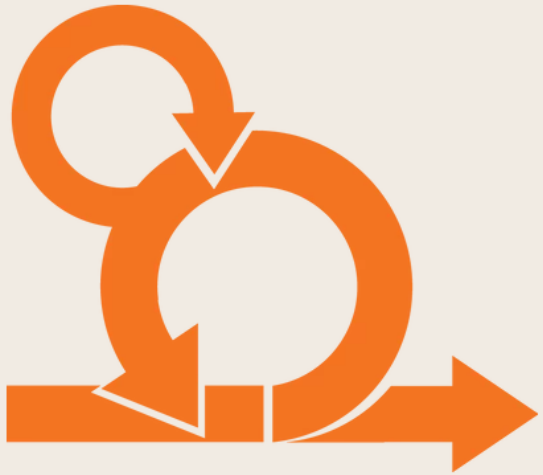


AGILE SOFTWARE DEVELOPMENT

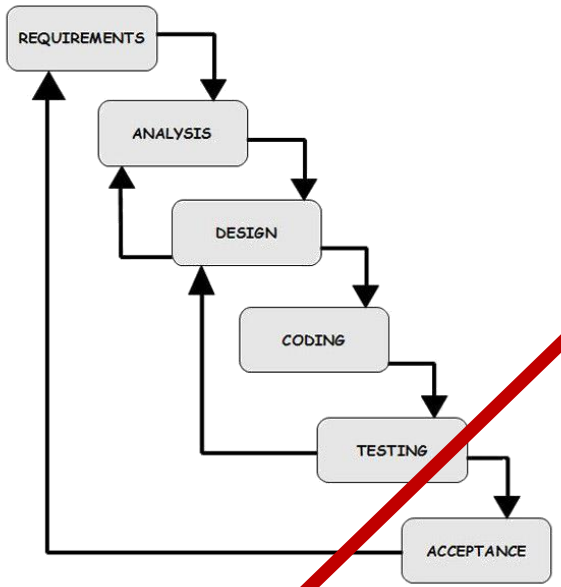
Introduction in Agile Methodologies

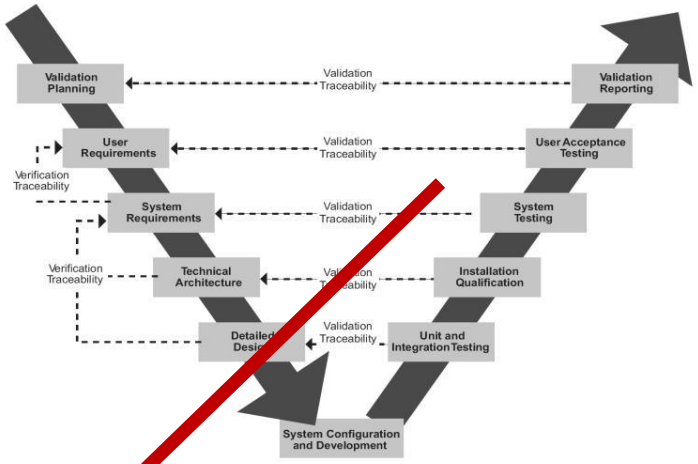
Course content

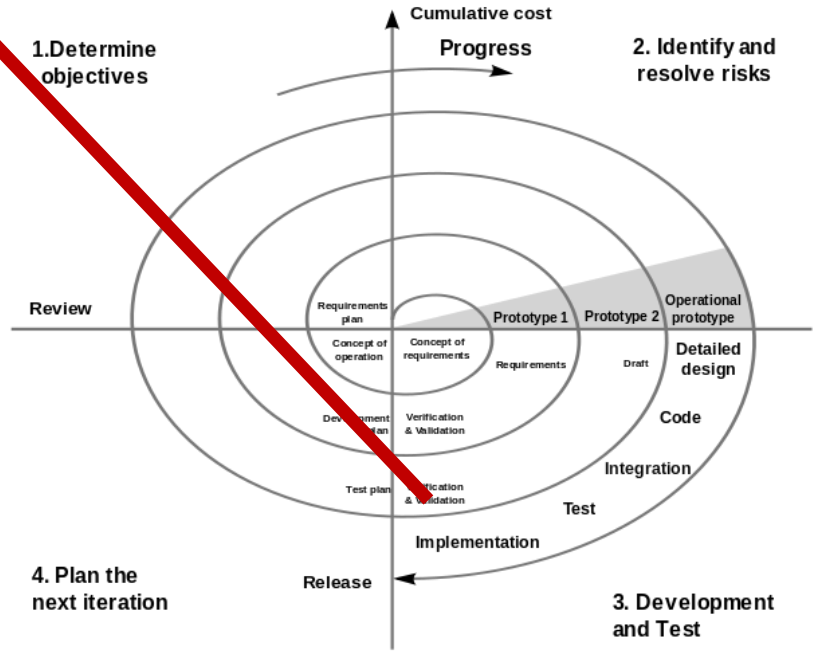
- Agile Methodologies - Overview
- Scrum
- Extreme Programming
- Kanban
- Lean Product Development
- Other Methodologies: Crystal, AUP, DAD
- The future of Agile

Seminar content

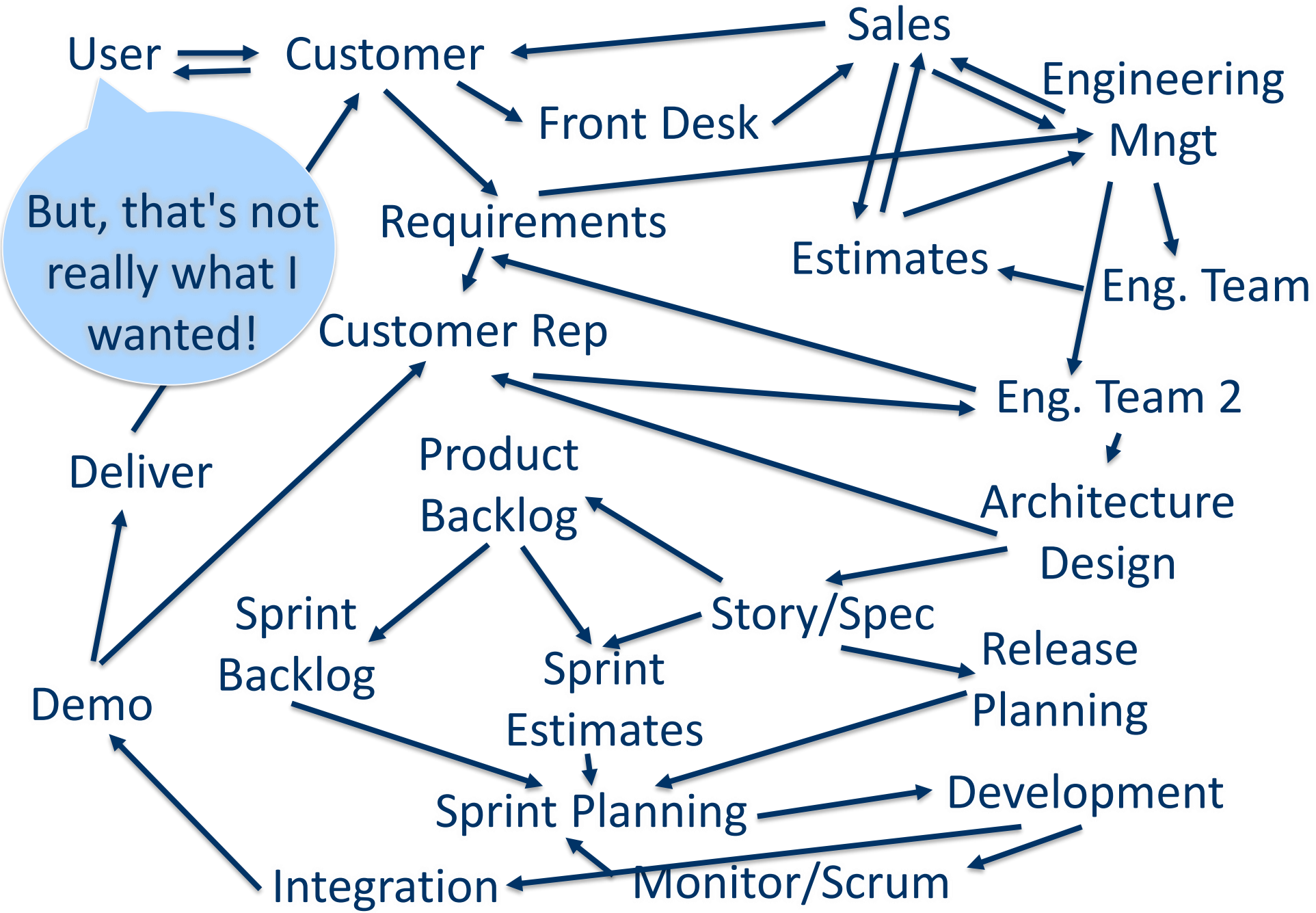
- Agile Problem Solving
- Agile Mindset
- Agile Estimation
- Team Organization
- Limiting Work in Progress


REQUIREMENTS
ANALYSIS
DESIGN
CODING
TESTING
ACCEPTANCE


Validation Planning
Validation Reporting
Validation Traceability
User Requirements
User Acceptance Testing
Verification Traceability
System Requirements
System Testing
Technical Architecture
Installation Qualification
Unit and Integration Testing
System Configuration and Development

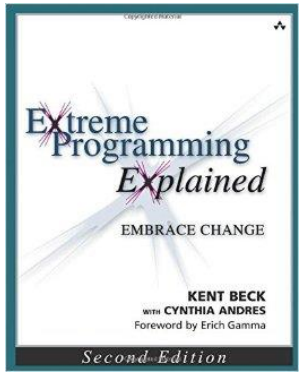

Cumulative cost
Progress
1.Determine objectives
2. Identify and resolve risks
Review
Requirements plan
Concept of operation
Concept of requirements
Prototype 1
Prototype 2
Operational prototype
Requirements
Draft
Detailed design
Verification & Validation
Code
Integration
Test
Implementation
Release
4. Plan the next iteration
3. Development and Test


agile
development
software
change
work
code
project
process
processes
team
principles
techniques
lifecycle
must
documentation
design
significant
experience
rate
used
common
transition
respond
value
clients
methods
instead
projects
also
business
per
working
people
progress
early
waterfall
system
study
iterative
releasing
changing
functional
continuous
found
collaboration
systems
environment
requirement
weeks
discourage
claim
specification
including
activities
possible
one
give
guide
writing
develop
even
showing
first
frequently
managers
less
quality
discuss
offer
need
define
malleable
larman
tools
conceptual
often
little
come
source
factors
compared
approach
amount
testing
essential
hard
adopting
throughout
product
detailed
features
impact
activity
productivity
manufacturing
resulted
serve
using
agility
integration
month
manifesto
stakeholders
specifications
client
analysis
comfort
goal
better
plan
evidence
iterations
expected
defect
valuable
difficult
behavior
knowing
adopt
shorter
teams
however
supporting
following
lines
may
structure
advantage
feedback
items
make
new
developers
defects
communication
various
trend
initial
complete
help
lies
within
many
completely
always
movement
recognize
releasing
phase
experts



Source: Peter Leeson - The Battle for Success (ITCamp 2014)

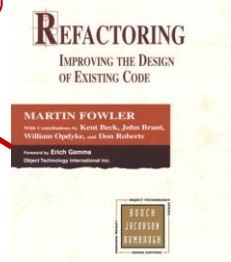
Agile Manifesto (2001)

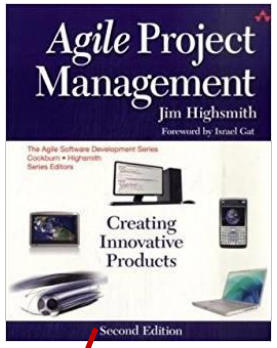


XP

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler

Crystal





James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick

Scrum

Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

Model Driven Architecture

Agile Manifesto

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*

Principle 1

Our highest priority is to satisfy the customer through **early and continuous** delivery of **valuable outcome.**

CUT THE SCOPE IN MILESTONES AND DELIVER AS SOON AS POSSIBLE

BE AWARE THAT OUTPUT IS NOT NECESSARILY OUTCOME.

Principle 2

Welcome
changing requirements,
even late in development.
Agile processes harness change for the customer's competitive advantage

CUSTOMERS DON'T KNOW WHAT THEY WANT. THAT'S OK.

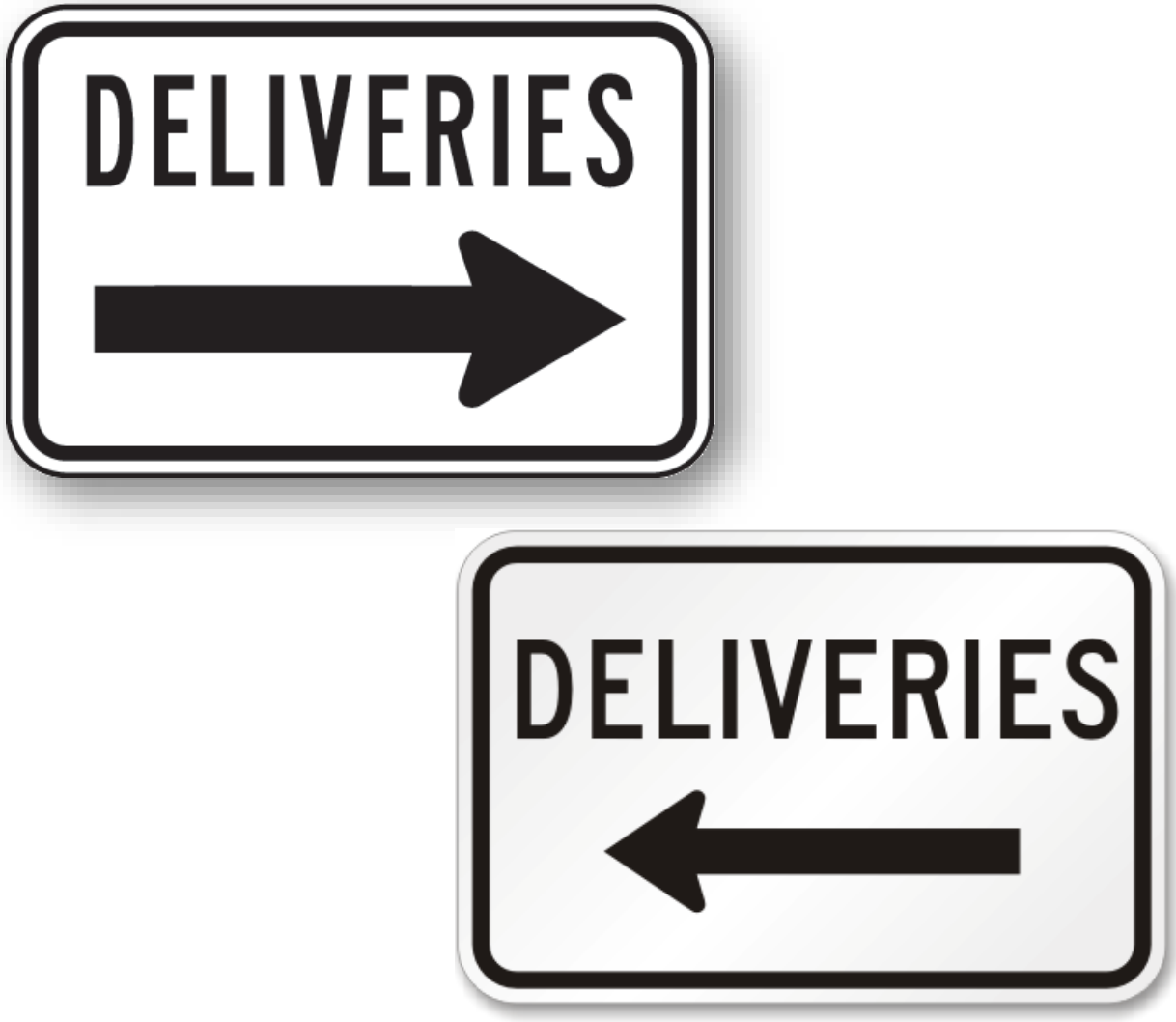
PRODUCT BACKLOG IS ALWAYS CHANGING. THAT'S ALSO OK.

Principle 3

Deliver
working versions **frequently**,
from a couple of weeks to a
couple of months,
with a preference to the
shorter timescale.

ALWAYS PAY ATTENTION TO QUALITY.

DELIVER A FUNCTIONAL VERSION AS OFTEN AS POSSIBLE.



Principle 4

Business people and developers must **work together** daily throughout the project.

KEEP STAKEHOLDERS AS CLOSE AS POSSIBLE.

WORKING TOGETHER MEANS BEING TRANSPARENT, INSPECTING AND ADAPTING CONTINUOUSLY.

Principle 5

Build projects around **motivated individuals**. Give them **the environment** and support they need and **trust them** to get the job done.

AGILE IS NOT MOTIVATING PEOPLE. LEADERS AND PROJECTS DO.

ENVIRONMENT MEANS CONTEXT, CONSTRAINTS AND OBJECTIVES.

TRUST IS ESSENTIAL TO AGILE PRACTICE.

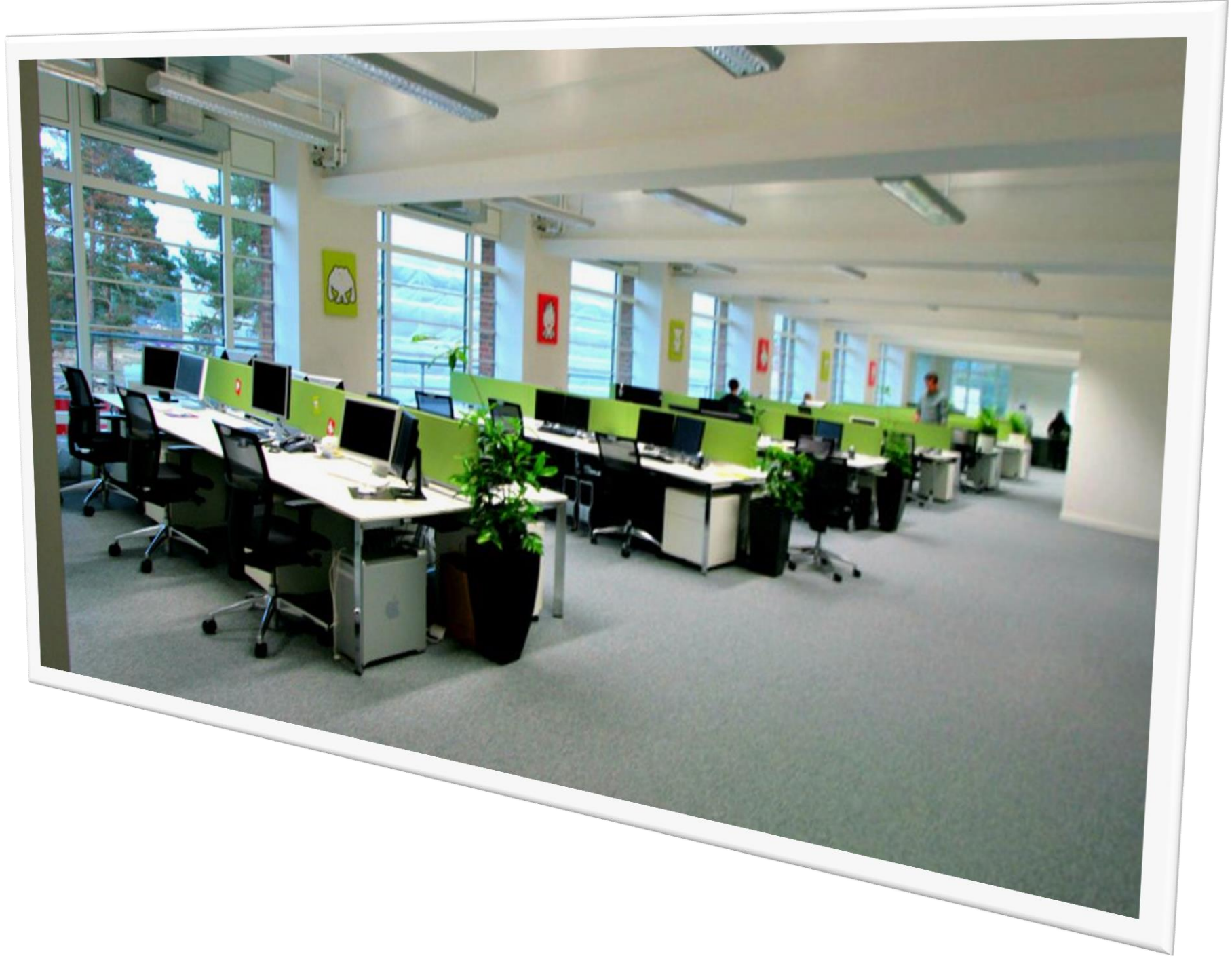
Principle 6

The most **efficient** and **effective** method of conveying information to and within a development team is **face-to-face conversation.**

EFFICIENT MEANS BEING CONCERNED ABOUT CONSUMED RESOURCES.

EFFECTIVE MEANS BEING CONCERNED ABOUT GOAL ACHIVEMENT.

INDIRECT COMMUNICATION SHOULD BE USED WITH PARSIMONY.



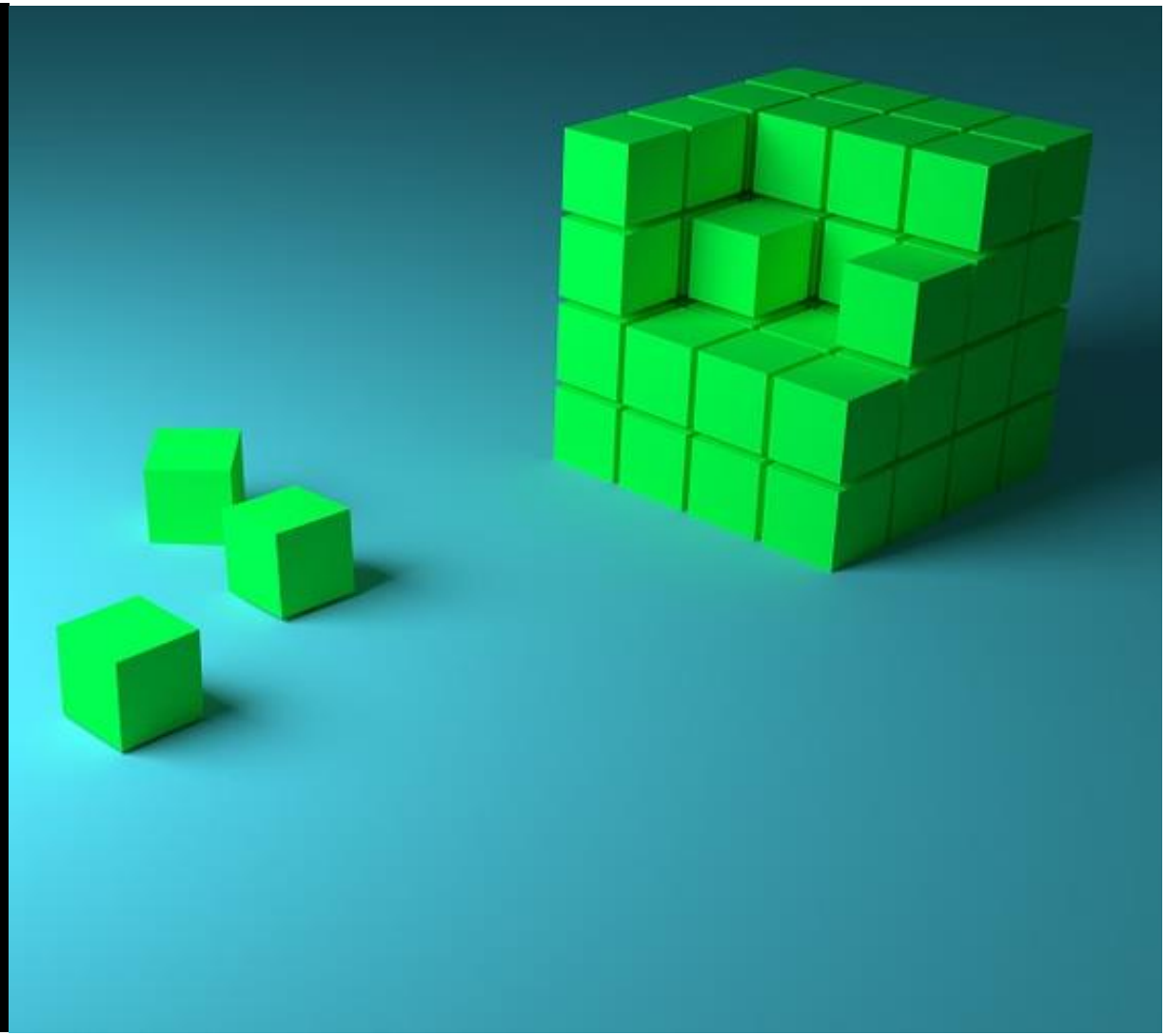
Principle 7

Working deliverables
are the primary measure of
progress.

PAY ATTENTION TO ACCEPTANCE CRITERIA.

MAKE SURE TO DEFINE EXACTLY WHAT ARE THE DELIVERABLES.

MEASURING PROGRESS IS CRUCIAL TO AGILE PROJECTS.



Principle 8

Agile processes promote **sustainable development.** The sponsors, developers and users should be able to maintain a **constant pace** indefinitely.

SUSTAINABILITY REFERS TO BUDGET, SCOPE AND EFFORT.

AGILE PROJECTS ARE MARATHONS, NOT 100m HURDLES.

Principle 9

Continuous attention
to **technical excellence** and
good design
enhances agility.

CONTINUOUS ATTENTION IMPLIES FROM THE VERY BEGINNING.

TECHNICAL EXCELLENCE IS CHOOSING THE RIGHTEST SOLUTION DEPENDING ON PROJECT'S OBJECTIVES.

Principle 10

Simplicity
– the art of maximizing the amount of work not done – is essential.

NO UNNECESSARY COMPLEXITY.

Principle 11

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

SELF-ORGANIZING MEANS COLLECTIVELY ASSUMING AND PRACTICING MANAGEMENT PRINCIPLES.

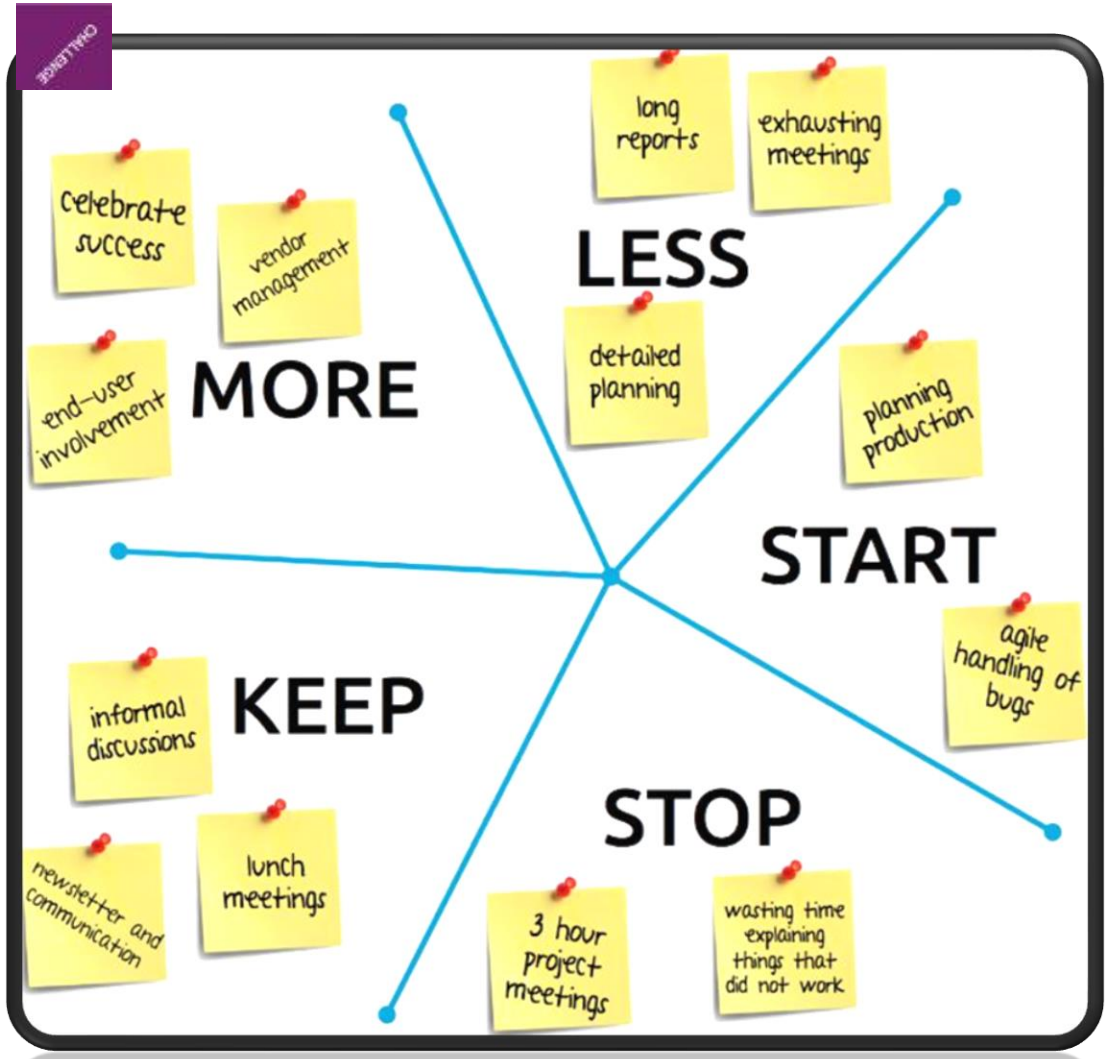
Principle 12

At **regular intervals**, the team **reflects** on how to become more effective, then **tunes and adjusts** its behavior accordingly.

NEVER GIVE UP ON RETROSPECTIVES.

REFLECTING INVOLVES A GENUINE AND HONEST CONCERN.

ADJUSTING MEANS A MEASURABLE IMPROVEMENT.



Agile manifesto

Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.

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

Deliver working deliverables frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

The most efficient and effective method of conveying information to and within a team is face-to-face conversation.

Working deliverables are the primary measure of progress.

Business people and team members must work together daily throughout the project.

Continuous attention to technical excellence and good design enhances agility.

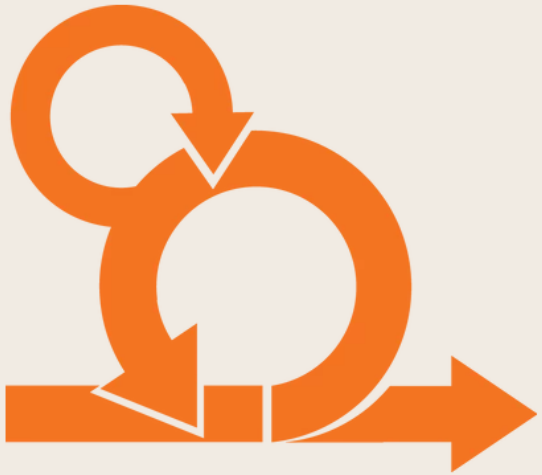
Our highest priority is to satisfy the customer through early and continuous delivery of valuable deliverables.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile processes promote sustainable work. The sponsors, team members, and users should be able to maintain a constant pace indefinitely.

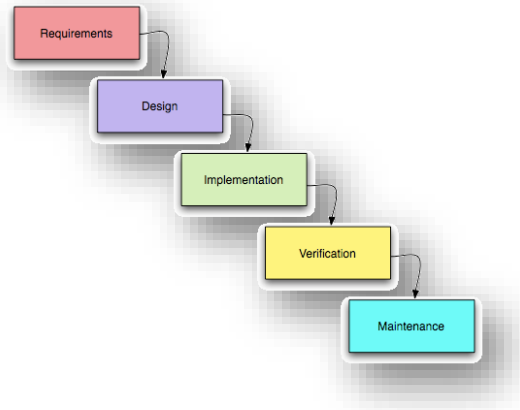
Welcome changing requirements, even late in the project. Agile processes harness change for the customer's competitive advantage.

Simplicity – the art of maximizing the amount of work not done – is essential.

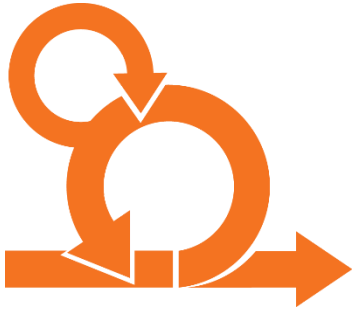


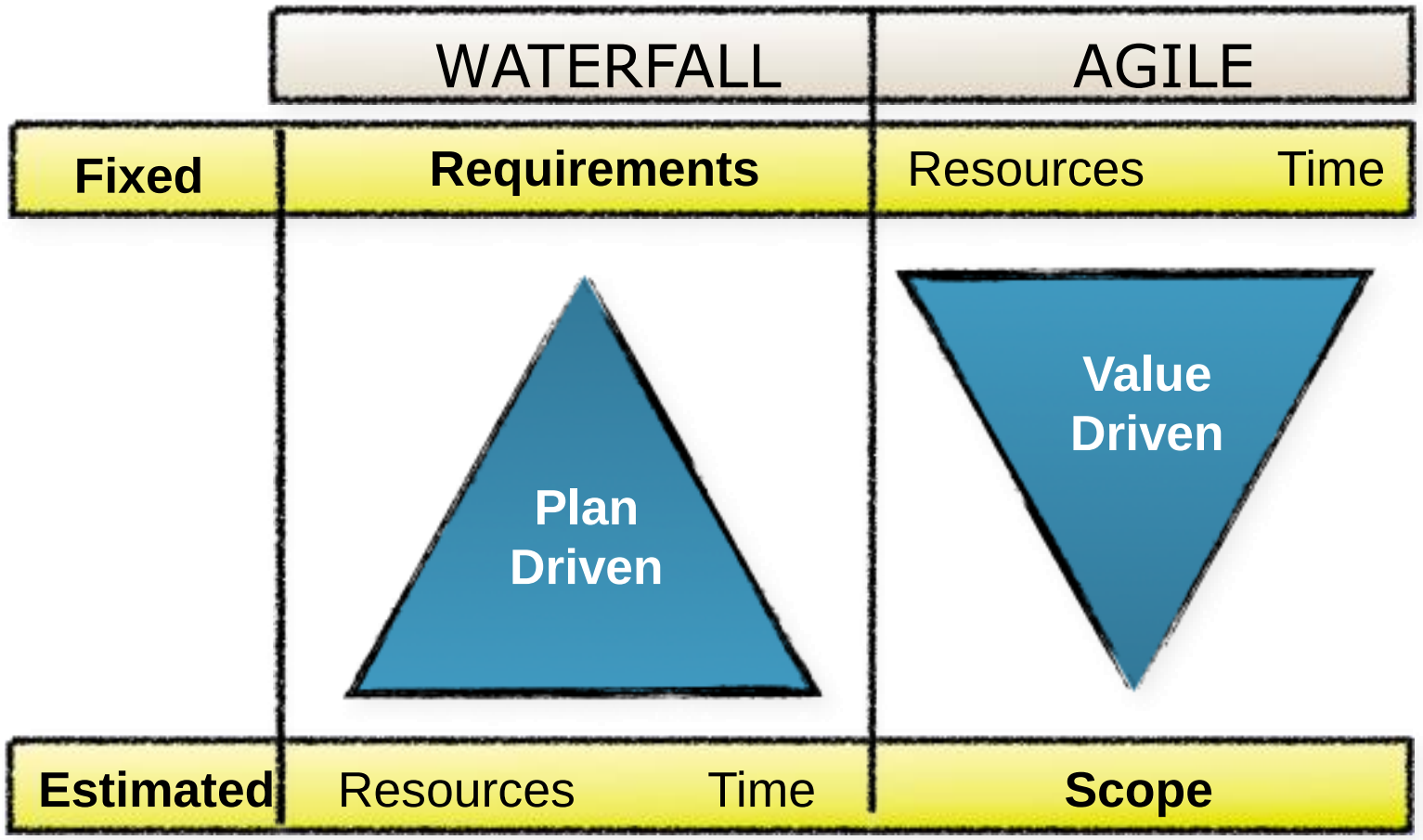
AGILE SOFTWARE DEVELOPMENT

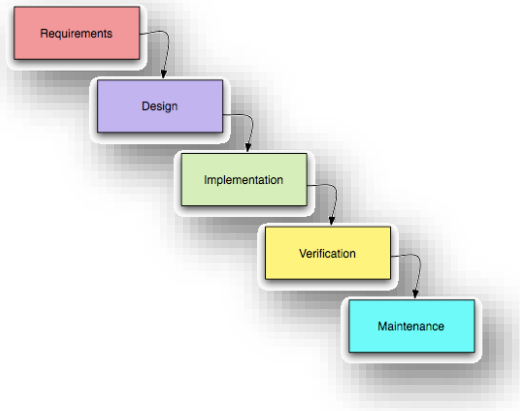
Agile
vs
Waterfall Methodologies



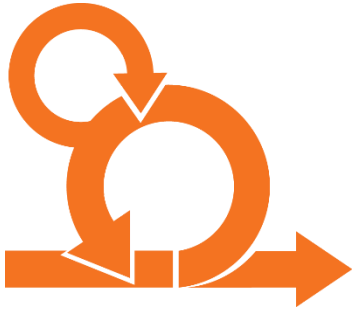
Overall picture




WATERFALL
AGILE
Fixed
Requirements
Resources
Time
Plan
Driven
Value
Driven
Estimated
Resources
Time
Scope


Requirements
Design
Implementation
Verification
Maintenance

Change

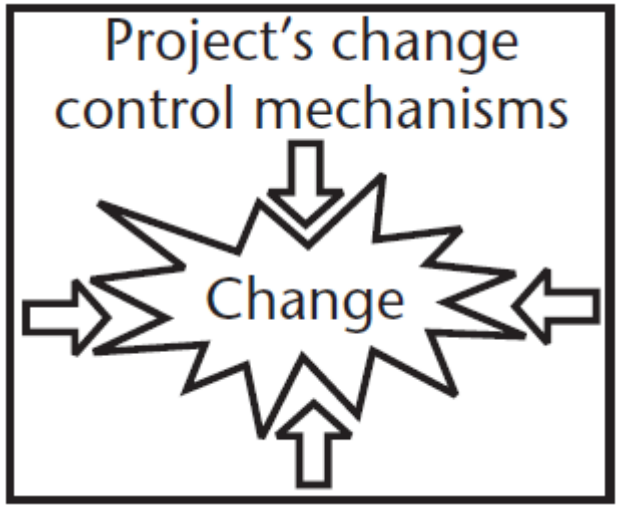



CHA
NGE

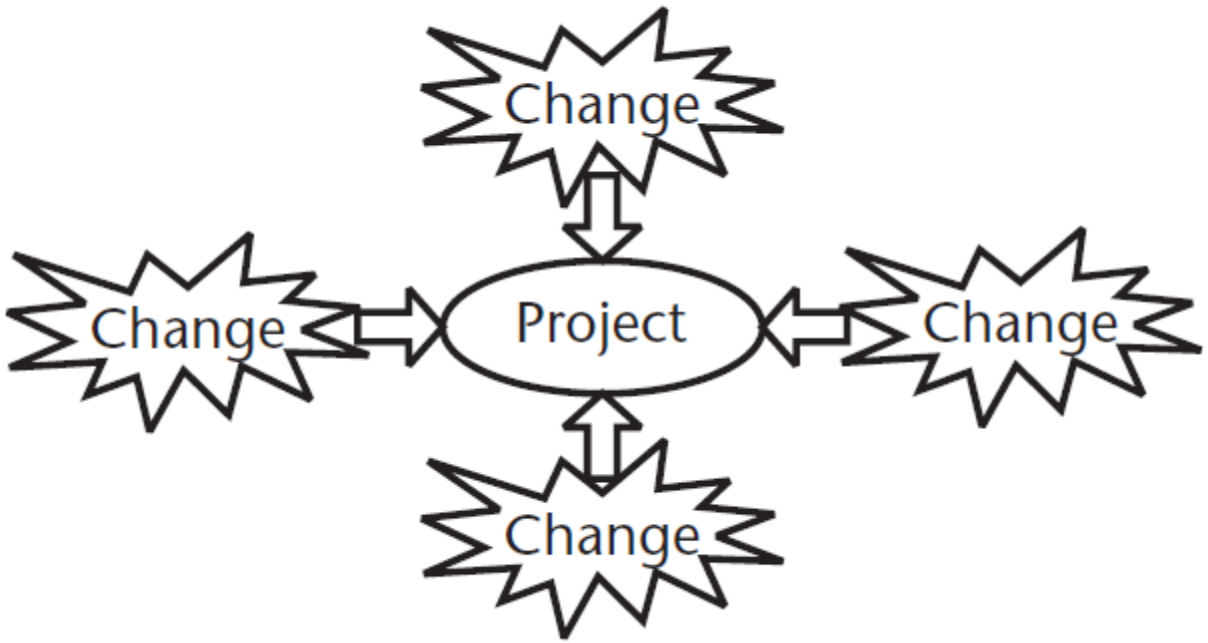
- Incomplete specifications

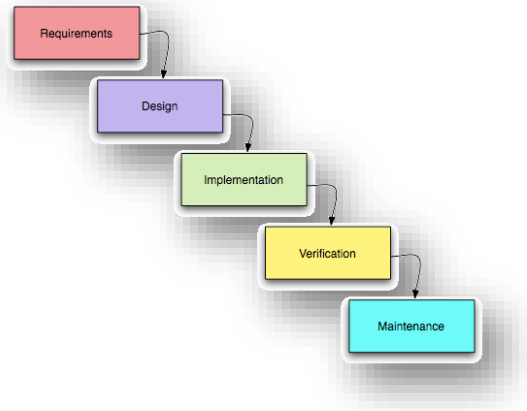
- Significant estimation errors

Controlling Change



Reacting to Change




Requirements
Design
Implementation
Verification
Maintenance

Planning

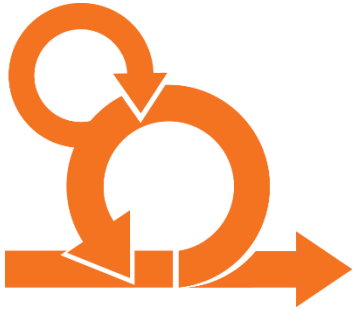
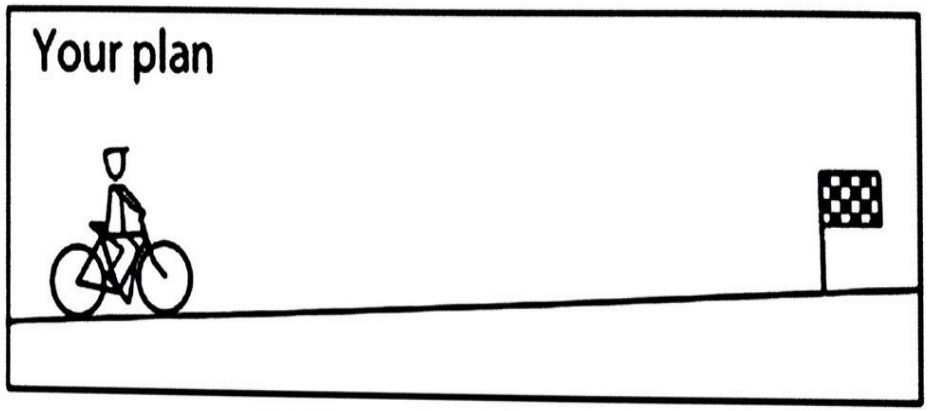
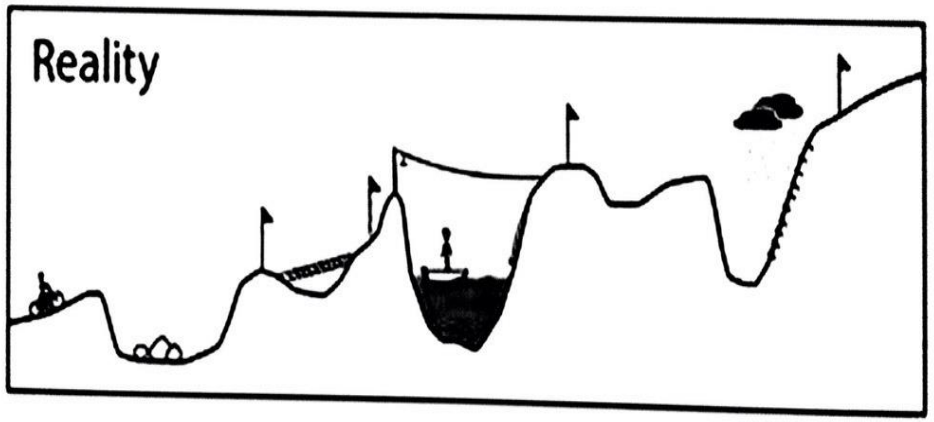
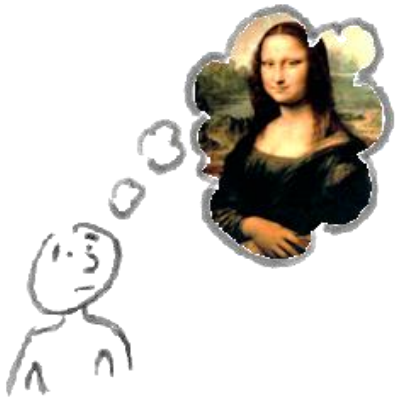


