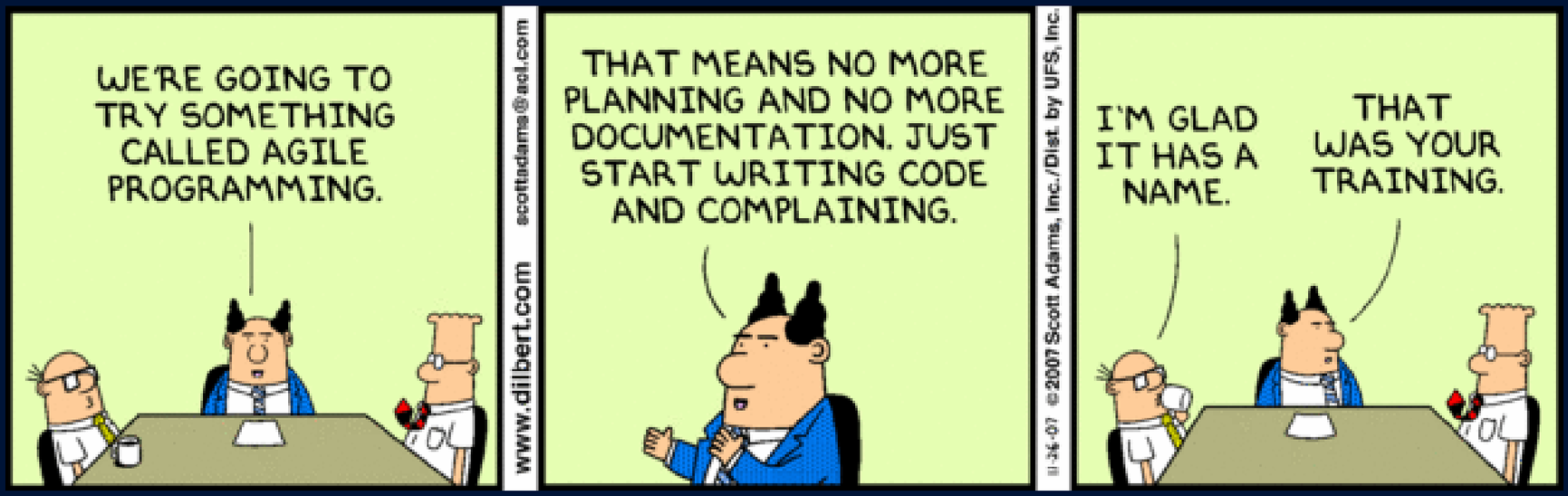
Being Agile

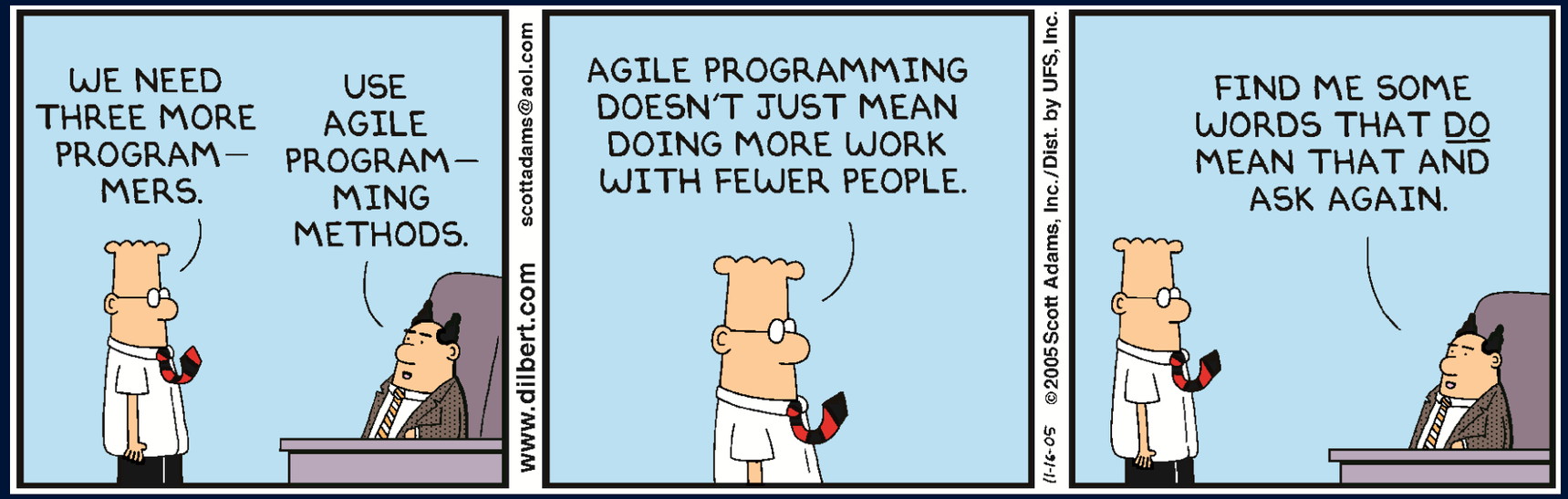
Introduction

Andrea Boni

12 November 2021

Agile


WE'RE GOING TO TRY SOMETHING CALLED AGILE PROGRAMMING.
www.dilbert.com scottadams@aol.com
THAT MEANS NO MORE PLANNING AND NO MORE DOCUMENTATION. JUST START WRITING CODE AND COMPLAINING.
11-26-07 ©2007 Scott Adams, Inc./Dist. by UFS, Inc.
I'M GLAD IT HAS A NAME.
THAT WAS YOUR TRAINING.


