Agile Project management

» 22-23-24 April-2017. Boughdiri Aymen

- Aymen is an Agile Coach, Software Quality & Testing Consultant / Mentor with more than 4 years of extensive experience in managing development, quality and testing teams in Banking, Social Networking domains with extensive experience in testing web, desktop and mobile apps.

- Aymen has over 8 years of experience in the IT industry.

- He provides training, consulting, coaching and mentoring services in software development and testing, especially in Agile methodologies, Functional Testing, Usability Testing, Requirements-Based Testing, Risk-Based Testing, Test Management, Test Design, and Software Quality Improvements.

- His areas of expertise include Practicing Agile Methodologies, Creating Software Testing Frameworks, Test Process Models Establishment and Improvement, Developing Test Policy and Test Strategies, Risk Based Testing, Test Analysis & Design, Defects Life Cycle Management.

- Aymen holds ISTQB Foundation, Agile Tester, Test Manager, Test Analyst. And holds a Bachelor on Finance- Master degree on Finance engineering from High Institute of management on Tunis.

ScrumAlliance®
Transforming the World of Work

Aymen Boughdiri

is awarded the designation Certified ScrumMaster® on this day, February 23, 2015, for completing the prescribed requirements for this certification and is hereby entitled to all privileges and benefits offered by SCRUM ALLIANCE®.

Member: 000394856 Certification Expires: 23 February 2017

Michel Goldenberg
Certified Scrum Trainer®

Harvey Wheaton
Chairman of the Board

Agenda

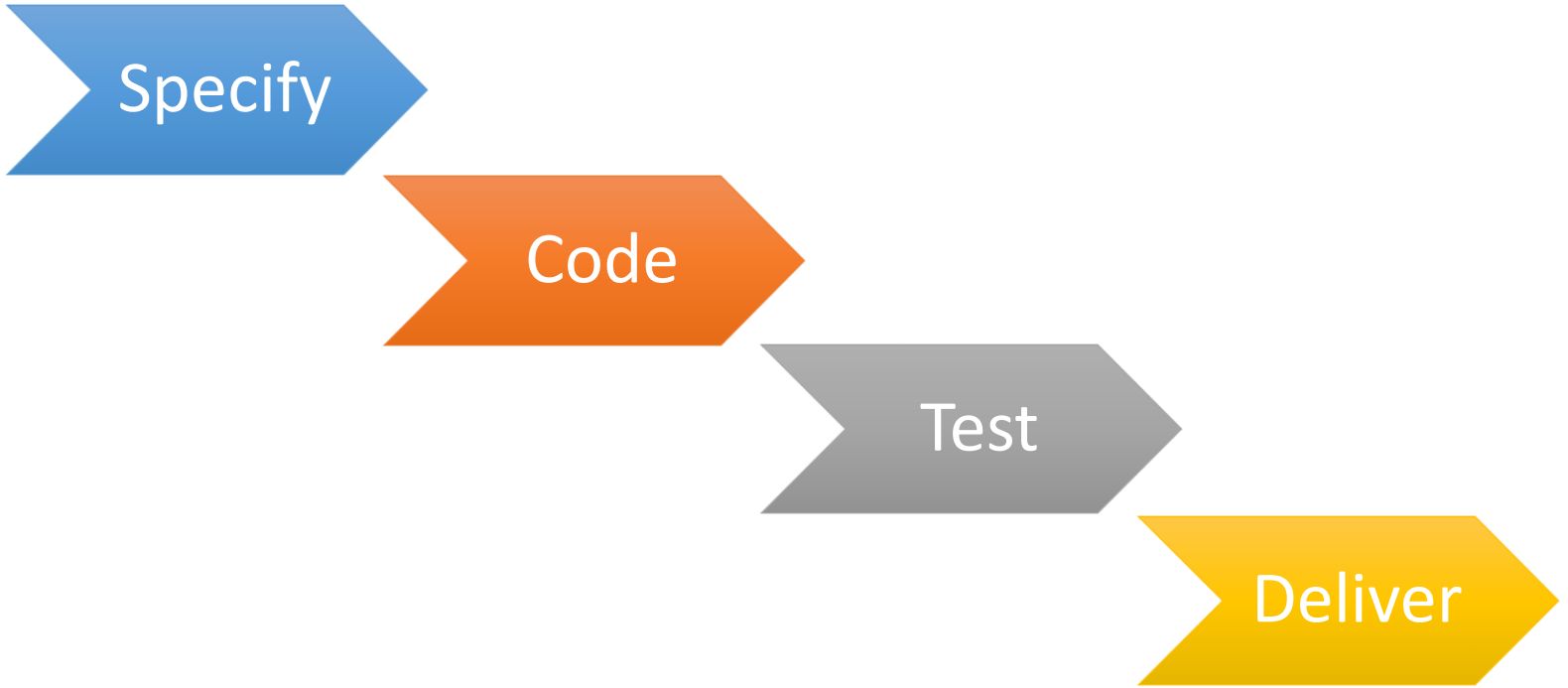
- History and Background
- Lean Principles
- Agile Manifesto
- Scrum
- Excercise on Scrum
- Summary

History

Ford Motors and industrialization

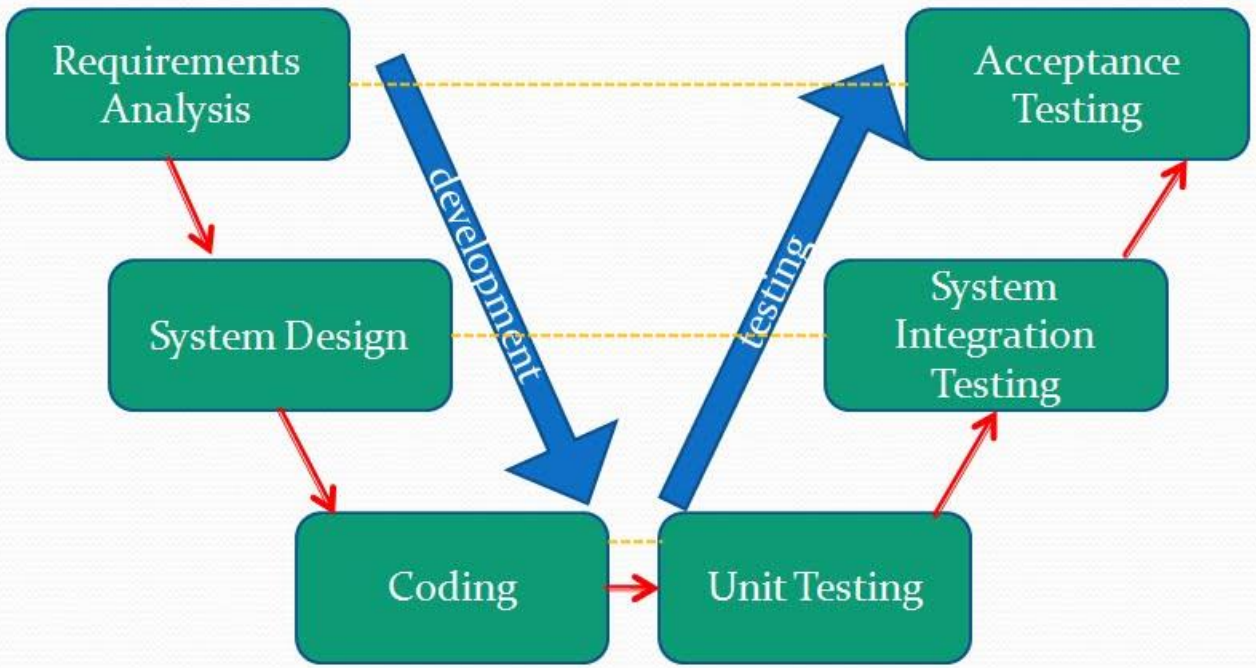
Waterfall

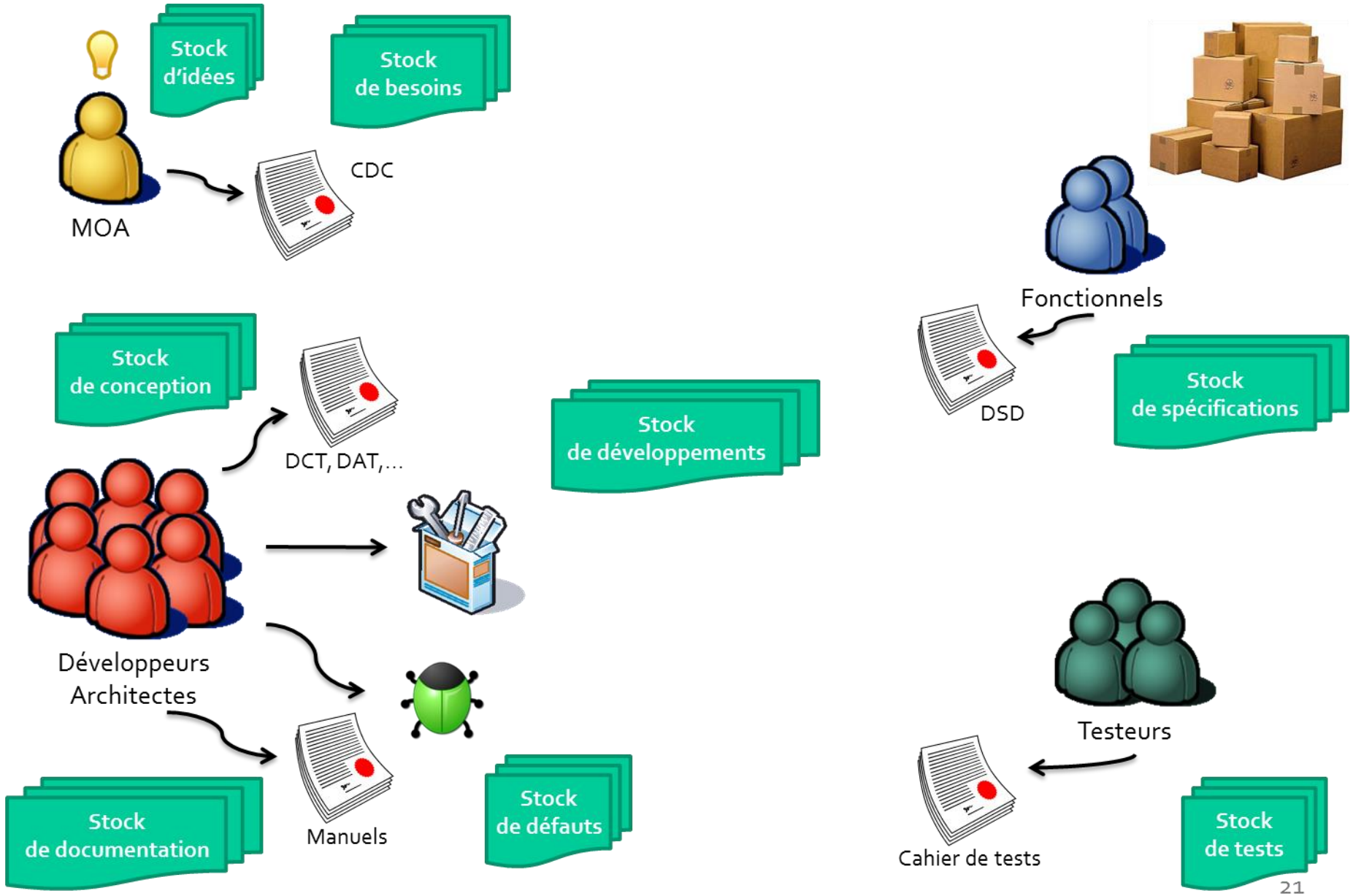
Cycle de development en cascade

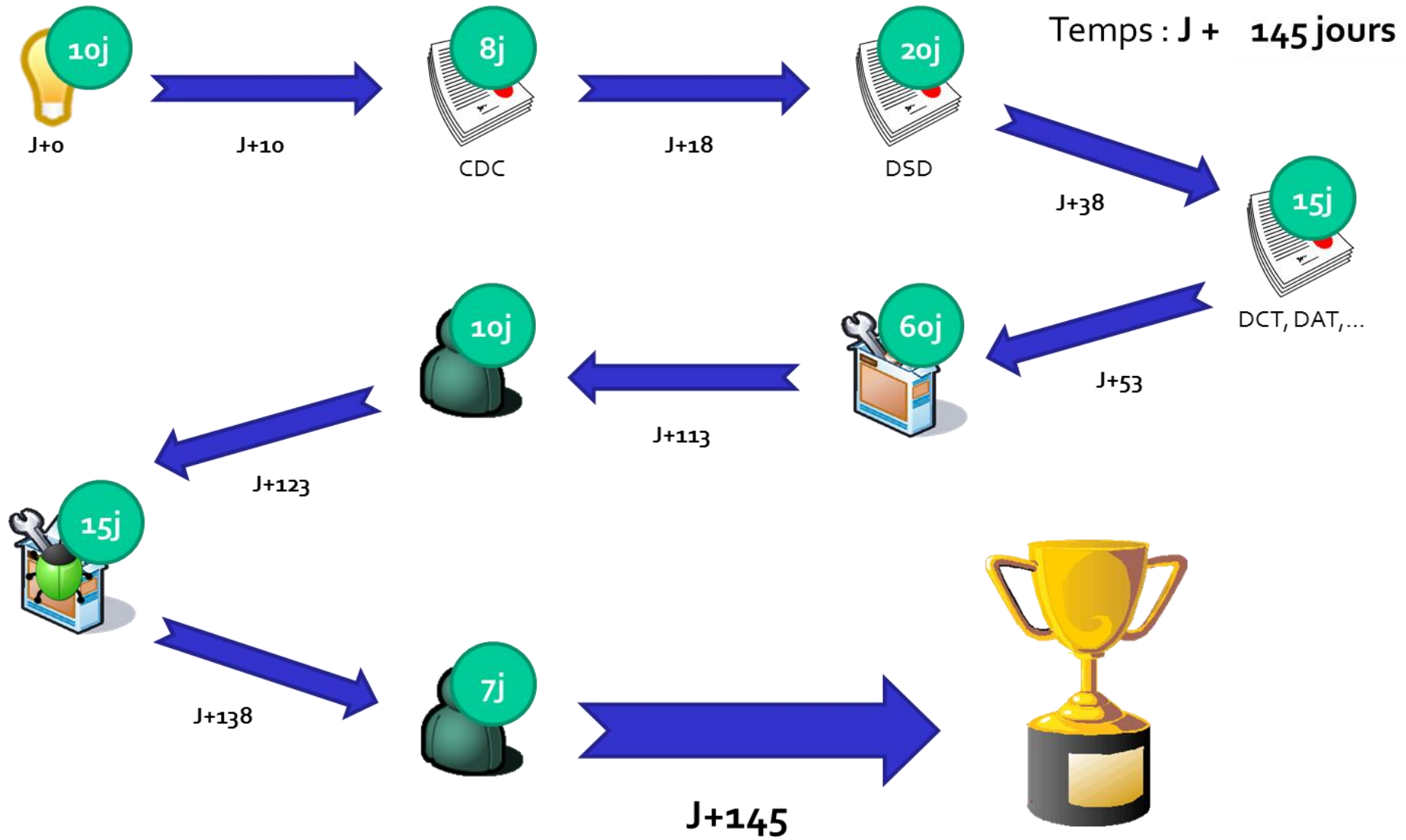


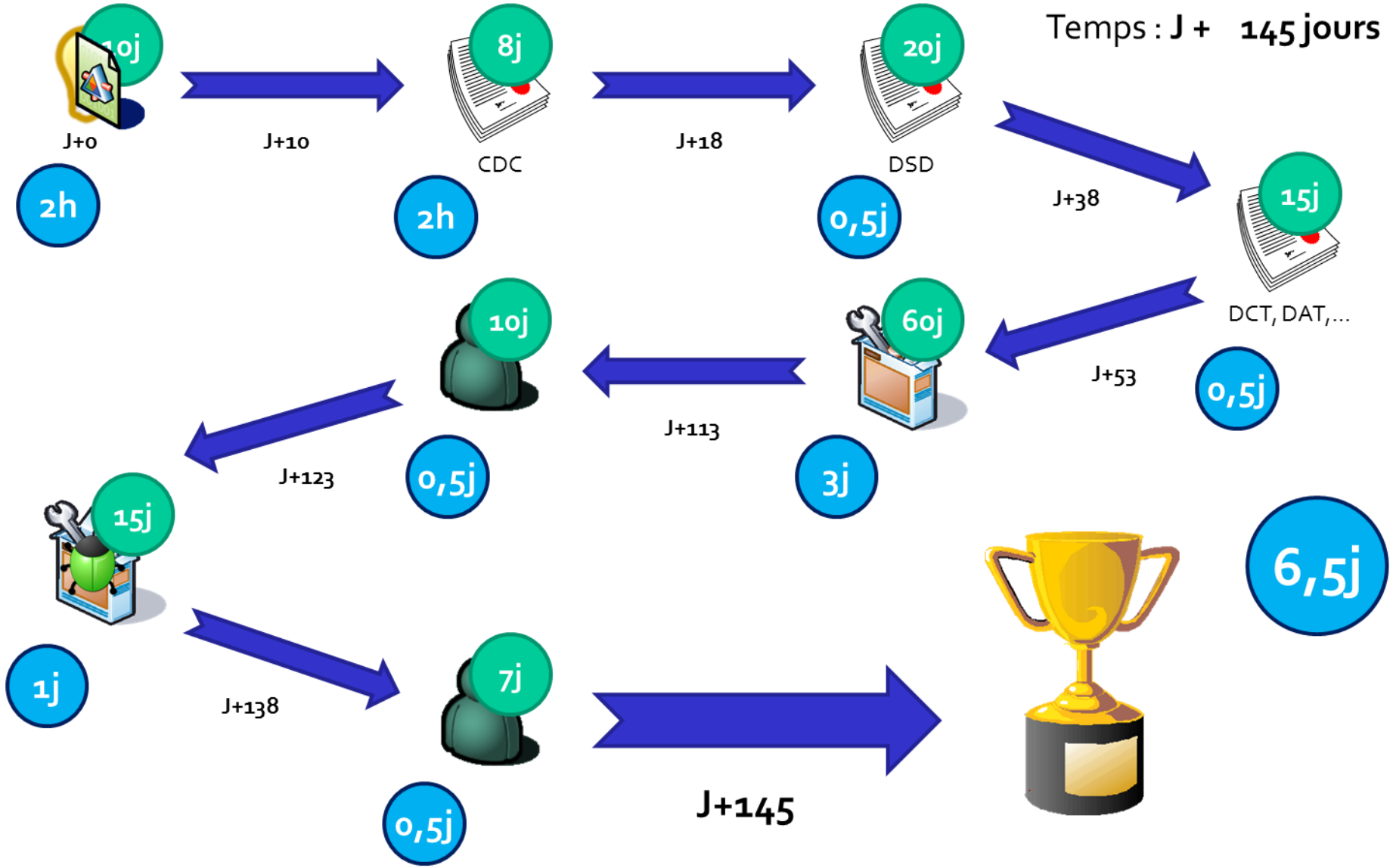
The Software Development Process

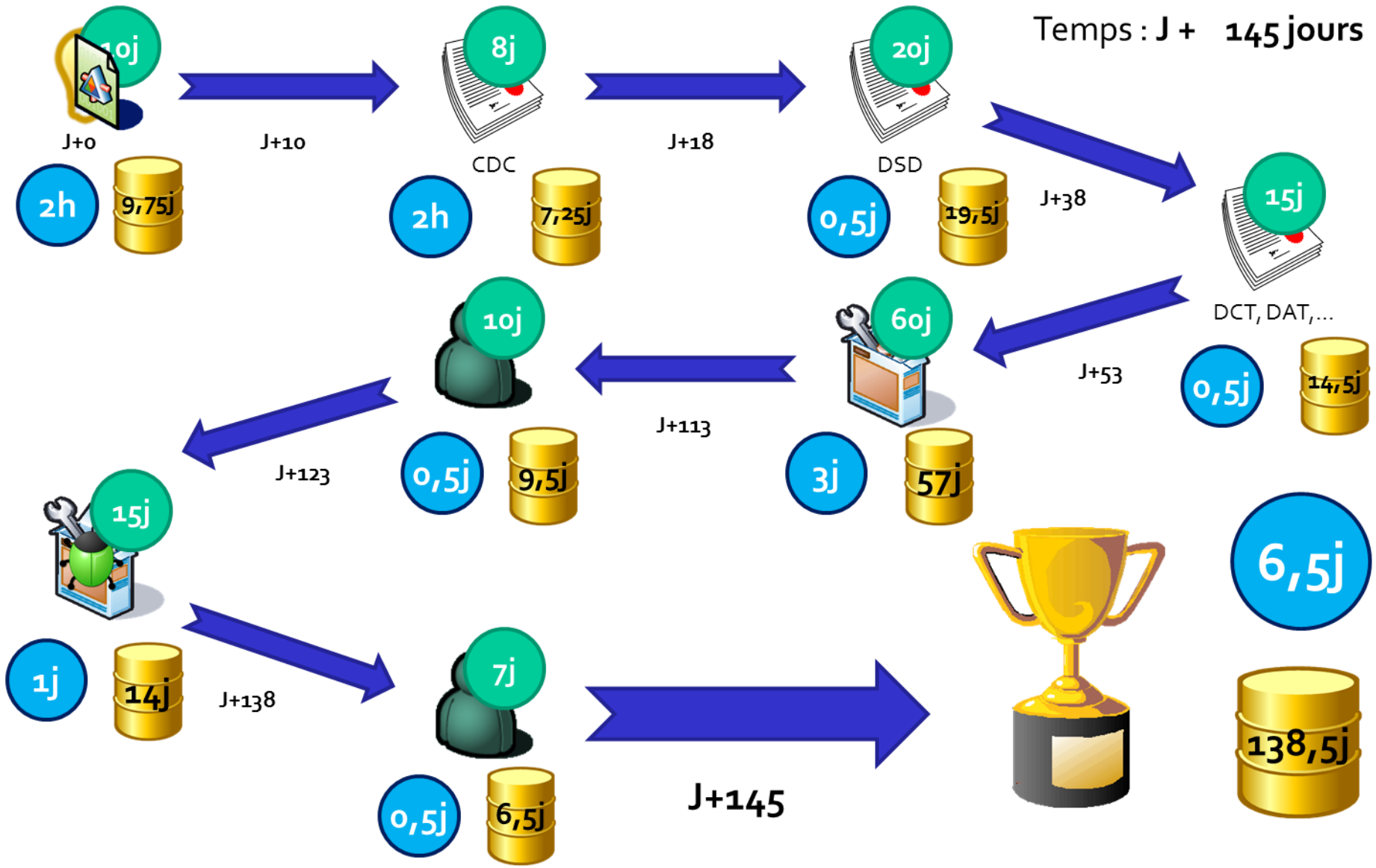
V Model


Requirements Analysis
System Design
Coding
Unit Testing
System Integration Testing
Acceptance Testing
development
testing

