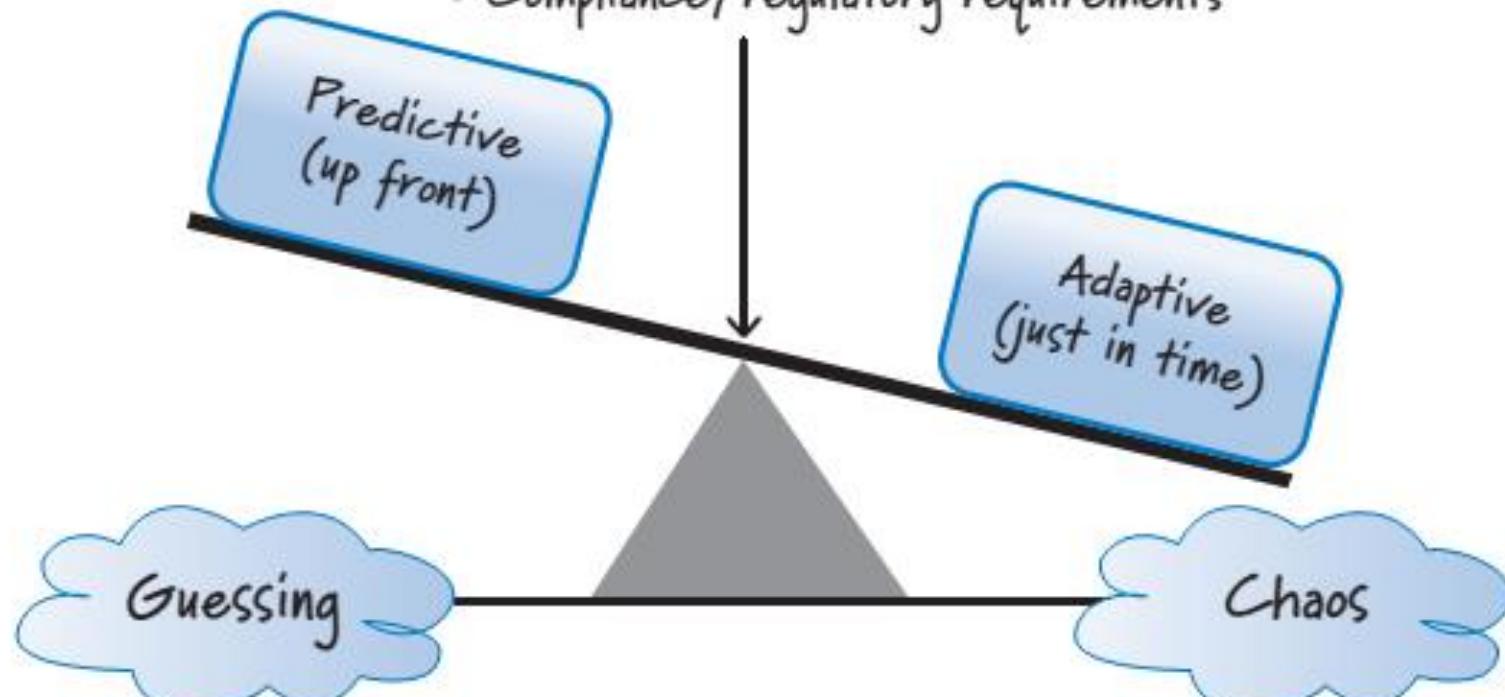
- A fundamental belief of plan-driven development is that detailed up-front requirements and planning are critical and should be completed before moving on to later stages.
- In Scrum, we believe that up-front work should be helpful without being excessive. With Scrum, we acknowledge that it is not possible to get requirements and plans precisely right up front.
- Scrum is about finding balance—balance between predictive up-front work and adaptive just-in-time work.

یک باور اساسی در توسعه برنامه محور این است که الزامات و برنامه ریزی دقیق اولیه ضروری هستند و باید قبل از رفتن به مراحل بعدی تکمیل شوند.

در اسکرام، ما معتقدیم که کار اولیه باید بدون زیاده روی مفید باشد. با اسکرام، ما تصدیق می کنیم که دریافت نیازها و برنامه ها دقیقاً از قبل امکان پذیر نیست.

اسکرام در مورد یافتن تعادل است - تعادل بین کار پیشگویانه و کار به موقع تطبیقی.

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



هنگام توسعه یک محصول، نقطه تعادل باید به روشی معقول اقتصادی تنظیم شود تا میزان سازگاری مداوم بر اساس بازخورد سریع به حداکثر برسد و میزان پیشبینی اولیه به حداقل برسد، در حالی که همچنان اهداف انطباق، نظارتی و/یا شرکتی برآورده میشود.

# Balance Prediction

## Adaptive Just-in-

اینکه دقیقاً چگونه این تعادل حاصل می شود تا حدی به نوع محصول ساخته شده، درجه عدم قطعیتی که هم در مورد آنچه می خواهیم بسازیم (عدم قطعیت) و چگونه می خواهیم آن را ایجاد کنیم (به معنی عدم اطمینان) و محدودیت ها بستگی دارد. بر روی توسعه قرار داده شده است.

- When developing a product, 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.
- Exactly how that 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 (end uncertainty) and how we want to build it (means uncertainty), and the constraints placed on the development.

# Balance Predictive Adaptive Just-in

یش بینی بیش از حد ما را ملزم می کند که در حضور عدم قطعیت زیاد، مفروضات زیادی داشته باشیم. سازگاری بیش از حد می تواند باعث شود که ما در یک وضعیت تغییر مداوم زندگی کنیم و باعث شود کار ما ناکارآمد و بی نظم باشد.

برای توسعه سریع محصولات نوآورانه، ما باید در فضایی کار کنیم که سازگاری با پیشینی کافی برای جلوگیری از فرو رفتن در هرج و مرچ، متعادل شود.  
چارچوب اسکرام در این نقطه تعادل نظم و هرج و مرچ به خوبی عمل می کند.

- 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**.

# Agile Principles(IV)

Dr. Elham Mahmoudzadeh

Isfahan University of Technology

[mahmoudzadeh@iut.ac.ir](mailto:mahmoudzadeh@iut.ac.ir)

2020

# Validated Learning

وقتی دانشی به دست می آوریم که فرضی را که ما انجام داده ایم تأیید یا رد می کند.  
سه اصل چاک مریبوط به این موضوع.  
مفروضات مهم را سریع تأیید کنید.  
از چندین حلقه یادگیری همزمان استفاده کنید.  
گردش کار را برای بازخورد سریع سازماندهی کنید.

- 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?

یک فرض، حدس یا باوری است که درست، واقعی یا قطعی فرض میشود، حتی اگر هیچ یادگیری معتبری برای دانستن درستی آن نداشته باشیم. توسعه برنامه محور نسبت به Scrum نسبت به مفروضات طولانی مدت تحمل بیشتری دارد. الزامات و برنامههای اولیه گستردهای را تولید کنید که احتمالاً بسیاری از مفروضات مهم را در خود جای میدهند، مفروضاتی که تا مرحله بسیار بعدی توسعه تأیید نمیشوند. نشان دهنده خطر توسعه قابل توجهی است.

- 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

در اسکرام سعی می کنیم تعداد مفروضات مهمی را که در هر زمان وجود دارد به حداقل برسانیم.

همچنین اجازه ندهید که مفروضات مهم بدون اعتبار برای مدت طولانی وجود داشته باشند.

ترکیب توسعه تکراری و افزایشی همراه با تمرکز بر اکتشاف کم هزینه می تواند برای اعتبارسنجی سریع مفروضات استفاده شود.

در نتیجه، اگر در هنگام استفاده از اسکرام، اساساً فرض بدی داشته باشیم، احتمالاً اشتباه خود را به سرعت کشف خواهیم کرد و فرصتی برای بهبودی از آن خواهیم داشت.

در توسعه متوالی مبتنی بر طرح، همان فرض بد اگر دیر تأیید شود ممکن است باعث شکست اساسی یا کلی تلاش توسعه شود.

- 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.

یادگیری وجود دارد که هنگام استفاده از توسعه متوالی رخ می‌دهد. با این حال، شکل مهمی از یادگیری تنها پس از ایجاد، ادغام و آزمایش ویژگیها اتفاق میافتد، به این معنی که یادگیری قابل توجهی در پایان تلاش رخ میدهد.

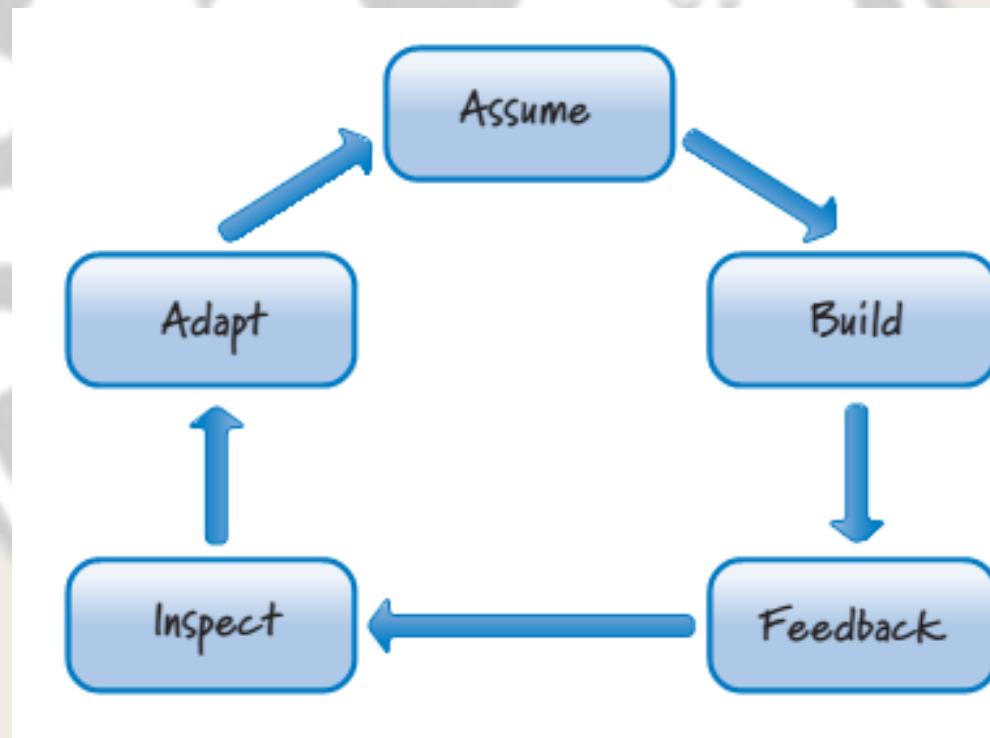
یادگیری دیرهنگام مزایای کمتری را به همراه دارد زیرا ممکن است زمان کافی برای اهرم یادگیری وجود نداشته باشد یا هزینه استفاده از آن ممکن است بسیار زیاد باشد.

# Leverage Mu Loops(II)

در اسکرام، ما درک می کنیم که یادگیری مداوم کلید موفقیت ما است.  
هنگام استفاده از اسکرام، حلقهای بازخورد را برای افزایش یادگیری شناسایی کرده و از آن استفاده میکنیم.  
یک الگوی تکرارشونده در این سبک از توسعه محصول به یک فرضیه بسازید (یا یک هدف تعیین کنید)،  
چیزی بسازید (انجام برخی فعالیت ها)،  
در مورد آنچه ساخته ایم بازخورد دریافت کنید،  
از این بازخورد برای بررسی آنچه انجام دادیم نسبت به آنچه که فرض میکردیم استفاده کنید.  
بر اساس آنچه آموختیم، با محصول، فرآیند و/یا باورهای خود سازگاری ایجاد کنیم.

- 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.

اسکرام از چندین حلقه یادگیری از پیش تعریف شده استفاده می کند.

به عنوان مثال، اسکرام روزانه یک حلقه روزانه و بررسی اسپرینت یک حلقه در سطح تکرار است.

# 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.

چارچوب Scrum همچنین به اندازه کافی منعطف است که بسیاری از حلقه های یادگیری دیگر را در بر می گیرد.

برای مثال، اگرچه توسط اسکرام مشخص نشده است، اما حلقه های باز خورد تمرین فنی، مانند برنامه نویسی جفتی (باز خورد در ثانیه) و توسعه آزمایش محور (باز خورد در چند دقیقه)، اغلب در توسعه اسکرام استفاده می شوند.

# 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.

تحمل مفروضات طولانی مدت همچنین فرآیندهای برنامه محور را نسبت به یادگیری دیرهنگام تحمل می کند، بنابراین بازخورد سریع یک تمرکز نیست.

اسکرام، ما برای بازخورد سریع تلاش میکنیم، زیرا برای کمک به کوتاه کردن سریعتر مسیرهای اشتباہ بسیار مهم است و برای کشف سریع و بهرهبرداری از فرصتهای حساس به زمان و ظهور حیاتی است.

# Organize Workflow for Fast Feedback(II)

در یک تلاش توسعه مبتنی بر برنامه، هر فعالیتی برنامه ریزی می شود که در زمان معینی بر اساس توالی فاز به خوبی تعریف شده انجام شود.

این رویکرد فرض می کند که فعالیت های قلی را می توان بدون بازخورد تولید شده توسط فعالیت های بعدی تکمیل کرد. در نتیجه، ممکن است یک دوره زمانی طولانی بین انجام کاری و گرفتن بازخورد در مورد کاری که انجام داده ایم وجود داشته باشد (از این رو حلقه یادگیری را می بینیم).

- 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.

ما در حال توسعه سه جزء به صورت موازی هستیم. در برخی مواقع این اجزا باید یکپارچه شده و قبل از اینکه محصولی قابل حمل داشته باشیم، آزمایش شوند. تا زمانی که سعی نکنیم ادغام را نجام دهیم، واقعاً نمی دانیم که آیا مؤلفه ها را به درستی توسعه داده ایم یا خیر. تلاش برای ادغام، بازخورد مهمی را در مورد کار توسعه مؤلفه ارائه می دهد.

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

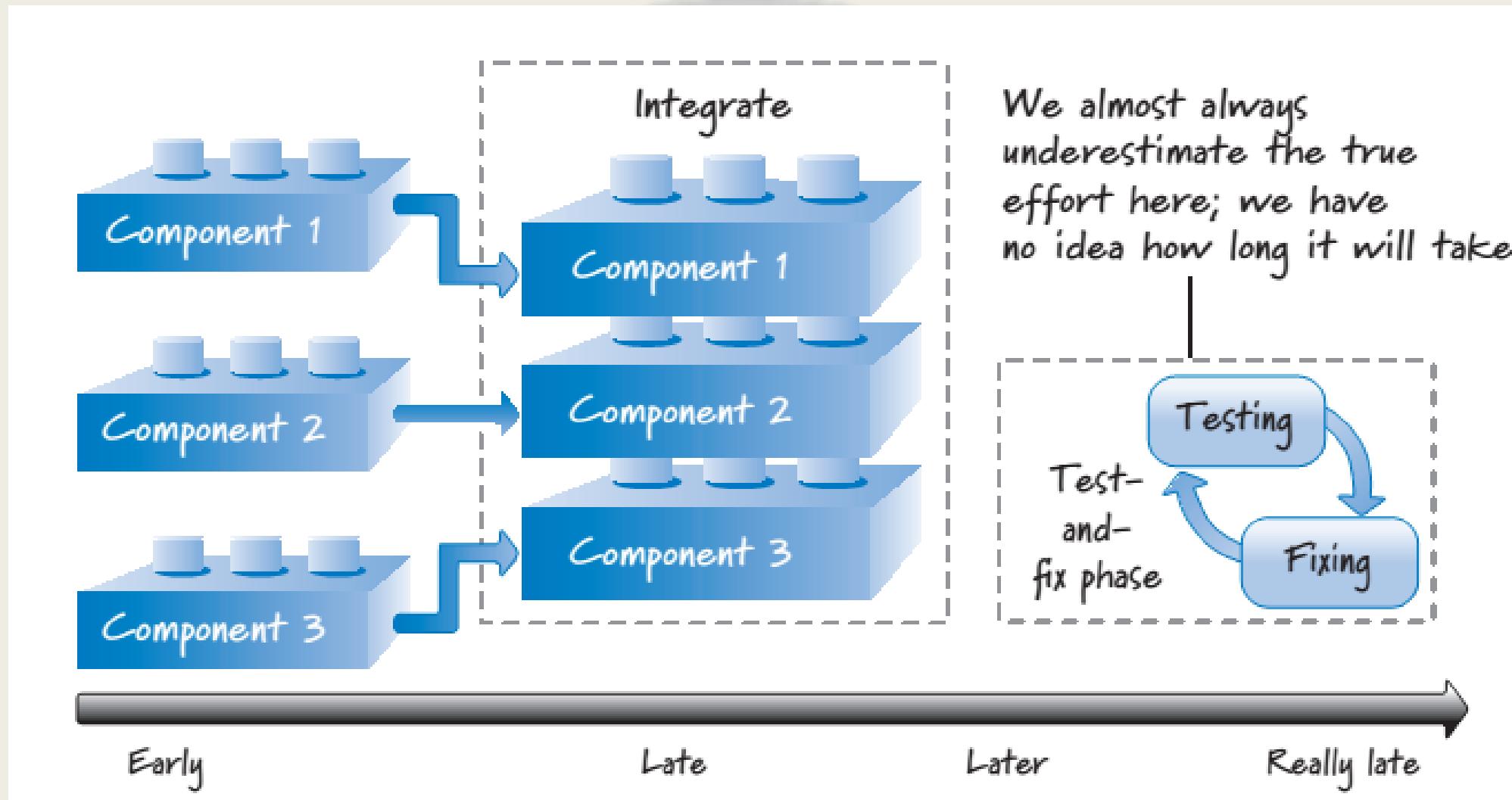
- Using sequential development, integration and testing wouldn't happen until the predetermined downstream phase, where many or all components would be integrated.
- Unfortunately, the idea that we can develop a bunch of components in parallel and then later, in an integration phase, smoothly bring them 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.

استفاده از توسعه متوالی، ادغام و آزمایش تا مرحله پایین دستی از پیش تعیین شده، جایی که بسیاری یا همه مؤلفه ها ادغام می شوند، انفاق نمی افتد.

متأسفانه، این ایده که ما میتوانیم مجموعهای از مؤلفهها را به صورت موازی توسعه دهیم و سپس بعداً در مرحله

یکپارچهسازی، به آرامی آنها را در یک کل منسجم گرد هم بیاوریم، بعيد است به نتیجه برسد.

در واقع، حتی با وجود اینترفیس‌های خوب که قبل از توسعه مؤلفهها تعریف شده‌اند، این احتمال وجود دارد که هنگام ادغام آنها مشکلی پیش بیاید.



# 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?

فعالیتهای بازخوری که مدت‌ها پس از توسعه اتفاق میافتد، دارای عوارض جانبی ناخواهشاندی هستند، مانند تبدیل ادغام به یک مرحله آزمایش و اصلاح بزرگ، زیرا اجزایی که بهطور مجزا از یکدیگر توسعه مییابند، اغلب به آرامی با هم ادغام نمی‌شوند.  
در این مرحله فقط می‌توان حدس زد که چقدر طول می‌کشد و چقدر هزینه برای رفع مشکل دارد؟

# 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**.

در اسکرام، جریان کار را به گونه ای سازماندهی می کنیم که از طریق حلقه یادگیری حرکت کند و در اسرع وقت به بازخورد برسیم. با انجام این کار، اطمینان حاصل می کنیم که فعالیت های بازخورد آفرین در مجاورت زمانی کار اصلی رخ می دهد.

بازخورد سریع مزایای اقتصادی برتری را به همراه دارد زیرا وقتی بازخورد را به تأخیر میاندازیم، خطاهای ترکیب میشوند و منجر به شکستهای بزرگتر میشوند.

# 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**.

هنگامی که اجزا را طراحی کردیم، فرضیات مهمی در مورد نحوه ادغام آنها داشتیم. بر اساس آن مفروضات، ما مسیر طراحی را ادامه دادیم. ما در این مرحله نمی‌دانیم که مسیر طراحی انتخاب شده درست است یا غلط. این فقط بهترین حدس ماست. با این حال، هنگامی که یک مسیر را انتخاب می‌کنیم، تصمیمات زیادی را می‌گیریم که بر اساس آن انتخاب است. هر چه مدت بیشتری منتظر اعتبار فرضیه طراحی اصلی باشیم، تعداد تصمیمات وابسته بیشتر می‌شود.

- 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.

اگر بعداً (از طریق بازخورد در مرحله ادغام) تشخیص دهیم که فرض اصلی اشتباه بوده است، یک آشفتگی بزرگ و مرکب در دستان خود خواهیم داشت.

مانند ترتیب تصمیمات بد زیادی خواهیم داشت که باید دوباره کار شوند. ما همچنین باید پس از گذشت زمان زیادی این کار را انجام دهیم.

از آنجایی که خاطرات افراد محو شده است، آنها زمان خود را صرف می کنند تا به کارهای قبلی خود سرعت ببخشند.

# 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.

هنگامی که هزینه کل بازنگری تصمیمات بالقوه بد وابسته و هزینه تأخیر برای محصول را در نظر بگیریم، مزایای اقتصادی بازخورد سریع بسیار قانع کننده است.

بازخورد سریع حلقه یادگیری را به سرعت می بندد و به ما این امکان را می دهد که مسیرهای توسعه بد را قبل از اینکه آسیب اقتصادی جدی وارد کنند کوتاه کنیم.

# Agile Principles(V)

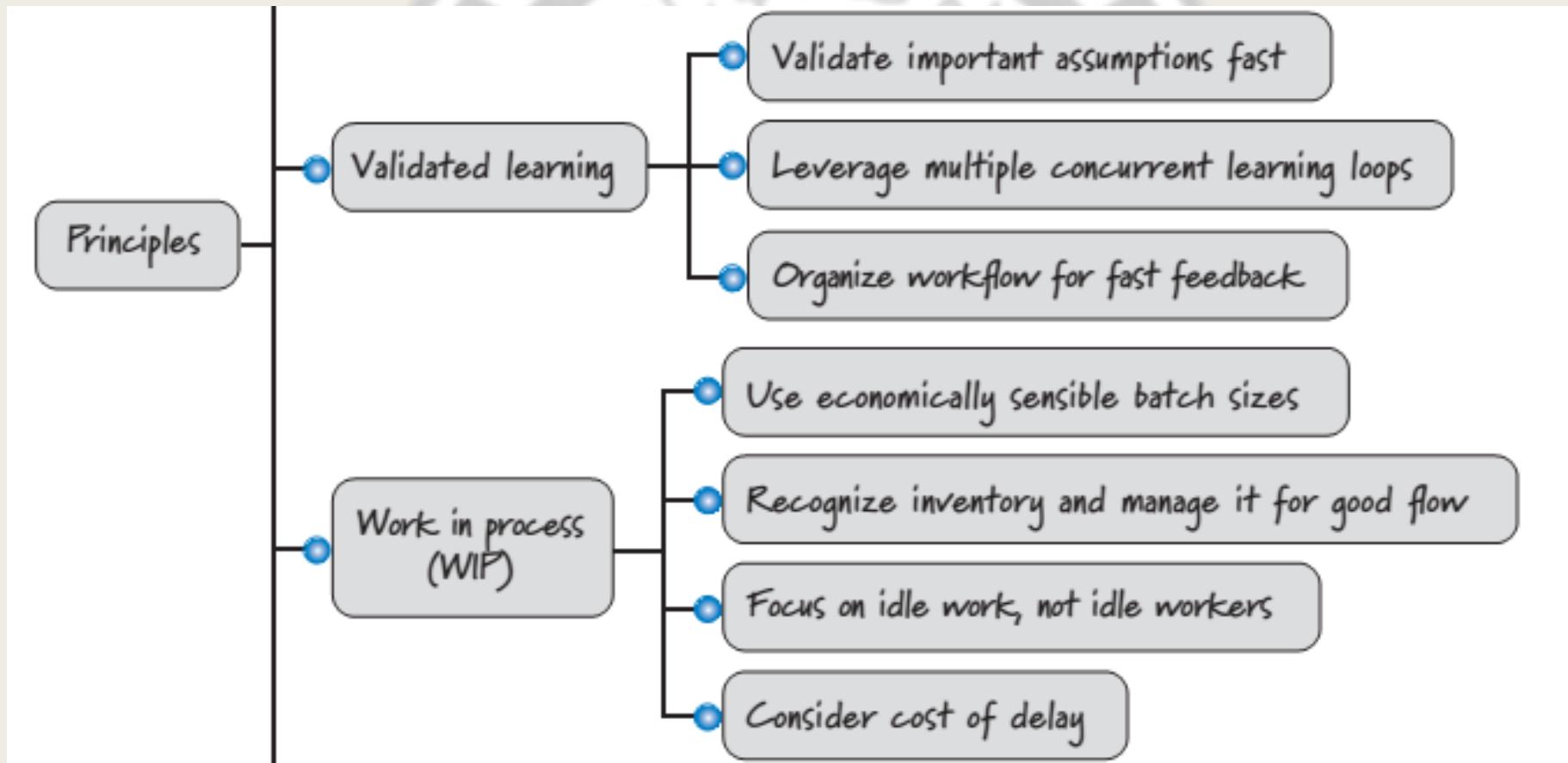
Dr. Elham Mahmoudzadeh

Isfahan University of Technology

[mahmoudzadeh@iut.ac.ir](mailto:mahmoudzadeh@iut.ac.ir)

2020

# Categorization of principles (Middle)



به کاری اطلاق می شود که شروع شده اما هنوز تمام نشده است. در طول توسعه محصول WIP باید شناسایی و به درستی مدیریت شود.

چهار اصل چاپک مربوط به این موضوع.

از اندازه های دسته ای معقول اقتصادی استفاده کنید.

موجودی را بشناسید و آن را برای جریان خوب مدیریت کنید.

روی کار بیکار تمرکز کنید نه کارگران بیکار.

هزینه ناخیر را در نظر بگیرید.

# Work in Process (WIP)

- Refers to the **work** that has **been started but not yet finished**. During product development **WIP** must be **recognized** and **properly managed**.
- Four **agile principles** related to this topic.
  1. Use **economically sensible batch sizes**.
  2. Recognize **inventory** and **manage it for good flow**.
  3. Focus on **idle work, not idle workers**.
  4. Consider **cost of delay**.

# Use Economically Sensible Batch Sizes(I)

یکی دیگر از باورهای اصلی نهفته در فرآیندهای توسعه متوالی و مبتنی بر برنامه این است که ترجیح داده می شود همه یک نوع کار را دسته بندی کرد و آن را در یک فاز انجام داد.

به این به عنوان رویکرد همه چیز قبل از هر گونه اشاره کنید، جایی که ما تمام (یا تقریباً همه) یک فعالیت را قبل از شروع فعالیت بعدی تکمیل می کنیم. بیاید بگوییم که ما تمام الزامات را در مرحله تجزیه و تحلیل ایجاد می کنیم. بعد، دسته ای از نیازمندی ها را به مرحله طراحی منتقل می کنیم. از آنجا که ما مجموعه کاملی از نیازمندی ها را ایجاد کردیم، اندازه دسته ما در این مثال 100٪ است.

- Another **core belief** underlying **plan-driven, sequential development** processes is that it is **preferable to batch up** all of one type of work and **perform it in a single phase**.
- Refer to this as the **all-before-any approach**, where we complete all (or substantially all) of one activity before starting the next.
- Let's say we create **all of the requirements** during the analysis phase. Next, we **move the batch of requirements** into the **design phase**. Because we generated the complete **set of requirements**, our **batch size** in this example is **100%**.

# Use Economically Sensible Batch Sizes(II)

- The **all-before-any approach** is, in part, a **consequence** of believing that the **old manufacturing principle** of economies of **scale** applies to **product development**.
- This principle states that the **cost of producing a unit** will **go down** as we **increase the number of units** (the batch size) that are produced.

رویکرد همه چیز قبل از هر چیزی، تا حدی نتیجه این باور است که اصل تولید قدیمی صرفه جویی در مقایس برای توسعه محسوب اعمال می شود.  
این اصل بیان می کند که با افزایش تعداد واحدهای تولید شده (اندازه دسته ای) هزینه تولید یک واحد کاهش می یابد.

# Use Economically Sensible Batch Sizes(III)

- In Scrum, we accept that although economies-of-scale thinking has been a bedrock principle in manufacturing, applying it dogmatically to product development will cause significant economic harm.
- Working in smaller batches during product development has many benefits.

در اسکرام، ما می پذیریم که اگرچه تفکر صرفه جویی در مقیاس یک اصل اساسی در تولید بوده است، اما به کار بردن آن به طور جزئی در توسعه محصول آسیب اقتصادی قابل توجهی را به همراه خواهد داشت.

کار در دسته های کوچکتر در طول توسعه محصول مزایای بسیاری دارد.

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).
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.

# Use Economically Sensible Batch Sizes(V)

- If small batches are better than large batches, shouldn't we just use a batch size of one, meaning that we work on only one requirement at a time and flow it through all of the activities until it is done and ready for a customer?
- A batch size of one might be appropriate in some cases, but assuming that "one" is the goal might sub-optimize the flow and our overall economics.

اگر دستههای کوچک بهتر از دستههای بزرگ هستند، آیا نباید از اندازه یک دسته استفاده کنیم، به این معنی که در هر زمان فقط روی یک نیاز کار میکنیم و آن را در تمام فعالیتها جریان میدهیم تا زمانی که انجام شود و برای مشتری آماده شود؟  
اندازه یک دسته ممکن است در برخی موارد مناسب باشد، اما با فرض اینکه "یک" هدف است، ممکن است جریان و اقتصاد کلی ما را بهینه کند.

# Recognize Inventory and Manage It for Good Flow(I)

- There is, one lesson that manufacturing has learned that we should apply to product development and yet often do not. That lesson has to do with the high cost of inventory, also known as work in process or WIP.
- Manufacturers are acutely aware of their inventories and the financial implications of those inventories.
- Inventory quickly starts to pile up on the floor, waiting to be processed. Not only is factory inventory physically visible; It is also financially visible.

یک درسی وجود دارد که تولید آموخته است که ما باید در توسعه محصول به کار ببریم اما اغلب این کار را نمی کنیم. این درس مربوط به هزینه بالای موجودی است که به عنوان کار در فرآیند یا WIP نیز شناخته می شود.

تولیدکنندگان به شدت از موجودی های خود و پیامدهای مالی آن موجودی ها آگاه هستند.

موجودی به سرعت روی زمین انبانشه می شود و در انتظار پردازش است. نه تنها موجودی کارخانه از نظر فیزیکی قابل مشاهده است. از نظر مالی نیز قابل مشاهده است.

# Recognize Invent Good Flow (II)

اما، چه اتفاقی میافتد اگر یک کامیون قطعات بخریم و سپس طراحی محصول را تغییر دهیم؟ با همه آن قسمت‌ها چه کنیم؟

- But, what happens if we purchase a truckload of parts, and then change the design of the product? What do we do with all of those parts?
- Maybe we rework the parts so that they fit into the new design.
- Or worse, maybe we discard the parts because they can no longer be used.
- Or, to avoid incurring waste on the parts we already purchased, are we going to not change our design (even though doing so would be the correct design choice) so we can use those parts—at the risk of producing a less satisfying product?

شاید قطعات را دوباره کار کنیم تا در طرح جدید جا بیفتد.  
یا بدتر، شاید قطعات را دور می‌اندازیم زیرا دیگر قابل استفاده نیستند.  
یا، برای جلوگیری از اتلاف قطعاتی که قبلًا خریداری کردهایم، آیا طراحی خود را تغییر نمیدهیم (حتی اگر انجام این کار انتخاب طراحی صحیحی است) تا بتوانیم از آن قطعات استفاده کنیم - با خطر تولید محصول کمتر رضایت‌بخش؟

# Recognize Inventory and Manage It for Good Flow(III)

- It's obvious that if we sit on **a lot of inventory** and then something **changes**, we experience one or more forms of **significant waste**.
- To **minimize risks**, competent manufacturers manage inventory in an **economically sensible way**.
- They **keep some inventory on hand** but use a **healthy dose of just-in-time inventory management**.

بديهي است که اگر روی موجودی زيادي بنشينيم و بعد چيزی تغيير کند، يک يا چند شکل از ضایعات قابل توجه را تجربه می کنیم.  
برای به حداقل رساندن خطرات، تولیدکنندگان ذيصلاح موجودی را به روشی معقول اقتصادی مدیریت می کنند.  
آنها مقداری موجودی را در دسترس نگه می دارند، اما از دوز سالمی از مدیریت موجودی به موقع استفاده می کنند.

# Recognize Inventory and Manage It for Good Flow(IV)

- In product development we deal with **knowledge assets** that **aren't physically visible** in the same way as parts on the factory floor.
- Knowledge assets are **far less intrusive**, such as **code on a disk**, a **document in a file cabinet**, or a **visual board on the wall**.
- Inventory in product development is also typically **not financially visible**.

در توسعه محصول، ما با دارایی های دانشی سروکار داریم که از نظر فیزیکی مانند قطعات موجود در کف کارخانه قابل مشاهده نیستند.

دارایی های دانش به مراتب کمتر مداخله گر هستند، مانند کد روی دیسک، یک سند در یک کابینت فایل، یا یک تابلوی بصری روی دیوار.

موجودی در توسعه محصول نیز معمولاً از نظر مالی قابل مشاهده نیست.

# Recognize Inventory Good Flow(V)

موجودی (WIP) یک متغیر حیاتی است که باید در طول توسعه محصول مدیریت شود و رویکردهای سنتی توسعه محصول بر مدیریت آن تمرکز ندارند.

با تنظیم اندازه دسته برای بسیار بزرگ (غلب 100٪)، توسعه سنتی در واقع به نفع ایجاد مقادیر زیادی از موجودی است.

یک پیامد مهم داشتن WIP زیاد در توسعه محصول این است که به طور قابل توجهی بر هزینه تغییر تأثیر می‌گذارد.

- Inventory (WIP) is a critical variable to be managed during product development, and the traditional approaches to product development don't focus on managing it.
- By setting the batch size to be quite large (frequently 100%), traditional development actually favors the creation of large amounts of inventory.
- An important consequence of having a lot of WIP in product development is that it significantly affects the cost-of-change.

# Recognize Inventory Good Flow(VI)

اگرچه اگر میخواهیم توسعه را شروع کنیم به برخی الزامات نیاز داریم، اما نیازی به داشتن همه لزامات نداریم. اگر نیازمندی های زیادی داشته باشیم، احتمالاً در صورت تغییر نیازمندی ها، ضایعات موجودی را تجربه خواهیم کرد.

ز طرف دیگر، اگر موجودی مورد نیاز کافی نداشته باشیم، جریان سریع کار را مختل می کنیم که این نیز نوعی اتلاف است.

- Although we need some requirements if we are going to start development, we don't need to have all of the requirements. If we have too many requirements, we will likely experience inventory waste when requirements change.
- On the other hand, if we don't have enough requirements inventory, we will disrupt the fast flow of work, which is also a form of waste.

# Recognize Inventory and Manage It for Good Flow(VII)

- In Scrum, our goal is to find the proper balance between just enough inventory and too much inventory.
- It is important to realize that requirements are just one form of inventory that exists in product development.
- There are many different places and times during product development where we have WIP. We need to proactively identify and manage those as well.

در اسکرام، هدف ما یافتن تعادل مناسب بین موجودی کافی و موجودی بیش از حد است.  
درک این نکته مهم است که الزامات فقط یک شکل از موجودی است که در توسعه محصول وجود دارد.  
مکان ها و زمان های مختلفی در طول توسعه محصول وجود دارد که در آن ما WIP داریم. ما باید به طور فعال آنها را شناسایی و مدیریت کنیم.

# Focus on Idle Work, Not Idle Workers(I)

- **Idle work** is **work that we want to do** (such as **building** or **testing** something) but **can't do** because **something is preventing us**.
- Perhaps we are **blocked waiting on another team** to do something, and until that team completes its work, we can't do ours. Or maybe we just have **so much work to do** that it can't all be done at once. In this case, some of the **work sits idle** until we become available to work on it.

کار بیکار کاری است که ما می خواهیم انجام دهیم (مانند ساختن یا آزمایش چیزی) اما نمی توانیم انجام دهیم زیرا چیزی مانع ما می شود.  
ساید ما در انتظار انجام کاری توسط تیم دیگری بمانیم و تا زمانی که آن تیم کار خود را کامل نکند، ما نمی توانیم کار خود را انجام دهیم. یا شاید آنقدر کار برای انجام دادن داریم که نمی توان همه آن ها را یکجا انجام داد. در این حالت، برخی از کارها بیکار می مانند تا زمانی که برای کار روی آن در دسترس قرار بگیریم.

# Focus on Idle Work, Not Idle Workers(II)

- **Idle workers**, are people who have available capacity to do more work because they are not currently 100% utilized.
- In Scrum, we believe that **idle work** is far more wasteful and economically damaging than idle workers.

کارگران بیکار، افرادی هستند که به دلیل عدم استفاده 100% از آنها، ظرفیت کافی برای انجام کارهای بیشتر را دارند.

در اسکرام، ما بر این باوریم که کار بیکار بسیار زیان آورتر و از نظر اقتصادی مضرتر از کارگران بیکار است.

# Focus on Idle Work, Not Idle Workers(III)

- Many product development organizations focus more on eliminating the waste of idle workers than on the waste of idle work.
- For example, in traditional thinking, if I hire you to be a tester, I expect you to spend 100% of your time testing. If you spend less than 100% of your time testing, I incur waste (you're idle when you could be testing).
- To avoid this problem, I will find you more testing work to do—perhaps by assigning you to multiple projects—to get your utilization up to 100%.

بسیاری از سازمان های توسعه محصول بیشتر بر روی حذف ضایعات کارگران بیکار تمرکز می کنند تا اتلاف کار بیکار. به عنوان مثال، در تفکر سنتی، اگر شما را به عنوان یک آزمایشگر استخدام کنم، از شما انتظار دارم که 100% وقت خود را صرف تست کنید. اگر کمتر از 100% وقت خود را صرف آزمایش کنید، من متحمل اتلاف می شوم (زمانی که می توانید آزمایش کنید، بیکار هستید). برای جلوگیری از این مشکل، کارهای آزمایشی بیشتری را برای شما پیدا خواهم کرد - شاید با اختصاص دادن شما به چندین پروژه - تا میزان استفاده شما تا 100% برسد.

# Focus on Idle Work, Not Idle Workers(IV)

- Unfortunately, this approach reduces one form of waste (idle-worker waste) while simultaneously increasing another form of waste (idle-work waste).
- Most of the time, the cost of the idle work is far greater than the cost of an idle worker.

متسفانه، این رویکرد یک شکل از ضایعات (ضایعات بیکار) را کاهش می دهد در حالی که به طور همزمان شکل دیگری از زباله را افزایش می دهد (ضایعات بیکار).

در بیشتر مواقع، هزینه کار بیکار به مراتب بیشتر از هزینه یک کارگر بیکار است.

# Focus On

برای نشان دادن موضوع، اجازه دهید استراتژی نگه داشتن کارگران-100% -مشغله را در مسابقه امدادی  $4 \times 100$  متر در المپیک اعمال کنیم.  
بر اساس استراتژی مشغول نگه داشتن آنها، این مسابقه بسیار ناکارآمد به نظر می‌رسد. من برای دویدن به مردم پول می‌دهم و به نظر می‌رسد که آنها فقط یک چهارم زمان می‌دونند. بقیه زمان آنها فقط در اطراف ایستاده اند.  
خب این درست نیست! من به آنها 100٪ حقوق می‌دهم، بنابراین می‌خواهم آنها 100٪ کار کنند.  
اگر وقتی با خود را به دوش نمیکشند، فقط از جایگاهها بالا و پایین میدونند یا شاید مسابقه دیگری را در یک مسیر مجاور اجرا کنند، چطور؟ به این ترتیب آنها 100٪ در دویدن مورد استفاده قرار خواهند گرفت.

- To illustrate the issue, let's apply the **keep-workers-100 % -busy strategy** to the  **$4 \times 100$ -meter relay race** at the Olympics.
- Based on the **keep-them-busy strategy**, this race seems **highly inefficient**. I pay people to run and they seem to be running only one-quarter of the time. The rest of the time they are just standing around.
- Well, that's not right! I pay them 100% salary so I want them to run 100% of the time.
- How about if when they're not carrying the baton, they just run up and down the stands or perhaps run another race on an adjacent track? That way they will be utilized 100% at running.

# Focus on Idle Work, Not Idle Workers(VI)

- You don't win the relay gold medal by keeping the runners 100% busy.  
You win the gold medal by getting the baton across the finish line first.  
So, the important takeaway is "Watch the baton, not the runners".

با مشغول نگه داشتن 100% دونده ها، مdal طلای رله را به دست نمی آورید. با زدن باتوم ابتدا از خط پایان، مdal طلا را به دست می آورید. بنابراین، نکته مهم این است که "مواظب باتوم باشید، نه دونده ها".