WE NEED THREE MORE PROGRAM— MERS.
USE AGILE PROGRAM— MING METHODS.
www.dilbert.com scottadams@aol.com
AGILE PROGRAMMING DOESN'T JUST MEAN DOING MORE WORK WITH FEWER PEOPLE.
11-16-05 ©2005 Scott Adams, Inc./Dist. by UFS, Inc.
FIND ME SOME WORDS THAT DO MEAN THAT AND ASK AGAIN.

Agile development: what they say

Agile software development describes an approach to software development under which requirements and solutions evolve through the collaborative effort of self-organizing cross-functional teams and their customer(s)/end user(s). It advocates adaptive planning, evolutionary development, early delivery, and continuous improvement, and it encourages rapid and flexible response to change.

Wikipedia

Agile?

Being "Agile" is about responding to changes effectively

Why manage changes?

Driving a car is not about pointing it in the right direction.

In theory it works, in practice you keep on making small (or big) adjustments.

Agile methodology give frameworks to be effective in reacting to changes

How to react to changes effectively?

What prevents the organization from being “Agile”?

Recurring themes in Agile methodologies

- **Quality and fight against wastes**
  Lack of quality is one of most common impediment to Agile. Be sure "to do things right"
- **Centrality of the customer & Early feedback**
  One of the most important aspect is continuous feedback from the "customer" to be sure to "do the right thing". Reduce the risk of "doing the wrong thing" so to give the chance of steering away from bad decisions
- **Ownership**
  Developers own the code and the architecture, they are responsible for it
- **Communication/transparency**
  Lack of communication and transparency prevents effective decision making
- **Continuous improvement**
  Being Agile is an empiric process based on continuously measuring performance and tune behaviors

Agile Manifesto - Values

Agile Manifesto

Individuals and interactions	over	Processes and Tools
Working Product	over	Comprehensive Documentation
Customer Collaboration	over	Contract Negotiation
Responding to change	over	Following a plan

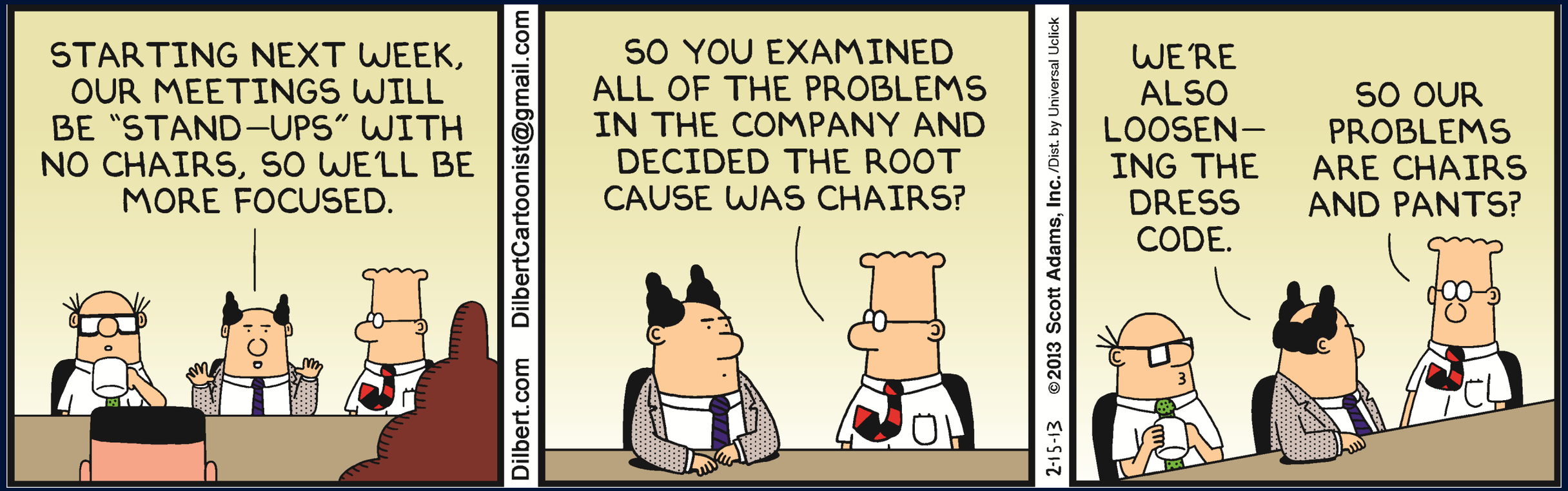
That is, while there is value in the items on the right, we value the items on the left more.