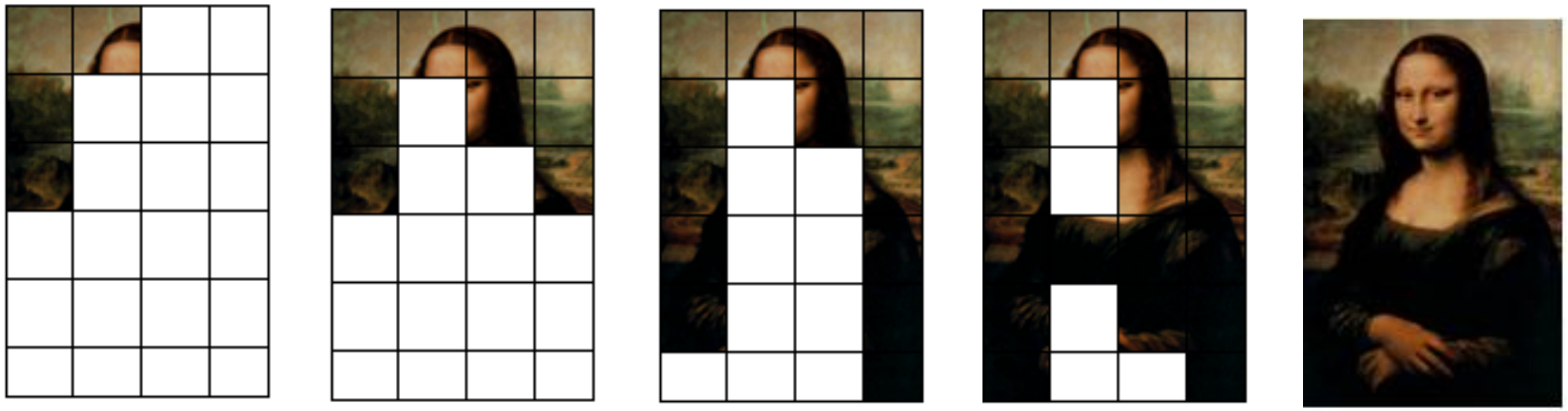

Фото с сайта images.yandex.ru


Your plan


Reality

Incremental

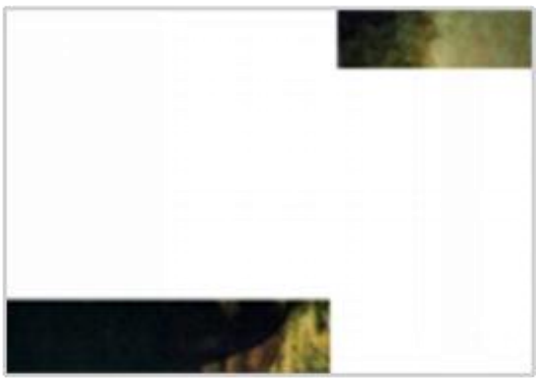
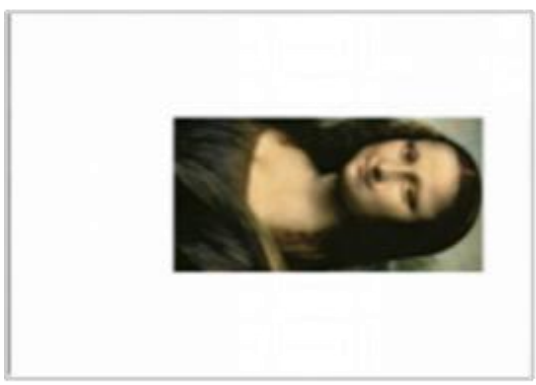
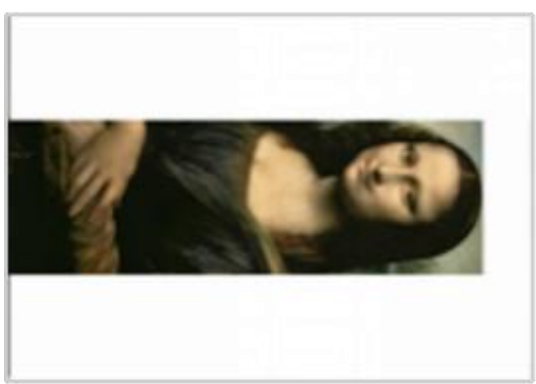
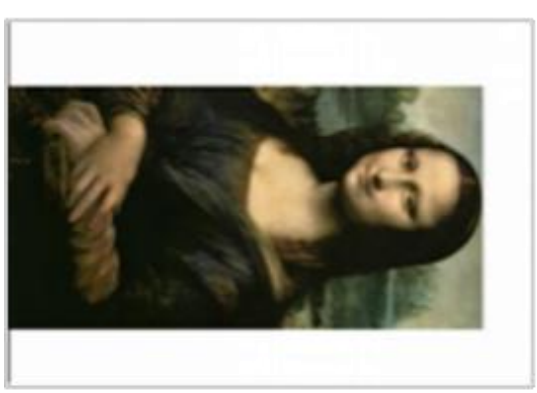




Iterative

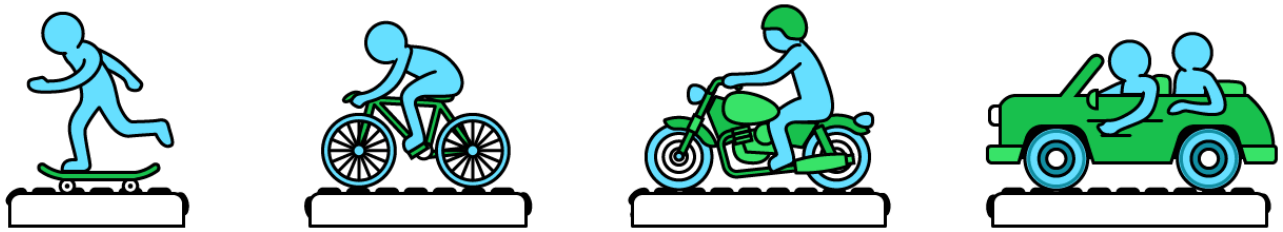
Jeff Patton: http://www.agileproductdesign.com/blog/dont_know_what_i_want.html

"Tahitians at rest" Paul Gauguin

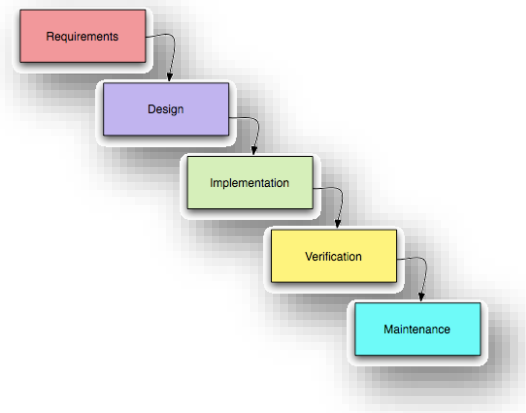
Agile approach	Traditional approach
	
	
	
	
	

Iterative & Incremental

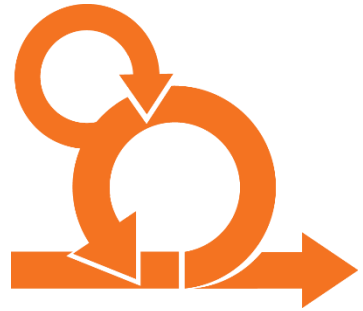
Agile



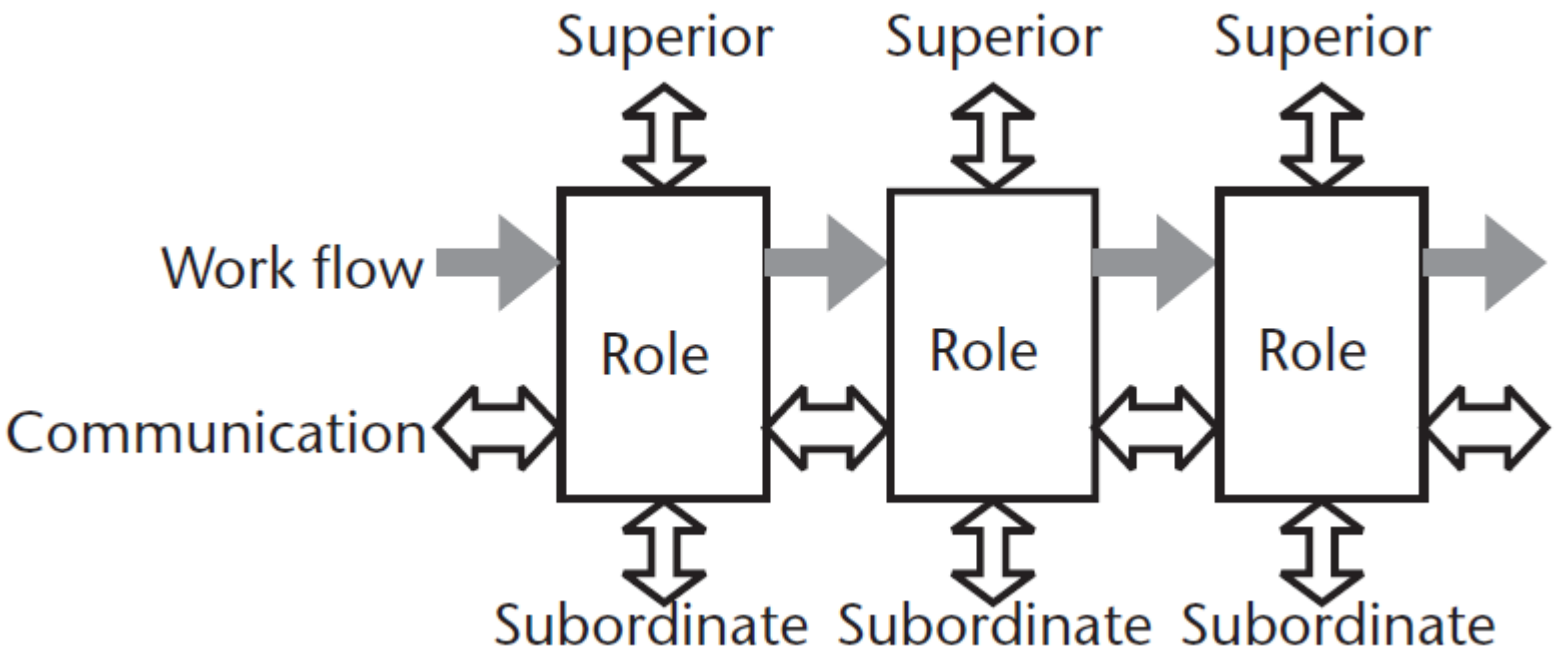
Waterfall



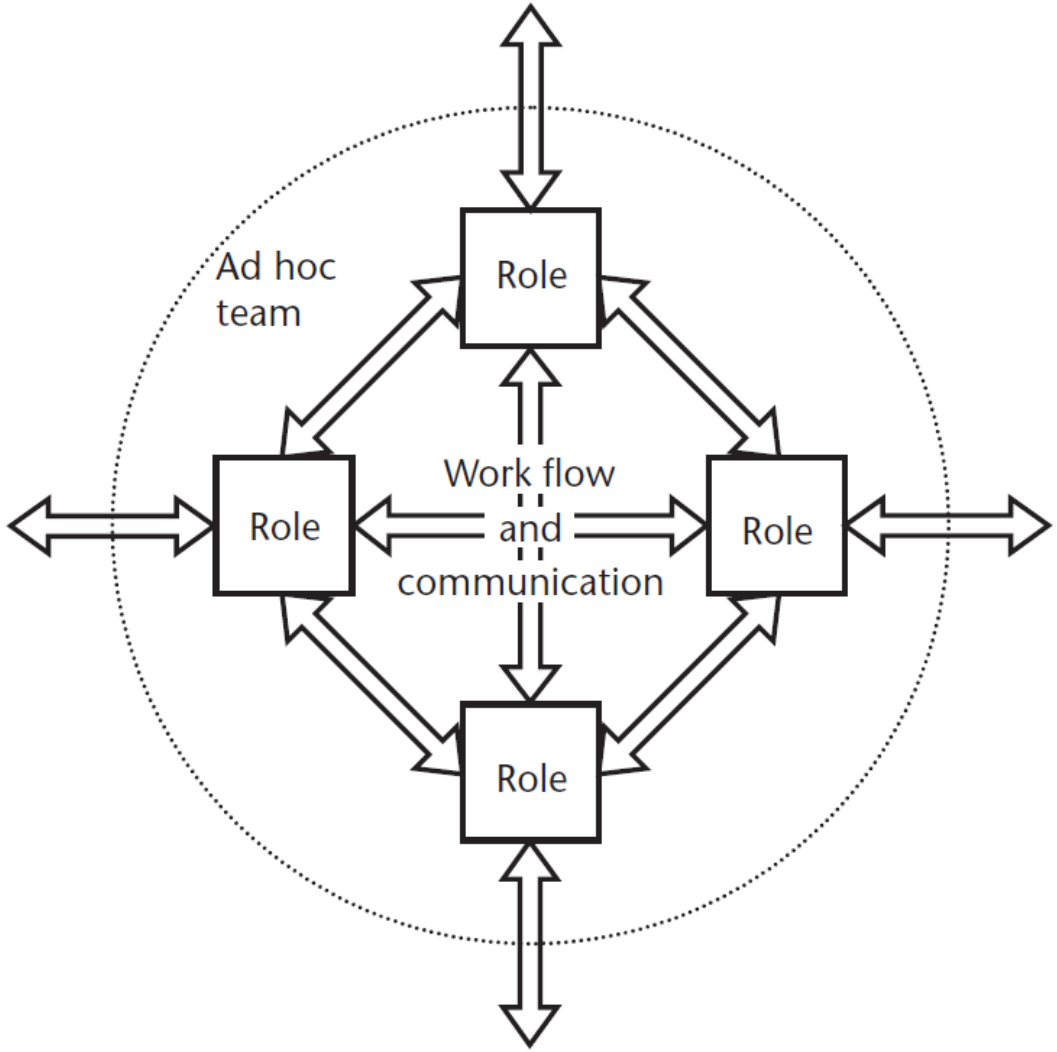
Communication

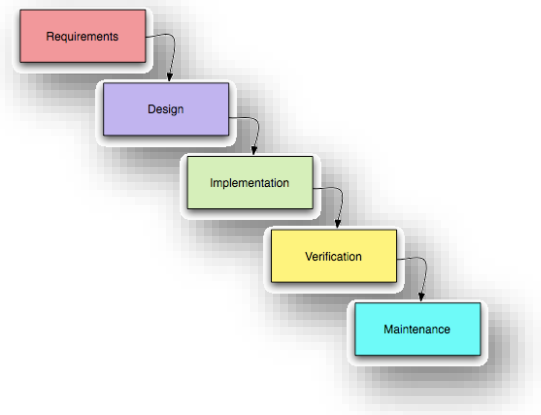


Hierarchical Organization

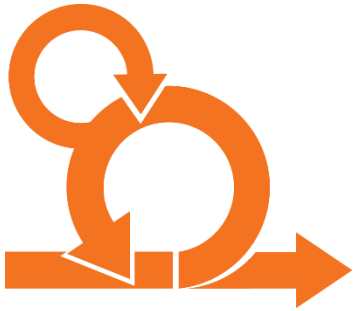


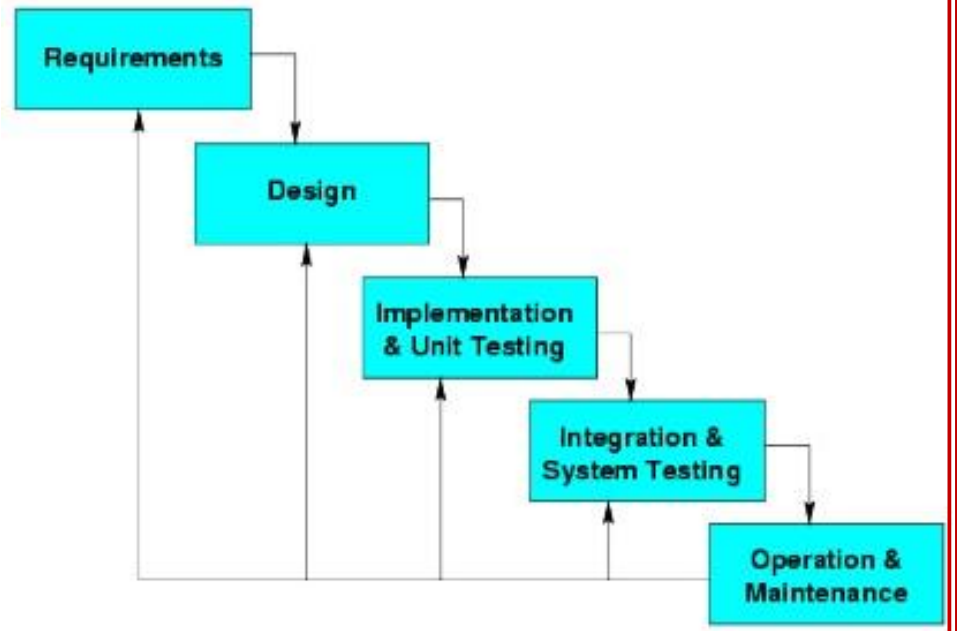
Cooperative Organization

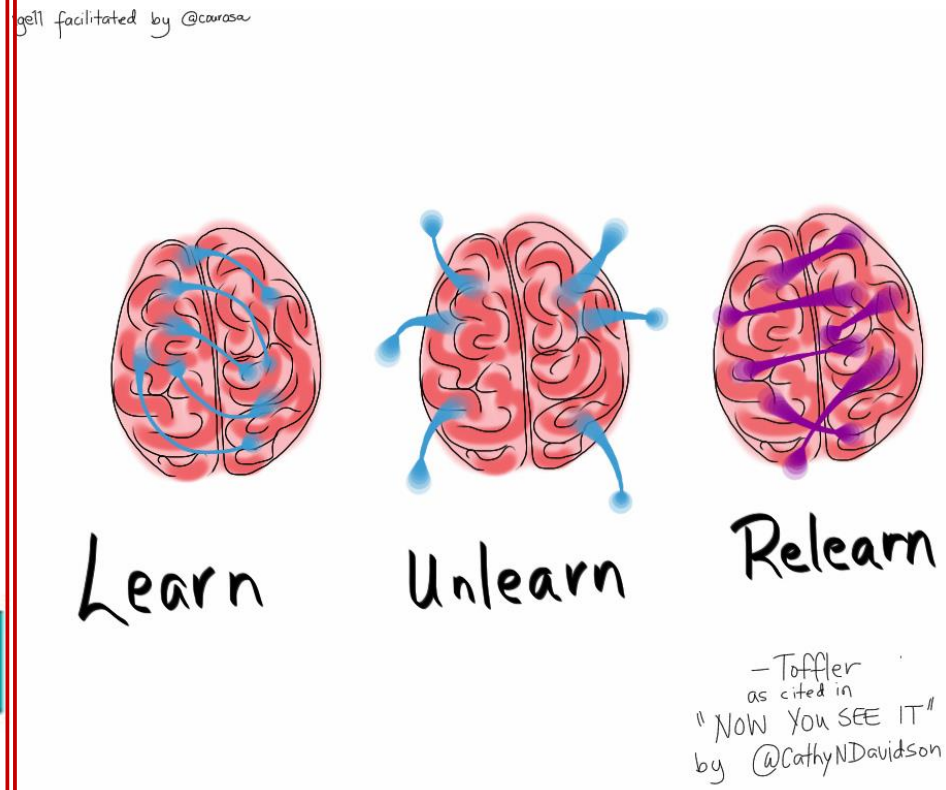



Requirements
Design
Implementation
Verification
Maintenance

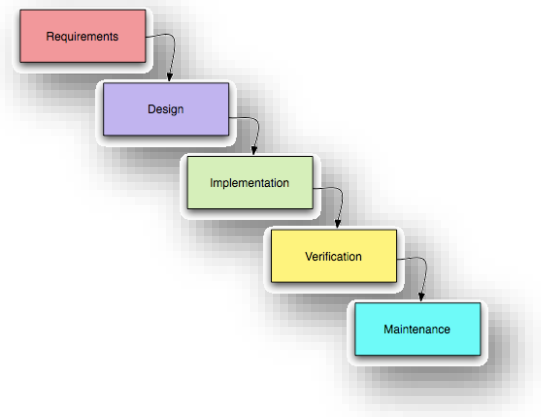
Learning



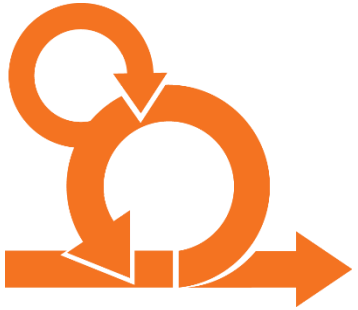

Requirements
Design
Implementation & Unit Testing
Integration & System Testing
Operation & Maintenance

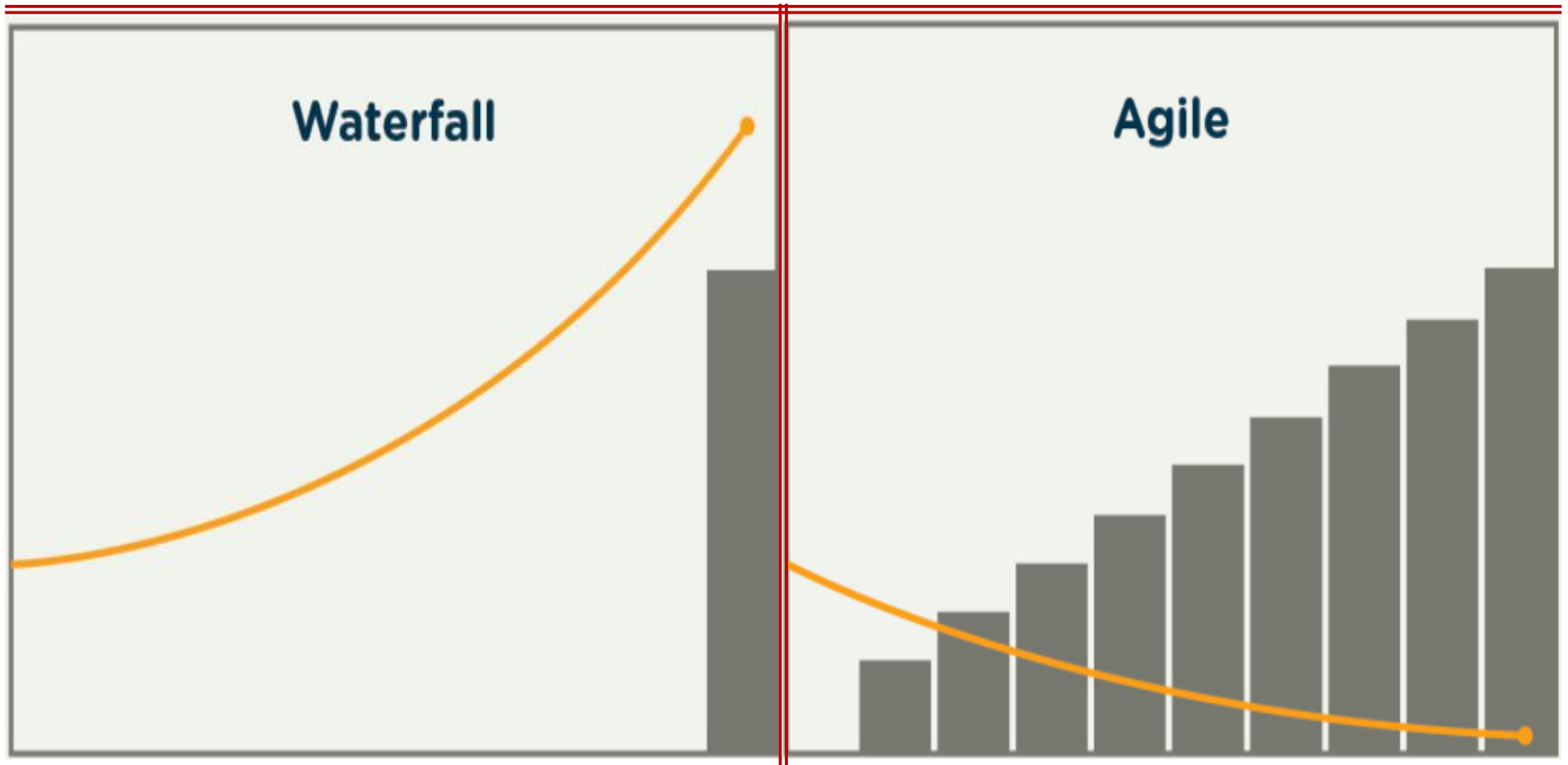

gell facilitated by @carrasa
Learn
Unlearn
Relearn
– Toffler
as cited in
"NOW YOU SEE IT"
by @CathyNDavidson

The Backwards Brain Bicycle



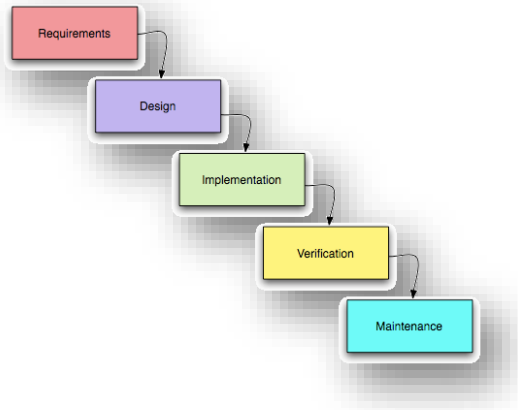
Value vs Risk



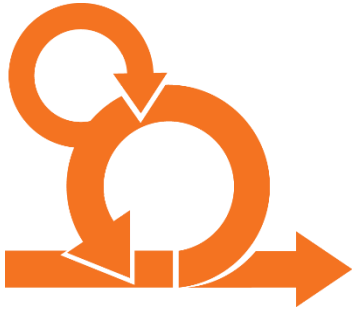


Software is cheapest in lots of SMALL cartons

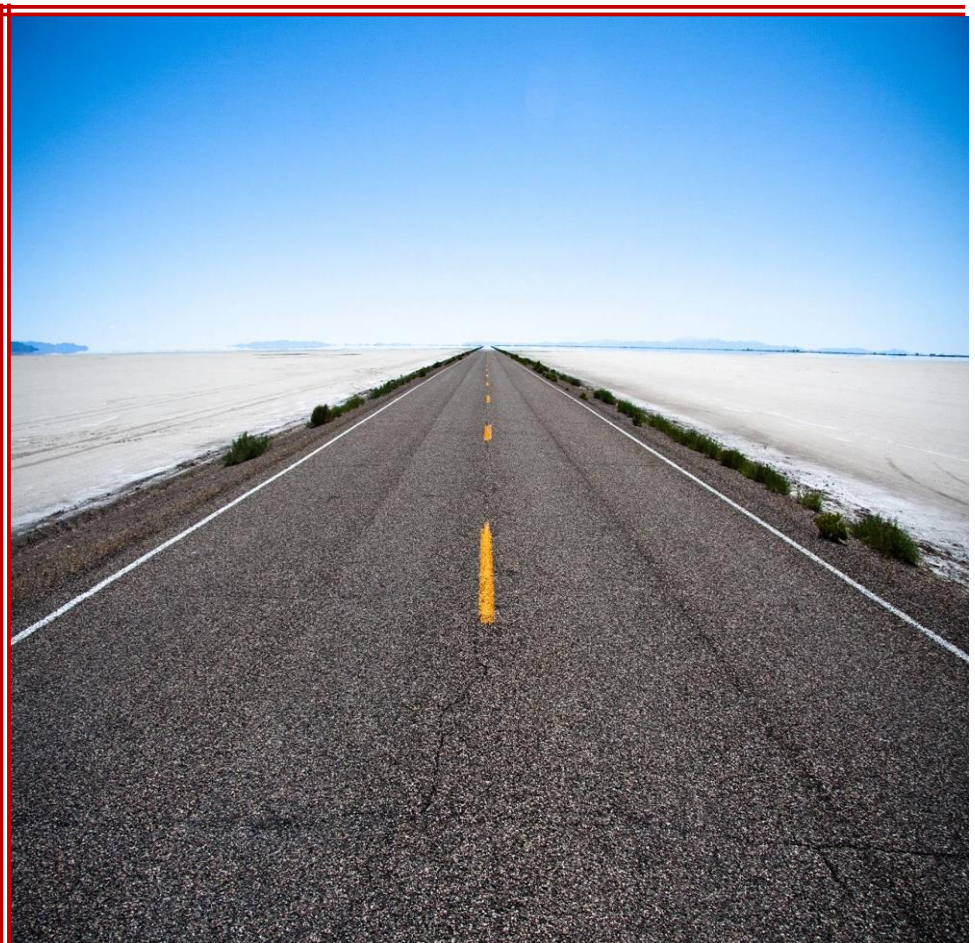
Milk is cheapest in BIG cartons


Requirements
Design
Implementation
Verification
Maintenance

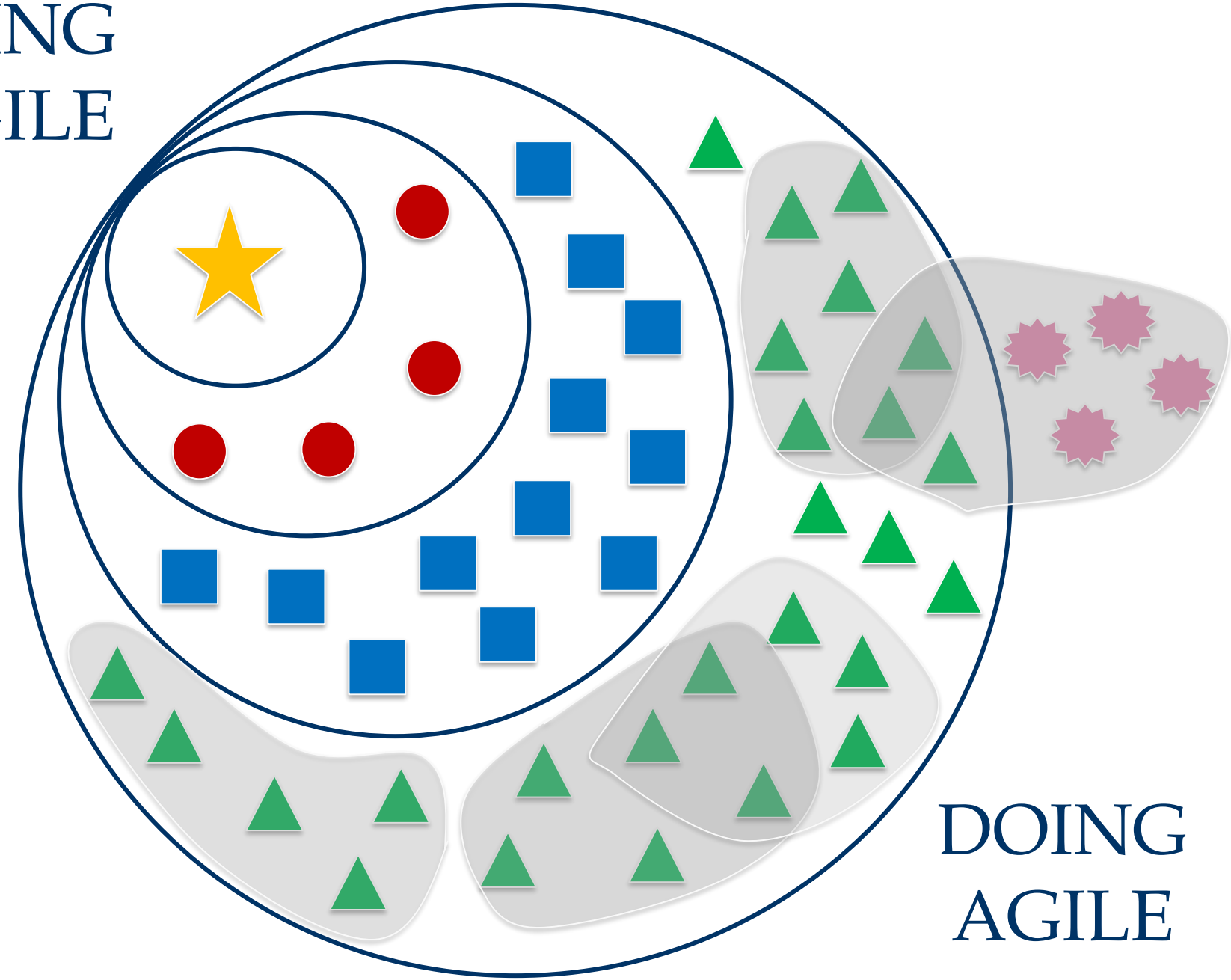
Definition



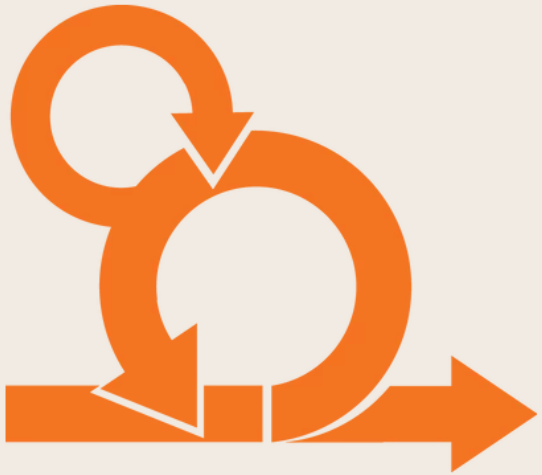

THE END



BEING AGILE



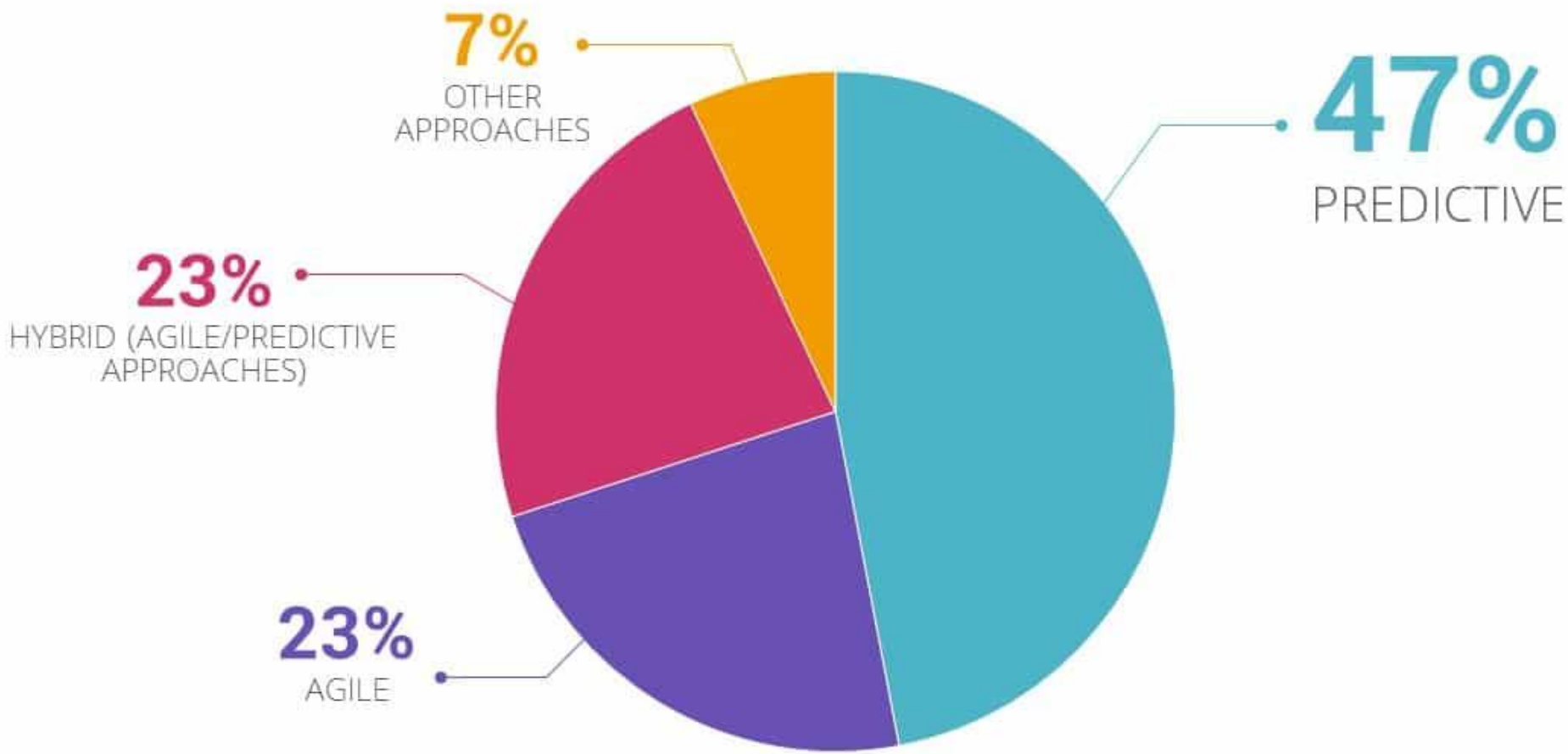
DOING AGILE



AGILE SOFTWARE DEVELOPMENT

Hybrid Approaches
Scrum Methodology. Roles

DEVELOPMENT APPROACH USED



Source: PMI 2018 Pulse of The Profession

WWW.VITALITYCHICAGO.COM

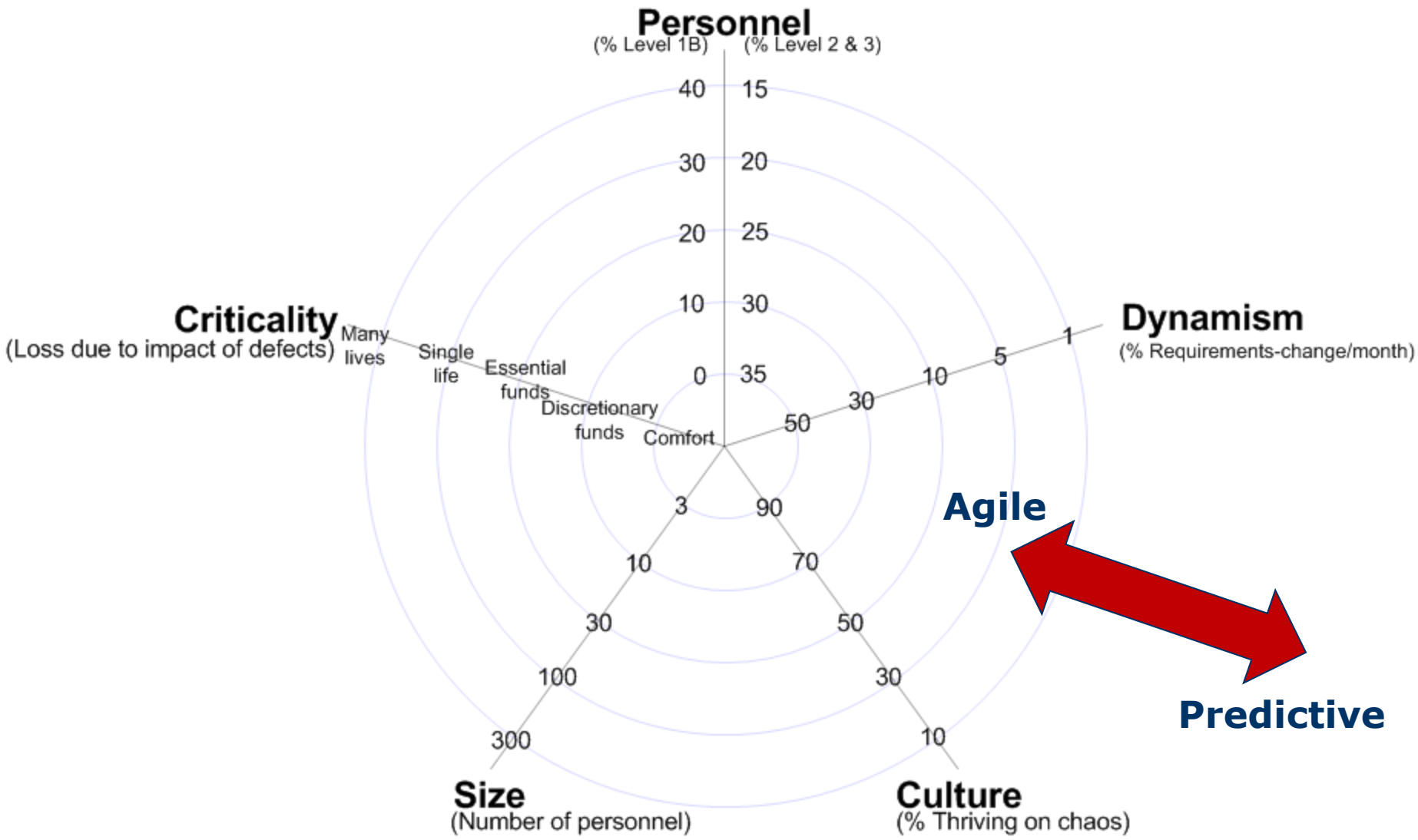
Levels of Software Method Understanding and Use

Level	Characteristics
3	Able to revise a method (break its rules) to fit an unprecedented new situation
2	Able to tailor a method to fit a precedented new situation
1A	With training, able to perform discretionary method steps (e.g., sizing stories to fit increments, composing patterns, compound refactoring, complex COTS integration). With experience can become Level 2.
1B	With training, able to perform procedural method steps (e.g. coding a simple method, simple refactoring, following coding standards and CM procedures, running tests). With experience can master some Level 1A skills.
-1	May have technical skills, but unable or unwilling to collaborate or follow shared methods.