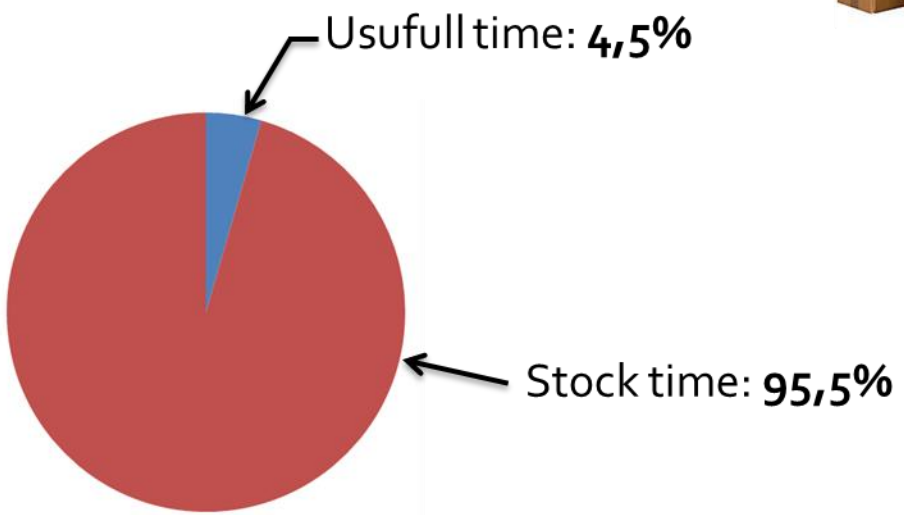

Stock d'idées
Stock de besoins
MOA
CDC
Fonctionnels
DSD
Stock de spécifications
Stock de conception
DCT, DAT,…
Stock de développements
Développeurs Architectes
Testeurs
Cahier de tests
Stock de tests
Stock de documentation
Manuels
Stock de défauts


Temps : J + 145 jours
10j
J+0
J+10
8j
CDC
J+18
20j
DSD
J+38
15j
DCT, DAT,...
J+53
6oj
J+113
10j
J+123
15j
J+138
7j
J+145


Temps : J + 145 jours
10j
J+0
2h
J+10
8j
CDC
2h
J+18
20j
DSD
0,5j
J+38
15j
DCT, DAT,...
0,5j
J+53
60j
3j
J+113
10j
0,5j
J+123
15j
1j
J+138
7j
0,5j
J+145
6,5j

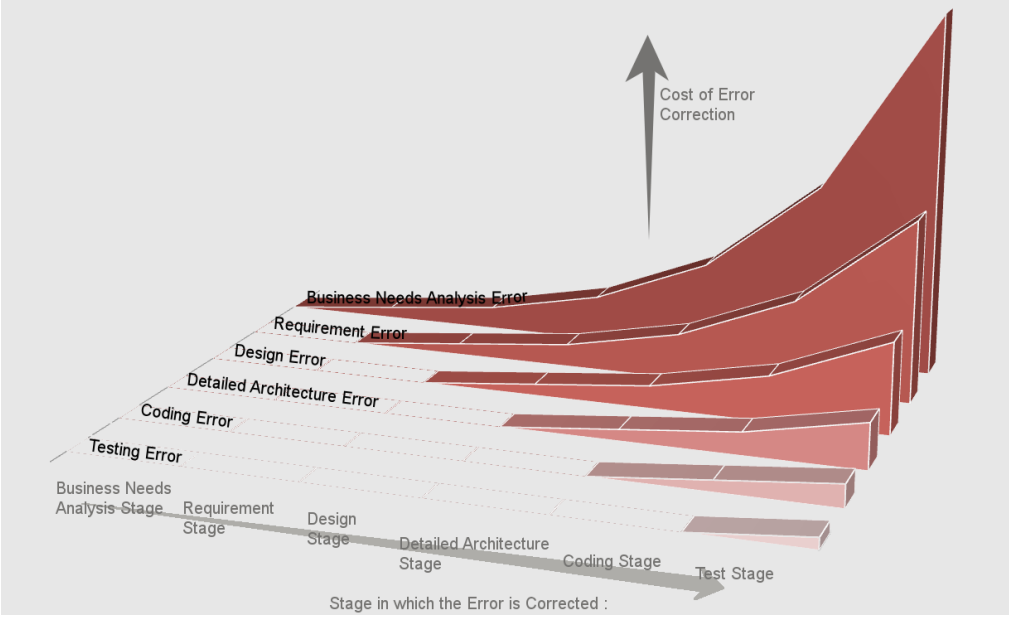

Temps : J + 145 jours
10j
J+0
2h
9,75j
J+10
8j
CDC
2h
7,25j
J+18
20j
DSD
0,5j
19,5j
J+38
15j
DCT, DAT,...
0,5j
14,5j
J+53
60j
3j
57j
J+113
10j
0,5j
9,5j
J+123
15j
1j
14j
J+138
7j
0,5j
6,5j
J+145
6,5j
138,5j

The stock time is very huge



- Time to Market >= 145 j
- The added value of the developement decrease

- The issues are dissimulated on the stock
- The issue rae discovered always at late phase



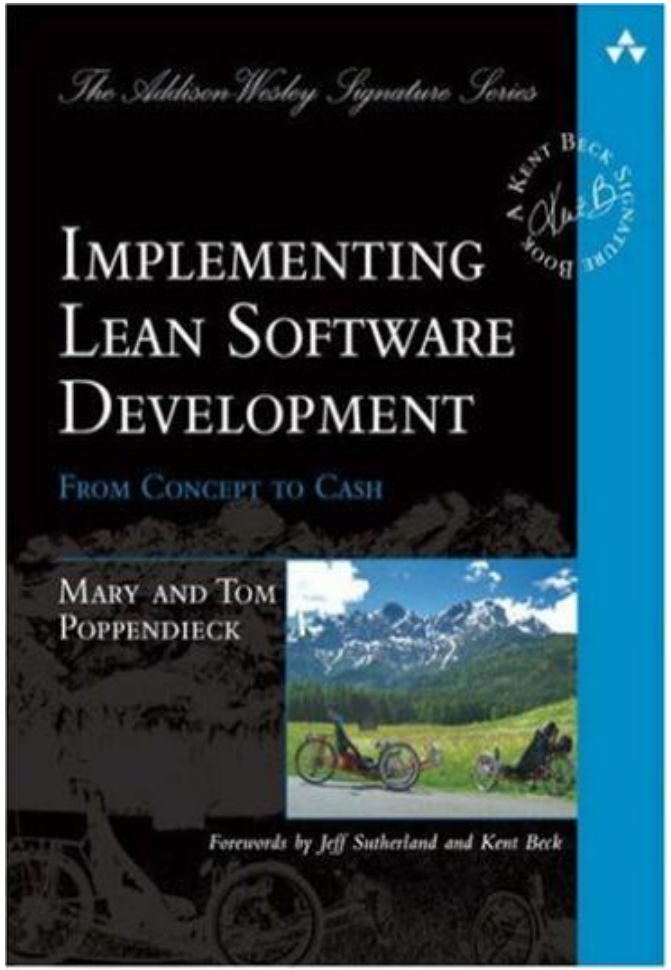
History

Toyota Carmaking

Success factors of Toyota

- Stop the line
- Five Why's analysis
- Errors could not propagate along the production line
- Just-in-Time
- Respect of the workers
- Time-to-Market

Lean Principles and Practices



Agenda

History and Background

Lean Principles

Agile Manifesto

Scrum

Excercise on Scrum

Summary

Lean Software Principles

1. Eliminate waste
2. Build integrity in
3. Amplify learning / share knowledge
4. Decide as late as possible
5. Deliver fast
6. Empower the team
7. See the whole / improve the system

Eliminate waste

- Lean philosophy regards everything not adding value to the customer as waste (muda). Such waste may include:
    - **Partially done work**: .Partially done coding eventually abandoned during the development process is waste.
    - **Extra processes** : If some activity could be bypassed or the result could be achieved without it, it is waste Extra processes like paperwork …
    - **Extra features** : features not often used by customers are waste
    - **Task switching:** Switching people between tasks is waste
    - **Waiting**: Waiting for other activities, teams, processes is waste
    - **Motion**: Motion required to complete work is waste
    - **Defects**: .. Defects and lower quality are waste
    - **Management activities:** Managerial overhead not producing real value is waste.
- In order to eliminate waste, one should be able to recognize it.
- A value stream mapping technique is used to identify waste. The second step is to point out sources of waste and to eliminate them. Waste-removal should take place iteratively until even seemingly essential processes and procedures are liquidated.

Build integrity in

- The customer needs to have an overall experience of the System. This is the so-called perceived integrity: how it is being advertised, delivered, deployed, accessed, how intuitive its use is, its price and how well it solves problems.

- Conceptual integrity means that the system's separate components work well together as a whole with balance between flexibility, maintainability, efficiency, and responsiveness. This could be achieved by understanding the problem domain and solving it at the same time, not sequentially. The needed information is received in small batch pieces – not in one vast chunk - preferably by face-to-face communication and not any written documentation. The information flow should be constant in both directions – from customer to developers and back, thus avoiding the large stressful amount of information after long development in isolation.

Amplify learning / share knowledge

- Software development is a continuous learning process based on iterations when writing code. Software design is a problem solving process involving the developers writing the code and what they have learned. Software value is measured in fitness for use and not in conformance to requirements.

- Instead of adding more documentation or detailed planning, different ideas could be tried by writing code and building. The process of user requirements gathering could be simplified by presenting screens to the end-users and getting their input. The accumulation of defects should be prevented by running tests as soon as the code is written.

- The learning process is sped up by usage of short iteration cycles – each one coupled with refactoring and integration testing. Increasing feedback via short feedback sessions with customers helps when determining the current phase of development and adjusting efforts for future improvements. During those short sessions both customer representatives and the development team learn more about the domain problem and figure out possible solutions for further development. Thus the customers better understand their needs, based on the existing result of development efforts, and the developers learn how to better satisfy those needs. Another idea in the communication and learning process with a customer is set-based development – this concentrates on communicating the constraints of the future solution and not the possible solutions, thus promoting the birth of the solution via dialogue with the customer.

Decide as late as possible

- As software development is always associated with some uncertainty, better results should be achieved with an options-based approach, delaying decisions as much as possible until they can be made based on facts and not on uncertain assumptions and predictions. The more complex a system is, the more capacity for change should be built into it, thus enabling the delay of important and crucial commitments. The iterative approach promotes this principle – the ability to adapt to changes and correct mistakes, which might be very costly if discovered after the release of the system.

- An agile software development approach can move the building of options earlier for customers, thus delaying certain crucial decisions until customers have realized their needs better. This also allows later adaptation to changes and the prevention of costly earlier technology-bounded decisions. This does not mean that no planning should be involved – on the contrary, planning activities should be concentrated on the different options and adapting to the current situation, as well as clarifying confusing situations by establishing patterns for rapid action. Evaluating different options is effective as soon as it is realized that they are not free, but provide the needed flexibility for late decision making.

Deliver fast

- In the era of rapid technology evolution, it is not the biggest that survives, but the fastest. The sooner the end product is delivered without major defects, the sooner feedback can be received, and incorporated into the next iteration. The shorter the iterations, the better the learning and communication within the team. With speed, decisions can be delayed. Speed assures the fulfilling of the customer's present needs and not what they required yesterday. This gives them the opportunity to delay making up their minds about what they really require until they gain better knowledge. Customers value rapid delivery of a quality product.

- The just-in-time production ideology could be applied to software development, recognizing its specific requirements and environment. This is achieved by presenting the needed result and letting the team organize itself and divide the tasks for accomplishing the needed result for a specific iteration. At the beginning, the customer provides the needed input. This could be simply presented in small cards or stories – the developers estimate the time needed for the implementation of each card. Thus the work organization changes into self-pulling system – each morning during a stand-up meeting, each member of the team reviews what has been done yesterday, what is to be done today and tomorrow, and prompts for any inputs needed from colleagues or the customer. This requires transparency of the process, which is also beneficial for team communication. Another key idea in Toyota's Product Development System is set-based design. If a new brake system is needed for a car, for example, three teams may design solutions to the same problem. Each team learns about the problem space and designs a potential solution. As a solution is deemed unreasonable, it is cut. At the end of a period, the surviving designs are compared and one is chosen, perhaps with some modifications based on learning from the others - a great example of deferring commitment until the last possible moment. Software decisions could also benefit from this practice to minimize the risk brought on by big up-front design

Empower the team

- There has been a traditional belief in most businesses about the decision-making in the organization – the managers tell the workers how to do their own job. In a "Work-Out technique", the roles are turned – the managers are taught how to listen to the developers, so they can explain better what actions might be taken, as well as provide suggestions for improvements. The lean approach follows the Agile Principle[6] "find good people and let them do their own job,"[7] encouraging progress, catching errors, and removing impediments, but not micro-managing.

- Another mistaken belief has been the consideration of people as resources. People might be resources from the point of view of a statistical data sheet, but in software development, as well as any organizational business, people do need something more than just the list of tasks and the assurance that they will not be disturbed during the completion of the tasks. People need motivation and a higher purpose to work for – purpose within the reachable reality, with the assurance that the team might choose its own commitments. The developers should be given access to the customer; the team leader should provide support and help in difficult situations, as well as ensure that skepticism does not ruin the team’s spirit.

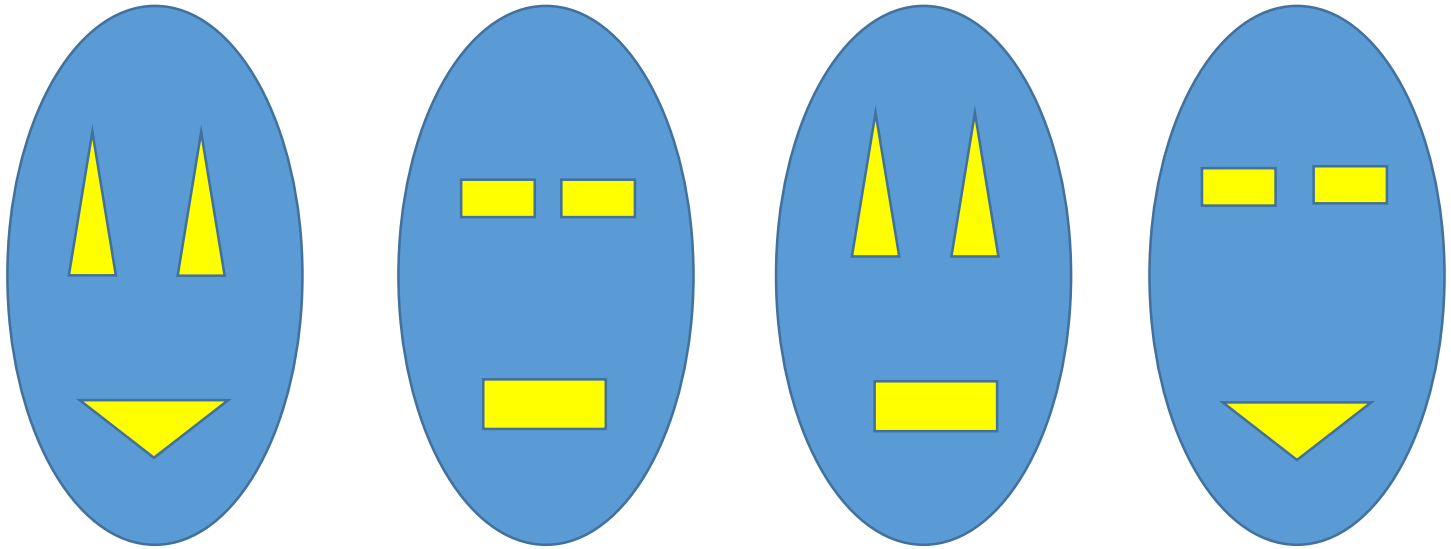
See the whole / improve the system

- Software systems nowadays are not simply the sum of their parts, but also the product of their interactions. Defects in software tend to accumulate during the development process – by decomposing the big tasks into smaller tasks, and by standardizing different stages of development, the root causes of defects should be found and eliminated. The larger the system, the more organizations that are involved in its development and the more parts are developed by different teams, the greater the importance of having well defined relationships between different vendors, in order to produce a system with smoothly interacting components. During a longer period of development, a stronger subcontractor network is far more beneficial than short-term profit optimizing, which does not enable win-win relationships.

- Lean thinking has to be understood well by all members of a project, before implementing in a concrete, real-life situation. "Think big, act small, fail fast; learn rapidly"[8] – these slogans summarize the importance of understanding the field and the suitability of implementing lean principles along the whole software development process. Only when all of the lean principles are implemented together, combined with strong "common sense" with respect to the working environment, is there a basis for success in software development.