Agile Manifesto - Principles

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity—the art of maximizing the amount of work not done—is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly

Agile is not about rituals

- Objective is agility
- "Methodologies" provides frameworks, not recipes
- Adaptation is a key element, and it is an empirical iterative process
- Rituals are not the goal, they are a mean



General Concepts

What you need to understand before we dive into Agile

General concepts

The problem

- Cost of changes
- Cost of defects & Technical debt
- Wastes

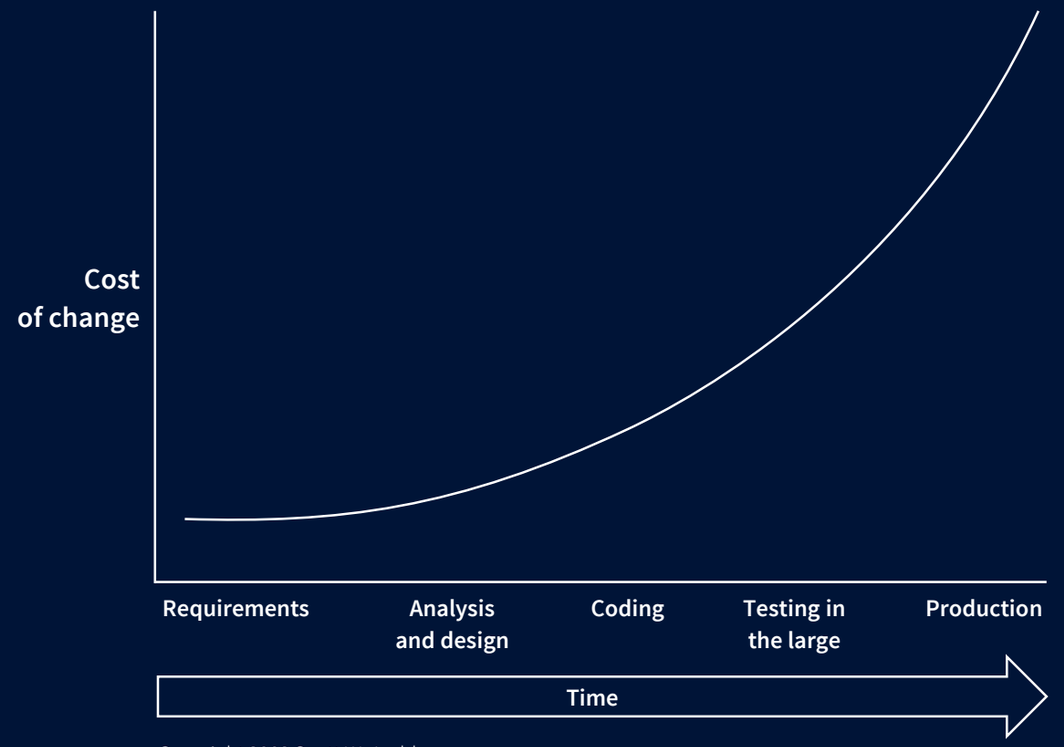
The solution

- Iterative development
- Embedded quality
- Multi-skills self-organizing team
- User-centric backlog
- Structured continuous improvement