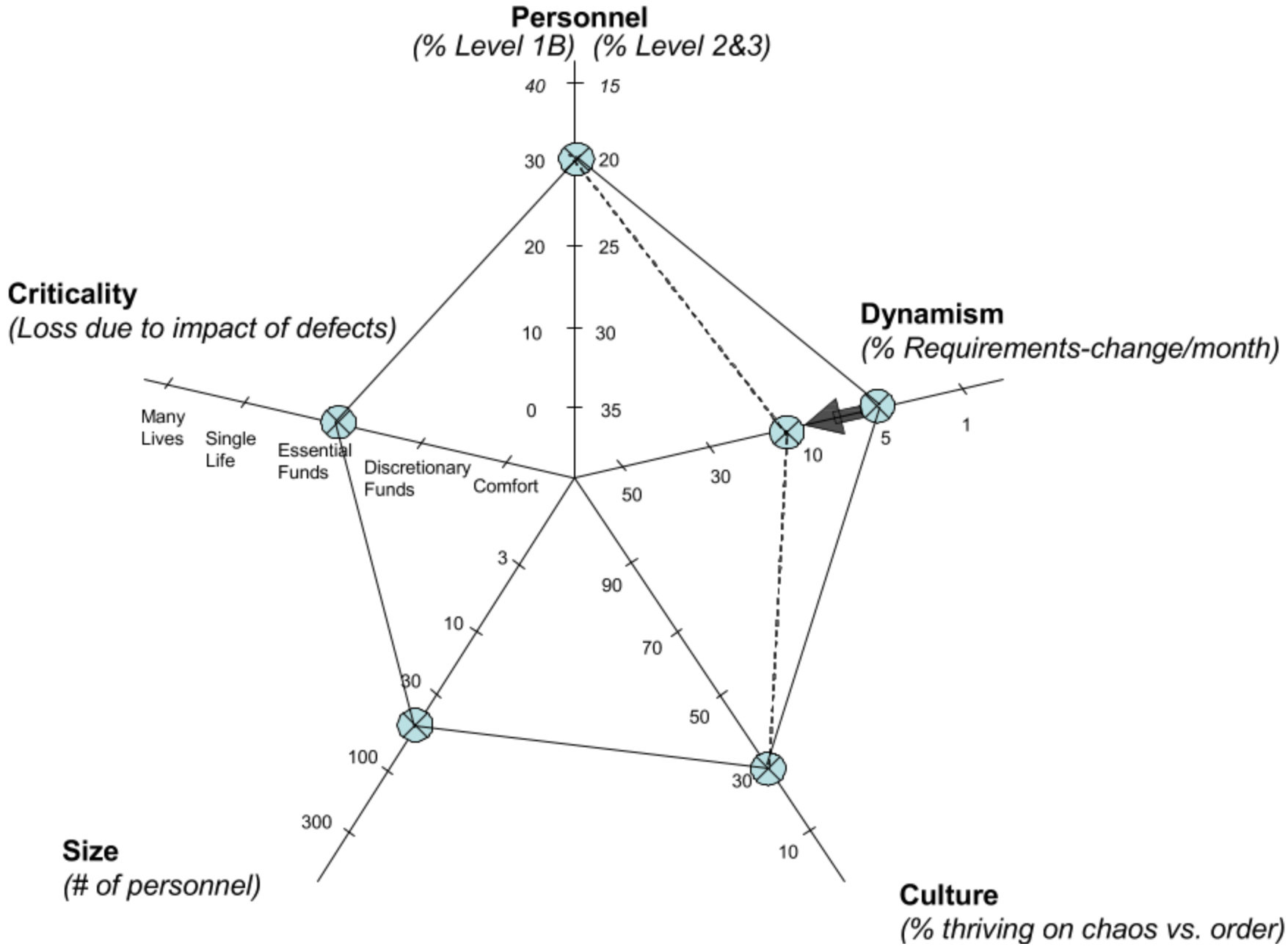
From "Agile Software Development", A. Cockburn (Addison-Wesley, 2002)

Factor	Agility Considerations	Discipline Considerations
Size	Well-matched to small products and teams. Reliance on tacit knowledge limits scalability.	Methods evolved to handle large products and teams. Hard to tailor down to small projects.
Criticality	Untested on safety-critical products. Potential difficultiies with simple design and lack of documentation.	Methods evolved to handle highly critical products. Hard to tailor down to low-criticality products.
Dynamism	Simple design and continuous refactoring are excellent for highly dynamic environments, but a source of potentially expensive rework for highly stable environments.	Detailed plans and Big Design Up Front excellent for highly stable environment, but a source of expensive rework for highly dynamic environments.
Personnel	Requires continuous presence of a critical mass of scarce Cockburn Level 2 or 3 experts. Risky to use non-agile Level 1B people.	Needs a critical mass of scarce Cockburn Level 2 and 3 experts during project definition, but can work with fewer later in the project—unless the environment is highly dynamic. Can usually accommodate some Level 1B people.
Culture	Thrives in a culture where people feel comfortable and empowered by having many degrees of freedom.	Thrives in a culture where people feel comfortable and empowered by having their roles defined by clear policies and procedures.

from "Balancing Agility and Discipline", Barry Boehm & Richard Turner



from "Balancing Agility and Discipline", Barry Boehm & Richard Turner

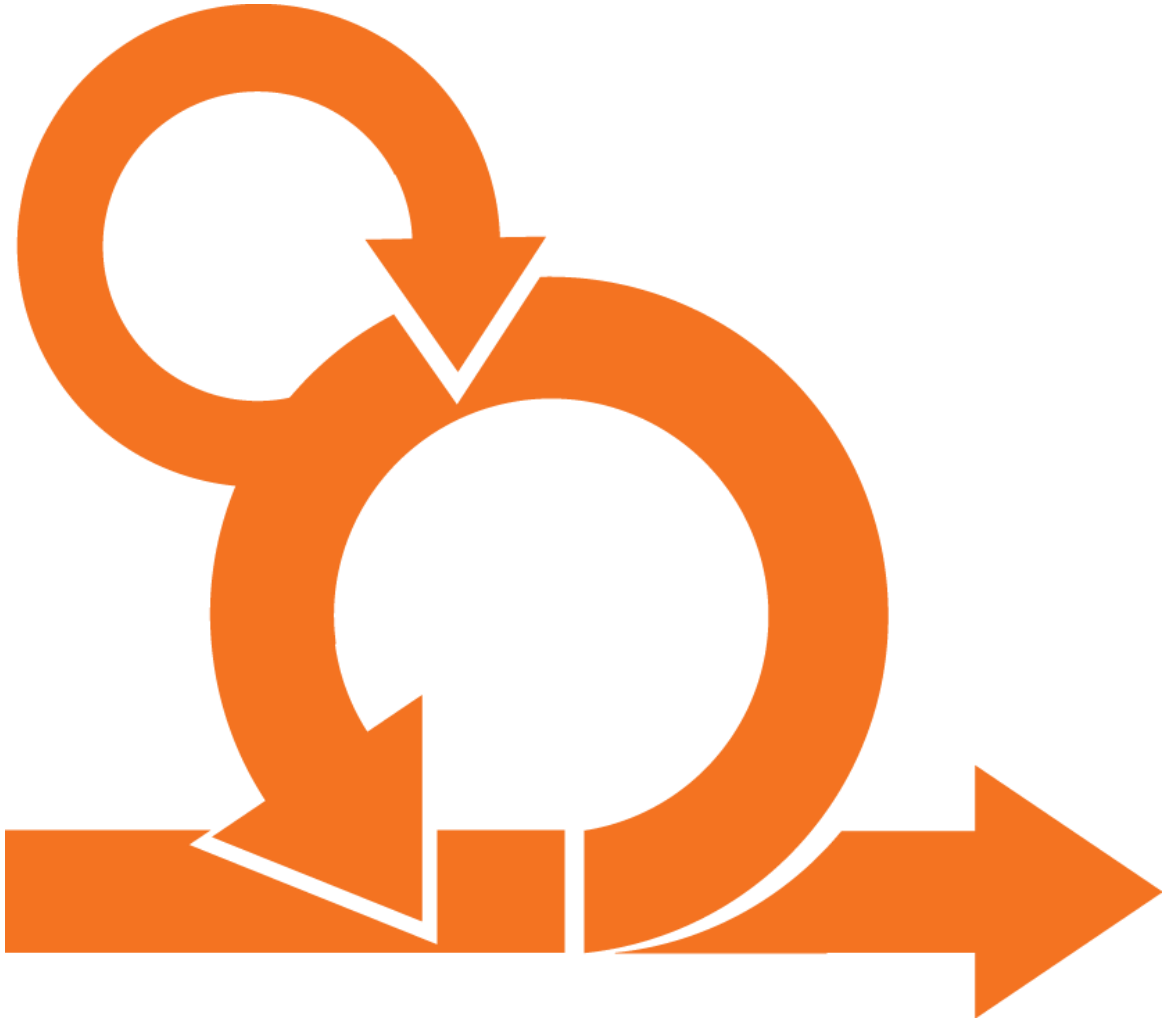

Personnel
(% Level 1B) (% Level 2&3)
40
15
30
20
20
25
10
30
0
35
Criticality
(Loss due to impact of defects)
Many Lives
Single Life
Essential Funds
Discretionary Funds
Comfort
Dynamism
(% Requirements-change/month)
1
5
10
30
50
3
10
30
100
300
90
70
50
30
10
Size
(# of personnel)
Culture
(% thriving on chaos vs. order)

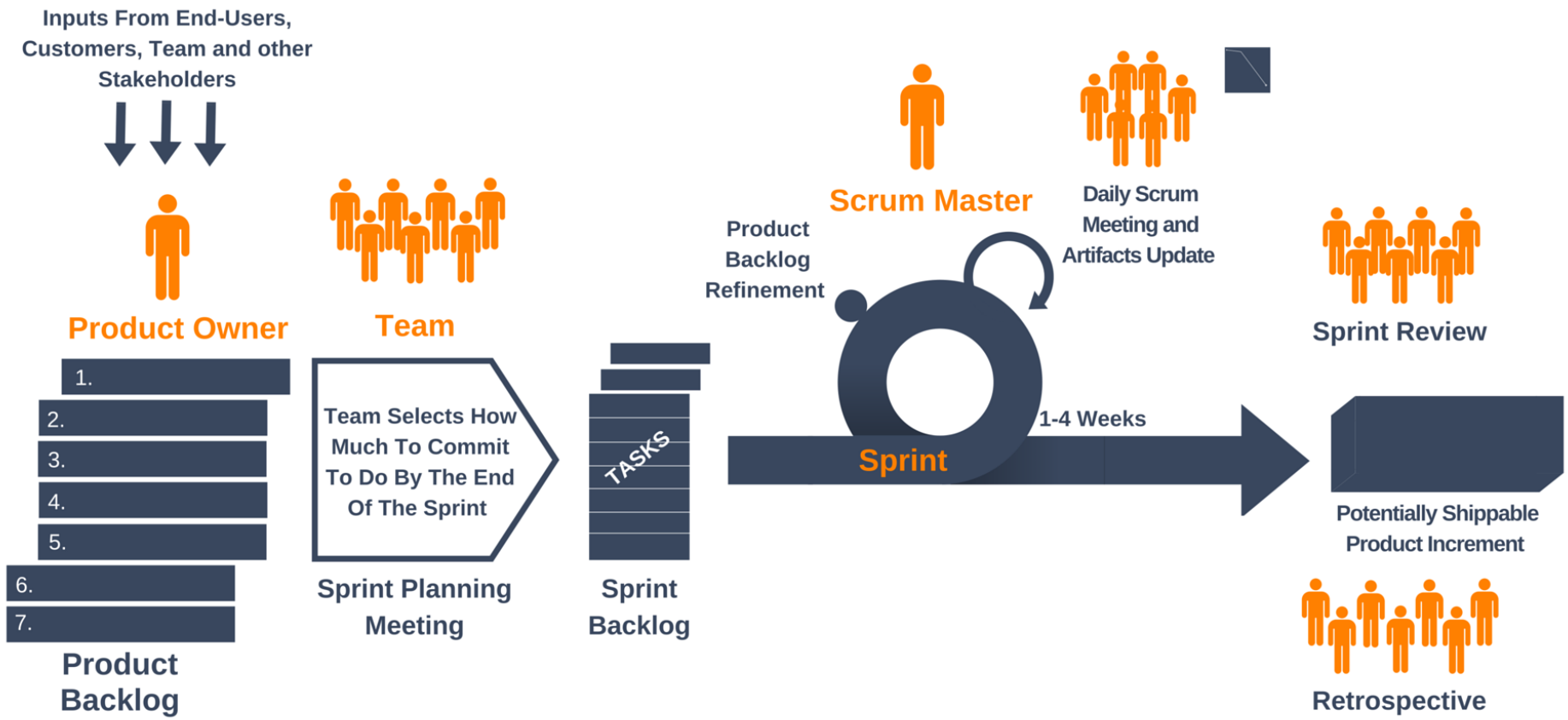
Christopher Avery

"We can never direct a living system, only disturb it and wait to see the response…

We can't know all the forces shaping an organization we wish to change, so all we can do is provoke the system in some way be experimenting with a force we think might have some impact, then watch to see what happens."

SCRUM



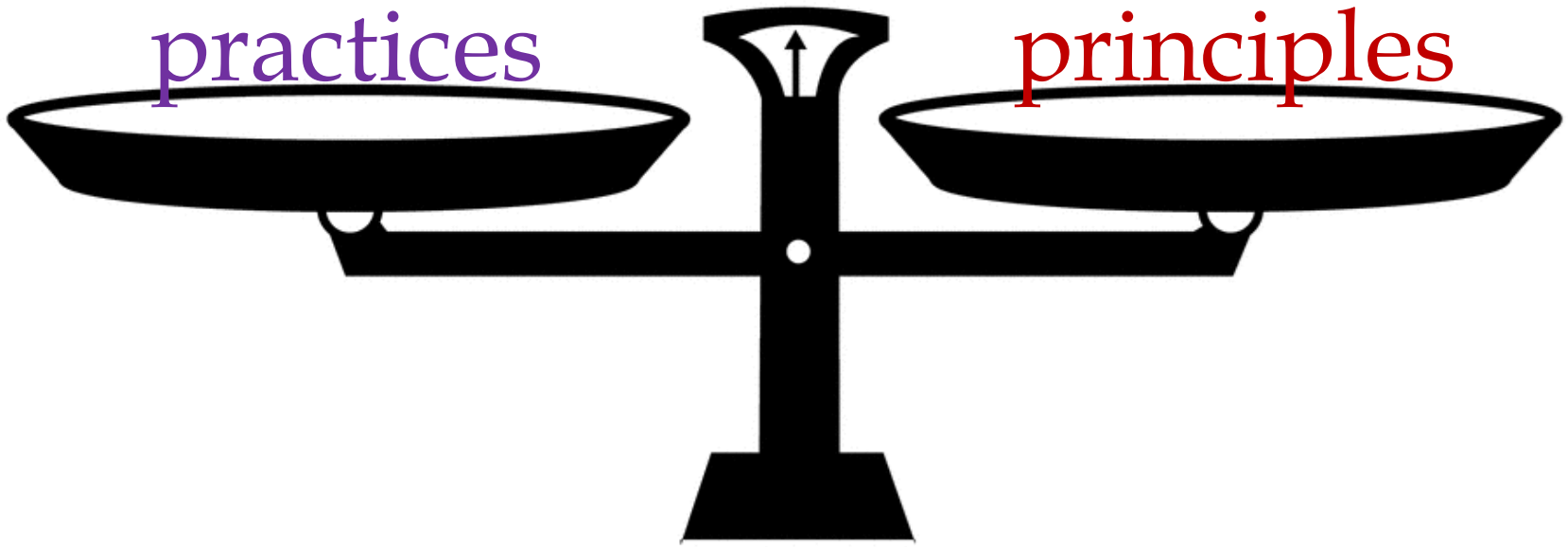

Inputs From End-Users, Customers, Team and other Stakeholders
Product Owner
Team
Scrum Master
Daily Scrum Meeting and Artifacts Update
Product Backlog Refinement
Sprint Review
1.
2.
3.
4.
5.
6.
7.
Team Selects How Much To Commit To Do By The End Of The Sprint
TASKS
1-4 Weeks
Sprint
Potentially Shippable Product Increment
Product Backlog
Sprint Planning Meeting
Sprint Backlog
Retrospective

SCRUM

…. hits an ideal balance between

concrete practices

abstract principles



SCRUM

concrete practices

abstract principles

SCRUM

learn fast?

is a **lightweight** framework designed to help

7 ± 2

small, close-knit teams of people

develop **complex** products.

remember
Cynefin framework

Scrum Pillars

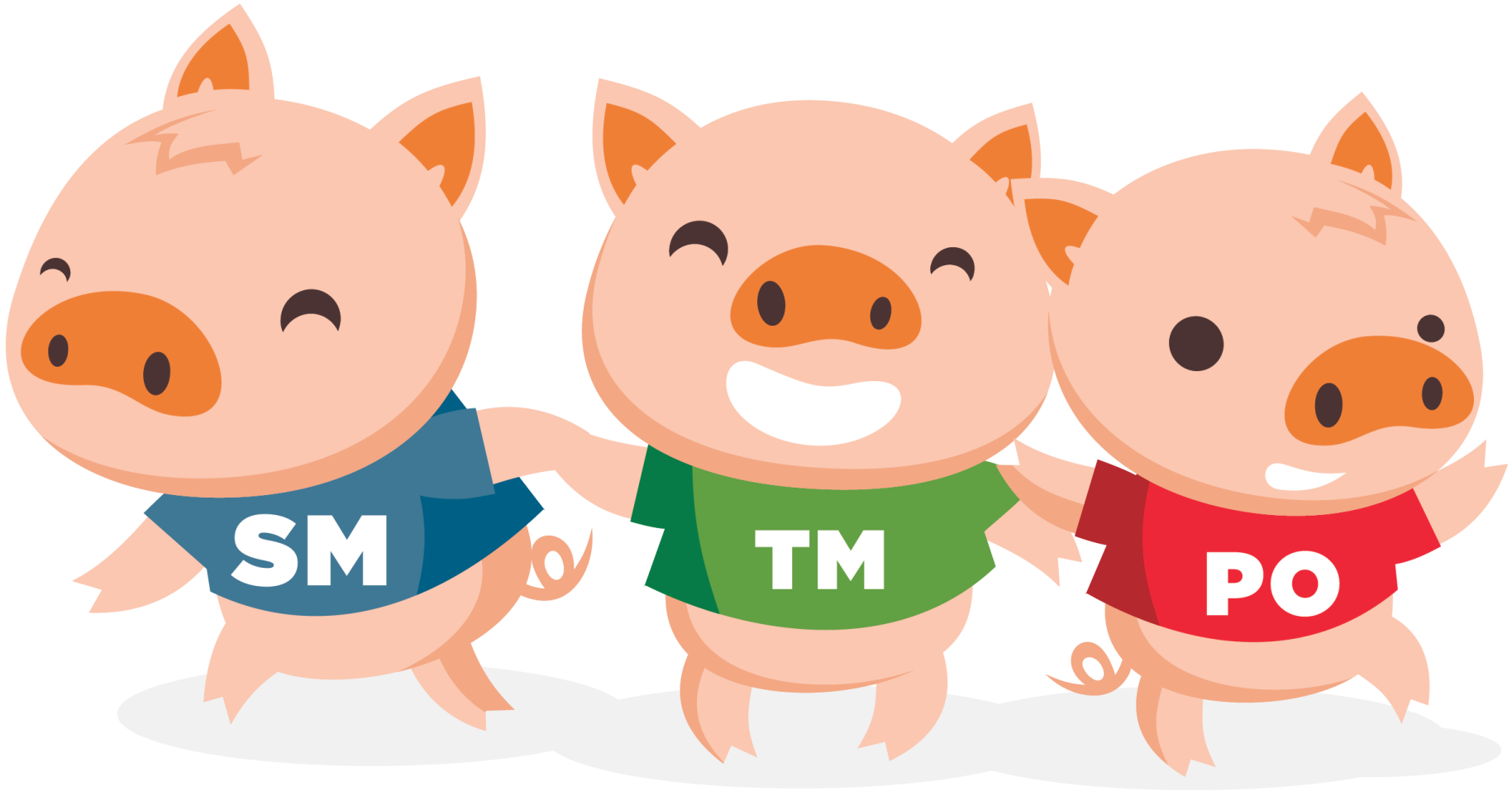
The pillars of Scrum

- Transparency
- Inspection
- Adaptation

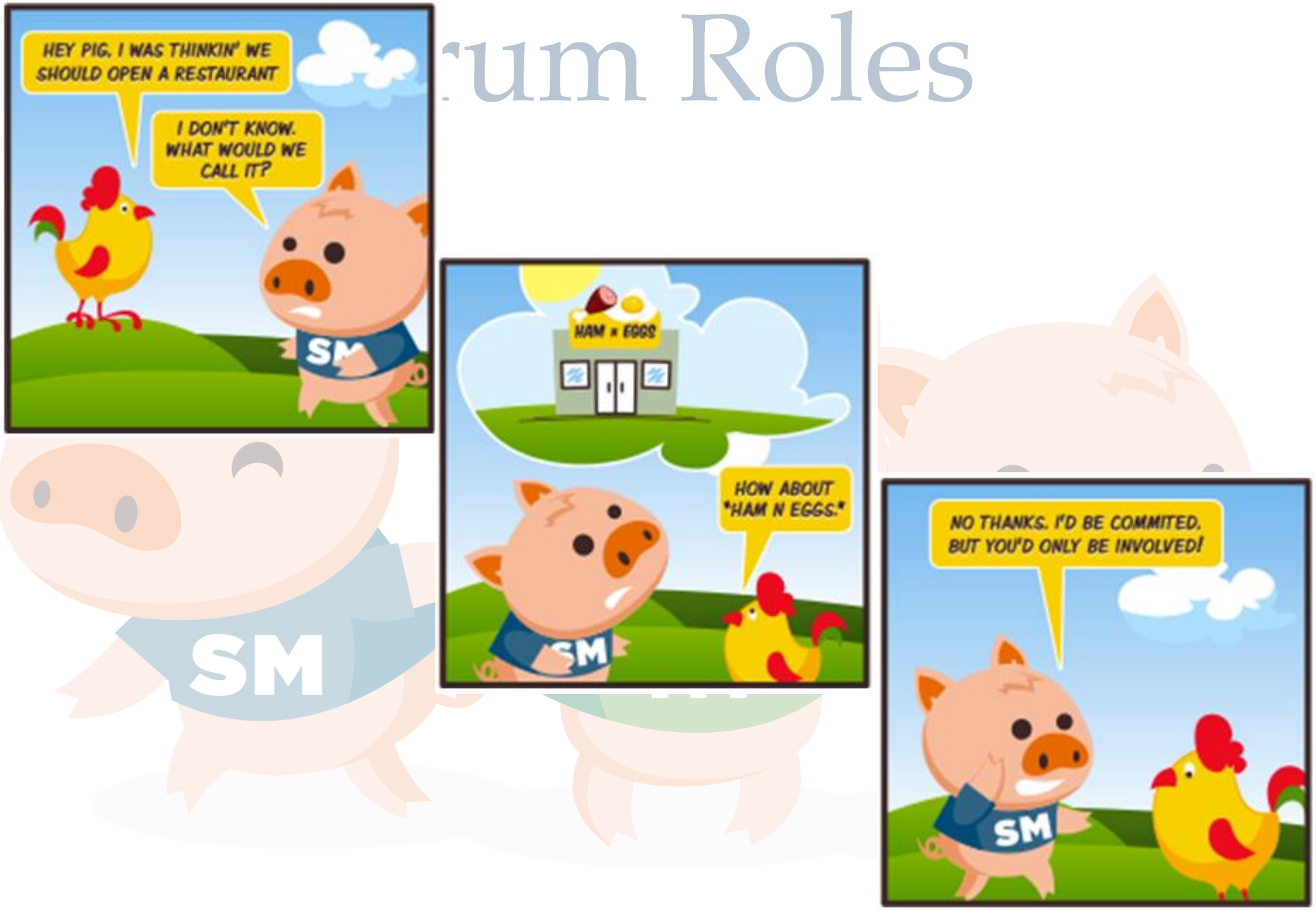
They refer to

- Process
- Results

Scrum Roles



rum Roles


HEY PIG, I WAS THINKIN' WE SHOULD OPEN A RESTAURANT
I DON'T KNOW. WHAT WOULD WE CALL IT?
HAM n EGGS
HOW ABOUT "HAM N EGGS."
NO THANKS. I'D BE COMMITED, BUT YOU'D ONLY BE INVOLVED!
SM

Scrum Roles

Product Owner

- holds the vision for the product
- represents the interests of the business
- represents the customers
- owns the product backlog
- orders (prioritizes) the items in the product backlog
- creates acceptance criteria for the backlog items
- is available to answer the team members' questions

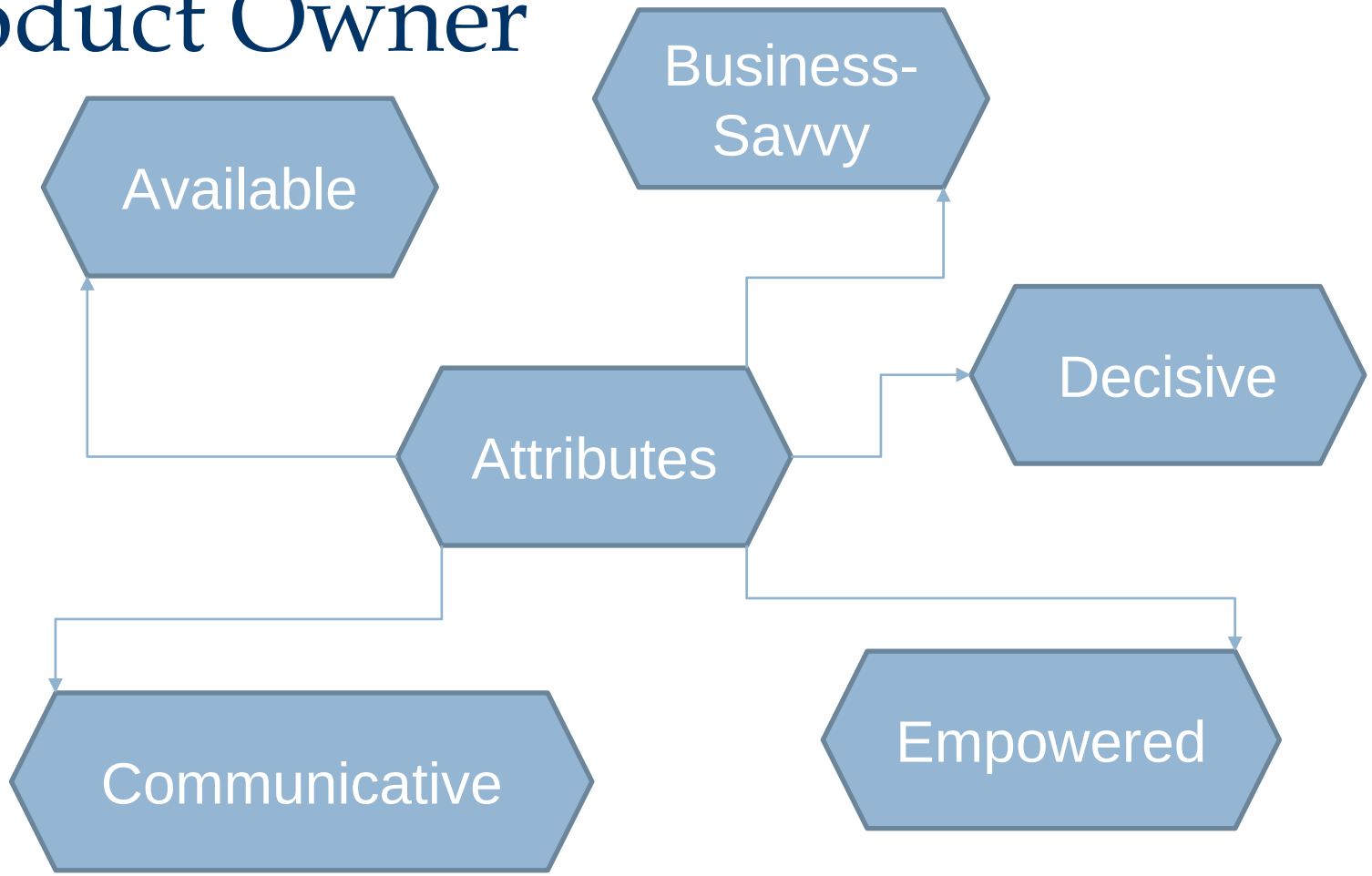
Business objectives
Project objectives

Features
vs.
User Stories

One person?

Scrum Roles

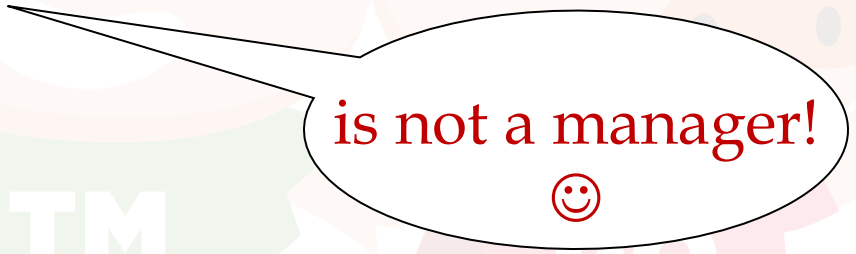
Product Owner



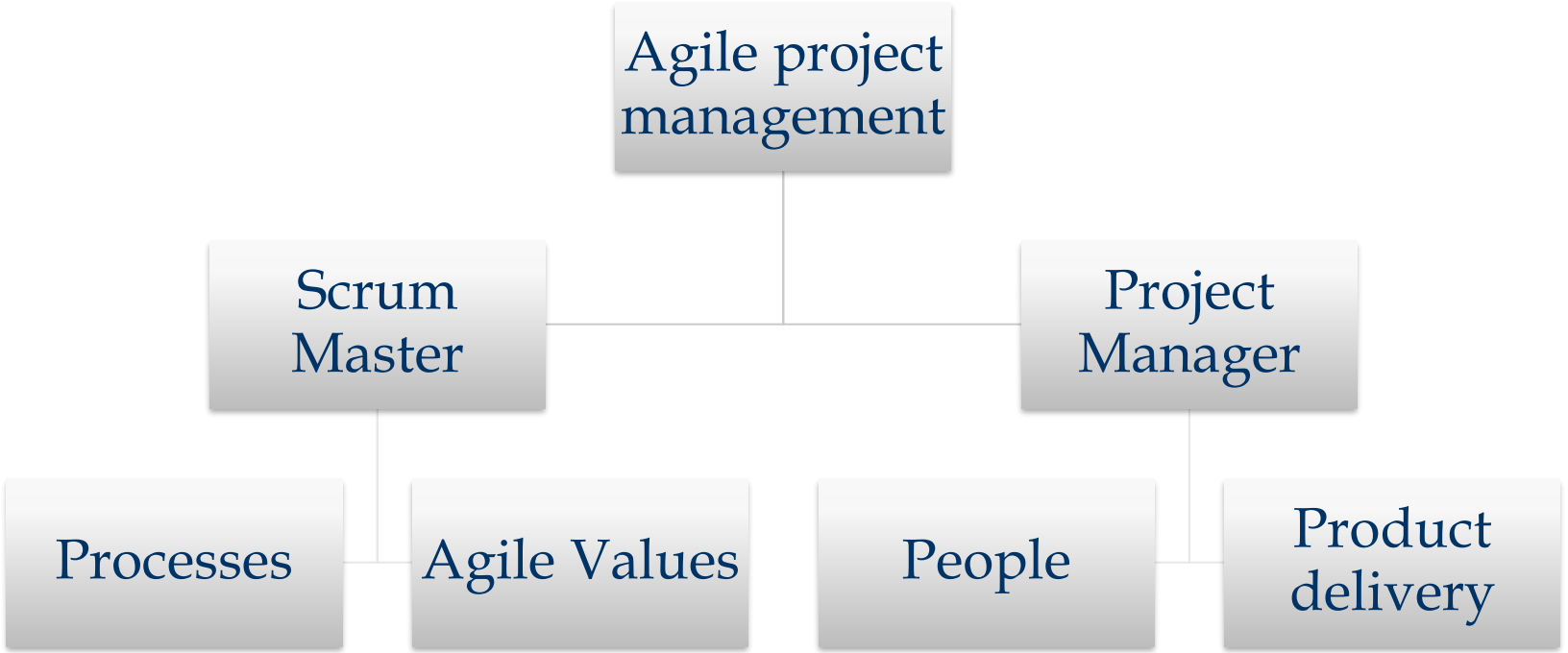
Scrum Roles

Scrum Master

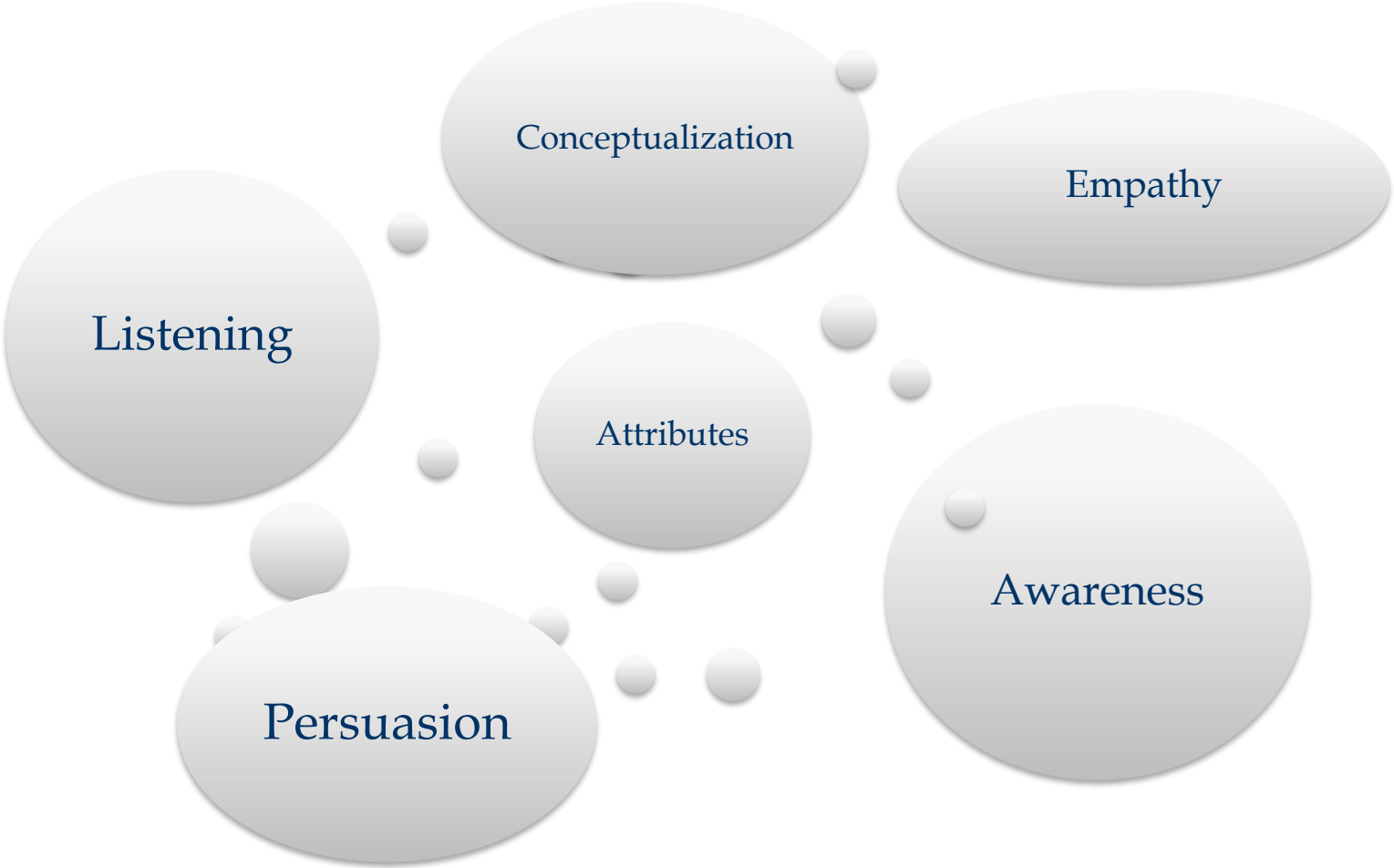
- the team's good shepherd
- coach
- guardian
- facilitator
- scrum expert
- impediment bulldozer