Exercise, Mr Happy Face ☺

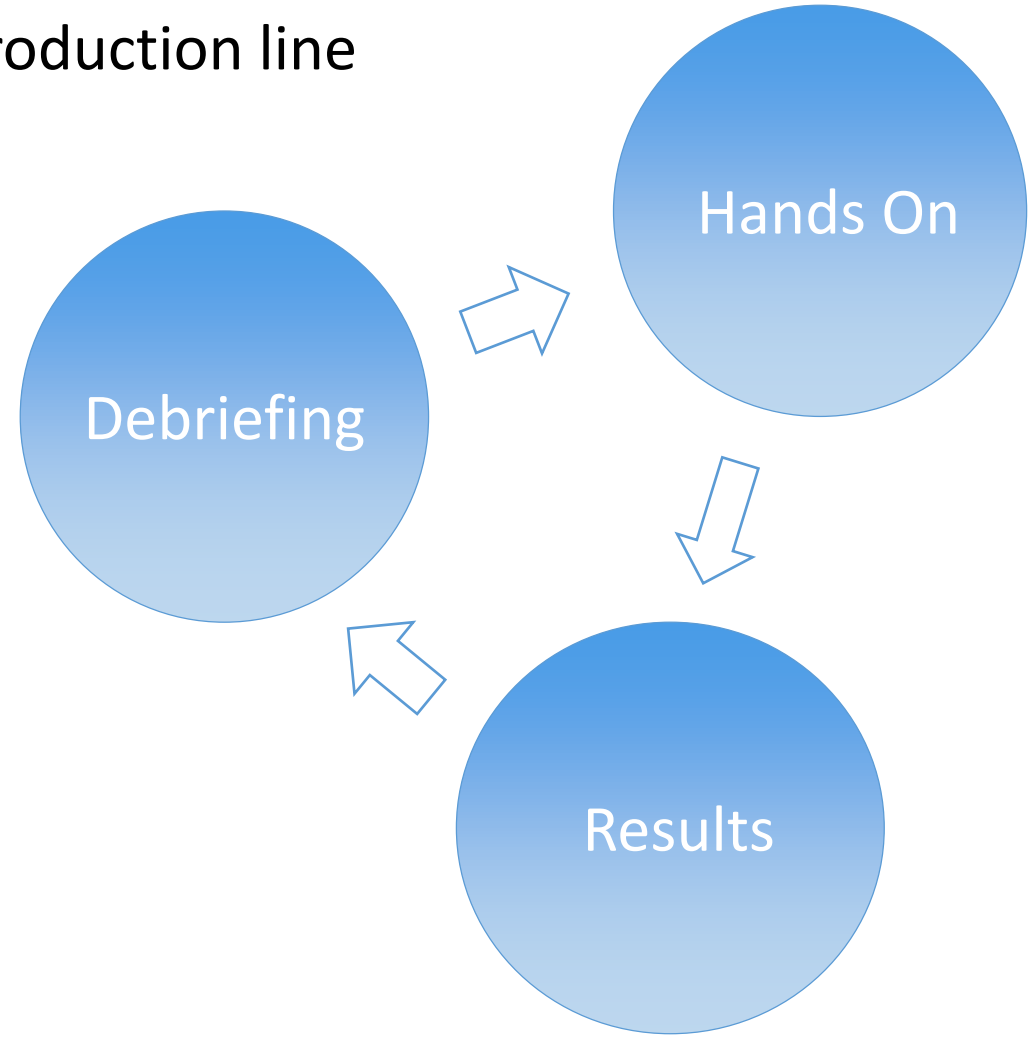
Happy Faces – The Models



Adelaide Laurent Pierre Sofie

Exercise

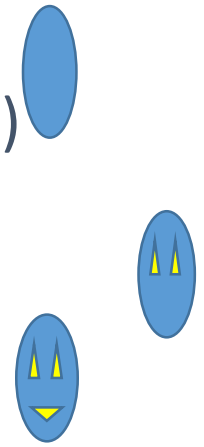
Simulating a production line



4x

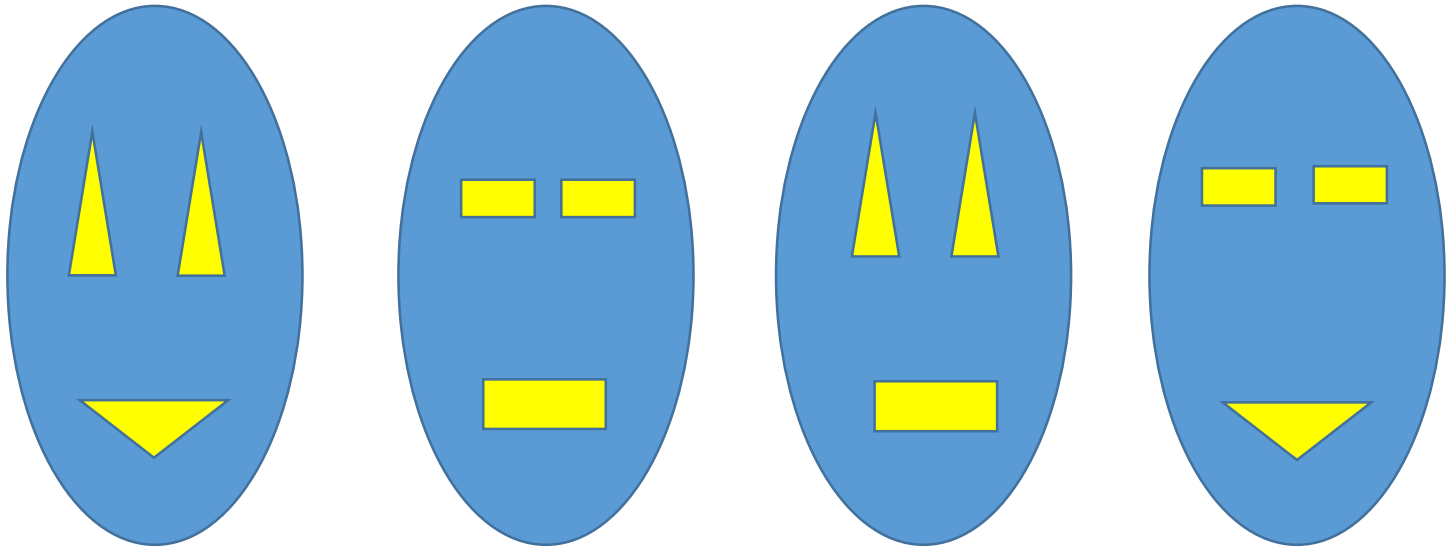
First Run – Push Process

1. Roles:
    1. Sales Manager
    2. Development team
2. A face is cut from the blue paper
3. Draw the Eyes and Mouth (design)
4. The appropriate eyes are affixed
5. The appropriate mouth is affixed
6. The face is taped to the wall
7. **Create one face before starting, to try it out**
8. *Wait for my signal* ☺



4 minutes

Happy Faces – The Models



Adelaide Laurent Pierre Sofie

Calculate profit and loss

Every sold face = +$400
Every unsold complete face = -$200
Every unsold eye = -$25
Every unsold mouth = -$50
Every uncompleted face = -$100

Retrospective

What good things did we do?
What could have been better?
What can we improve?

Waste

Visible Inventory
Production

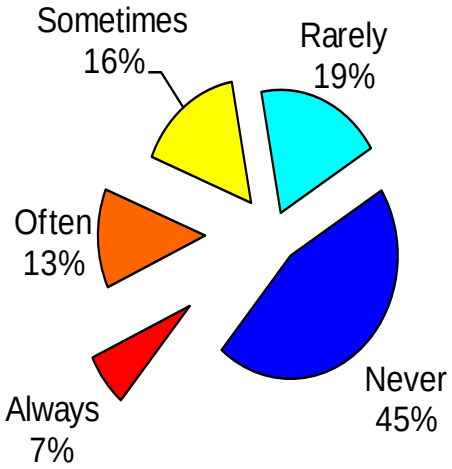
Over/Under

1. Eliminate waste

Remove all nonvalue-added activities

Example: Developing more than can be tested

Features and Functions Used in a Typical System



Standish Group Study Reported at XP2002 by Jim Johnson, Chairman

7 wastes in software development

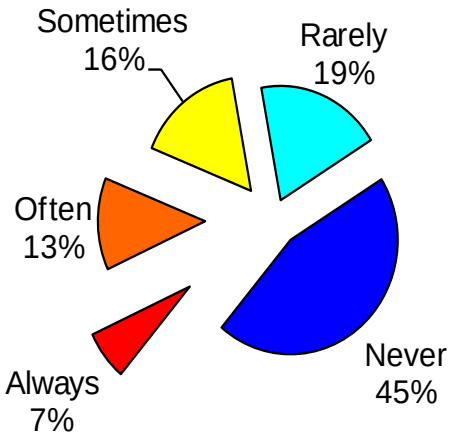
Overproduction	Extra features	Develop for today
Waiting	Waiting for decisions	Small increments
Transportation	Handoffs	Include customers
Overprocessing	Extra steps (specifications)	Code from stories
Inventory	Work in progress Unfinished work	Detail just in time
Motion (people)	Finding information	All in same room
Defects	Bugs	Test early

Myth: Early specification reduces waste

Do not force customer to make a list of what they need,

they will create a list with everything, necessary or not!

Features and Functions Used in a Typical System



Standish Group Study Reported at XP2002 by Jim Johnson, Chairman

2. Build quality in

Myth:

Get the product perfect in the first release

- Release in iterations
- Release the minimum scope first
- Refactor to increase quality
- Refactor to evolve architecture
- Use continuous integration

The goal: Lasting value

Myth:
The aim of testing is to find defects

The aim of testing is to prevent defects

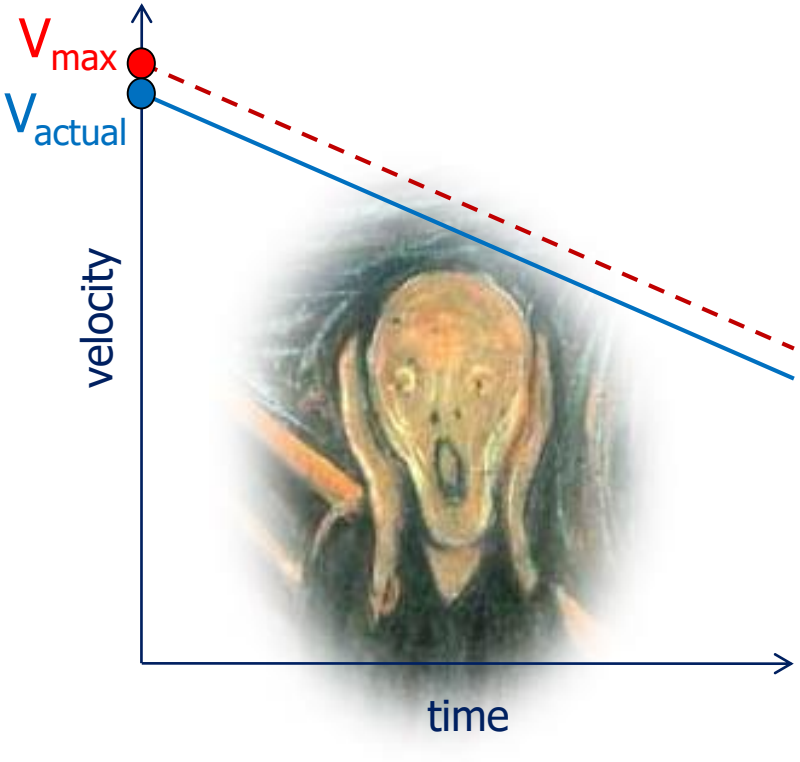
Final verification is crucial

but finding defects shall be an

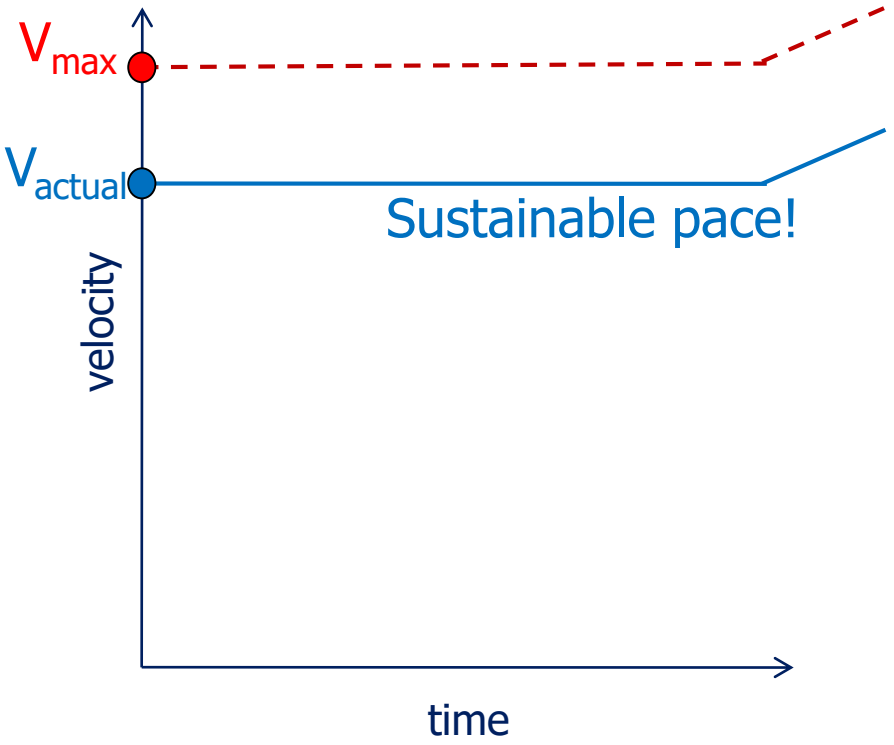
exception

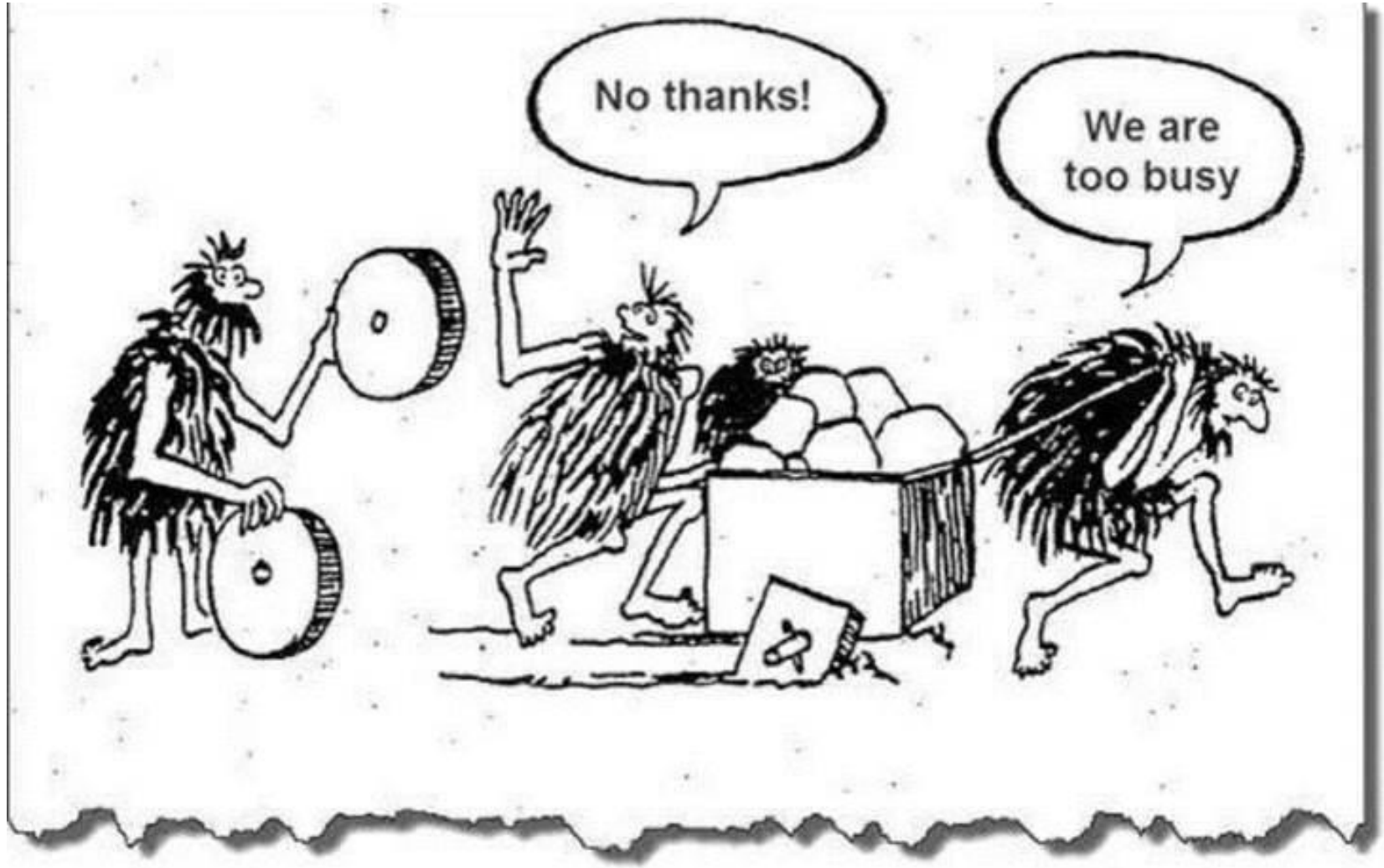
Technical debt = Financial debt

No refactoring
No unit tests
Little or no attention to quality
Overtime



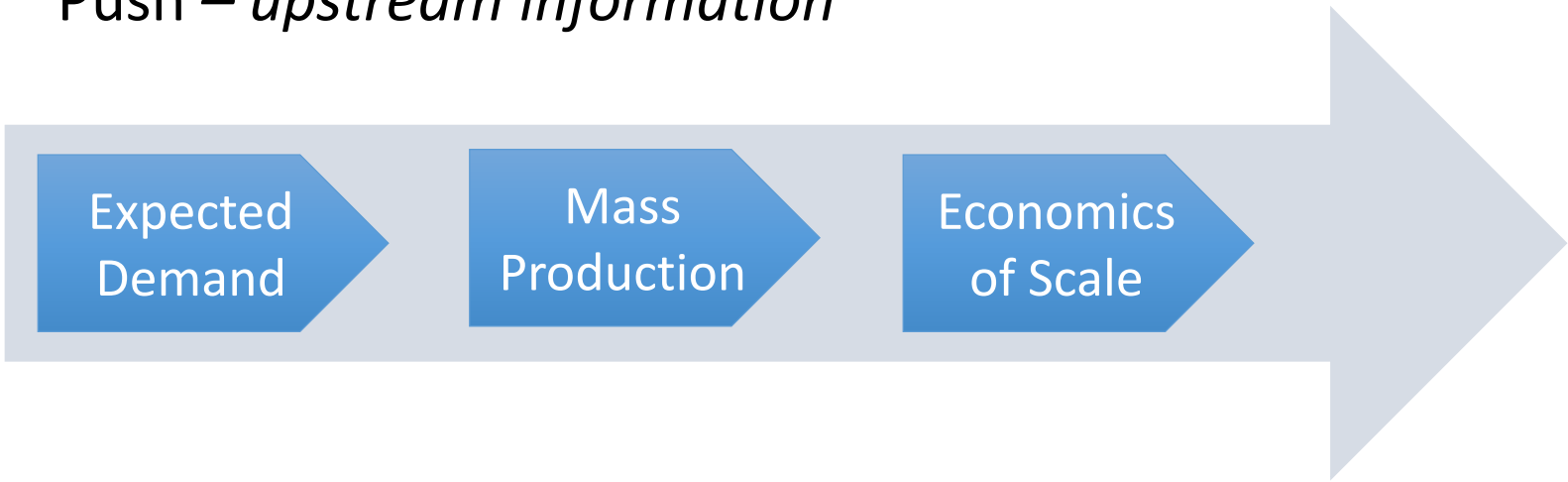
Refactoring
Unit tests
Attention to quality and excellence
Normal work hours




No thanks!
We are too busy

Push and Pull Systems

Push – *upstream information*



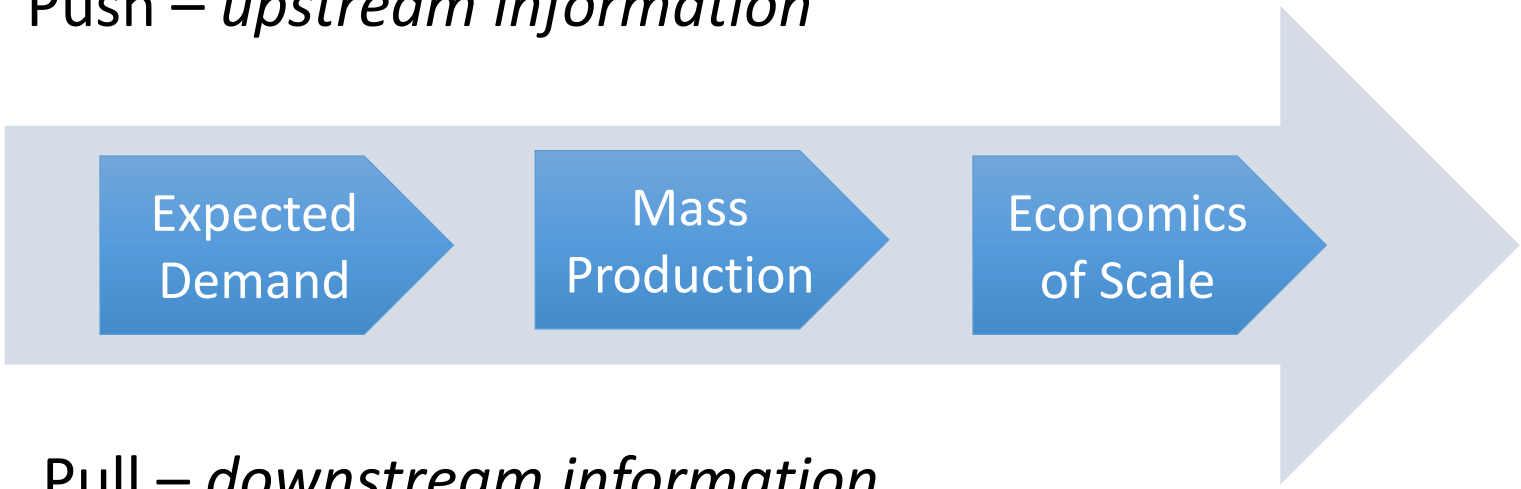
Push

”You can have any color, as long as it’s black”