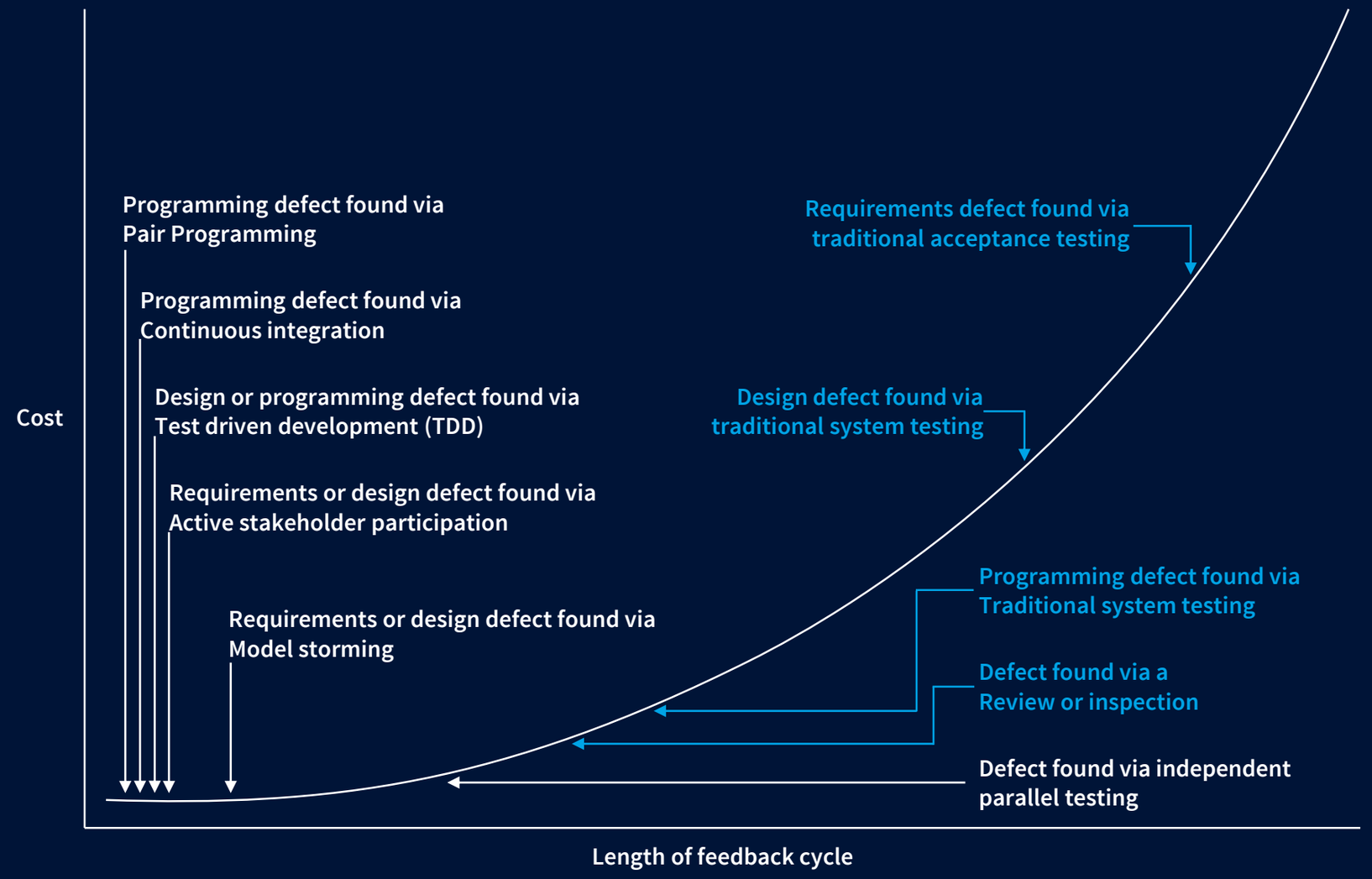
Cost of changes

Cost of introducing changes in a traditionally managed product

- Non-linear with time
- The more it takes to realize the requirement has changed the more expensive it becomes
- The most likely moment the need for changes becomes evident is when the product is DELIVERED
- To the extreme, project failure is likely to happen when the entire cost of the project is spent



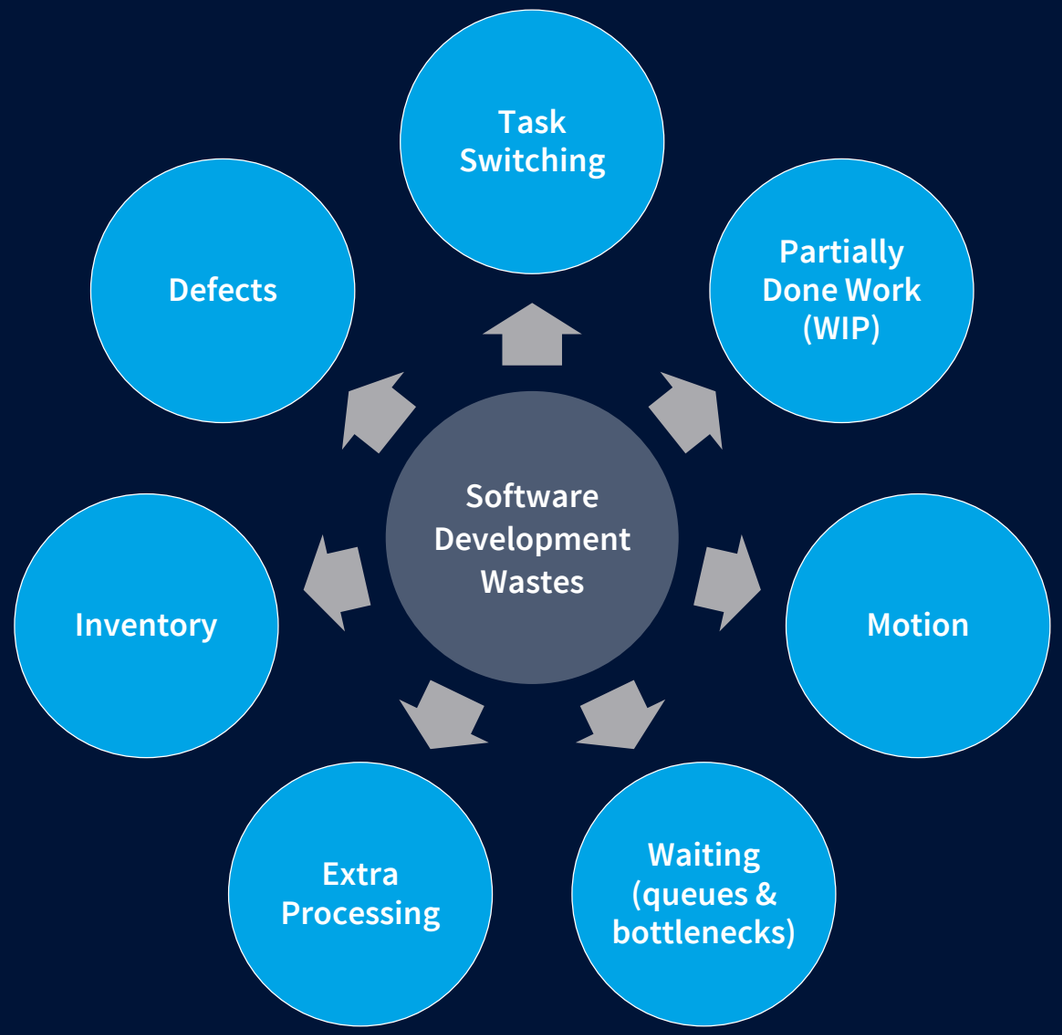
Cost of defects



Wastes

Waste is anything that does not add value to the product.