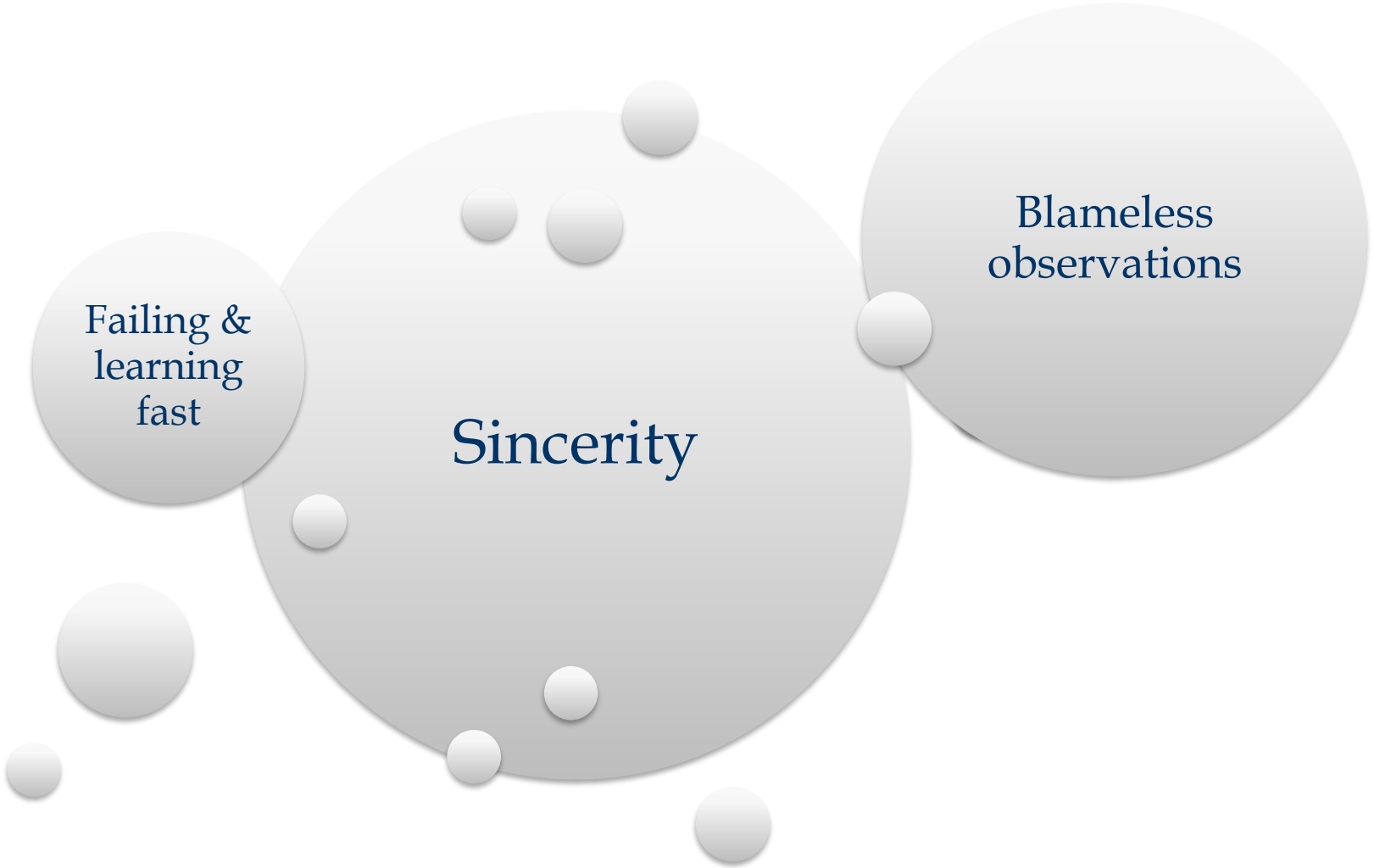
Scrum Master + PM



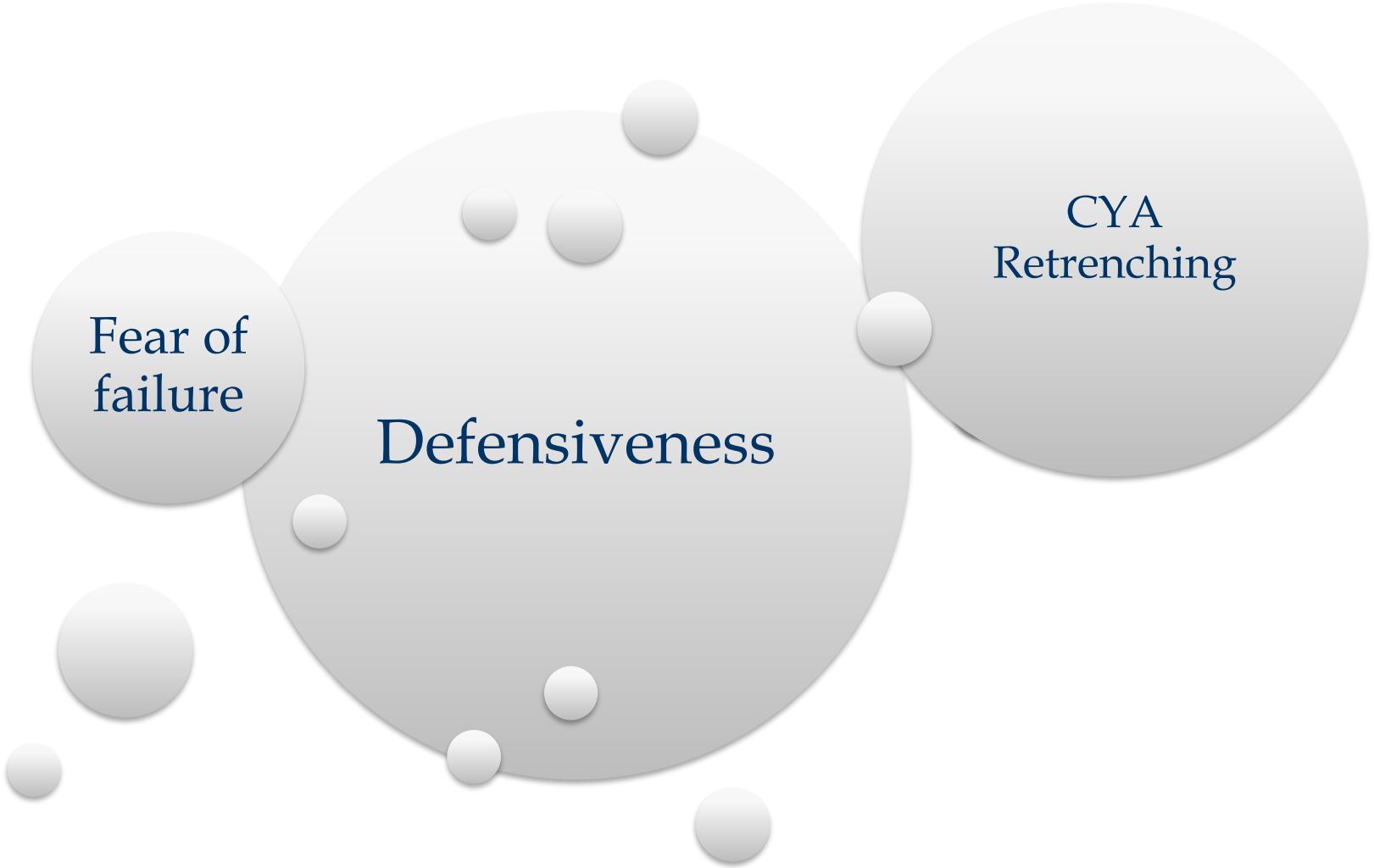
Scrum Master = Servant Leader



Scrum Master Encourages



Scrum Master Discourages



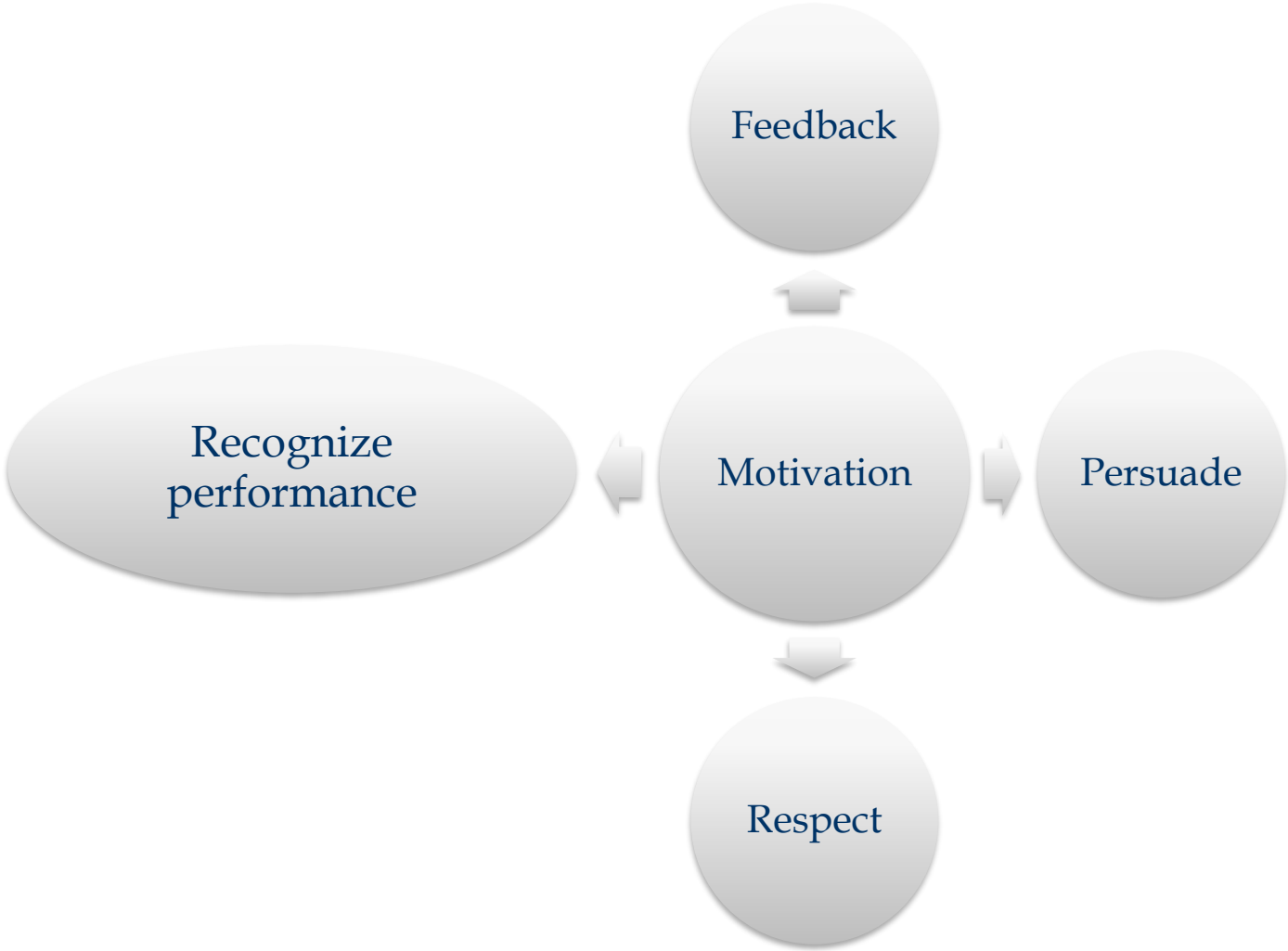
Scrum Roles

Team Member

- responsible for completing user stories to incrementally increase the value of the product
- self-organizes to get all of the necessary work done
- creates and owns the estimates
- owns the " how to do the work" decisions
- avoids siloed "not my job" thinking

The team responsibilities

The team motivation



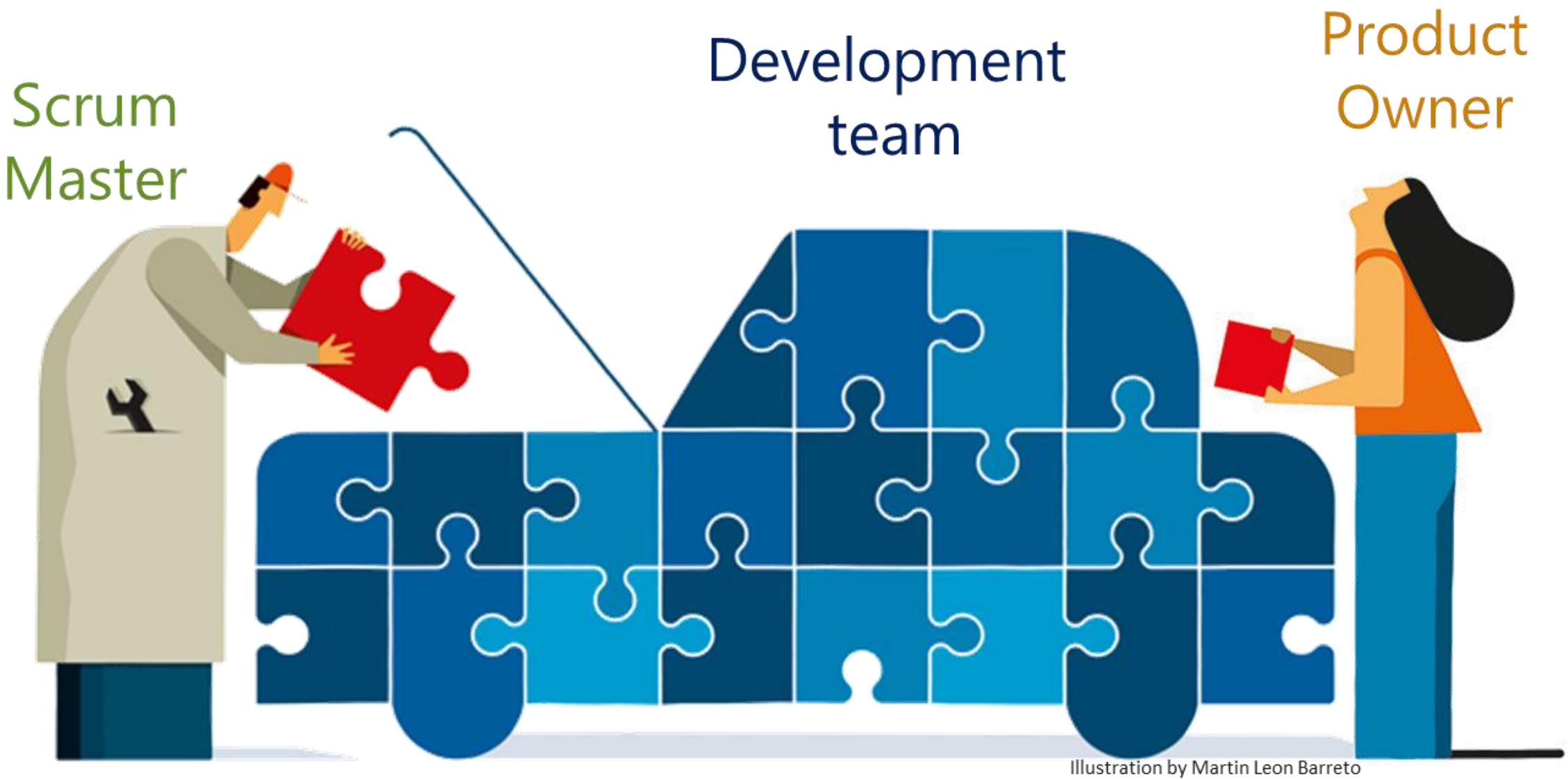
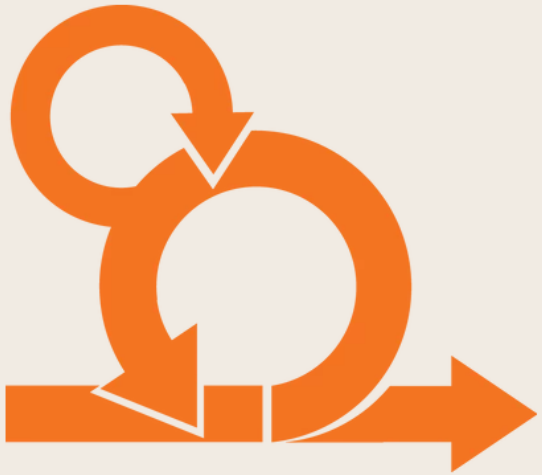


Illustration by Martin Leon Barreto



AGILE SOFTWARE DEVELOPMENT

Scrum Methodology: Artifacts

Scrum Artifacts

the tools Scrum practitioners use to make the process visible

Scrum Artifacts

The Product Backlog

- the cumulative list of desired deliverables for the product
- includes:
  - features
  - bug fixes
  - documentation changes
  - etc (anything meaningful & valuable to produce)

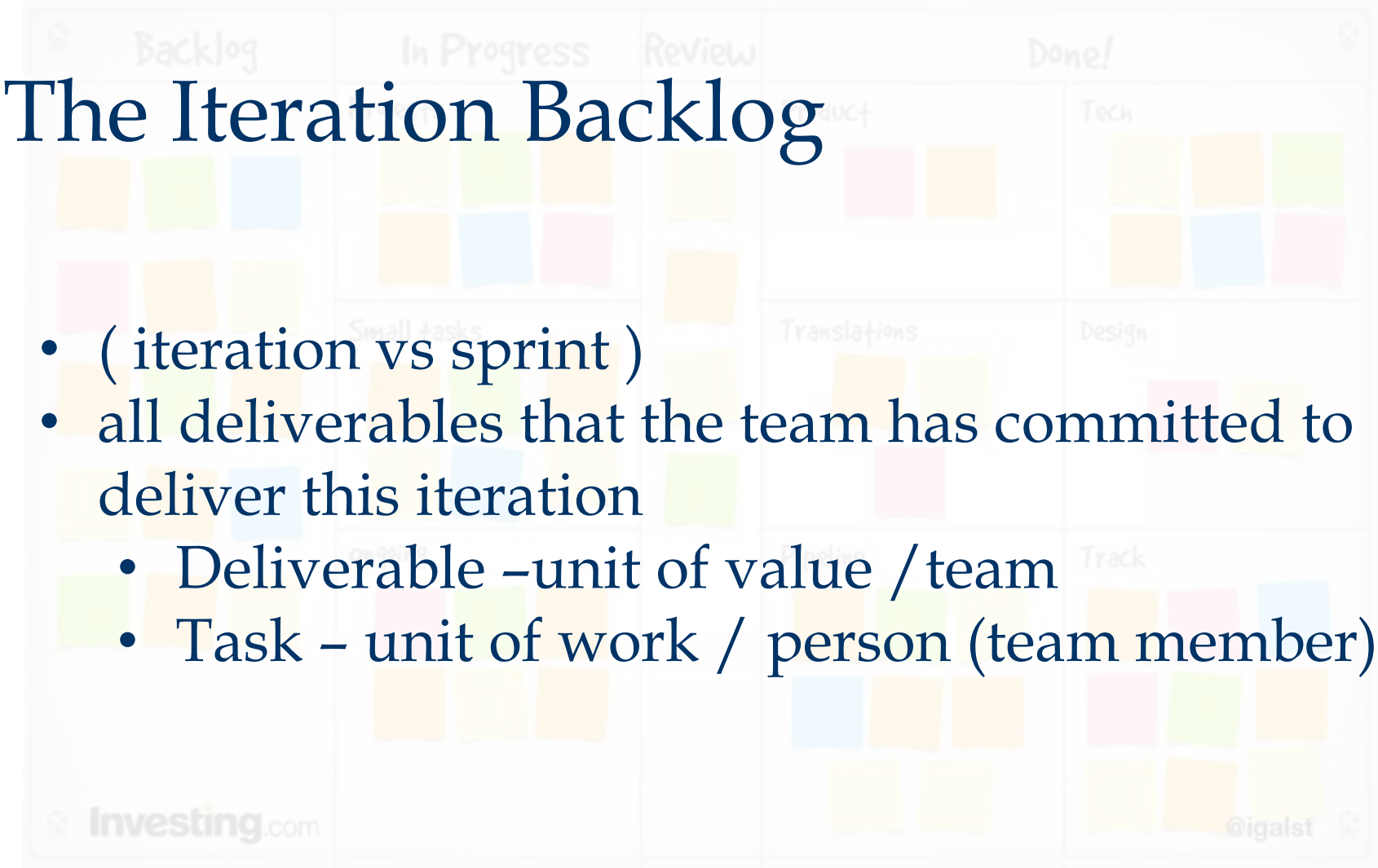
Scrum Artifacts

The Product Backlog

- for each deliverable from backlog we should know:
  - Who is it for?
  - What needs to be built?
  - Why we should do it?
  - How much work requires to implement?
  - Acceptance criteria
  - Priority

Scrum Artifacts

The Iteration Backlog



- (iteration vs sprint)
- all deliverables that the team has committed to deliver this iteration
  - Deliverable –unit of value /team
  - Task – unit of work / person (team member)

Scrum Artifacts

No changes during a Sprint!

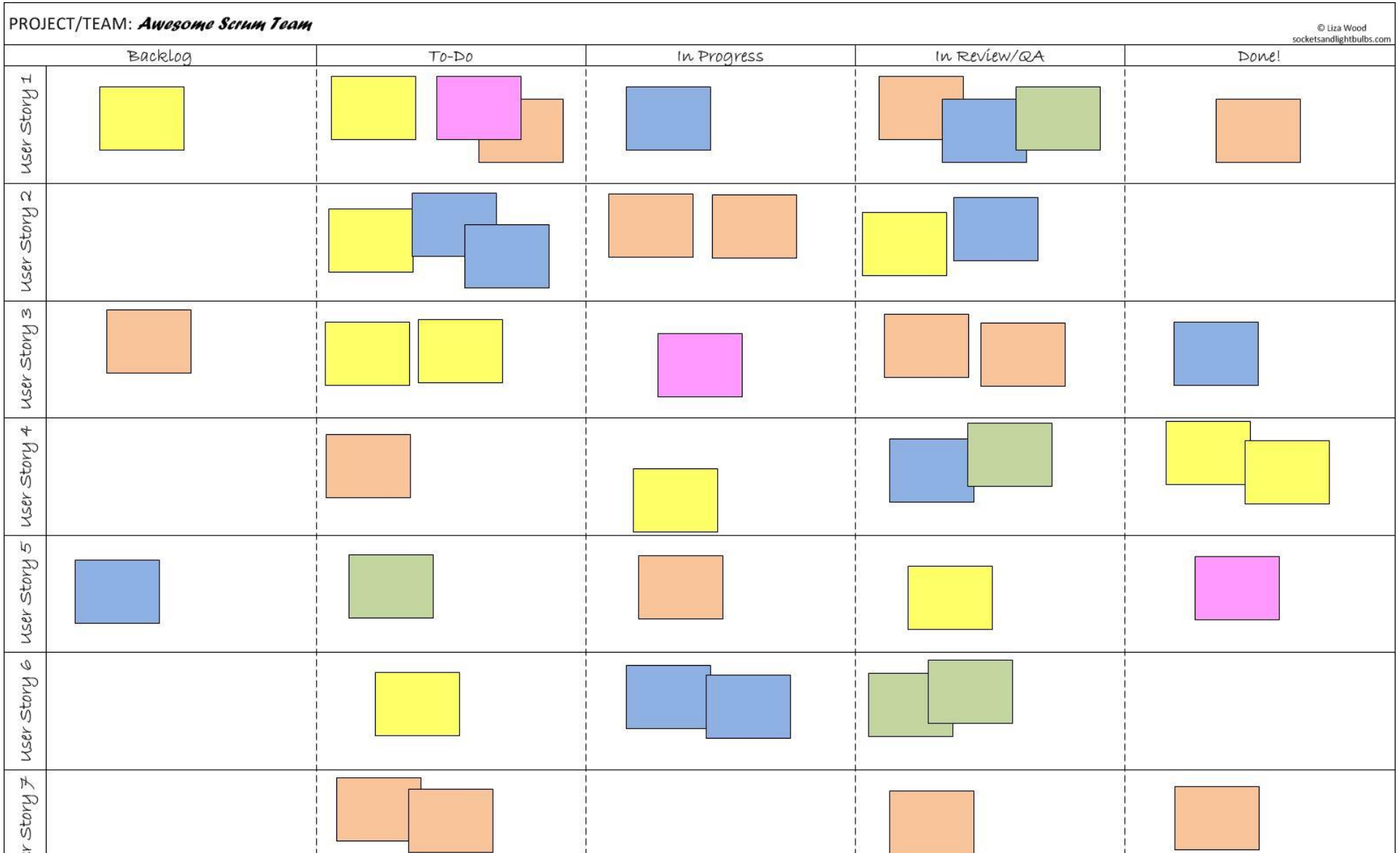
Plan sprint durations around how long you can commit to keeping change out of the sprint!

Scrum Artifacts

Product Increment

- Sum of all the Product Backlog items completed during a sprint and all previous sprints
- At the end of a sprint, the new Increment must be Done

Task Board


PROJECT/TEAM: Awesome Scrum Team
© Liza Wood
socketsandlightbulbs.com
Backlog
To-Do
In Progress
In Review/QA
Done!
User Story 1
User Story 2
User Story 3
User Story 4
User Story 5
User Story 6
User Story 7

Scrum Artifacts

Definition of Ready

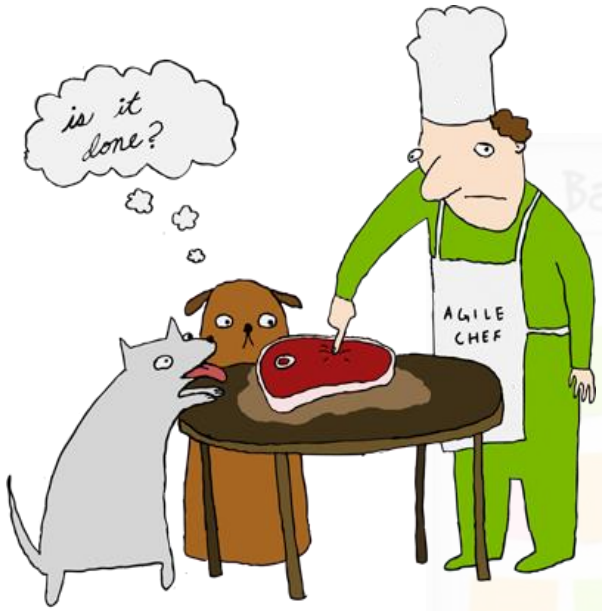
- The Definition of Ready (DoR) sets out the criteria for the stories needed to make the Sprint succeed
- The DoR is drawn up by the Development Team, cooperating with the Product Owner
- The Development Team determines whether Product Backlog items meet the DoR
- The PO respects the DoR. That means that a Product Backlog item is only included in a Sprint if it meets the DoR.

Scrum Artifacts

Definition of Ready

- Example:
  - Story Statement
  - Specification by Example
  - Flow chart, if needed
  - Use Case, if Acceptance Criteria missing
  - Wireframe, if needed, delivered
  - UX [mock-up] if needed, delivered

Scrum Artifacts

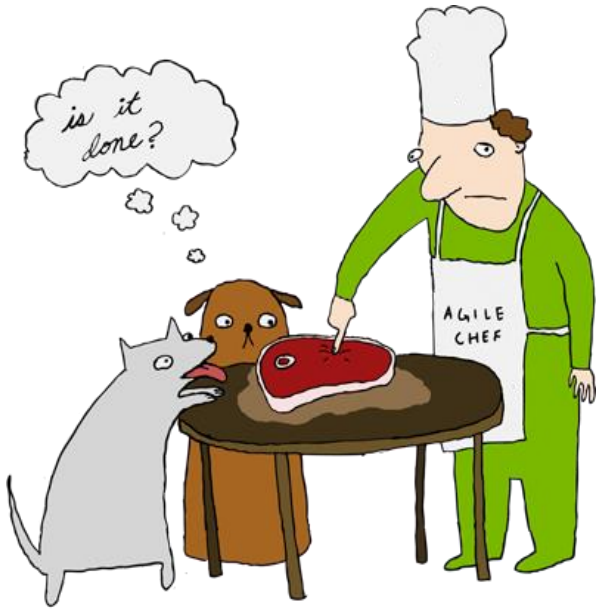


Definition of Done

- "when the code has been written" (programmer)
- "all of the tests have passed" (tester)
- "it's been loaded onto the production servers" (operations)
- "we can now sell it to customers"(business person)

- Each team creates its own "definition of done"

Scrum Artifacts

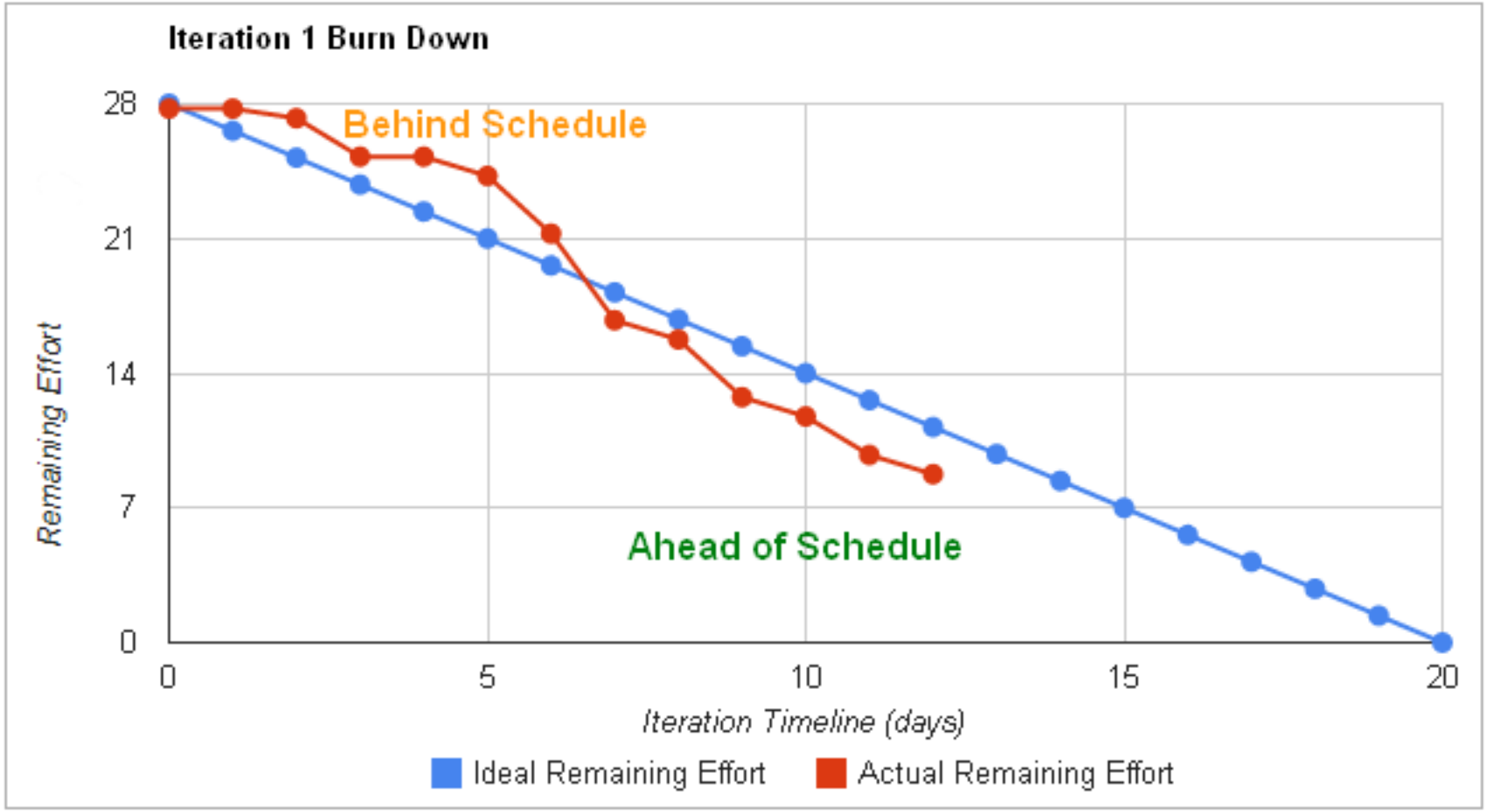


Definition of Done

- Is a crucial tool for making sure that the developed product is **satisfying stakeholders expectations**
- The team is **only finished with a Sprint Backlog item once it meets the DoD**
- The DoD is **drawn up by the Development Team**
- A Definition of Done should exist for:
  - User stories
  - Releases
  - Final project deliverables

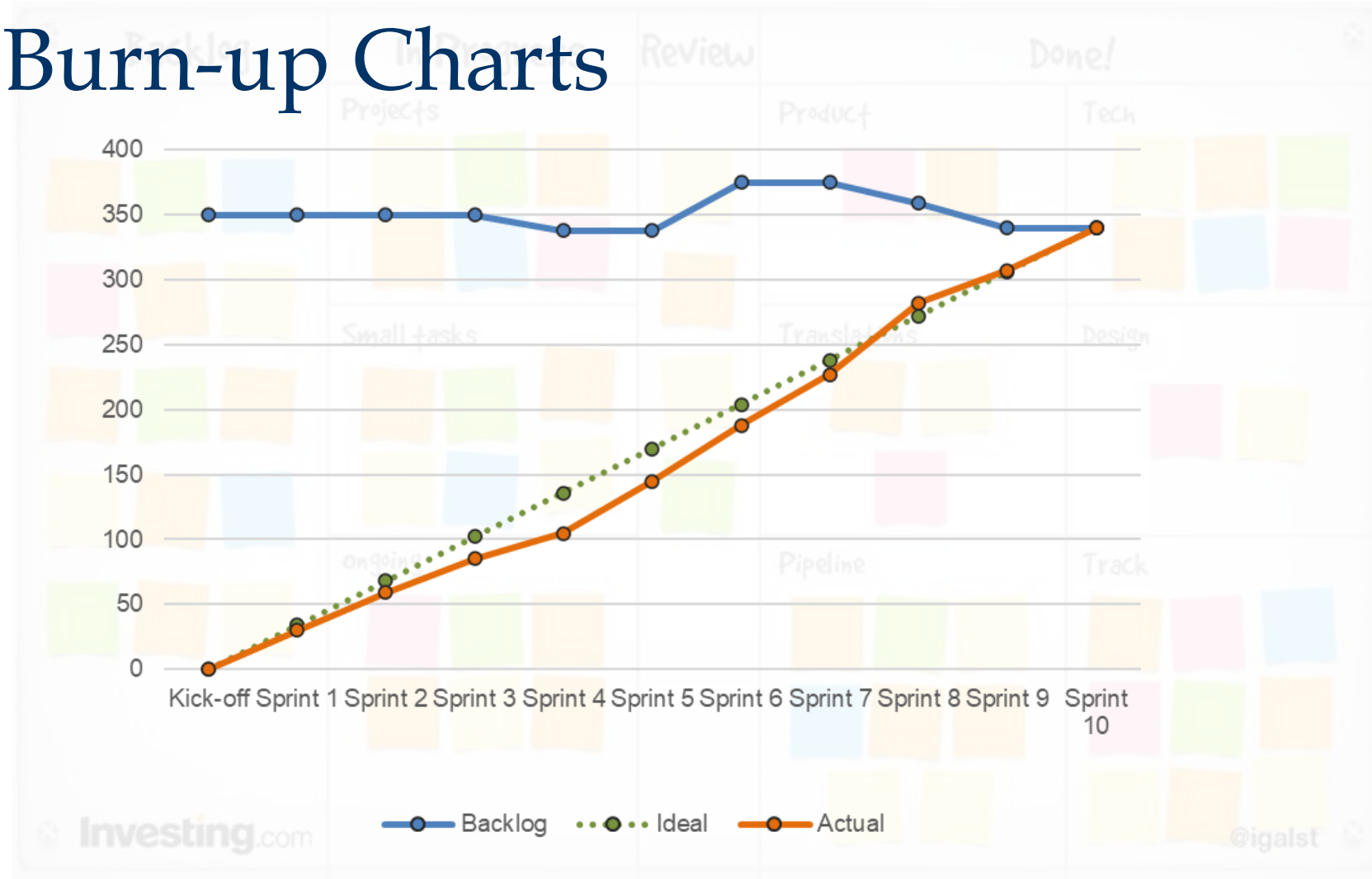
Scrum Artifacts

Burndown Charts



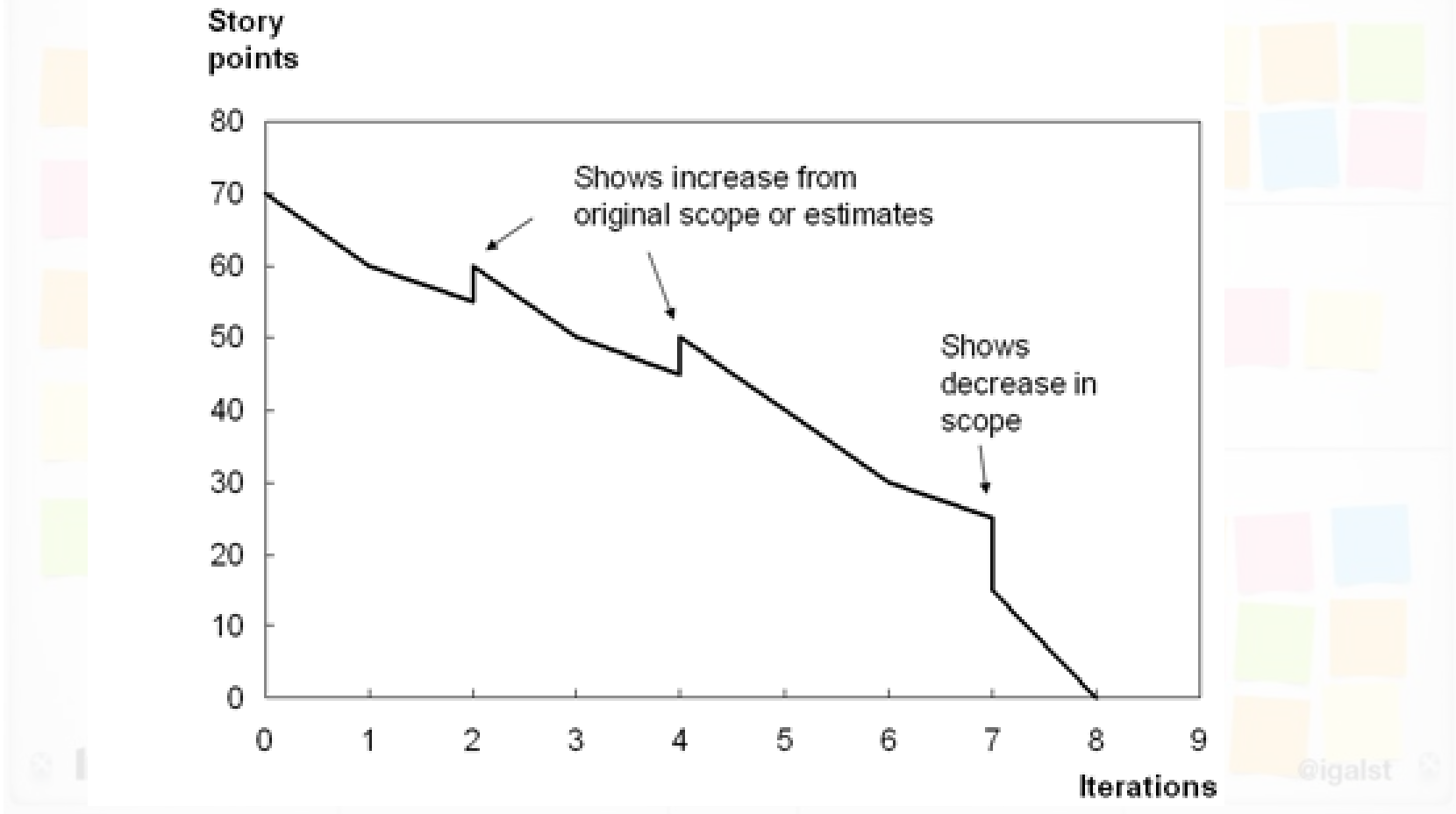
Scrum Artifacts

Burn-up Charts

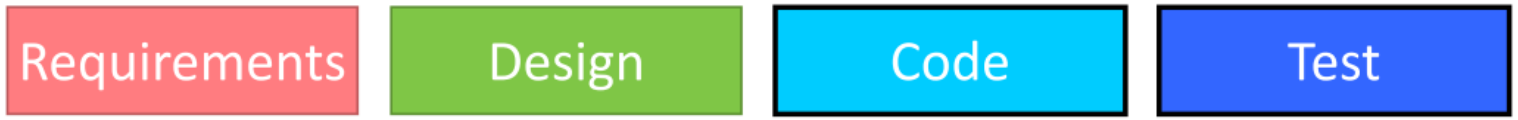


Scrum Artifacts

Burndown Charts

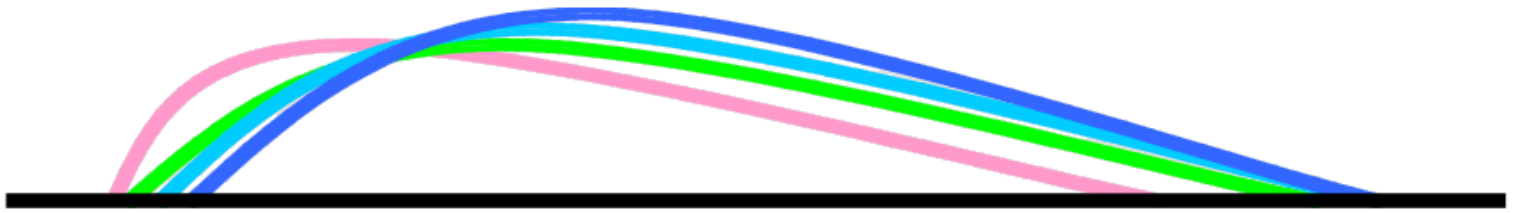


Sequential vs Overlapping Development

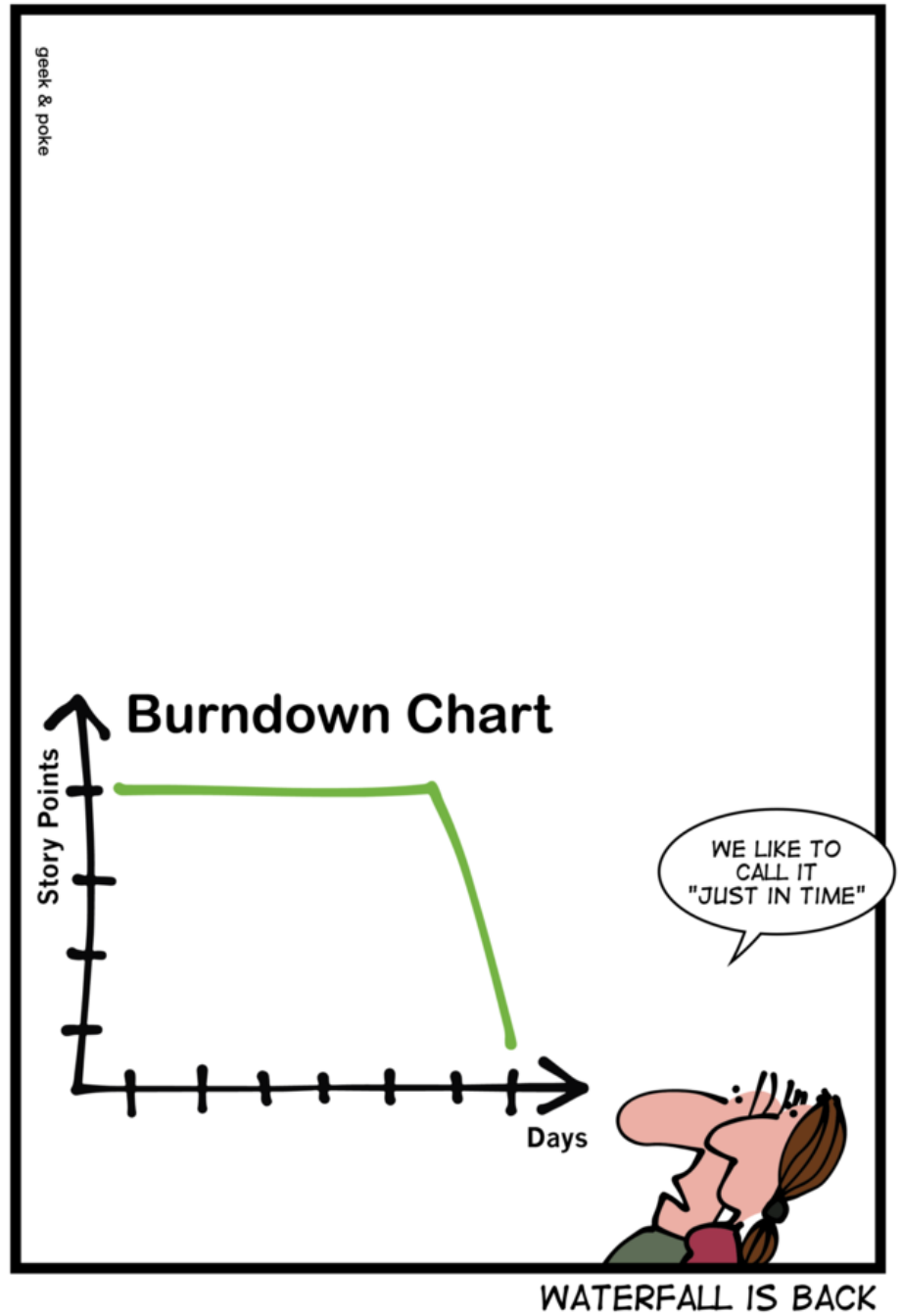


Rather than doing all of one thing at a time

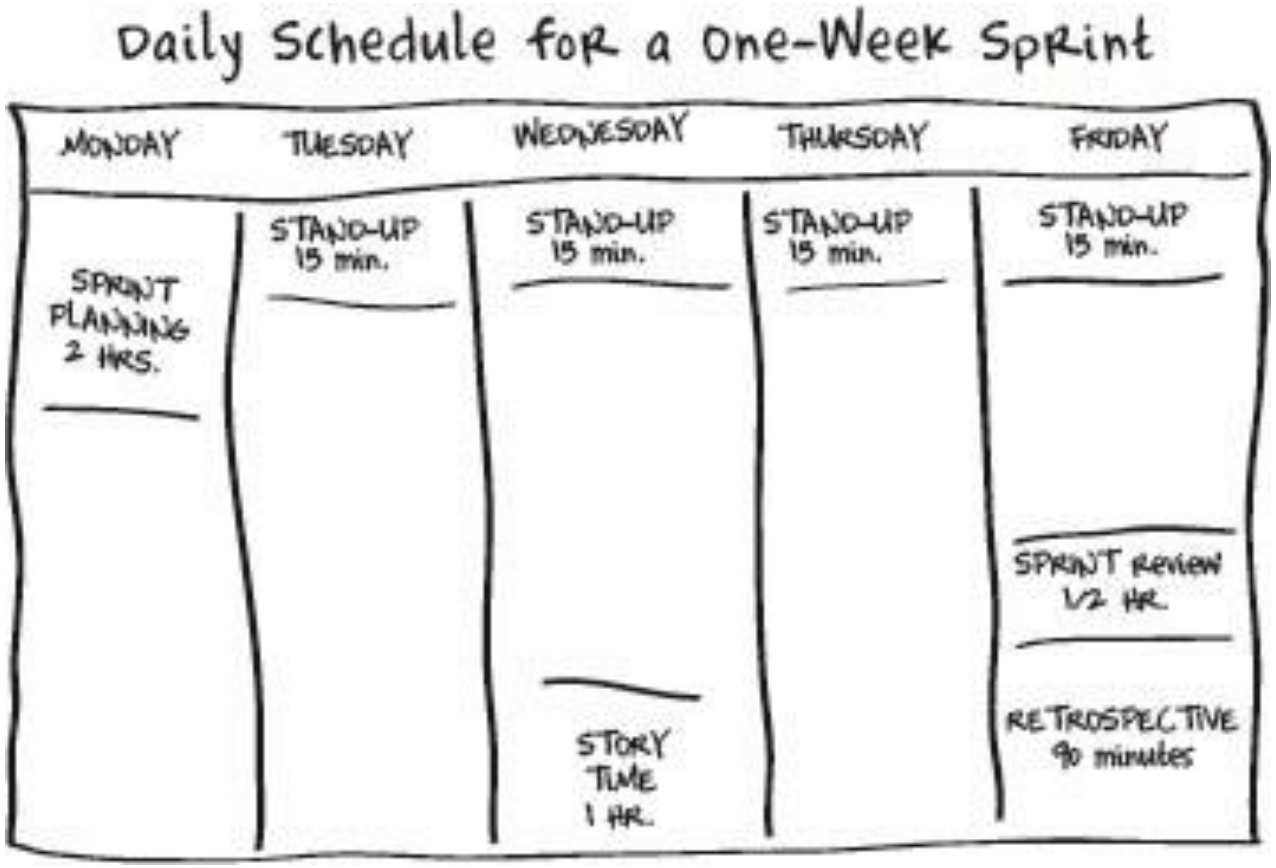
…Scrum teams do a little of everything all the time