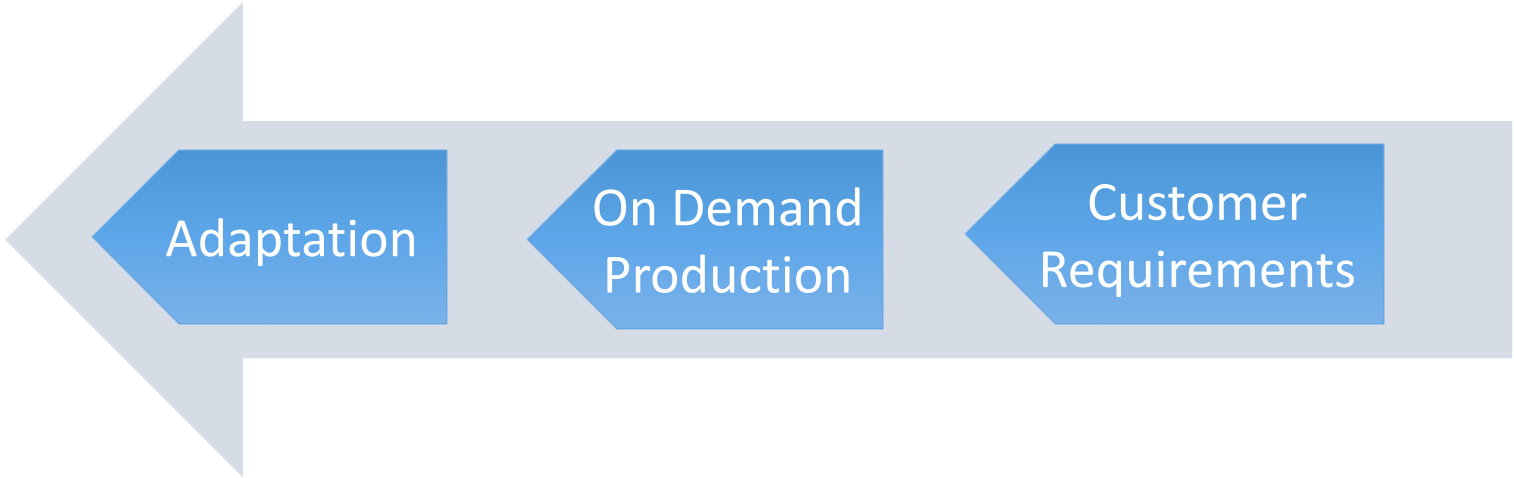
Henry Ford

Push and Pull Systems

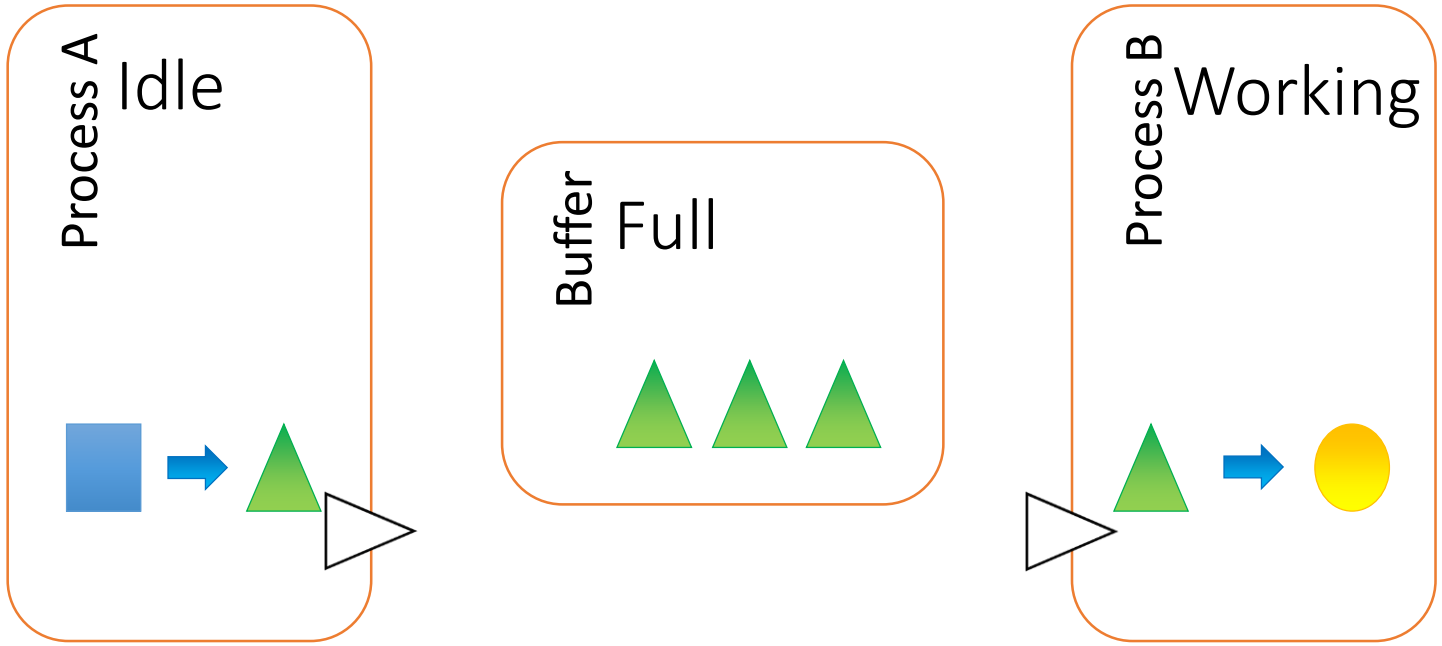
Push – *upstream information*



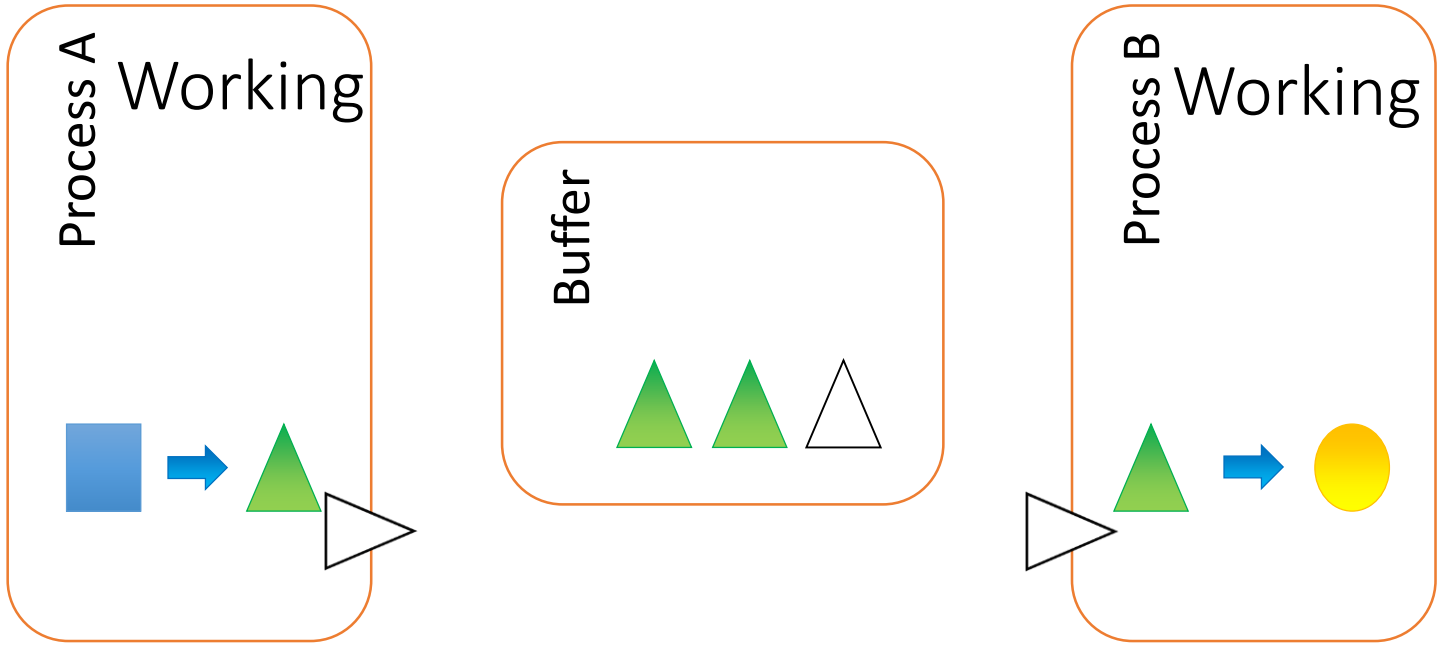
Pull – *downstream information*



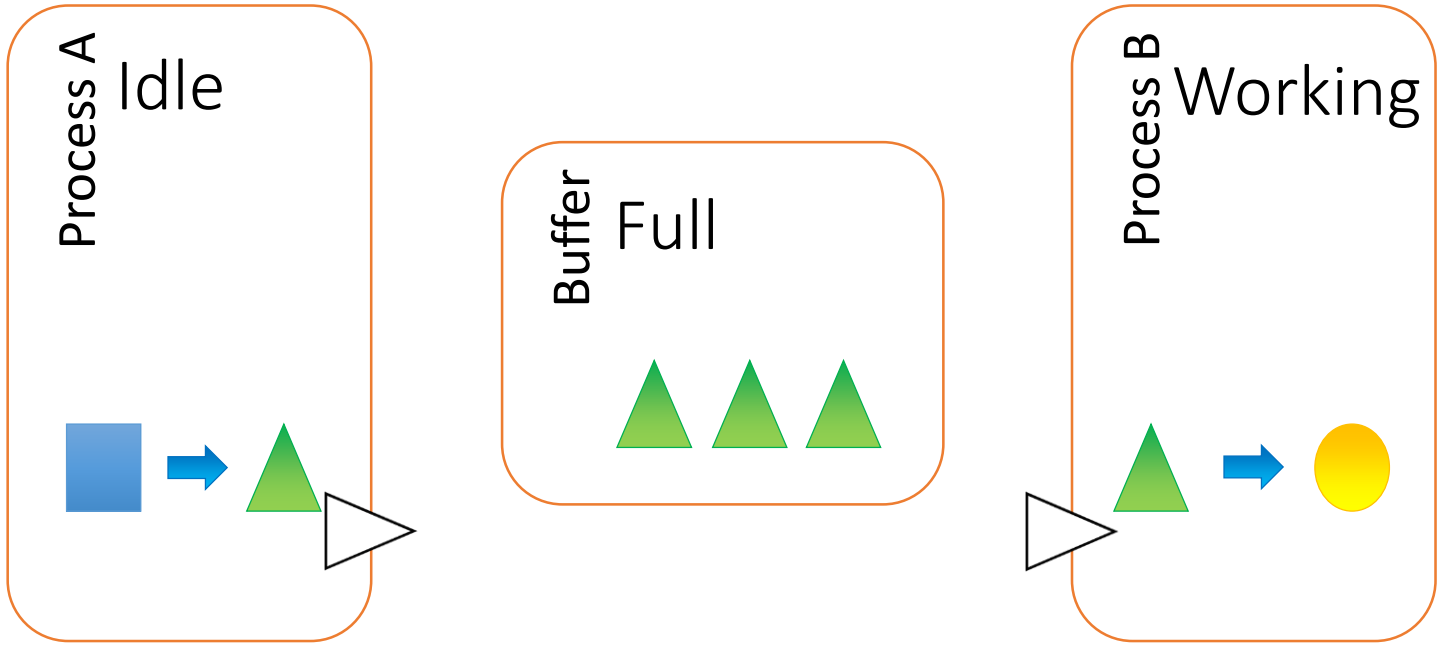
Kanban



Kanban



Kanban



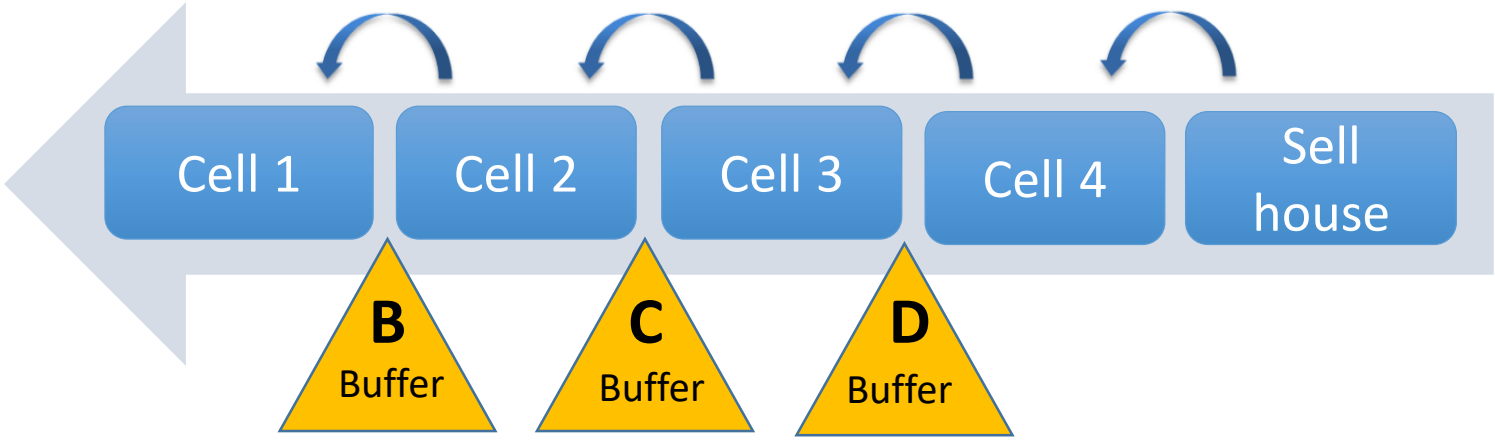
Kanban

Limits Work in Progress (WIP) on a pull system

Aids visual control →

Helps to get an even and maximized flow

Second Run – Pull Process



Pull System with Kanban

Demand comes first

Items are produced to fill gaps in the buffers

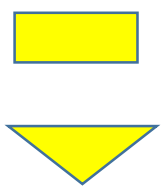
Setup buffers at intermediate steps

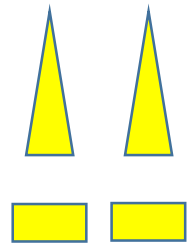
Second Run – Pull Process

1. Roles:
    1. Sales Manager
    2. Development team
2. FQ -Face Queue: One blank face
3. EQ- Eye Queue: Two sets of eyes, rectangular and triangular
4. UFQ: Two uncompleted faces with eyes attached only.
5. MQ: Two mouths, one rectangular, one triangular
6. **Each queue is created on one piece of paper**
7. **Fill all queues before starting**
8. *Wait for my signal* ☺
   4 minutes

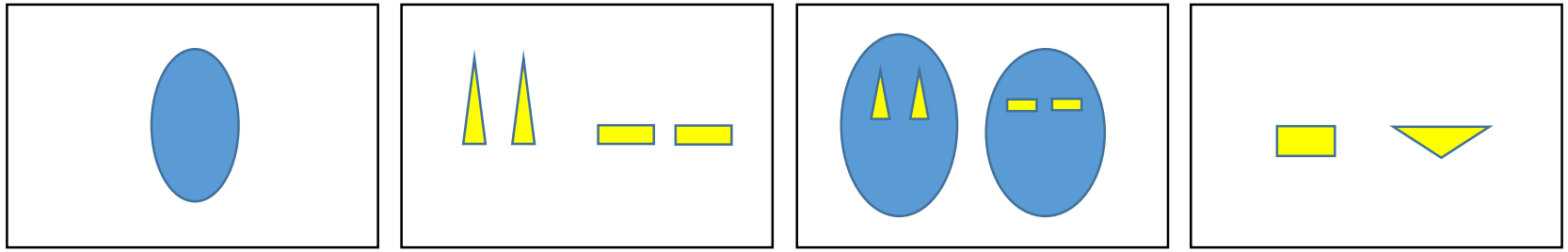
4 minutes

Happy Faces – The Models

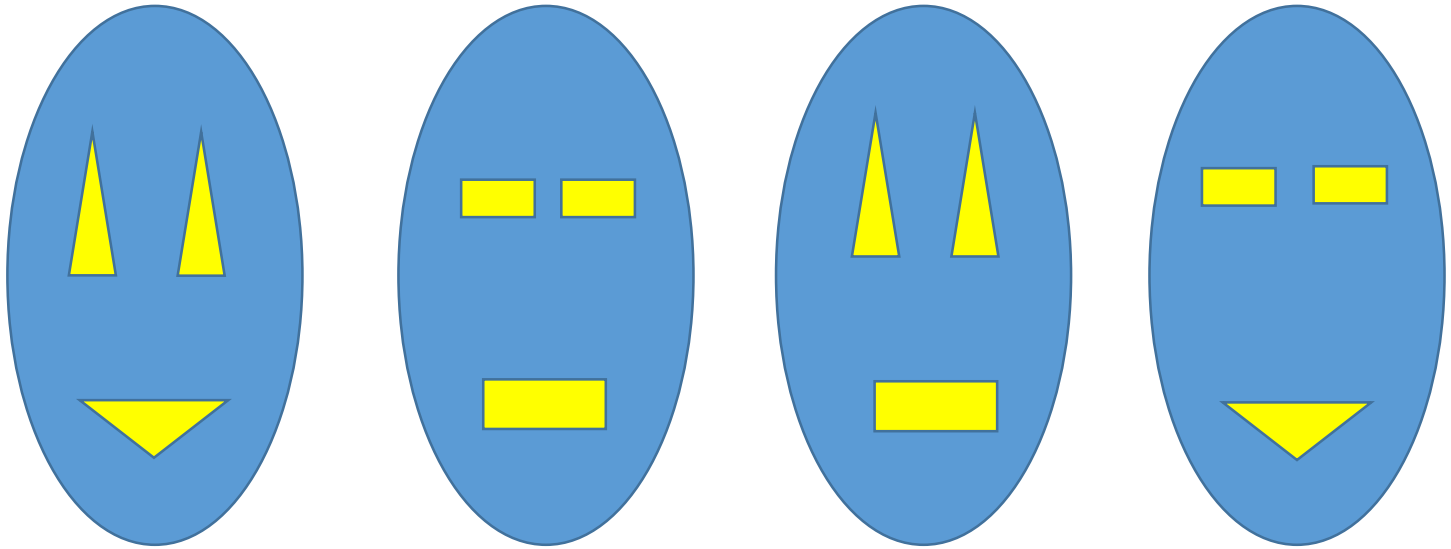




FQ EQ UFQ MQ



Happy Faces – The Models



Adelaide Laurent Pierre Sofie

Calculate profit and loss

Every sold face = +$400
Every unsold complete face = -$200
Every unsold eye = -$25
Every unsold mouth = -$50
Every uncompleted face = -$100

Retrospective

What good things did we do?
What could have been better?
What can we improve?