# Focus on Idle Work, Not Idle Workers(VII)

- In the context of product development, the baton sitting on the ground equates to **work that is ready to be performed** but is **blocked waiting for necessary resources**.
- You don't win the race (**deliver products**) when the **baton is on the ground**.

در زمینه توسعه محصول، باتوم نشستن روی زمین برابر با کاری است که آمده انجام است اما در انتظار منابع لازم مسدود شده است.

وقتی باتوم روی زمین است، در مسابقه (تحویل محصولات) برنده نمی شوید.

# Focus on Idle Work, Not Idle Workers(X)

- What happens if you **run your computer at 100%** (full processor and memory utilization)? It starts to **thrash** and **every job** on the computer **slows down**. In other words, the computer is working on more things and actually **gets less productive** work completed.
- Once you get into that state, it is very **difficult to get out of it** (you probably have to start killing jobs or reboot the machine).
- Your computer would be much **more efficient** if you **ran it at closer to 80% utilization**.

اگر رایانه خود را 100% اجرا کنید (استفاده از پردازنده و حافظه کامل) چه اتفاقی میافتد؟ شروع به خراب شدن می کند و هر کار روی رایانه کند می شود. به عبارت دیگر، رایانه روی چیزهای بیشتری کار می کند و در واقع کار کم ثمر تری را تکمیل می کند.

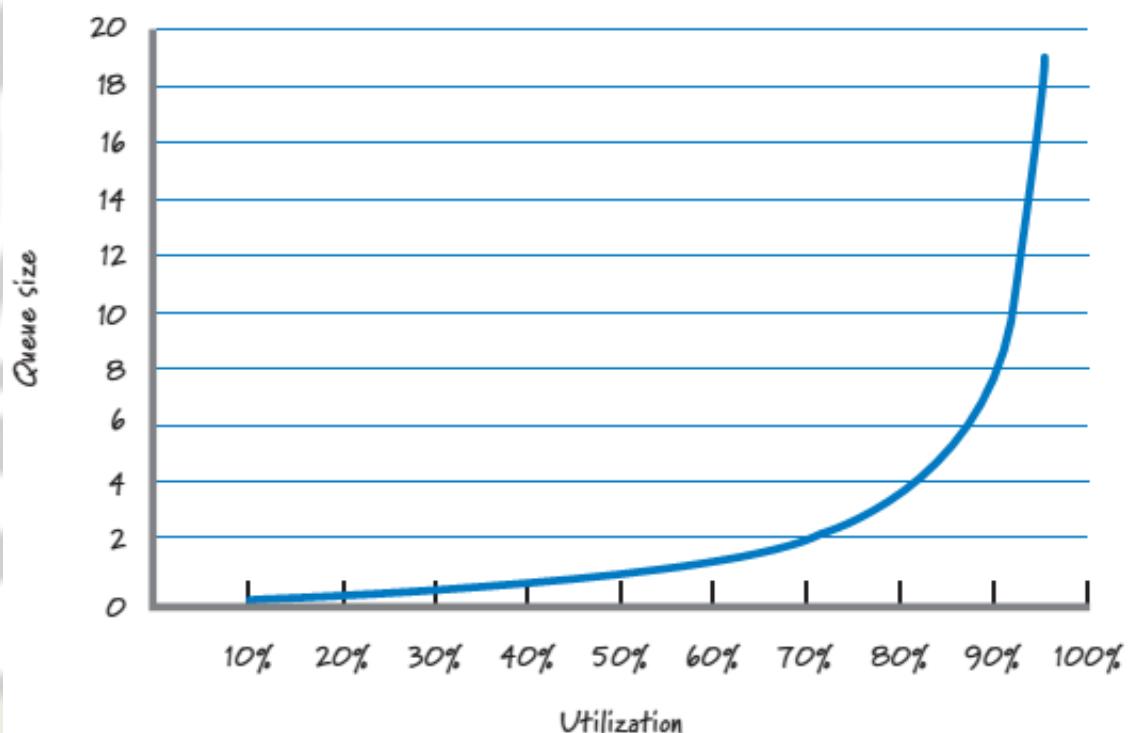
هنگامی که به آن حالت رسیدید، خارج شدن از آن بسیار دشوار است (احتمالاً باید شروع به کشتن مشاغل یا راه اندازی مجدد دستگاه کنید).

اگر رایانه شما با استفاده نزدیک به 80 درصد آن را اجرا کنید بسیار کارآمدتر خواهد بود.

# How utilization affects queue size

- obvious damage caused when striving for 100% utilization.
- As capacity utilization increases, queue size and delay increase.

آسیب آشکاری که هنگام تلاش برای استفاده 100٪ ایجاد می شود.  
با افزایش استفاده از ظرفیت، اندازه صف و تاخیر افزایش می یابد.



# Focus on Idle Work, Not Idle Workers(XI)

- The **idle work** (**delayed work**) **grows exponentially** once we get into the **high levels of utilization**.
- And that idle work can be **very expensive**, frequently many times more expensive than the cost of idle workers.
- So, in Scrum, we are **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.

کار بیکار (کار با تأخیر) به مغض اینکه به سطوح بالای بهره برداری رسیدیم به طور تصاعدی رشد می کند.  
و این کار بیکار می تواند بسیار گران باشد، اغلب چندین برابر هزینه کارگران بیکار.  
بنابراین، در اسکرام، ما میدانیم که یافتن گلوهای کار و تمرکز تلاش‌هایمان برای از بین بردن آنها، از لحاظ اقتصادی فعالیتی بسیار معقولتر از تلاش برای مشغول نگهداشتن 100% همه است.

# Consider Cost of Delay

هزینه تأخیر هزینه مالی مربوط به تأخیر کار یا تأخیر در دستیابی به یک نقطه عطف است.  
با کاهش ضایعات کارگران بیکار (با افزایش بهره برداری از آنها)، به طور همزمان ضایعات مرتبط با کار بیکار را افزایش می دهیم (کار نشسته در صفحه انتظار خدمات).

با استفاده از هزینه تأخیر، می توانیم محاسبه کنیم که کدام زباله از نظر اقتصادی آسیب بیشتری دارد.  
این را با این واقعیت ترکیب کنید که اکثر سازمانهای توسعه متوجه نمیشوند که کار (موجودی) را در صفها جمع کرده‌اند، و به راحتی میتوان فهمید که چرا رفتار پیشفرض آنها تمرکز بر حذف ضایعات قابل مشاهده کارگران بیکار است.

- Cost of delay is the financial cost associated with delaying work or delaying achievement of a milestone.
- By reducing the waste of idle workers (by increasing their utilization), we simultaneously increase the waste associated with idle work (work sitting in queues waiting to be serviced).
- Using cost of delay, we can calculate which waste is more economically damaging.
- Combine that with the fact that most development organizations don't realize they have accumulated work (inventory) sitting in queues, and it is easy to see why their default behavior is to focus on eliminating the visible waste of idle workers.

# Agile Principles(VI)

Dr. Elham Mahmoudzadeh

Isfahan University of Technology

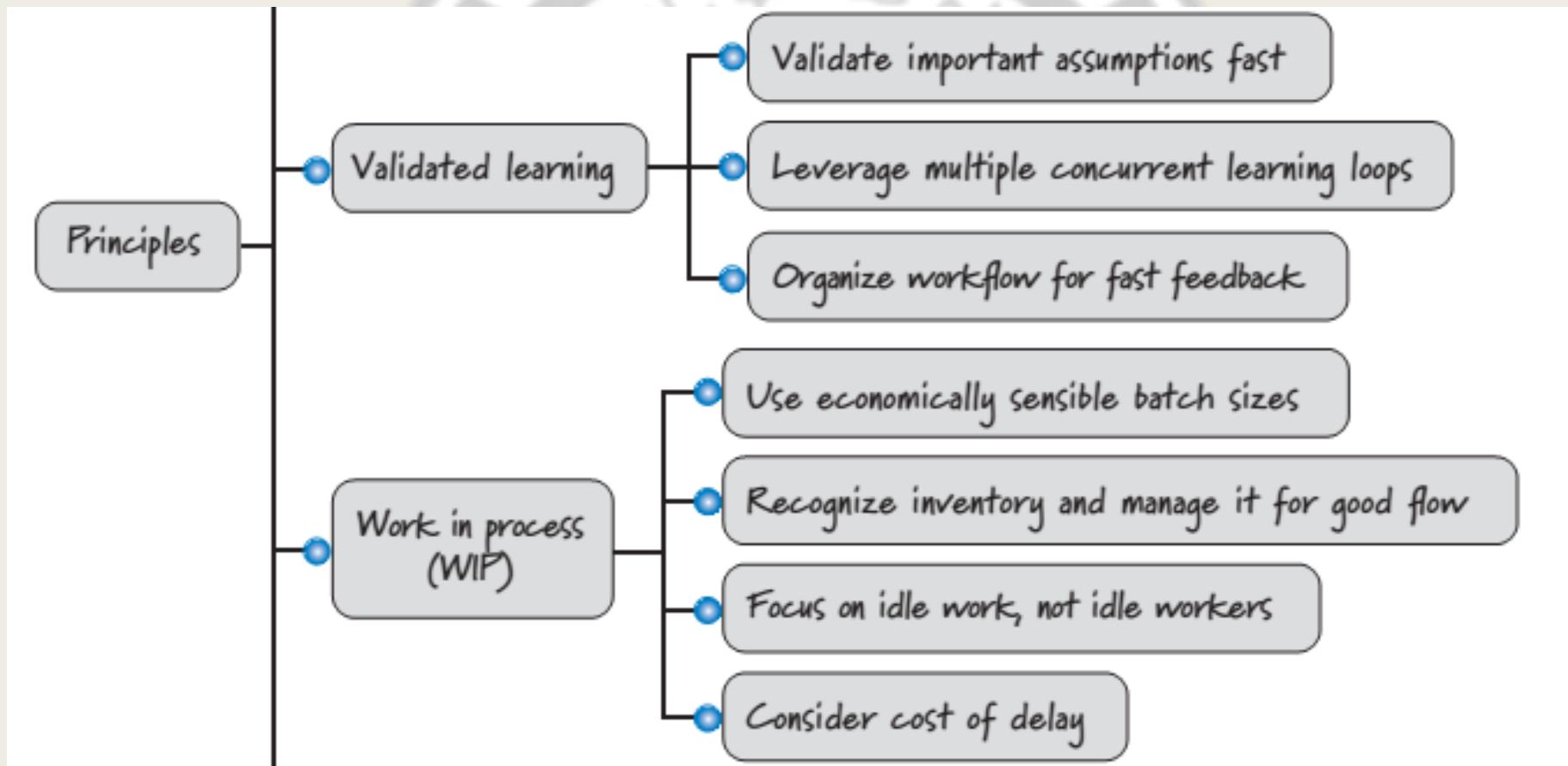
[mahmoudzadeh@iut.ac.ir](mailto:mahmoudzadeh@iut.ac.ir)

2021

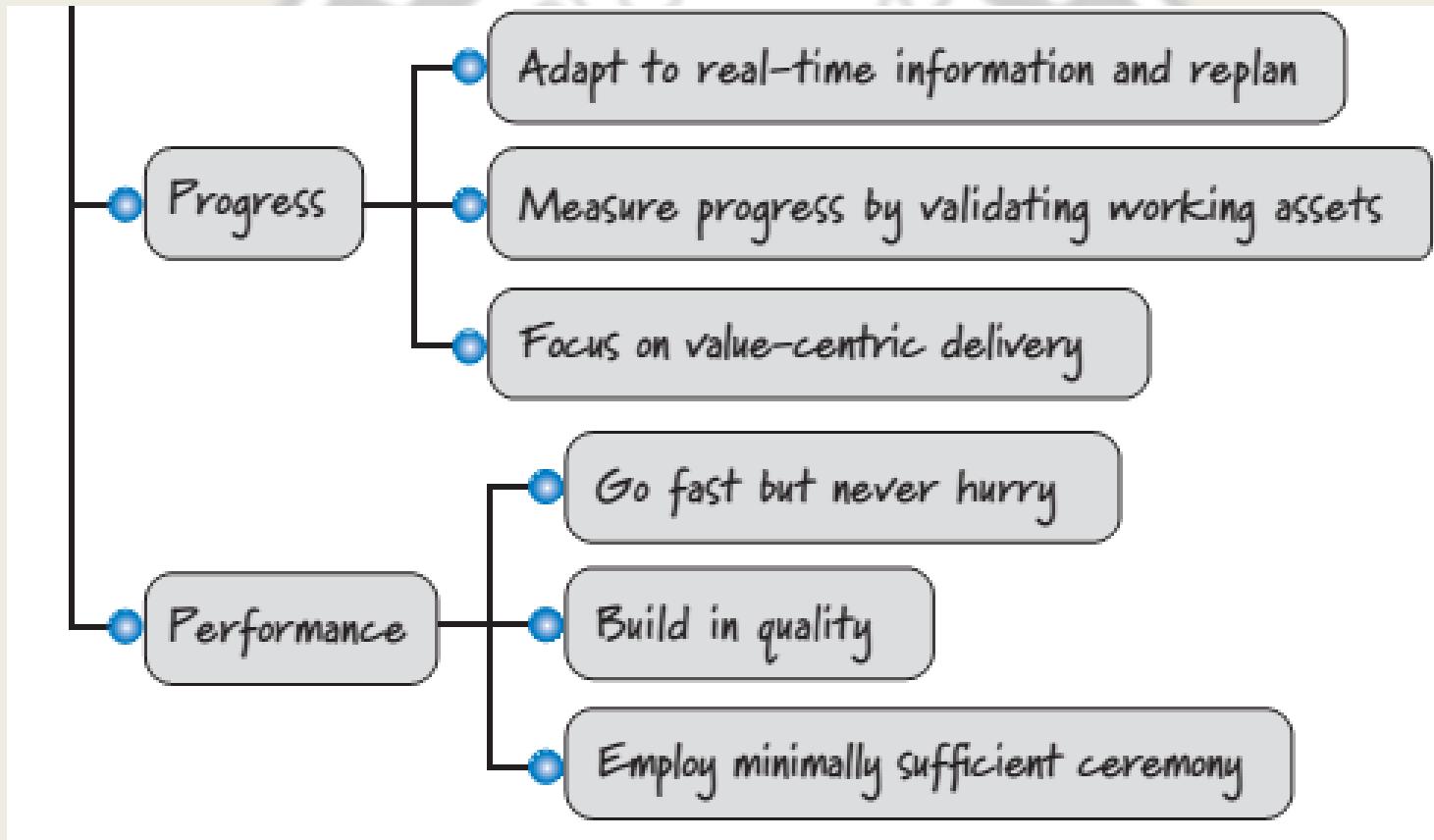
# Categorization of principles (Up)



# Categorization of principles (Middle)



# Categorization of principles (Bottom)



# Progress

هنگام استفاده از اسکرام، پیشرفت را با آنچه ارائه و تأیید کرد هایم انداز هگیری میکنیم، نه بر اساس اینکه چگونه طبق برنامه از پیش تعریف شده پیش میرویم یا تا چه حد در مرحله یا مرحله خاصی از توسعه قرار داریم.

سه اصل چاپک مربوط به این موضوع.

انطباق با اطلاعات زمان واقعی و برنامه ریزی مجدد.

پیشرفت را با اعتبار سنجی دارایی های کاری اندازه گیری کنید.

بر تحويل ارزش محور تمرکز کنید.

- When using Scrum, we **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 related to this topic.
  1. **Adapt to real-time information and re-plan.**
  2. **Measure progress** by **validating working assets.**
  3. **Focus on value-centric delivery.**

# Adapt to Real-Time Information and Re-plan

- In a plan-driven, the **plan** is the **authoritative source** on **how** and **when** work should occur. As such, **conformance** to the plan is expected.
- In Scrum we believe that unbridled faith in the plan will frequently **blind us** to the fact that the **plan might be wrong**.
- Instead, our goal is to **rapidly re-plan** and **adapt** to the stream of economically important information that is continuously arriving during the development effort.

در یک برنامه محور، برنامه منبع معتبری است که چگونه و چه زمانی کار باید انجام شود. به این ترتیب، انطباق با طرح مورد انتظار است.

در اسکرام ما معتقدیم که ایمان لجام گسیخته به طرح، اغلب ما را از این واقعیت که ممکن است طرح اشتباه باشد، کور می کند.

در عوض، هدف ما برنامه‌ریزی سریع و انطباق با جریان اطلاعات مهم اقتصادی است که به طور مداوم در طول تلاش توسعه به دست می‌آیند.

# Measure Progress by Validating Working Assets(I)

- Progress during a sequential, plan-driven development 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.
- Yet in the end, the product we created in full accordance with the plan might deliver far less customer value than anticipated.

پیشرفت در طول یک تلاش توسعه پی در پی و برنامه محور با تکمیل یک مرحله و اجازه ورود به فاز بعدی نشان داده می شود. در نتیجه، اگر هر مرحله همانطور که انتظار می رود شروع و تکمیل شود، به نظر می رسد که تلاش توسعه محصول به خوبی پیش می رود. با این حال، در پایان، محصولی که ما مطابق با طرح ایجاد کردیم، ممکن است ارزش بسیار کمتری نسبت به آنچه پیش‌بینی میشد به مشتری ارائه دهد.

# Measure Progress by Validating Working Assets(II)

- With Scrum, we measure progress by building working, validated assets that deliver value and that can be used to validate important assumptions.
- This gives us the feedback to know what the right next step is.
- In Scrum, it's not about how much work we start; it's all about what customer-valuable work we finish.

با اسکرام، ما پیشرفت را با ساخت داراییهای معتبر و کارآمد که ارزش ارائه میدهند و میتوانند برای اعتبارسنجی مفروضات مهم استفاده شوند، اندازهگیری میکنیم.  
این به ما بازخورد می دهد تا بدانیم قدم بعدی درست چیست.  
در اسکرام، این نیست که چقدر کار را شروع می کنیم. همه چیز به این بستگی دارد که چه کار ارزشمندی برای مشتری به پایان می رسانیم.

# Focus on Value-Centric Delivery(I)

- Plan-driven, sequential 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.

توسعه متواالی مبتنی بر برنامه بر پیگیری مجدانه فرآیند مرکز است. با ساختار خود، ادغام و تحویل ویژگی ها در طول توسعه متواالی در پایان تلاش اتفاق می افتد.

با این رویکرد این خطر وجود دارد که قبل از اینکه همه ارزش های مهم را به مشتریان خود تحویل دهیم، منابع (زمان یا پول) ما تمام شود.

# Focus on Value-Centric Delivery(II)

اعقاد مرتبط توسعه سنتی این است که مصنوعات برنامه ریزی و مستندسازی که در مسیر ارائه ویژگی ها تولید می شوند، خود ارزشمند هستند.

اگر این مصنوعات واقعاً ارزشمند هستند، بیشتر اوقات فقط برای فرآیند پایین دستی ارزشمند هستند و نه برای مشتریان. و اگر برای مشتری ارزشمند باشند، این ارزش تنها در صورتی به دست می آید که محصول مطلوب در نهایت به مشتری تحویل داده شود. تا زمانی که این اتفاق نیفتد، این مصنوعات هیچ ارزش مستقیمی برای مشتری ایجاد نمی کنند.

- A related belief of traditional development is that the planning and document artifacts that get produced enroute to delivering features are themselves valuable.
- If these artifacts are indeed valuable, most of the time they are valuable only to the downstream process and not the customers. And, if they are valuable to the customer, that value accrues only if a desirable product is ultimately delivered to the customer. Until that happens, these artifacts provide no direct customer value.

# Focus on Value-Centric Delivery(III)

- Scrum is a customer-value-centric form of development.
- It is 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.

اسکرام شکلی از توسعه مبتنی بر ارزش مشتری است.  
این مبتنی بر یک مدل افزایشی و اولویت‌بندی شده از تحویل است که در آن ویژگی‌های با بالاترین ارزش به طور مداوم ساخته شده و در تکرار بعدی ارائه می‌شوند.  
در نتیجه، مشتریان زودتر جریان مستمر ویژگی‌های با ارزش را دریافت می‌کنند.

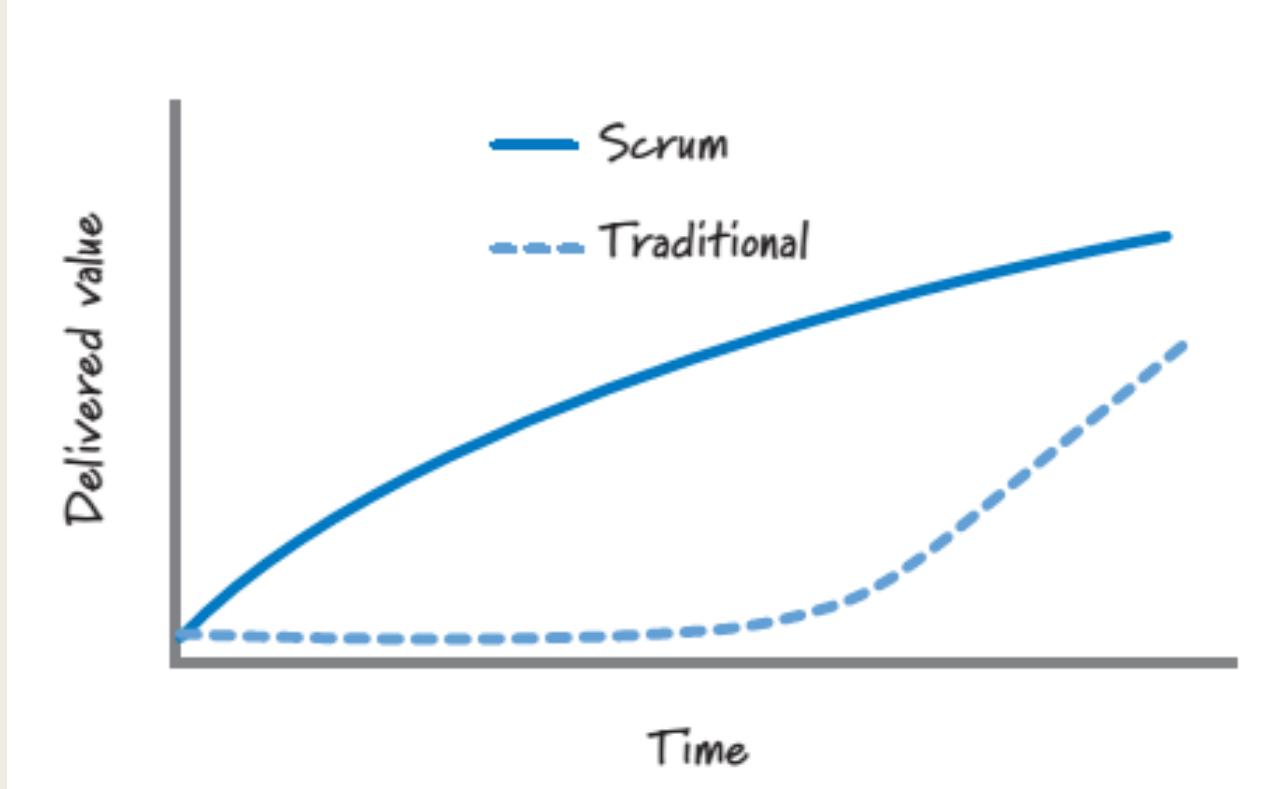
# Focus on Value-Centric Delivery(IV)

- In Scrum, value is generated by delivering working assets to customers, by validating important assumptions, or by acquiring valuable knowledge.
- In Scrum, we believe that 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.

در اسکرام، ارزش با تحویل دارایی های کاری به مشتریان، اعتبارسنجی مفروضات مهم، یا با کسب دانش ارزشمند ایجاد می شود.

در اسکرام، ما بر این باوریم که مصنوعات میانی هیچ ارزش درک شده ای برای مشتری ارائه نمی دهند و اگر خودشان نتوانند برای ایجاد بازخورد مهم یا کسب دانش مهم مورد استفاده قرار گیرند، صرفاً وسیله ای برای رسیدن به هدف هستند.

# Deliver high-value features sooner



# Performance

- There are specific **performance-related characteristics** we expect when using Scrum.

هنگام استفاده از اسکرام، ویژگیهای مرتبط با عملکرد خاصی وجود دارد.

سه اصل چابک مربوط به این موضوع.  
سریع برو اما هرگز عجله نکن.

ساخت با کیفیت  
از مراسم حداقل کافی استفاده کنید.

- **Three agile principles** related to this topic.

1. **Go fast but never hurry.**
2. **Build in quality.**
3. **Employ minimally sufficient ceremony.**

# Go fast but never hurry(I)

توسعه مبتنی بر برنامه بر این باور است که اگر از برنامه پیروی کنیم و کارها را در اولین بار به درستی انجام دهیم، از دوباره کاری پرهزینه و وقت گیر جلوگیری خواهیم کرد. حرکت سریع از یک مرحله به مرحله دیگر البته مطلوب است، اما یک هدف اصلی نیست.

در اسکرام، یک هدف اصلی این است که زیرک، سازگار و سریع باشد.  
با حرکت سریع، ما سریع تحويل میدهیم، سریع بازخورد دریافت میکنیم و ارزش را زودتر به دست مشتریانمان میرسانیم. یادگیری و واکنش سریع به ما این امکان را می دهد که زودتر درآمدزایی کنیم و/یا هزینه ها را کاهش دهیم.

- 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. Moving from step to step quickly is of course desirable, but it isn't a principal goal.
- In Scrum, 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.

# Go fast but never hurry(II)

- Do not, however, mistake going fast for being hurried.
- In Scrum, time is of the essence, but we don't rush to get things done. Doing so would likely violate the Scrum principle of sustainable pace—people should be able to work at a pace that they can continue for an extended period of time.
- In addition, hurrying will likely come at the expense of quality.

با این حال، سریع رفتن را با عجله اشتباه نکنید.

در اسکرام، زمان بسیار مهم است، اما ما برای انجام کارها عجله نداریم. انجام این کار احتمالاً اصل سرعت پایدار اسکرام را نقض می کند - مردم باید بتوانند با سرعتی کار کنند که بتوانند برای مدت زمان طولانی ادامه دهند.

علاوه بر این، عجله به احتمال زیاد به قیمت از دست دادن کیفیت تمام خواهد شد.

# Build In Quality(I)

در طول توسعه برنامه محور، اعتقاد بر این است که از طریق انجام دقیق و متوالی کار، محصولی با کیفیت بالا به دست می آوریم. با این حال، تا زمانی که آزمایش محصول یکپارچه را انجام ندهیم، واقعاً نمیتوانیم این کیفیت را تأیید کنیم، که در مرحله پایانی فرآیند اتفاق میافتد. اگر آزمایش نشان دهد که کیفیت وجود ندارد، ما باید وارد مرحله تست و تعمیر پر هزینه شویم تا کیفیت را آزمایش کنیم. همچنین، از آنجایی که تیم متفاوتی اغلب در هر مرحله کار می کند، تیم آزمایش اغلب به عنوان مالک کیفیت نتیجه در نظر گرفته می شود.

- During plan-driven development, the belief is that through careful, sequential performance of work we get a high-quality product.
- However, 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.
- 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.

# Build In Quality(II)

در اسکرام، کیفیت چیزی نیست که تیم آزمایش در پایان آن را «آزمایش» کند. این چیزی است که یک تیم متقابل اسکرام مالک آن است و به طور مداوم هر سرعتی را ایجاد می کند. هر افزایش ارزشی که ایجاد می شود تا سطح بالایی از اطمینان کامل می شود و این پتانسیل را دارد که در تولید قرار گیرد یا به مشتریان ارسال شود. نیاز به هر آزمایش دیرهنگام قابل توجهی برای حفظ کیفیت به میزان قابل توجهی کاهش می یابد.

- In Scrum, 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.
- 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.
- Need for any significant late testing to tack on quality is substantially reduced.

# Employ Minimally Sufficient Ceremony(I)

برگزاری مراسم حداقل کافی

- Plan-driven processes tend to be high-ceremony, document-centric, process-heavy approaches.
- A side effect of Scrum's being value-centric is that very little emphasis is put on process-centric ceremonies.
- I am referring to ceremony that is unnecessary formality.
- Some might call it “process for the sake of process.” Such ceremony has a cost but adds little or no value (in other words, it's a type of waste).

فرآیندهای برنامه محور معمولاً رویکردهای تشریفاتی بالا، سند محور و فرآیندی سنگین هستند.

یک عارضه جانبی ارزش محور بودن اسکرام این است که تأکید بسیار کمی بر مراسم فرآیند محور است. منظورم مراسمی است که تشریفات غیر ضروری است.

برخی ممکن است آن را «فرآیند به خاطر فرآیند» بنامند. چنین مراسمی هزینه دارد اما ارزش چندانی ندارد (به عبارت دیگر، نوعی ضایعات است).

# Employ Minimally Sufficient Ceremony(II)

- In Scrum, our goal is to eliminate unnecessary formality.
- Therefore, we set the ceremonial bar at a low level, one that is minimally sufficient or good enough.

در اسکرام هدف ما حذف رسمیت غیر ضروری است.  
بنابراین، نوار تشریفات را در سطح پایینی قرار می دهیم، سطحی که حداقل کافی یا به اندازه کافی خوب باشد.

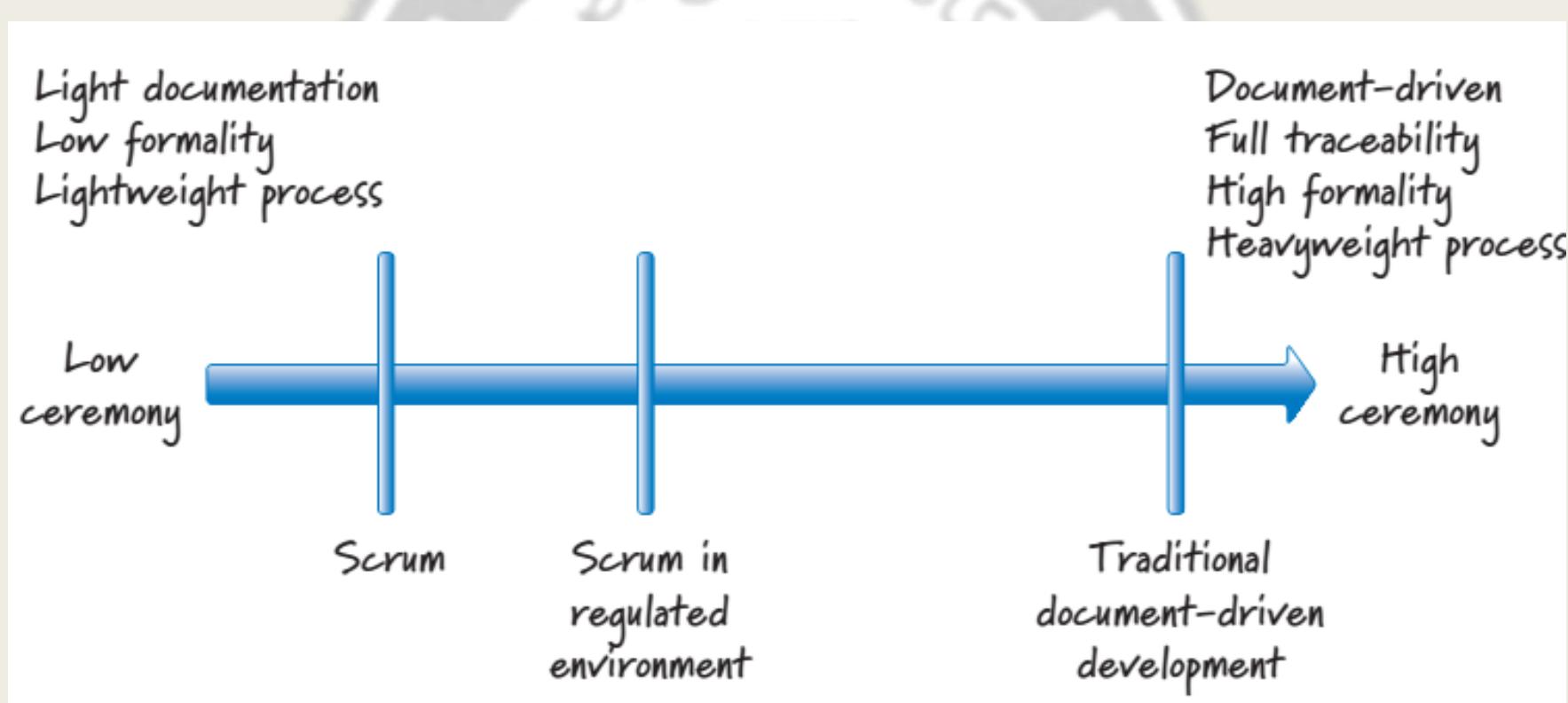
# Employ Minimally Sufficient Ceremony(III)

- Frequently the Scrum focus on minimally sufficient ceremony is misinterpreted to mean things like "Scrum is anti-documentation." Scrum isn't anti-documentation.
- Rather, when using Scrum, we adopt an economic perspective and carefully review which documents we create. If we write a document that adds no value, we have wasted our time and money creating a dead document. However, not all documents are dead.

اغلب تمرکز اسکرام بر مراسم حداقل کافی به اشتباه به معنای مواردی مانند "اسکرام ضد اسناد است" تعبیر می شود. اسکرام ضد اسناد نیست.

در عوض، هنگام استفاده از اسکرام، یک دیدگاه اقتصادی اتخاذ می کنیم و اسنادی را که ایجاد می کنیم به دقت بررسی می کنیم. اگر سندی بنویسیم که هیچ ارزشی نداشته باشد، وقت و پول خود را با ایجاد یک سند مرده تلف کرده ایم. با این حال، همه اسناد مرده نیستند.

# Ceremony scale



# we will likely write a document if

این یک محصول قابل تحویل به عنوان بخشی از محصول است (به عنوان مثال دستورالعمل نصب، راهنمای کاربر و غیره). هدف ما این است که یک بحث، تصمیم یا توافق مهم را به تصویر بکشیم تا در آینده خاطره روشنی از آنچه مورد بحث، تصمیم گیری یا توافق شده است داشته باشیم.

این روش با ارزشی است که به اعضای جدید تیم کمک می کند تا سریعاً سرعت بگیرند.  
یک الزام قانونی وجود دارد که یک سند خاص نوشته شود (هزینه انجام تجارت در یک صنعت تحت نظارت).

- It is a **deliverable** as part of the product (for example, **installation instructions**, **user's guide**, and so on).
- Our goal is to capture an important **discussion**, **decision**, or **agreement** so that in the future we will have a **clear recollection** of what was **discussed**, **decided**, or **agreed** to.
- It is the high-value way of **helping new team members** come up to speed quickly.
- There is a **regulatory requirement** that a certain document be written (a cost of doing business in a regulated industry).

چیزی که ما سعی می کنیم از آن اجتناب کنیم، کاری است که هیچ ارزش اقتصادی کوتاه مدت یا بلندمدتی اضافه نمی کند.  
در اسکرام، ما معتقدیم که زمان و پول بهتر صرف ارائه ارزش مشتری می شود.

What we are trying to **avoid** is work that  
**adds no short-term or long-term economic**  
**value.**

In Scrum, we believe that **time and money**  
**are better spent delivering customer value.**

# Comparison Summary of Plan-Driven and Agile Principles(I)

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.

# Comparison Summary of Plan-Driven and Agile Principles(II)

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

اکتشاف در مقابل  
بهره برداری

# Comparison Summary of Plan-Driven and Agile Principles(III)

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.

انطباق و پیروی از  
برنامه

# Comparison Summary of Plan-Driven and Agile Principles(IV)

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.

# Requirements and User Stories(I)

Dr. Elham Mahmoudzadeh

Isfahan University of Technology

[mahmoudzadeh@iut.ac.ir](mailto:mahmoudzadeh@iut.ac.ir)

2022

# Introduction

- Scrum and sequential product development treat requirements very differently.
- With sequential product development, requirements are nonnegotiable, detailed up front, and meant to stand alone.
- In Scrum, the details of a requirement are negotiated through conversations that happen continuously during development and are fleshed out just in time and just enough for the teams to start building functionality to support that requirement.

# What is a requirement?

- New system's capability.
- A *requirement* is simply a statement of what the system must do or what characteristic it must have.
- They focus on the “what” of the system.

# Requirements in sequential product development

- Are treated much as they are in manufacturing.
- They are required, nonnegotiable specifications to which the product must conform.
- These requirements are created up front and given to the development group in the form of a highly detailed document.
- It is the job of the development group, then, to produce a product that conforms to the detailed requirements.

# Requirements in sequential product development(Cnt'd)

- When a change from the original plan is deemed necessary, it is managed through a formal change control process.
- Because conformance to specifications is the goal, these deviations are undesirable and expensive.
- After all, much of the work in process (WIP), in the form of highly detailed requirements (and all work based on them), might need to be changed or discarded.

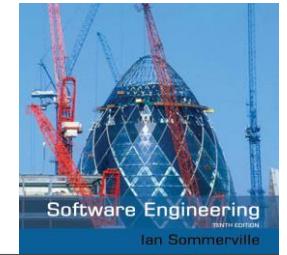
# Requirements in Scrum

- Scrum views requirements as an **important degree of freedom** that we can **manipulate** to meet our **business goals**.
- For example, if we're **running out of time** or **money**, we can **drop low-value requirements**.
- If, during development, new information indicates that the **cost/benefit ratio** of a **requirement** has become significantly less favorable, we can **choose to drop the requirement from the product**.
- And if a **new high-value requirement emerges**, we have the ability to **add it to the product**, perhaps **discarding a lower-value requirement** to make room.

# The fact is

- When developing innovative products, you can't create complete requirements or designs up front by simply working longer and harder.
- Some requirements and design will always emerge once product development is under way;
- No amount of comprehensive up-front work will prevent that.

## Agile methods and requirements



- ✧ Many agile methods argue that producing detailed system requirements is a waste of time as requirements change so quickly.
- ✧ The requirements document is therefore always out of date.
- ✧ Agile methods usually use incremental requirements engineering and may express requirements as 'user stories'.
- ✧ This is practical for business systems but problematic for systems that require pre-delivery analysis (e.g. critical systems) or systems developed by several teams.

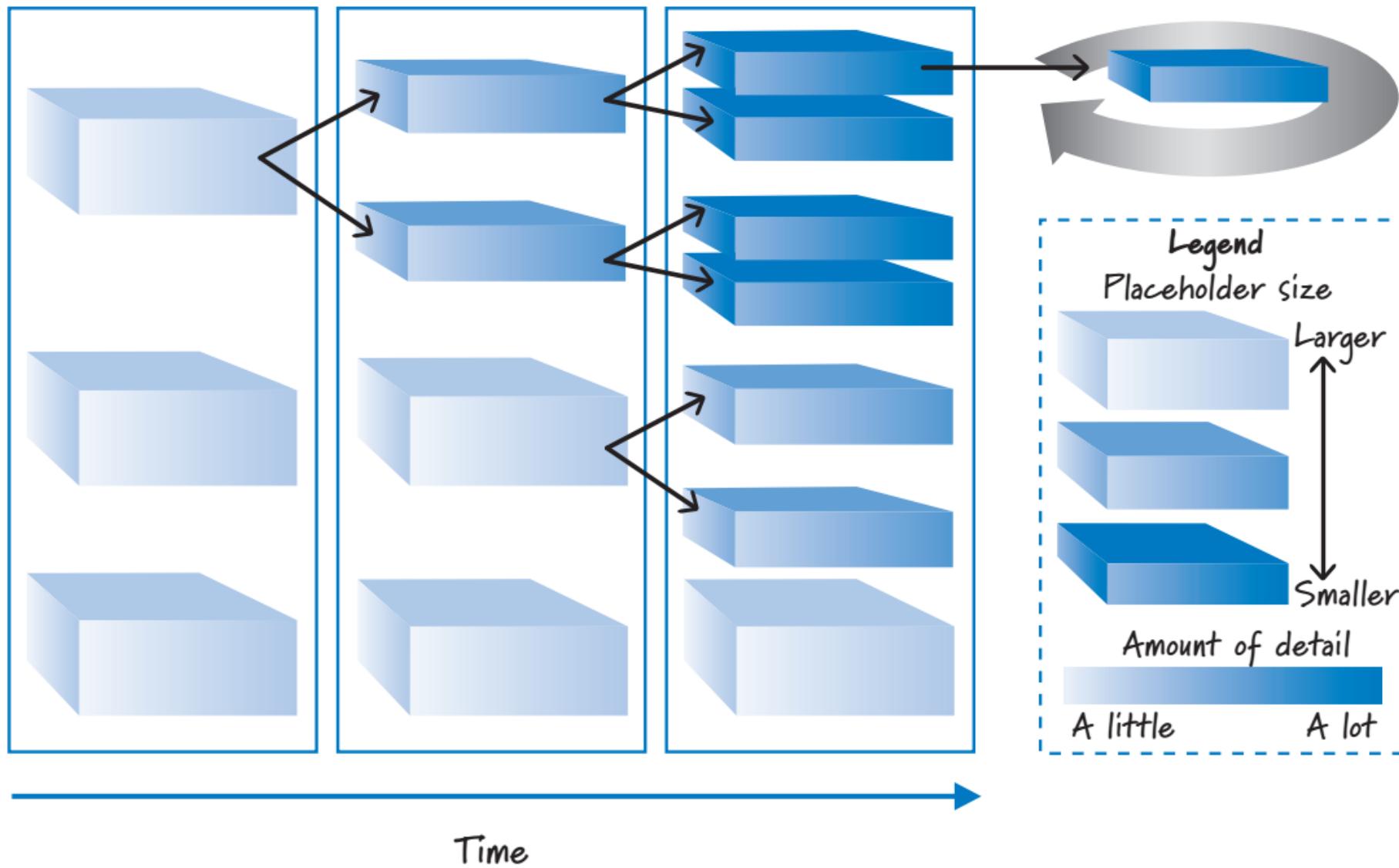
# In Scrum,

- We **don't invest a great deal of time** and **money** in fleshing out the details of a requirement up front.
- Because we **expect the specifics to change** as time passes and as we **learn more about** what we are building, we **avoid overinvesting** in requirements that we **might later discard**.
- Instead of **compiling a large inventory** of detailed requirements up front, we create **placeholders** for the requirements, called **product backlog items (PBIs)**.
- Each product backlog item **represents desirable business value**.