In "lean" terms, anything that does not belong to the Value Chain



Wastes

Why wastes are bad:

- Reduce speed (or increase drag)
  - They are an impediment to fast feedback loop
  - Disperse resources increasing costs (usually non-linearly)
- Reduce clarity on value chain
  - They are an impediment to effective decision making
  - Compromise ROI
  - Disperse resources increasing costs (usually non-linearly)
  - Promote local optimization (as opposed to system optimization, that is good)
- Create information, cultural or organizational barriers
  - They are an impediment to effective decision making
  - They are an impediment to continuous improvement

Iterative development

How to implement effective feedback loop

Iterative evolutionary development

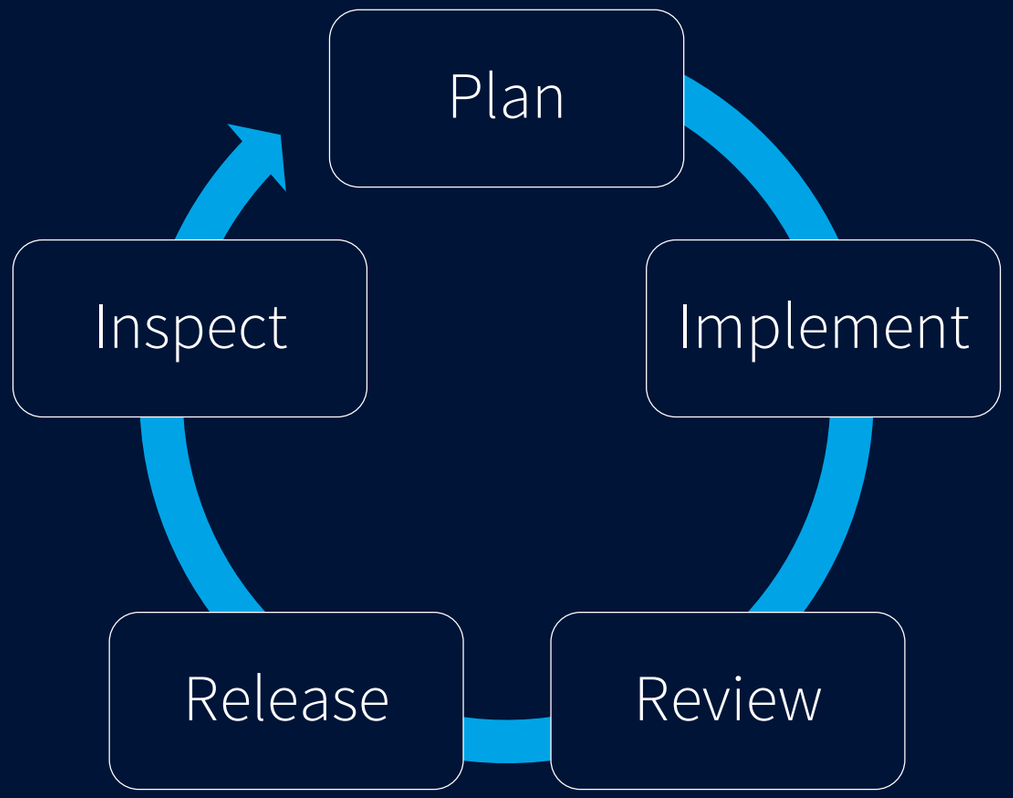
Short development cycles (sprints)

Cycles includes everything it takes to release the software (planning, analysis, design, tests, documentation etc.)

The customer (or somebody else on his behalf) uses the software (accept)

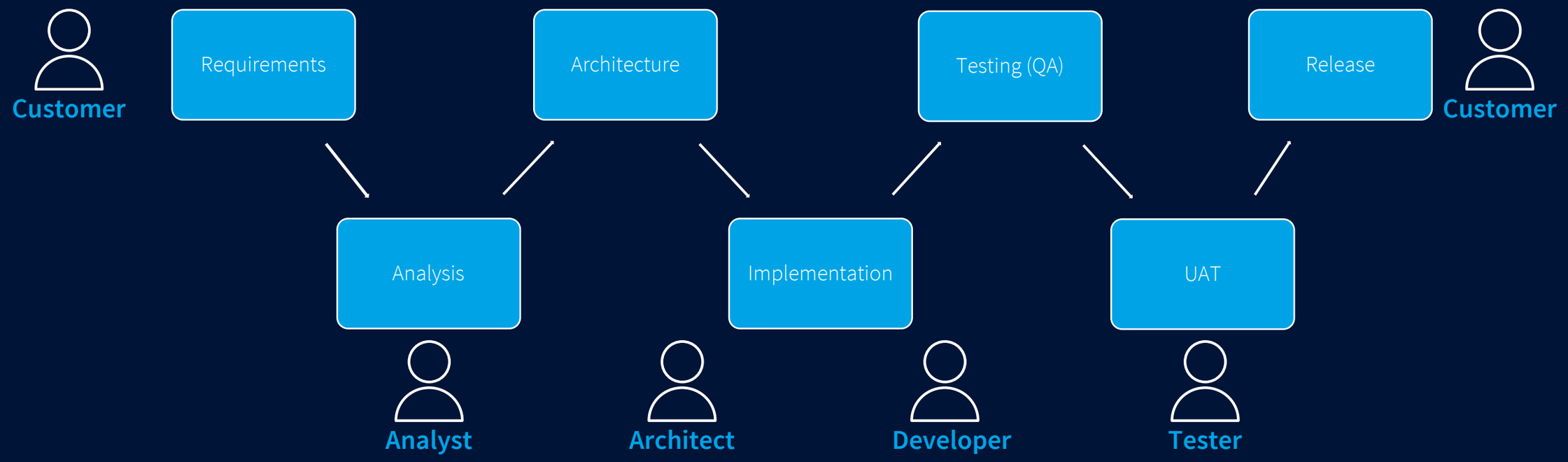
Each increment delivers something that the user can use (potentially shippable software)