Unleveled Process
or
Production Smoothing

Get the production line smooth

Heijunka (stop the line when needed) 平準化

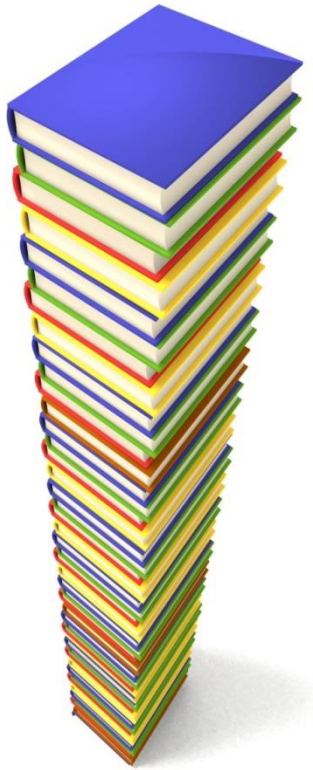
Some people working more than others

Reduce workers being idle

Sustainable pace

3.Amplify learning

Experiment and collect feedback
Stop and reflect
Share knowledge
Learning is essential



Kaizen = Continuous Improvement

改善

Reflect and adapt

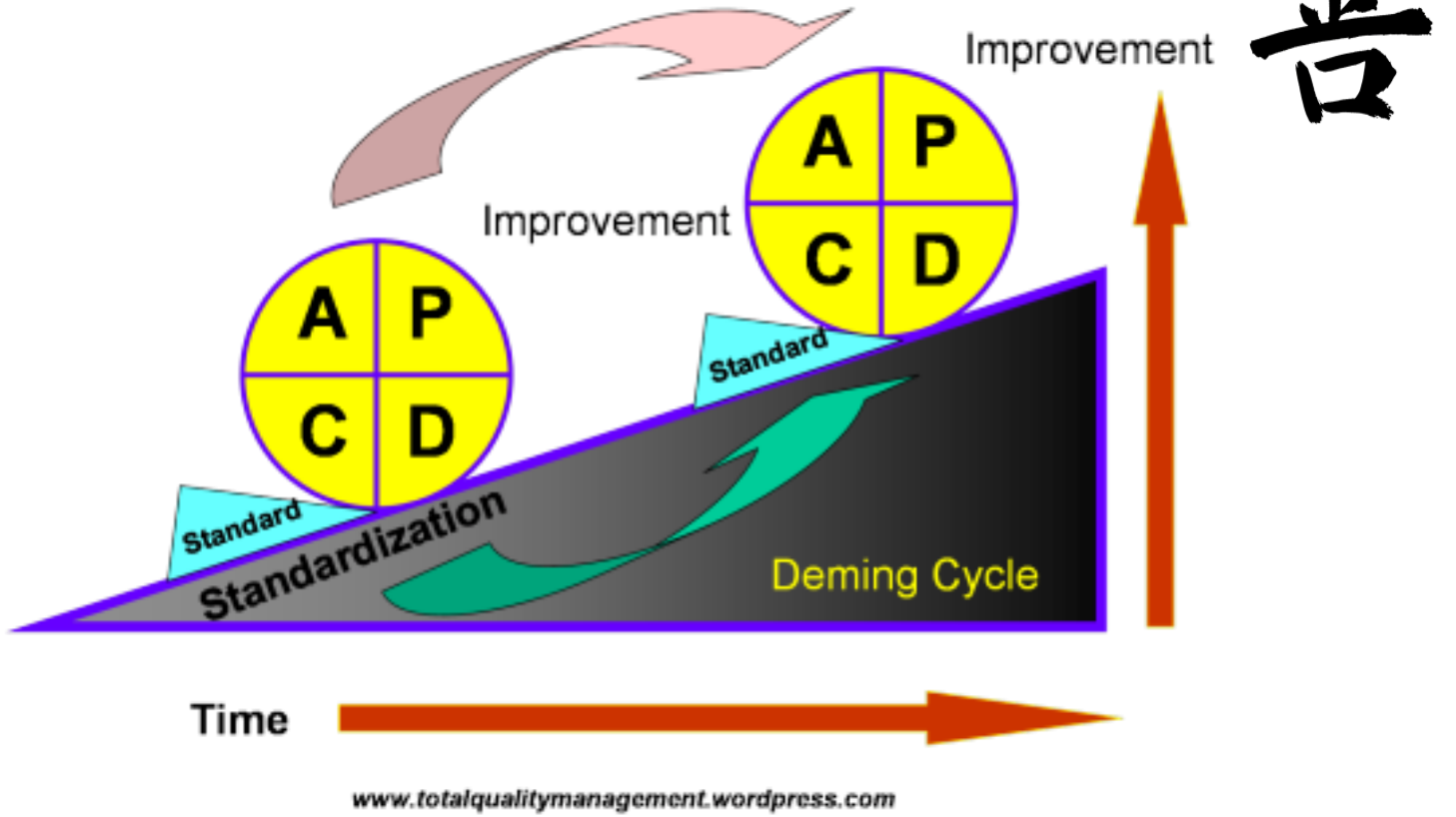
Long term thinking

Deming circle P-D-C-A

Learn by standardizing

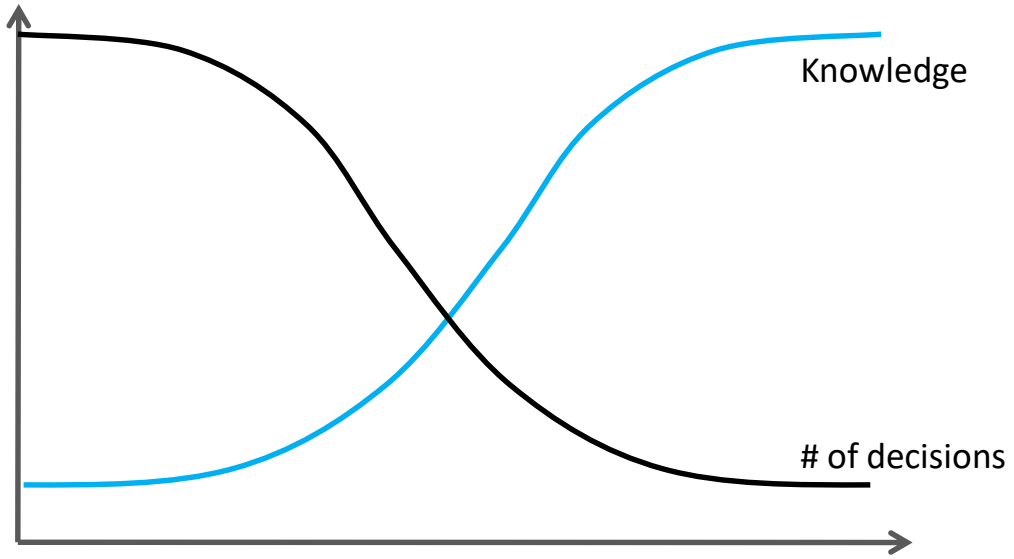
Kaizen = Improvement

改善



4. Decide as late as possible

Decide when the decision should be made



Abolish the idea that it is a good idea to start development with a complete specification

Myth: Planning is commitment

”In preparing for battle I have always found that plans are useless, but planning is indispensable”

General Dwight D. Eisenhower

A plan can and will change

Welcome it !

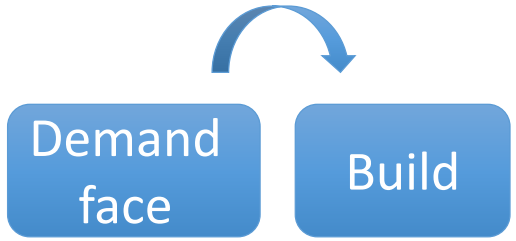
Break dependencies

The architecture shall support addition of any feature at any time

Maintain the possibilty of options

The architecture shall support addition of any feature at any time

Third Run – WorkCells



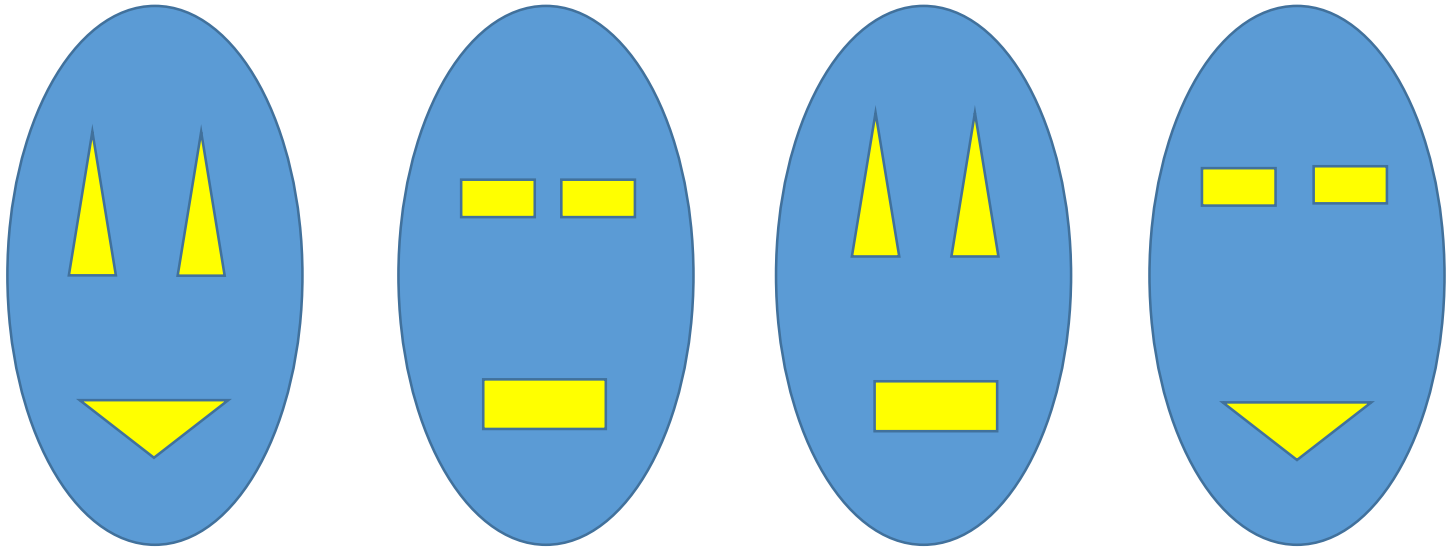
Work Cell

Each person builds a complete face

One round

4 minutes

Happy Faces – The Models



Adelaide Laurent Pierre Sofie

Calculate profit and loss

Every sold face = +$400
Every unsold complete face = -$200
Every unsold eye = -$25
Every unsold mouth = -$50
Every uncompleted face = -$100

Retrospective

What good things did we do?
What could have been better?
What can we improve?

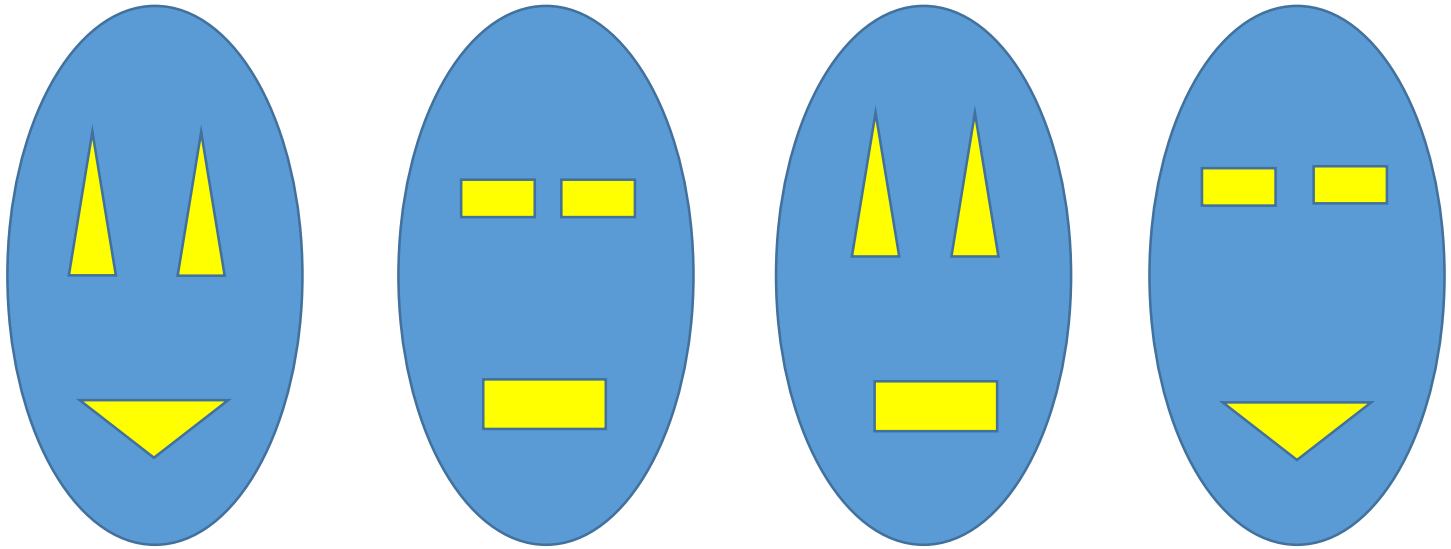
Forth Run – Improve the Process

Discuss your own process (2 min)

1 round

4 minutes

Happy Faces – The Models



Adelaide Laurent Pierre Sofie

Calculate profit and loss

Every sold face = +$400
Every unsold complete face = -$200
Every unsold eye = -$25
Every unsold mouth = -$50
Every uncompleted face = -$100

Retrospective

What good things did we do?
What could have been better?
What can we improve?

5. Deliver fast

Quick feedback

Deliver what the customer wants now

“Instead of having

many people spending a long time doing a big thing

you have

a small team spending short time doing a small thing”

6. Empower the team

Engaged people provide sustainable development advantage

Teams thrive on pride, commitment, trust and applause

The team knows best how things are done

Provide Servant Leadership

Effective teams have servant leaders who bring out the best in the team

Respect partners

7. See the whole

Optimize the whole

Do not sub optimize

Work for a common goal

Focus on the entire value stream

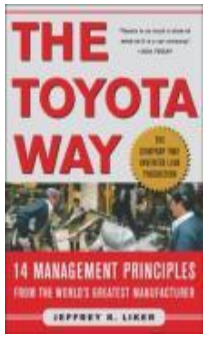
- From concept to cash
- From customer request to deployed software
- Cross organizations

Deliver a complete product

Complete products are built by complete teams

TAG
V-POWER
V-POWER

Lean Principles of Toyota



- **Kaizen**
- **Go see for yourself** (*Genchi Genbutsu*)
- Make **decisions slowly** by consensus, **implement rapidly** (*wemawashi*)

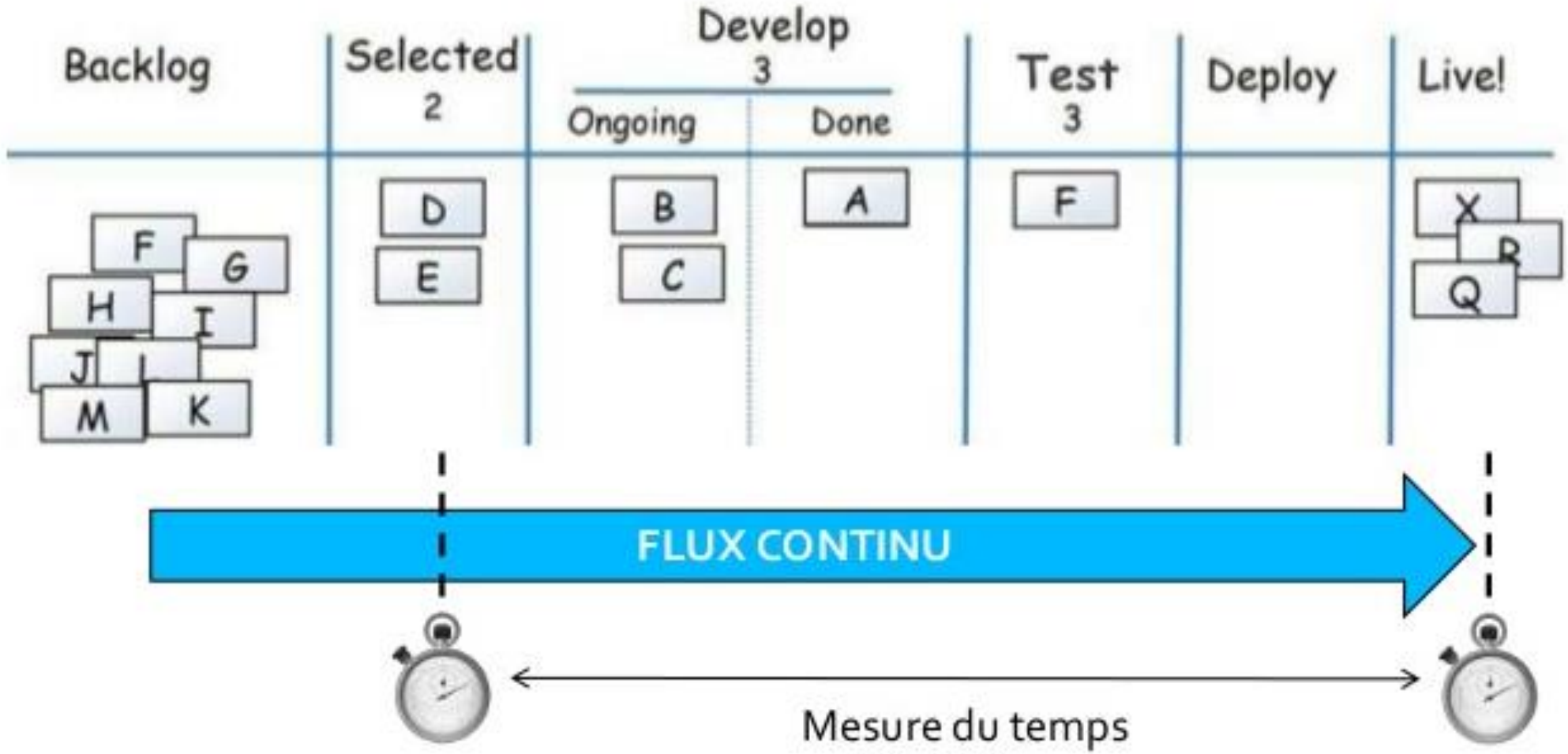
- Grow leaders who **live the philosophy**
- **Respect**, develop, and challenge your people and **teams**
- **Respect**, challenge, and help your **suppliers** and **partners**

- Create **process "flow"** to surface problems
- Use **pull systems** to avoid overproduction
- **Level** out the workload (*Heijunka)*
- **Stop** when there is a quality problem (*Jidoka)*
- **Standardize tasks** for **continuous improvement**
- Use **visual control** so no problems are hidden
- Use only reliable, thoroughly tested technology

- Base management decisions on a **long-term philosophy**, even at the expense of short-term financial goals

Apply Lean principles

Adjust it to fit **YOUR** reality


Backlog
Selected
2
Develop
3
Ongoing
Done
Test
3
Deploy
Live!
F
G
H
I
J
L
M
K
D
E
B
C
A
F
X
R
Q
FLUX CONTINU
Mesure du temps
TAKT Time