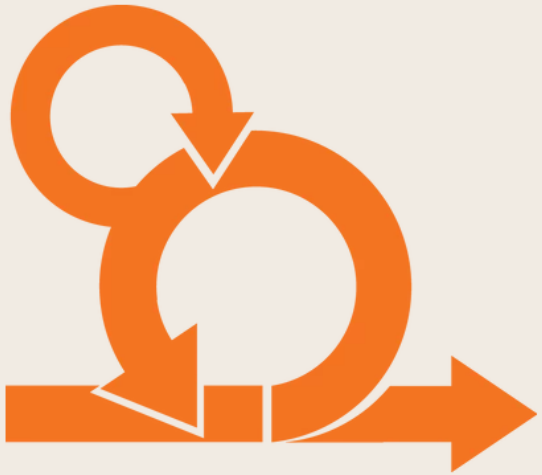


Source: “The New New Product Development Game”
Takeuki, Nonaka, HBR, Jan 1986


geek & poke
Burndown Chart
Story Points
Days
WE LIKE TO CALL IT "JUST IN TIME"
WATERFALL IS BACK

Next: Scrum Ceremonies

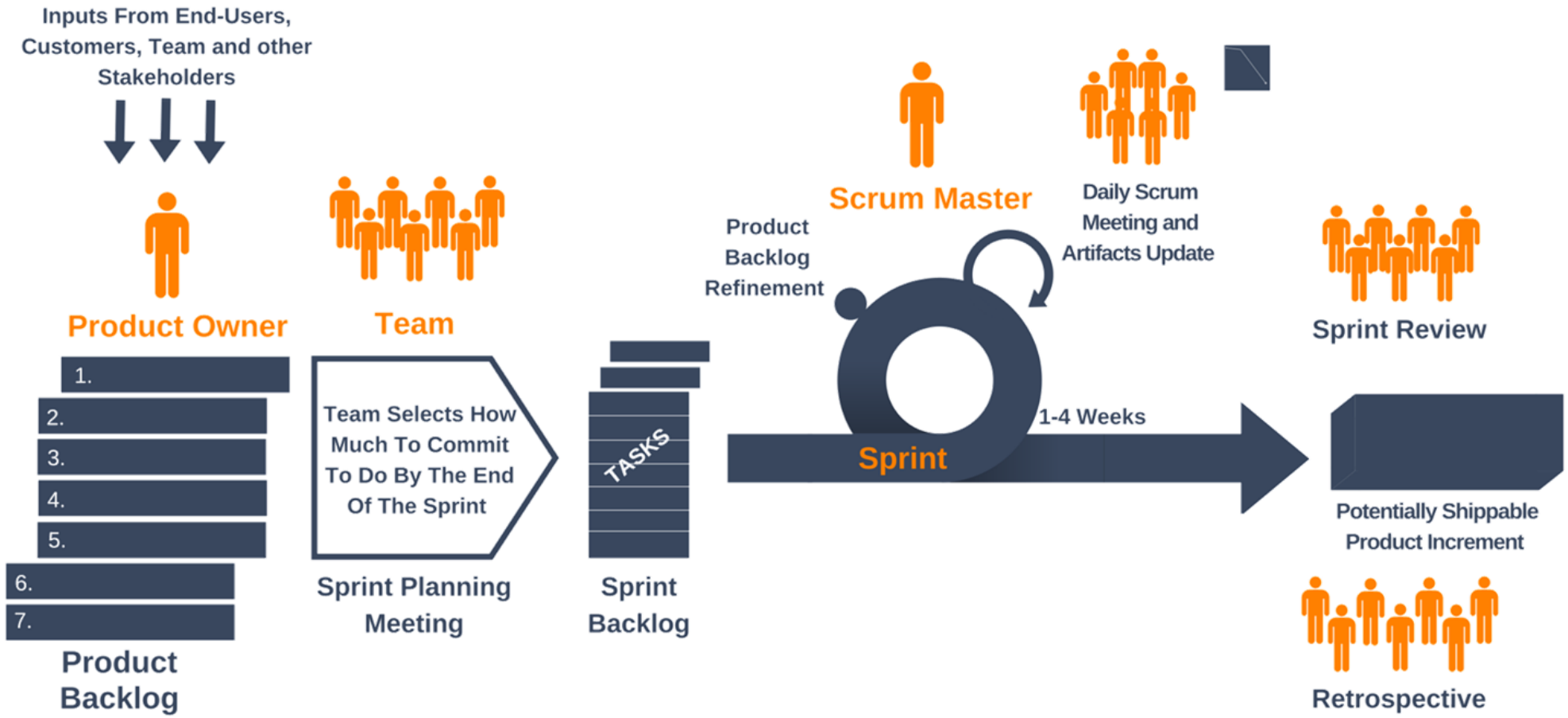




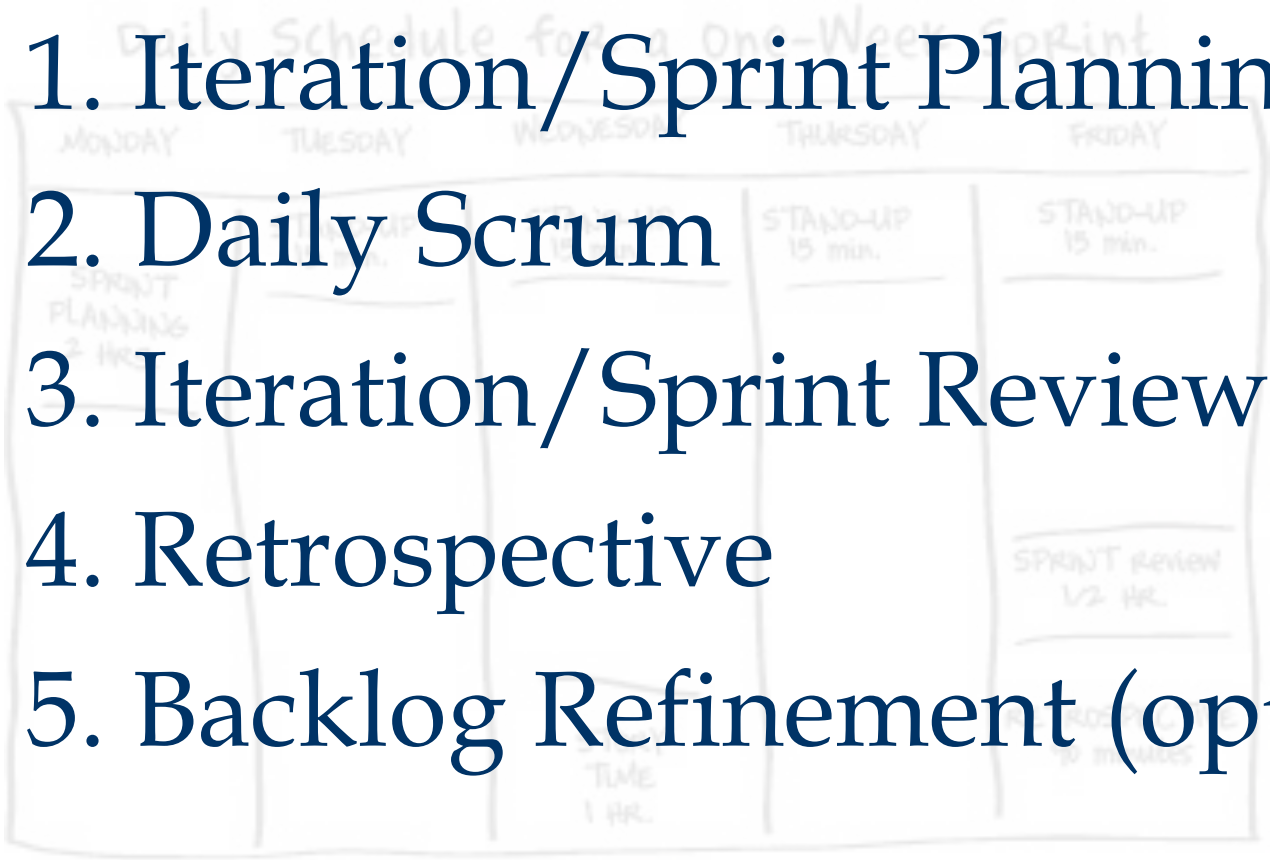
AGILE SOFTWARE DEVELOPMENT

Scrum Methodology: Ceremonies

Scrum Ceremonies



Scrum Ceremonies



1. Iteration/Sprint Planning
2. Daily Scrum
3. Iteration/Sprint Review
4. Retrospective
5. Backlog Refinement (optional)

1. Iteration/Sprint Planning

- **Rules**
  - Duration: timeboxed to maximum 2h/sprint week
  - Participants: ***Product Owner*** and ***Development Team*** (external stakeholders that may contribute are optional)
  - Facilitator: ***Scrum Master***
- **Goal**
  - Project predictability
- **Objectives**
  - To create a Sprint Backlog
  - To get Team commitment for the Sprint

1. Iteration/Sprint Planning

- **Structure**
  - Part 1: PO presents the Stories that are prioritized and ready for the Sprint (half of time)
  - Part 2: The Team is sizing Stories and slicing them in Tasks, then commits to PO (half of time)
- **Common traps**
  - Long and boring meetings with too detailed technical discussions
  - Superficial approach, team members do not really understand the requirements
  - Wrong and/or manipulated estimations


mbna
mbna
LAND ROVER
GILBERT

2. Daily Scrum

- **Rules**
  - Duration: timeboxed to maximum 15 minutes
  - Participants: Development Team (PO is welcome, but optional and must be silent)
  - Facilitator: Scrum Master or a Team member
- **Goal**
  - Self-organizing
- **Objectives**
  - To have a common Team understanding of the Sprint progress
  - To collect impediments and decide who will tackle each of them

2. Daily Scrum

- **Structure**
  - Each Team member explains what he/she achieved since last Daily Scrum, what he/she will work until next Daily Scrum and what impediments prevent him/her to work (if any)

- **Common traps**
  - Long and boring meetings with too detailed technical discussions
  - Superficial approach, team members do not really understand the Sprint progress
  - False impediments

3. Iteration/Sprint Review

- **Rules**
  - Duration: timeboxed to maximum 1h/sprint week
  - Participants: PO and Development Team (stakeholders are welcome, but optional)
  - Facilitator: Scrum Master
- **Goal**
  - Delivering incrementally
- **Objectives**
  - To review and accept the Stories finalized by the Team
  - To discuss potential changes of the Product Backlog resulting from the Sprint Review (priorities)

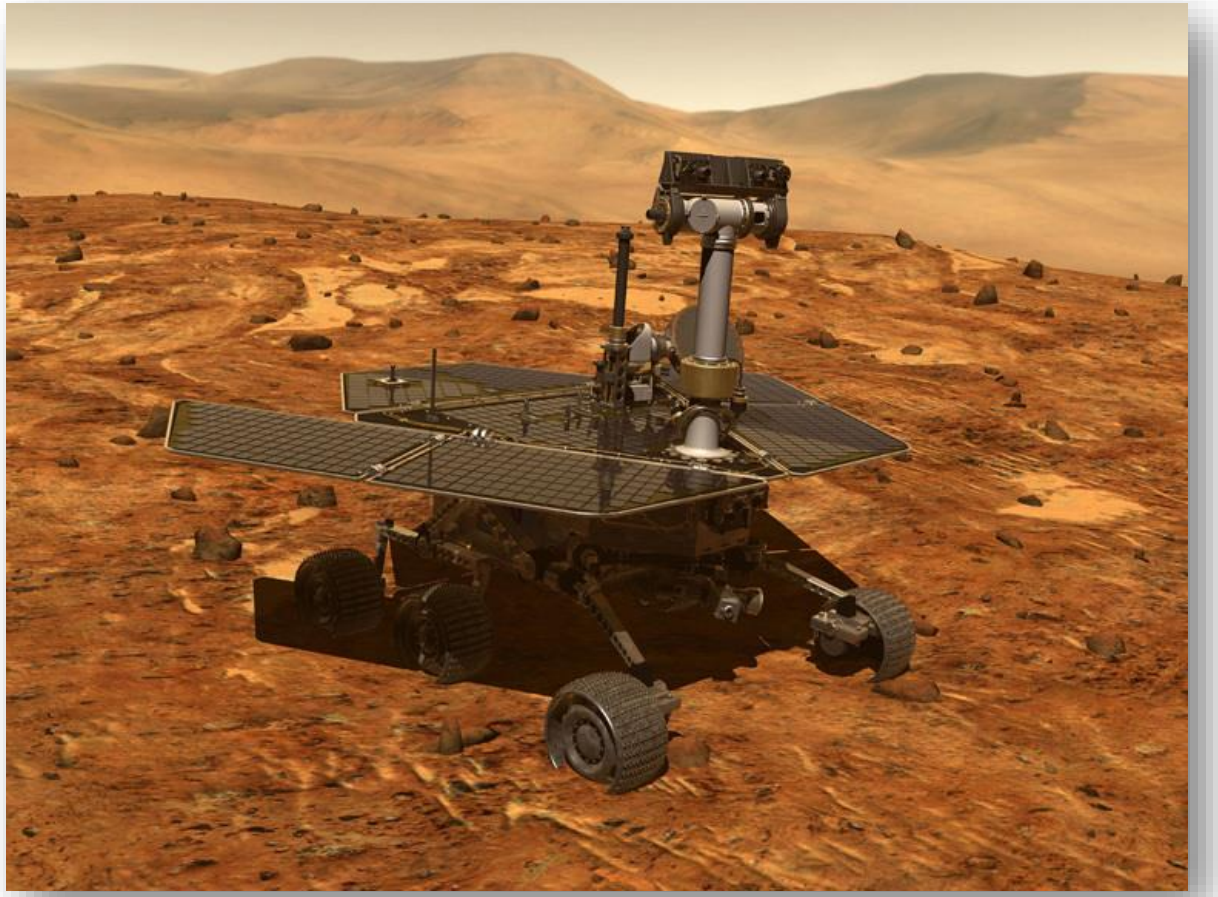
3. Iteration/Sprint Review

- **Structure**
  - Team is organizing a demonstration of finalized Stories
  - PO is reviewing and accepting depending on DoD and Acceptance Criteria
  - PO and team discuss the results of the Sprint and the consequences on subsequent Sprints

- **Common traps**
  - Acceptance Criteria not checked during review (incomplete testing)
  - DoD ignored

3. Iteration/Sprint Review

Fast feedback



Remote driving Mars Rover

- 10 minutes to send radio signals between Earth and Mars

3. Iteration/Sprint Review

Feedback

- Bugs
- Small cosmetic things
- New ideas

4. Retrospective

- **Rules**
  - Duration: timeboxed to maximum 45mins/sprint week
  - Participants: Development Team (PO only if invited by the Team)
  - Facilitator: Scrum Master
- **Goal**
  - Continuous improvement
- **Objectives**
  - To review the efficiency and effectiveness of the teamwork and methods used
  - To determine and decide upon action items that will lead to improving Team's performance

4. Retrospective

- **Structure**
  - Collecting Team member's opinions about what they should keep doing, stop doing or doing differently
  - Discussing and collectively deciding the improvements applied in the next Sprint

- **Common traps**
  - False harmony, avoiding to discuss about "the elephant in the room"
  - Finger-pointing, personal conflicts
  - Failing to identify effective action items

4. Retrospective

Retrospective Structure

01 Set the stage

Encourage the people to get in the mood for active participation

02 Gather data

Create a shared image of what happened during the sprint

03 Generate insights

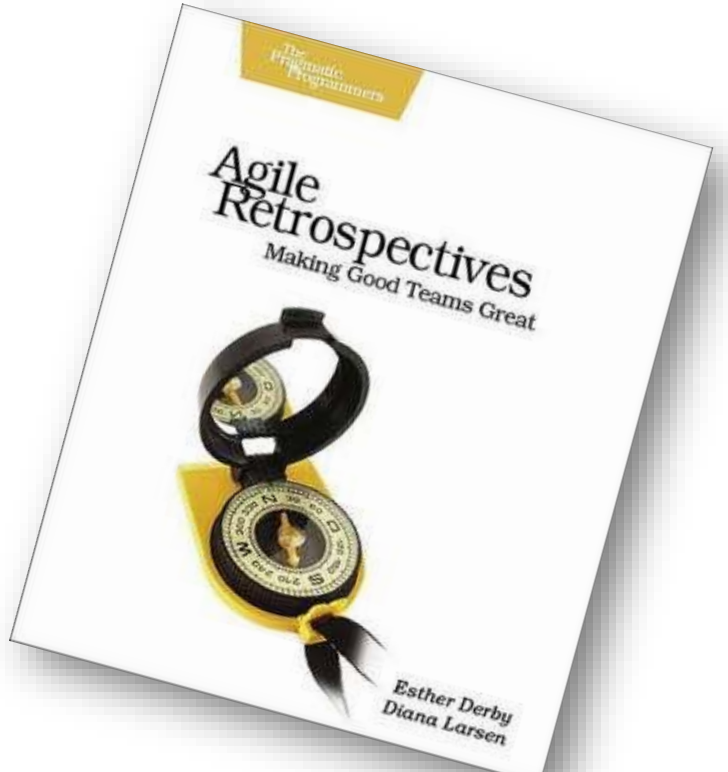
Understand implications of findings and discussions

04 Decide what to do

Identify highest priority action items that can be planned for next sprint

05 Close retrospective

Reflect on what helped during the retrospective



4. Retrospective

Set the stage

Goal: getting team members in the right mood for retrospective

Check-in

Two words summary for what they hope to get from retrospective or how they are feeling about retrospective

Focus on/Focus off

Discuss on terms like “Dialogue rather than Debate” or “Conversation rather than Argument”. Discuss what to do to move people on the right/focus on?

ESVP

Explorers: discover new ideas
Shoppers: look over new useful ideas
Vacationers: happy to be away from work
Prisoners: forced to come to retrospective

Working agreements

Brainstorming over working agreements they would like to put in place for the retrospective

4. Retrospective
Gather data

Goal: remembering what happened during last iteration

Follow-up

Discuss the success or failure of improvement actions decided during previous retrospective

Timeline

Team members recall good, problematic, significant events happened on a timeline (happy/sad faces bottom side)

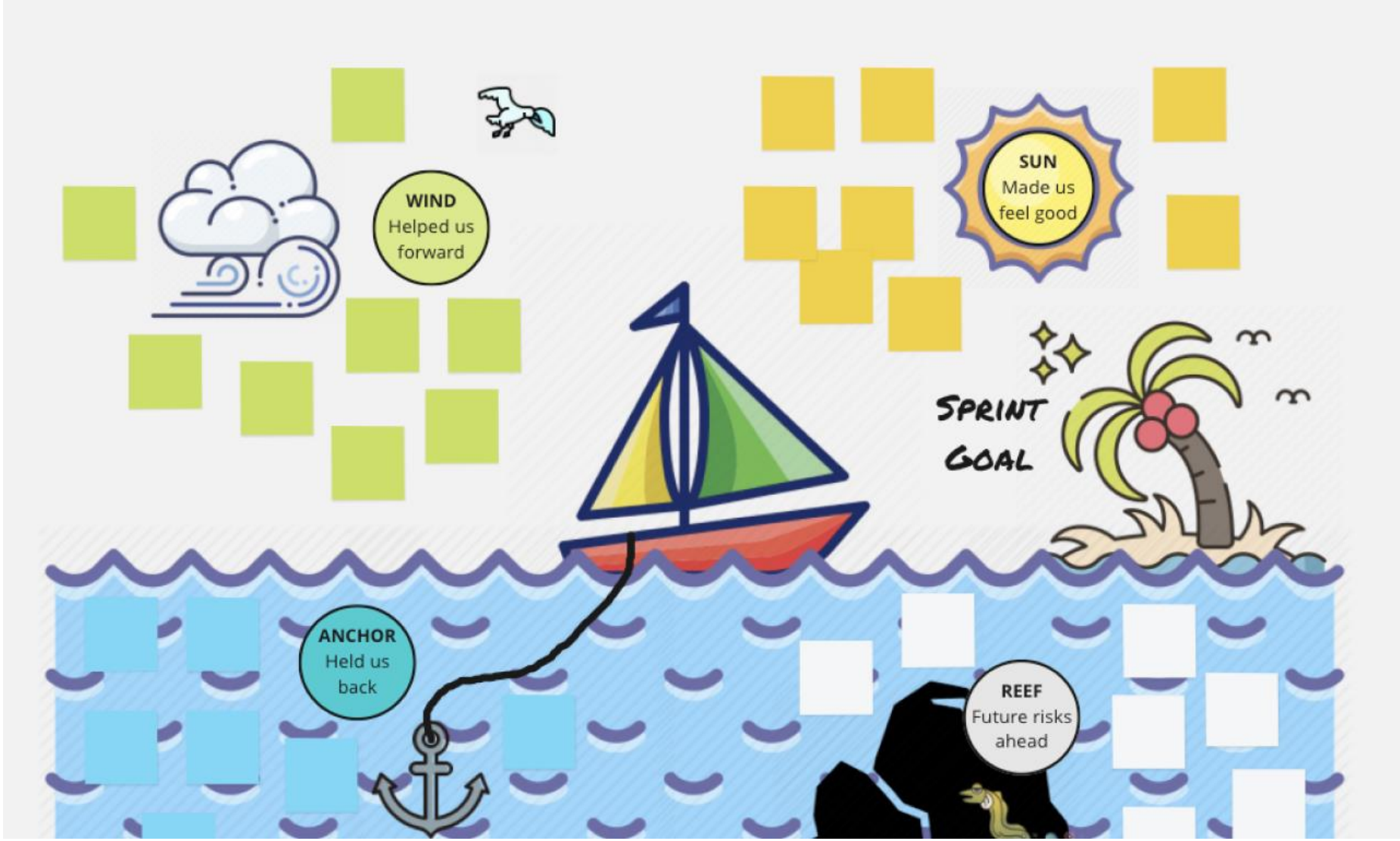
Triple nickel

Spend 5 minutes gathering at least 5 ideas related to a specific issue. Team is divided in groups of 5 if team is bigger than 7. Pass written issues to the right and give another 5 minutes to reflect and expand the ideas

Others

Mad-Sad-Glad,
Satisfaction histogram,
Team radar etc

4. Retrospective
Gather data



4. Retrospective
Generate insights

Goal: identifying potential improvements of collaboration and teamwork

Brainstorming

Quiet writing: silent writing of ideas
Round-robin: pass token around the group
Free-for-all: ad-hoc information generation

5 Whys

Ask why five times
to discover cause-and-effect relation

Fishbone

Visual tool to display
root cause analysis of problems: procedures, policies, system, skills etc

4. Retrospective
Decide what to do

Goal: deciding on actionable improvements for the next iteration

Short subjects

Start doing
Stop doing
Continue doing

SMART goals

Specific, Measurable,
Attainable, Relevant,
Timely

Dot voting

Prioritize potential improvements
and focus on
most relevant 2-3 actions

4. Retrospective
Close retrospective

Goal: ending retrospective and conclude upon its success

Plus/Delta

Plus: what is going well in retrospectives?
Delta: what should we change?

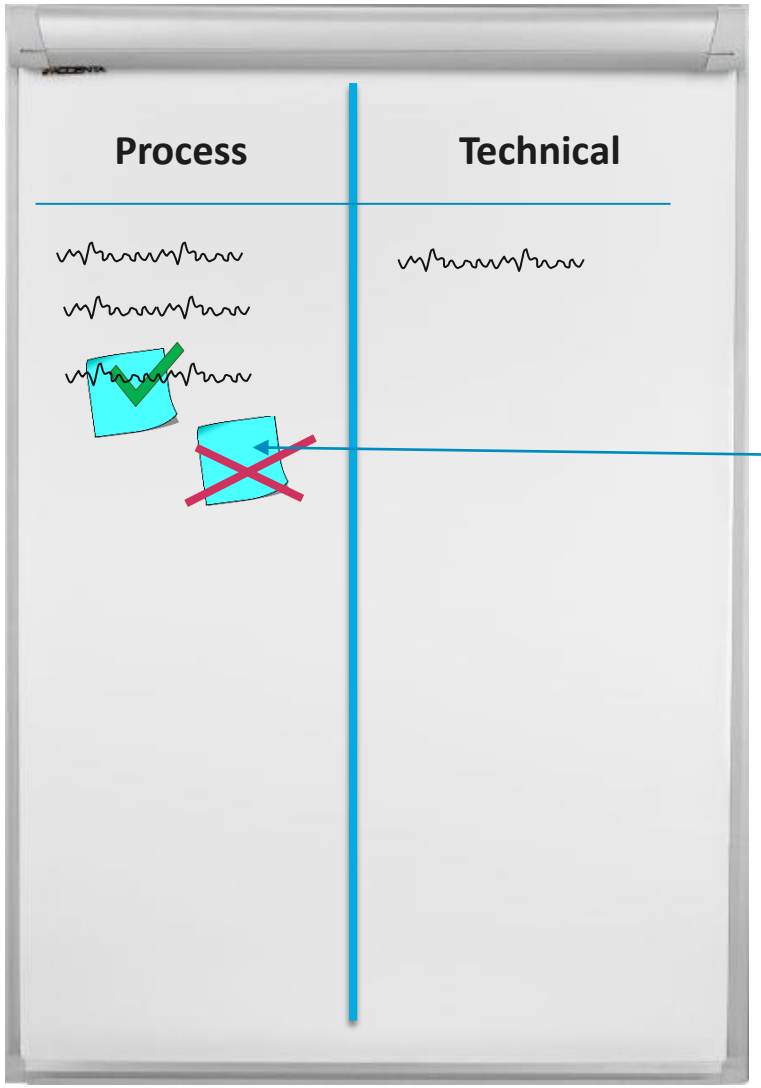
Helped, Hindered, Hypothesis

Helped: what is going well
Hindered: what was a hindrance
Hypothesis: ideas of improvement

Return on time invested

Feedback focused on how
people believe their
time was spent

4. Retrospective



What we are going to change
What we expect to happen
Who is responsible to implement it

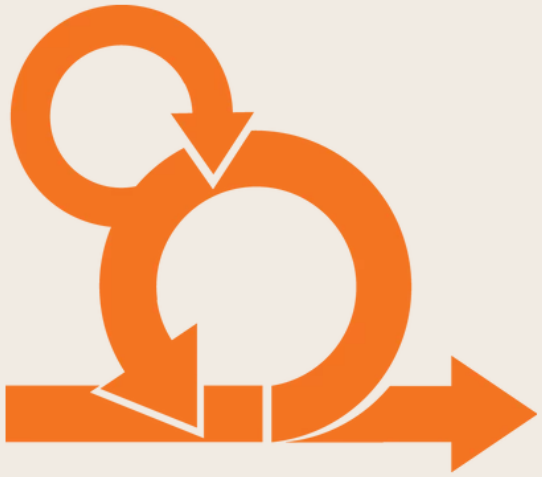
5. Backlog Refinement (Grooming)

- **Rules**
  - Duration: timeboxed to a duration negotiated by PO and Team
  - Participants: PO and Development Team
  - Facilitator: Scrum Master
- **Goal**
  - Improving the planning process
- **Objectives**
  - To communicate significant changes in the Product Backlog
  - To slice backlog items that are too big, then optionally to roughly estimate the resulted Stories
  - To collect questions about the Stories that are prioritized for next Sprint

5. Backlog Refinement (Grooming)

- **Structure**
  - Flexible agenda, proposed by Product Owner

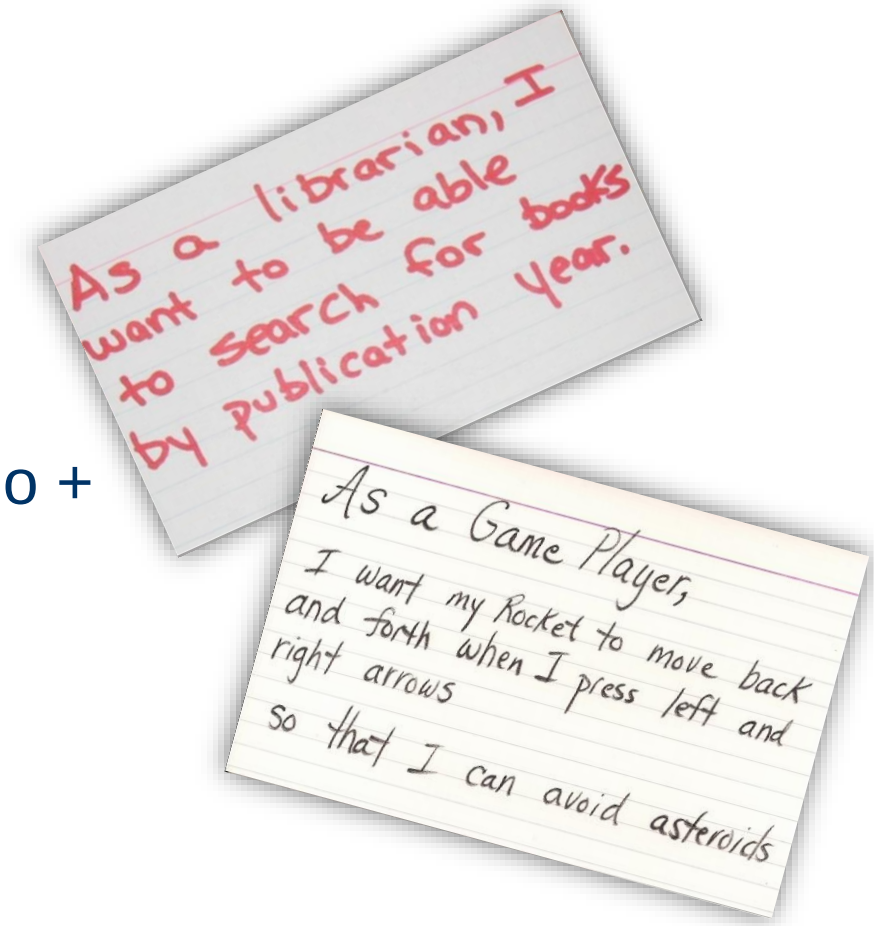
- **Common traps**
  - Getting into too detailed discussions about backlog items
  - Slicing backlog items using technical criteria (Technical Stories instead of User Stories)



AGILE SOFTWARE DEVELOPMENT

User stories

User stories



- A user story is one or more sentences in the everyday/business language that captures what a user does or needs to do +
  - a description
  - acceptance criteria

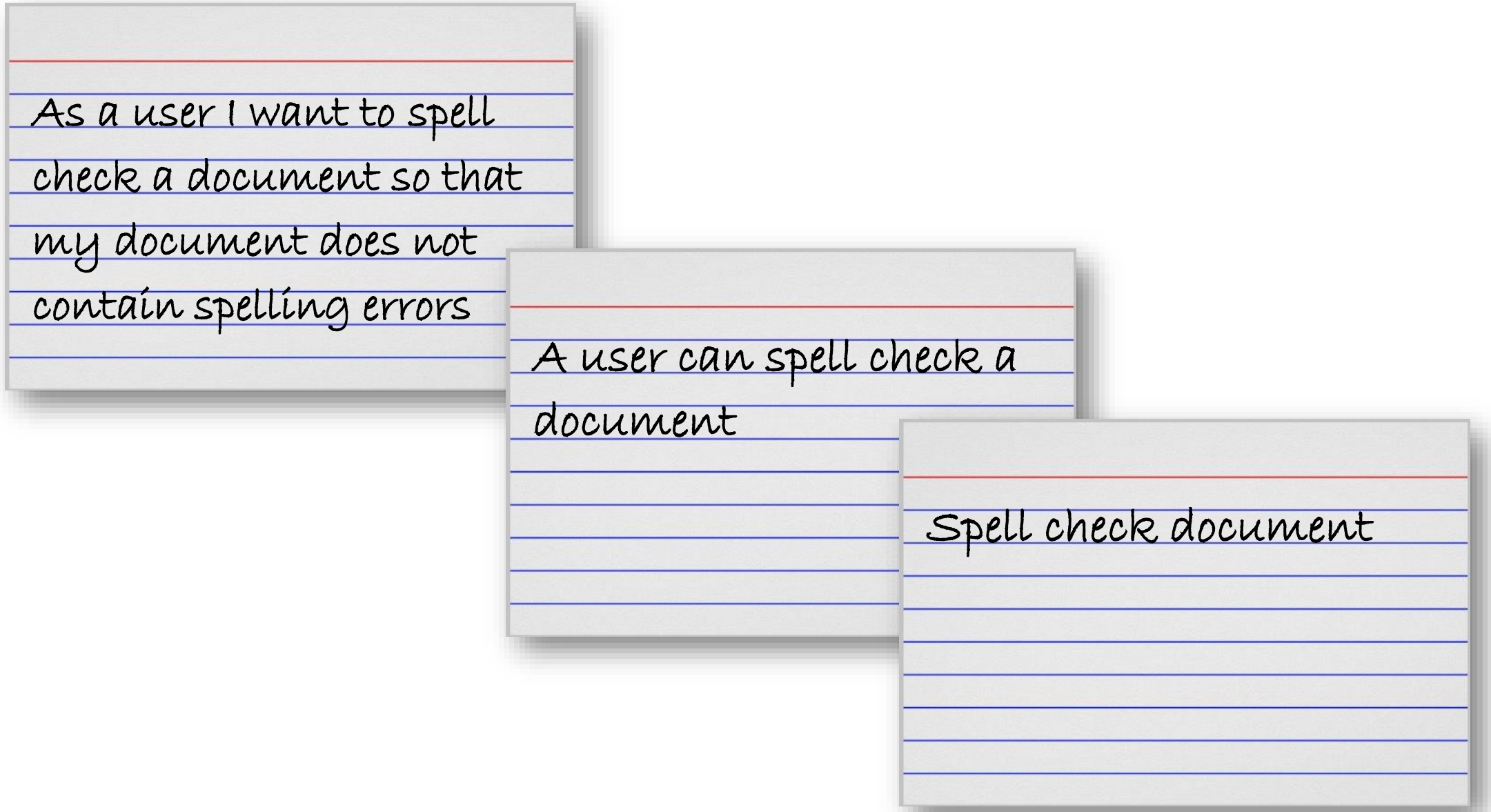
- Format:

  As a type of user,

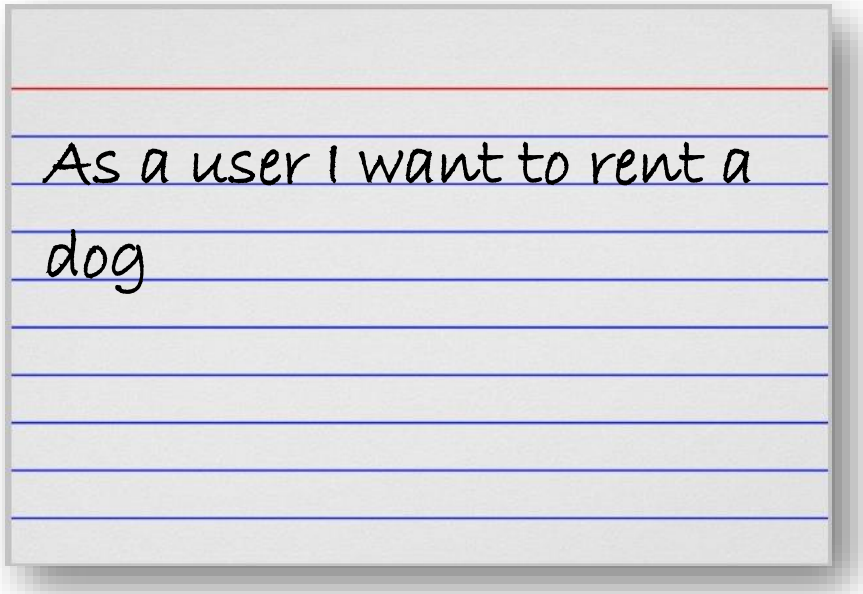
  I {**want** / **can** / **need** / **am required to**} <some goal>

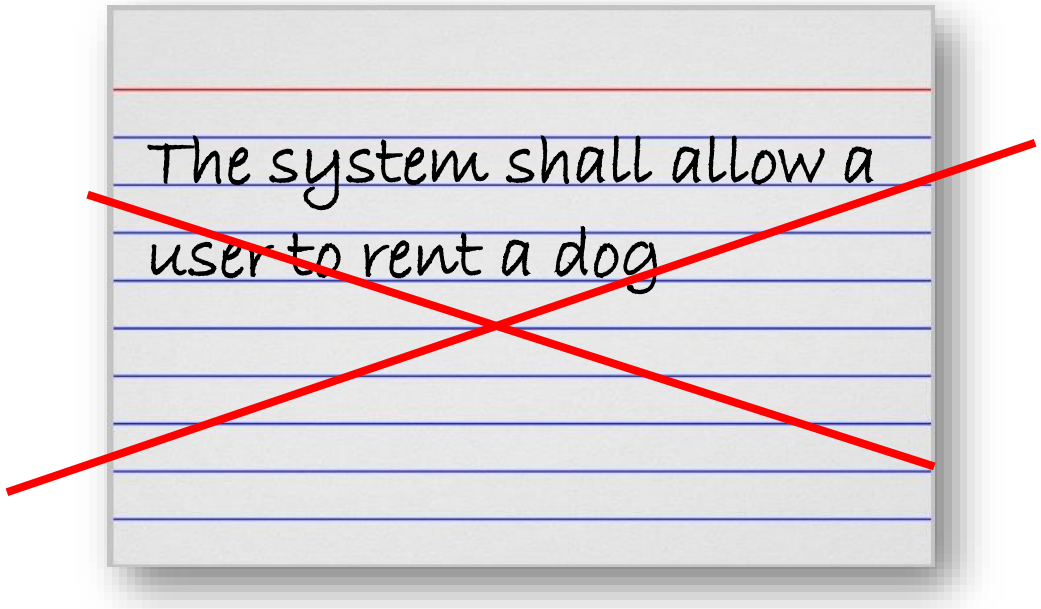
  so that <some reason>

User stories



User stories





INVEST mnemonic

- ***I****ndependent* – allow to reprioritize in any order
- ***N****egotiable* – discuss and make tradeoffs
- ***V****aluable* – clear business benefits
- ***E****stimable* – team is able to estimate the effort
- ***S****mall* – easier to estimate and test
- ***T****estable* – how do you know when it is done?