# The Product Backlog Items

- Initially the product backlog items are large (representing large swaths of business value), and there is very little detail associated with them.
- Over time, we flow these product backlog items through a series of conversations among the stakeholders, product owner, and development team, refining them into a collection of smaller, more detailed PBIs.
- Eventually a product backlog item is small and detailed enough to move into a sprint, where it will be designed, built, and tested.
- Even during the sprint, however, more details will be exposed in conversations between the product owner and the development team.

## Product backlog over time





The product backlog is simply a snapshot of the current collection of items and their associated details.

# Format of requirements

- While Scrum doesn't specify any standard format for these product backlog items, many teams represent PBIs as user stories.
- You don't have to. Some teams prefer use cases, and others choose to represent their PBIs in their own custom formats.

# Using Conversations

- As a communication vehicle, requirements facilitate a shared understanding of what needs to be built.
- They allow the people who understand what should be created to clearly communicate their desires to the people who have to create it.

# Sequential Product Development

- Relies heavily on written requirements, which look impressive but can easily be misunderstood.
- It seemed unlikely that there would be no ambiguities in document, detailed use case document. English just isn't that precise; even if it were, people just aren't that precise with their writing.



A way to better ensure that the desired features are being built is for the people who know what they want to have timely conversations with the people who are designing, building, and testing features.

# Using conversation in Scrum

- We leverage conversation as a **key tool** for ensuring that requirements are **properly discussed** and **communicated**.
- **Verbal communication** has the benefit of being **high-bandwidth** and providing **fast feedback**, making it **easier** and **cheaper** to gain a **shared understanding**.
- Conversations enable **bidirectional communication** that can spark ideas about problems and opportunities—discussions that would not likely arise from reading a document.

# Using conversation in Scrum (Cnt'd)

- Conversation is just a tool.
- It doesn't replace all documents.
- In Scrum, the product backlog is a “living document,” available at all times during product development.
- Those who still want or must have a requirements specification document can create one at any time, simply by collecting the product backlog items and all of their associated details into a document formatted however they like.

# Progressive Refinement

- With sequential product development all requirements must be at the same level of detail at the same time.
- Approved requirements document must specify each and every requirement so that the teams doing the design, build, and test work can understand how to conform to the specifications.
- There are no details left to be added.

# Disadvantages of forcing all requirements to be at the same level of detail at the same time

- We must predict all of these details early during product development when we have the least knowledge that we'll ever have.
- We treat all requirements the same regardless of their priority, forcing us to dedicate valuable resources today to create details for requirements that we may never build.
- We create a large inventory of requirements that will likely be very expensive to rework or discard when things change.
- We reduce the likelihood of using conversations to elaborate on and clarify requirements because the requirements are already “complete.”

# Refinement is Scrum

- When using Scrum, not all requirements have to be at the same level of detail at the same time.
- Requirements that we'll work on sooner will be smaller and more detailed than ones that we won't work on for some time.
- We employ a strategy of progressive refinement to disaggregate, in a just-in-time fashion, large, lightly detailed requirements into a set of smaller, more detailed items.

ما از یک استراتژی پالایش تدریجی استفاده میکنیم تا، بهموقع، نیازمندیهای بزرگ و با جزئیات جزئی را در مجموعه‌ای از موارد کوچکتر و دقیق‌تر تفکیک کنیم.

# What Are User Stories?

- User stories are a convenient format for expressing the desired business value for many types of product backlog items.
- User stories are crafted in a way that makes them understandable to both business people and technical people.
- They are structurally simple and provide a great placeholder for a conversation.
- Additionally, they can be written at various levels of granularity and are easy to progressively refine.

# What Are User Stories (Cnt'd)?

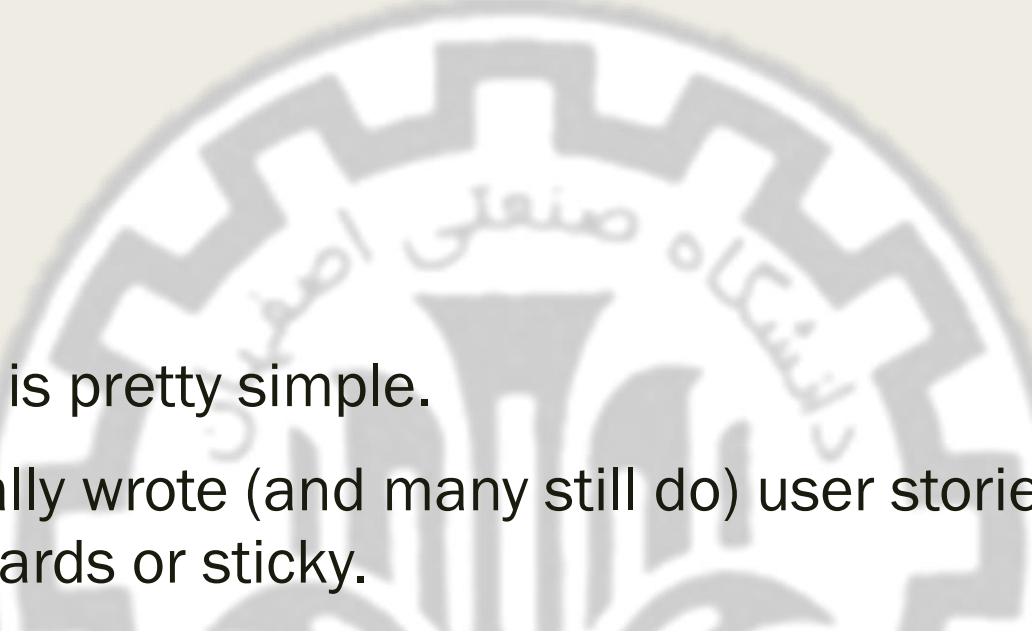
- They are simply a **lightweight approach** that **match** nicely with **core agile principles** and our need for an **efficient** and **effective placeholder**.
- User stories are **central placeholder** to attach any other relevant information and they are helpful for **detailing a requirement**.

# What Are User Stories (Cnt'd)?

- Ron Jeffries offers a simple yet effective way to think about user stories (Jeffries 2001).
- He describes them as **three Cs**: **Card**, **Conversation**, and **Confirmation**.

# Card

- The card idea is pretty simple.
- People originally wrote (and many still do) user stories directly on 3 × 5-inch index cards or sticky.



User Story Title

As a <user role> I want to <goal> so  
that <benefit>.

Template

Find Reviews Near Address

As a typical user I want to see unbiased  
reviews of a restaurant near an address  
so that I can decide where to go for  
dinner.

# Card (Cnt'd)

- A common template format for writing user stories is to specify a class of users (the **user role**), what that class of users wants to achieve (the **goal**), and why the users want to achieve the goal (the **benefit**).
- The “so that” part of a user story is optional, but unless the purpose of the story is completely obvious to everyone, we should include it with every user story.

# Card (Cnt'd)

- The card isn't intended to capture all of the information that makes up the requirement.
- We deliberately use **small cards** with **limited space** to promote brevity.
- A card should hold a few sentences that **capture the essence** or **intent of a requirement**.
- It serves as the **placeholder** for **more detailed discussions** that will take place among the **stakeholders**, **product owner**, and **development team**.

اختصار

# Conversation

- The **details of a requirement** are **exposed** and **communicated** in a conversation among the development team, product owner, and stakeholders.
- The **user story** is simply a **promise** to have that conversation.

# Conversation (Cnt'd)

- Conversation is typically **not a one-time event**, but rather an **ongoing dialogue**.
- There can be an **initial conversation** **when the user story is written**, another conversation when it's **refined**, yet another when it's **estimated**, another **during sprint planning** (when the team is diving into the task-level details), and finally, ongoing conversations while the user story is being **designed**, **built**, and **tested** during the **sprint**.

# Conversation (Cnt'd)

- One of the **benefits of user stories** is that they **shift** some of the **focus** away **from writing** and **onto conversations**.
- These **conversations** enable a **richer form of exchanging information** and **collaborating** to **ensure** that the **correct requirements** are **expressed** and **understood** by everyone.
- Although **conversations** are **largely verbal**, they can be and frequently are supplemented **with documents**.

# User story with additional data attached

User story can reference an entire article for future reading and conversation.

## Johnson Visualization of MRI Data

As a radiologist I want to visualize MRI data using Dr. Johnson's new algorithm.

For more details see the January 2007 issue of the Journal of Mathematics, pages 110-118.

# Conversation (Cnt'd)

- User stories are simply a good starting point for eliciting the initial essence of what is desired, and for providing a reminder to discuss requirements in more detail when appropriate.
- However, user stories can and should be supplemented with whatever other written information helps provide clarity regarding what is desired.

# Confirmation

- A user story also contains **confirmation information** in the form of **conditions of satisfaction**.
- These are **acceptance criteria** that **clarify** the desired behavior.
- They are used by the **development team** to **better understand** what to build and **test** by the **product owner** to confirm that the user story has been implemented to **his satisfaction**.

# Confirmation (Cnt'd)

- If the front of the card has a few-line description of the story, the back of the card could specify the conditions of satisfaction.

## Upload File

As a wiki user I want to upload a file to the wiki so that I can share it with my colleagues.

## Conditions of Satisfaction

Verify with .txt and .doc files  
Verify with jpg, gif, and png files  
Verify with .mp4 files <= 1 GB  
Verify no DRM-restricted files

# Confirmation (Cnt'd)

- These conditions of satisfaction can be expressed as high-level acceptance tests.
- These tests would not be the only tests that are run when the story is being developed.
- In fact, for the handful of acceptance tests that are associated with a user story, the team will have many more tests (perhaps 10 to 100 times more) at a detailed technical level that the product owner doesn't even know about.

# Reasons of Confirmation

- First, they are an important way to capture and communicate, from the product owner's perspective, how to determine if the story has been implemented correctly.
- These tests can also be a helpful way to create initial stories and refine them as more details become known. This approach is sometimes called specification by example or acceptance-test-driven development (ATTD).

# An example

- Initially, let's limit uploaded file sizes to be 1 GB or less. Also, make sure that we can properly load common text and graphics files. And for legal reasons we can't have any files with digital rights management (DRM) restrictions loaded to the wiki.

## Upload File

As a wiki user I want to upload a file to the wiki so that I can share it with my colleagues.

## Conditions of Satisfaction

Verify with .txt and .doc files  
Verify with .jpg, .gif, and .png files  
Verify with .mp4 files <= 1 GB  
Verify no DRM-restricted files

# Confirmation (Cnt'd)

- If we were using a tool, we could conveniently define these tests in a table, which shows examples of different file sizes and whether or not they are valid.
- By elaborating on specific examples, we can drive the story creation and refinement process and have (automated) acceptance tests available for each story.

Size	Valid()
0	True
1,073,741,824	True
1,073,741,825	False

# Level of Detail

- User stories are an excellent vehicle for carrying items of customer or user value through the Scrum value-creation flow.
- If we have only one story size (the size that would comfortably fit within a short-duration sprint), it will be difficult to do higher-level planning and to reap the benefits of progressive refinement.

# Level of Detail(Cnt'd)

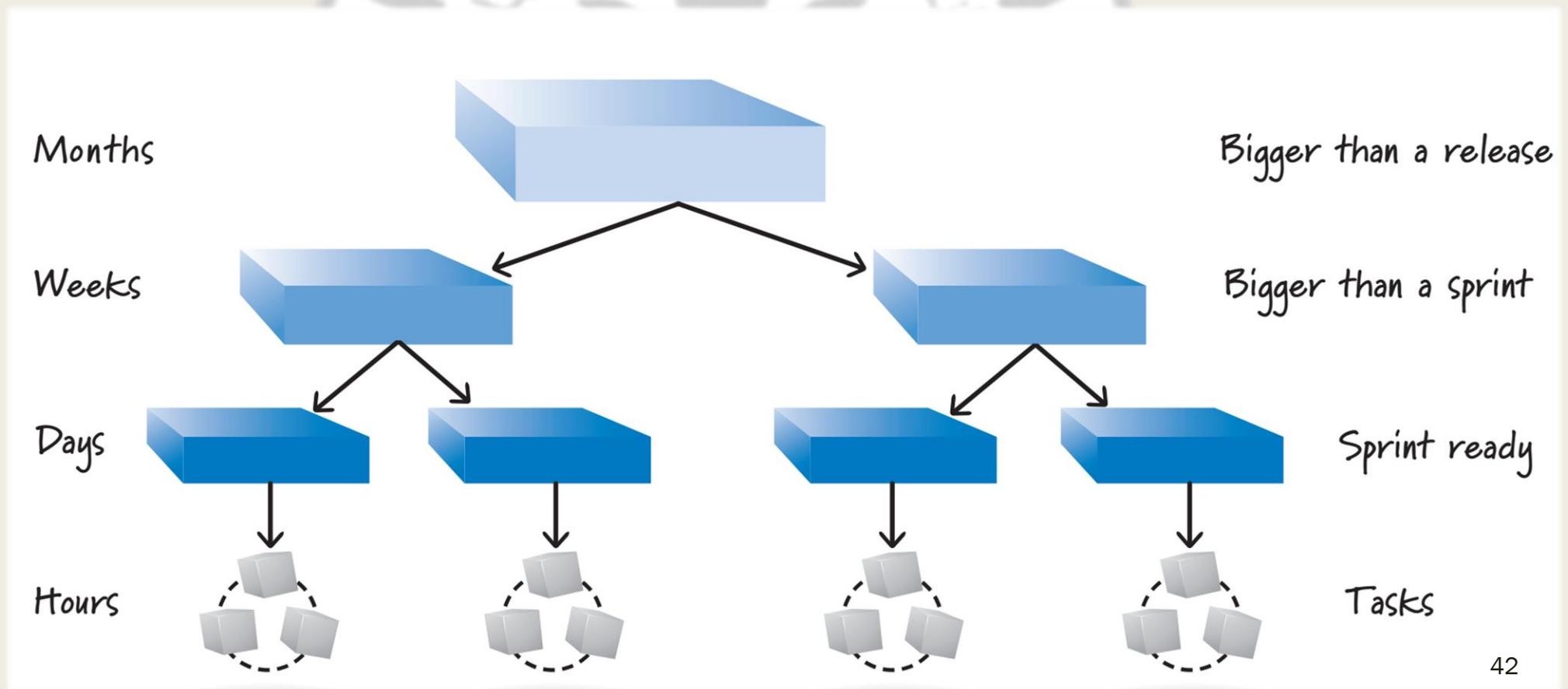
- Small stories used at the sprint level are too small and too numerous to support higher-level product and release planning. At these levels we need fewer, less detailed, more abstract items.
- Otherwise, we'll be mired in a swamp of mostly irrelevant detail.
- Imagine having 500 very small stories and being asked to provide an executive-level description of the proposed product to secure your funding. Or try to prioritize among those 500 really small items to define the next release.

# Level of Detail(Cnt'd)

تنهای داشتن داستانهای کوچک، از مزایای اصلاح تدریجی نیازمندیها بر مبنای کافی و به موقع جلوگیری میکند.

- Having only small stories precludes the benefit of progressively refining requirements on a just enough, just-in-time basis.
- Fortunately, user stories can be written to capture customer and user needs at various levels of abstraction.

# User story abstraction hierarchy



# Stories at multiple levels of abstraction- epic

- The **largest** would be stories that are **a few to many months** in size and might **span an entire release** or **multiple releases**. Many people refer to these as **epics**.
- **Epics** are **helpful** because they give **a very big-picture, high-level overview** of what is desired.
- We would never move an epic into a sprint for development because it is **too big** and **not very detailed**.
- Instead, **epics** are **excellent placeholders** for a **large collection** of more **detailed stories** to be created at an **appropriate future time**.

# Example epic

## Preference Training Epic

As a typical user I want to train the system on what types of product and service reviews I prefer so it will know what characteristics to use when filtering reviews on my behalf.

# Stories at multiple levels of abstraction- feature and story

- The next-size stories are often on the **order of weeks** in size and therefore **too big for a single sprint**. Some teams might call these **features**.
- The **smallest forms of user stories** are **stories**.
- Call these stories either **sprintable stories** or **implementable stories** to indicate that they are on the **order of days** in **size** and therefore **small enough** to fit into a sprint and be implemented.

# Stories at multiple levels of abstraction-task

- Tasks are the layer below stories, typically worked on by only one person, or perhaps a pair of people.
- Tasks typically require hours to perform. When we go to the task layer, we are specifying how to build something instead of what to build (represented by epics, features, and stories).
- Tasks are not stories, so we should avoid including task-level detail when writing stories.

# Stories at multiple levels of abstraction

- It really **doesn't matter** what **labels** you use as long as you use them **consistently**.
- What does matter is **recognizing** that **stories** can exist **at multiple levels of abstraction**, and that doing so nicely supports our efforts to **plan at multiple levels of abstraction** and to **progressively refine** **big items** into small items over time.

# Reference

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

# Requirements and User Stories(III)

Dr. Elham Mahmoudzadeh

Isfahan University of Technology

[mahmoudzadeh@iut.ac.ir](mailto:mahmoudzadeh@iut.ac.ir)

2022

# INVEST in Good Stories

- How do we know if the **stories** that we have written **are good stories?**
- **Six criteria** (summarized by the acronym INVEST) that have been proved **useful** when evaluating whether our stories are fit for their intended use or require **some additional work.**
- The **INVEST criteria** are ***Independent, Negotiable, Valuable, Estimatable, Small*** (sized appropriately), and ***Testable.***

چگونه بفهمیم داستان هایی که نوشته ایم داستان های خوبی هستند؟  
شش معیار (خلاصه شده با مخفف INVEST) که هنگام ارزیابی اینکه آیا داستانهای ما برای استفاده مورد نظرشان مناسب هستند یا نیاز به کار اضافی دارند، مفید هستند.

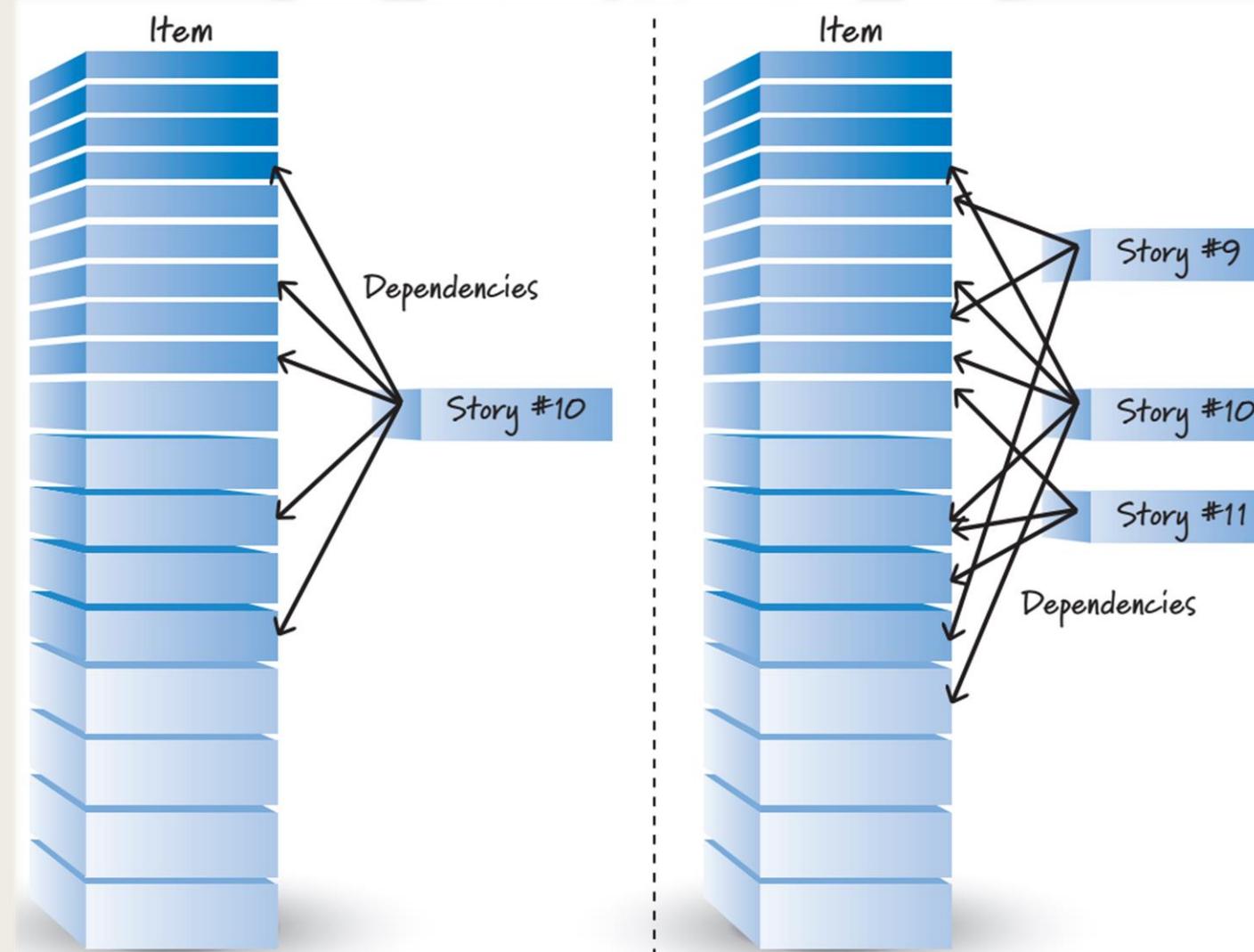
معیارهای INVEST متنقل، قابل مذاکره، ارزشمند، قابل تخمین، کوچک (اندازه مناسب) و قابل آزمایش هستند.

# INVEST: Independent

- User stories should be *independent* or at least only loosely coupled with one another.
- Stories that exhibit a high degree of interdependence complicate estimating, prioritizing, and planning.
- When applying the *independent* criteria, the goal is not to eliminate all dependencies, but instead to write stories in a way that minimizes dependencies.

داستان های کاربر باید مستقل باشند یا حداقل فقط به طور ضعیف با یکدیگر مرتبط باشند.  
داستان هایی که میزان بالایی از وابستگی متقابل را نشان می دهند، تخمین، اولویت بندی و برنامه ریزی را پیچیده می کنند.  
لنگام اعمال معیارهای مستقل، هدف حذف همه وابستگی ها نیست، بلکه در عوض نوشتن داستان به گونه ای است که وابستگی ها را به حداقل برساند.

# Highly dependent stories



# Independent(Cnt'd)

در سمت چپ شکل، داستان شماره 10 به بسیاری از داستان های دیگر بستگی دارد. قبل از اینکه بتوانیم روی داستان شماره 10 کار کنیم، ابتدا باید تمام داستان های وابسته را توسعه دهیم. در این مورد ممکن است چندان بد نباشد.

با این حال، همانطور که در سمت راست شکل نشان داده شده است، تصور کنید که داستانهای مختلفی با درجه بالایی از وابستگی متقابل دارید.

تلash برای تعیین اولویت‌بندی همه این داستانها و تصمیمگیری برای کار روی کدام داستانها در یک مسابقه سرعت دشوار است.

- On the left side of Figure, story #10 depends on many other stories.
- Before we can work on story #10, we must first develop all of the dependent stories. In this single case that might not be so bad.
- However, imagine that you have many different stories with a high degree of interdependence, as illustrated by the right side of Figure.
- Trying to determine how to prioritize all of these stories and deciding which stories to work on in a sprint would be difficult to say.

# INVEST: Negotiable

جزئیات داستان ها نیز باید قابل مذاکره باشد.

داستان ها یک قرارداد مكتوب در قالب یک سند الزامات اولیه نیستند.

در عوض، استوری ها مکان هایی برای مکالمات هستند که در آن جزئیات مذاکره می شود.

داستان های خوب به وضوح ماهیت عملکرد تجاری مورد نظر و دلیل مطلوب بودن آن را نشان می دهد.

آنها فضایی را برای مالک محصول، سهامداران و تیم برای مذاکره درباره جزئیات باقی می گذارند.

- The **details of stories** should also be **negotiable**.
- Stories are not a written contract in the form of an up-front requirements document.
- Instead, stories are **placeholders for the conversations** where the **details will be negotiated**.
- Good stories clearly capture the **essence** of what **business functionality** is desired and **why it is desired**.
- They **leave room** for the **product owner**, the **stakeholders**, and the **team** to negotiate the details.

# Negotiable(Cnt'd)

این قابلیت مذاکره به همه افراد درگیر کمک می کند تا از ذهنیت اشاره انگشت ما به ما در مقابل آنها که در اسناد الزامات اولیه دقیق رایج است اجتناب کنند. قطی داستانها قابل مذاکره هستند، توسعه‌دهندگان واقعاً نمیتوانند بگویند: «هی، اگر میخواستی، باید آن را در سند میدادی»، زیرا جزئیات فرار است با توسعه‌دهندگان مذاکره شود. و تجار واقعاً نمیتوانند بگویند: «هی، شما بدیهی است که سند الزامات را متوجه نشدید، زیرا چیز اشتباهی ساخته‌اید»، زیرا افراد تجاری مکرراً با توسعه‌دهندگان گفتگو خواهند کرد تا از وضوح مشترک اطمینان حاصل کنند.

- This negotiability helps everyone involved **avoid the us-versus-them, finger pointing** mentality that is commonplace with detailed up-front requirements documents.
- When **stories are negotiable**, **developers** can't really say, "Hey, if you wanted it, you should have put it in the document," because the details are going to be negotiated with the developers.
- And the **business people** can't really say, "Hey, you obviously didn't understand the requirements document because you built the wrong thing," because the business people will be in frequent dialogue with the developers to make sure there **is shared clarity**.

# Negotiable(Cnt'd)

نوشتن داستانهای قابل مذاکره با روشن ساختن ضروری بودن گفتگو از مشکلات مرتبط با الزامات دقیق اولیه جلوگیری میکند.  
یک مثال رایج از مواردی که قابلیت مذاکره نقض می شود زمانی است که مالک محصول به تیم می گوید چگونه یک داستان را پیاده سازی کنند.  
داستان ها باید درباره چیستی و چرا باید باشند، نه چگونه.

- Writing **negotiable stories** avoids the **problems** associated with **up-front detailed requirements** by making it **clear** that a dialogue is **necessary**.
- A common example of where **negotiability** is **violated** is when the product owner tells the team *how* to implement a story.
- Stories should be about **what** and **why**, not **how**.

# Negotiable(Cnt'd)

با این حال، موقعي وجود دارد که نحوه ساخت یک چیز واقعاً برای صاحب محصول مهم است. به عنوان مثال، ممکن است یک الزام قانونی برای توسعه یک ویژگی به روشي خاص وجود داشته باشد، یا ممکن است یک محدودیت تجاری برای استفاده از یک فناوری خاص وجود داشته باشد. در چنین مواردی، داستان ها کمتر قابل مذاکره خواهند بود، زیرا برخی از جنبه های "چگونه" مورد نیاز است.

همه داستان ها کاملاً قابل مذاکره نیستند، اما بیشتر داستان ها باید چنین باشند.

- There are **times**, however, when **how something is built** is actually **important to the product owner**. For example, there might be a regulatory obligation to develop a feature in a particular way, or there might be a **business constraint** directing the use of a **specific technology**. In such cases the stories will be a bit **less negotiable** because some aspect of the “**how**” is required.

Not all stories are fully negotiable, but most stories should be.

# INVEST: Valuable

داستان ها باید برای مشتری، کاربر یا هر دو ارزشمند باشند.  
مشتریان (یا انتخاب کنندگان) محصول را انتخاب کرده و هزینه آن را پرداخت می کنند. کاربران در واقع از محصول استفاده می کنند.  
اگر داستانی برای هیچکدام ارزشی نداشته باشد، در بک لاغ محصول قرار نمیگیرد.  
یا داستان را بازنویسی میکردیم تا برای مشتری یا کاربر ارزشمند باشد، یا آن را کنار میگذاشتیم.  
داستان هایی که برای توسعه دهنده ارزشمند هستند اما برای مشتریان یا کاربران ارزش آشکاری ندارند چطور؟ آیا داشتن های فنی خوب است؟

- Stories need to be **valuable** to a customer, user, or **both**.
- **Customers** (or choosers) **select and pay for the product**. **Users** actually **use the product**.
- If a **story isn't valuable** to either, it **doesn't belong** in the product backlog.
- We would either **rewrite the story** to make it valuable to a customer or user, or we would **just discard it**.
- How about **stories** that are **valuable to the developers** but aren't of obvious value to the customers or users? **Is it OK to have technical stories**.

# Example technical story

## Migrate to New Version of Oracle

As a developer I want to migrate the system to work with the latest version of the Oracle DBMS so that we are not operating on a version that Oracle will soon retire.

# Technical stories

- The fundamental problem with technical stories is that the product owner might not perceive any value in them, making it difficult if not impossible to prioritize them against business-valuable stories.
- For a technical story to exist, the product owner should understand why he is paying for it and therefore what value it will ultimately deliver.

# Technical stories(Cnt'd)

در مورد داستان «مهاجرت به نسخه جدید اوراکل»، مالک محصول ممکن است در ابتدا متوجه نشود که چرا تغییر پایگاه داده ارزشمند است. با این حال، هنگامی که تیم خطرات ادامه توسعه بر روی یک نسخه پشتیبانیشده از پایگاه داده را توضیح میدهد، مالک محصول ممکن است تصمیم بگیرد که انتقال پایگاههای داده به اندازه‌های ارزشمند است که ساخت برخی ویژگیهای جدید را تا زمان انجام انتقال به تعویق بیندازد. با درک ارزش، صاحب محصول می‌تواند با داستان فنی مانند هر داستان ارزشمند تجاری دیگری رفتار کند و مبادلات آگاهانه انجام دهد. در نتیجه، این داستان فنی ممکن است در بک لاغ محصول گنجانده شود.

- In the case of the “Migrate to New Version of Oracle” story, the product owner might not initially understand why it is valuable to change databases.
- However, once the team explains the risks of continuing to develop on an unsupported version of a database, the product owner might decide that migrating databases is valuable enough to defer building some new features until the migration is done.
- By understanding the value, the product owner can treat the technical story like any other business-valuable story and make informed trade-offs.
- As a result, this technical story might be included in the product backlog.

# Valuable(Cnt'd)

در عمل، اکثر داستان های فنی نباید در بک لاغ محصول گنجانده شوند.  
در عوض، این نوع داستانها باید وظایفی باشند که با انجام داستانهای ارزشمند تجاری مرتبط هستند.  
اگر تیم توسعه تعریف قوی ای از انجام شده داشته باشد، نیازی به نوشتن داستان هایی مانند این نیست.

- In practice, most technical stories should not be included in the product backlog.
- Instead, these types of stories should be tasks associated with getting business valuable stories done.
- If the development team has a strong definition of done, there should be no need to write stories like these.

همه داستانهای موجود در بک لاغ باید از دیدگاه صاحب محصول ارزشمند (ارزش سرمایه‌گذاری در آن) باشند، که نمایانگر دیدگاه مشتری و کاربر است.

همه داستان‌ها مستقل نیستند و همه داستان‌ها کاملاً قابل مذاکره نیستند، اما همه آنها باید ارزشمند باشند.

All stories in the backlog must be valuable (worth investing in) from the product owner's perspective, which represents the customer and user perspectives.

Not all stories are independent, and not all stories are fully negotiable, but they all must be valuable.

# INVEST: Estimatable

- Stories should be **estimatable** by the **team** that will **design**, **build**, and **test them**.
- Estimates provide an **indication of the size** and therefore the **effort** and **cost** of the stories (bigger stories require more effort and therefore cost more money to develop than smaller stories).

داستان ها باید توسط تیمی که آنها را طراحی، ساخت و آزمایش می کند، قابل ارزیابی باشد.  
تخمین ها نشانی از اندازه و در نتیجه تلاش و هزینه داستان ها را ارائه می دهند (داستان های بزرگتر به تلاش بیشتری نیاز دارند و بنابراین هزینه بیشتری برای توسعه نسبت به داستان های کوچکتر هزینه می کنند).

# Estimatable(Cnt'd)

دانستن اندازه یک استوری اطلاعات عملی را در اختیار تیم اسکرام قرار می دهد.  
صاحب محصول باید هزینه یک داستان را بداند تا اولویت نهایی آن را در پس مانده محصول تعیین کند.

از سوی دیگر، تیم اسکرام میتواند از روی حجم داستان تعیین کند که آیا نیاز به اصلاح یا تفکیک اضافی است. یک داستان بزرگ که قصد داریم به زودی روی آن کار کنیم، باید به مجموعه ای از داستان های کوچکتر تقسیم شود.

- Knowing a story's size provides actionable information to the Scrum team.
- The product owner, needs to know the cost of a story to determine its final priority in the product backlog.
- The Scrum team, on the other hand, can determine from the size of the story whether additional refinement or disaggregation is required. A large story that we plan to work on soon will need to be broken into a set of smaller stories.

# Estimatable(Cnt'd)

اگر تیم نتواند اندازه یک داستان را اندازه‌گیری کند، داستان یا آنقدر بزرگ یا مبهم است که اندازه آن نمیتواند باشد، یا اینکه تیم داشت کافی برای تخمین اندازه ندارد. اگر خیلی بزرگ باشد، تیم باید با مالک محصول کار کند تا آن را به داستانهای قابل مدیریتتری تقسیم کند. اگر تیم فاقد دانش باشد، نوعی فعالیت اکتشافی برای به دست آوردن اطلاعات مورد نیاز خواهد بود.

- If the team isn't able to size a story, the story is either just too big or ambiguous to be sized, or the team doesn't have enough knowledge to estimate a size.
- If it's too big, the team will need to work with the product owner to break it into more manageable stories.
- If the team lacks knowledge, some form of exploratory activity will be needed to acquire the information.

# INVEST: Sized Appropriately (Small)

داستان ها باید برای زمانی که قصد داریم روی آنها کار کنیم اندازه مناسبی داشته باشند.

داستان هایی که در دوی سرعت روی آنها کار می شود باید کوچک باشند.

اگر یک اسپرینت چند هفتاهی انجام میدهیم، میخواهیم روی چندین داستان کار کنیم که هر کدام چند روز هستند.

اگر دو هفته ای دوی سرعت داشته باشیم، داستانی به اندازه دو هفته نمی خواهیم، زیرا خطر کامل نشدن داستان بسیار زیاد است.

- Stories should be sized appropriately for when we plan to work on them.
- Stories worked on in sprints should be small.
- If we're doing a several-week sprint, we want to work on several stories that are each a few days in size.
- If we have a two-week sprint, we don't want a two-week-size story, because the risk of not finishing the story is just too great.

# Sized Appropriately (Cnt'd)

بنابراین در نهایت ما به داستان های کوچک نیاز داریم، اما فقط به این دلیل که یک داستان بزرگ است، به این معنی نیست که بد است. بباید بگوییم که ما یک داستان حماسی داریم که قصد نداریم یک سال دیگر روی آن کار کنیم. مسلماً آن داستان برای زمانی که قصد داریم روی آن کار کنیم، اندازه مناسبی دارد. در واقع، اگر امروز وقت صرف کنیم تا آن حماسه را به مجموعهای از داستانهای کوچکتر تبدیل کنیم، به راحتی میتوانیم وقتمن را تلف کنیم. البته، اگر حماسهای داشته باشیم که بخواهیم در اسپرینت بعدی روی آن کار کنیم، اندازه آن مناسب نیست و باید کارهای بیشتری انجام دهیم تا آن را به اندازه کاهش دهیم. هنگام اعمال این معیار باید در نظر بگیرید که چه زمانی روی داستان کار می شود.

- So ultimately we need **small stories**, but just because a story is large, that doesn't mean it's bad.
- Let's say we have an **epic-size story** that we **aren't planning** to work on for another year. Arguably that story is sized appropriately for **when we plan to work on it**.
- In fact, if we **spent time today breaking** that epic down into a collection of smaller stories, it could easily be **a complete waste of our time**.
- Of course, if we have an epic that we want to **work on in the next sprint**, it's not sized appropriately and we have more work to do to bring it down to size.
- You must consider **when** the **story will be worked on** **when** applying this criterion.

# INVEST: Testable

داستانها باید بهصورت دودویی قابل آزمایش باشند—آنها یا در آزمونهای مرتبط خود موفق میشوند یا شکست میخورند.

قابل آزمایش بودن به معنای داشتن معیارهای پذیرش خوب (مریبوط به شرایط رضایت) مرتبط با داستان است که جنبه "تأیید" داستان کاربر است.

بدون معیارهای قابل آزمایش، چگونه میتوانیم بفهمیم که داستان در پایان اسپرینت انجام شده است؟ همچنین، از آنجایی که این تستها اغلب جزئیات مهم داستان را ارائه میدهند، ممکن است قبل از اینکه تیم حتی بتواند داستان را تخمين بزند، به آنها نیاز باشد.

- Stories should be **testable** in a **binary way**—they either **pass** or **fail** their associated tests.
- Being **testable** means **having good acceptance criteria** (related to the conditions of satisfaction) associated with the story, which is the "**confirmation**" aspect of a user story.
- Without **testable criteria**, how would we know if the story is done at the end of the sprint?
- Also, because these **tests** frequently provide important story details, they may be **needed** before the team can even estimate the story.

# Testable(Cnt'd)

ممکن است همیشه آزمایش یک داستان ضروری یا ممکن نباشد.

به عنوان مثال، داستانهایی با اندازه حماسی احتمالاً تستهای مرتبط با آنها ندارند، و به آنها هم نیاز ندارند (ما مستقیماً حماسهها را نمیسازیم).

همچنین، گاهی ممکن است داستانی وجود داشته باشد که صاحب محصول آن را ارزشمند بداند، اما ممکن است راه عملی برای آزمایش آن وجود نداشته باشد.

اینها به احتمال زیاد الزامات غیر کاربردی هستند، مانند "به عنوان یک کاربر، من می خواهم سیستم 99.999٪ آپتايم داشته باشد." اگرچه معیارهای پذیرش ممکن است واضح باشند، اما ممکن است هیچ مجموعه ای از آزمایشات وجود نداشته باشد که بتوان در هنگام تولید سیستم اجرا کرد که بتواند ثابت کند که این سطح از زمان به روز برآورده شده است.

- It may **not always** be **necessary** or **possible** to test a story.
- For example, **epic-size stories** probably **don't have tests associated** with them, nor do they need them (we don't directly build the epics).
- Also, on occasion there might be **a story** that the product owner deems valuable, yet there might **not be a practical way to test it**.
- These are more likely to be **nonfunctional requirements**, such as "As a user I want the system to have **99.999% uptime**." Although the acceptance criteria might be clear, there may be **no set of tests** that **can be run** when the system is put into production that can **prove** that this level of uptime has been met.

# Nonfunctional Requirements

- Represent system-level constraints.

## Internationalization

As a user I want an interface in English, a Romance language, and a complex language so that there is high statistical likelihood that it will work in all 70 required languages.

## Web Browser Support

System must support IE8, IE9, Firefox 6, Firefox 7, Safari 5, and Chrome 15.

# Nonfunctional Requirements(Cnt'd)

- As system-level constraints, nonfunctional requirements are important because they affect the design and testing of most or all stories in the product backlog.
- For example, having a “Web Browser Support” nonfunctional requirement would be common on any website project. When the team develops the website features, it must ensure that the site features work with all of the specified browsers.

به عنوان محدودیتهای سطح سیستم، نیازمندیهای غیر عملکردی مهم هستند زیرا بر طراحی و آزمایش بیشتر یا همه داستانها در بک لاگ محصول تأثیر می‌گذارند.  
به عنوان مثال، داشتن یک نیاز غیر کاربردی "پشتیبانی از مرورگر وب" در هر پروژه وب سایتی رایج است. وقتی تیم ویژگی های وب سایت را توسعه می دهد، باید اطمینان حاصل کند که ویژگی های سایت با همه مرورگرهای مشخص شده کار می کند.