Agenda

History and Background

Lean Principles

Agile Manifesto

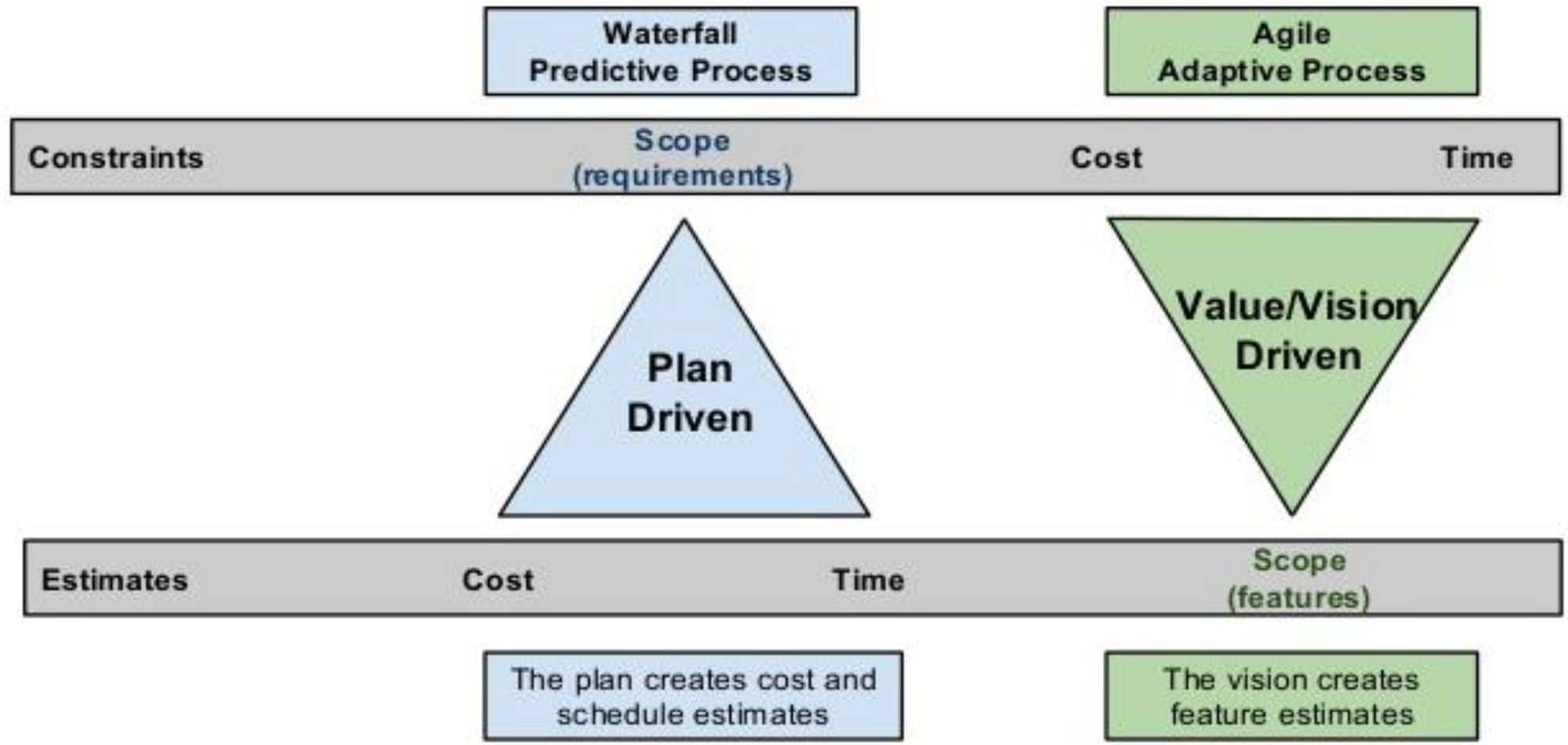
Scrum

Excercise on Scrum

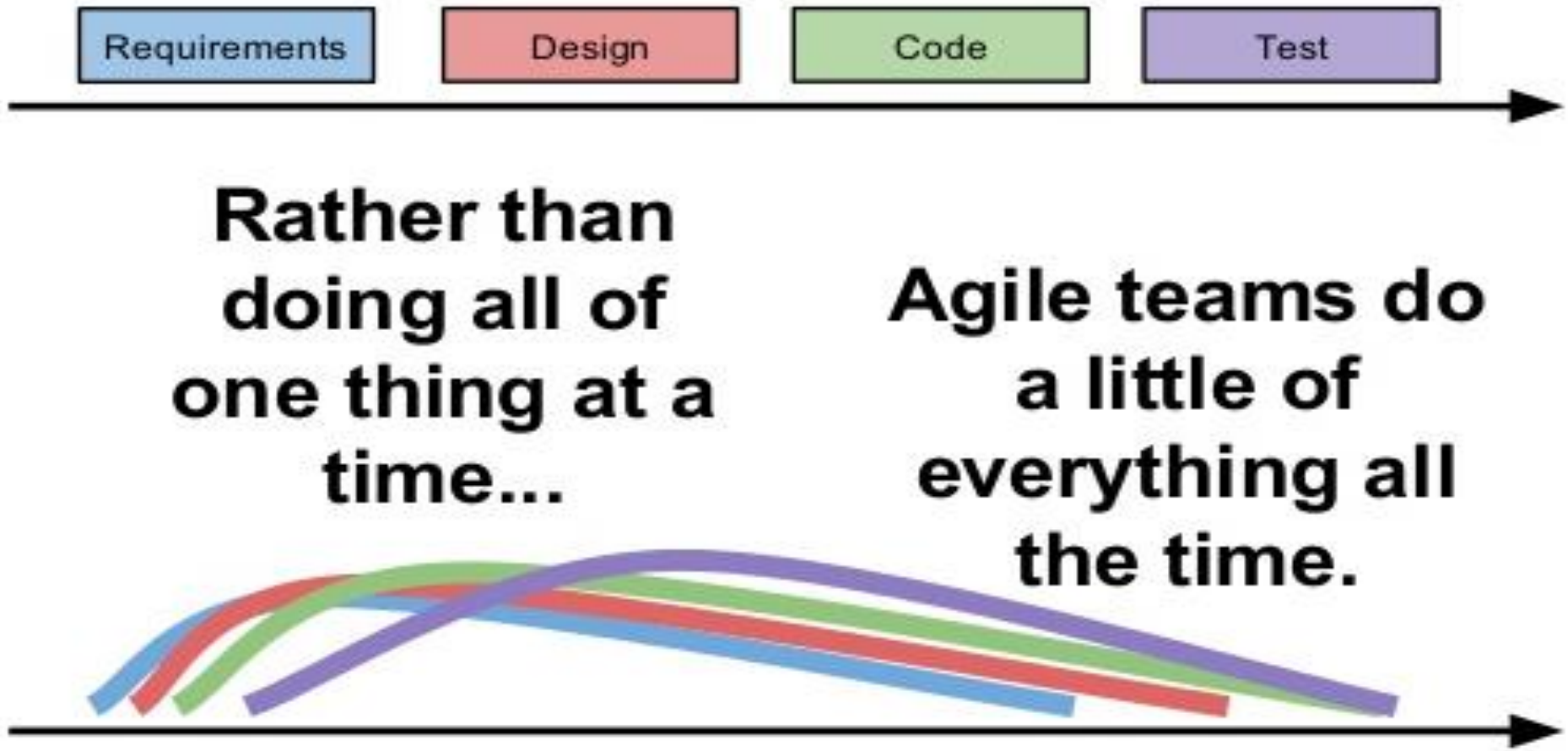
Summary

Agile

Prescriptive **vs.** Adaptive

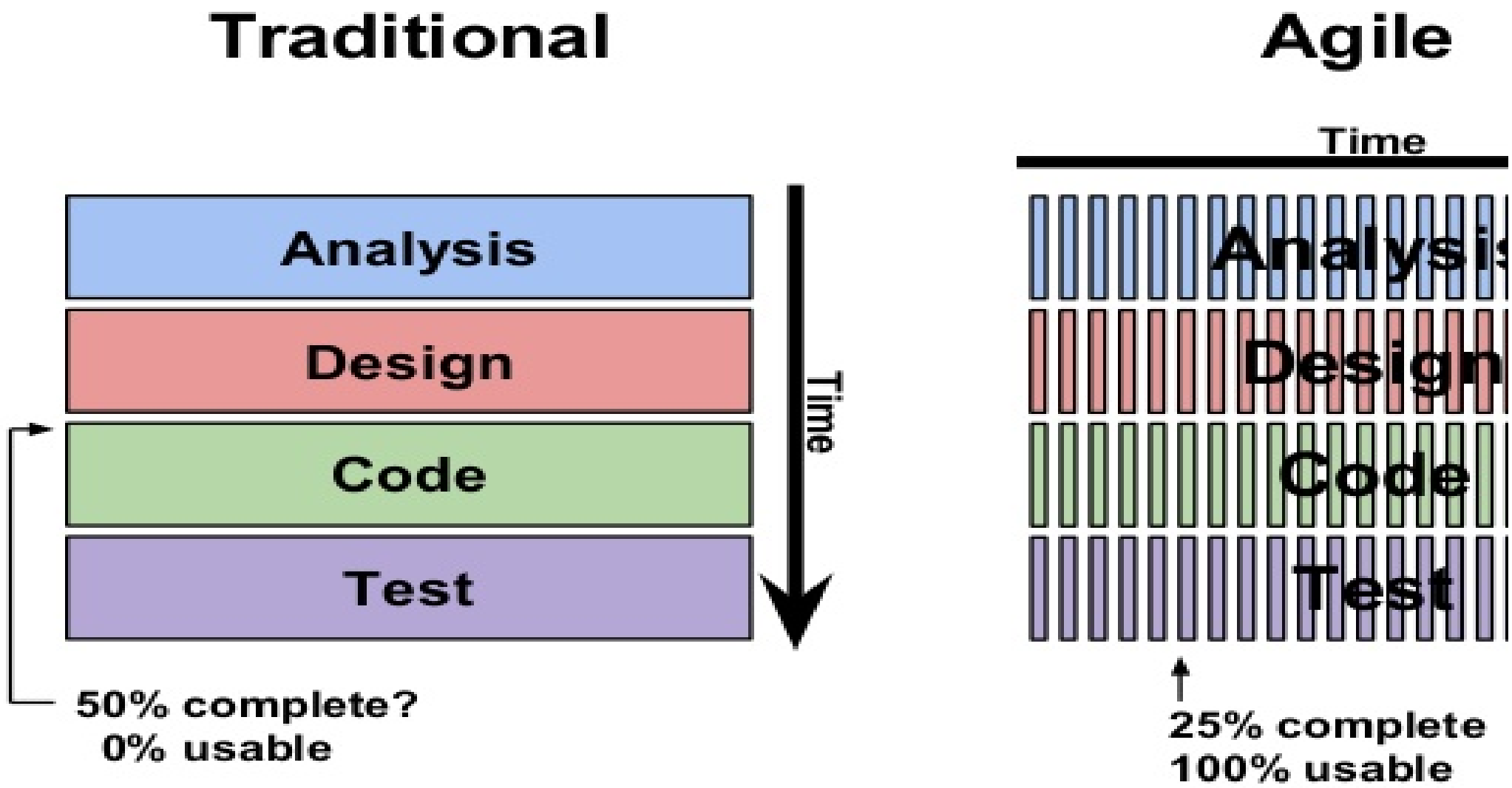


Sequential vs. Overlapping development



Source: "The New New Product Development Game" by Takeuchi and Nonaka. Harvard Business Review, January 1986.

Software development process



What is Agile?

The english term agile means

- Light
- Alerte
- Quick
- Ability to move or think easily and quickly

Agile software development

- Has no definition
- Umbrella term for similar projects
- The term coined in 2001

Manifeste Agile

Nous sommes arrivés à préférer et favoriser:

Les individus et leurs interactions plus **que les processus et les outils**

Du logiciel qui fonctionne plus **qu'une documentation exhaustive**

La collaboration avec les clients plus **que la négociation contractuelle**

L'adaptation au changement plus **que le suivi d'un plan**

Cela signifie que bien qu'il y-ait de la valeur dans les items situés à **droite**, *notre préférence se porte sur les items qui se trouvent sur la* **gauche**.

Principles behind the Agile Manifesto
6 out of 12

Highest priority to satisfy customer

through early and continuous delivery of valuable software

Welcome changing requirements

even late in development

Business people and developers must work together daily

Build projects around motivated individuals

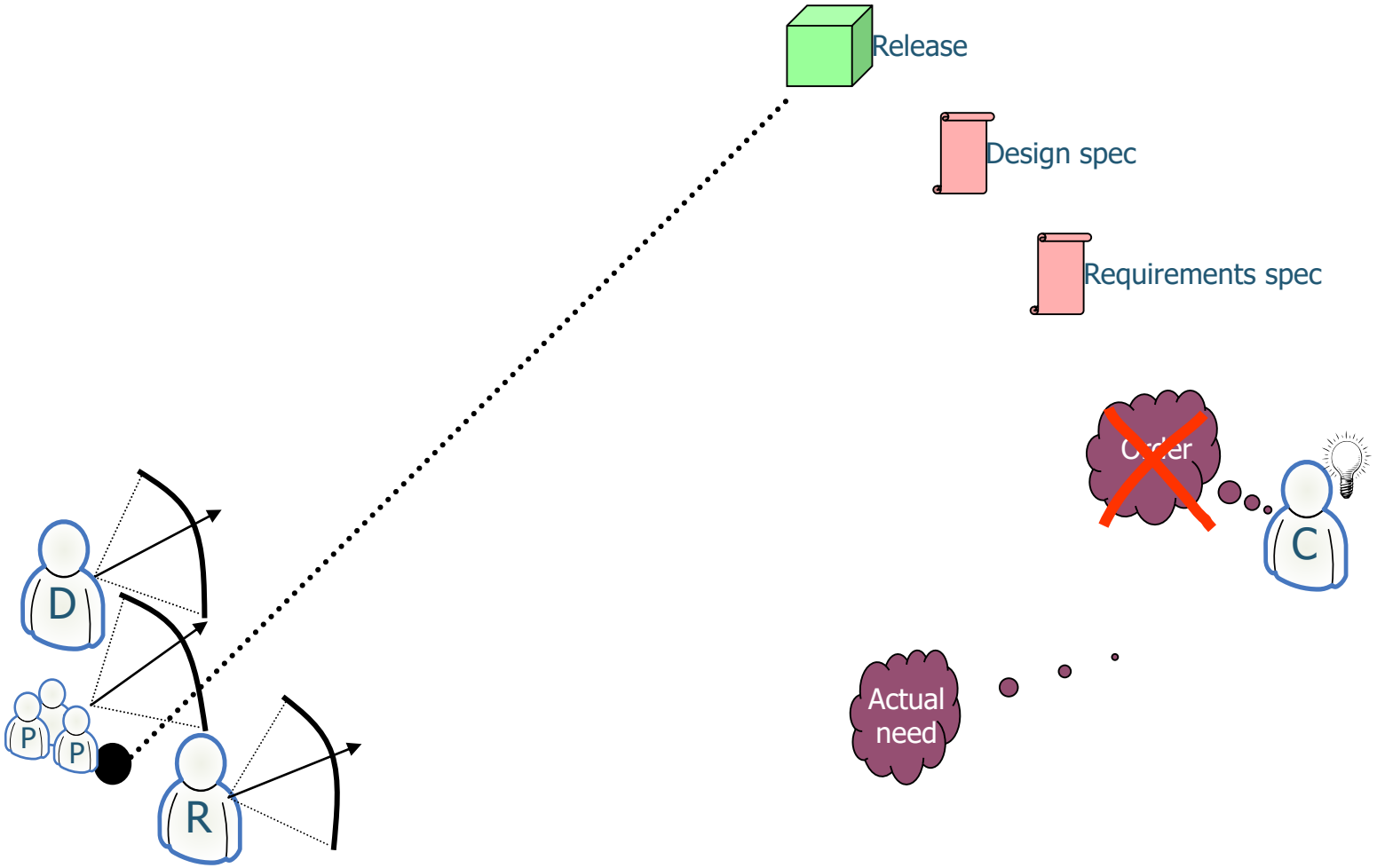
Give environment and support, trust them to get the job done

The best architectures, requirements, and designs emerge from self-organizing teams

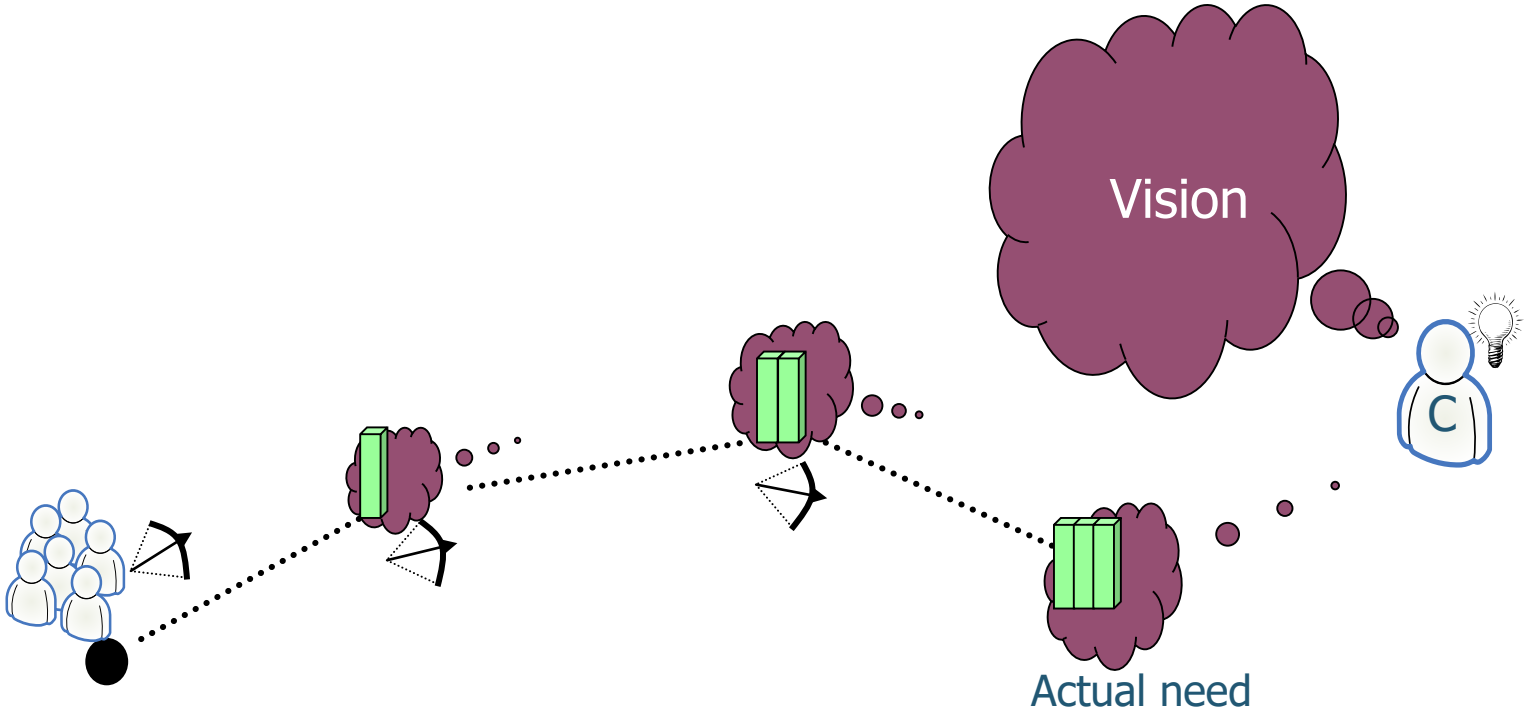
At regular intervals the team reflect on how to become more efficient