Each sprint delivers what is most important to the customer



As opposed to waterfall

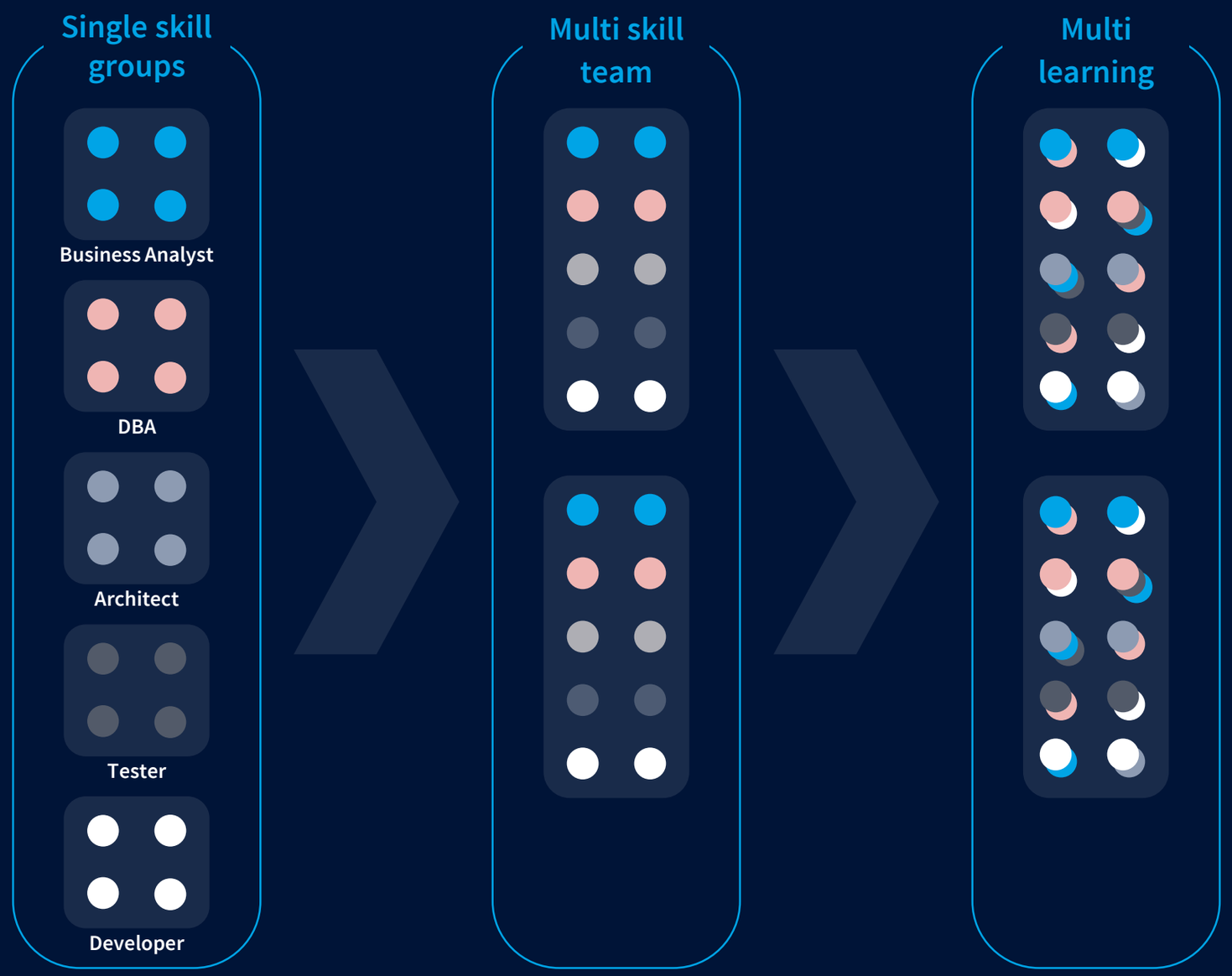
- Dilution of knowledge along the line
- If something changes the organization reacts very late
- Developers are the farthest from the customers



Multi-skills, self organizing teams

How iterative development can possibly work?

The journey


Single skill groups
Business Analyst
DBA
Architect
Tester
Developer
Multi skill team
Multi learning

Multi-skills empowered team

Multi-skills:

- The team incorporates all the necessary skills to get the job done (release working quality software)
- The team: designs the software, document the software, tests the software, maintain the environment and in some cases runs the system

Why the team must be multi-skilled?

To avoid wastes:

- Handover, bottlenecks, long feedback loop, reduced decision making, …
- Facilitate continuous improvement

Because otherwise other activities would have to be carried out by others and handover would be necessary (waste). Handover is bad because it creates bottlenecks, knowledge transfer cost (and dilution) and eventually delays.

Because if the team owns the entire development process it can think in terms of system optimization instead of local optimizations. Remember: continuous improvement!