# Nonfunctional Requirements(Cnt'd)

تیم همچنین باید تصمیم بگیرد که چه زمانی همه مرورگرها را آزمایش کند.

اگر تیم الزامات غیر کاربردی «پشتیبانی از مرورگر وب» را در تعریف انجام شده لحاظ کند، تیم باید هر ویژگی جدید اضافه شده در اسپرینت را با همه مرورگرهای فهرست شده آزمایش کند. اگر با همه آنها کار نکرد، داستان تمام نشده است.

سعی کنید تا جایی که ممکن است بسیاری از الزامات غیر کاربردی را در تعاریف انجام شده قرار دهید.

انتظار برای آزمایش الزامات غیر عملکردی تا اوخر تلاش توسعه، دریافت بازخورد سریع در مورد ویژگی های عملکرد حیاتی سیستم را به تعویق می اندازد.

- The team must also decide when to test all of the browsers.
- If the team includes the “Web Browser Support” nonfunctional requirement in the definition of done, the team will have to test any new features added in the sprint with all of the listed browsers. If it doesn’t work with all of them, the story isn’t done.
- Try to include as many of the nonfunctional requirements in their definitions of done as they possibly can.
- Waiting to test nonfunctional requirements until late in the development effort defers getting fast feedback on critical system performance characteristics.

# Knowledge-Acquisition Stories

گاهی اوقات ما نیاز داریم که یک آیتم بک لاگ محصول ایجاد کنیم که بر کسب دانش تمرکز دارد. شاید ما دانش قابل بهره برداری کافی در مورد محصول یا فرآیند ساخت محصول برای حرکت رو به جلو نداشته باشیم. چنین کاوشی با نام های بسیاری شناخته می شود: نمونه اولیه، اثبات مفهوم، آزمایش، مطالعه و غیره. همه آنها اساساً فعالیت های اکتشافی هستند که شامل خرید اطلاعات می شوند. سعی کنید از یک داستان کاربر به عنوان مکان نگهدار برای کار کاوش استفاده کنید.

- Sometimes we need to **create a product backlog item** that focuses on **knowledge acquisition**.
- Perhaps we **don't have enough exploitable knowledge** about the product or the **process of building the product** to move forward.
- Such **exploration** is known by many names: **prototype**, **proof of concept**, **experiment**, **study**, and so on.
- They are all basically exploration activities that involve **buying information**.
- Try to **employ a user story** as the **placeholder** for the **exploration work**.

# Knowledge-acquisition story

## Filtering Engine Architecture Eval

As a developer I want to prototype two alternatives for the new filtering engine so that I know which is a better long-term choice.

## Conditions of Satisfaction

- Run speed test on both prototypes.
- Run scale test on both prototypes.
- Run type test on both prototypes.
- Write short memo describing experiments, results, and recommendations.

# Example

تیم می خواهد دو معماری ممکن را برای موتور فیلتر جدید ارزیابی کند.  
پیشنهاد میکند هر دو معماری را نمونهسازی کند و سپس تستهای سرعت، مقیاس و نوع را در برابر هر دو نمونه اولیه اجرا کند.

قابل تحویل از فعالیت نمونه سازی یک یادداشت کوتاه خواهد بود که آزمایش های انجام شده، نتایج به دست آمده و توصیه های تیم را برای چگونگی ادامه شرح می دهد.

- Team wants to evaluate two possible architectures for the new filtering engine.
- It is proposing to prototype both architectures and then run speed, scale, and type tests against both prototypes.
- The deliverable from the prototyping activity will be a short memo that describes the experiments that were performed, the results that were obtained, and the team's recommendation for how to proceed.

# Knowledge-acquisition story (Cnt'd)

این داستان کسب دانش خاص شبیه یک داستان فنی است.

ارزش تجاری هر داستان فنی باید برای صاحب مخصوص قابل توجیه باشد.

از آنجایی که صاحبان مخصوص به لحاظ اقتصادی فکر می کنند، نیاز به توجیه اقتصادی برای انجام این کار نمونه سازی وجود دارد.

احتمالاً یک استدلال فنی قانعکننده برای انجام یک داستان کسب دانش وجود دارد، زیرا تیم معمولاً تا زمانی که دانش تولید شده توسط داستان را بدست نیاورد، از پیشرفت رو به جلو چلوگیری میکند.

- This specific knowledge-acquisition story looks like a technical story.
- The business value of any technical story has to be justifiable to the product owner.
- Because product owners think in economic terms, there needs to be an economic justification for doing this prototyping work.
- There is likely a compelling technical argument for doing a knowledge-acquisition story because the team is typically blocked from making forward progress until it has the knowledge produced by the story.

# Knowledge-acquisition story (Cnt'd)

- The question for the Scrum team is whether the value of the acquired information exceeds the cost of getting it.
- Here is how a Scrum team could approach answering that question.
- First, we need to know the cost of the prototyping.
- No good product owner will authorize unbounded exploration.

سؤالی که برای تیم اسکرام وجود دارد این است که آیا ارزش اطلاعات به دست آمده از هزینه دریافت آن بیشتر است یا خیر.  
در اینجا آمده است که چگونه یک تیم اسکرام می تواند به این سوال پاسخ دهد.  
ابتدا باید هزینه نمونه سازی را بدانیم.  
هیچ صاحب محصول خوبی اجازه اکتشاف نامحدود را نمی دهد.

# Knowledge-acquisition story (Cnt'd)

بن تیم ممکن است تا زمانی که یک تصمیم معماری گرفته نشود، نتواند به سوالات خاصی پاسخ دهد، اما باید بتواند به این سوال پاسخ دهد که چقدر تلاش می کند تا اطلاعات لازم برای تصمیم گیری معماری را خریداری کند.

بنابراین، از تیم میخواهیم که داستان نمونهسازی را اندازهگیری کند.  
باید بگوییم که برآورد اندازه نشان میدهد که تیم کامل باید برای یک سرعت روی داستان کار کند. ما می دانیم که چه کسی در تیم است و طول دوی سرعت، بنابراین ما همچنین از هزینه به دست آوردن اطلاعات مطلع هستیم. فرض کنید 10 هزار دلار است.

- The **team** might **not be able** to **answer** particular **questions** until an **architectural decision has been made**, but it must be able to answer the question of **how much effort** it wants to spend **to buy the information** necessary to make the architectural decision.
- So, we ask the team to **size the prototyping story**.
  - Let's say that the **size estimate** indicates that the **full team** would need to work on the story for **one sprint**. We know who **is on the team** and the **length of the sprint**, so we also know the cost of acquiring the **information**. Let's say it is \$10K.

# Knowledge-acquisition story (Cnt'd)

در اینجا یکی از راه هایی است که می توانیم ارزش را تخمین بزنیم. تصور کنید که من یک سکه را برگردانم. اگر بالا بباید، معماری A را انجام خواهیم داد. اگر بالا آمد، معماری B را انجام خواهیم داد.

اکنون از تیم می خواهم هزینه اشتباه را برآورد کند. به عنوان مثال، اگر سکه را برگردانم و سرها بالا بباید و ما شروع به ساختن ویژگی های تجاری بر روی معماری A کنیم، و معماری A رویکرد اشتباهی به نظر برسد، باز کردن تصمیم بد و بازسازی همه چیز چه هزینه ای دارد. بالای معماری ب؟ بباید بگوییم که تیم هزینه را 500 هزار دلار تخمین زده است.

- Here is one way we might **estimate the value**. Imagine that I flip a coin. If it comes up heads, we'll do architecture A; if it comes up tails, we'll do architecture B.
- Now, I ask the team **to estimate the cost of being wrong**. For example, if I flip the coin and it comes up heads and we start building business features on top of architecture A, and architecture A turns out to be the wrong approach, what would be the **cost to unwind the bad decision** and **rebuild everything** on top of architecture B? Let's say the team estimates the cost to be \$500K.

# Knowledge-acquisition story (Cnt'd)

- Now we have enough information to make a sensible economic decision.
- Are we willing to spend \$10K to purchase information that has an expected value of \$250K (half the time we flip the coin we would be correct)?
- Sure, that seems like a sensible business decision.
- Now the product owner can justify why this story is in the backlog.

# Knowledge-acquisition story (Cnt'd)

- What if the team's response to "What would it cost if we were wrong?" is \$15K? In this case it would be a bad decision to do the prototyping story.
- Why spend \$10K to buy information that has an expected value of \$7.5K? We would be better off just flipping the coin (or making an educated guess) and, if we're wrong, simply redoing the work using the other architecture.
- It's an example of what some people call a fail-fast strategy (try something, get fast feedback, and rapidly inspect and adapt).

# Reference

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

# Requirements and User Stories(IV)

Dr. Elham Mahmoudzadeh  
Isfahan University of Technology

[mahmoudzadeh@iut.ac.ir](mailto:mahmoudzadeh@iut.ac.ir)

2022

# Gathering Stories

- How do user stories come into existence?
- Traditional approaches to requirements gathering involve asking the users what they want.
- Users are far better critics than they are authors.
- So, if you ask a user, “What do you want?” she may or may not be able to answer.

# Gathering Stories(Cnt'd)

- A better approach is to involve the users as part of the team that is determining what to build and is constantly reviewing what is being built.
- To promote this level of participation, many organizations prefer to employ user-story-writing workshops as a principal means of generating at least the initial set of user stories.
- Some also employ story mapping to organize and provide a user-centered context to their stories.

# User-Story-Writing Workshop

- The goal is to collectively brainstorm desired business value and create user story placeholders for what the product or service is supposed to do.
- The workshop frequently includes the product owner, ScrumMaster, and development team, in conjunction with internal and external stakeholders.
- Most workshops last anywhere from a few hours to a few days.
- The goal isn't to generate a full and complete set of user stories up front. Instead, the workshop typically has a specific focus.

# User-Story-Writing Workshop(Cnt'd)

- If it is the first workshop, prefer to start by performing user role analysis.
- The goal is to determine the collection of user roles that can be used to populate the user role part of our stories.

# User-Story-Writing Workshop(Cnt'd)

شخصیت ها

افراد نمونه اولیه که نشان دهنده ویژگی های اصلی یک نقش هستند

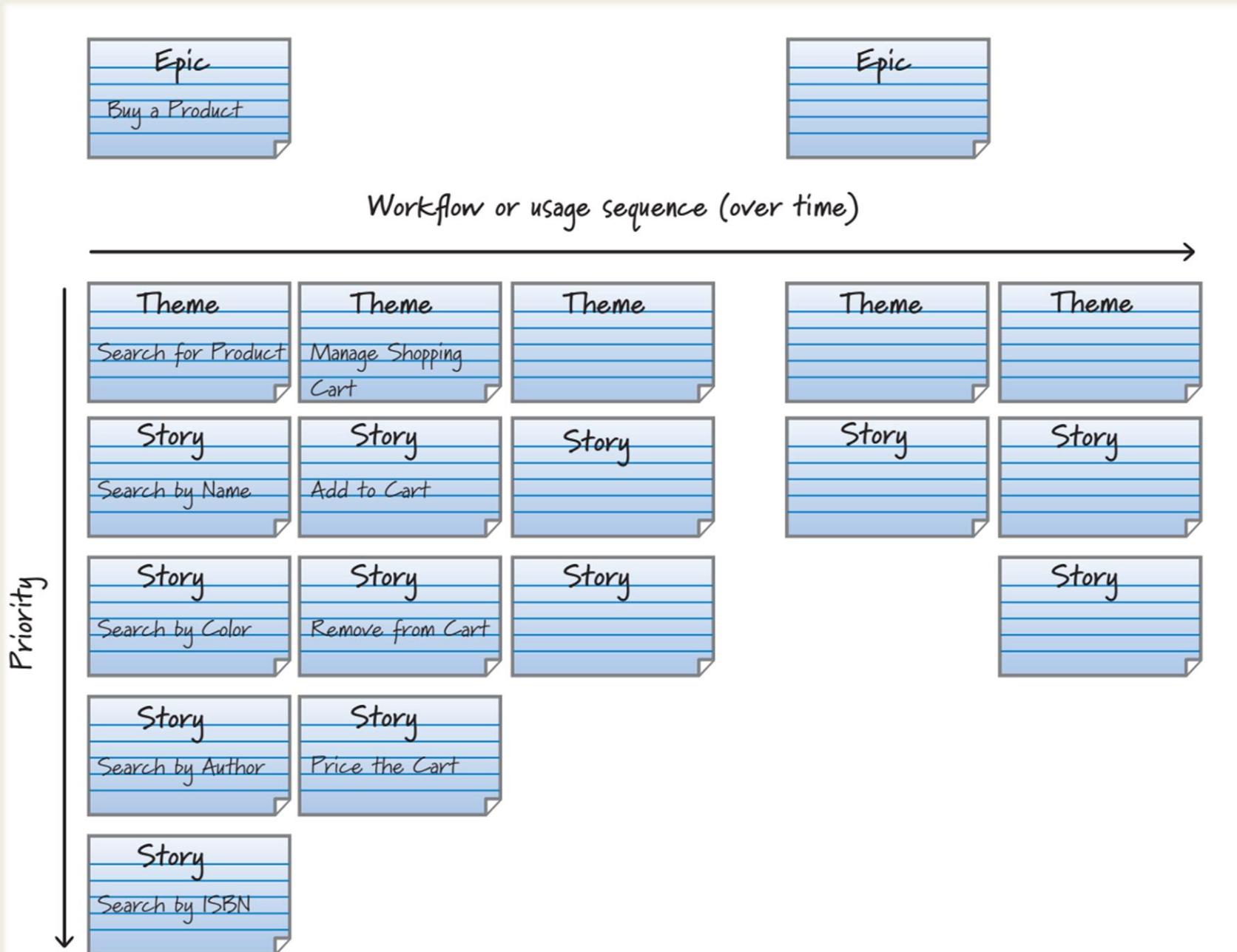
- We might also **have personas**, which are **prototypical individuals** that represent core characteristics of a role.
- For example, “**Lilly**,” along with **her associated description**, might be the persona corresponding to the role of the seven- to nine-year-old female player of a young girl’s video game. Once Lilly is defined, we would write stories with Lilly in the user role position, instead of a more abstract role such as “Young Female Player.”
- For example, “**As Lilly**, I want to select from among many different dresses so that I can customize my avatar to my liking.”

# User-Story-Writing Workshop(Cnt'd)

- During the workshop there is no standard way of generating user stories.
- Some teams prefer to work top-down and others prefer to work bottom-up.
- The top-down approach involves the team starting with a large story (like an epic) and then concentrating its efforts on generating a reasonable collection of smaller stories associated with the epic.
- An alternative is to work more bottom-up and start immediately brainstorming stories that are associated with the next release of an existing system.
- There isn't a right or wrong approach; use whatever approach works well, or switch approaches to get the best of both.

# Story Mapping

- Story mapping is a technique that takes a user-centric perspective for generating a set of user stories.
- The basic idea is to decompose high-level user activity into a workflow that can be further decomposed into a set of detailed tasks.
- At the highest level are the epics, representing the large activities of measurable economic value to the user—for example, the “Buy a Product” epic.



# Story Mapping(Cnt'd)

- Next we think about the sequence or common workflow of user tasks that make up the epic.
- We lay out the themes along a timeline, where themes in the workflow that would naturally occur sooner are positioned to the left of the ones that would occur later.
- For example, the “Search for product” theme would be to the left of the “Manage Shopping Cart” theme.

# Story Mapping(Cnt'd)

- Each theme is then decomposed into a set of implementable stories that are arranged vertically in order of priority.
- Not all stories within a theme need to be included in the same release.
- For example, the “Search by Color” story might not be slated for the first release, whereas the “Search by Name” story probably would be.

# Story Mapping(Cnt'd)

- Story mapping combines the concepts of user-centered design with story decomposition.
- Good story maps show a flow of activities from the users' perspective and provide a context for understanding individual stories and their relationship to larger units of customer value.

# Story Mapping(Cnt'd)

- Even if you don't do formal story mapping, the idea of **using workflows** is helpful during story-writing workshops.
- They focus the **discussion** on **writing stories** within the **context of delivering a complete workflow** of value to the user.
- By having the **workflow context**, it is easier for us to determine if we have missed any important stories associated with the workflow.

# Story Mapping(Cnt'd)

- One difference between traditional story-writing workshops and story mapping is that during the workshop we are primarily focused on generating stories and not so focused on prioritizing them (the vertical position of implementable stories within a story map).
- Story maps provide a two-dimensional view of a product backlog instead of the traditional linear (one dimensional) product backlog representation.



We can **use story mapping** as a **complement to the workshop**,  
as a **technique for helping**  
to **visualize the prioritization of stories**.

# Reference

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

# Sprint Rules(I)

Dr. Elham Mahmoudzadeh

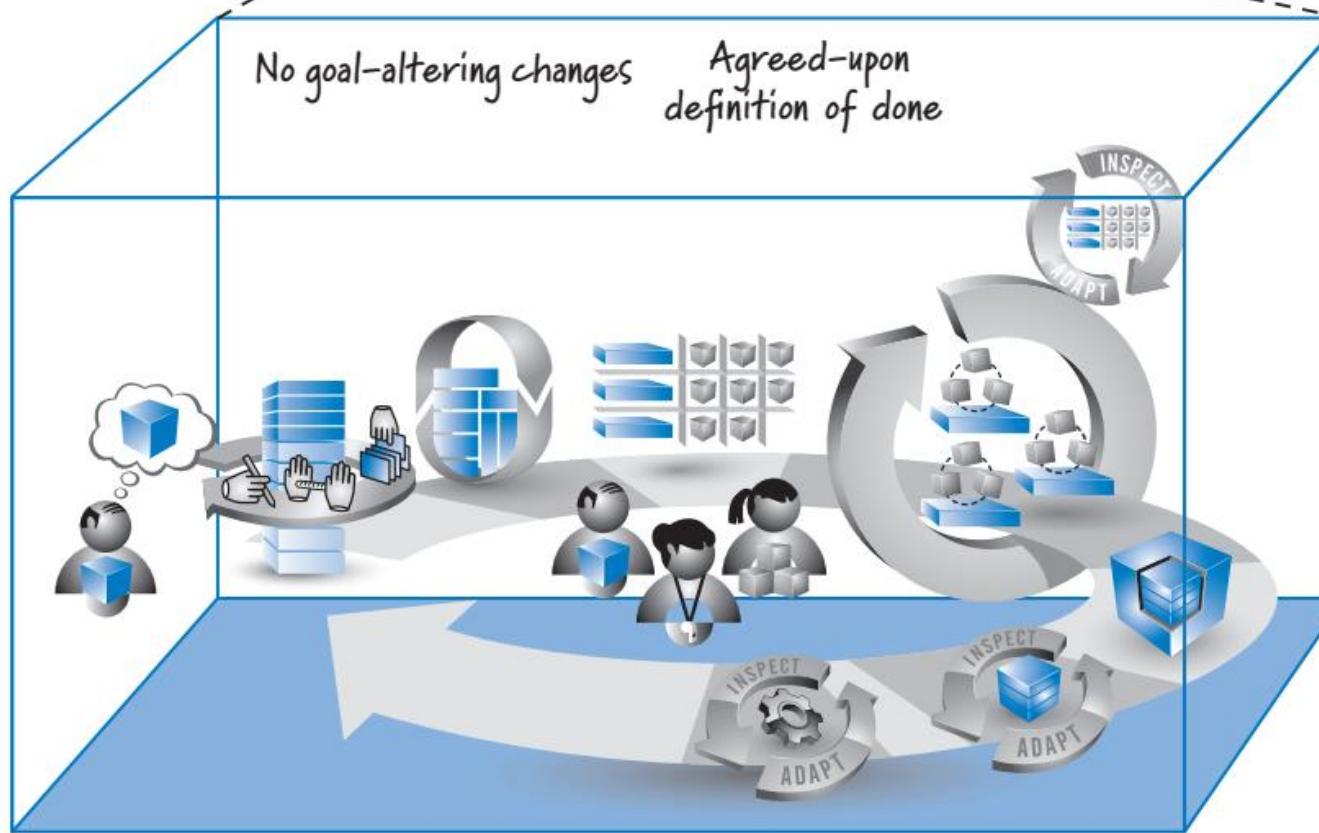
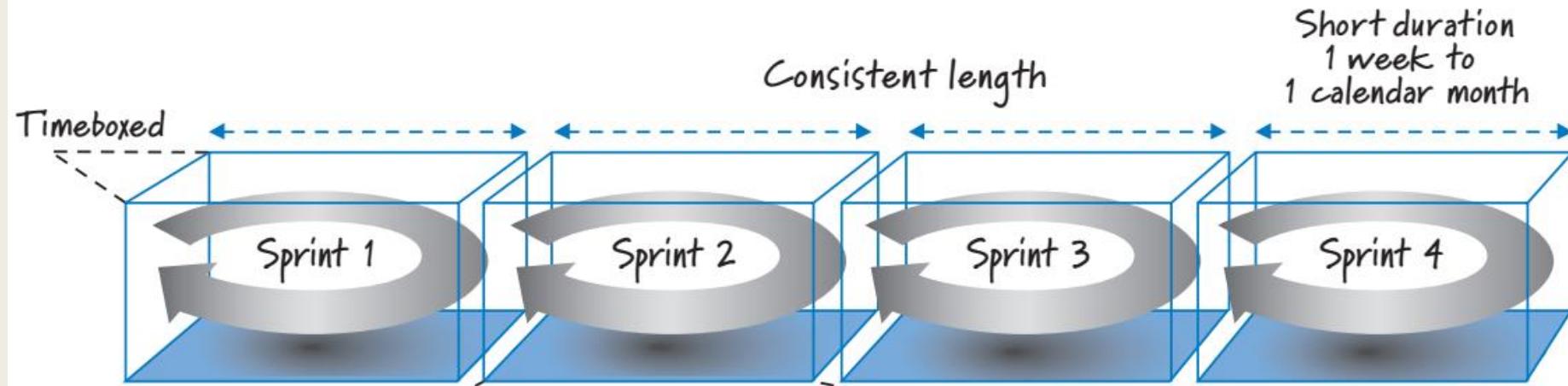
Isfahan University of Technology

[mahmoudzadeh@iut.ac.ir](mailto:mahmoudzadeh@iut.ac.ir)

2023

# Introduction

- Scrum organizes work in iterations or cycles of up to a calendar month called sprints.
- Sprints are the skeleton of the Scrum framework.
- A sprint spans: Sprint Planning, Sprint Execution, Sprint Review, and Sprint Retrospective.



# Sprint Rules

- All sprints are timeboxed: They have fixed start and end dates.
- Sprints must also be short: Between one week and a calendar month.
- Sprints should be consistent in length, though exceptions are permitted under certain circumstances.
- No goal-altering changes in scope or personnel are permitted during a sprint.
- During each sprint, a potentially shippable product increment is completed in conformance with the Scrum team's agreed-upon “definition of done.”

# First Rule: Timeboxing

- Timeboxing: a time-management technique that helps organize the performance of work and manage scope.
- Each sprint takes place in a time frame with specific start and end dates, called a timebox.
- Inside this timebox, the team is expected to work at a sustainable pace to complete a chosen set of work that aligns with a sprint goal.

# Benefits of Timeboxing

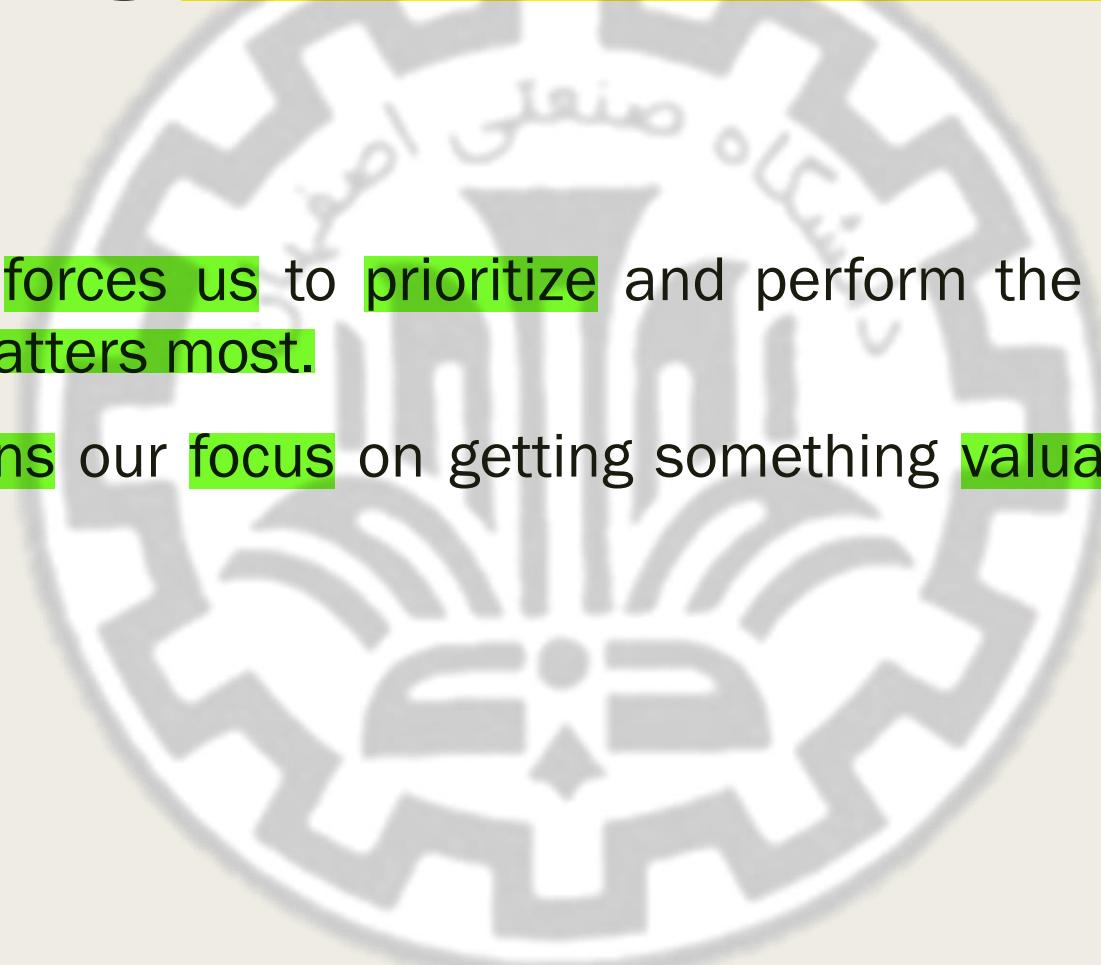
Timeboxing benefits

- Establishes a WIP limit
- Forces prioritization
- Demonstrates progress
- Avoids unnecessary perfectionism
- Motivates closure
- Improves predictability

# Timeboxing Establishes a WIP Limit

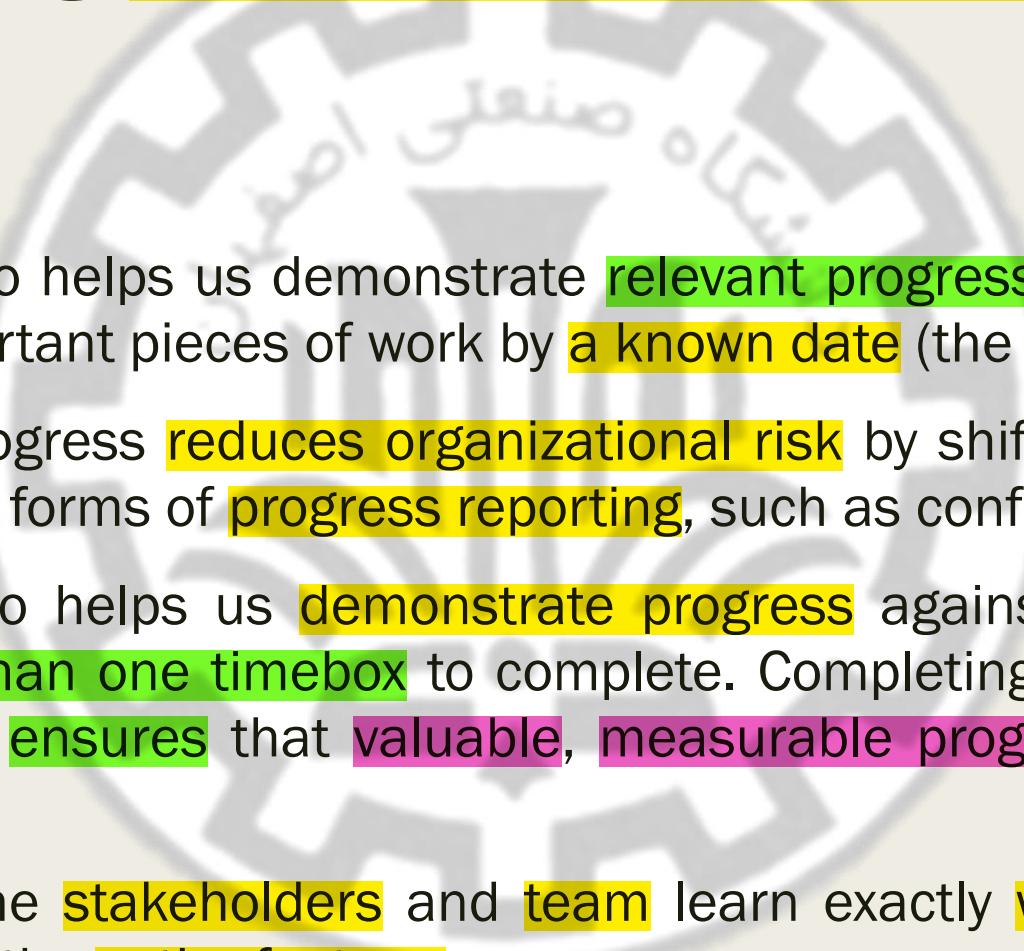
- Timeboxing is a technique for limiting the amount of WIP (work in process).
- WIP represents an inventory of work that is started but not yet finished.
- Failing to properly manage WIP can have serious economic consequences.
- Because the team will plan to work on only those items that it believes it can start and finish within the sprint, timeboxing establishes a WIP limit each sprint.

# Timeboxing **Forces Prioritization**



- Timeboxing forces us to prioritize and perform the small amount of work that matters most.
- This sharpens our focus on getting something valuable done quickly.

# Timeboxing **Demonstrates Progress**



- Timeboxing also helps us demonstrate **relevant progress** by **completing** and **validating** important pieces of work by **a known date** (the end of the sprint).
- This type of progress **reduces organizational risk** by shifting the focus away from **unreliable** forms of **progress reporting**, such as conformance to plan.
- Timeboxing also helps us **demonstrate progress** against **big features** that require **more than one timebox** to complete. Completing some work toward those features **ensures** that **valuable, measurable progress** is being made each sprint.
- It also helps the **stakeholders** and **team** learn exactly **what remains** to be done to deliver the **entire feature**.

# Timeboxing Avoids Unnecessary Perfectionism

- Timeboxing helps avoid unnecessary perfectionism.
- At one time or another we have all spent too much time trying to get something “perfect” or to do “gold plating” when “good enough” would suffice.
- Timeboxing forces an end to potentially unbounded work by establishing a fixed end date for the sprint by which a good solution must be done.

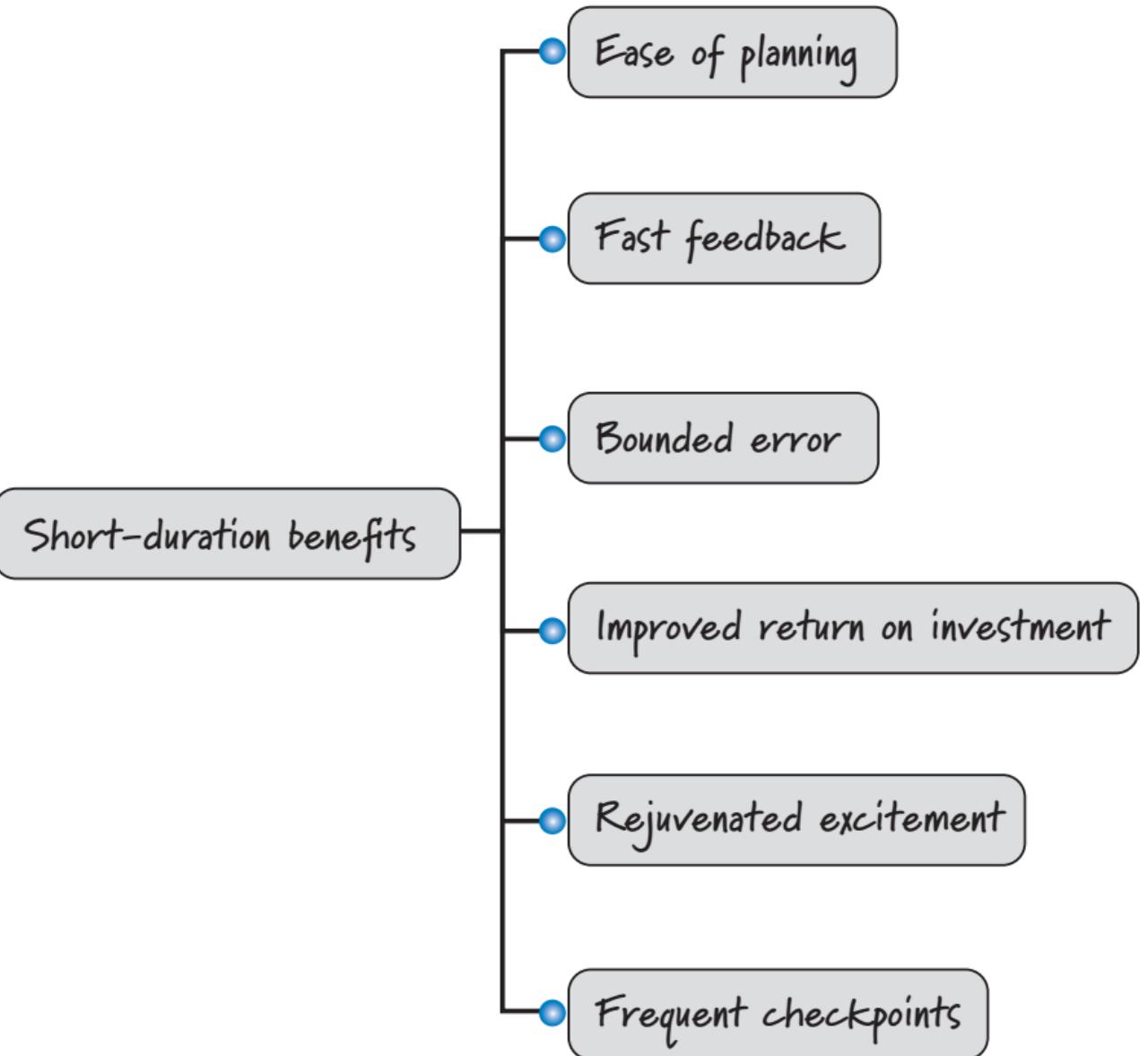
# Timeboxing Motivates Closure

- Things are more likely to get done when teams have a known end date.
- The fact that the end of the sprint brings with it a hard deadline encourages team members to diligently apply themselves to complete the work on time.
- Without a known end date, there is less of a sense of urgency to complete the job.

# Timeboxing Improves Predictability

- Although we can't predict with great certainty exactly the work we will complete a year from now, it is completely reasonable to expect that we can predict the work we can complete in the next short sprint.

# Second Rule: Short Duration



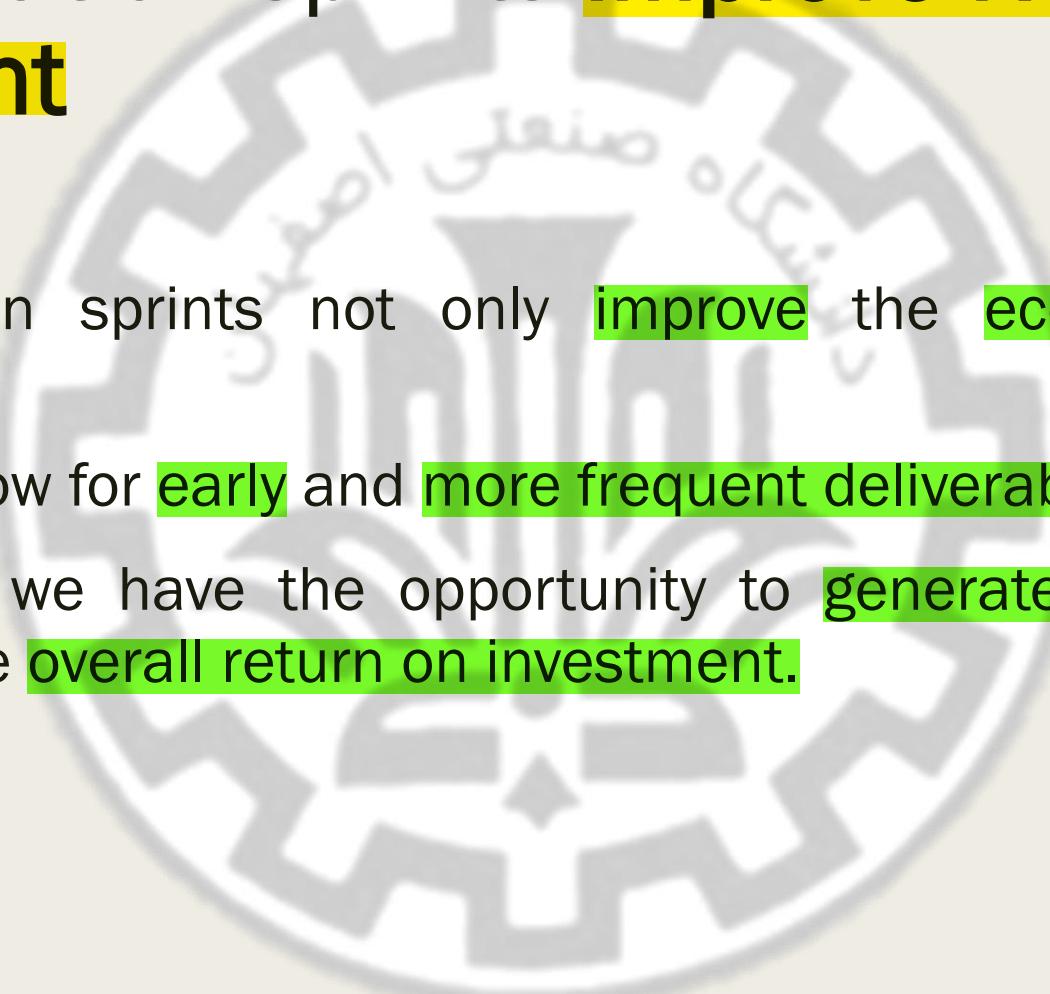
# Short duration sprints yields ease of planning

- Short-duration sprints make it easier to plan.
- It is easier to plan a few weeks' worth of work than six months' worth of work.
- Also, planning on such short time horizons requires far less effort and is far more accurate than longer-horizon planning.

# Short-duration sprints generate fast feedback

- During each sprint we **create working software** and then have the opportunity to **inspect and adapt** what we built and how we built it.
- This **fast feedback** enables us to **quickly prune unfavorable product paths** or **development approaches** before we compound a bad decision with many **follow-on decisions** that are coupled to the bad decision.
- Fast feedback also allows us to **more quickly uncover** and **exploit time-sensitive emergent opportunities**.

# Short Duration sprints **Improve Return on Investment**



- Short-duration sprints not only **improve** the **economics** via fast feedback;
- They also allow for **early** and **more frequent deliverables**.
- As a result, we have the opportunity to **generate revenue sooner**, improving the **overall return on investment**.

# Short Duration sprints **bound error**

- How wrong can we be in a two-week sprint?
- Even if we fumble the whole thing, we have lost only two weeks.
- We insist on short duration sprints because they provide frequent coordination and feedback.
- That way, if we're wrong, at least we're wrong in a small way.