The team tunes and adjust, KAIZEN 改善

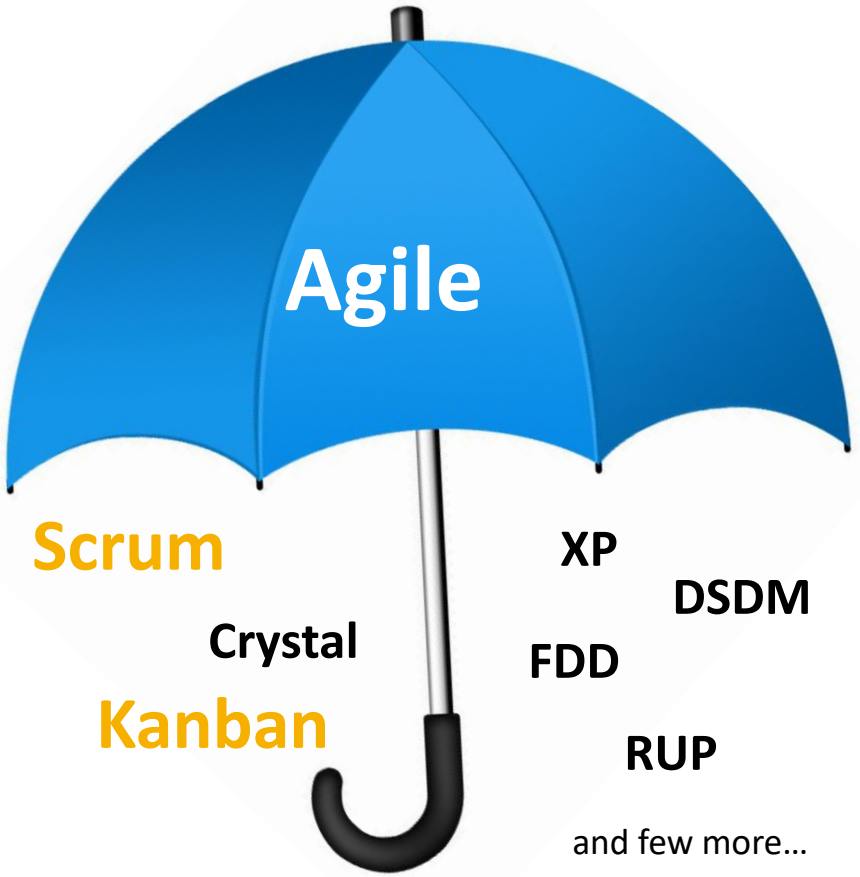
Predictive approach



Adaptive approach



Agile Umbrella



More Prescriptive
more rules to follow

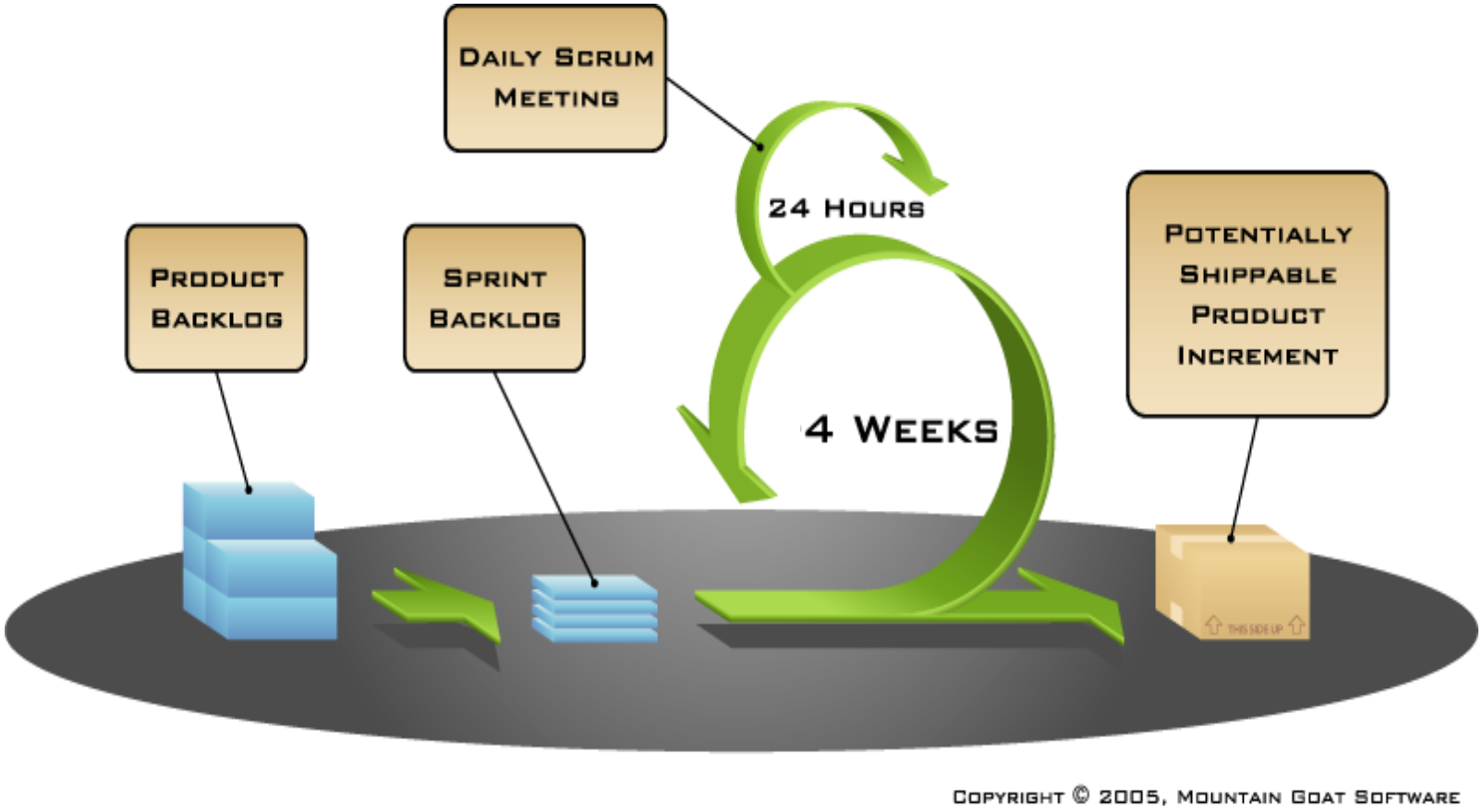
- **RUP (120+)**
  RUP has over 30 roles, over 20 activities, and over 70 artifacts
- **XP (13)**
- **Scrum (9)**
- **Kanban (3)**
- **Do Whatever!! (0)**

More Adaptive
fewer rules to follow

Scrum

Scrum

Process overview



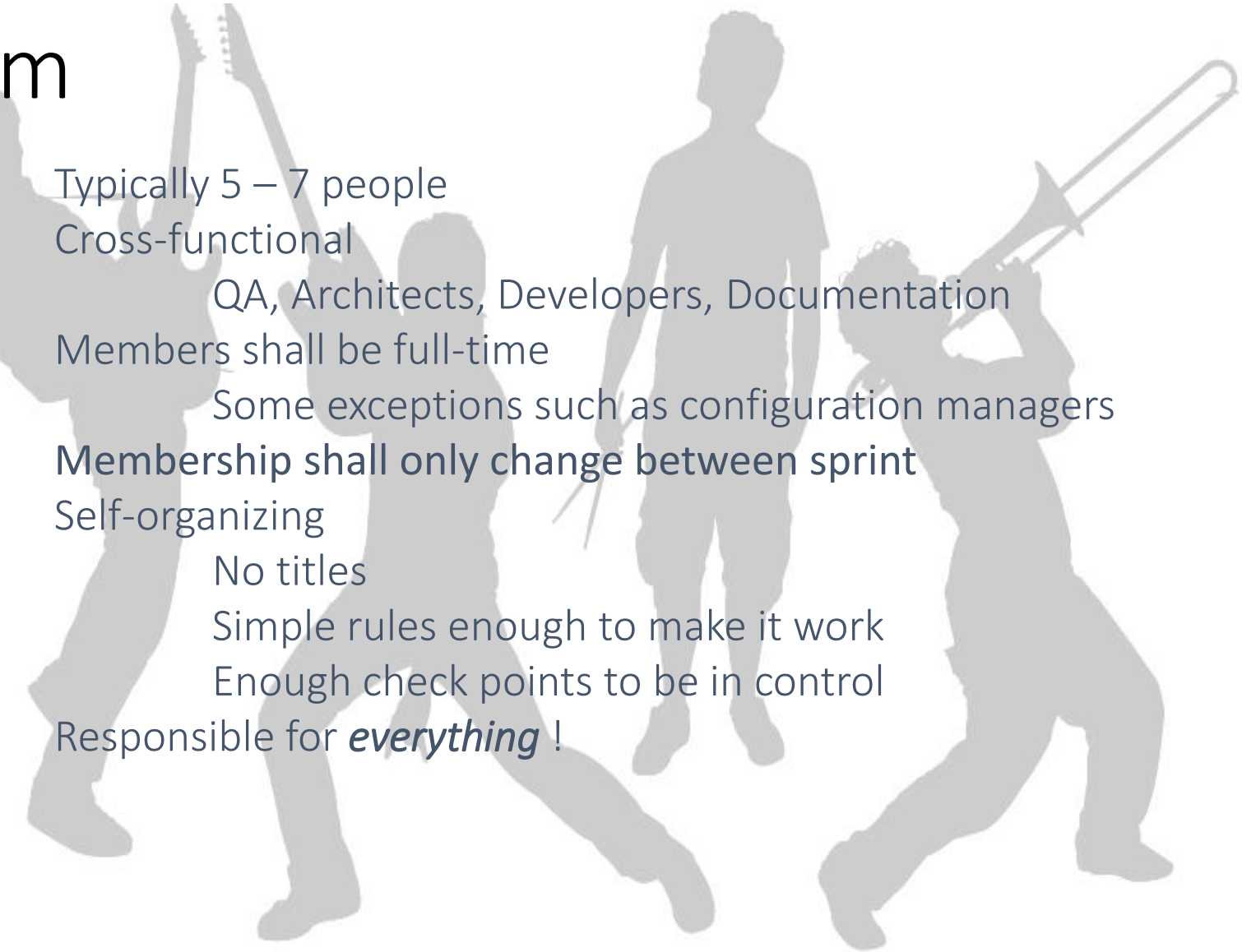
Roles

Propriétaire
du produit

L’équipe

Scrum
master

The team



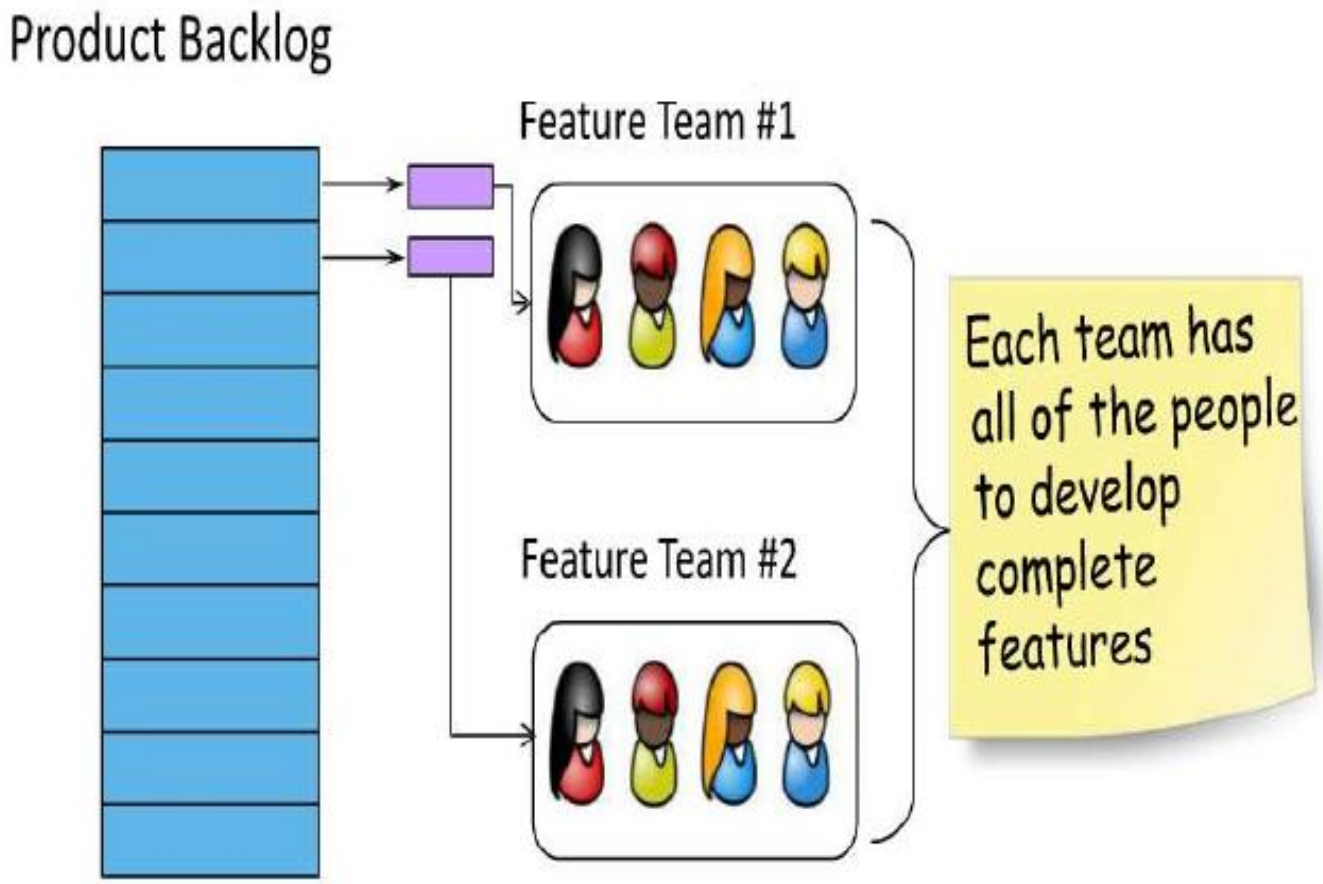
Typically 5 – 7 people
Cross-functional
- QA, Architects, Developers, Documentation

Members shall be full-time
- Some exceptions such as configuration managers

Membership shall only change between sprint
Self-organizing
- No titles
- Simple rules enough to make it work
- Enough check points to be in control

Responsible for ***everything*** !

Feature Team



Scrum Master

Facilitator and leader

- For the team and the Product Owner (PO)

Ensures that the process is followed

Remove *Impediments*

- Shields the team from external interferences

Sees to that manual processes are automated

Improves the productivity of the team

Improves the engineering, practices and tools

Keeps information about progress up to date and visible

Product Owner

Owns the product backlog

Keeps the backlog updated

Keeps the backlog prioritized and estimated

Decides, with the team if new bugs are added to the sprint backlog

Have a good understanding of what the team shall deliver

Participates in the Scrum activities

Communicates with all stake holders

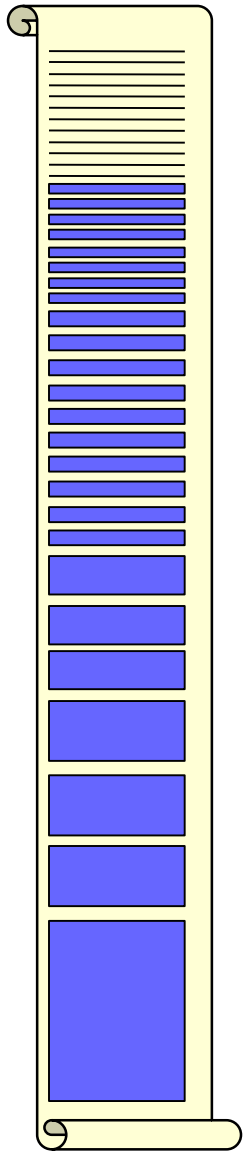
Product backlog

A list of features and qualities

Prioritized and estimated continuously

Planning should be done in levels

- Long term plans are high level view
- More and more detail for near future



User Story

As a <Role>, I want <goal/desire> so that <benefit>

- As a retail user, I want to pay with a credit card, so that I can pay immediately
- As a trader, I want to modify quantity of quote in market, so that I can stay in the spread
- Bad: A user must find the software easy to use
- Bad: All error handling and logging is done through a set of common classes

User Story, a Requirement

INVEST

I	Independent	Self-contained No inherent dependency on another user story.
N	Negotiable	User stories, up until they are part of an iteration, can always be changed and rewritten.
V	Valuable	Deliver value to the end user.
E	Estimable	Possible to estimate the size of a user story.
S	Sized appropriately or Small	Possible to plan with a certain level of certainty.
T	Testable	Make test development possible.

User Stories: Further reading

Sprint Planning Meeting

Two parts - what and how?

As much as needed to get started

Sustainable pace

Review the Definition of Done

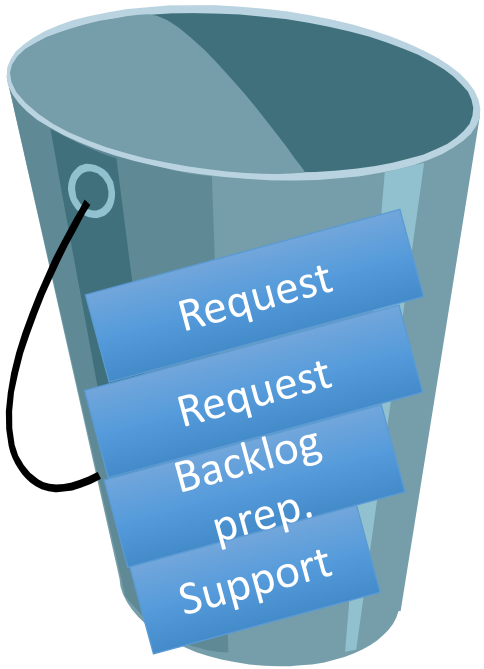
Result:

Sprint length

Scope

Sprint goals

Time of daily meetings



Definition of Done (example)

- Development and admin
  - SPR implemented and tested by developer.
  - SPR updated with relevant information .
    - Test
      - SPR tested on by another person than developer
      - Automated unit tests implemented
        - Documentation
          - SPR description field complete and clear
          - Solution description field complete and clear

The Sprint

The team owns the sprint

Target duration is one month, +/- one week

All items in the Sprint backlog must be Done

- Product is designed, coded, documented and tested

No changes during the sprint

- Plan duration to fit the time you can keep change out

A sprint may be interrupted in exceptional cases

- Inform stakeholders
- Make a new planning and continue sprint

Sprint backlog – Scrum board

Daily Scrum Meetings

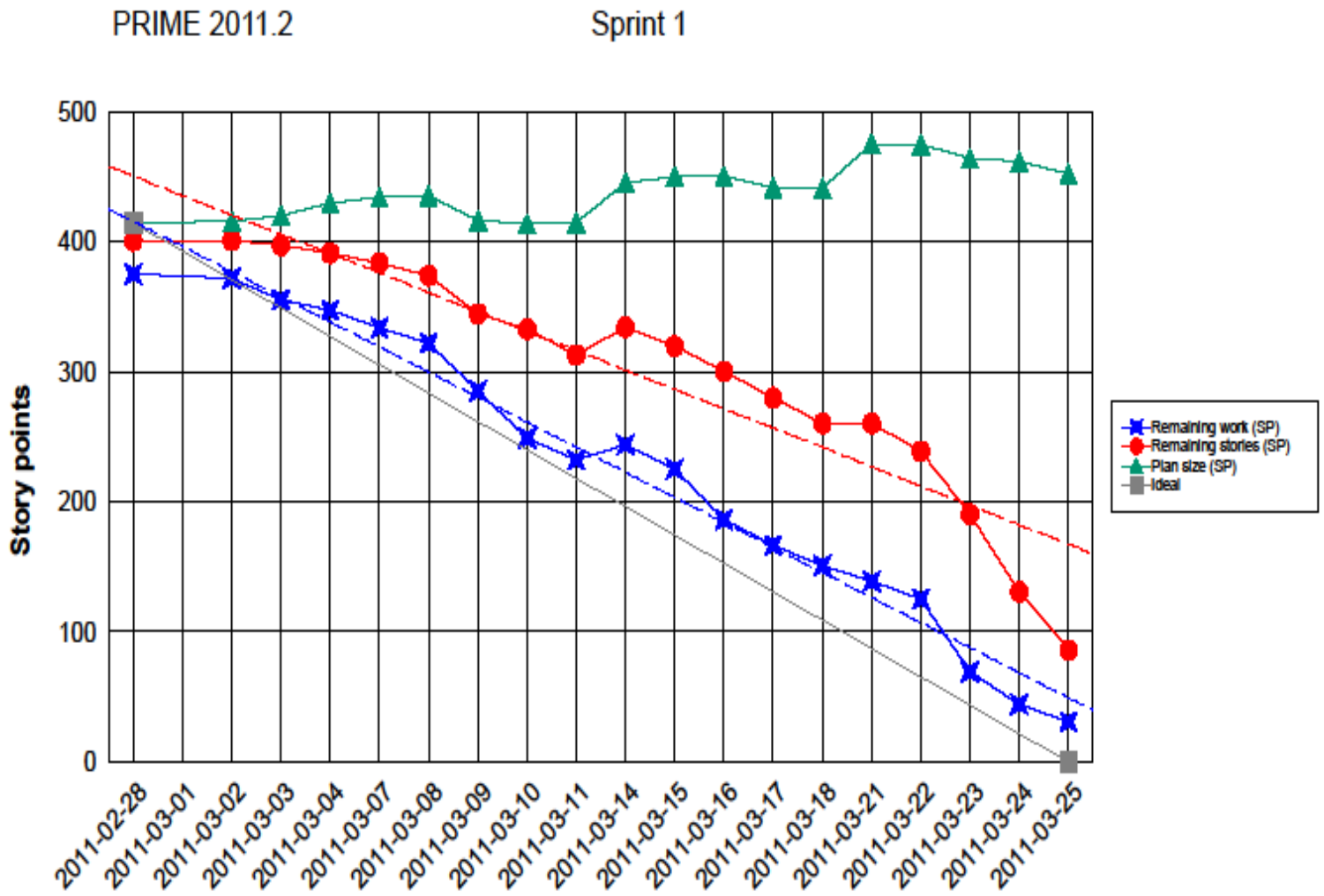
- Not more than 15 min
- Stand-up
- Three questions
  - What have you done since last meeting?
  - What will you do until next meeting?
  - What impediments are in your way of being as efficient as you could?
- Report to the **team**, not the Scrum master
- Synchronization, not status reporting
- Review time estimates
- If possible solve impediments within the team