Independent

As a customer I can pay for the items in my cart with Visa

1

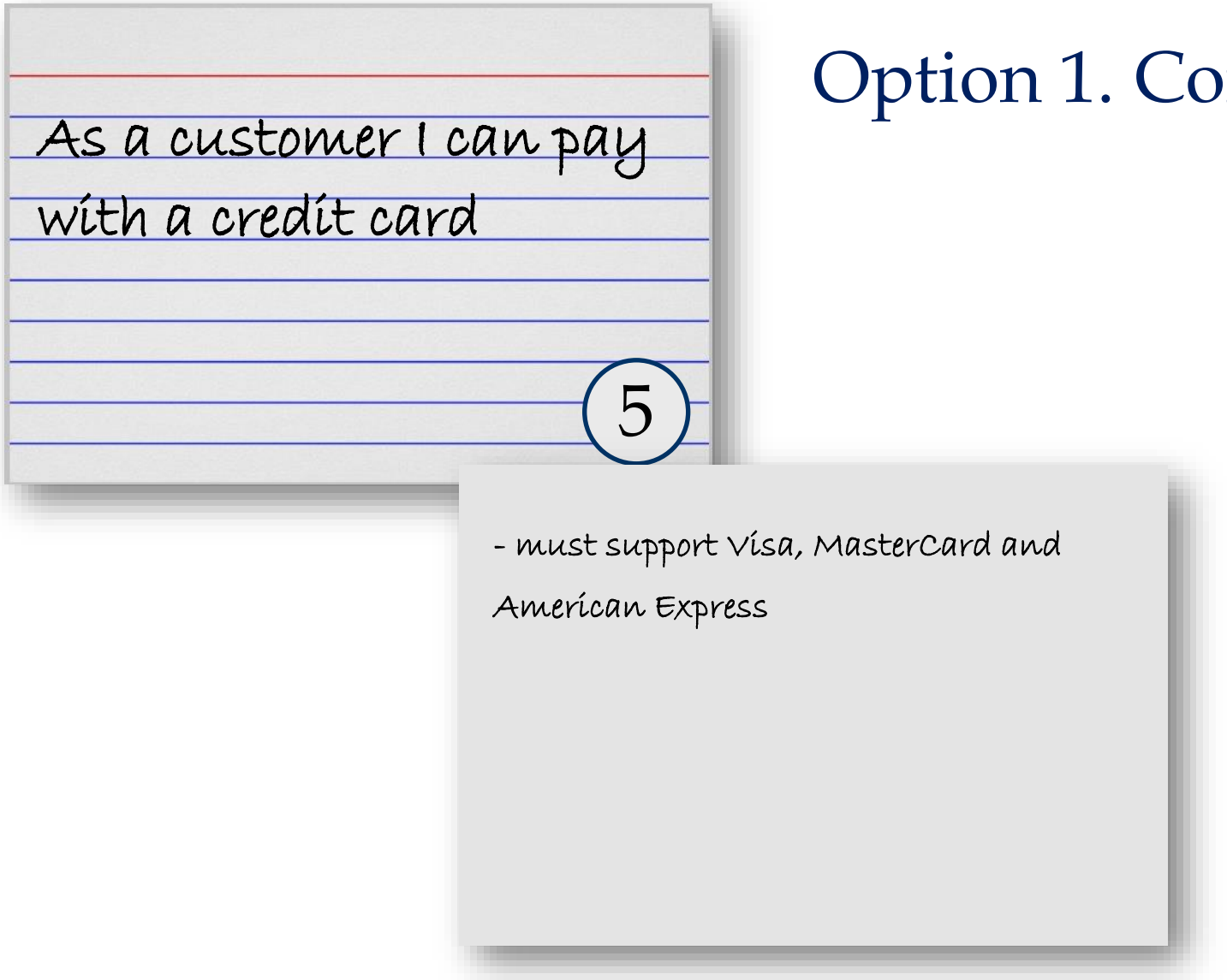
As a customer I can pay for the items in my cart with a MasterCard

1

As a customer I can pay for the items with an American Express card

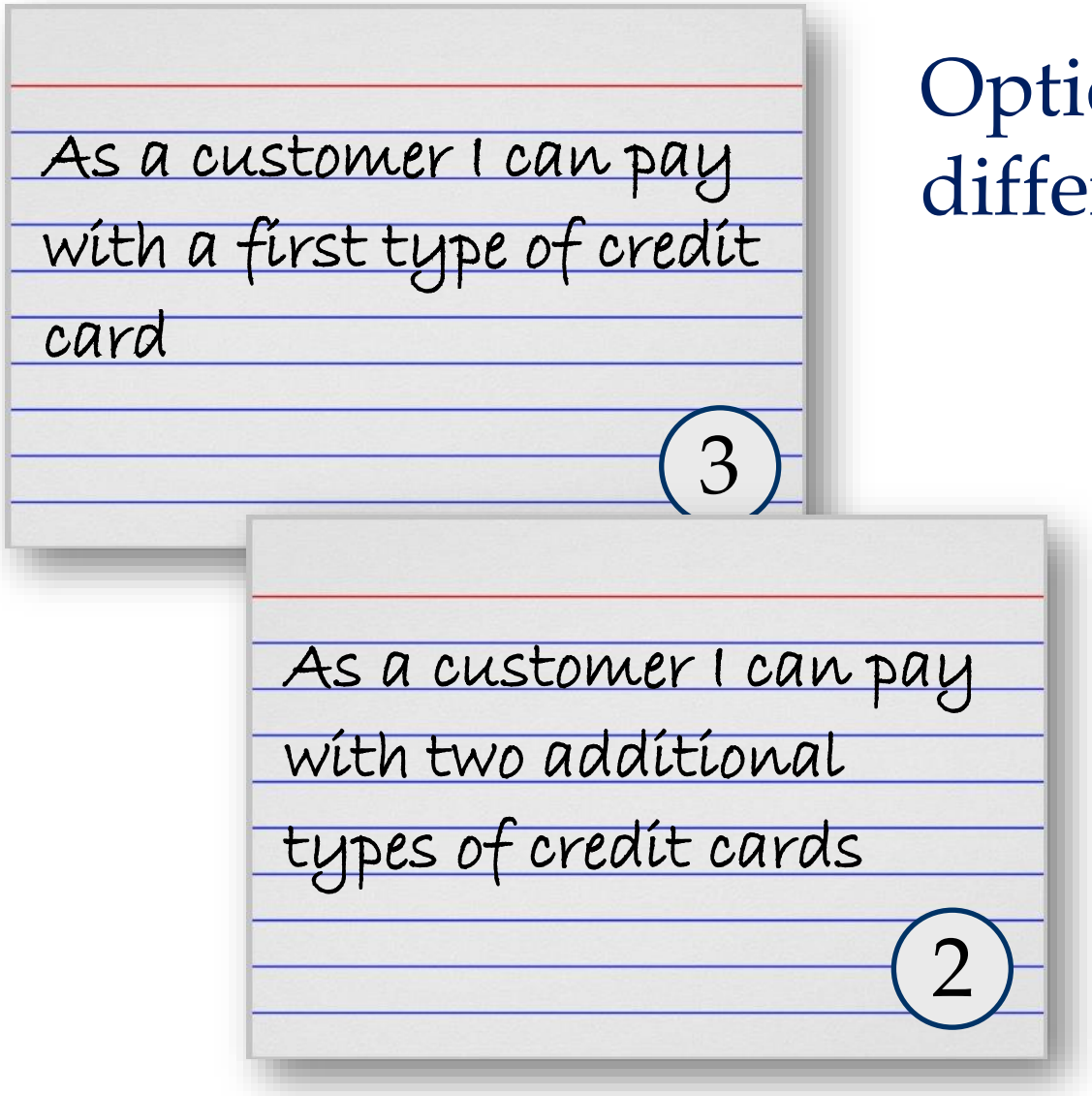
3

Independent



Option 1. Combine the stories

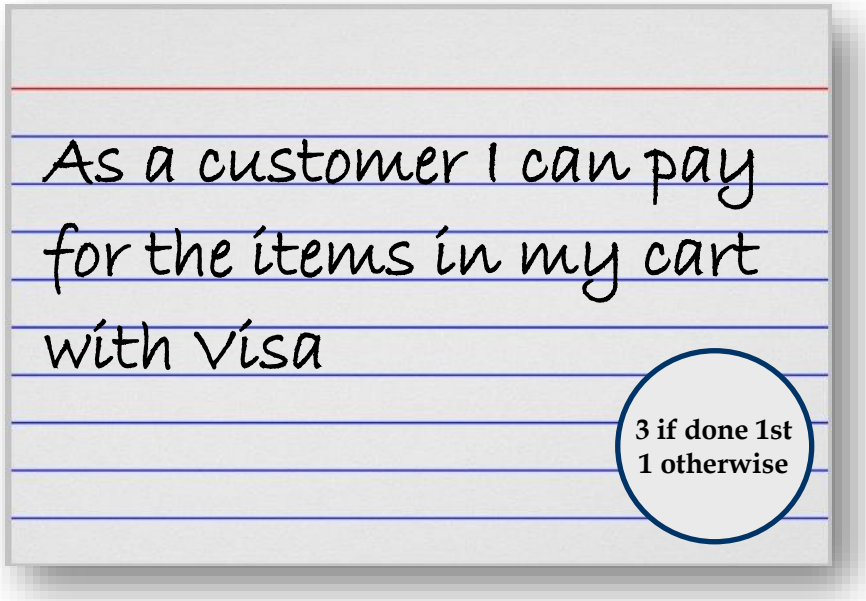
Independent

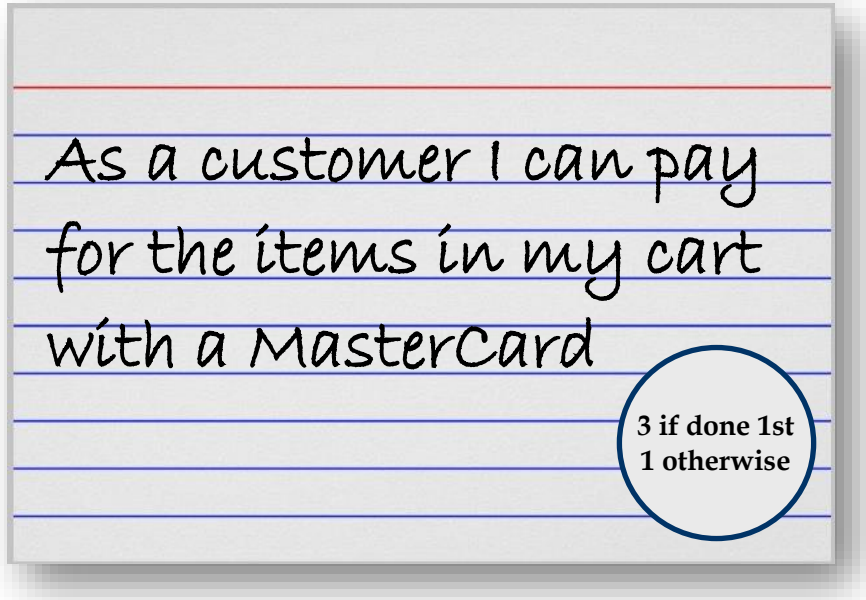


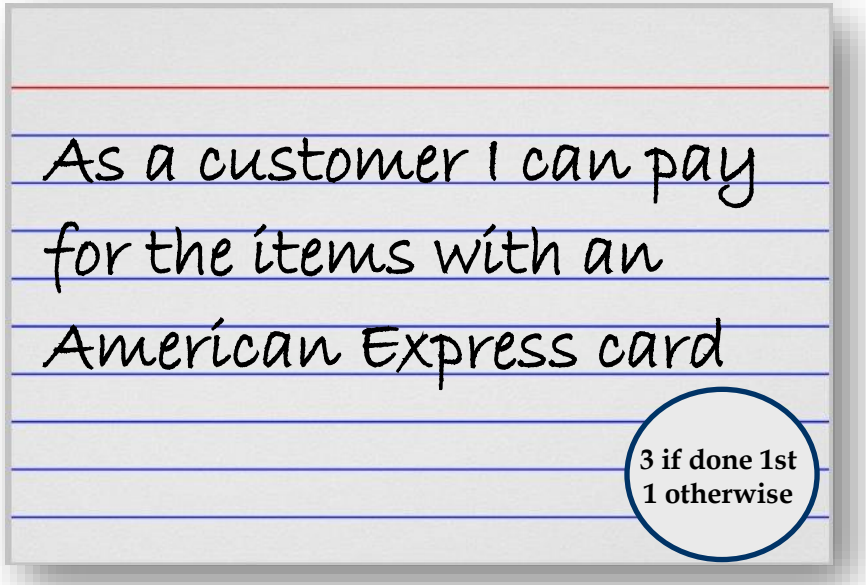
Option 2. Split across a different dimension

Independent

Option 3. Write two estimates

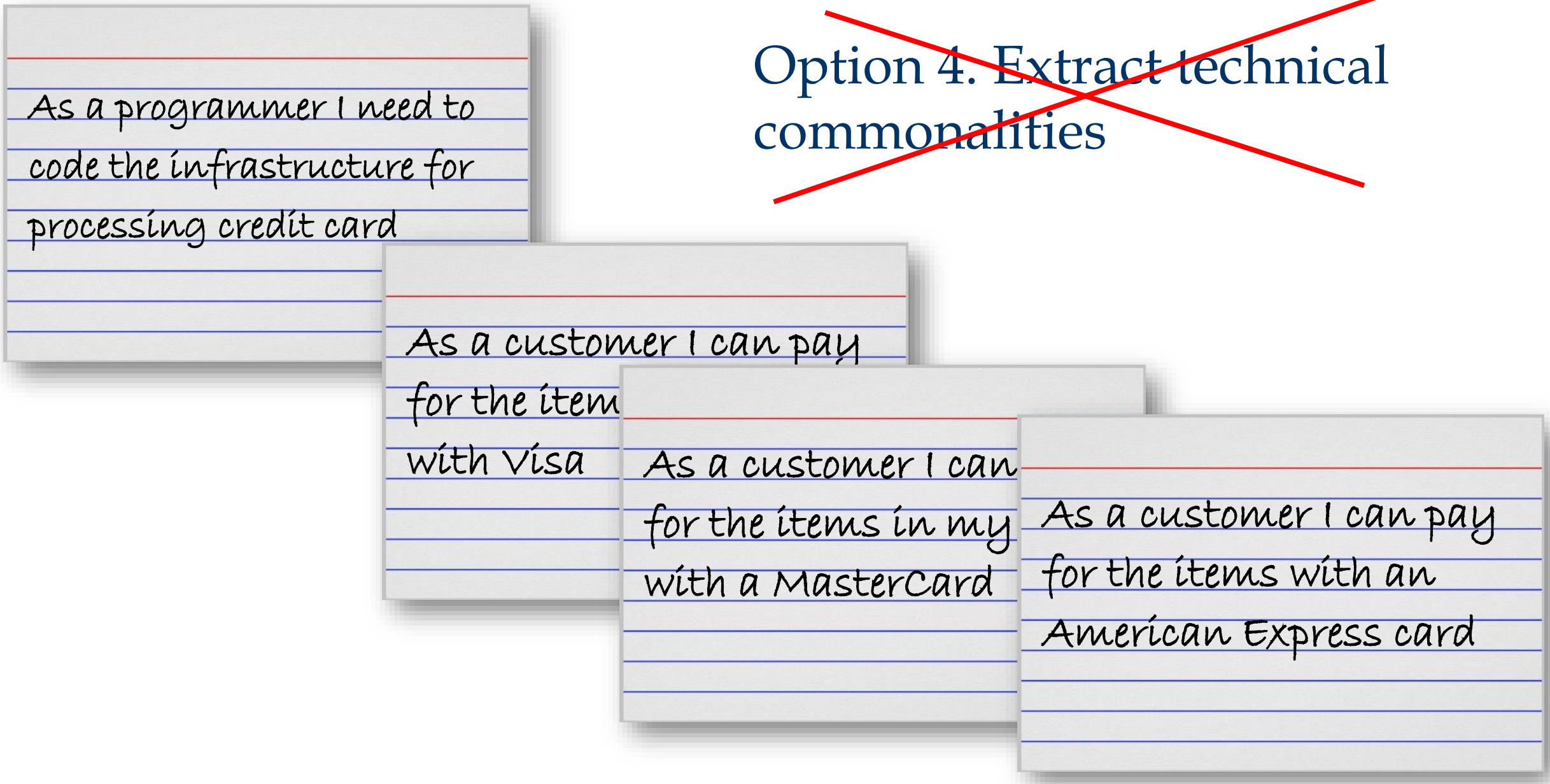




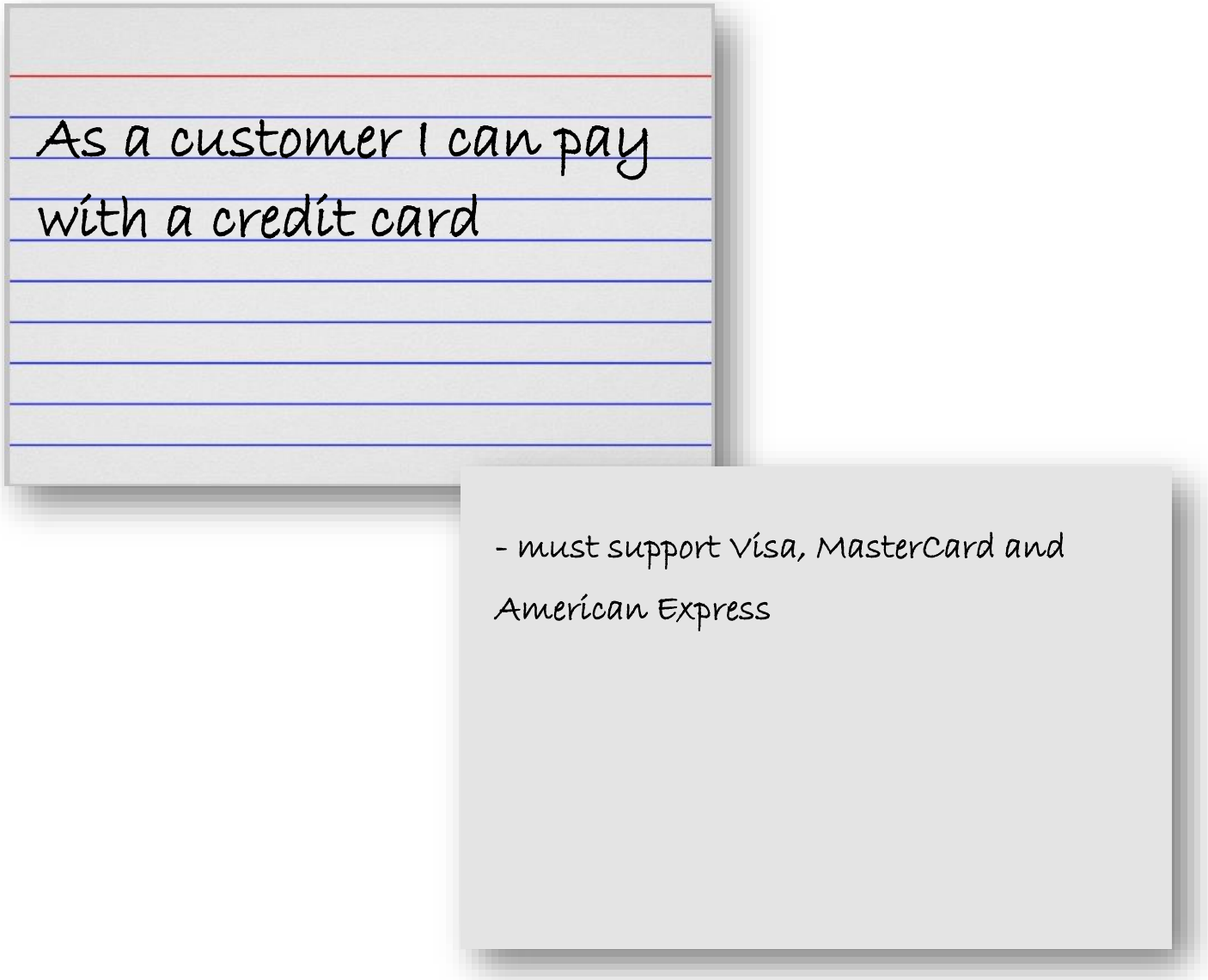


Independent

~~Option 4. Extract technical commonalities~~



Negotiable


As a customer I can pay with a credit card
- must support Visa, MasterCard and American Express

Valuable

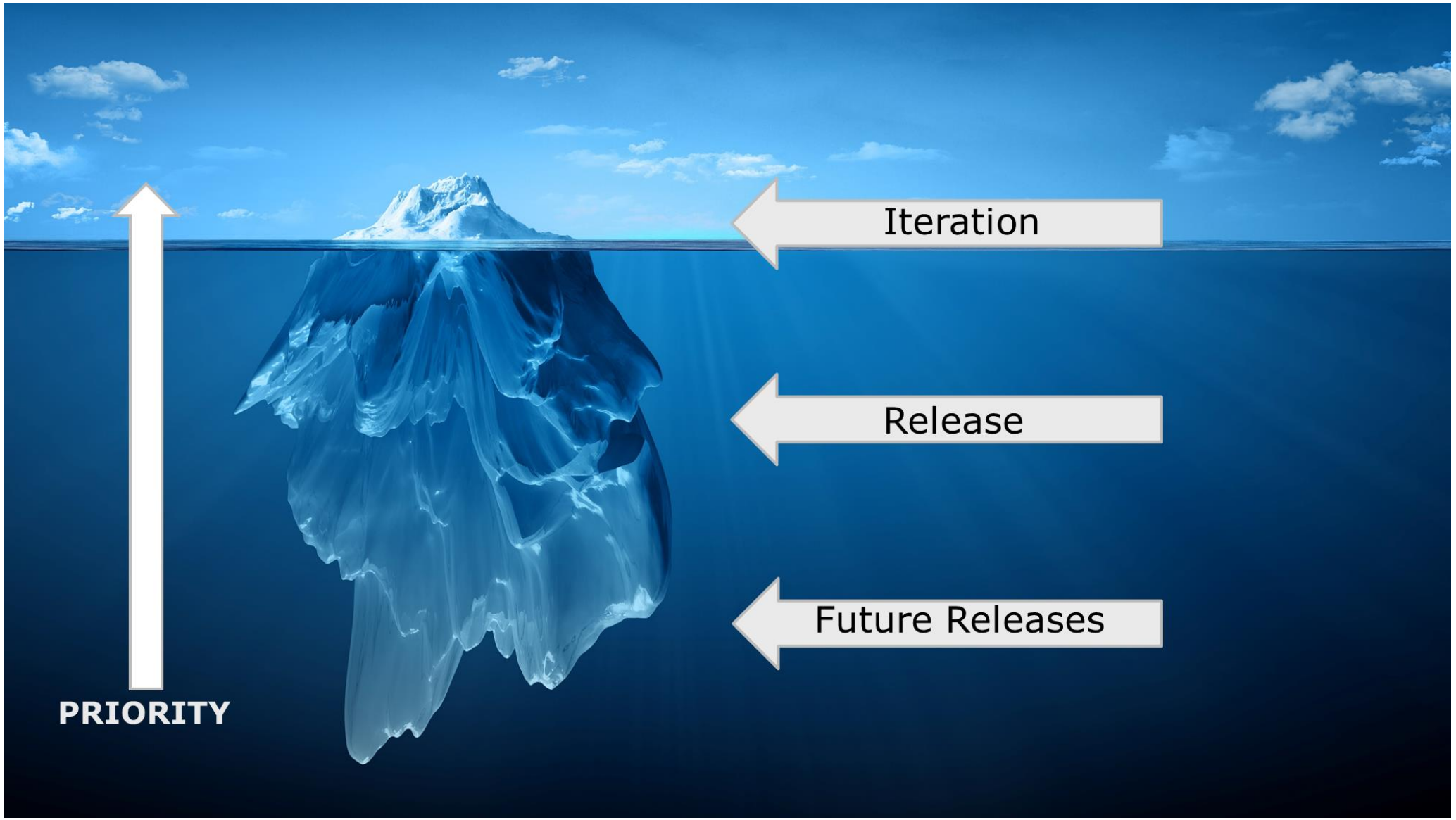
As a parent, I want to restrict my child to only using apps appropriate for children of a certain age

As a system administrator, I want all configuration information for all users stored in a central location

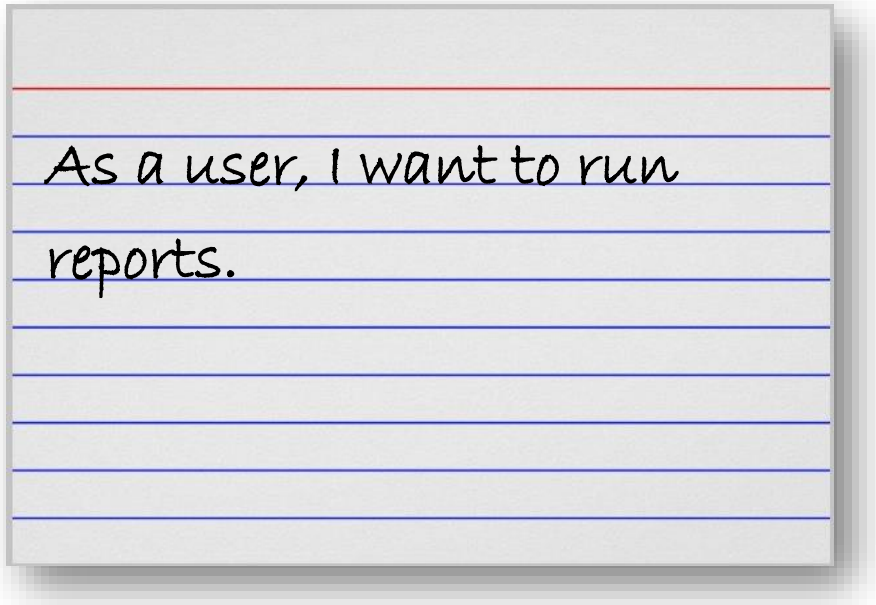
Estimable / Estimatable

- The story is too big
- Too much is unknown about the story
- Developers lack domain knowledge
- Developers lack technical knowledge

Small / Sized appropriately

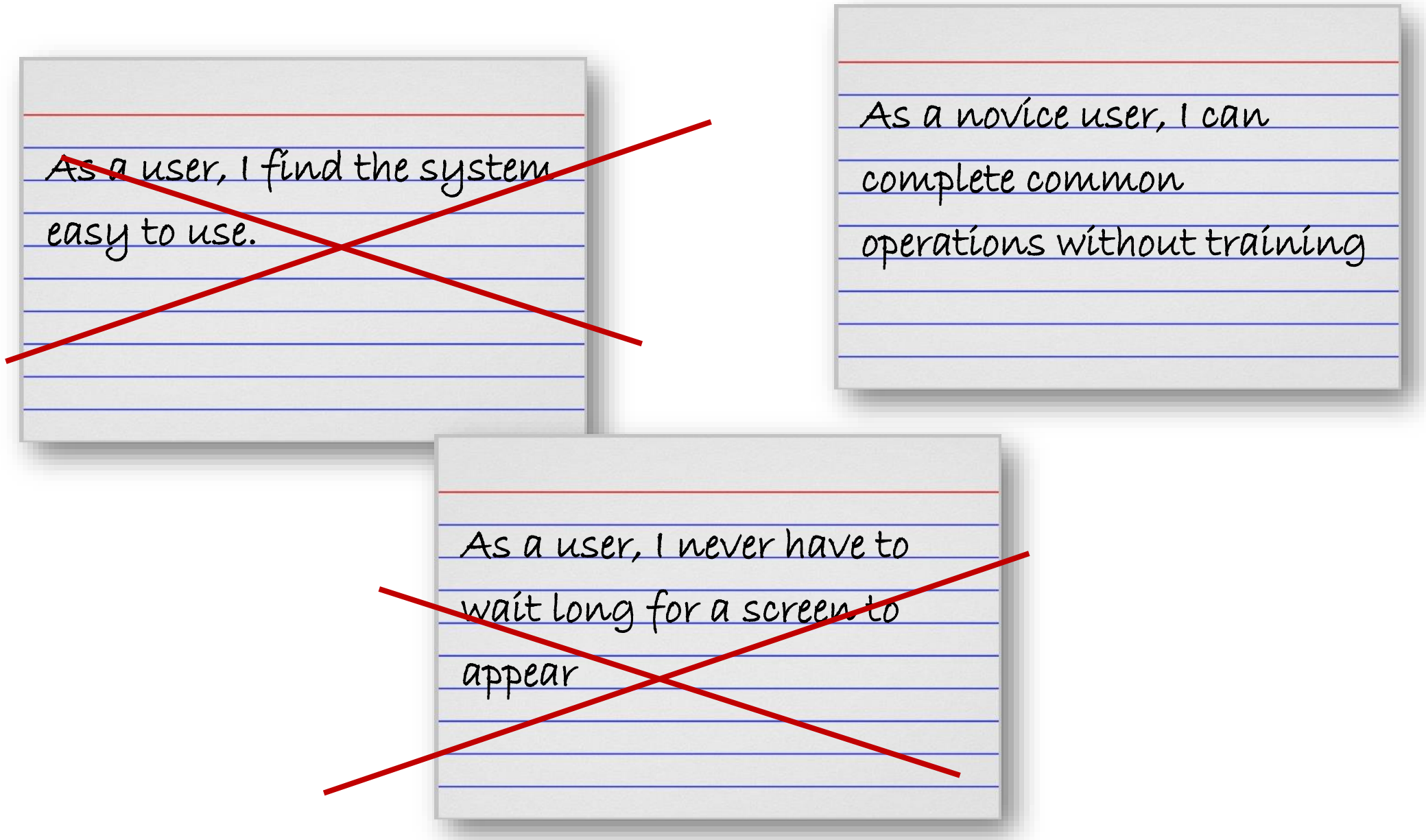


Epics

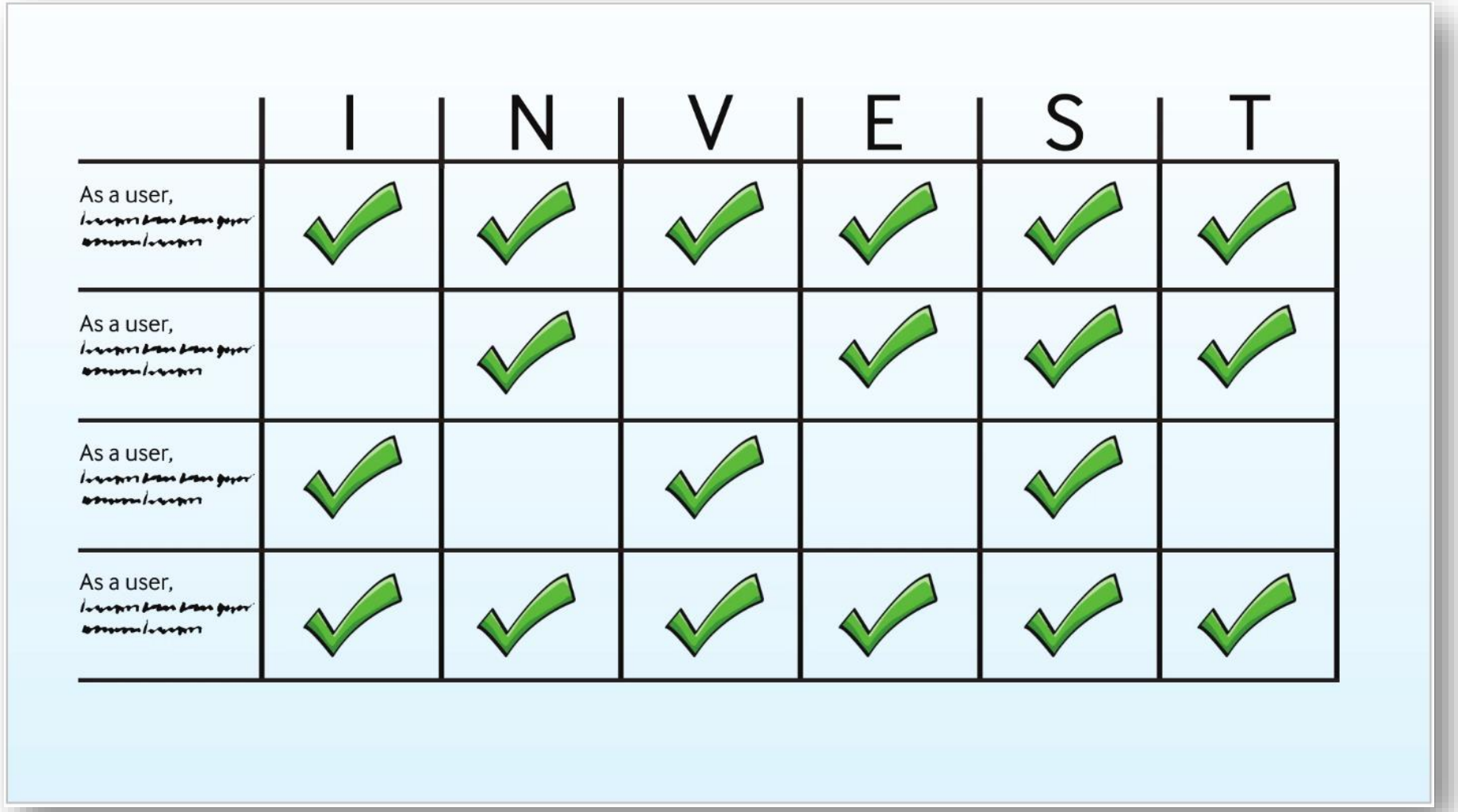


a story that is bigger than one team can do in one iteration

Testable


As a user, I find the system easy to use.
As a novice user, I can complete common operations without training
As a user, I never have to wait long for a screen to appear

INVEST mnemonic



When to write user stories?

- Randomly, whenever a new idea occurs
- During sprint review meetings
- During product backlog refinement / grooming meetings,
- During story-writing workshops

When to write user stories?

- Minimum Viable Product (MVP)
  - the version of a product which allows a team to collect the maximum amount of information with the least effort
- Minimum Marketable Feature (MMF)
  - A chunk of functionality that delivers a subset of the requirements, and that is capable of returning value to the customer when released as an independent entity

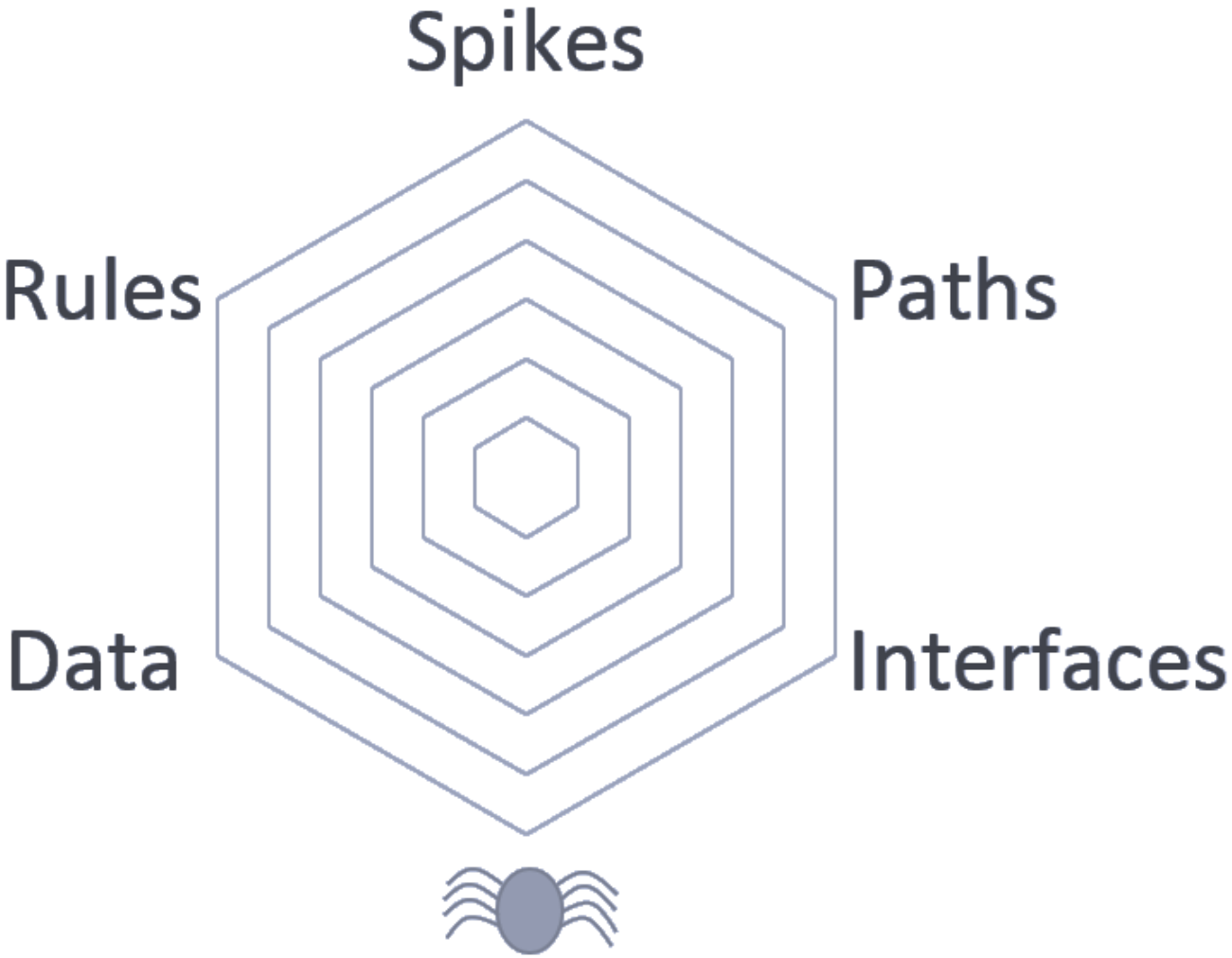
Splitting user stories

Lack of time

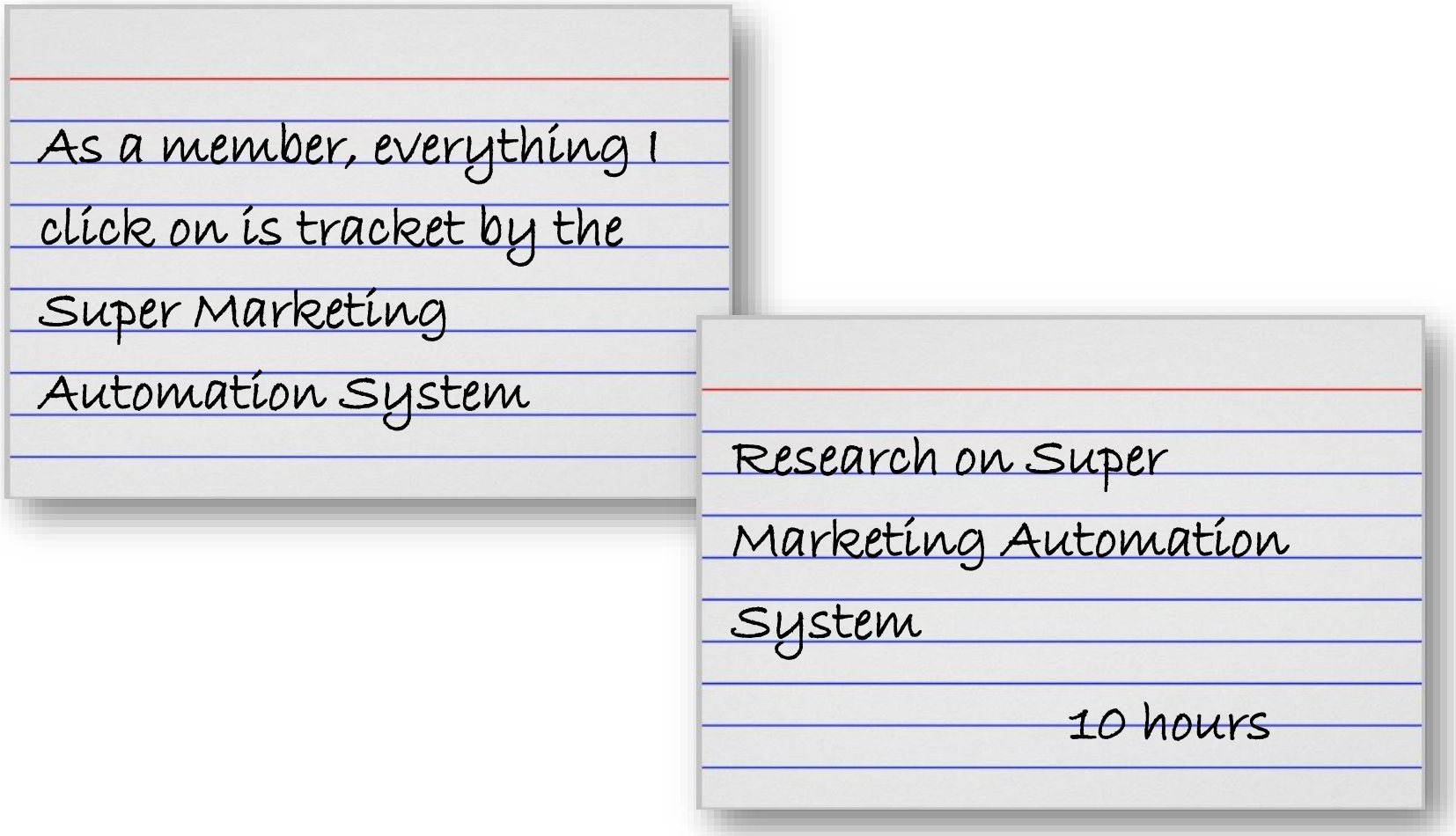
Different priorities

Too big

SPIDR approach to splitting user stories

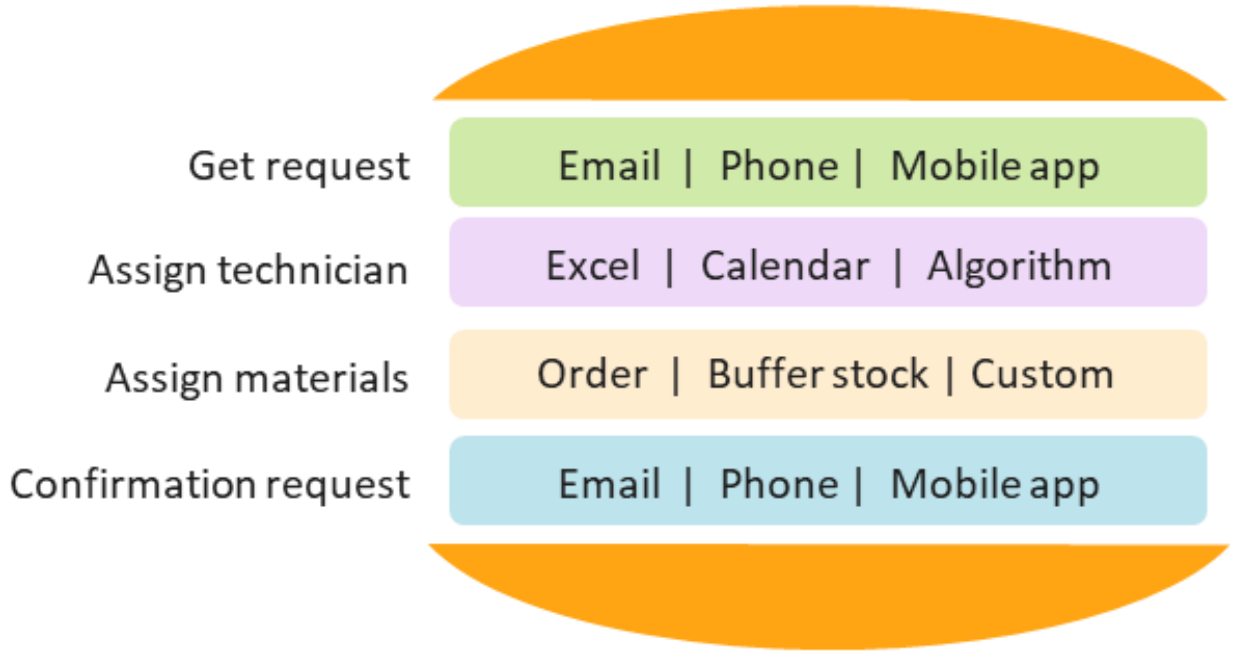


Splitting stories with spikes



Splitting stories along paths

As a user I want to be informed about my request so that I can manage my schedule