# Short Duration sprints **helps Rejuvenated Excitement**

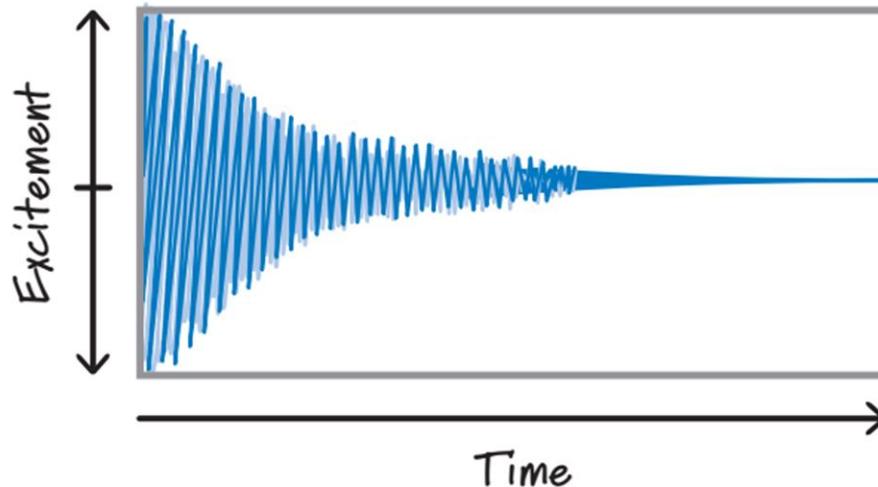
دوباره از سر گرفتن

لذت، شادی

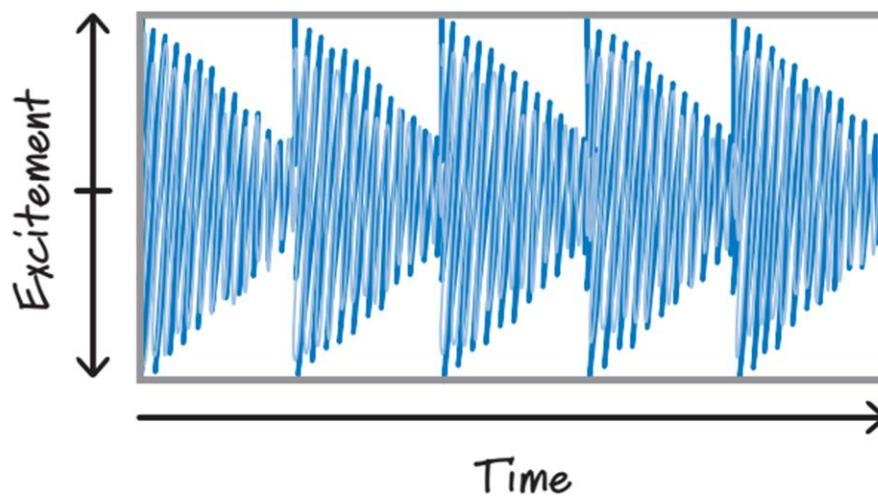
- It is human nature for **interest** and **excitement** to decline the longer we have to wait for **gratification**.
- If we work on a very **long-duration project**, not only are we more **likely to fail**;
- We are also more likely to eventually **lose enthusiasm** for the effort.

# Excitement over time

Boil the ocean



Short-duration  
incremental  
releases



# “boil-the-ocean” projects

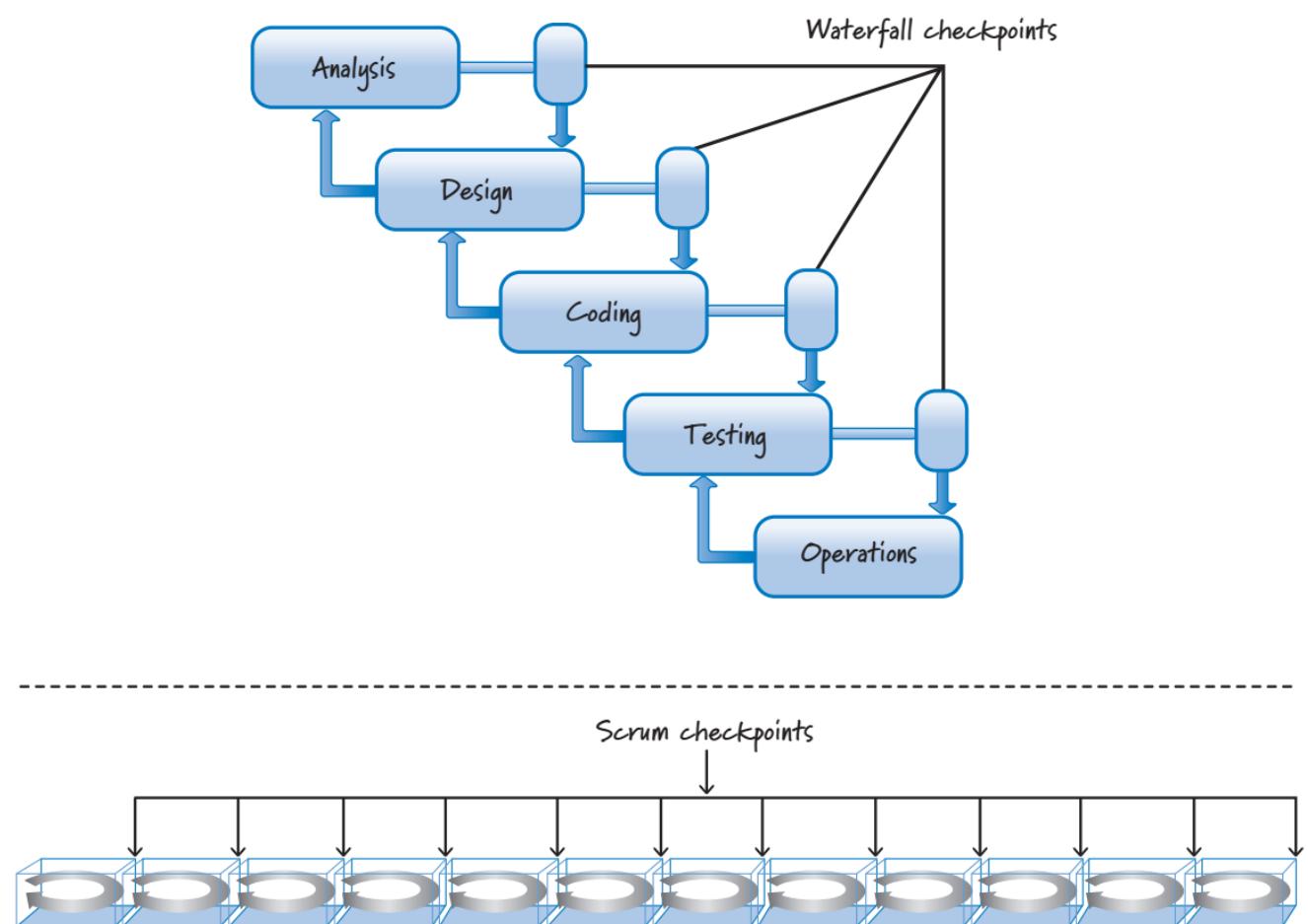
- In “boil-the-ocean” projects, they would take a really long time and a lot of effort to complete, if ever, like trying to boil an ocean.)
- With no visible progress and no end in sight, people begin to grow disinterested.
- Toward the end, they may be willing to pay someone to get moved to a different product!

# Short-duration sprints keep excitement

- Keep participant excitement high by delivering working assets frequently.
- The gratification from early and frequent deliverables rejuvenates our interest and our desire to continue working toward the goal.

# Short Duration sprints have Frequent Checkpoints(I)

- Provide multiple, meaningful checkpoints.



# Short Duration sprints have Frequent Checkpoints(II)

- Scrum provides managers, stakeholders, product owners, and others with many more checkpoints than they would have with sequential projects.
- At the end of each short sprint there is a meaningful checkpoint (the sprint review) that allows everyone to base decisions on demonstrable, working features.
- People are better able to deal with a complex environment when they have more actionable checkpoint opportunities to inspect and adapt.

# Reference

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

# Sprint Rules(II)

Dr. Elham Mahmoudzadeh

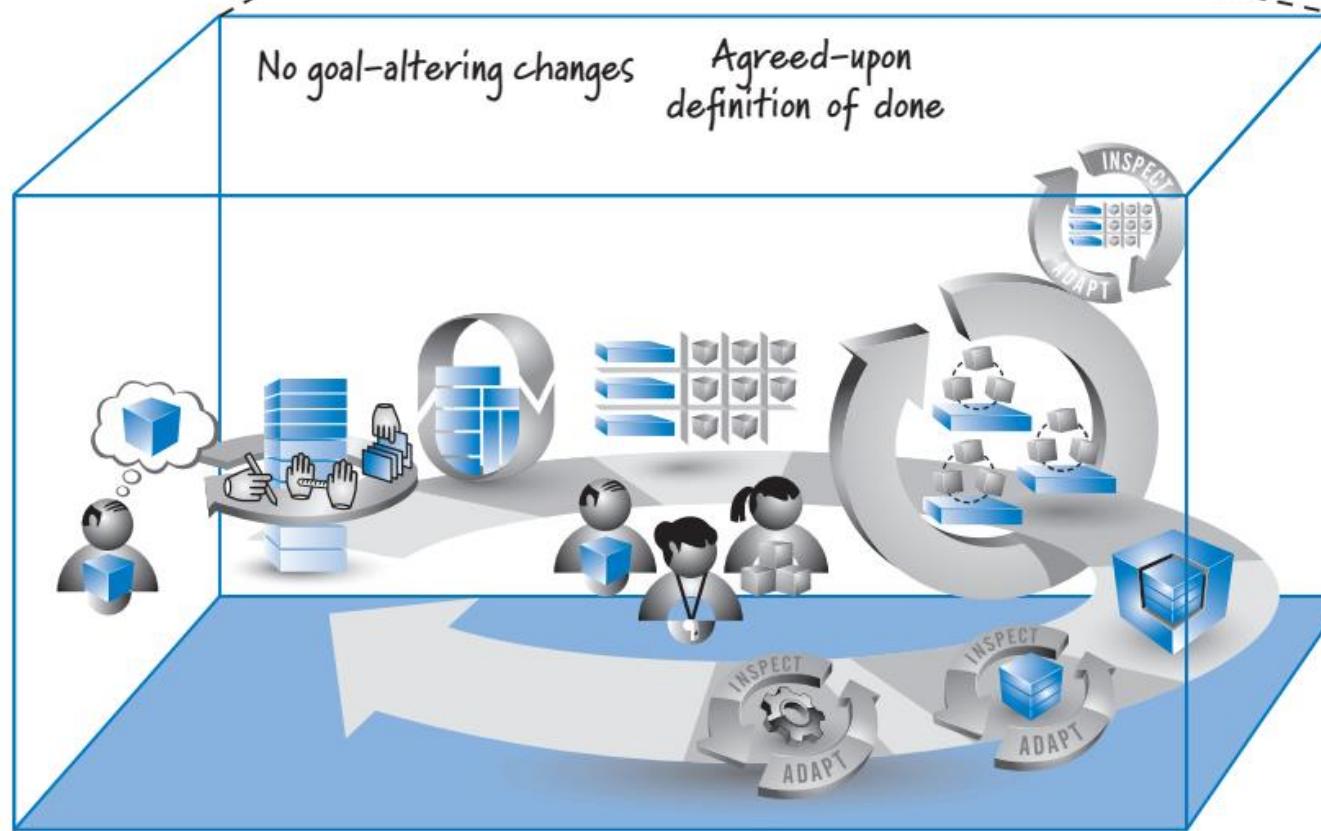
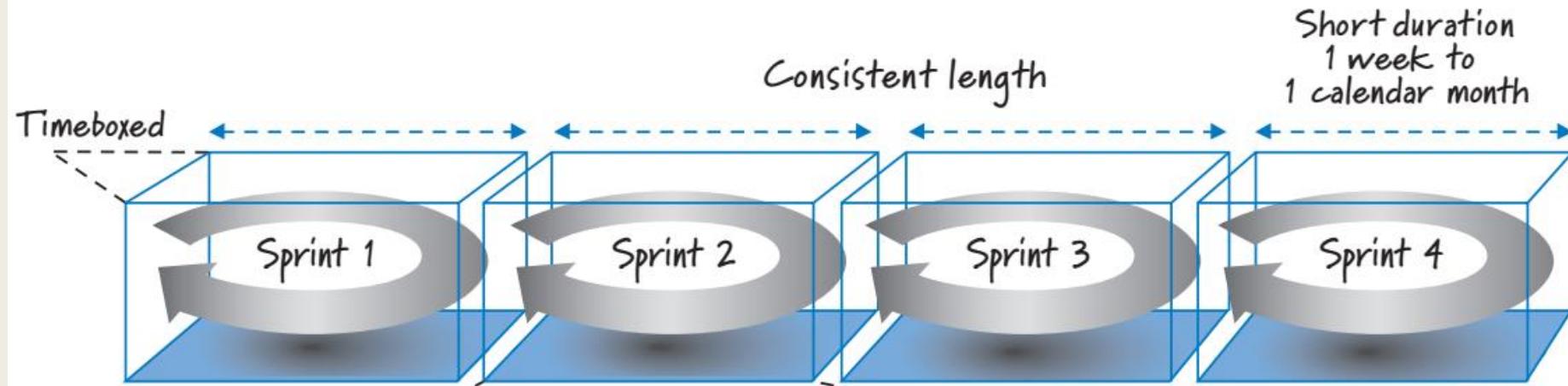
Isfahan University of Technology

[mahmoudzadeh@iut.ac.ir](mailto:mahmoudzadeh@iut.ac.ir)

2023

# Introduction

- Scrum organizes work in iterations or cycles of up to a calendar month called sprints.
- Sprints are the skeleton of the Scrum framework.
- A sprint spans: Sprint Planning, Sprint Execution, Sprint Review, and Sprint Retrospective.



# Sprint Rules

- All sprints are **timeboxed**: They have fixed start and end dates.
- Sprints must also be **short**: Between one week and a calendar month.
- Sprints should be **consistent in length**, though exceptions are permitted under certain circumstances.
- **No goal-altering changes** in scope or personnel are permitted during a sprint.
- During each sprint, a **potentially shippable product increment** is completed in conformance with the Scrum team's agreed-upon “definition of done.”

# Third Rule: Consistent Duration(I)

به عنوان یک قاعده، در یک تلاش توسعه داده شده، یک تیم باید مدت زمان ثابتی را برای اسپرینت های خود انتخاب کند و آن را تغییر ندهد مگر اینکه دلیل قانع کننده ای وجود داشته باشد.

شما در نظر دارید از اسپرینتهاي چهار هفته اي به دو هفته اي حرکت کنيد تا بازخورد بيشتری دریافت کنید، اما میخواهید قبل از تصمیمگیری نهايی، چند دوی سرعت دو هفته اي را امتحان کنيد.

تعطیلات سالانه یا پایان سال مالی، اجرای یک اسپرینت سه هفته ای را نسبت به دو هفته معمولی عملی تر می کند.  
عرضه محصول در یک هفته اتفاق می افتد، بنابراین یک سرعت دو هفته ای بیهوده خواهد بود.

- As a rule, on a given development effort, a team should pick a consistent duration for its sprints and not change it unless there is a compelling reason.
  - You are considering moving from four-week sprints to two-week sprints in order to obtain more frequent feedback but want to try a couple of two-week sprints before making a final decision.
  - The annual holidays or end of the fiscal year make it more practical to run a three-week sprint than the usual two-week sprint.
  - The product release occurs in one week, so a two-week sprint would be wasteful.

# Third Rule: Consistent Duration (II)

- The fact that the team cannot get all the work done within the current sprint length is not a compelling reason to extend the sprint length.
- Neither is it permissible to get to the last day of the sprint, realize you are not going to be done, and lobby for an extra day or week. These are symptoms of dysfunction; they are not good reasons to change the sprint length.

این واقعیت که تیم نمی‌تواند تمام کارها را در طول سرعت فعلی انجام دهد دلیل قانع کننده‌ای برای افزایش طول سرعت نیست.

نمچنین نمی‌توان به آخرین روز دوی سرعت رسید، متوجه شد که کارتان تمام نمی‌شود و یک روز یا هفته اضافی لابی کنید. اینها علائم اختلال عملکرد هستند. آنها دلایل خوبی برای تغییر طول اسپرینت نیستند.

# Third Rule: Consistent Duration(III)

- A week usually means five calendar weekdays.
- If there is a one-day holiday or training event during the sprint, it reduces the team's capacity for that sprint but doesn't necessitate a sprint length change.

# Cadence as a benefit of Consistent Duration(I)

- Sprints of the same duration provide us with cadence: regular, predictable rhythm or heartbeat to a Scrum development effort.
- A steady, healthy heartbeat allows the Scrum team and the organization to acquire an important rhythmic familiarity with when things need to happen to achieve the fast, flexible flow of business value.

اسپرینتهاي با مدت زمان يكسان، آهنگي را برای ما فراهم میکنند: ریتم یا ضربان قلب منظم و قابل پیشبینی در تلاش توسعه اسکرام.  
ضربان قلب ثابت و سالم به تیم اسکرام و سازمان این امکان را میدهد تا آشنایی ریتمیک مهمی با زمانی که برای دستیابی به جریان سریع و انعطاف‌پذیر ارزش تجاری باید اتفاق بیفتد، به دست آورند.

# Cadence as a benefit of Consistent Duration(II)

- Having a regular cadence to sprints enables people to “get into the zone,”
- This happens because regular cadence makes the mundane but necessary activities habitual, thereby freeing up mental capacity to stay focused on the fun, value-added work.
- Generally, It enables people to get comfortable with the project.

داشتن آهنگ منظم در دوی سرعت، افراد را قادر میسازد تا «به منطقه وارد شوند». این به این دلیل انفاق میافتد که آهنگ منظم، فعالیتهای پیش پا افتاده اما ضروری را عادی میکند، در نتیجه ظرفیت ذهنی را آزاد میکند تا روی کار سرگرمکننده و دارای ارزش افزوده متمرکز بماند. به طور کلی، مردم را قادر می سازد تا با پروژه راحت باشند.

# Cadence as a benefit of Consistent Duration(III)

- Having a short sprint cadence also tends to level out the intensity of work.
- Unlike a traditional sequential project where we see a steep increase in intensity in the latter phases, each sprint has an intensity profile that is similar to that of the other sprints.
- Sprint cadence enables teams to work at a sustainable pace.

داشتن یک سرعت دوی سرعت کوتاه نیز باعث کاهش شدت کار می شود.  
ر خلاف پروژهای متوالی سنتی که در آن شاهد افزایش شدید شدت در مراحل آخر هستیم، هر دوی سرعت دارای مشخصات شدتی مشابه با سرعتهای دیگر است.  
سرعت سرعت تیم ها را قادر می سازد تا با سرعتی پایدار کار کند.

# Cadence as Duration(IV)

دویدن با سرعت منظم نیز به طور قابل توجهی سربار هماهنگی را کاهش می دهد.  
ا دوی سرعت ثابت میتوانیم بهطور پیشبینیشده برنامه ریزی سرعت، بررسی سرعت و فعالیتهای گذشته نگر دوی سرعت را برای بسیاری از اسپرینتها بهطور همزمان برنامه ریزی کنیم.  
از آنجایی که همه میدانند این فعالیتها چه زمانی انجام میشوند، سربار مورد نیاز برای برنامه ریزی آنها برای دسته بزرگی از سرعتها به میزان قابل توجهی کاهش میباید.  
اگر اجازه میدهیم مدت زمان اسپرینت از دوی سرعتی به اسپرینت دیگر متفاوت باشد، تلاش بیشتری را برای هماهنگ کردن برنامه های زیاد نیافعان در مورد آنچه ممکن است فقط یک یا دو هفته برای بررسی آتی اسپرینت باشد، تصور کنید!

- Sprinting on a regular cadence also significantly reduces coordination overhead.
- With fixed-length sprints we can predictably schedule the sprint-planning, sprint review, and sprint retrospective activities for many sprints at the same time.
- Because everyone knows when the activities will occur, the overhead required to schedule them for a large batch of sprints is substantially reduced.
- If we allowed sprint durations to vary from sprint to sprint, imagine the extra effort we would need to coordinate the schedules of the stakeholders on what might be just one or two weeks' notice for an upcoming sprint review!

# Cadence as a benefit of Consistent Duration(V)

- If we have multiple teams on the same project, having all teams with a similar sprint cadence allows for synchronization of the work across all of the teams.

اگر چندین تیم در یک پروژه داشته باشیم، داشتن همه تیمها با سرعت اسپرینت مشابه امکان همگامسازی کار را در همه تیمها فراهم میکند.

# Simplified planning as a benefit of Consistent Duration(I)

- When all sprints are the same length (even when they might have a day or less capacity per sprint because of a holiday), the team gets comfortable with the amount of work that it can accomplish in a typical sprint (referred to as its velocity).
- Velocity is typically normalized to a sprint. If the length of the sprint can vary, we really don't have a normalized sprint unit.
- While it is certainly possible to compute a team's velocity even if it uses variable length sprints, it is more complicated.
- Sticking with a consistent sprint duration simplifies the computations we perform on a team's historical velocity data.

# Simplified planning as a benefit of Consistent Duration(II)

- Consistent sprint durations also simplify the rest of the planning.
- If the sprint durations were allowed to vary, calculating the number of sprints in the release could be significantly more challenging (because we would have to do extensive early planning), involve unnecessary overhead, and likely be far less reliable than with consistent sprint durations.

# Forth Rule: No Goal-Altering Changes(I)

- Once the **sprint goal has been established** and **sprint execution** has begun, **no change is permitted** that can materially affect the sprint goal.

# What Is a Sprint Goal?

- Each sprint can be summarized by a sprint goal that describes the business purpose and value of the sprint. Typically the sprint goal has a clear, single focus.
- Examples
  - Support initial report generation.
  - Load North America map data.
- During sprint planning, the development team should help refine and agree to the sprint goal and use it to determine the product backlog items that it can complete by the end of the sprint.

# No Goal-Altering Changes: Mutual Commitment

- The sprint goal is the foundation of a mutual commitment made by the team and the product owner.
- The team commits to meet the goal by the end of the sprint, and the product owner commits to not altering the goal during the sprint.
- This mutual commitment demonstrates the importance of sprints in balancing the needs of the business to be adaptive to change, while allowing the team to concentrate and efficiently apply its talent to create value during a short, fixed duration.
- By defining and adhering to a sprint goal, the Scrum team is able to stay focused (in the zone) on a well-defined, valuable target.

# No Goal-Altering Changes: Change versus Clarification(I)

- Although the sprint goal should not be materially changed, it is permissible to clarify the goal.
- A change is any alteration in work or resources that has the potential to generate economically meaningful waste, harmfully disrupt the flow of work, or substantially increase the scope of work within a sprint. Adding or removing a product backlog item from a sprint or significantly altering the scope of a product backlog item that is already in the sprint typically constitutes change.
- Example of goal changes: Adding the ability to search based on a picture likely represents substantially more effort and almost certainly would affect the team's ability to meet a commitment to deliver search based on last name and first name.

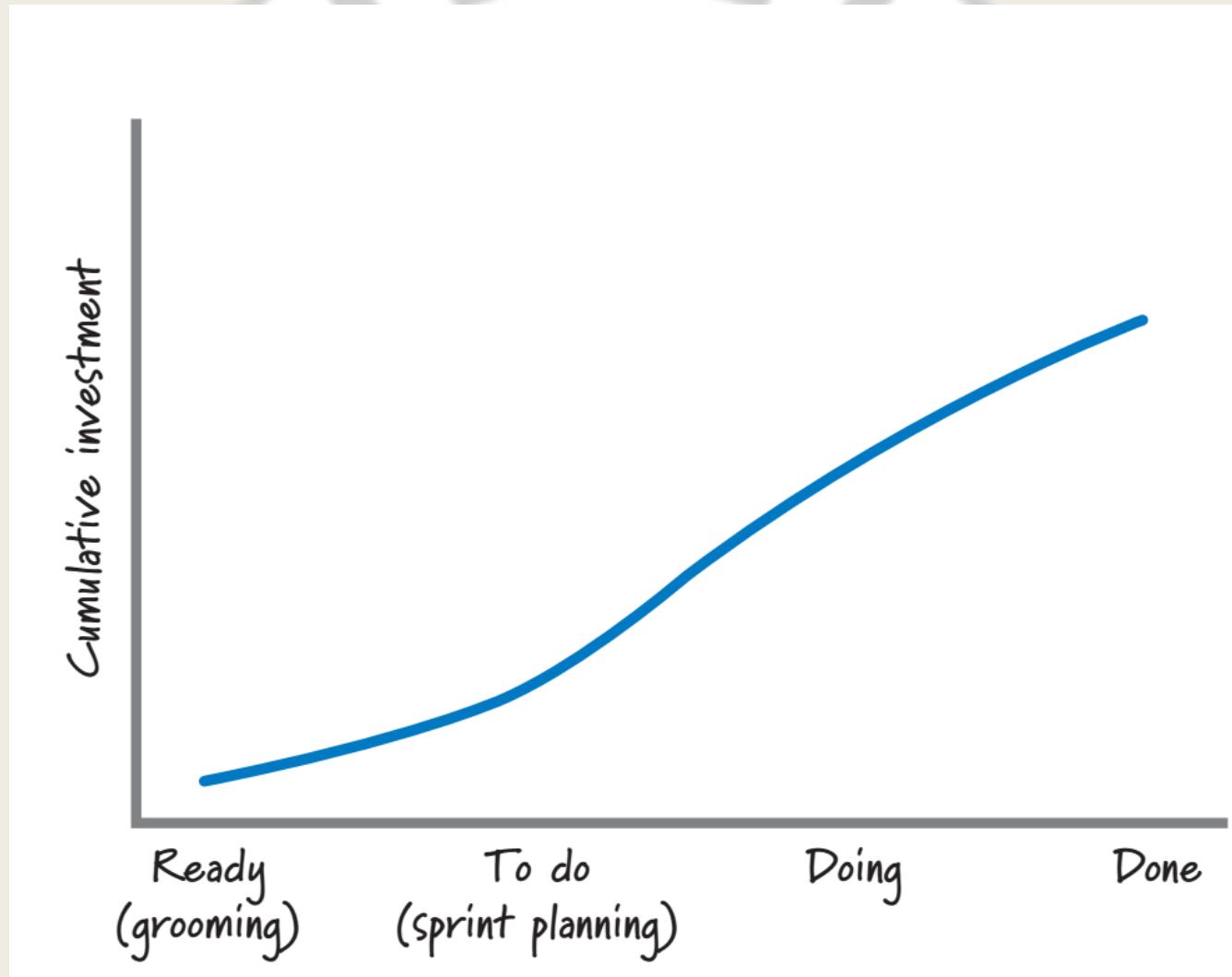
# No Goal-Altering Changes: Change versus Clarification (II)

- Clarifications are additional details provided during the sprint that assist the team in achieving its sprint goal.
- All of the details associated with product backlog items might not be fully known or specified at the start of the sprint. Therefore, it is completely reasonable for the team to ask clarifying questions during a sprint and for the product owner to answer those questions.
- Example of clarification: did you have a preference for how that list is to be ordered? Yes, order alphabetically.

# No Goal-Altering Changes: Consequences of Change(I)

- It may appear that the no-goal-altering-change rule is in direct conflict with the core Scrum principle that we should embrace change. We do embrace change, but we want to embrace it in a balanced, economically sensible way.
- The economic consequences of a change increase as our level of investment in the changed work increases.

# Cumulative investment at different states



# No Goal-Altering Changes: Consequences of Change(III)

- We invest in product backlog items to get them ready to be worked on in a sprint.
- However, once a sprint starts, our investment in those product backlog items has increased (because we spent time during sprint planning to discuss and plan them at a task level).
- If we want to make a change after sprint planning has occurred, we not only jeopardize the planning investment, but we also incur additional costs for having to replan any changes during the sprint.

# No Goal-Altering Changes: Consequences of Change(IV)

- In addition, once we begin sprint execution, our investment in work increases even more as product backlog items transition through the states of to do (work not yet started), doing (work in process), and done (work completed).

# No Goal-Altering Changes: Consequences of Change(V)

- Let's say we want to swap out feature X, currently part of the sprint commitment, and substitute feature Y, which isn't part of the existing commitment.
- Even if we haven't started working on feature X, we still incur planning waste. In addition, feature X might also have dependencies with other features in the sprint, so a change that affects feature X could affect one or more other features, thus amplifying the effect on the sprint goal.

# No Goal-Altering Changes: Consequences of Change(VI)

- If work on feature X has already begun, in addition to the already-mentioned waste, we could have other potential wastes.
- For example, all of the work already performed on feature X might have to be thrown away. And we might have the additional waste of removing the partially completed work on feature X, which we may never use in the future.
- And, of course, if feature X is already completed, we might have wasted the full investment we made in feature X. All of this waste adds up!

# No Goal-Altering Changes: Consequences of Change(VII)

- In addition to the **direct economic consequences** of waste, the economics can be **indirectly affected** by the **potential deterioration** of **team motivation** and **trust** that can accompany a change.
- When the **product owner** makes a commitment **to not alter the goal** and then **violates the commitment**, the team naturally will be **demotivated**, which will almost certainly affect **its desire to work** diligently to complete other product backlog items.
- In addition, **violating the commitment** can **harm the trust** within the Scrum team, because the development team will not trust that the product owner is willing to stick to his commitments.

# Being Pragmatic(I)

- The no-goal-altering-change rule is just that—a rule, not a law. The Scrum team has to be pragmatic.
- What if business conditions change in such a way that making a change to the sprint goal seems warranted? Say a competitor launches its new product during our sprint. After reviewing the new product, we conclude that we need to alter the goal we established for our current sprint because what we are doing is now economically far less valuable given what our competitor has done. Should we blindly follow the rule of no goal-altering changes and not alter our sprint? Probably not.
- What if a critical production system has failed miserably and some or all of the people on our team are the only ones who can fix it? Should we not interrupt the current sprint to fix it? Do we tell the business that we will fix the production failure first thing next sprint? Probably not.

# Being Pragmatic(II)

- In the end, being pragmatic **trumps** the no-goal-altering-change **rule**. We must act in an **economically sensible way**.
- If the **economic consequences of the change** are **far less** than the **economic consequences of deferring the change**, making the change is the smart business decision.
- If the economics of changing versus not changing are **immaterial**, no change to the sprint goal should be made.
- As for **team motivation** and **trust**, when a product owner has a **frank**, **economically focused discussion** with the team about the necessity of the change, most teams **understand** and **appreciate** the need, so the **integrity of motivation** and trust is upheld.

# Abnormal Termination(I)

- When the **sprint goal** become **completely invalid**, the Scrum team may decide that **continuing** with the current sprint **makes no sense** and advise the product owner to abnormally **terminate the sprint**.
- When a sprint is abnormally **terminated**, the current sprint comes to an **abrupt end** and the **Scrum team** gathers to perform a **sprint retrospective**. The team then meets with the **product owner** to plan the next sprint, with **a different goal** and a different set of product backlog items.
- **Sprint termination** is used when an **economically significant event** has occurred, such as a **competitor's actions** that completely invalidate the sprint or product funding being materially changed.

# Abnormal Termination(II)

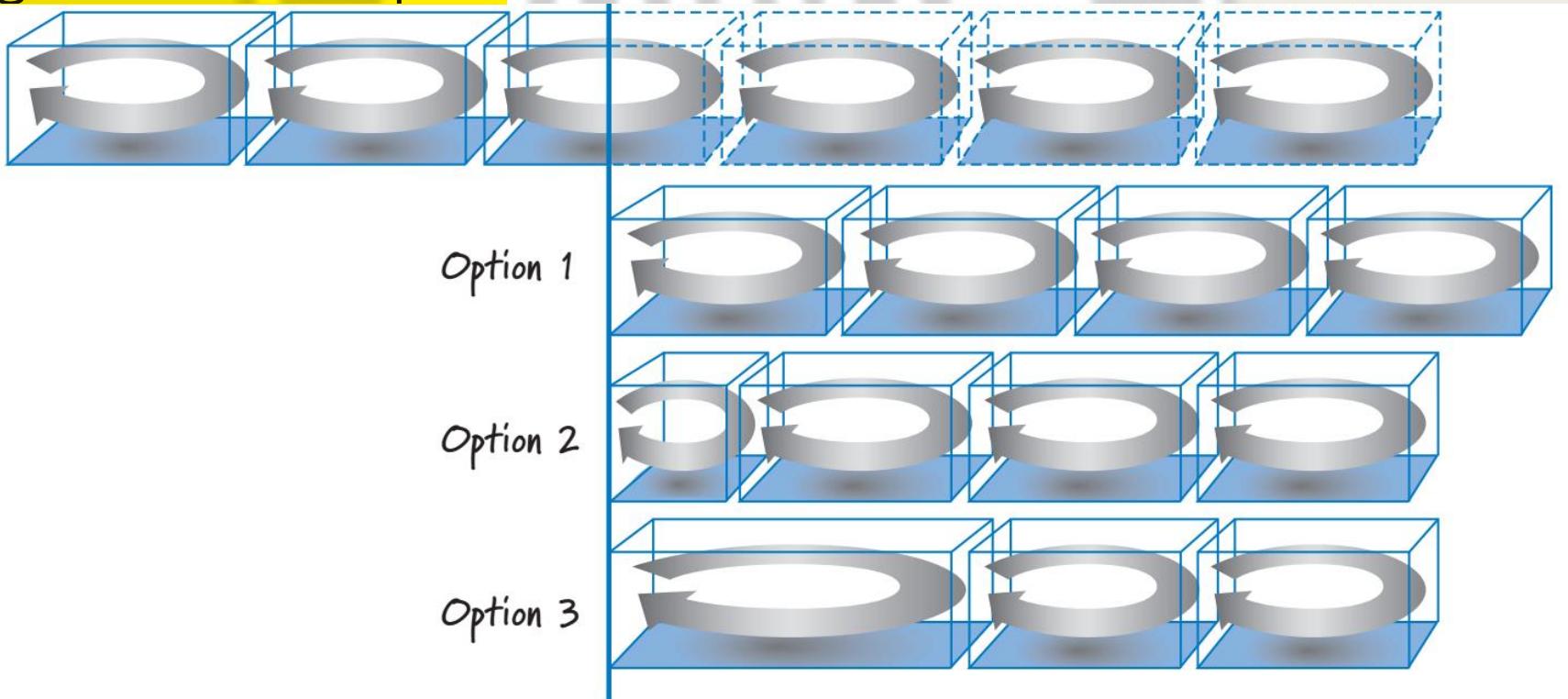
- It is important to **realize** that terminating the sprint **early**, in addition to having a negative effect on morale, is **a serious disruption of the fast, flexible flow of features** and **negates** many of the benefits of consistent-duration sprints.
- Terminating a sprint should **be the last resort.**

درک این نکته مهم است که پایان زودهنگام اسپرینت، علاوه بر تأثیر منفی بر روحیه، یک اختلال جدی در جریان سریع و انعطاف‌پذیر ویژگیها است و بسیاری از مزایای اسپرینت ثابت را نفی می‌کند.

■ پایان دادن به اسپرینت باید آخرین راه حل باشد

# Options in the case of termination(I)

- If a sprint is terminated, the Scrum team will have to determine the length of the next sprint.



# Fifth Rule: Conformance to the Definition of Done

- Definition of done is a checklist of the types of work that the team is expected to successfully complete before it can declare its work to be potentially shippable.
- Items on the checklist will depend on
  - *The nature of the product being built.*
  - *The technologies being used to build it.*
  - *The organization that is building it.*
  - *The current impediments that affect what is possible.*



# Example of definition of done

Definition of Done	
<input type="checkbox"/>	Design reviewed
<input type="checkbox"/>	Code completed
<input type="checkbox"/>	Code refactored
<input type="checkbox"/>	Code in standard format
<input type="checkbox"/>	Code is commented
<input type="checkbox"/>	Code checked in
<input type="checkbox"/>	Code inspected
<input type="checkbox"/>	End-user documentation updated
<input type="checkbox"/>	Tested
<input type="checkbox"/>	Unit tested
<input type="checkbox"/>	Integration tested
<input type="checkbox"/>	Regression tested
<input type="checkbox"/>	Platform tested
<input type="checkbox"/>	Language tested
<input type="checkbox"/>	Zero known defects
<input type="checkbox"/>	Acceptance tested
<input type="checkbox"/>	Live on production servers

# Definition of Done

- What if there is a significant defect that remains on the last day of the sprint; is the product backlog item done? No, it's not done!
- And because, as a rule, we don't extend sprints beyond the end of the planned timebox, we wouldn't extend the sprint by a day or two to fix the defect in the current sprint.
- Instead, at the planned end of the sprint, the incomplete product backlog item is taken from the current sprint and reinserted into the product backlog in the proper order based on the other items that are currently in the product backlog. The incomplete item might then be finished in some future sprint.

# Definition of Done

- Scrum teams need to have a robust definition of done, one that provides a high level of confidence that what they build is of high quality and can be shipped.
- “potentially shippable” doesn’t mean that what was built must actually be shipped. Shipping is a business decision that often occurs at a different cadence; in some organizations it may not make sense to ship at the end of every sprint.

# Definition of Done Can Evolve Over Time

- For some, real impediments might prevent team from reaching defined state of the work at the end of the sprint, at the start of development, even though it is the ultimate goal.
- As a result, they might (necessarily) start with a lesser end state and let their definition of done evolve over time as organizational impediments are removed.

# Example

- It is a product that includes hardware and software, where the hardware is late.
- If the team is building software and it doesn't have the actual hardware on which to test the software, it can't really claim that the results produced at the end of the sprint are potentially shippable.
- At best it might claim “emulator done,” because testing during the early sprints is typically performed against a software emulator of the actual hardware. Later, when the actual hardware is available, the definition of done will evolve to mean potentially shippable or at least something closer to it.

# Definition of Done versus Acceptance Criteria(I)

- The **definition of done** applies to the **product increment** being developed during the sprint. The product increment is **composed** of a **set of product backlog items**, so each backlog item must be completed in conformance with the work specified by the definition-of-done checklist.
- Each product backlog item that is **brought into the sprint** should have **a set of conditions of satisfaction** (item-specific acceptance criteria), specified by the **product owner**.

# Definition of Done versus Acceptance Criteria(II)

- Acceptance criteria eventually will be verified in acceptance tests that the product owner will conform to determine if the backlog item functions as desired.
- For example, if the product backlog item is “Allow a customer to purchase with a credit card,” the conditions of satisfaction might be “Works with AmEx, Visa, and MasterCard.”
- So each product backlog item will have its own appropriate set of acceptance criteria. These item-specific criteria are the done criteria specified by the definition-of-done checklist, which apply to all product backlog items.

# Done versus Done-Done

- **Done:** doing as much work as they are prepared to do.
- **Done-Done:** doing the work required for customers to believe it is done.