Daily Scrum Meetings

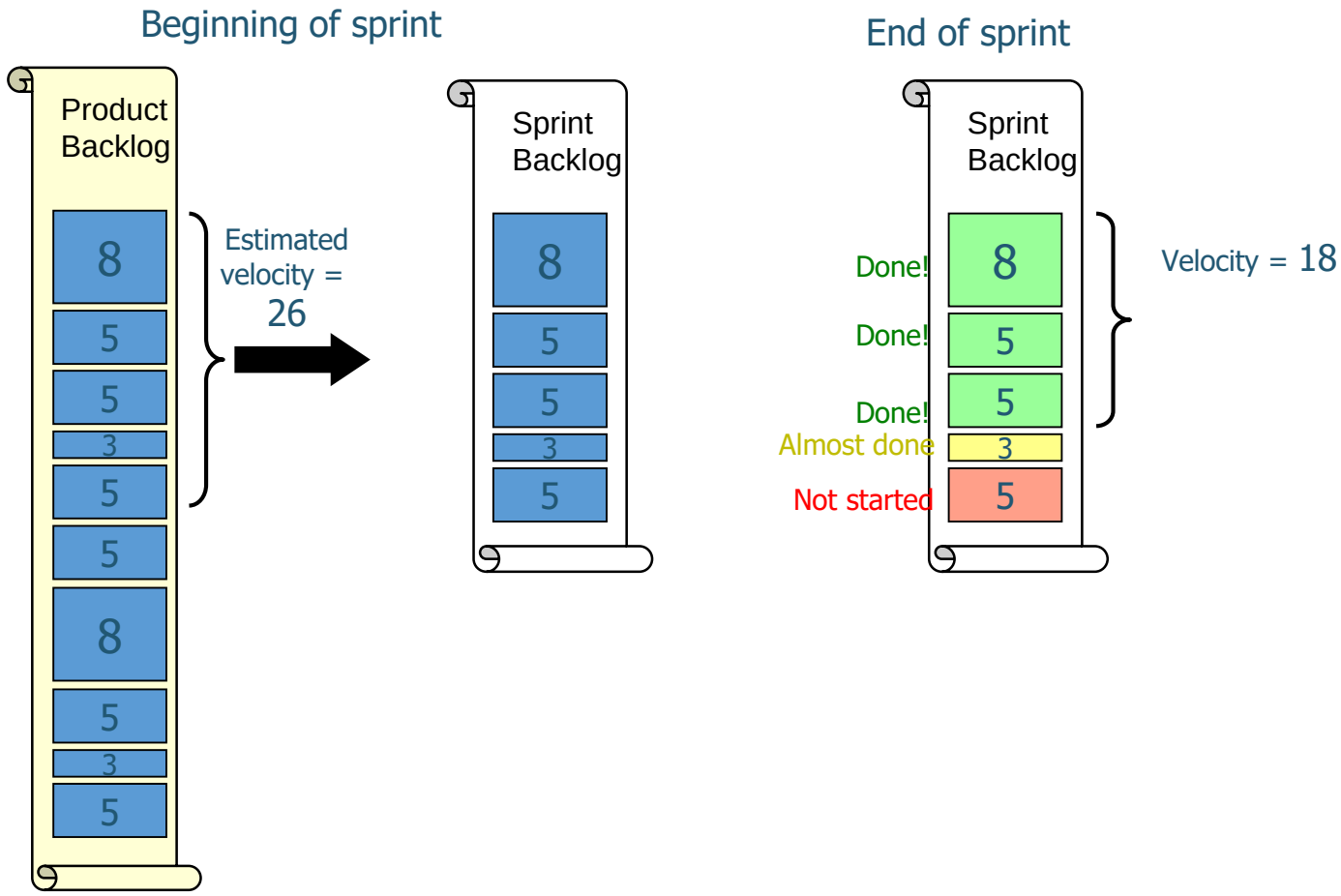
Why daily?

”How does a project get to be a year late?”
“One day at the time!”
Fred Brooks, The Mythical man-Month

Burndown


PRIME 2011.2
Sprint 1
Story points
500
400
300
200
100
0
2011-02-28
2011-03-01
2011-03-02
2011-03-03
2011-03-04
2011-03-07
2011-03-08
2011-03-09
2011-03-10
2011-03-11
2011-03-14
2011-03-15
2011-03-16
2011-03-17
2011-03-18
2011-03-21
2011-03-22
2011-03-23
2011-03-24
2011-03-25
Remaining work (SP)
Remaining stories (SP)
Plan size (SP)
Ideal

Velocity


Beginning of sprint
Product Backlog
8
5
5
3
5
5
8
5
3
5
Estimated velocity = 26
Sprint Backlog
8
5
5
3
5
End of sprint
Sprint Backlog
Done!
Done!
Done!
Almost done
Not started
8
5
5
3
5
Velocity = 18

Sprint Review Meeting

The team presents what it accomplished during the sprint

Informal, 2 hours preparation

Get feedback on new/improved product features

No preparations

Only DONE requirements

Product owner accepts or rejects

Prove that the product works!

Retrospective

What can we do to improve our process?

Reflect on how we work

- What good things did we do?
- What could have been better?
- What can we improve?

Identify actions and follow ups

Find the root cause

Make sure everyone has a voice

Do it in many different ways

Scrum of Scrums

All Scrum masters meet

Short meeting

Stand-up meeting

Work on dependencies between Scrum teams

Synchronize

Once a week

Engineering Practices

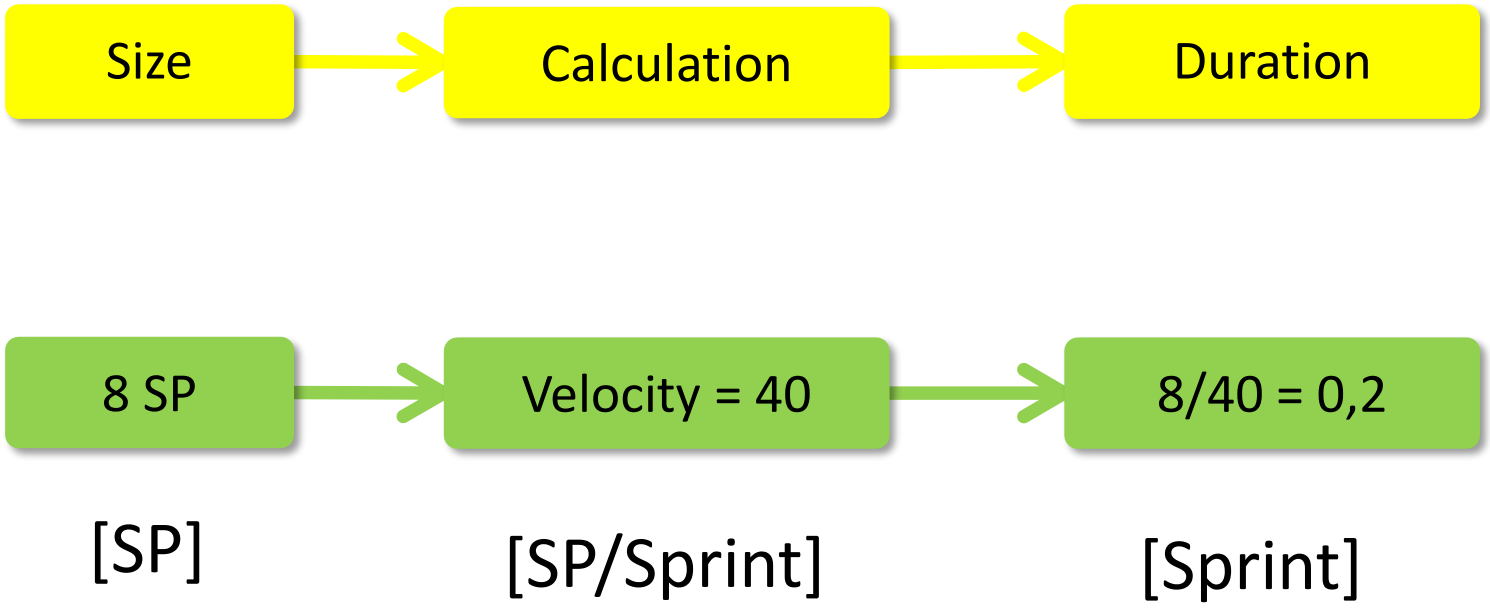
- Not prescribed, but necessary!
- Automatic Testing
- Pair Programming / Code Reviews
- Emergent Architecture
- Xtreme Programming is a good source

Story points

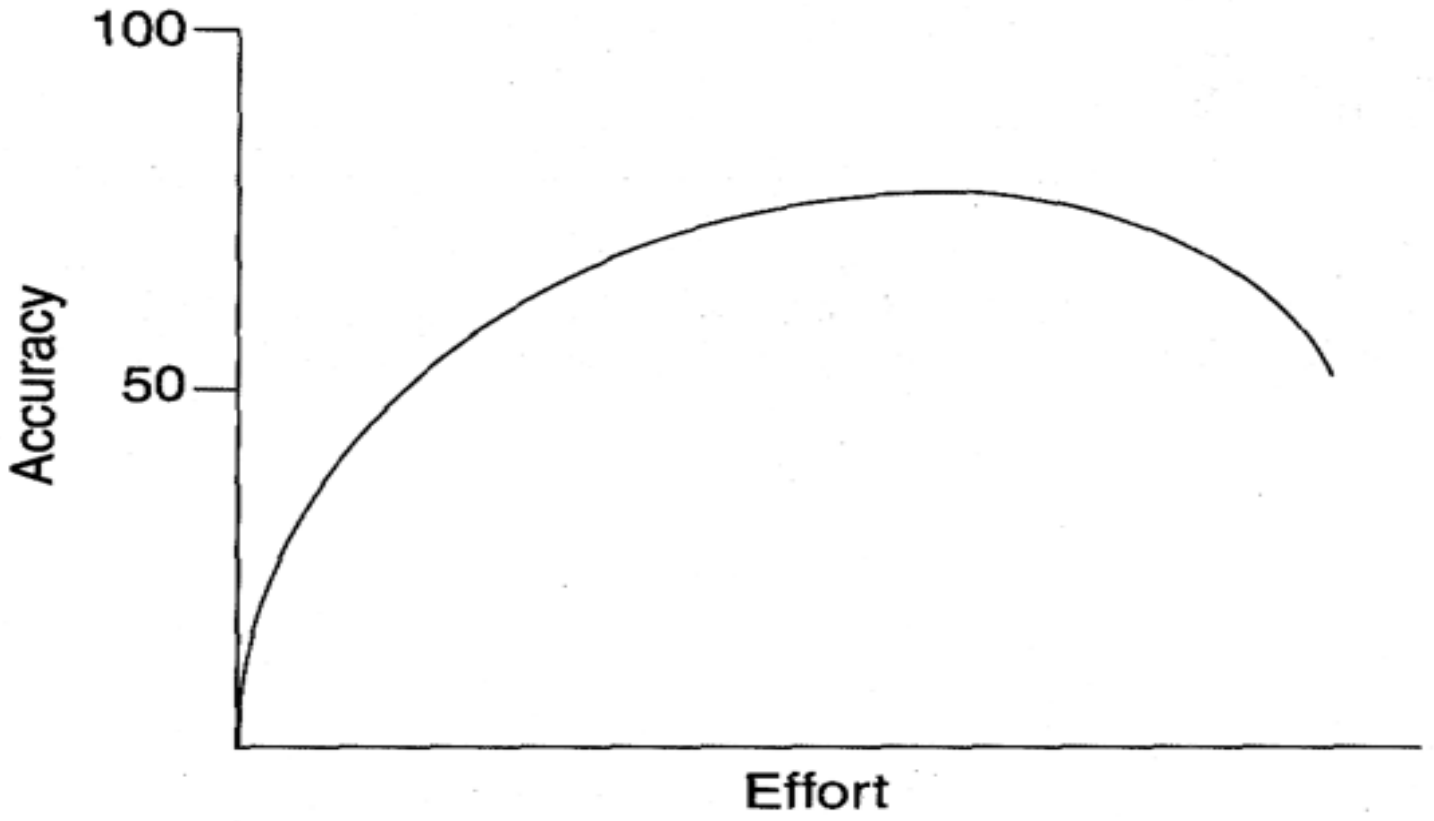
- The ”biggnes” of a task
- Influenced by
  - How hard it is
  - How much there is

    1,2,3,5,8,13,20,40,100
- Relative values
  - A login screen is a 2
  - A search feature is an 8
- Points are unit-less

Estimate Size – Derive Duration



Estimating



- Meet regularly and estimate the backlog

”It’s better to be roughly right
than precisely wrong.”

John Maynard Keynes

Planning Poker – An iterative process

Each estimator is given a deck of cards

1,2,3,5,8,13,20,40,100

Product Owner reads a story

Discuss it briefly

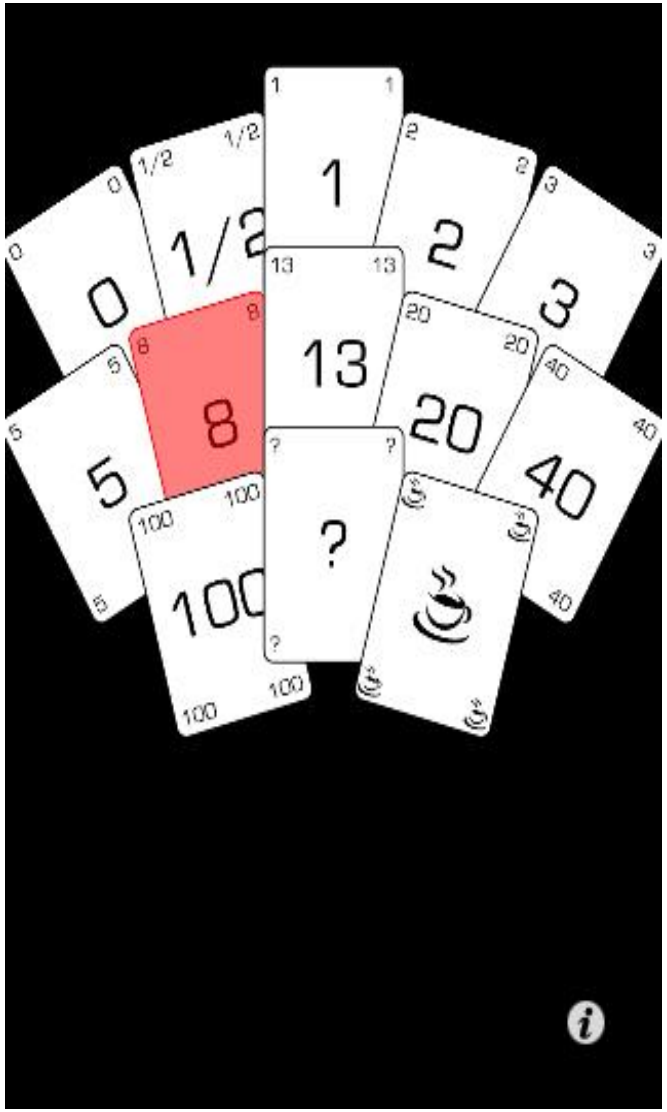
Each estimator selects a card

Cards are turned at the same time

Discuss differences, especially outliers

Re-estimate until estimates converge

Do not use average



The Scrum Exercise

Assign a Product Owner. The rest is the team.

1. Estimate the Product Backlog (Team, PO do not participates).
2. Prioritize the Product Backlog , Estimation + Business Value (PO).
3. Create the Sprint Backlog , the sprint length is **5 minutes** (Team).
4. Calculate your Planned Velocity.

5. **We run the Sprint**, start at the same time. Wait for *my* signal

6. Run the Demo. The PO accepts or rejects each item.
7. Calculate your Actual Velocity and Business Value based on tasks that are Done
8. Do the Retrospective and decide on Improvements for the next sprint

The Scrum Exercise - Rules

The PO prepares everything, e.g the deck of cards

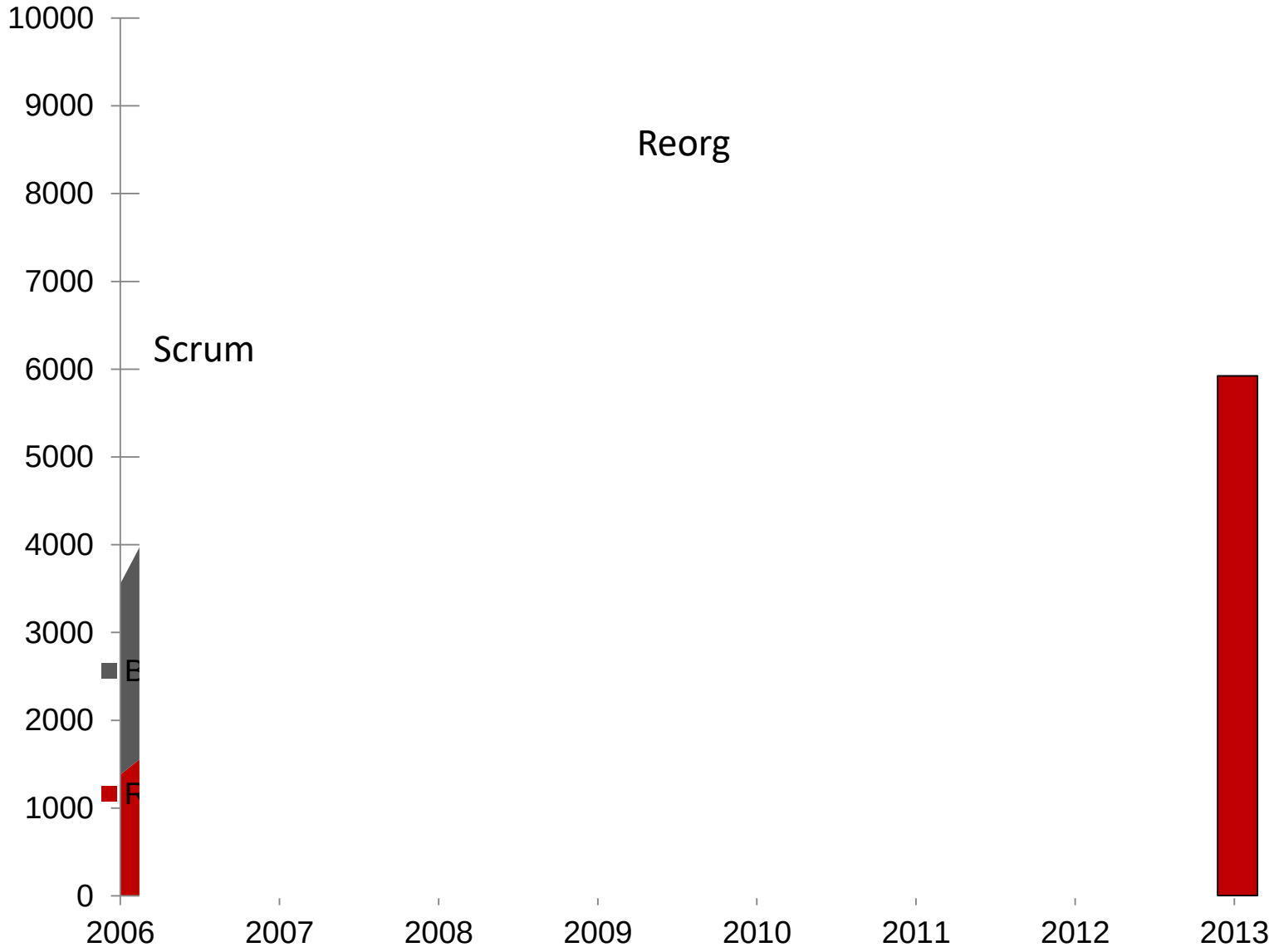
The PO do not participate in executing of the tasks

Team can only work on one item at a time, all together!

Calculators are not allowed

If you finish all tasks within 5 minutes, add more from the Product Backlog

Agile at XXX– a true story



Scrum	Kanban
Timeboxed interactions prescribed	Timeboxed interactions optional
Team commits to a specific amount of work for this iteration	**Commitment optional**
Uses **velocity** as default metric for planning and process improvement	Uses **lead time** as default metric for planning and process improvement
Cross–functional teams prescribed	Cross–functional teams optional, specialist teams allowed
Items broken down so they can be completed within one sprint	**No** particular **item size** is prescribed
Burndown chart prescribed	**No** particular **diagram** is prescribed
WIP limited indirectly (per sprint)	**WIP limited directly** (per per workflow state)
Estimation prescribed	Estimation optional
Cannot add items to ongoing iteration	**Can add new items** whenever capacity is available
A sprint backlog is **owned by one** specific team	A kanban board may be **shared** by multiple teams or individuals
Prescribes 3 roles (PO/SM/Team)	**Doesn’t prescribe** any **roles**
A scrum board is **reset** between each sprint	A kanban board is **persistent**
Prescribes a **prioritised** product **backlog**	**Prioritization** is **optional**

First exercise

Let’s look back

Retrospective of the Course

What good things did we do?
What could have been better?
What can we improve?