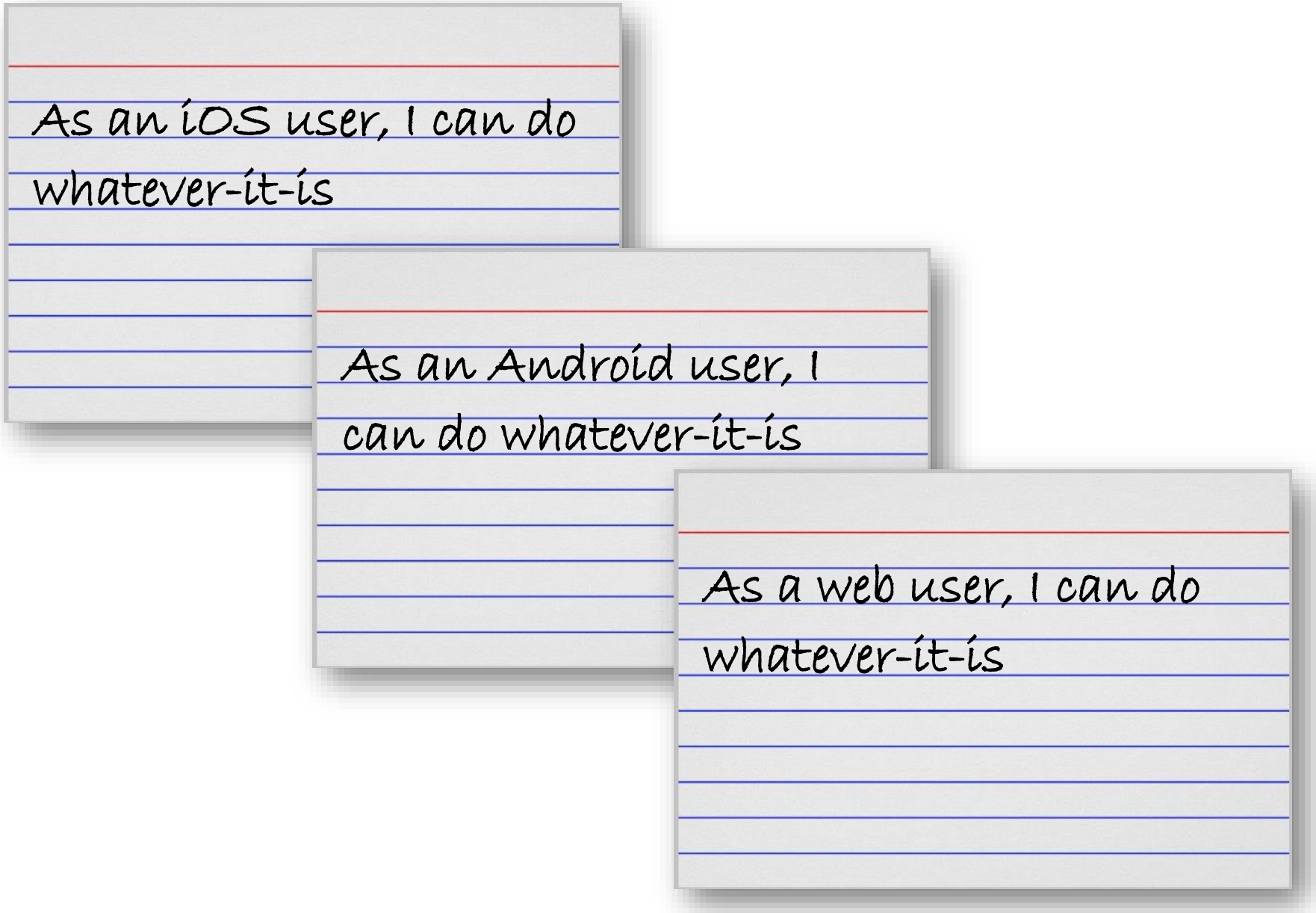
Hamburger technique

User story 1 - First "bite"
Phone call, assign technician using Excel, order materials, email confirmation

User story 2 - Second "bite"
Phone call, assign on calendar, assign materials ased on a buffer stock, email confirmation

User story 3 - third "bite"
etc

Splitting stories by interface

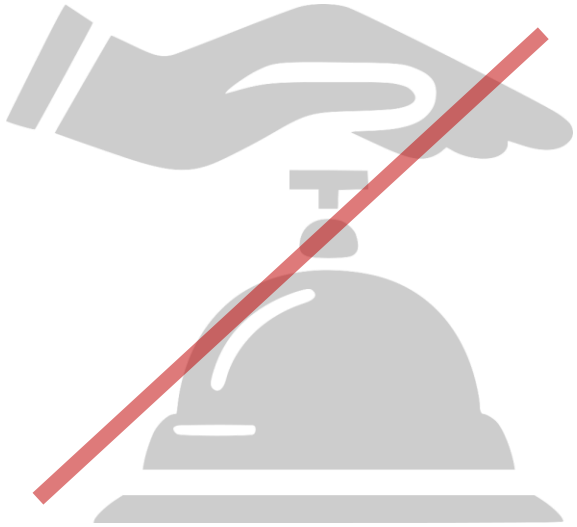


Splitting stories by data

As a passenger I want to look for the most suitable flights so that I can reach the destinations I choose in optimal conditions

Splitting stories by business rules

As a user I want to cancel my intervention requests so that I can give up the requested services



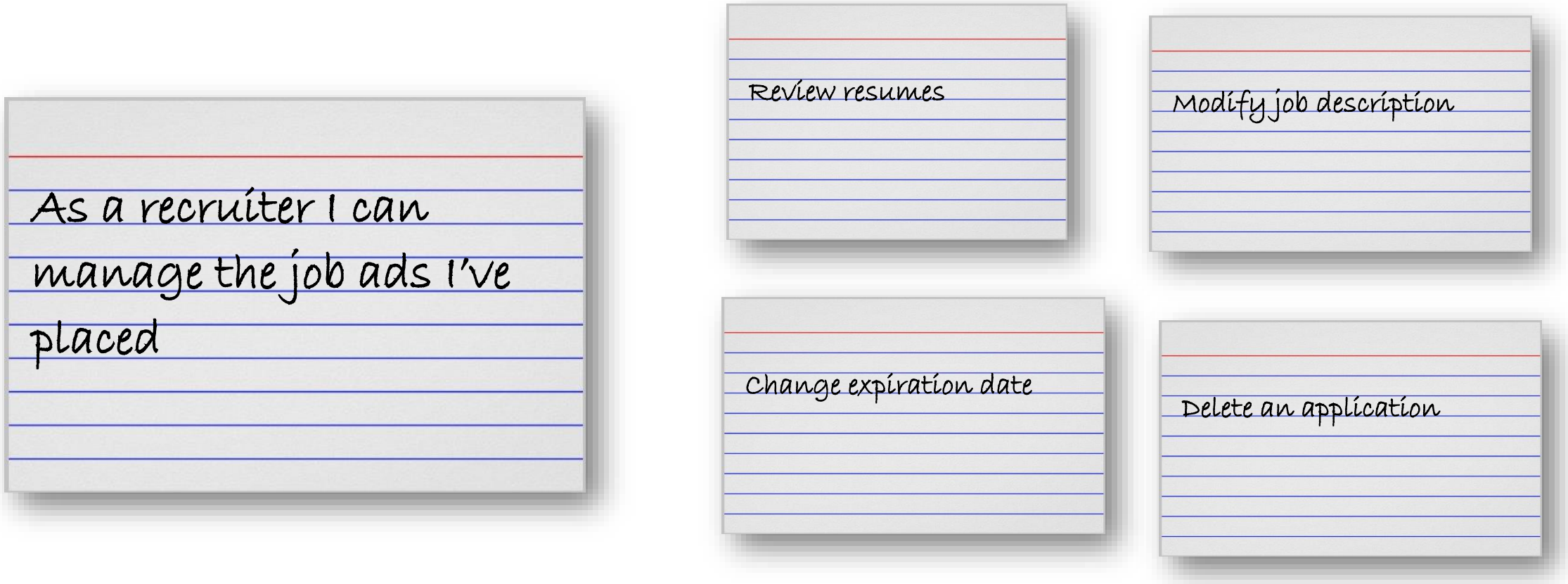
User story 1 – Cancel a request with no restrictions
Simply cancels the request

User story 2 - Cancel a request with an assigned technician
Cancel request and change technician allocation

User story 3 – Cancel a request after materials are ordered
Cancel the request and manage the financial implications

Closed stories

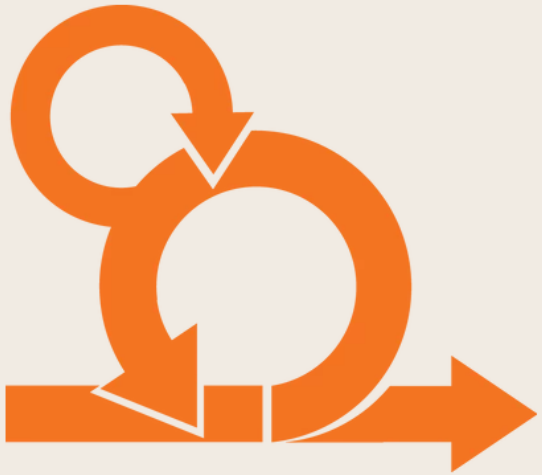
A closed user story is one that finishes with the user having achieved a meaningful goal.



Three steps when you can't split a story

1. Try harder
2. Let it take more than one iteration
3. Feel guilty

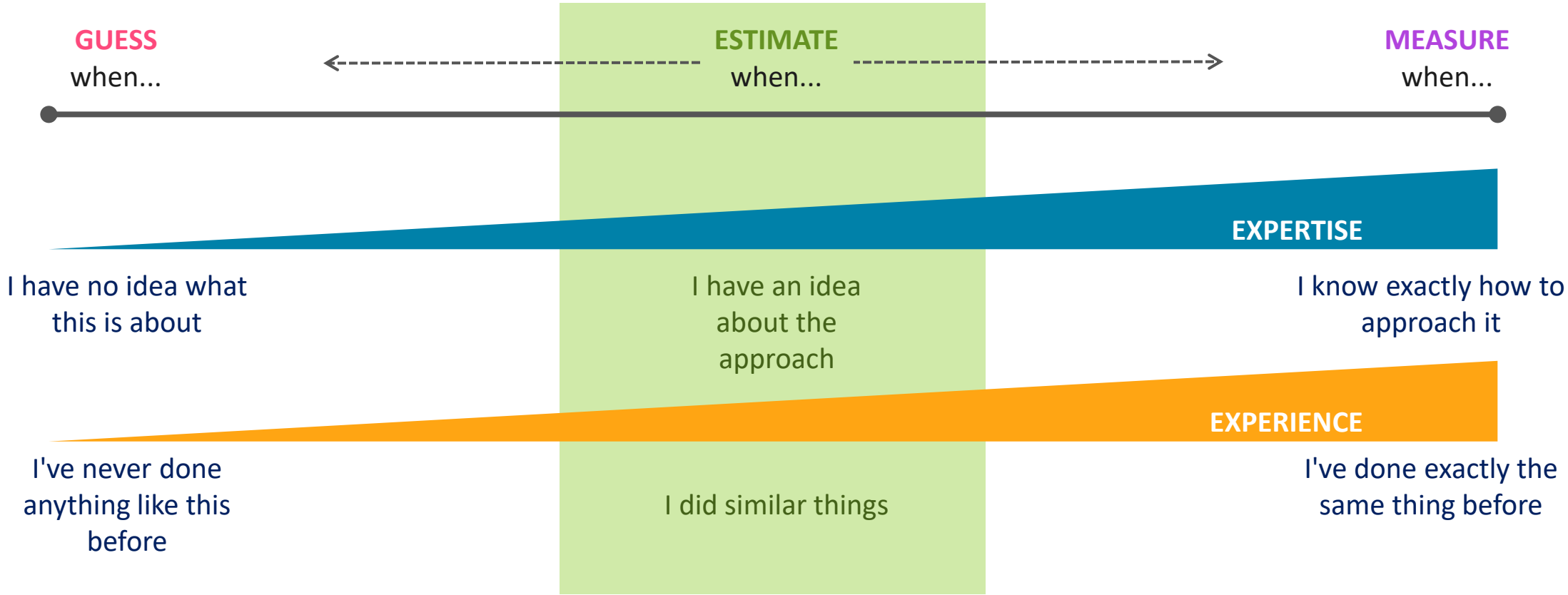
Source: Mike Cohn, Splitting user stories



AGILE SOFTWARE DEVELOPMENT

Estimations in Agile

What is estimation?

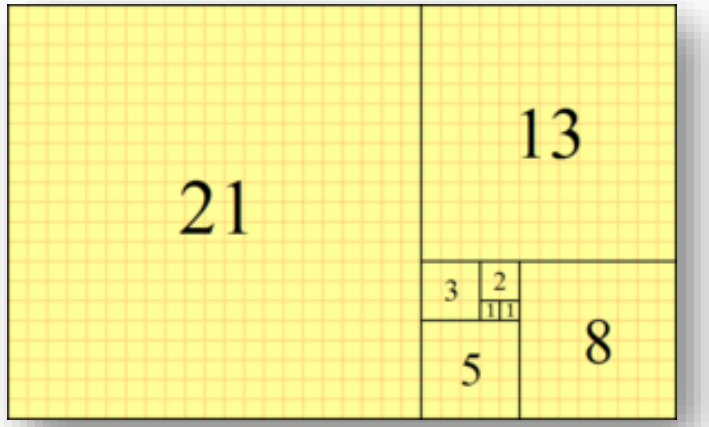


Cambridge Dictionary: *a guess or calculation about the cost, size, value, etc. of something*

Story points

- How long a user story will take (effort) in relative.
- Influenced by complexity, uncertainty, risk, volume of work etc

Relative sizing and story points



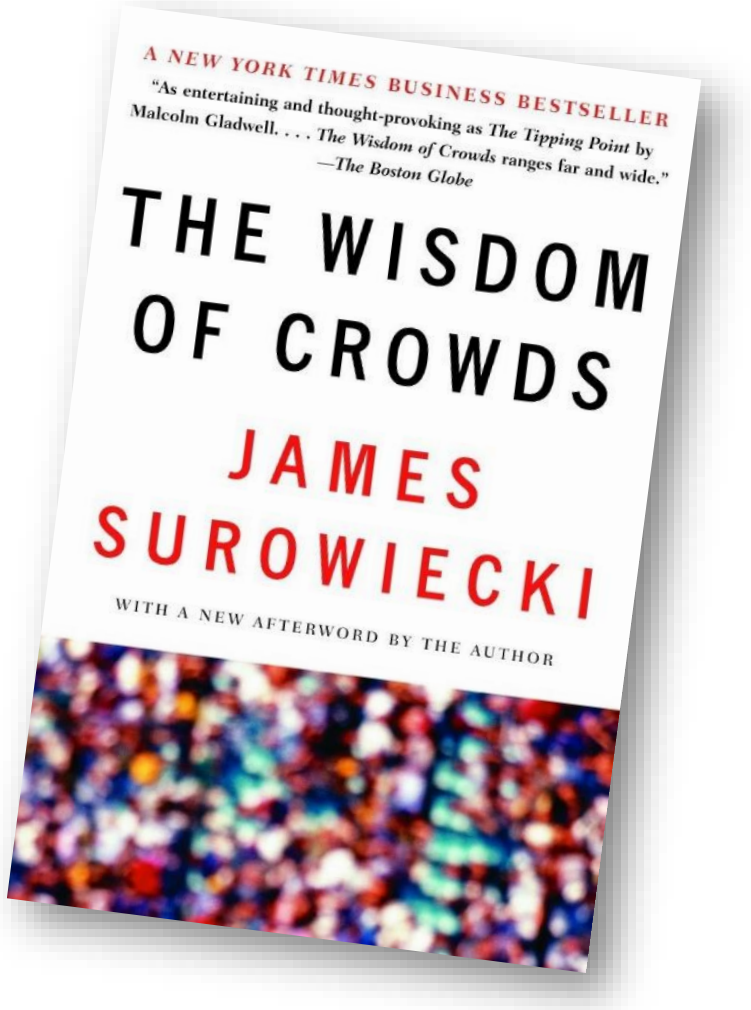
- Estimation technique that ranks stories by their size relative to each other and estimates based on those rankings.
- Story points are typically expressed using the Fibonacci sequence of numbers
  - Using this technique the estimation would be quick because one story it not evaluated from the scratch, it is evaluated by its position relative to other stories
  - It is useful when planning for the next iteration to decide which stories can be completed in that iteration
- Guidelines:
  - The team should own the definition of story points
  - Story points definition should be all-inclusive
  - Point sizes should be relative
  - When disaggregating, totals don’t need to match
  - Estimate should include complexity, effort and risk

Poker planning

- Planning poker is a consensus-based estimation technique, mostly used to estimate effort or relative size of user stories. After each player has selected a card, all cards are exposed at once and consensus is reached in steps.
- Advantages:
  - Minimizing the “bandwagon effect” (grouping around most popular opinion)
  - Preventing HIPPO decision making (Highest-Paid Person Opinion)
  - Minimizing the “groupthink” effect (excessive concern for group harmony)

Wisdom of Crowds

- Diversity of opinion
- Independence
- Decentralization
- Aggregation
- Trust



Wisdom of Crowds

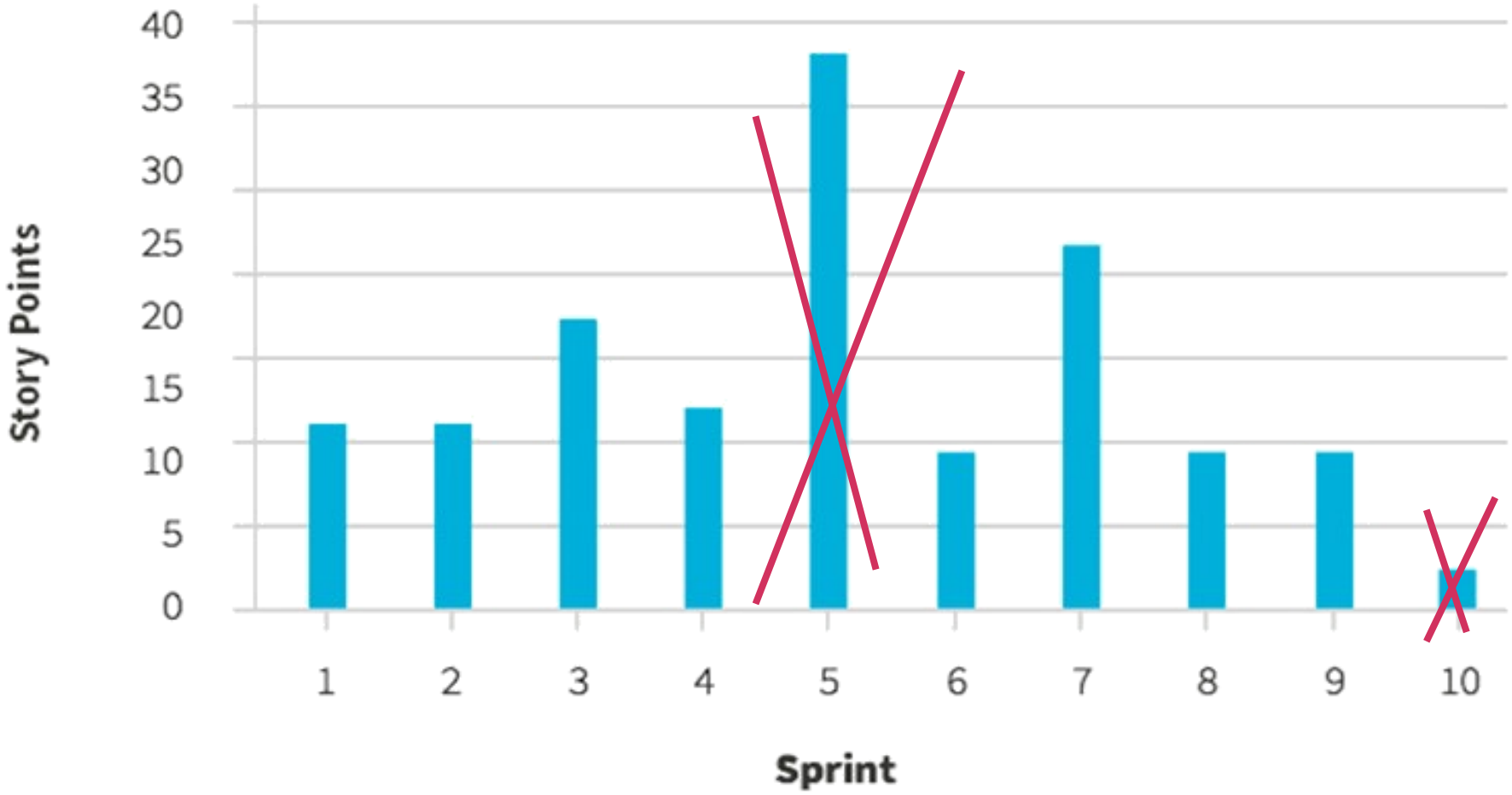
Planning Poker

- Diversity of opinion
- Independence
- Decentralization
- Aggregation
- Trust

Velocity

- The sum of the effort estimates associated with user stories that were completed during that iteration

Velocity



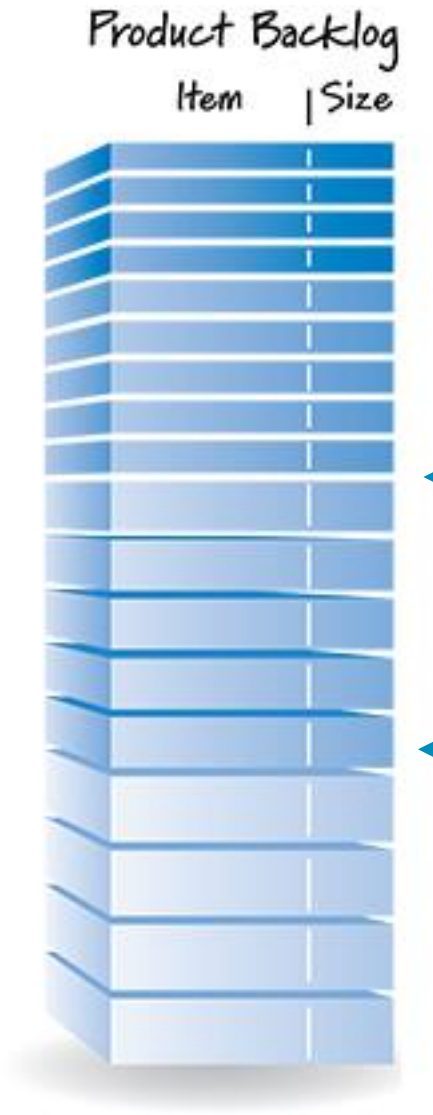
Velocity

Sorted velocities
27
34
35
38
39
40
40
41
45

90% confidence interval

No of historical iterations	Iterations to throw out from each end
0-7	0
8-10	1
11-12	2
13-15	3
16-17	4
18-20	5
21-22	6
23-25	7
26+	8

Extrapolate the velocity range



Will have

5%

5 * 34 = 170 story points – “We’ll be at least here in 5 sprints”

Might have

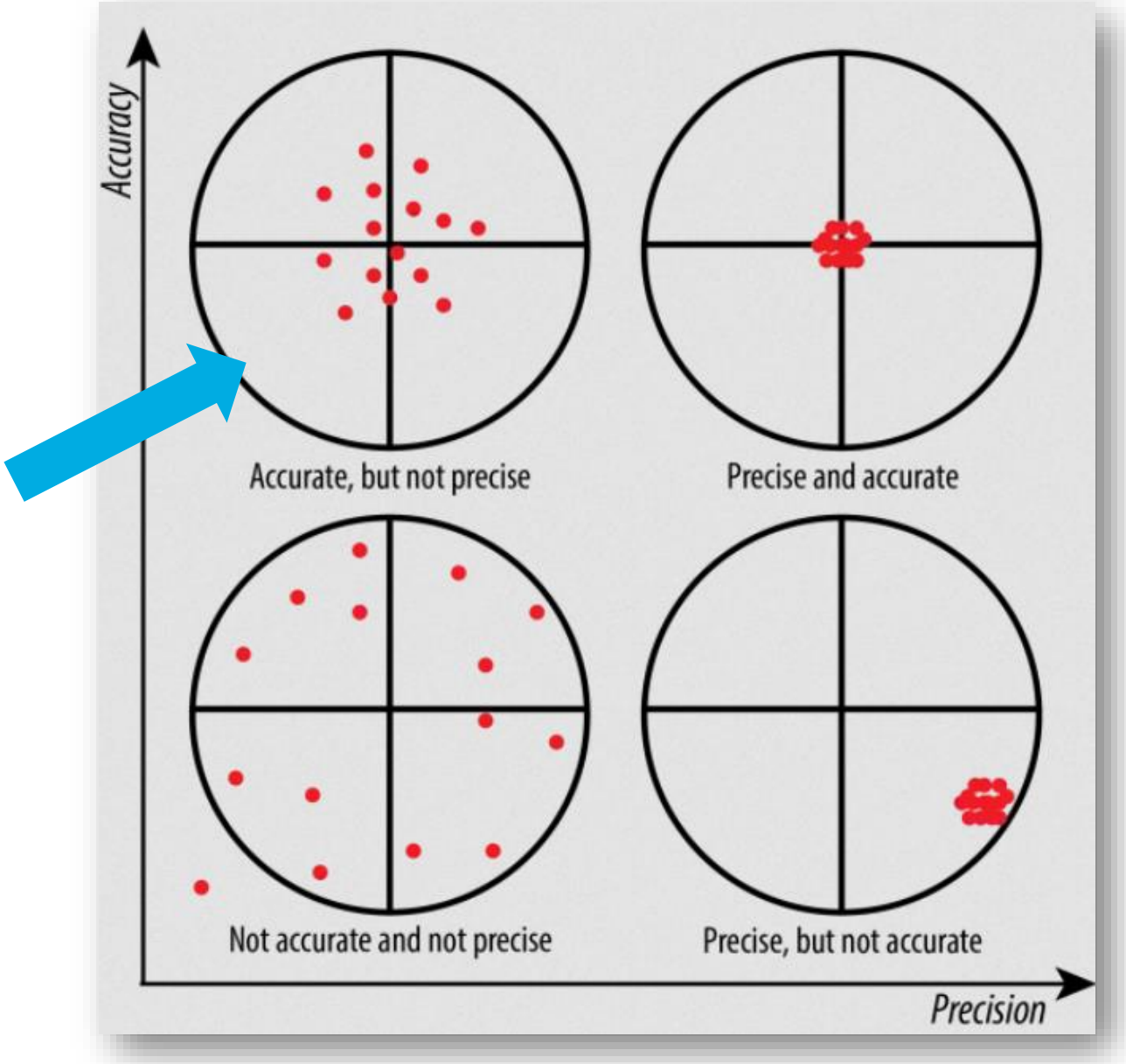
90%

5 * 41 = 205 story points – “We might be here in 5 sprints”

Won’t have

5%

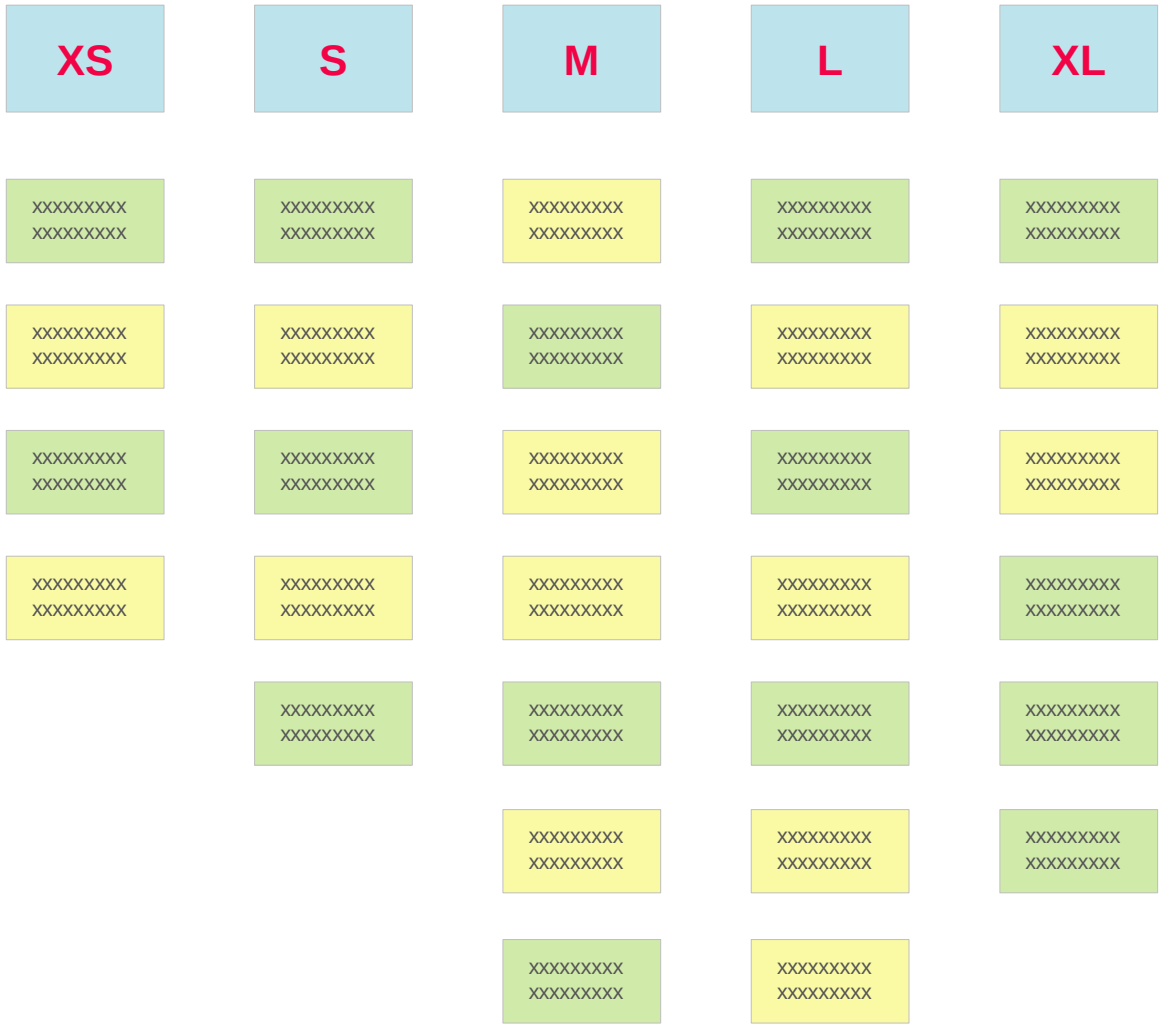
Accuracy vs Precision



"It is better to be roughly right than precisely wrong"

J.M. Keynes

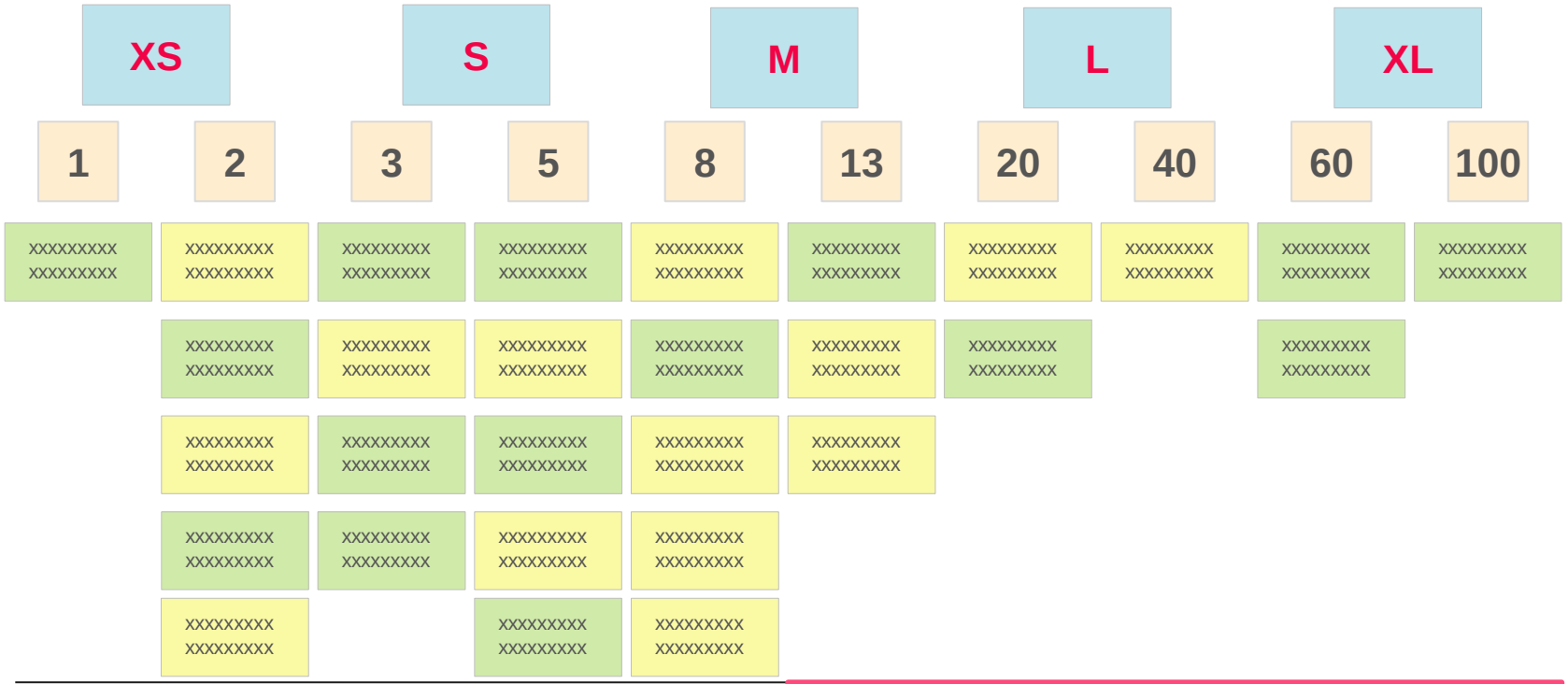
White Elephant Sizing



1. Prepare a set of user stories
2. Organize a workspace with t-shirt sizes as columns
3. Ask the project team to estimate each user story through placing it on a column
4. Answer the questions asked by the team, but don't go into too much detail

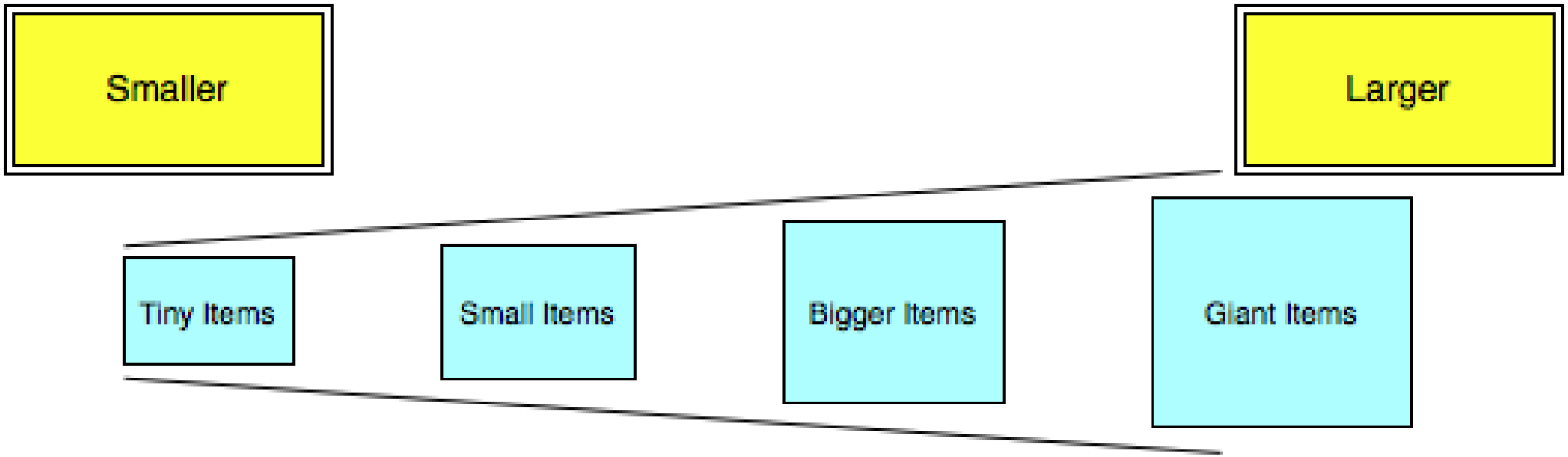
White Elephant Sizing

5. Prepare a second round using the Fibonacci sequence
6. Ask the project team to refine the estimates
7. Calculate the total size of the project user stories
8. Calculate the time estimated to complete using tolerance intervals