# Reference

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

# Product Backlog(I)

Dr. Elham Mahmoudzadeh

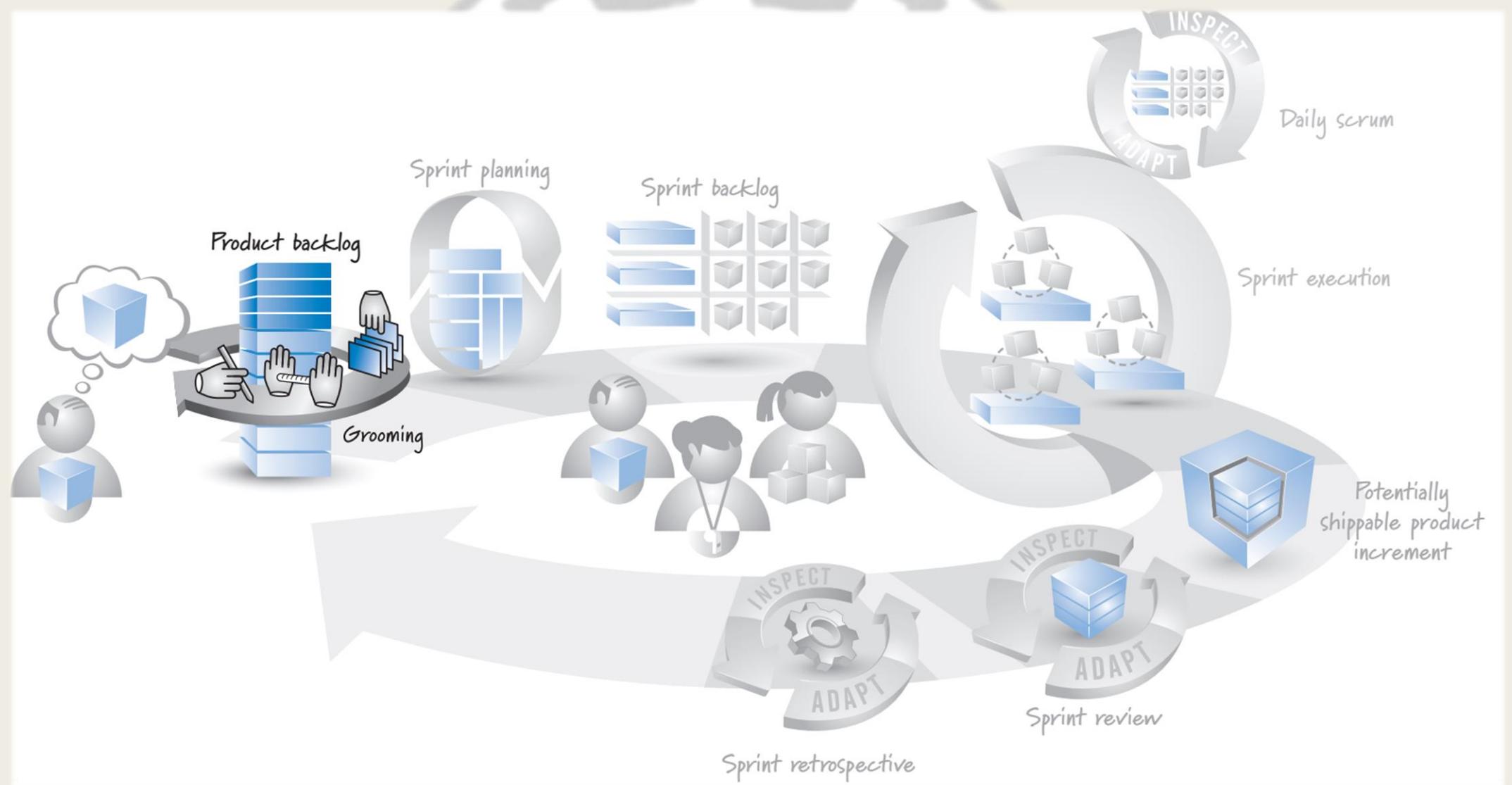
Isfahan University of Technology

[mahmoudzadeh@iut.ac.ir](mailto:mahmoudzadeh@iut.ac.ir)

2022

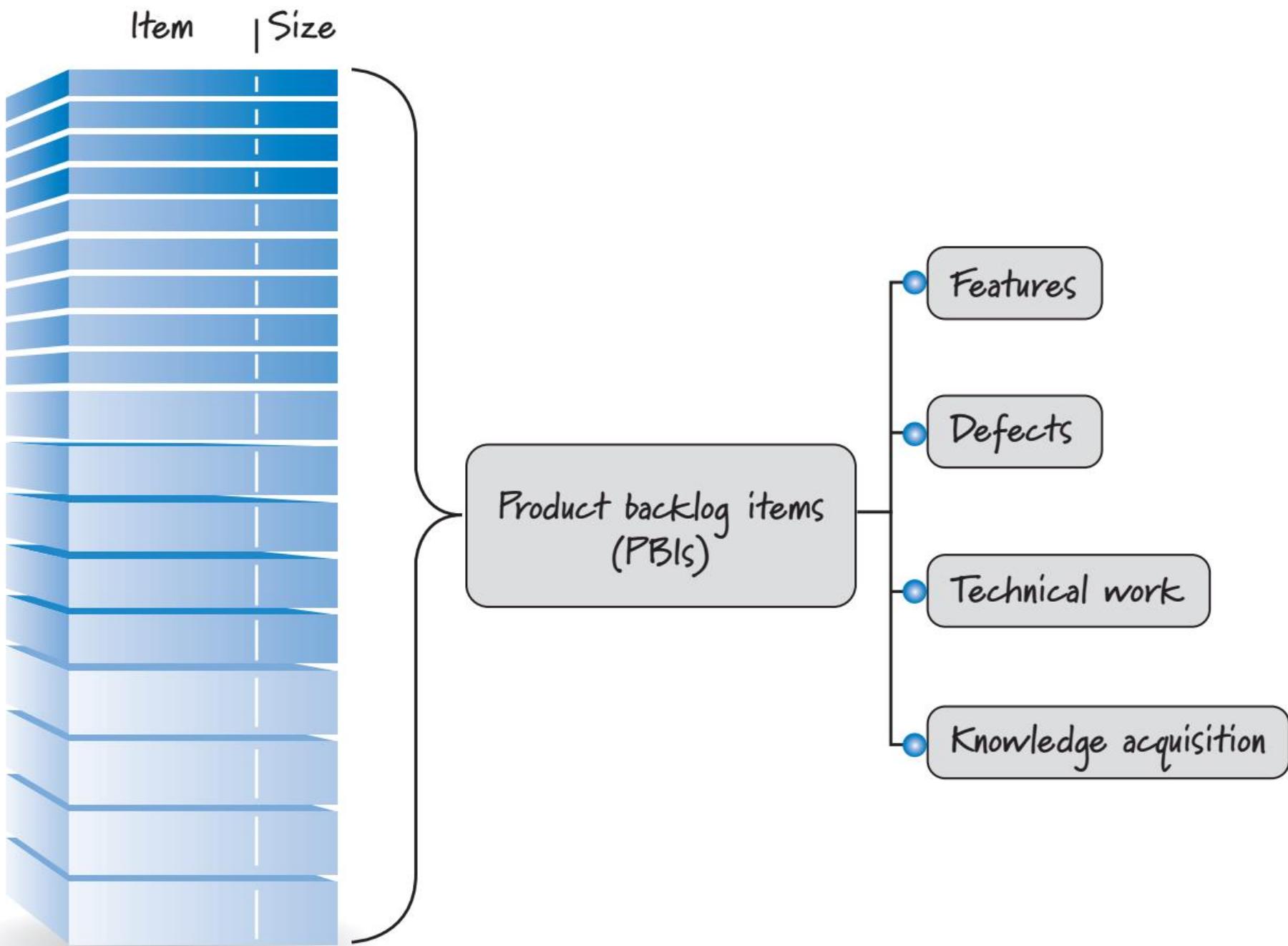
# Product backlog

- Is a prioritized list of desired product functionality.
- It provides a centralized and shared understanding of what to build and the order in which to build it.
- It is a highly visible artifact at the heart of the Scrum framework that is accessible to all project participants.
- As long as there is a product or system being built, enhanced, or supported, there is a product backlog.



# Product Backlog Items

- The product backlog is composed of backlog items, PBIs.
- Most PBIs are features, items of functionality that will have tangible value to the user or customer. These are often written as user stories (although Scrum does not specify the format of PBIs).
- Examples of features include something brand-new (a login screen for a new website), or a change to an existing feature (a more user-friendly login screen for an existing website). Other PBIs include defects needing repair, technical improvements, knowledge-acquisition work, and any other work the product owner deems valuable.



# Example Product Backlog Items

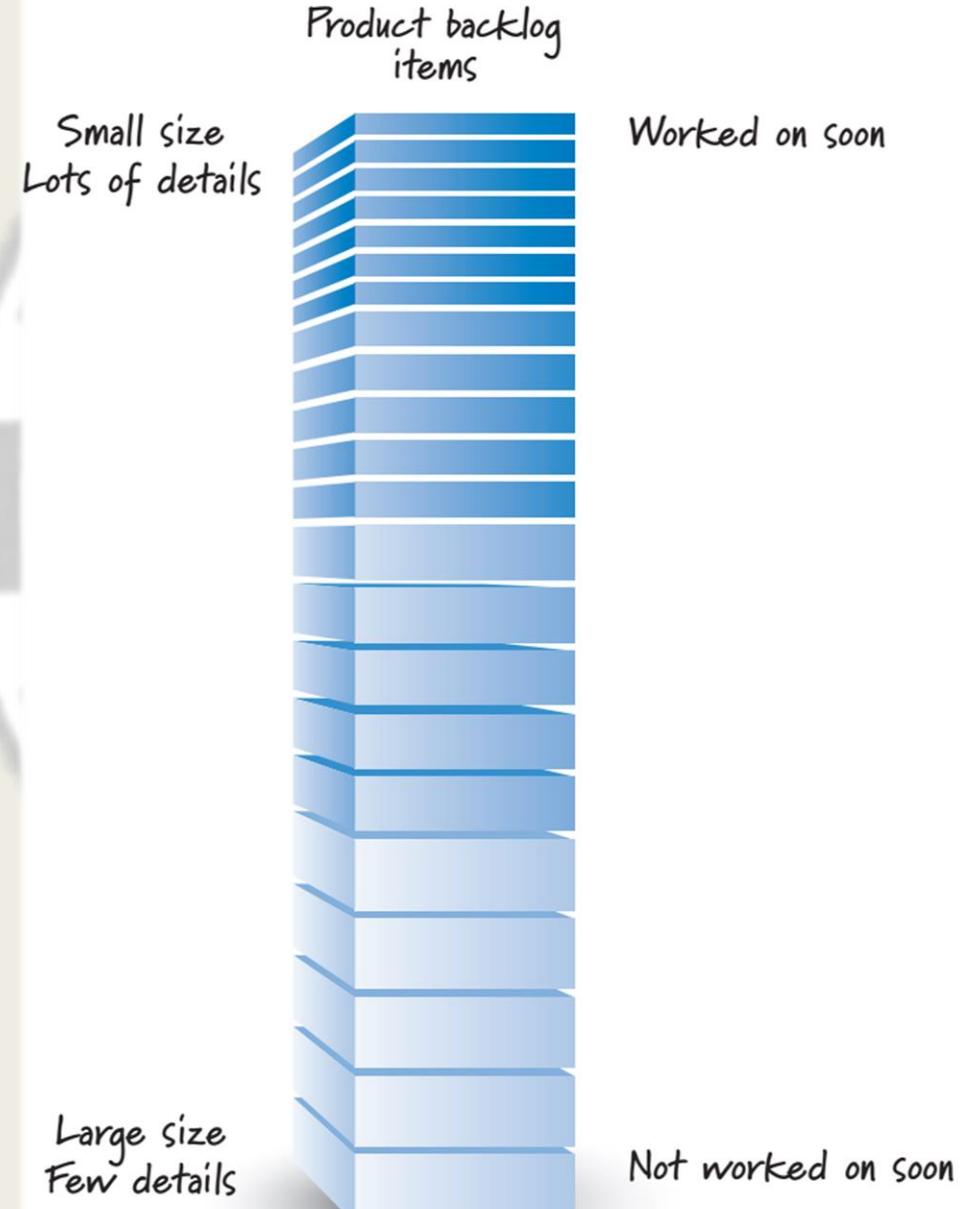
PBI Type	Example
Feature	As a customer service representative I want to create a ticket for a customer support issue so that I can record and manage a customer's request for support.
Change	As a customer service representative I want the default ordering of search results to be by last name instead of ticket number so that it's easier to find a support ticket.
Defect	Fix defect #256 in the defect-tracking system so that special characters in search terms won't make customer searches crash.
Technical improvement	Move to the latest version of the Oracle DBMS.
Knowledge acquisition	Create a prototype or proof of concept of two architectures and run three tests to determine which would be a better approach for our product.

# Good Product Backlog Characteristics

- **DEEP criteria** are useful for determining if **a product backlog** has been structured in a good way.
- *Detailed appropriately,*
- *Emergent,*
- *Estimated,*
- *Prioritized.*

# Detailed Appropriately

- Not all items in a product backlog will be at the same level of detail at the same time.
- PBIs that we plan to work on soon should be near the top of the backlog, small in size, and very detailed so that they can be worked on in a near-term sprint.
- PBIs that we won't work on for some time should be toward the bottom of the backlog, larger in size, and less detailed. We don't plan to work on those PBIs anytime soon.



# Detailed Appropriately (Cnt'd)

- As we get closer to working on a larger PBI, such as an epic, we will break that story down into a collection of smaller, sprint-ready stories. This should happen in a just-in-time fashion.
- If we refine too early, we might spend a good deal of time figuring out the details, only to end up never implementing the story.
- If we wait too long, we will impede the flow of PBIs into the sprint and slow the team down.  

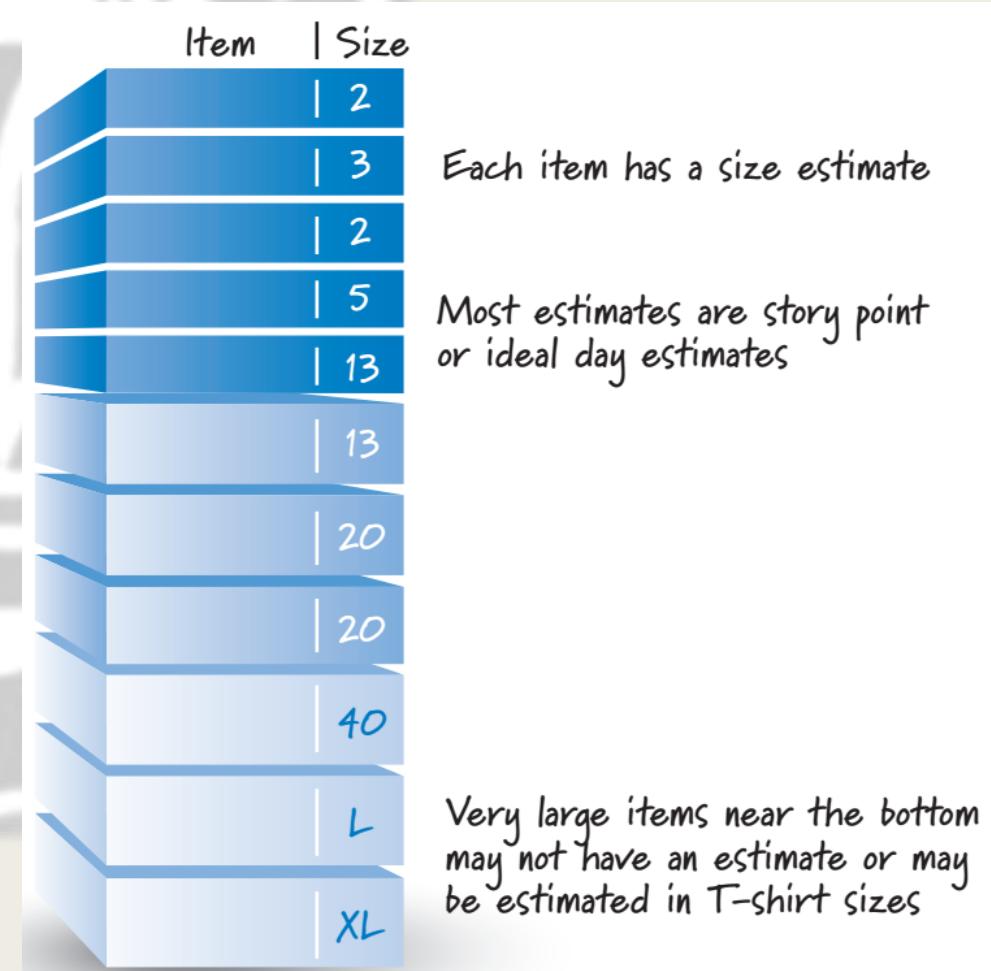

مانع شدن
- We need to find the proper balance of just enough and just in time.

# Emergent

- As long as there is a product being developed or maintained, the product backlog is never complete or frozen.
- It is continuously updated based on a stream of economically valuable information that is constantly arriving.
- The product backlog is designed to adapt to the occurrences.
- The structure of the product backlog is therefore constantly emerging over time.
- As new items are added or existing items are refined, the product owner must rebalance and reprioritize the product backlog, taking the new information into account.

# Estimated

- Each product backlog item has a **size** estimate corresponding to the **effort** required to develop the item.



# Estimated (Cnt'd)

- The product owner uses these estimates as one of several inputs to help determine a PBI's priority (and therefore position) in the product backlog.
- Also, a high-priority, large PBI (near the top of the backlog) signals to the product owner that additional refinement of that item is necessary before it can be moved into a near-term sprint.
- most PBIs are estimated in either story points or ideal days.
- These size estimates need to be reasonably accurate without being overly precise.
- Because items near the top of the backlog are smaller and more detailed, they will have smaller, more accurate size estimates.

# Estimated (Cnt'd)

- It may not be possible to provide numerically accurate estimates for larger items (like epics) located near the bottom of the backlog, so some teams might choose to not estimate them at all, or to use T-shirt-size estimates (L, XL, XXL, etc.).
- As these larger items are refined into a set of smaller items, each of the smaller items would then be estimated with numbers.

# Prioritized

- Although the product backlog is a prioritized list of PBIs, it is unlikely that all of the items in the backlog will be prioritized.
- It is useful to prioritize the near-term items that are destined for the next few sprints.
- Perhaps it is valuable to prioritize as far down in the backlog as we think we can get in Release 1.
- Going beyond that point at anything other than a gross level of prioritization is likely not worth our time.

# Prioritized (Cnt'd)

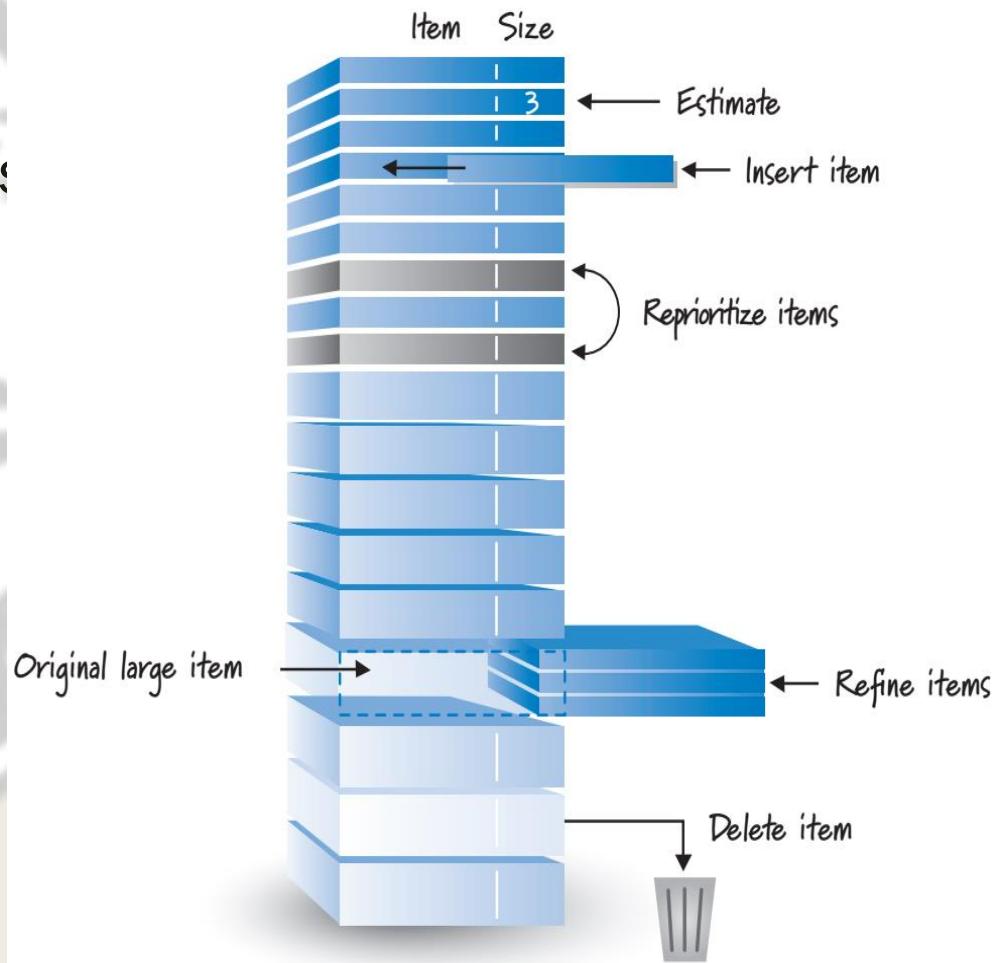
- For example, we might declare that an item is destined for Release 2 or Release 3 according to our product roadmap.
- However, if we are early in the development of Release 1 features, spending any of our valuable time worrying about how to prioritize features that we might work on someday in Release 2 or Release 3 is likely not a good investment.
- We might never end up actually doing a Release 2 or Release 3, or our ideas surrounding those releases might change significantly during the development of Release 1.
- So time spent prioritizing that far out has a high probability of being wasted.

# Product Backlog management

- To get a good, **DEEP** product backlog, we must **proactively manage**, **organize**, **administer**, or, as it has commonly come to be referred to, **groom** the product backlog.

# What Is Grooming?

- Creating and refining (adding details to) PBIs
- Deleting PBIs,
- Estimating PBIs,
- Prioritizing PBIs.

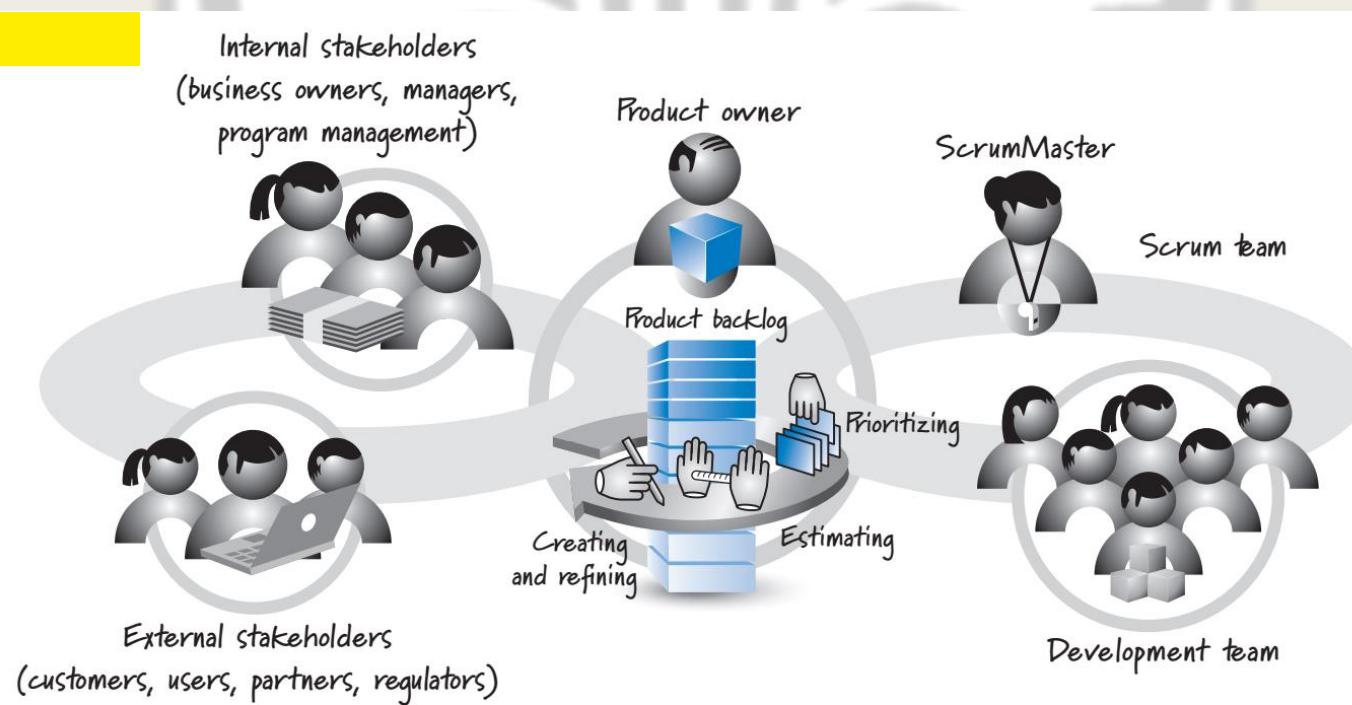


# What Is Grooming? (Cnt'd)

- At the appropriate time, all PBIs need to be estimated to help determine their order in the backlog and to help decide whether additional refinement work is warranted.
- Also, as important information becomes available, new items are created and inserted into the backlog in the correct order. Of course, if priorities shift, we'll want to reorder items in the backlog.
- And as we get closer to working on a larger item, we'll want to refine it into a collection of smaller items. We also might decide that a particular backlog item is just not needed, in which case we'll delete it.

# Who Does the Grooming?

- Grooming the product backlog is an **ongoing collaborative effort** **led by the product owner** and including significant participation from **internal** and **external stakeholders** as well as the **ScrumMaster** and **developers**.



# Who Does the Grooming? (Cnt'd)

- Ultimately there is one grooming decision maker: the product owner.
- However, good product owners understand that collaborative grooming fosters an important dialogue among all participants and leverages the collective intelligence and perspectives of a diverse group of individuals, thereby revealing important information that might otherwise be missed.

# Who Does the Grooming? (Cnt'd)

- Good product owners also know that by involving the diverse team members in the grooming, they ensure that everyone will have a clearer, shared understanding of the product backlog, so less time will be wasted in miscommunications and handoffs.
- Such collaborative efforts also go a long way toward bridging the historical gap between the business people and the technical people.

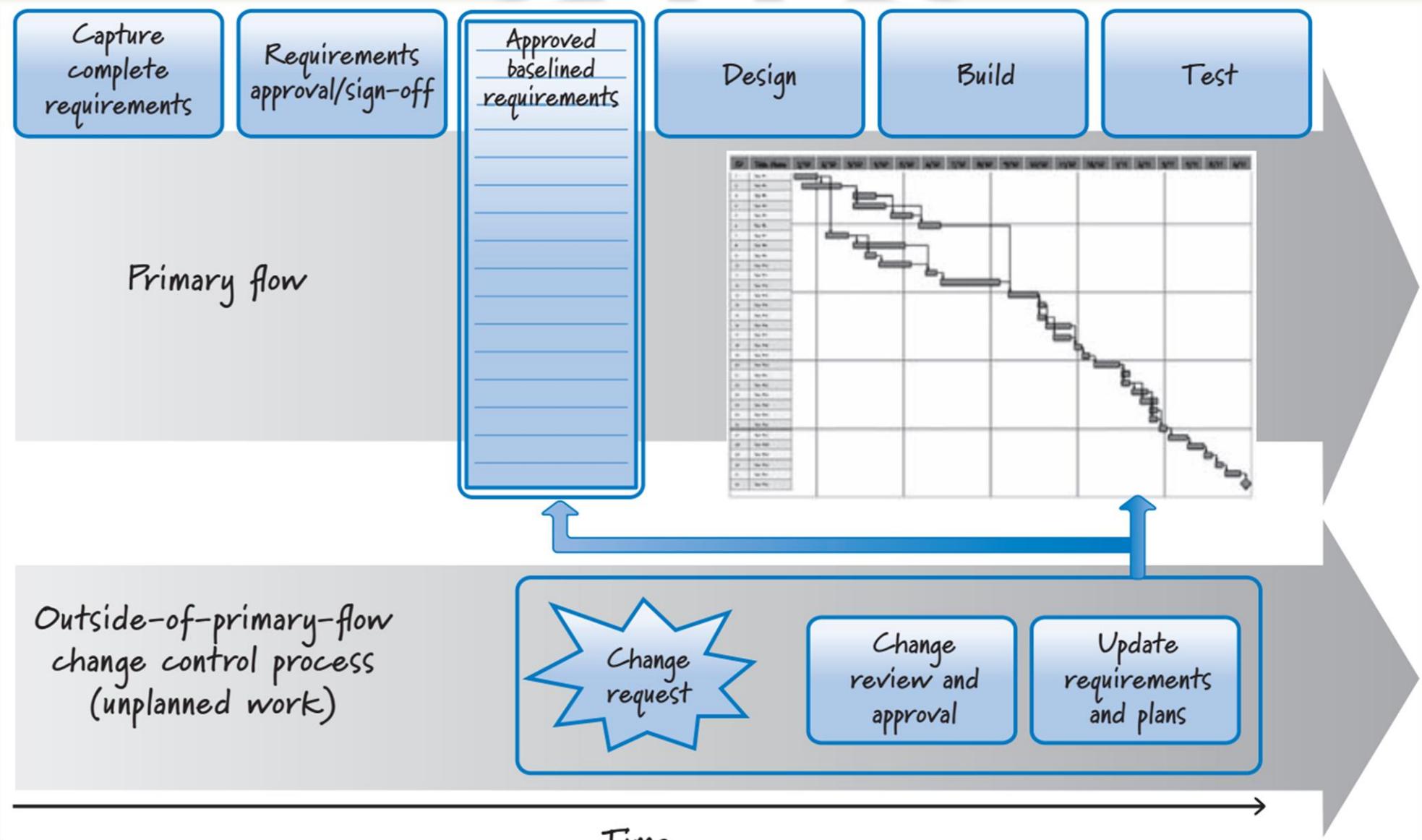
# Who Does the Grooming? (Cnt'd)

- Stakeholders should allocate a sufficient amount of time to grooming based on the nature of the organization and the type of project.
- As a general rule, the development team should allocate up to 10% of its time each sprint to assisting the product owner with grooming activities.
- The team will use this time to help create or review emergent product backlog items as well as progressively refine larger items into smaller items.
- The team will also estimate the size of product backlog items and help the product owner prioritize them based on technical dependencies and resource constraints.

# When Does Grooming Take Place?

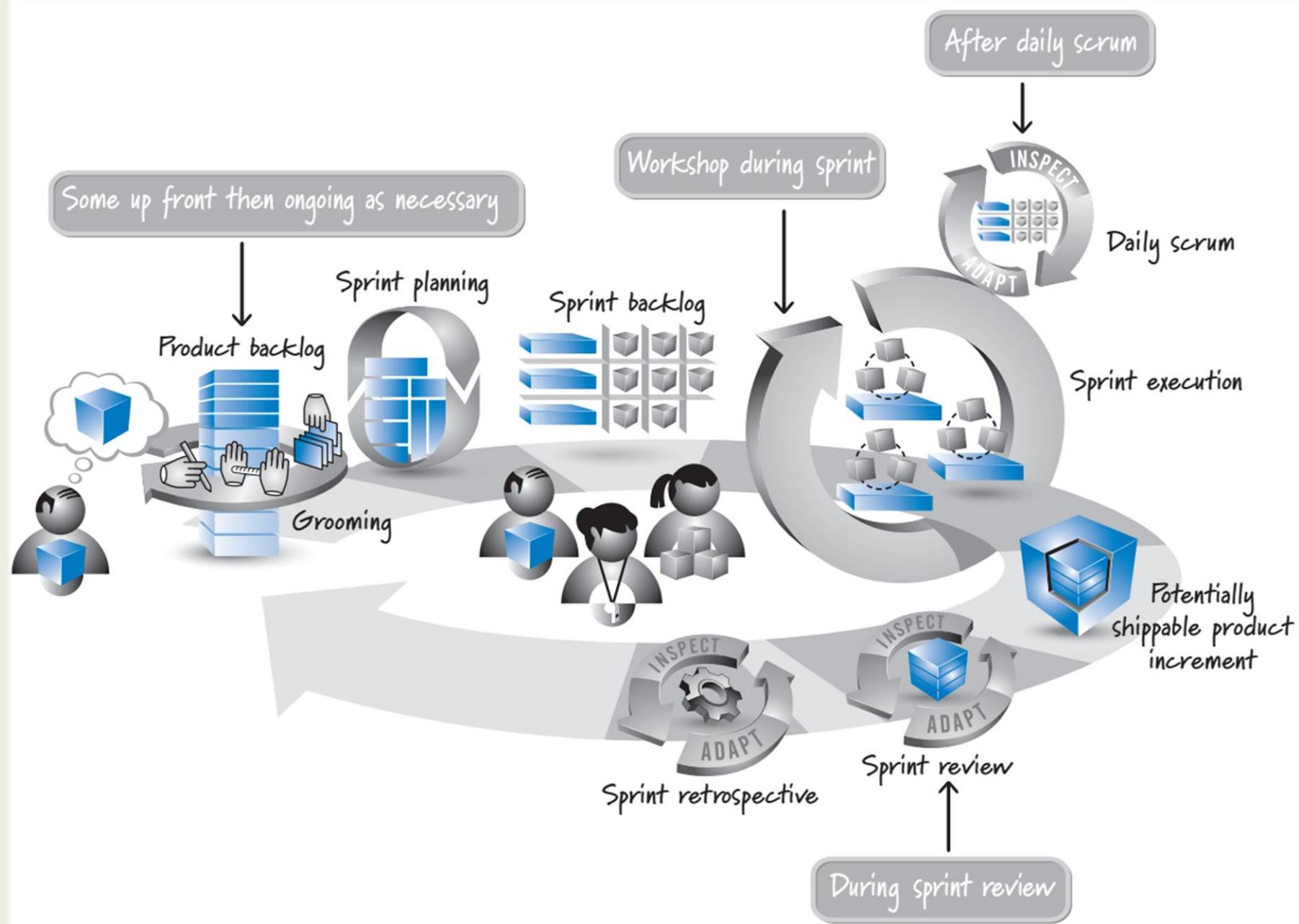
- The Scrum framework only indicates that grooming needs to happen; it doesn't specify when it should happen. So when does grooming actually take place?
- Using sequential development, we try to capture a complete and detailed description of the requirements up front, so little or no requirements grooming is scheduled after the requirements have been approved.
- In many organizations these baselined requirements may be changed only via a separate change control process, which is discontinuous to the primary development.

# Outside-of-primary-flow grooming with sequential projects



# When Does Grooming Take Place? (Cnt'd)

- Grooming during sequential development is an exceptional, unplanned, outside-of-primary-flow activity that we invoke only if we need to, making it disruptive to the fast flow of delivered business value.
- Using Scrum, we assume an uncertain environment and therefore must be prepared to constantly inspect and adapt.
- We expect the product backlog to evolve constantly rather than being locked down early and changed only through a secondary process for handling exceptional, undesirable occurrences.
- As a result, we must ensure that our grooming activities are an essential, intrinsic part of how we manage our work.



# When Does Grooming Take Place? (Cnt'd)

- Initial grooming occurs as part of the release-planning activity.
- During product development, the product owner meets with the stakeholders at whatever frequency makes sense to perform ongoing grooming.
- When working with the development team, the product owner might schedule either a weekly or a once-a-sprint grooming workshop during sprint execution.
- Doing so ensures that grooming occurs on a regular schedule and enables the team to account for that time during sprint planning. It also reduces the waste of trying to schedule ad hoc meetings (for example, determining when people are available, finding available space, and so on).

# When Does Grooming Take Place? (Cnt'd)

- Sometimes teams prefer to spread out the grooming across the sprint, rather than block out a predetermined period of time.
- They take a bit of time after their daily scrums to do some incremental grooming.
- This grooming doesn't have to include all of the team members. For example, after a daily scrum the product owner might ask for help refining a large story. Team members who are knowledgeable and interested stick around and assist the product owner. The next time, different team members might assist.

# When Does Grooming Take Place? (Cnt'd)

- Even if teams have regularly scheduled workshops or take some time each day to look at the backlog, most teams find that they naturally do some grooming as part of the sprint review.
- As everyone involved gains a better understanding of where the product is and where it is going, new PBIs are often created or existing PBIs are reprioritized, or deleted if they are no longer needed.
- When the grooming happens is less important than making sure it is well integrated into the Scrum development flow, to ensure flexible and fast delivery of business value.

# Reference

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

# Product Backlog(II)

Dr. Elham Mahmoudzadeh

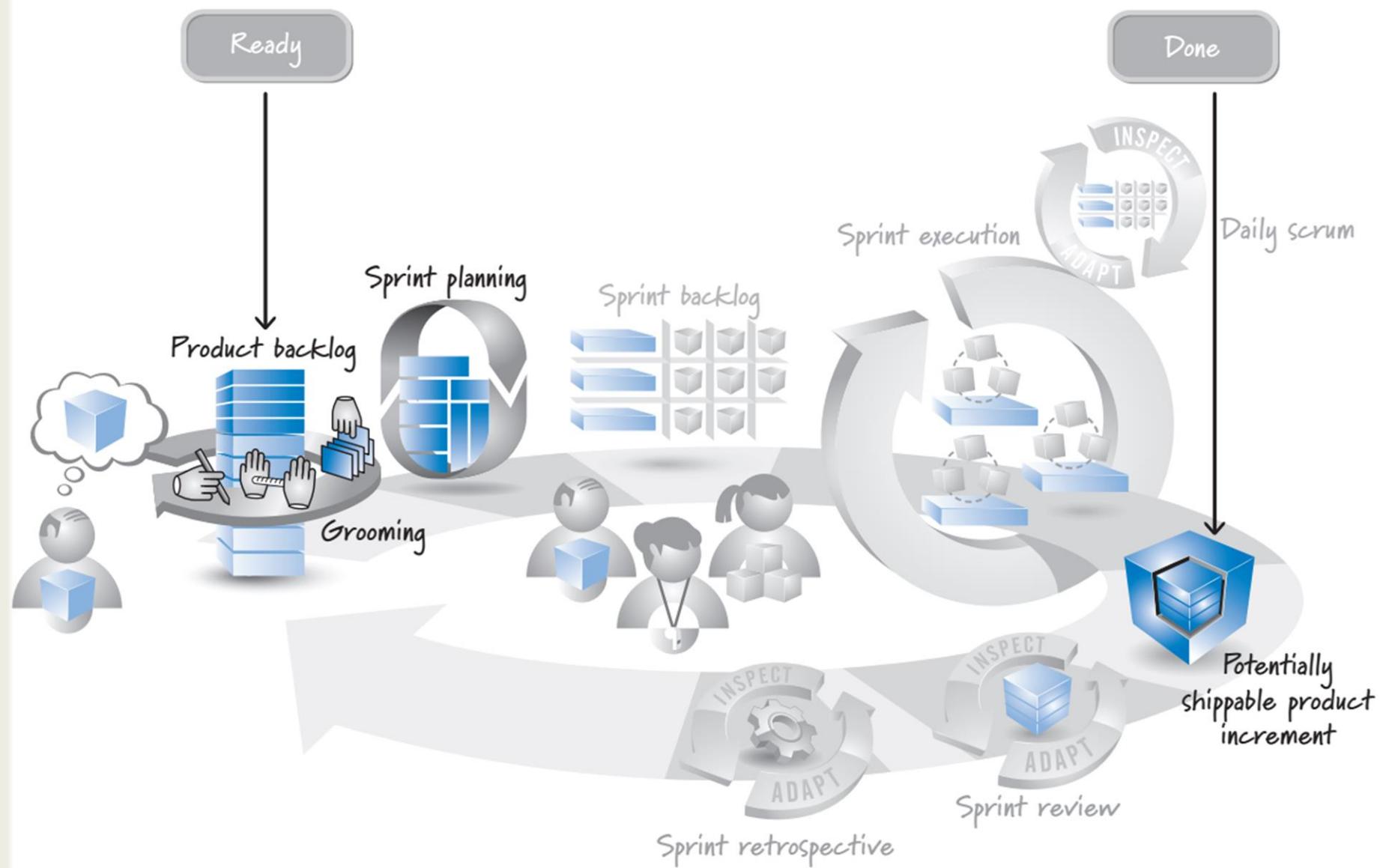
Isfahan University of Technology

[mahmoudzadeh@iut.ac.ir](mailto:mahmoudzadeh@iut.ac.ir)

2022

# Definition of Ready

- Grooming the product backlog should ensure that items at the top of the backlog are ready to be moved into a sprint so that the development team can confidently commit and complete them by the end of a sprint.
- Some Scrum teams formalize this idea by establishing a definition of ready.
- Think definition of ready and the definition of done as two states of product backlog items during a sprint.



# Definition of Ready (Cnt'd)

- Both the **definition of done** and the **definition of ready** are **checklists of the work** that **must be completed** before a product backlog item can be considered to be in the respective state.
- A **strong definition of ready** will substantially **improve** the Scrum team's chance of **successfully meeting** its sprint goal.

## Definition of Ready

واضح و مشخص  
بيان شده

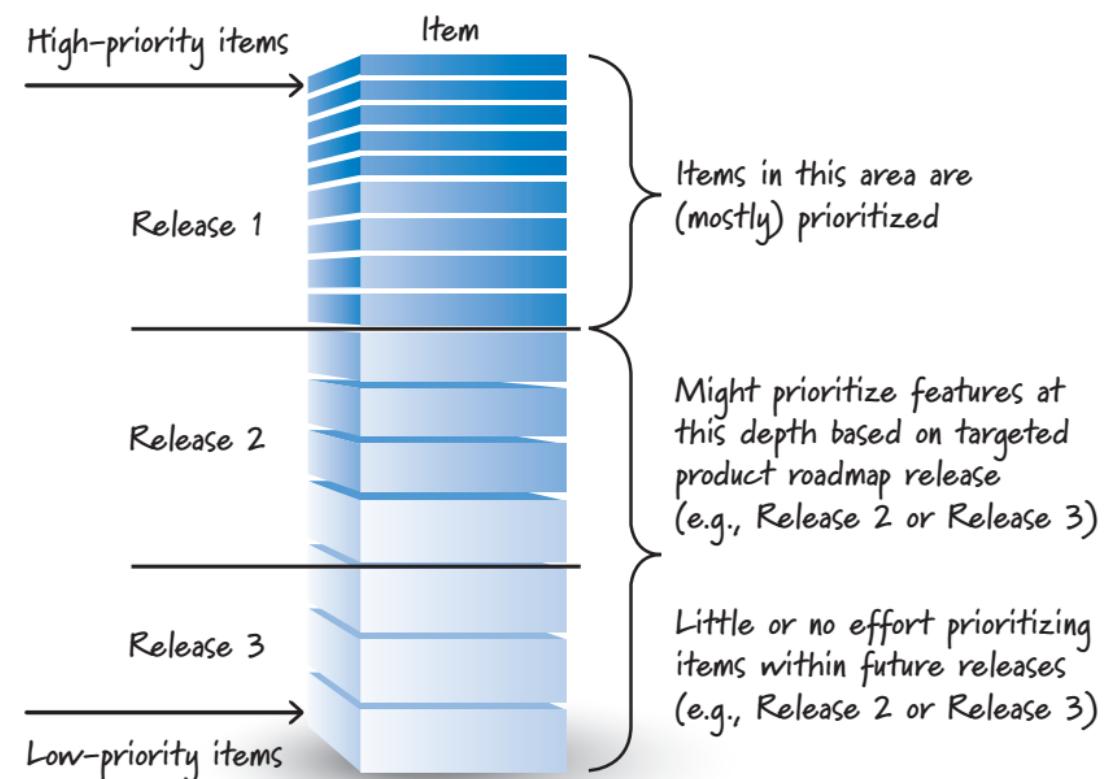
<input type="checkbox"/>	Business value is clearly articulated.
<input type="checkbox"/>	Details are sufficiently understood by the development team so it can make an informed decision as to whether it can complete the PBI.
<input type="checkbox"/>	Dependencies are identified and no external dependencies would block the PBI from being completed.
<input type="checkbox"/>	Team is staffed appropriately to complete the PBI.
<input type="checkbox"/>	The PBI is estimated and small enough to comfortably be completed in one sprint.
<input type="checkbox"/>	Acceptance criteria are clear and testable.
<input type="checkbox"/>	Performance criteria, if any, are defined and testable.
<input type="checkbox"/>	Scrum team understands how to demonstrate the PBI at the sprint review.