Multi-skills empowered team

Empowered:

- Accountable: the negotiate features, commit to execute and is ultimately responsible for the result
- Decisions: the team decides on technical aspects (design, implementation , tools, technology, …)
- Self-organizing: the team distribute tasks and organize the internal structure (continuous improvement is also a team prerogative)

Why the team must be empowered?

Reasons are many, some of them come from Theory X (the opposite of Theory Y)…
but the intuition with empowered team is that they are in the best position to take the decisions concerning technology, architecture, design, micro-planning, etc.

What does it take for a team to be empowered?

Enough information to make decisions, understand the domain (see user-centric backlog), technically excellence (see multi skills, continuous improvement & learning)

Embed quality

Zero-bugs policy and obsession all quality aspects

Different forms of technical debt

Technical Debt is the enemy and must be eliminated as soon as it is identified.

Or at least acknowledged and managed…

- Defects
- Lack of test suites
- Manual testing
- Excessive length of process
- Knowledge bottlenecks
- Inadequate tools
- Obsolete technology

The technical debt

Always write code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.

John F. Woods

If you wonder, he is a game programmer and he wrote this in a blog in the ninthies

Embed Quality in the process

Quality Assurance: the old way -> Quality is “assured” by handling the product over to the testers once it’s ready.

This creates delays and context switch within the development organization (all the bad things: handover, need to pass information over, queues & bottlenecks, delayed feedback, …)

“Embed quality” means that the process is engineered so to produce a zero-defects products at the end of each sprints.

Development practices:

- Everything must be **automated** because manual operations are a waste (it takes time and it is bad use of human time)
- ATDD – It’s necessary to validate the software match the user stories (see “Specification By Examples”)
- TDD – it’s necessary for a lot reasons
- Coding Standards
- Code Review (this is controversial – some authors think that this is post processing)
- Pair programming

User centric backlog

The “user stories”

User stories

From Wikipedia: “A user story is a tool used in Agile software development to capture a description of a **software feature from an end-user perspective**. The user story describes the type of user, what they want and why. A user story helps to create a simplified description of a requirement.”

A backlog is a list of “user stories” that the Product Owner (he/she represents the user) maintained always ordered by priority. The priority is given by the importance to the user.

User Stories are used by the team and the product owner **to facilitate the transfer of knowledge** and to be sure to **capture what the user wants.**

What a story looks like

A Story describes a feature from the perspective of the user.

To write a story we need to know:

- Who the user is
- What he/she has (pre-conditions)
- What he/she does
- What he/she expect to obtain
- Why this brings value to the user

As a {who},
I want {what}
Because {why}

Ex: As a home banking user (the user) I want the system to validate my identity (what) by asking my password again (expectation) before sending an order to market so I do not risk my session to be hijacked (why).
Pre-condition is to have a valid session open and to be in the final stage of sending an order.

Remember – stories are USER CENTRIC!

Creating a password

cabbage
Sorry, the password must be more than 8 characters.
boiled cabbage
Sorry, the password must contain 1 numerical character.
1 boiled cabbage
Sorry, the password cannot have blank spaces.
50[redacted]boiledcabbages
Sorry, the password must contain at least one upper case character.
50[redacted]Gboiledcabbages
Sorry, the password cannot use more than one upper case character consecutively.
50[redacted]gBoiledCabbagesShovedUpYourArse,IfYouDon'tGiveMeAccessImmediately
Sorry, the password cannot contain punctuation.
NowIAmGettingReallyPissedOff50[redacted]gBoiledCabbagesShovedUpYourArseIfYouDontGiveMeAccessImmediately

Sorry, that password is already in use!

User Stories - performance

PO: I want it responsive…

Team: Yes - but how responsive?

PO: When I digit 99 I want to see 4.12 almost instantaneously

Team: Ok - I'll show you a mock with 0.1s and a 0.01s

PO: The feeling is the same - I guess 0.1s is ok

Team: ok - the story then is "when I type the price, I want the yield to be calculated and shown in less than 0.1s

PO: The story is about importing trades from electronic markets

Team: How many? What's the expected load?

…

All: Ok, we all agree that 500 trades per second is ok and occasional bursts in the thousands can be absorbed within 10~20 seconds