# Flow Management

- The product backlog is a crucial tool that enables the Scrum team to achieve fast, flexible value-delivery flow in the presence of uncertainty.
- Uncertainty cannot be eliminated from product development.
- We must assume that a stream of economically important information will be constantly arriving and that we need to organize and manage the work (manage the product backlog) so that this information can be processed in a rapid, cost-effective way while maintaining good flow.

# Release Flow Management

- The product backlog must be groomed in a way that supports ongoing release planning(the flow of features within a release).
- A release can be visualized as a line through the product backlog. All of the PBIs above the release line are targeted to be in that release; the items below the line are not.

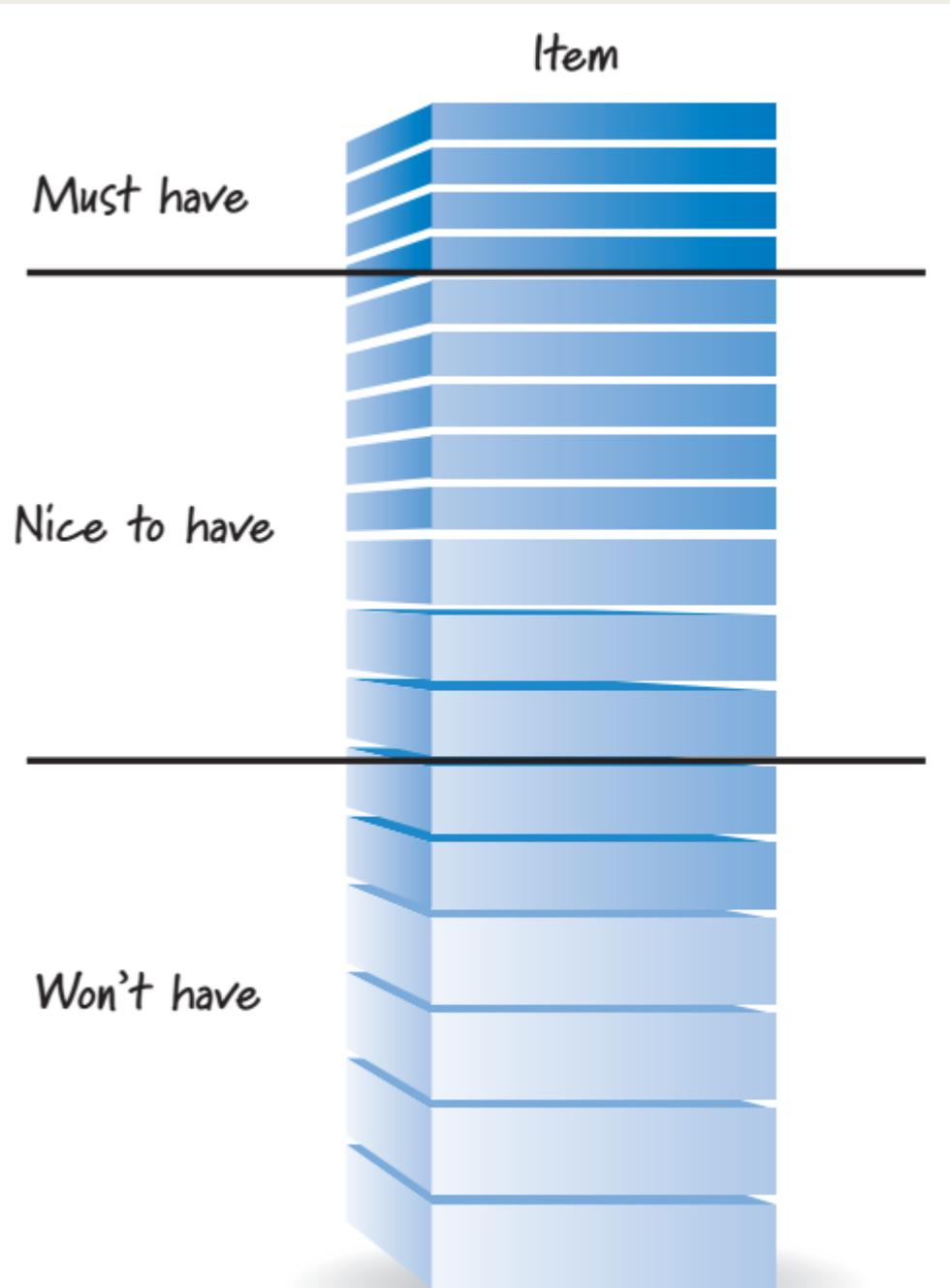


# Release Flow Management

- It is useful to actually partition the product backlog using two lines for each release.
- These two lines partition the backlog into three areas: must have, nice to have, and won't have.
- The must-have features represent the items that we simply must have in the upcoming release or else we don't have a viable customer release.
- The nice-to-have features represent items we are targeting for the next release and would like to include. If, however, we run short of time or other resources, we could drop nice-to-have features and still be able to ship a viable product.
- The won't-have features are items that we're declaring won't be included in the current release.

# Release-level view of the product backlog

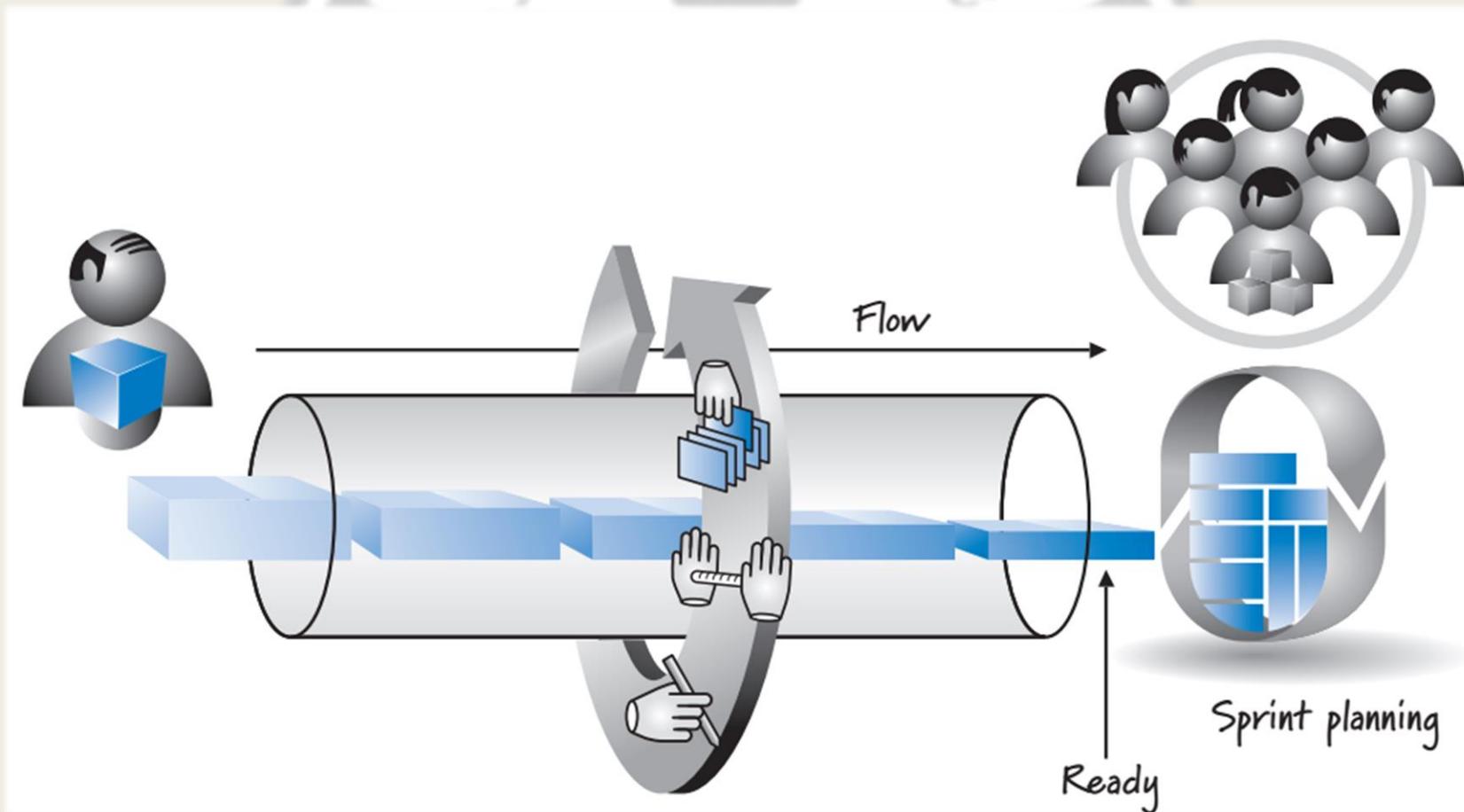
- The second line, the one that separates the won't-have items from the others, is the same as the Release 1 line .
- Maintaining the backlog in this fashion helps us better perform ongoing release planning.



# Sprint Flow Management

- Product backlog grooming is essential for effective sprint planning and the resulting flow of features into a sprint.
- If the product backlog has been detailed appropriately, the items at the top of the backlog should be clearly described and testable.
- When grooming for good sprint flow, it is helpful to view the product backlog as a pipeline of requirements that are flowing into sprints to be designed, built, and tested by the team.

# The product backlog as a pipeline of requirements



# Sprint Flow Management (Cnt'd)

- Larger, less-well-understood requirements are being inserted into the pipeline.
- As they progress through the pipeline and move closer to the time when they will flow out to be worked on, they are progressively refined through the grooming activity.
- At the right side of the pipeline is the team. By the time an item flows out of the pipeline, it must be ready—detailed enough that the team can understand it and be comfortable delivering it during a sprint.

# Sprint Flow Management (Cnt'd)

- If there is ever a mismatch or unevenness between the inflow and outflow of items, we have a problem.
- If the flow of groomed, detailed, ready-to-implement items is too slow, eventually the pipeline will run dry and the team won't be able to plan and execute the next sprint (a major flow disruption or waste in Scrum).
- On the other hand, putting too many items into the pipeline for refinement creates a large inventory of detailed requirements that we may have to rework or throw away once we learn more (a major source of waste).

rework or throw away once we learn more  
(a major source of waste).

Therefore, the ideal situation is to have just enough product backlog items in inventory to create an even flow but not so many as to create waste.

# Sprint Flow Management (Cnt'd)

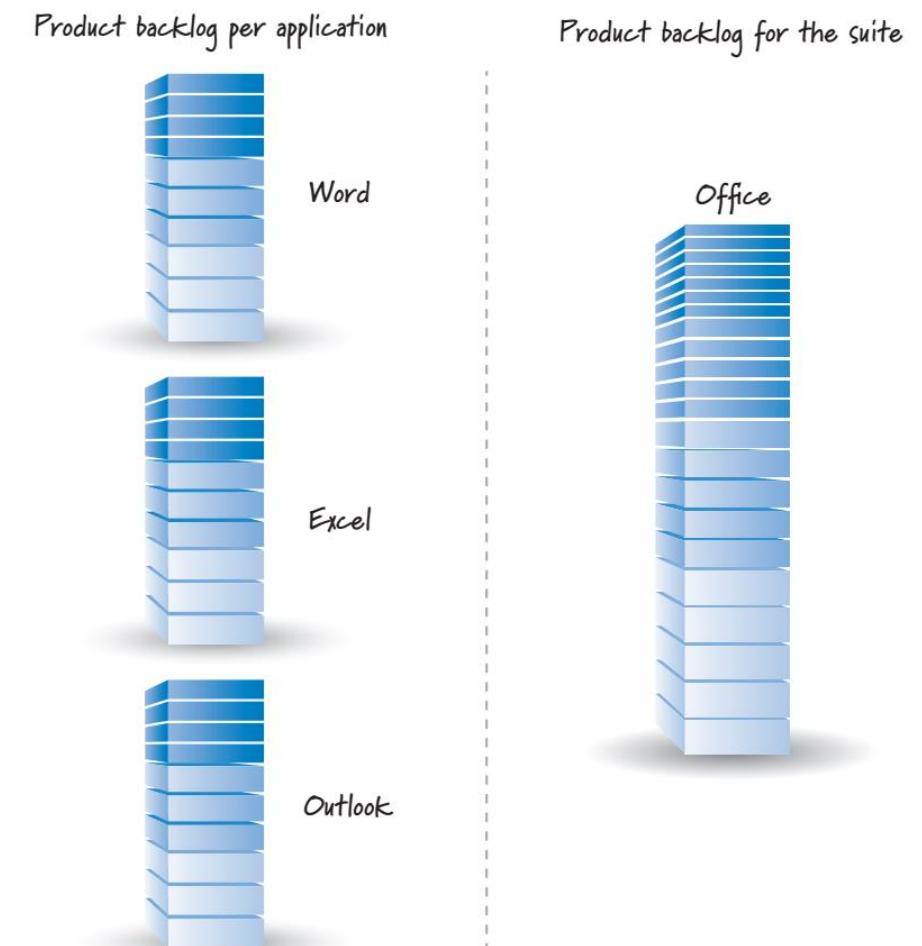
- One approach that Scrum teams use is to have an appropriate inventory of groomed and ready-to-implement items in the backlog.
- A heuristic that seems to work for many teams is to have about two to three sprints' worth of stories ready to go.
- So, for example, if the team can normally do about 5 PBIs per sprint, the team grooms its backlog to always have about 10 to 15 PBIs ready to go at any point in time.
- This extra inventory ensures that the pipeline won't run dry, and it also provides the team with flexibility if it needs to select PBIs out of order for capacity reasons or other sprint-specific constraints.

# Which and How Many Product Backlogs?

- When deciding on which and how many product backlogs to form, I start with a simple rule: one product, one product backlog, meaning that each product should have its own single product backlog that allows for a product-wide description and prioritization of the work to be done.
- There are, however, some occasions when we need to exercise care when applying this rule to ensure that we end up with a practical, workable product backlog structure.

# What Is a Product?

- Is Microsoft Word the product, or is it simply one facet of a larger product called Microsoft Office?



# What Is a Product? (Cnt'd)

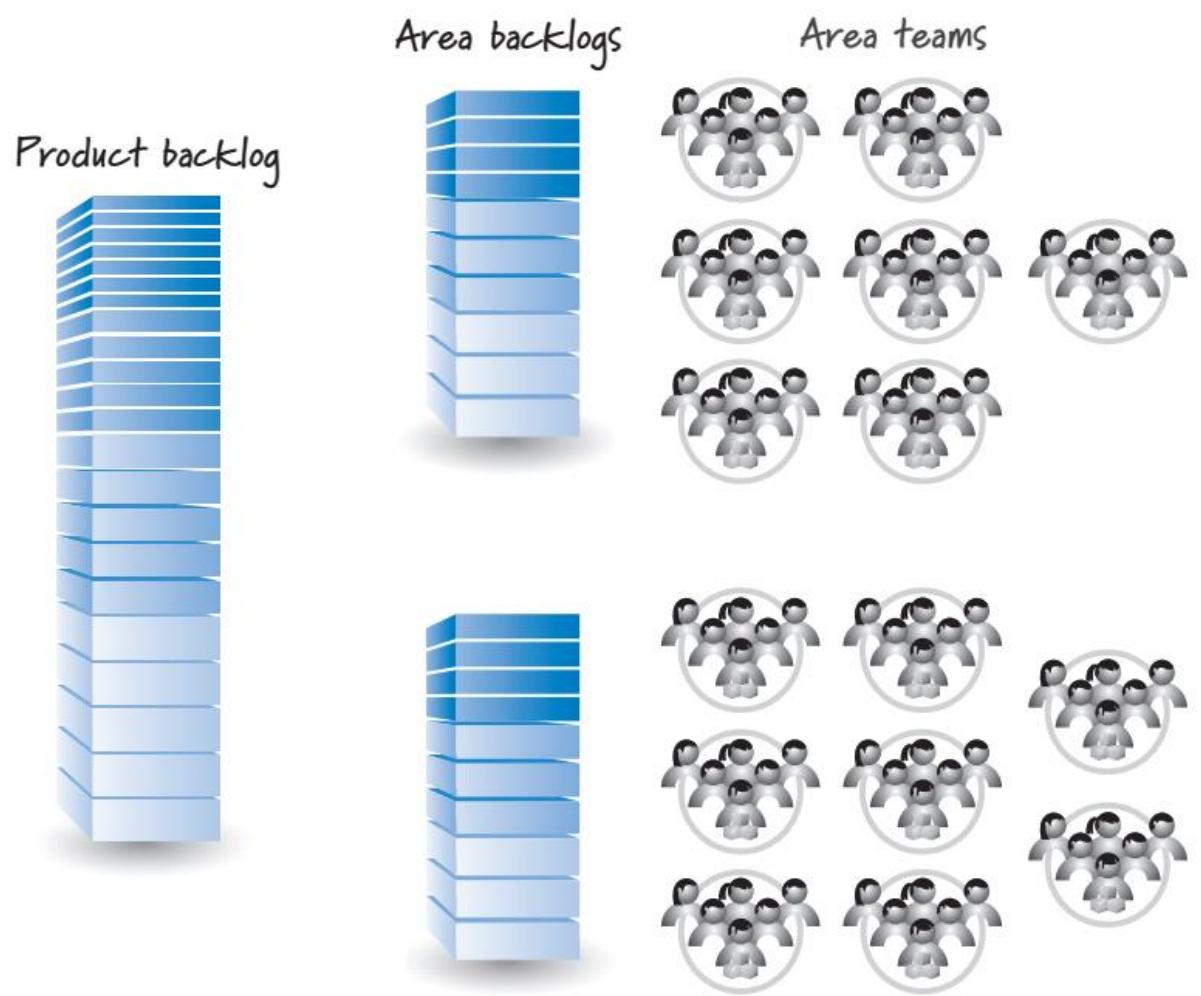
- IBM sold products from a catalog, so if you could put a PID on it, salespeople could include it on an order form and therefore it was a “product.”
- A product is something of value that a customer would be willing to pay for and something we’re willing to package up and sell.
- Think about what you create that is packaged, delivered, and adds end-customer value. Then align your product backlog with that offering.

# Large Products—Hierarchical Backlogs

- Whenever possible, I prefer one product backlog even for a large product like Microsoft Office.
- However, we need to be practical when applying this rule. On a large product development effort to create something like a cell phone, we can have many tens or hundreds of teams whose work must all come together to create a marketable device. Trying to put the PBIs from all of these teams into one manageable product backlog isn't practical (or necessary).
- Not all of these teams work in related areas.

# Large Products—Hierarchical Backlogs (Cnt'd)

- Most organizations address the large-product problem by creating hierarchical.



# Large Products—Hierarchical Backlogs (Cnt'd)

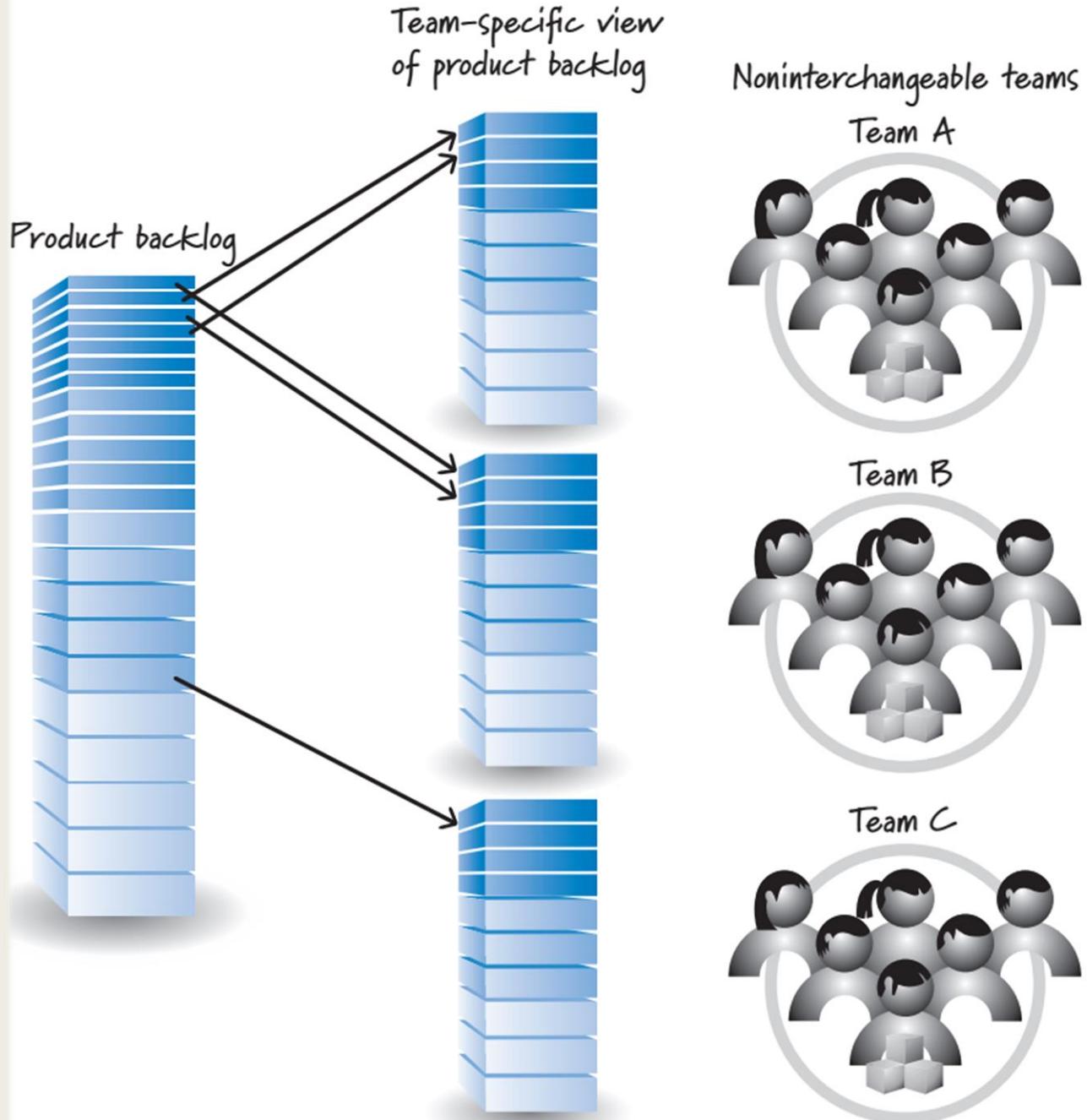
- At the top of the hierarchy we still have the one product backlog that describes and prioritizes the large-scale features (perhaps epics) of the product.
- Each of the related feature areas then has its own backlog. The PBIs at the feature-area level will likely be smaller in scale (feature or story size) than the corresponding items in the product backlog.

# Multiple Teams—One Product Backlog

- The one-product-one-product-backlog rule is designed to allow all of the teams working on the product to share a product backlog.
- Aligning all of the teams to a single backlog enables us to optimize our economics at the full-product level.
- We get this benefit because we put all of the features into one backlog and make them compete for priority against all other features, ensuring that the highest-priority features from the full-product perspective are identified and prioritized to be worked on first.
- If all of our teams are interchangeable, so that any team can work on any PBI in the one shared backlog, we actually get to realize the prioritization benefit enabled by the single product backlog.

# Multiple Teams—One Product Backlog (Cnt'd)

- But what if the teams aren't interchangeable?
- For example, a team that works on the Microsoft Word text-layout engine probably can't be assigned to work on the Microsoft Excel calculation engine.
- While not ideal, in some cases, not every team can work on every item in the product backlog.
- To work within this reality, we must know which items in the product backlog each team can work on.

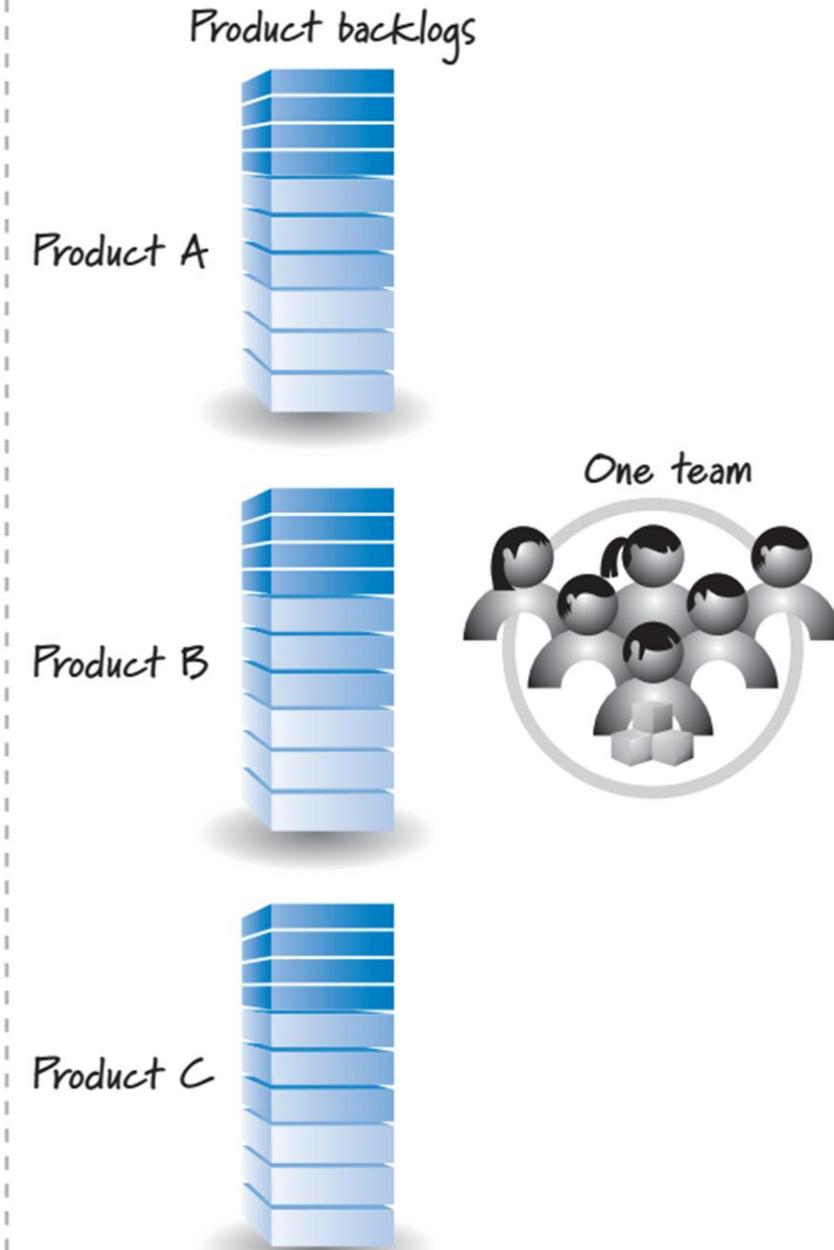
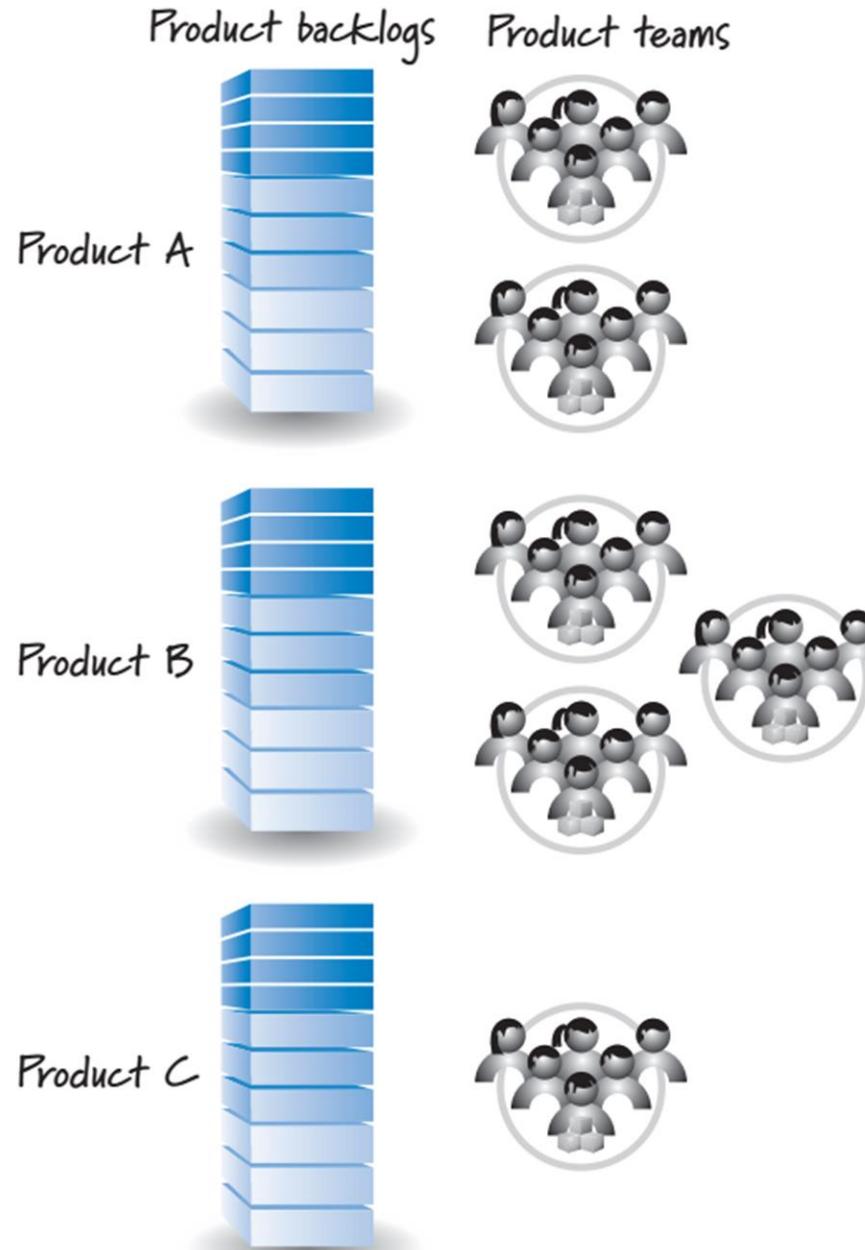


# Multiple Teams—One Product Backlog (Cnt'd)

- We have team-specific views of the shared backlog.
- There is one backlog, but it is structured in such a way that teams see and choose from only the features that are relevant to their skill sets.
- The highest-level item in the team C backlog is derived from an item that is not a very high priority in the product-level backlog.
- If the teams were interchangeable, team C's backlog would correspond to much higher priority product-level backlog items.
- This lack of flexibility is why many organizations strive for a high level of shared code ownership and more interchangeable teams, so that they too can reap the benefits that come from having teams that can work on multiple areas of the product.

# One Team—Multiple Products

- If an organization has multiple products, it will have multiple product backlogs.
- The best way to handle multiple product backlogs is to assign one or more teams to work exclusively on each product.



# One Team—Multiple Products

- Our goal should be to minimize the amount of multi-projecting that teams or team members perform.
- The first, and often the best, solution is to have the team work on one product at a time. In each sprint the team works only on the items from one product backlog.
- However, if organizational impediments force us to have the single team work on multiple products concurrently, we might consider merging the PBIs for all three products into one product backlog. This would require that the product owners for the three products come together and reach a single prioritization across all of the products.

# Reference

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

# Estimation and Velocity(I)

Dr. Elham Mahmoudzadeh

Isfahan University of Technology

[mahmoudzadeh@iut.ac.ir](mailto:mahmoudzadeh@iut.ac.ir)

2022

# Introduction

- When **planning** and **managing** the **development** of a product, we need to answer important questions.
  - “**How many features will be completed?**”
  - “**When will we be done?**”
  - “**How much will this cost?**”
- We need to **estimate the size** of what we are building and **measure the velocity** or **rate** at which we can **get work done**.
- With that information, we can **derive** the likely product development **duration** (and the **corresponding cost**).

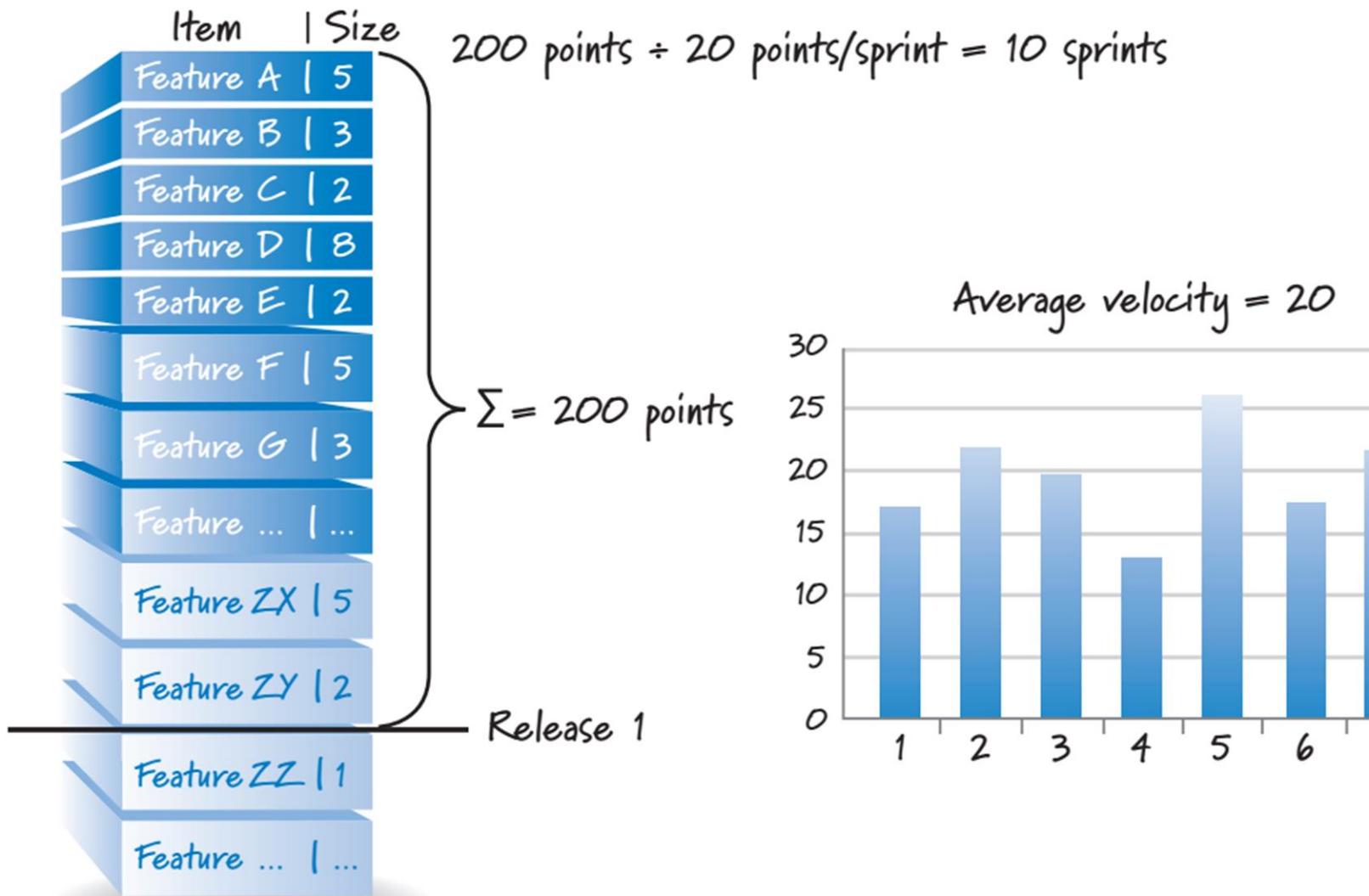
# Introduction(Cnt'd)

- Once we know the **approximate size of the release**, we turn our attention to the **team's velocity**, **how much work the team typically gets done each sprint.**
- **Velocity** is **easy to measure**. At the end of each sprint, we simply **add the size estimates** of every item that was completed during the sprint; if an item **isn't done**, it **doesn't count toward velocity**.
- The **sum of the sizes of all the completed product backlog items** in a sprint is the **team's velocity** for that sprint.

# Introduction(Cnt'd)

- Now that we have **estimated size** and **measured velocity**, we are in a position to **calculate (derive) the duration**.
- To do this, we simply **divide the size by the velocity**.

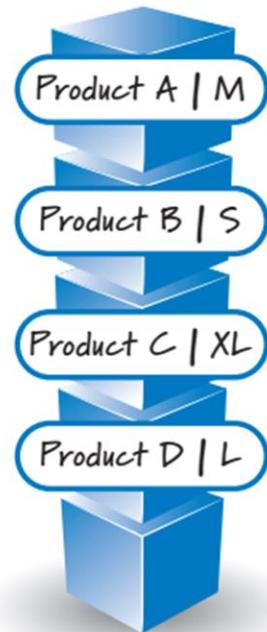
Estimated size ÷ measured velocity = (number of sprints)



# What and When We Estimate

- Throughout the development life of a product, however, we need to estimate at varying levels of granularity and, thus, will use different units to do so.

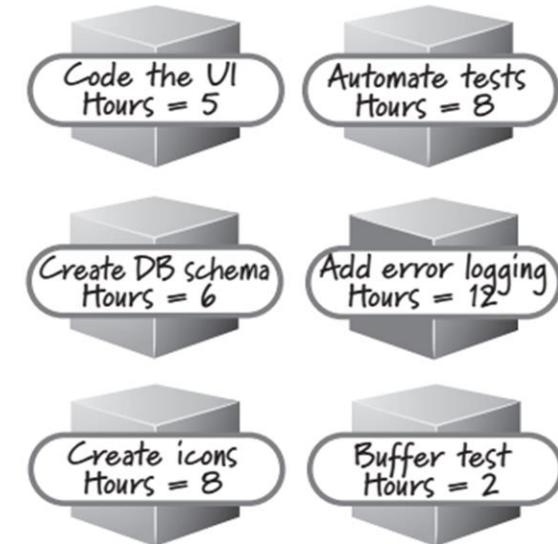
### Portfolio backlog



### Product backlog



### Sprint backlog tasks



Item	Portfolio backlog	Product backlog	Sprint backlog tasks
Unit	T-shirt sizes	Story points / ideal days	Ideal hours / effort-hours
When	Portfolio planning	Product backlog grooming	Sprint planning

# Portfolio Backlog Item Estimates

- Many organizations maintain one that contains a prioritized list of all of the products (or projects) that need to be built.
- To properly prioritize a portfolio backlog item we need to know the approximate cost of each item.
- Typically won't have a complete, detailed set of requirements at the time when this cost number is initially requested, so we can't use the standard technique of estimating each individual, detailed requirement and then summing those estimates to get an aggregate estimate of the total cost.
- Instead, to estimate portfolio backlog items, many organizations choose to use rough, relative size estimates like T-shirt sizes (such as small, medium, large, extralarge, and so on).

# Product Backlog Estimates

- Once a product or project is approved and we start adding more detail to its product backlog items, however, we need to estimate differently.
- When PBIs have risen in priority and been groomed to include more detail, most teams prefer to put numeric size estimates on them, using either story points or ideal days.

# Product Backlog Estimates(Cnt'd)

- Estimating PBIs is part of the overall product backlog grooming activity.
- Typically, PBI estimation occurs in “estimation meetings,”
- The product owner might also call additional estimation meetings during a sprint if any new PBIs need to be estimated.

# Reasons of PBI estimation

- Not all PBIs will be at the same size at the same time, so there will be some larger PBIs in the backlog even if we do have a collection of smaller, similarly sized items toward the top.
- Finally, and most importantly, one of the primary values of estimation is the learning that happens during the estimation conversations.

# Task Estimates

- At the most detailed level we have the tasks that reside in the sprint backlog.
- Most teams choose to size their tasks during sprint planning so that they can acquire confidence that the commitments they are considering are reasonable.
- Tasks are sized in ideal hours.
- The estimate simply states how much of the team's effort is expected to complete the task.

# PBI Estimation Concepts



PBI estimating concepts

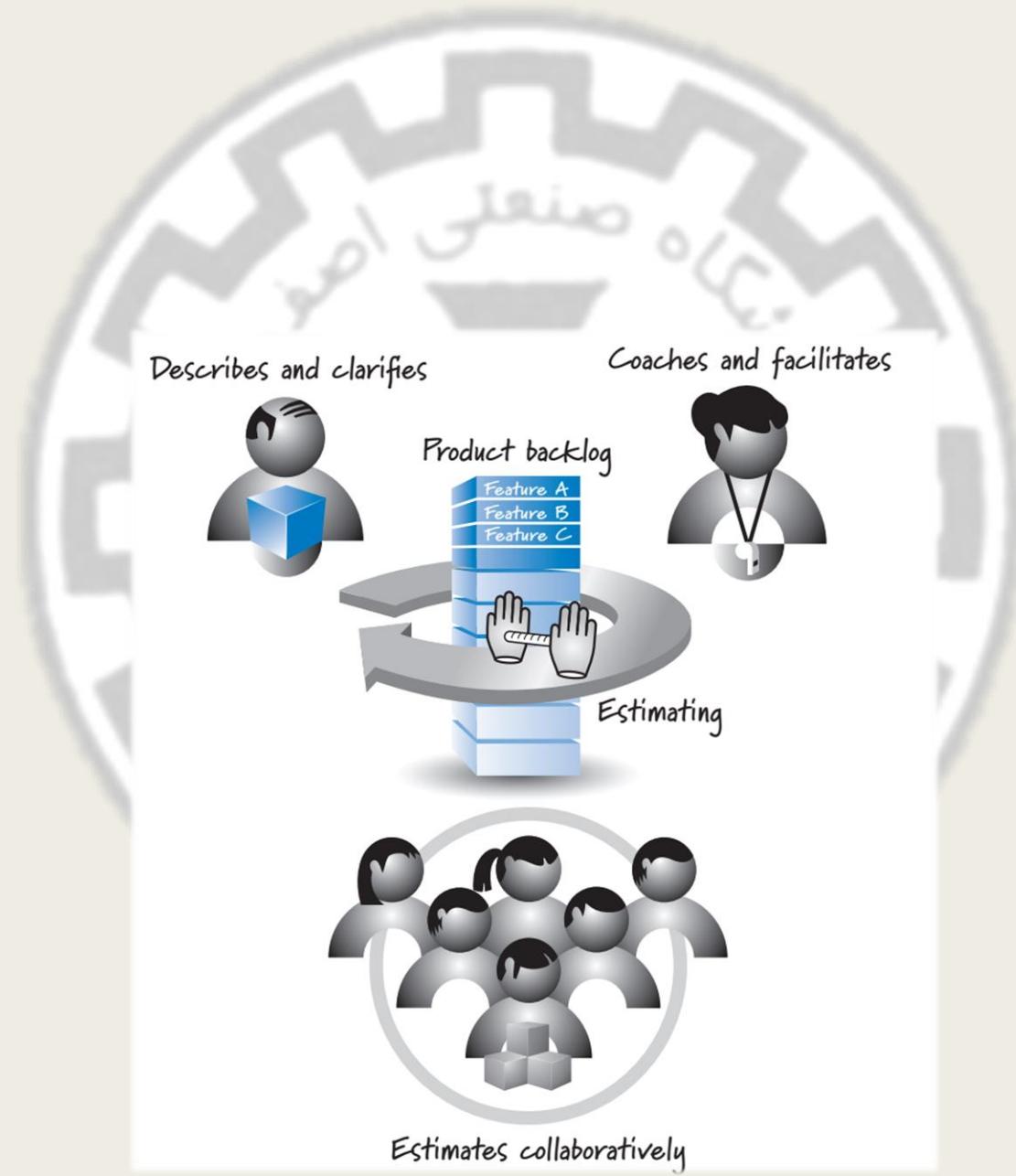
- Estimate as a team
- Estimates are not commitments
- Focus on accuracy, not precision
- Use relative versus absolute sizes

# Estimate as a Team

- In many traditional organizations the project manager, product manager, architect, or lead developer might do the initial size estimation. Other team members might get a chance to review and comment on those estimates at a later time.
- In Scrum, we follow a simple rule: The people who will do the work collectively provide the estimates. To be clear, when I say people who will do the work, I mean the development team that will do the hands-on work to design, build, and test the PBIs. The product owner and ScrumMaster don't provide estimates. Both of these roles are present when the PBIs are being estimated, but they don't do any hands-on estimation.

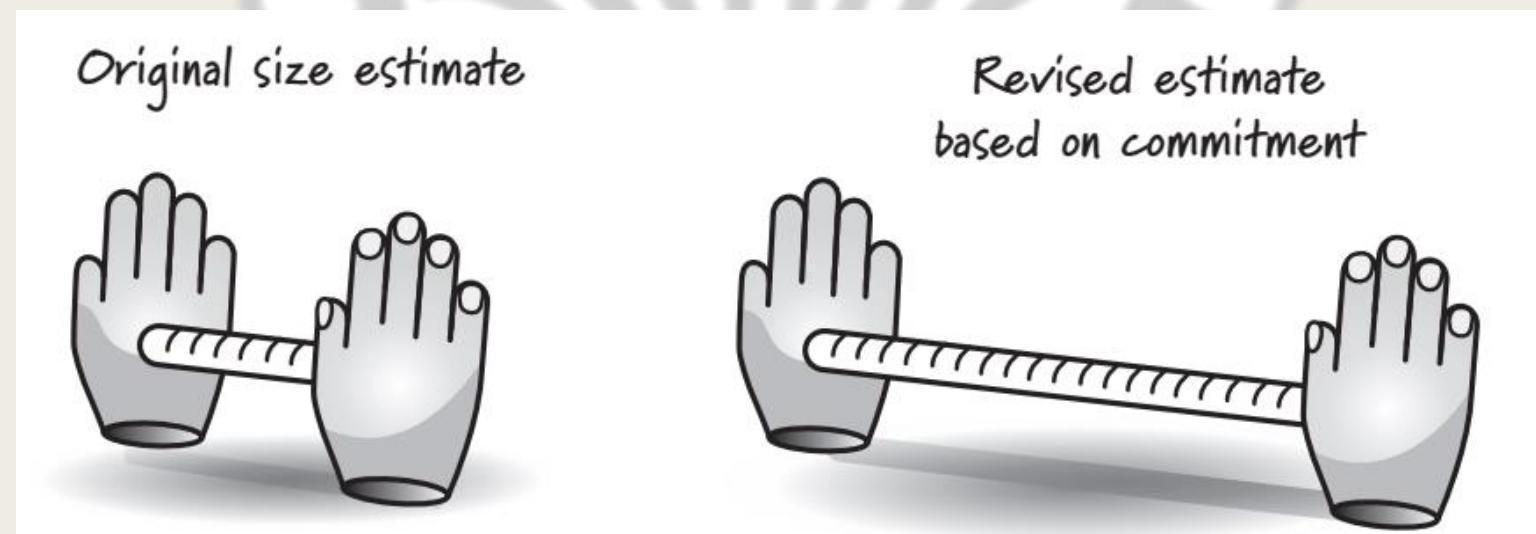
# Estimate as a team(Cnt'd)

- The product owner's role is to describe the PBIs and to answer clarifying questions that the team might ask. The product owner should not guide or “anchor” the team toward a desired estimate.
- The ScrumMaster's role is to help coach and facilitate the estimation activity.
- The goal is for the development team to determine the size of each PBI from its collective perspective.
- Because everyone sees a story from a different point of view, depending on his area of expertise, it is important that all members of the development team participate during estimation.



# Estimates Are Not Commitments

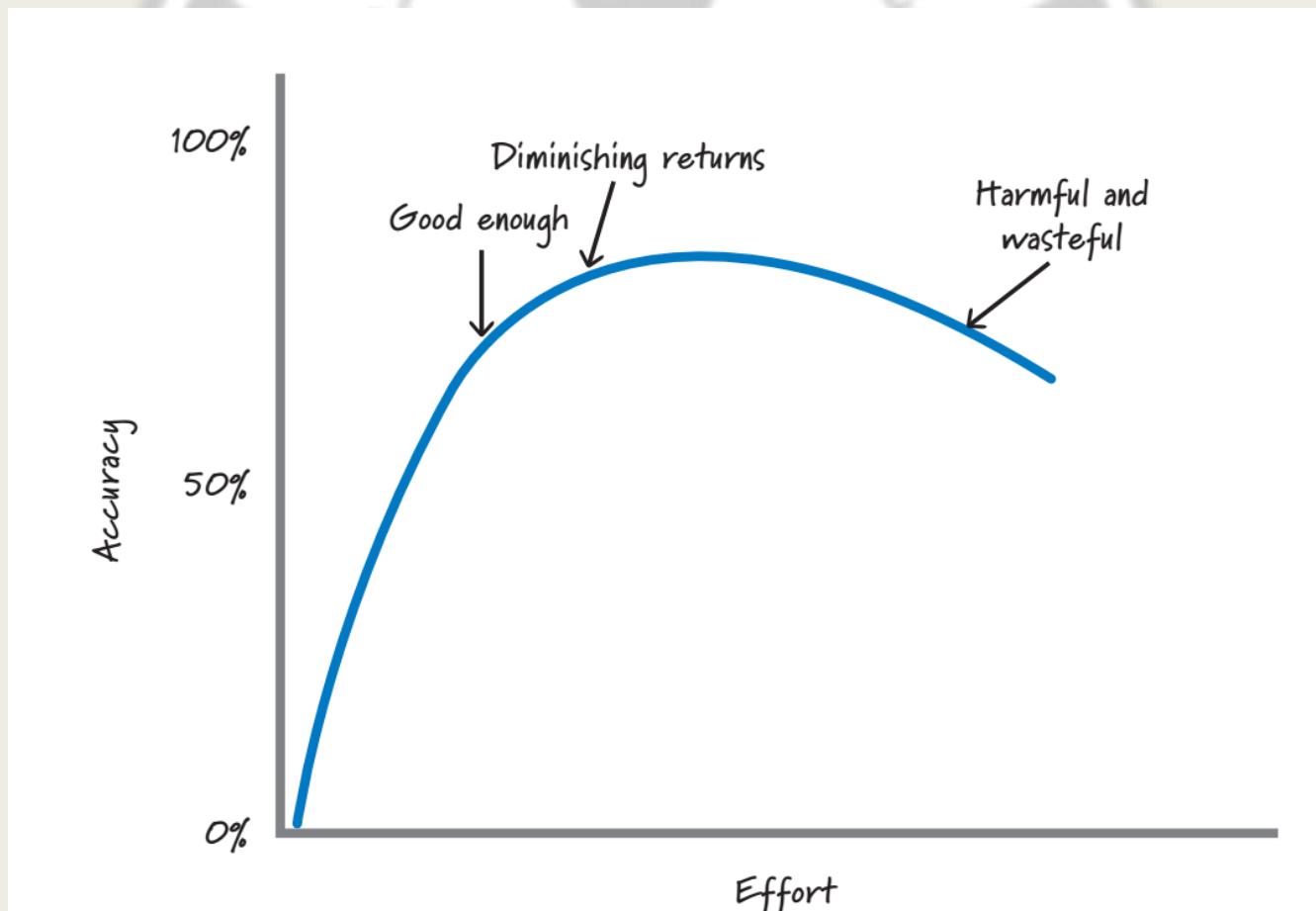
- Estimates are not commitments, and it is important that we **not treat them as such.**



# Accuracy versus Precision

- Our estimates should be accurate without being overly precise.
- Generating overly precise estimates is wasteful.
  - First, there is the wasted effort of coming up with the estimate, which can be considerable.
  - Second, there is the waste that occurs when we deceive ourselves by thinking we understand something that we don't, and then make important, wrong, and costly business decisions based on this deception. We should invest enough effort to get a good-enough, roughly right estimate.

# Effort versus accuracy when estimating

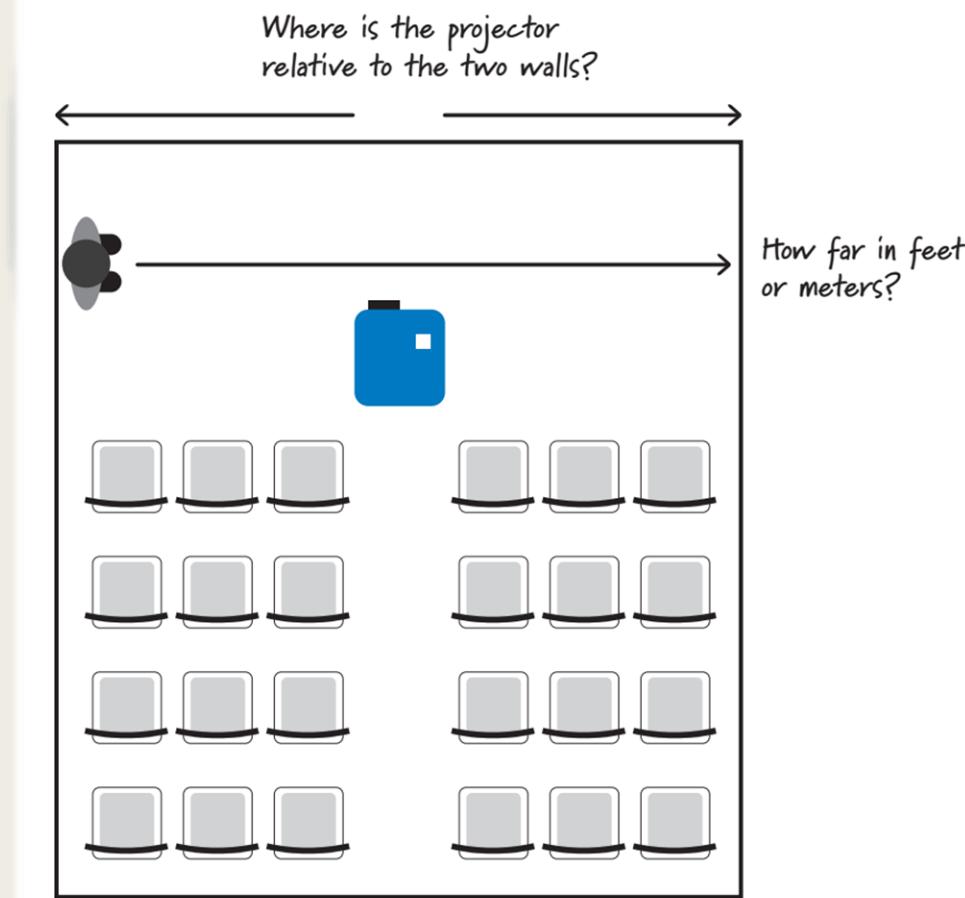


# Relative Size Estimation

- We should estimate PBIs using relative sizes, not absolute sizes.
- We compare items to determine how large an item is relative to the others.
- people are much better at relative size estimation than absolute size estimation



# Absolute versus relative size estimation: An example



# PBI Estimation Units

- Although there is no standard unit for PBI size estimates, by far the two most common units are story points and ideal days.
- There isn't a right or wrong choice when deciding between these two.

# Story Points

- Measure the **bigness** or **magnitude** of a PBI.
- Be influenced by several factors, such as **complexity** and **physical size**.
  - *The story might represent the development of a complex business algorithm. The end result won't be very large, but the effort required to develop it might be.*
  - *On the other hand, a story might be physically quite big but not complex.*
- Story points **combine factors** like **complexity** and **physical size** into one **relative size measure**.
- The goal is to be able to compare stories.
- Must reflect the **effort** associated with the story from the development<sup>13</sup>

# Ideal Days

- An **alternative** approach for **estimating PBIs** is to **use ideal days**.
- Ideal days are **a familiar unit**—they **represent the number of effort-days** or **person-days** needed to **complete a story**.
- An important factor against ideal time is the **risk of misinterpretation**.

# Comparison: An example

- For example, it's currently early afternoon on Tuesday and I show you a PBI and ask, "How big is this PBI?" You say, "Two days." I say, "OK, so you'll be done Thursday early in the afternoon." You say, "No, I'm finishing up a two-day activity this afternoon and tomorrow [Wednesday]."
- I need the entire day just to get caught up, so I can probably start the PBI on Thursday. But since I don't have any full days to dedicate to the PBI, I'm thinking I should be done sometime next Monday." I then say, "I don't understand; you told me it was a two-day PBI, so you should be done on Thursday." You say, "I said two ideal days, not two calendar days.

# Reference

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

# Estimation and Velocity(II)

Dr. Elham Mahmoudzadeh  
Isfahan University of Technology

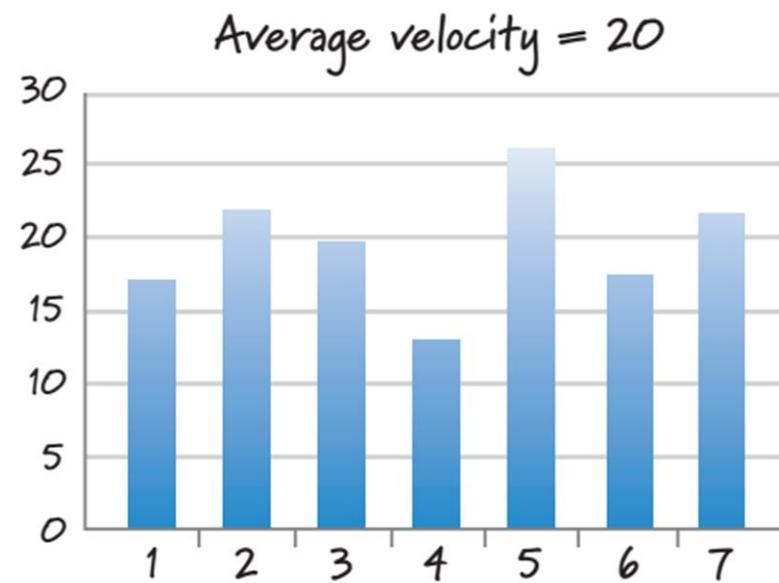
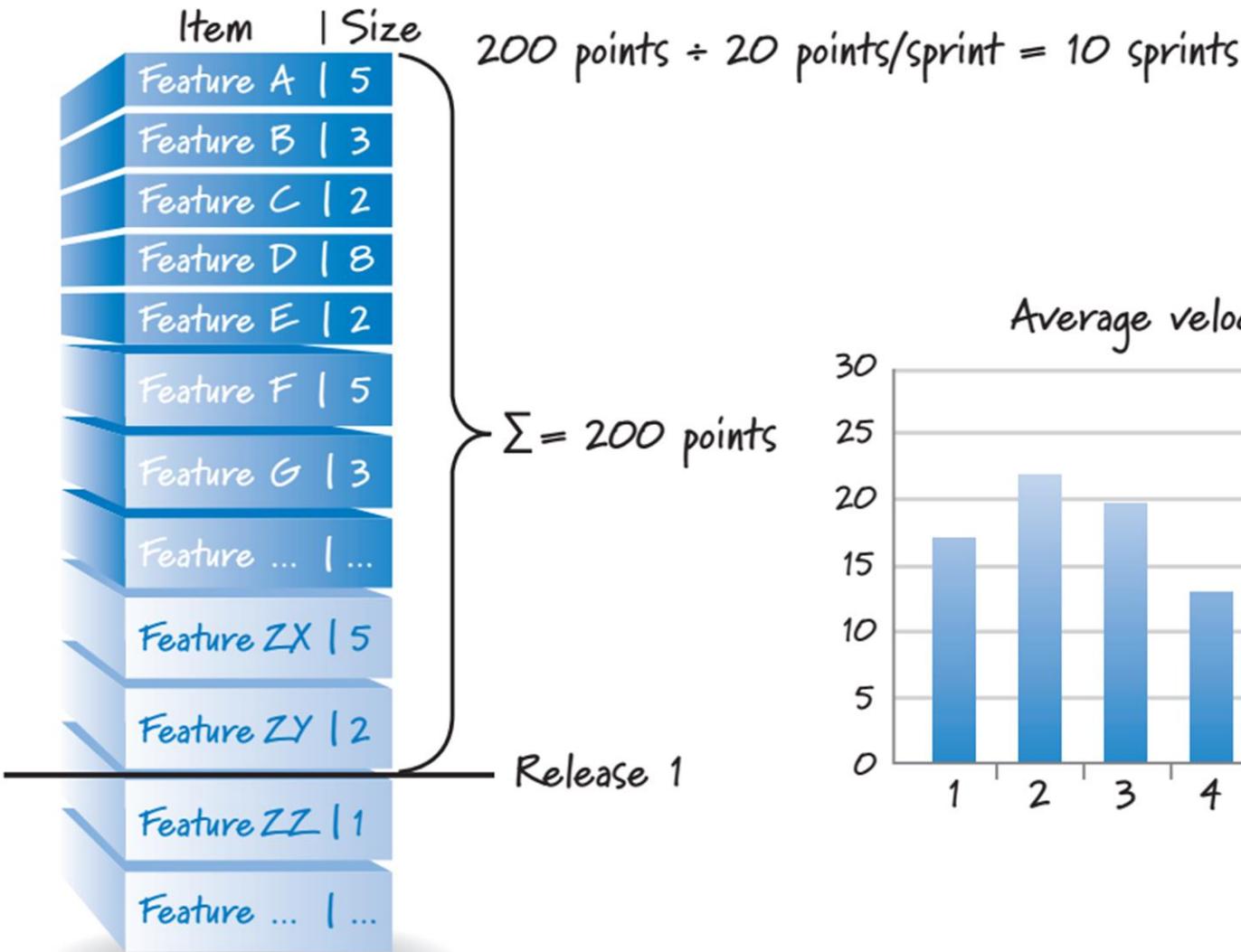
[mahmoudzadeh@iut.ac.ir](mailto:mahmoudzadeh@iut.ac.ir)

2022

# Introduction

- When **planning** and **managing** the **development** of a product, we need to answer important questions.
  - “**How many features will be completed?**”
  - “**When will we be done?**”
  - “**How much will this cost?**”
- We need to **estimate** the **size** of what we are building and **measure** the **velocity** or **rate** at which we can get work done.
- With that information, we can derive the likely **product development duration** (and the corresponding cost).

Estimated size ÷ measured velocity = (number of sprints)



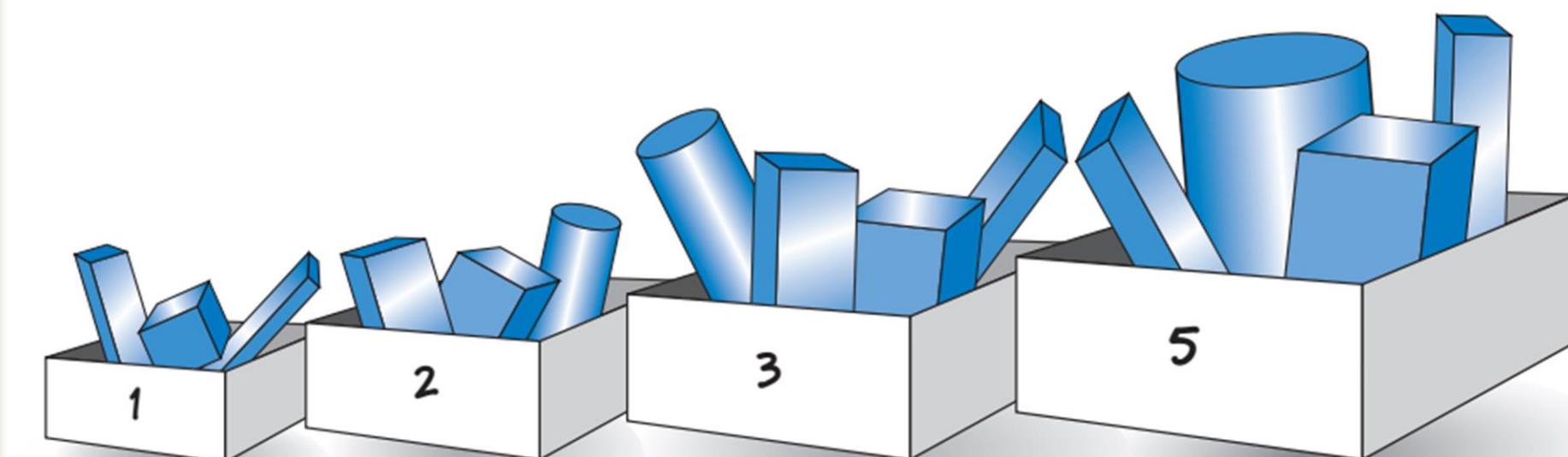
# Technique for sizing PBIs

- Consensus-based.
- Knowledgeable people (the experts) slate to work on a PBI engage in an intense discussion to expose assumptions, acquire a shared understanding, and size the PBI.
- Relative size estimates by accurately grouping or binning together items of similar size.
- The team leverages its established PBI estimation history to more easily estimate the next set of PBIs.

# Estimation Scale

- The team must decide which scale or sequence of numbers it will use for assigning estimates. Because our goal is to be accurate and not overly precise, we prefer to not use all of the numbers.
- When we receive a package, we need to decide which bin to place the package in. Now, not all packages in the same bin are or will be identically the same physical shape, size, or weight, so we need to examine the packages that are currently in the bins so that we can find the best-fit bin for the package we are estimating. Once we find the closest matching bin, we put the package in the bin and move on to the next package.
- Obviously, the more packages we put into the bins, the easier it should be to size and bin future packages because we'll have more points of comparison.

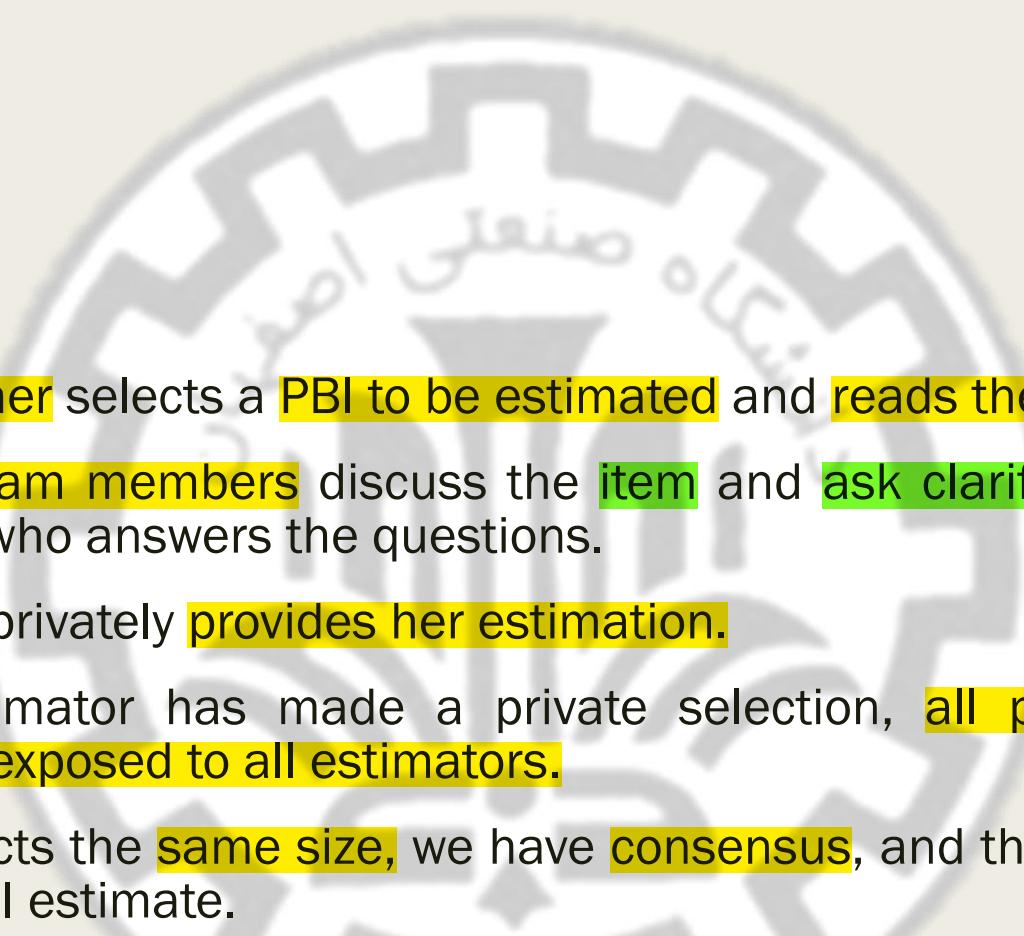
# Binning



# How

- The full Scrum team participates.
- During the session, the product owner presents, describes, and clarifies PBIs.
- The ScrumMaster coaches the team to help it better applying. The ScrumMaster is also constantly looking for people who, by their body language or by their silence, seem to disagree and helping them engage.
- And the development team is collaboratively generating the estimates.

# The rules

- 
1. The product owner selects a PBI to be estimated and reads the item to the team.
  2. Development team members discuss the item and ask clarifying questions to the product owner, who answers the questions.
  3. Each estimator privately provides her estimation.
  4. Once each estimator has made a private selection, all private estimates are simultaneously exposed to all estimators.
  5. If everyone selects the same size, we have consensus, and that consensus number becomes the PBI estimate.
  6. If the estimates are not the same, the team members engage in a focused discussion to expose assumptions and misunderstandings. Typically we start by asking the high and low estimators to explain or justify their estimates.
  7. After the discussion, we return to step 3 and repeat until consensus is reached.

# The estimation

- The goal is not to compromise, but instead for the development team to reach a consensus about the estimate of the story's overall size (effort) from the team perspective.
- Usually this consensus can be achieved within two or three rounds of voting, during which the team members' focused discussion helps obtain a shared understanding of the story.

# Benefits

- Brings together the diverse team of people who will do the work and allows them to reach consensus on an accurate estimate that is frequently much better than any one individual could produce.
- Discussion and better understanding that team members will share about the PBIs.

# What is Velocity?

- Is the amount of work completed each sprint.
- It is measured by adding the sizes of the PBIs that are completed by the end of the sprint.
- A PBI is either done or it's not done.
- The product owner doesn't get any value from undone items, so velocity does not include the size numbers of partially completed PBIs.
- Velocity measures output (the size of what was delivered), not outcome (the value of what was delivered).
- We assume that if the product owner has agreed that the team should work on a PBI, it must have some value to him.

# Calculate a Velocity Range

- For planning purposes, velocity is most useful when expressed as a range, such as “The team is typically able to complete between 25 and 30 points each sprint.”
- Using a range allows us to be accurate without being overly precise.
- With a velocity range we can more accurately provide answers to questions like “When will we be done?” “How many items can we complete?” or “How much will all this cost?”
- By using a range, we can communicate our uncertainty.

Item	Size
Feature A	5
Feature B	3
Feature C	2
Feature D	8
Feature E	2
Feature F	5
Feature G	3
Feature ...	...
Feature ZX	5
Feature ZY	2
Feature ZZ	1
Feature ...	...

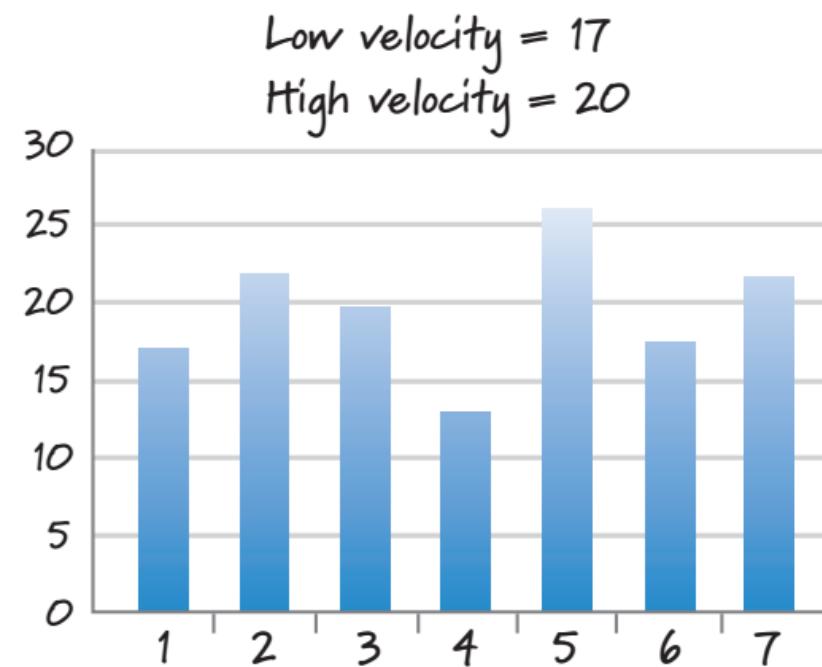
Release 1 will need 10 to 12 sprints to complete

$$200 \text{ points} \div 20 \text{ points/sprint} = 10 \text{ sprints}$$

$$200 \text{ points} \div 17 \text{ points/sprint} = 12 \text{ sprints}$$

$$\Sigma = 200 \text{ points}$$

Release 1



# Historical velocity

- The team had historical velocity data that we could use to predict future velocity.
- Certainly one of the benefits of having long-lived teams is that they will acquire such useful historical data.

# Forecasting Velocity

- But how do we handle the situation where we have a new team whose members haven't worked together and therefore have no historical data? We'll have to forecast it.
- One common way to forecast is to have the team perform sprint planning to determine what PBIs it could commit to delivering during a single sprint. If the commitment seems reasonable, we would simply add the sizes of the committed PBIs.
- Because what we really want is a velocity range, we could have the team plan two sprints and use one estimated velocity number as the high and the other as the low.

# Forecasting Velocity(Cnt'd)

- Alternatively, we could make some intuitive adjustments to one estimated velocity based on historical data for other teams.
- As soon as the team has performed a sprint and we have an actual velocity measurement, we should discard the forecast and use the actual.

# Increasing velocity

- If the team is constantly inspecting and adapting (continuously improving), its velocity should keep getting better and better. It is expected that a team that is aggressively trying to improve itself and is focused on delivering features in accordance with a robust definition of done to see an increase in velocity.
- There are a number of ways that the Scrum team and managers can help get velocity to the next plateau.
  - Introducing new tools or increasing training can have a positive effect on velocity.
  - Managers can strategically change team composition with the hope that the change will eventually lead to a greater overall velocity.
    - haphazardly moving people on and off teams can and probably will cause velocity to decline.

# Increasing velocity(Cnt'd)

- Although introducing new tools, getting training, or changing team composition can have a positive effect on velocity, these actions usually cause a dip in velocity while the team absorbs and processes the change.  
A large, faint watermark of a circular logo is visible in the background. The logo features a stylized figure in the center, surrounded by Arabic script at the top and English script at the bottom. A red arrow points from the word 'شيب' (dip) to a yellow rectangular callout box containing the Arabic word 'شيب' (dip). Another red arrow points from the word 'plateau' to a yellow rectangular callout box containing the Arabic word 'فلات' (plateau).  
شيب
- After this decline, there will probably be an increase to the point where the team establishes a new plateau until some other change causes yet another plateau to be achievable.

# Affecting Velocity

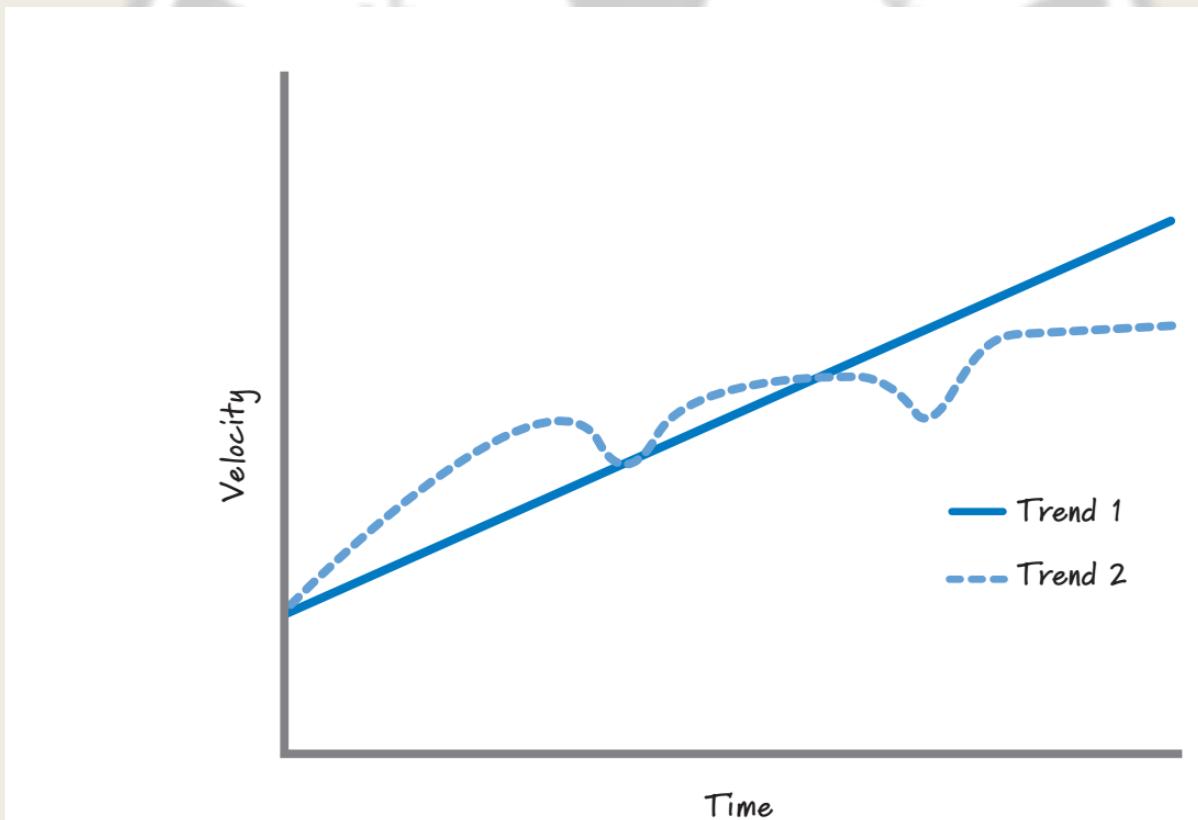


FIGURE 7.13 A team's velocity over time

# The effect of overtime on velocity

- there is one obvious thing we could do to try to improve velocity:  
work longer hours. Working a lot of consecutive overtime might initially cause velocity to increase.

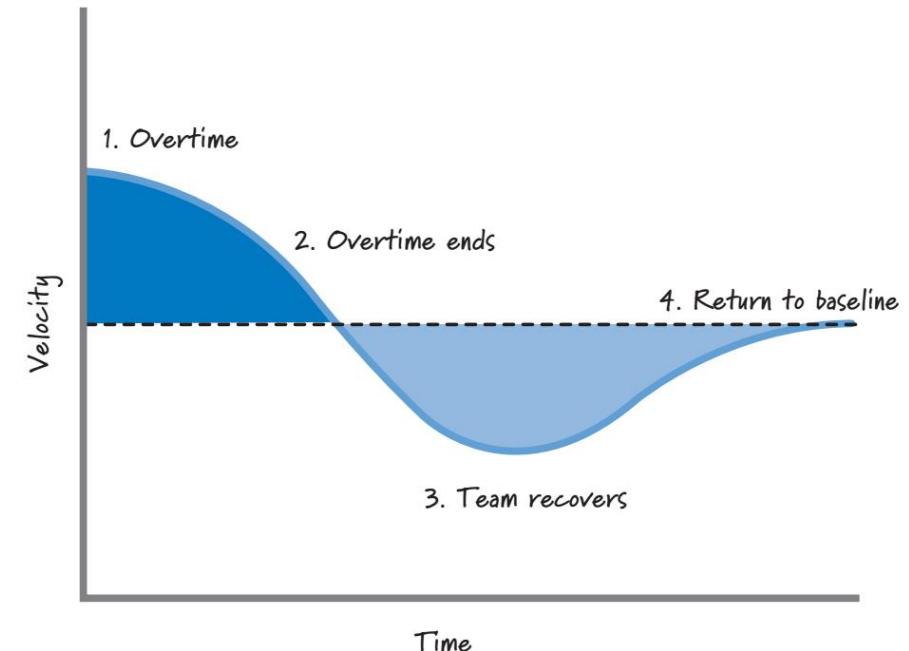


FIGURE 7.14 The effect of overtime on velocity (based on a figure from Cook 2008)

# The effect of overtime on velocity(Cnt'd)

- That increase will almost certainly be followed by an aggressive decline in velocity along with a simultaneous decline in quality.
- Even after the overtime period ends, the team will need some amount of time to recover before returning to its reasonable baseline velocity.
- Maybe the trough (decreased velocity area) during the recovery period is larger than the crest (increased velocity area) during the overtime period.
- The end result is that lots of overtime may provide some short-term benefits, but these are frequently far outweighed by the long-term consequences.

# Misusing Velocity

- Velocity is used as a planning tool and as a team diagnostic metric. It should not be used as a performance metric in an attempt to judge team productivity.
- When misused, velocity can motivate wasteful and dangerous behavior.
- We should judge velocity on how well it assists us with performing accurate planning and how well it helps a team to internally improve itself.
- Any other uses will likely promote the wrong behavior.

# Reference

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