…

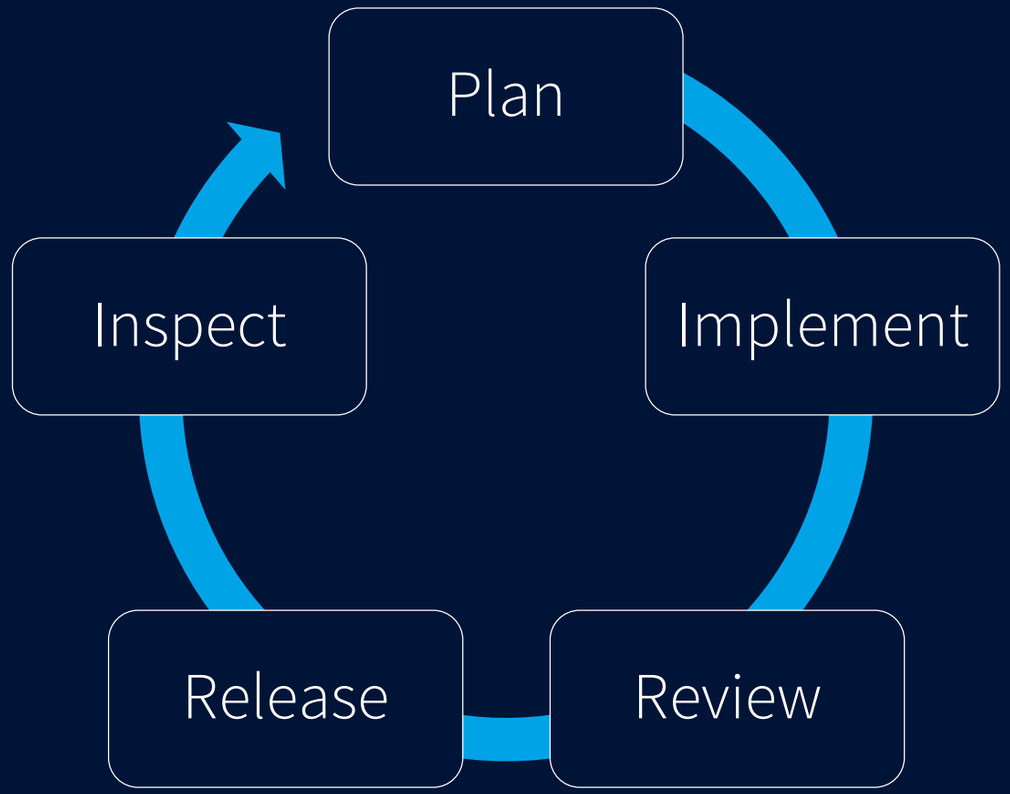
PO: I've found that our best competitors can do 250 trades per second. If we can really do 500 we will have strong a selling point.

Continuous improvement

A scientific approach

Improvement as part of the process

SCRUM introduces the event of Team Retrospective



Retrospective is about continuous improvement:

It focuses on:

- Identify wastes
- Elaborate solutions
- Elaborate metrics to evaluate the effectiveness of solutions

The what happens:

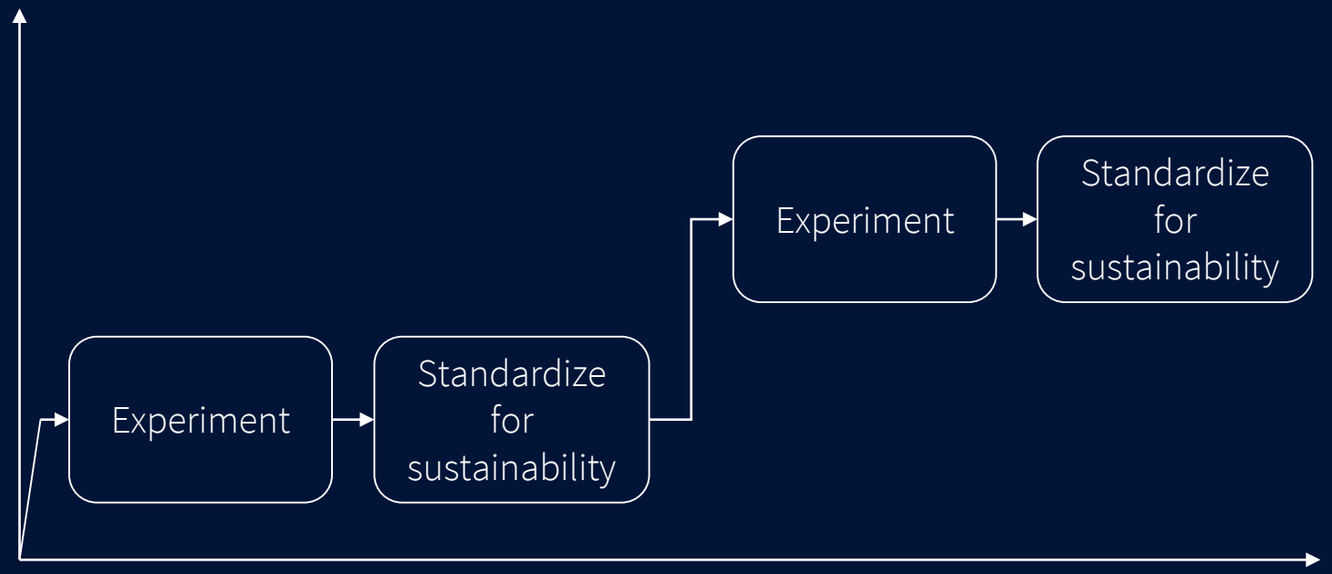
- Implement
- Inspect
- Decide if it works and in case do it again

Improvement as part of the process (Kaizen)

In Lean (see The Toyota Way) continuous improvement is at the base of the company culture.

It fosters:

- Multi-learning
- Technology evolution
- Total quality



Scrum

shortly

Ken Schwaber

“Scrum is not an acronym. It’s an event in the game of rugby where like-minded people get together and politely discuss ownership of a ball.”

“Scrum works with idiots! You can take a group of idiots and uniformly they will produce crap every increment.”

My favorite authors

Kent Beck – xTreme Programming: the toolkit of the agile developer

Martin Fowler – less talking and more design (and the wheel was invented before you were born)

Bob Martin – he is one of the founders of the “Craftmanship manifesto”

Ken Schwaber – he made money with Scrum

Craig Larman & Bas Vode – Large Scale Scrum