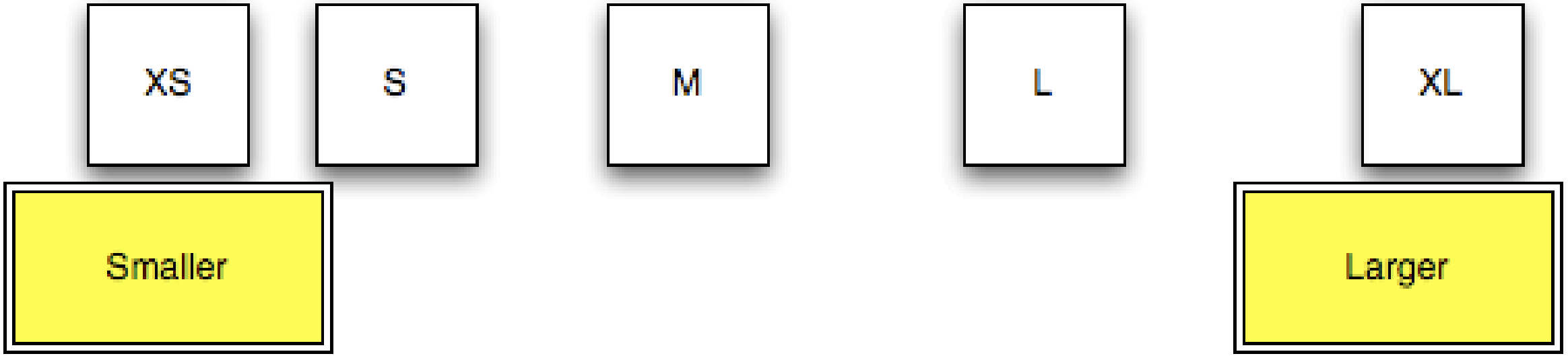


TOTAL **427 SP** **342 - 512 SP**

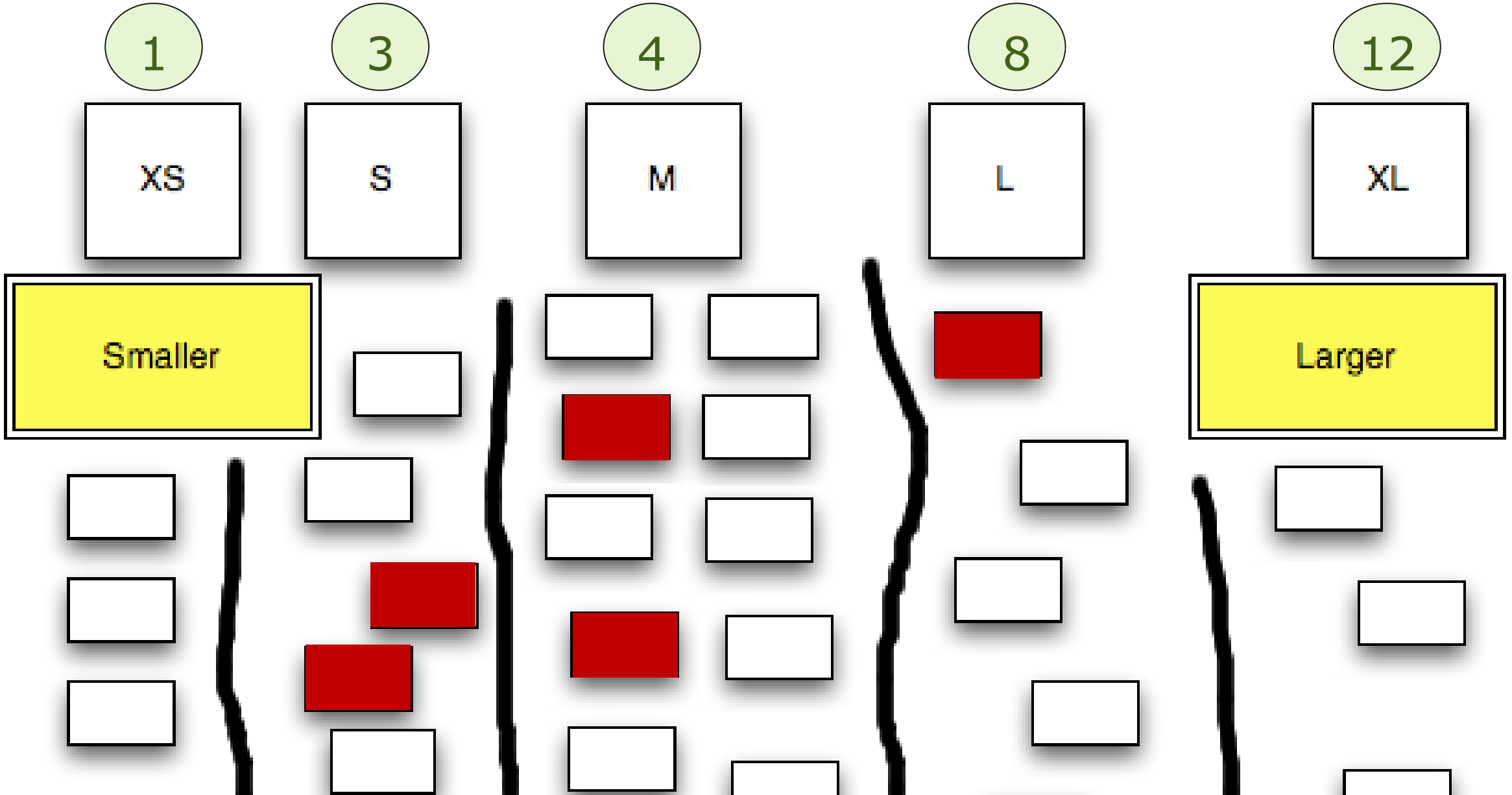
Affinity estimation



Affinity estimation



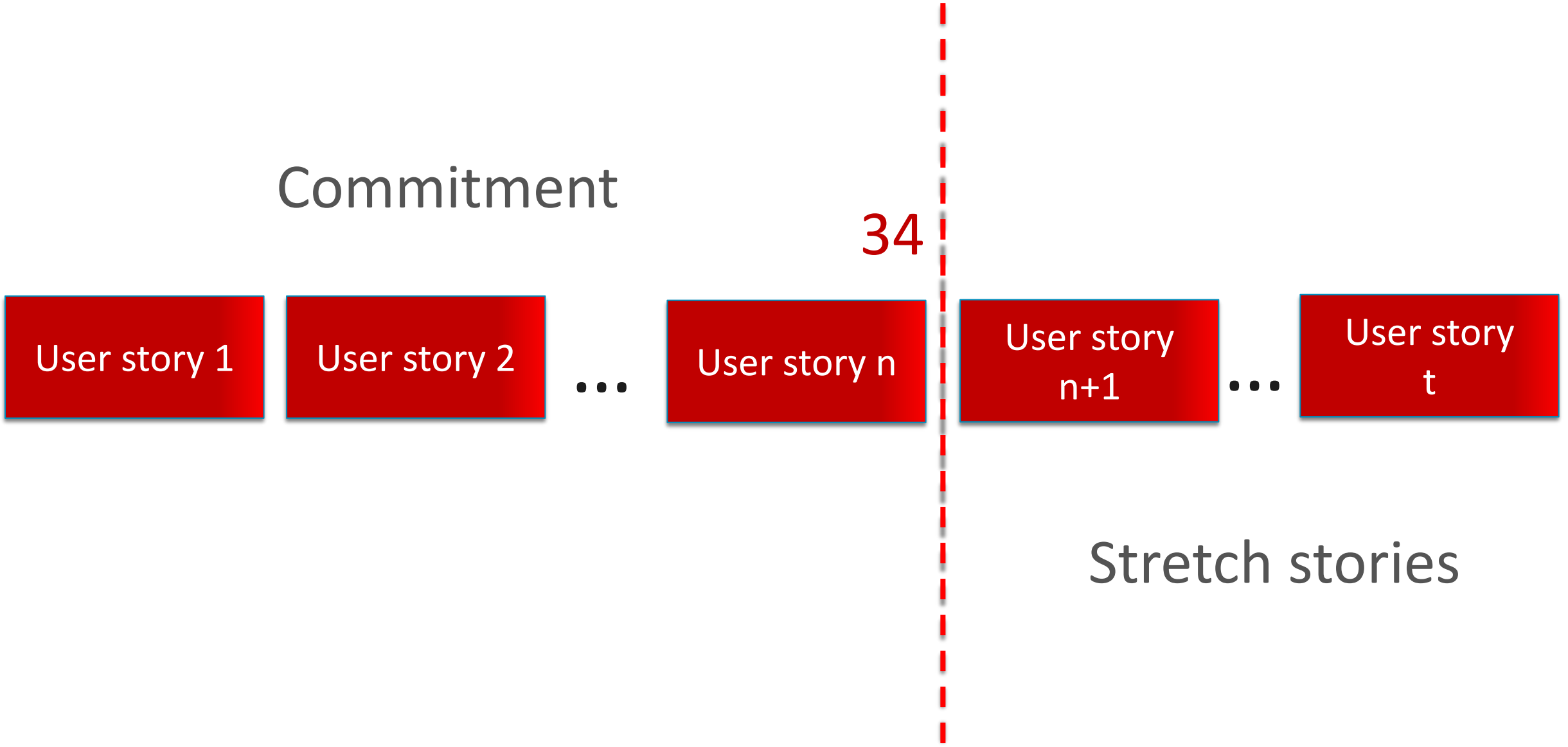
Affinity estimation

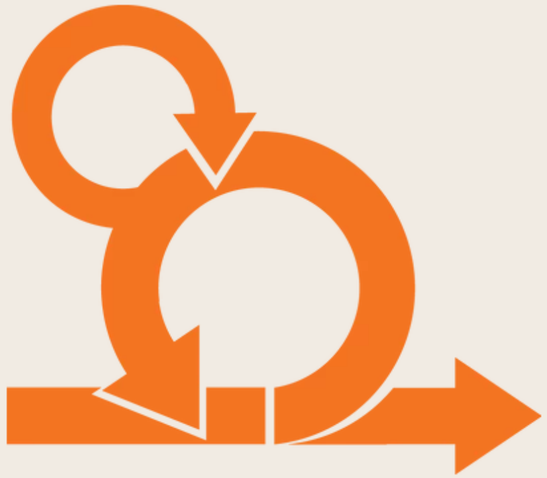


Velocity prediction

18

Velocity prediction





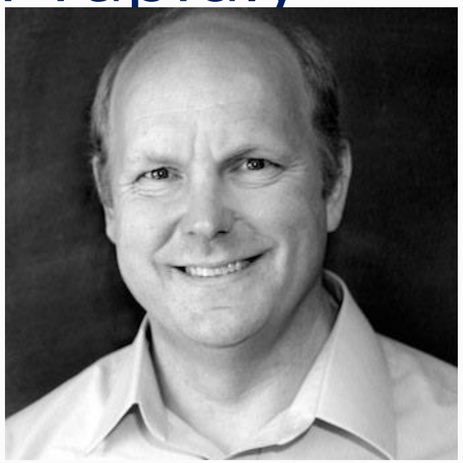
AGILE SOFTWARE DEVELOPMENT

eXtreme Programming 1

XP is…

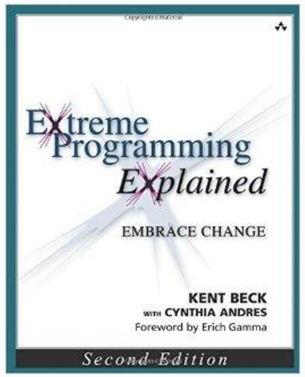
…. a lightweight software development methodology for small to medium sized teams developing software in the face of t vague or rapidly changing requirements”

Kent Beck



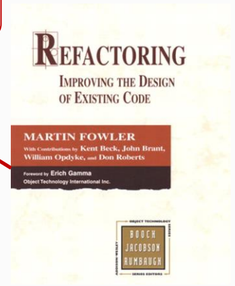
XP is…

…. a lightweight software development methodology

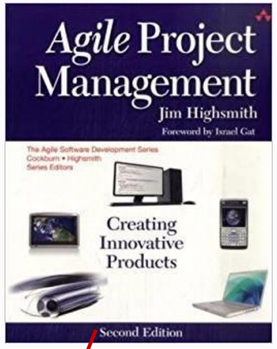


XP

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler

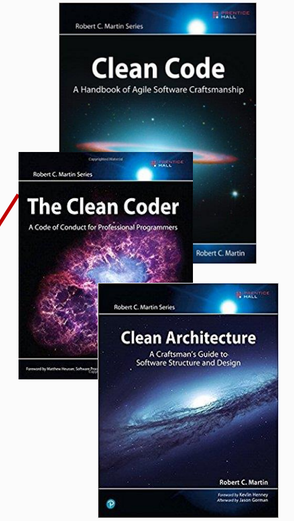


Crystal



James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick

Scrum

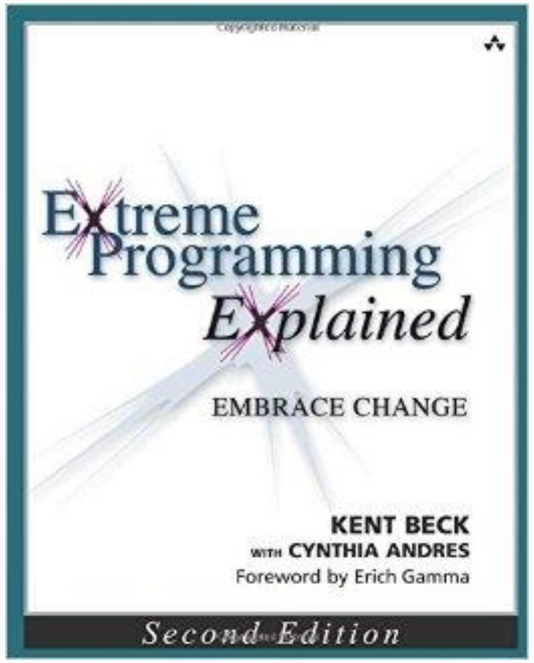


Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

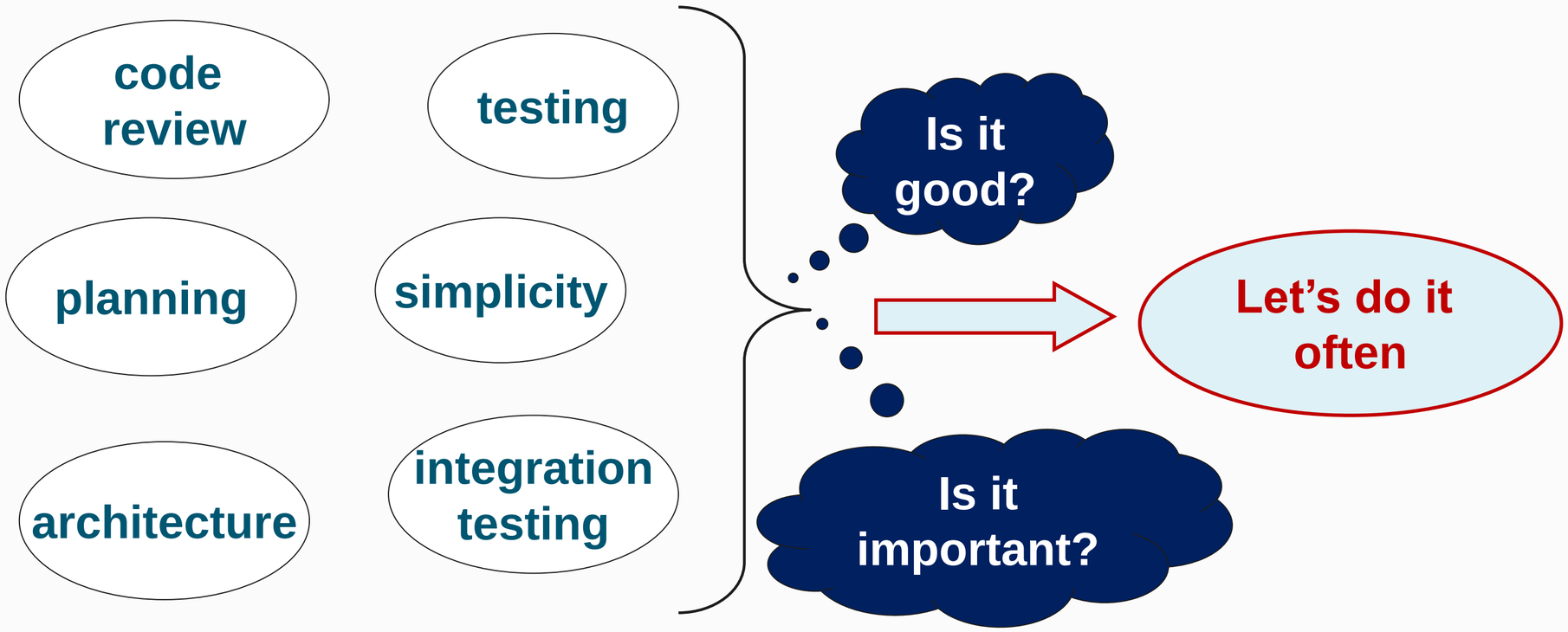
Model Driven Architecture

XP is…

…. a lightweight software development methodology



Extreme?


code review
testing
planning
simplicity
architecture
integration testing
Is it good?
Let's do it often
Is it important?

Mentality of sufficiency

How would you program if you had all the time in the word?

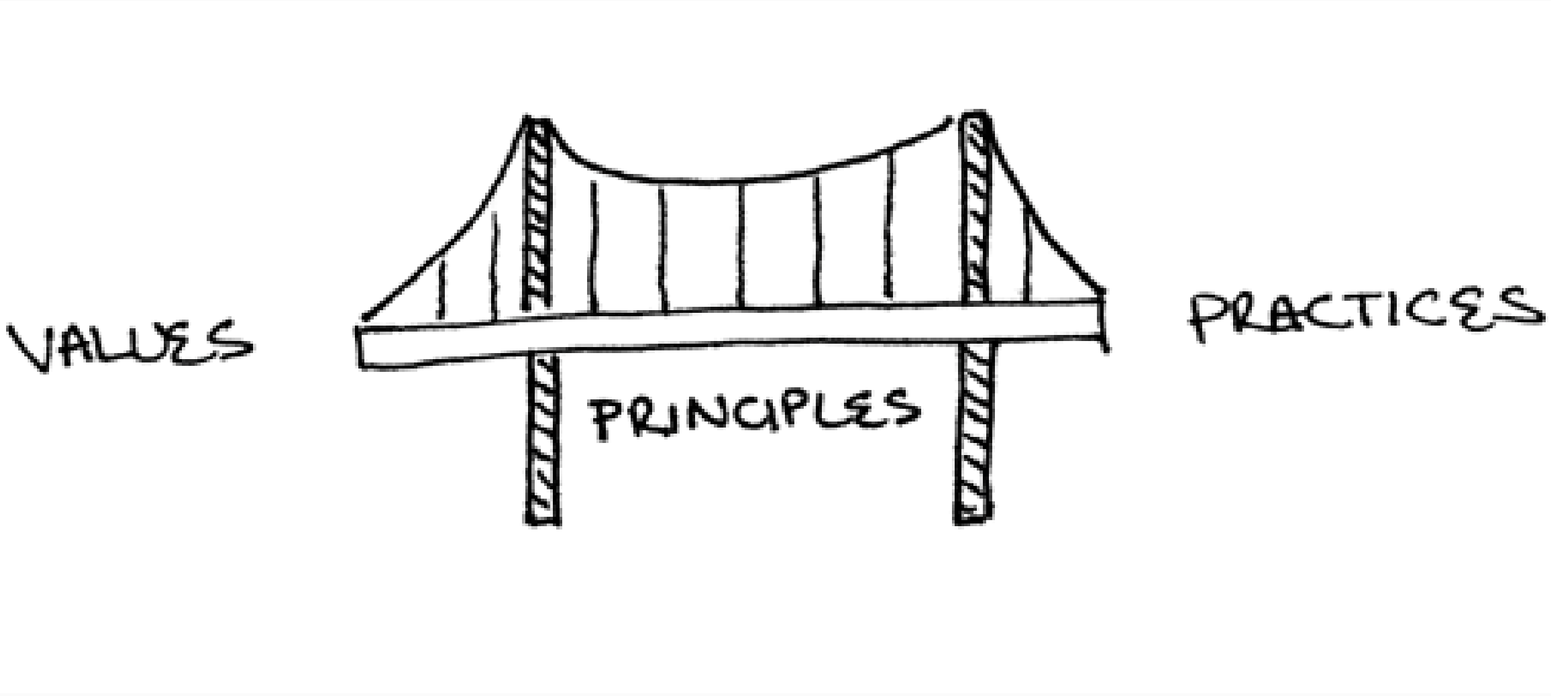
- Write tests
- Restructure often
- Talk with teammates and customer often

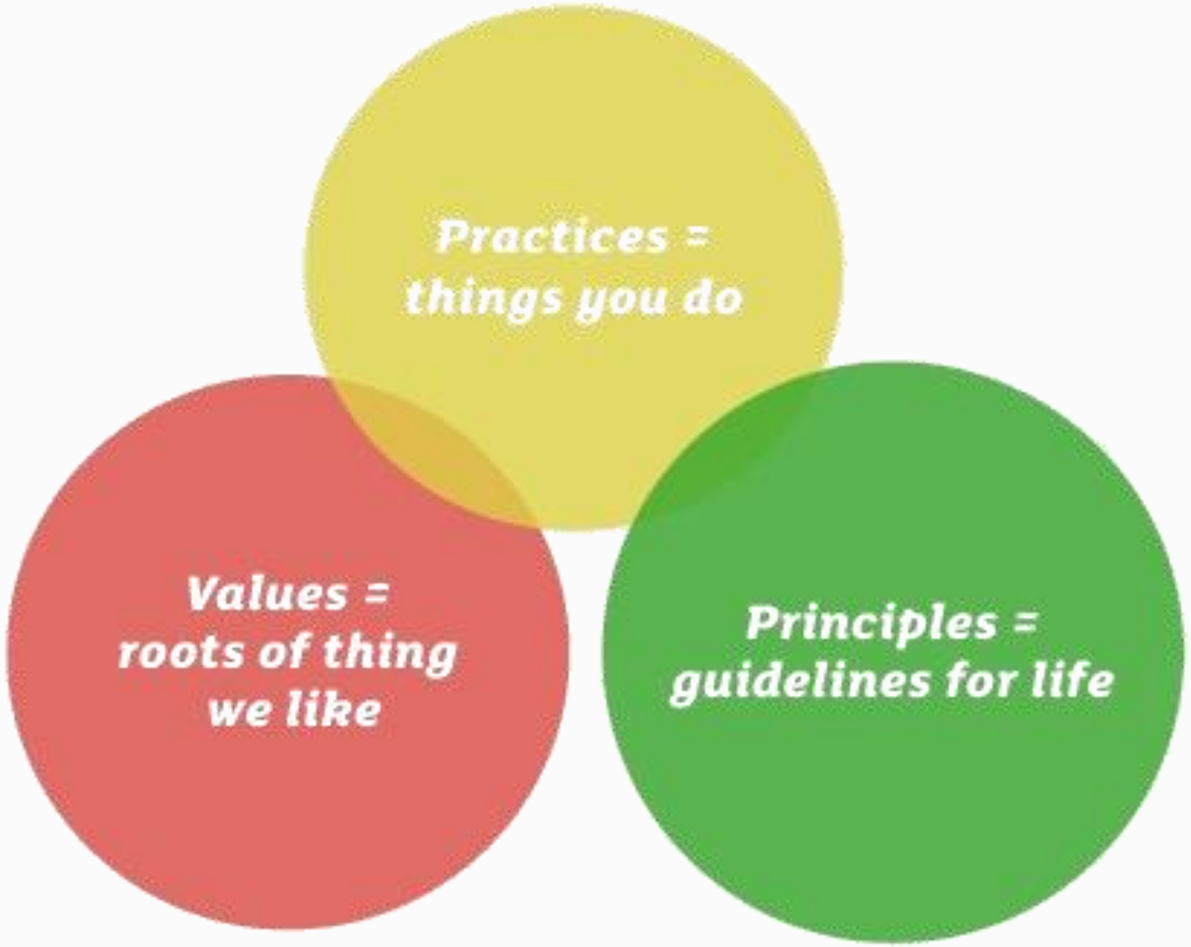
XP Paradigm

Stay aware.

Adapt.

Change.


VALUES
PRINCIPLES
PRACTICES



XP = Outstanding software

XP - Values

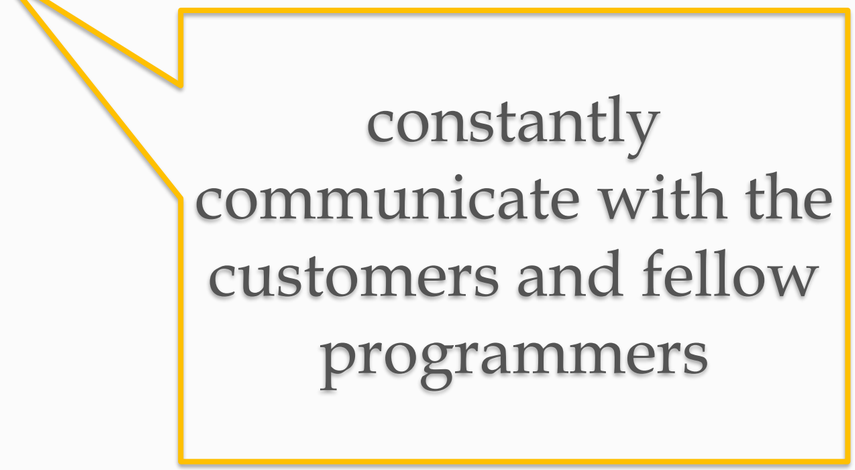
Communication

Simplicity

Feedback

Courage

Respect



XP - Values

Communication

Simplicity

Feedback

Courage

Respect

keep design simple and clean

XP - Values

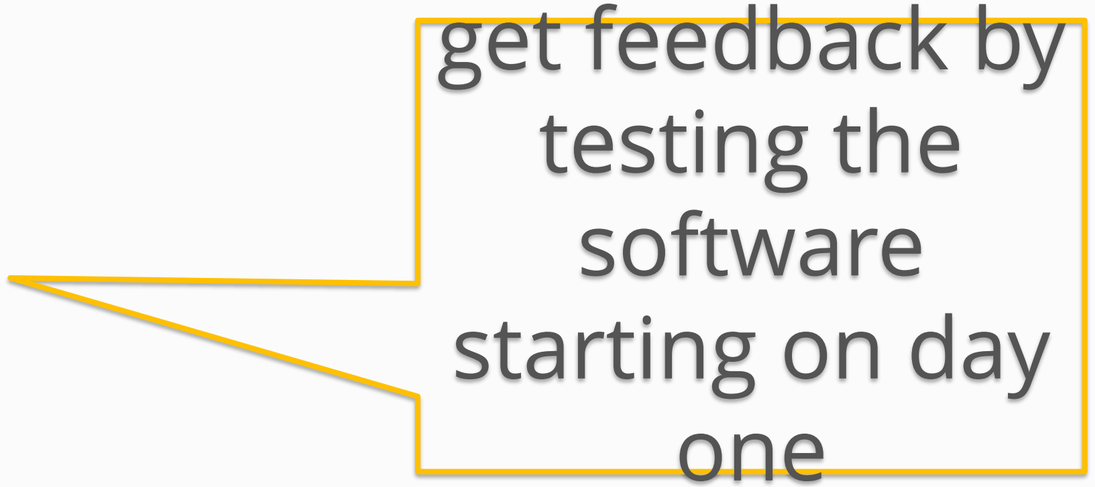
Communication

Simplicity

Feedback

Courage

Respect



XP - Values

Communication

Simplicity

Feedback

Courage

Respect

XP - Values

Communication

Simplicity

Feedback

Courage

Respect

show respect for the unique contributions of each and every team member

XP - Principles

Rapid feedback

Assume simplicity

Incremental change

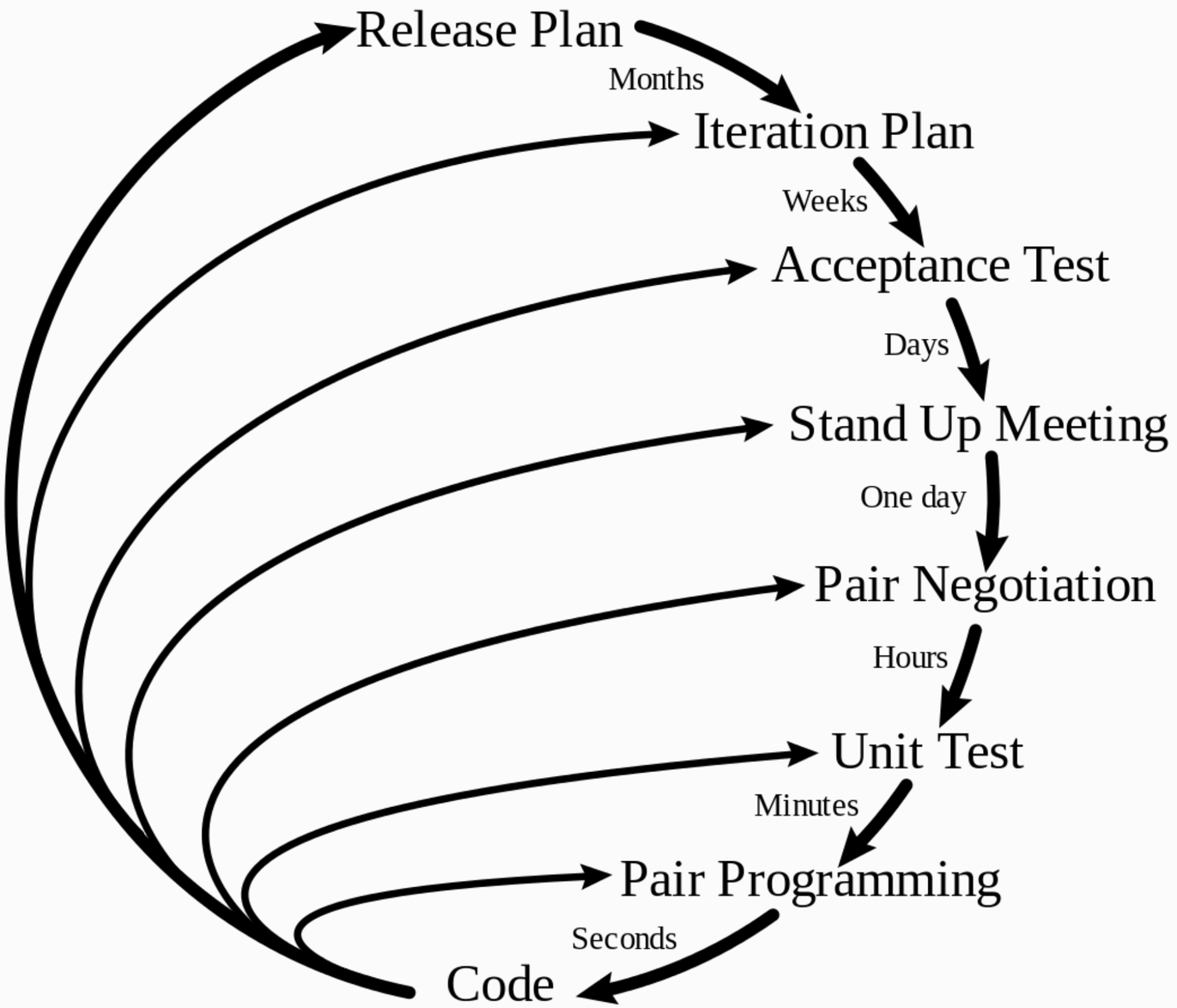
Embracing change

Quality work

get the feedback, understand it, and put the learning back into the system as quickly as possible

- Works as a catalyst for change
- Indicates progress
- Gives confidence to the developers that they are on the right track

Planning/Feedback Loops



XP - Principles

Rapid feedback

Assume simplicity

Incremental change

Embracing change

Quality work

to treat every problem as if it can be solved with simplicity

''Do the simplest thing that could possibly work''
KISS (“Keep It Simple, Stupid”
YAGNI (“You Aren’t Going to Need It”)

XP - Principles

Rapid feedback

Assume simplicity

Incremental change

Embracing change

Quality work

any problem is solved with a series of the smallest change that makes a difference

- The design changes a little at a time.
- The plan changes a little at a time.
- The team changes a little at a time.

XP - Principles

Rapid feedback

Assume simplicity

Incremental change

Embracing change

Quality work

the best strategy is the one that preserves the most options while actually solving your most pressing problem

XP - Principles

Rapid feedback

Assume simplicity

Incremental change

Embracing change

Quality work

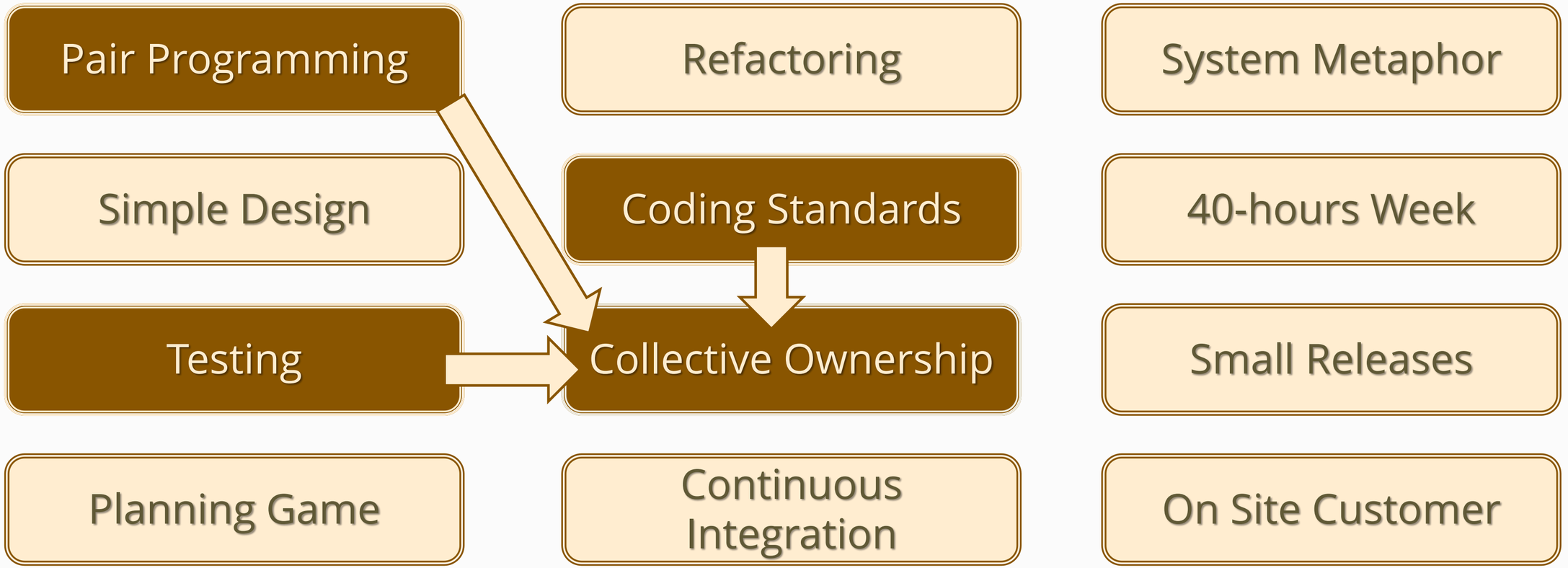
The team members try to produce the quality that they are proud of

The team:
Works well
Enjoys the work
Feels good in producing a product of value

12 Original **XP** Practices

Pair Programming	Refactoring	System Metaphor
Simple Design	Coding Standards	40-hours Week
Testing	Collective Ownership	Small Releases
Planning Game	Continuous Integration	On Site Customer

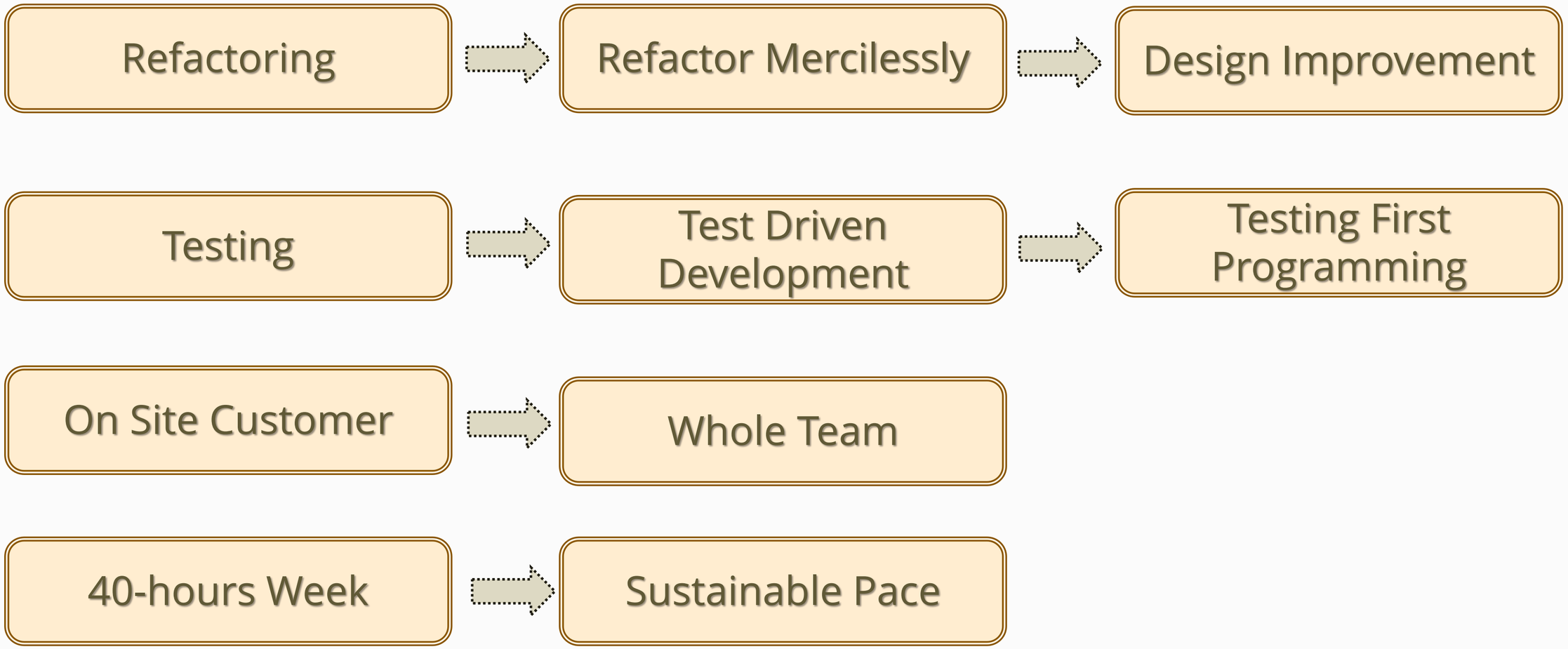
12 Original **XP** Practices

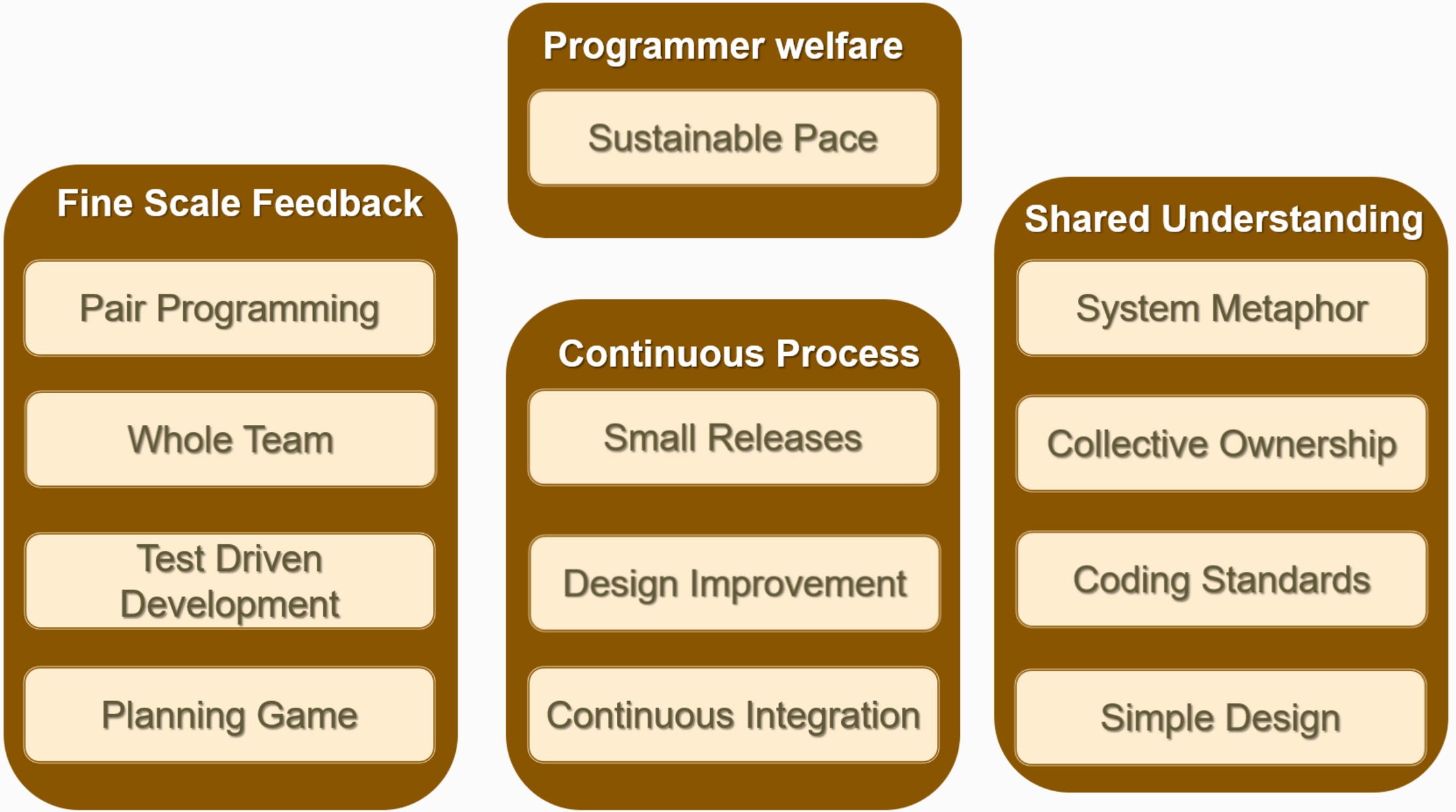


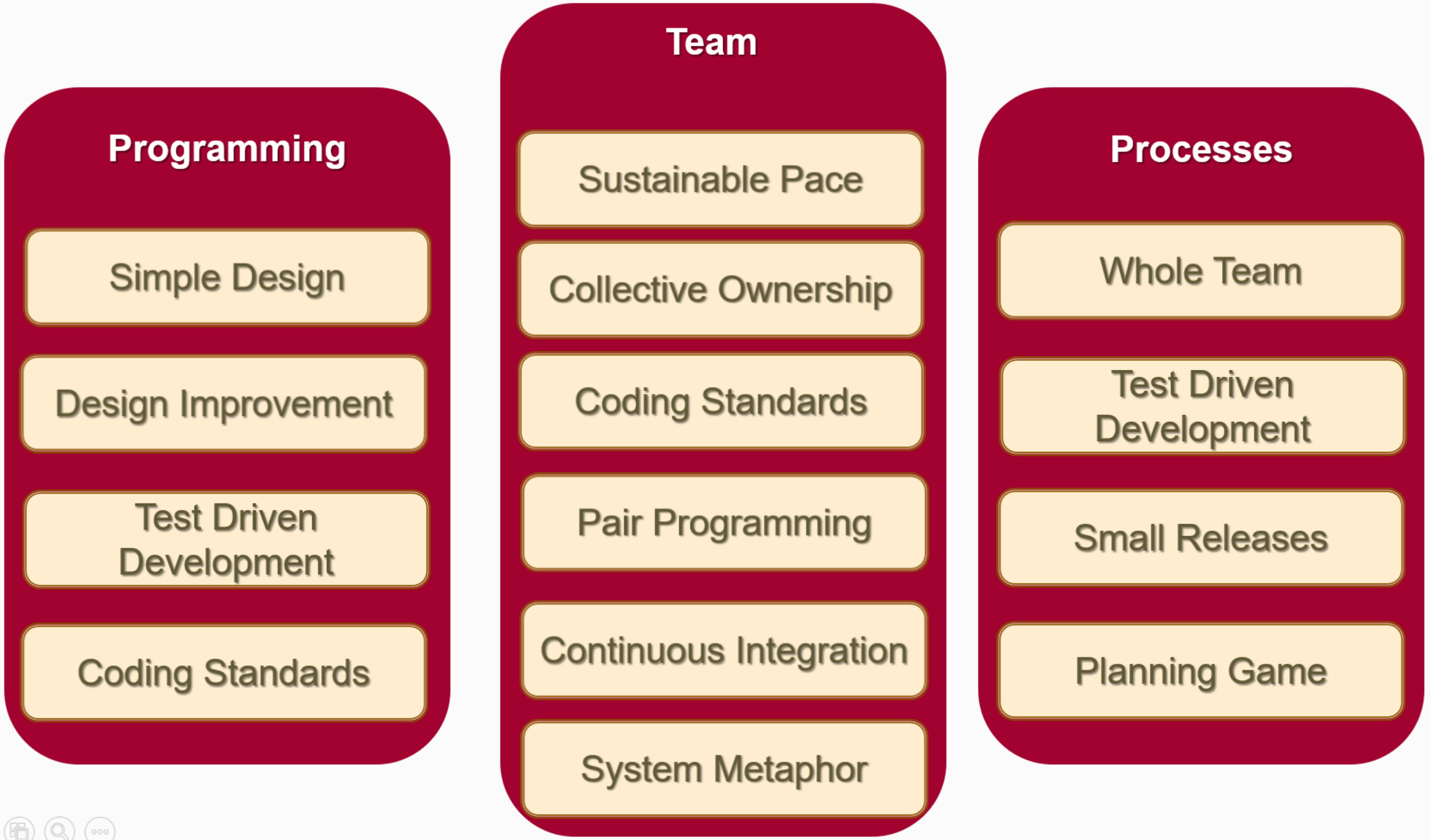
Some practices, if applied in isolation, could bring chaos

12 Original **XP** Practices

Evolution of some practices in time




Programmer welfare
Sustainable Pace
Fine Scale Feedback
Pair Programming
Whole Team
Test Driven Development
Planning Game
Continuous Process
Small Releases
Design Improvement
Continuous Integration
Shared Understanding
System Metaphor
Collective Ownership
Coding Standards
Simple Design


Programming
Simple Design
Design Improvement
Test Driven Development
Coding Standards
Team
Sustainable Pace
Collective Ownership
Coding Standards
Pair Programming
Continuous Integration
System Metaphor
Processes
Whole Team
Test Driven Development
Small Releases
Planning Game

Pair Programming

- Technique that requires 2 people, 1 computer

DRIVER:

- Controls the keyboard
- Writes the code and tests
- Tactics (how?)

NAVIGATOR:

- Has the role of reviewer
- Guides the driver
- Strategy (what?)

Important: they switch roles!!!

When to use it?

- When mentoring new hires
- For extremely high-risk tasks
- At the start of a new project
- When adopting a new technology

Benefits

- Better code, design
- Fewer bugs
- Higher morale
- Shared perspectives and knowledge
- Better time management
- Higher productivity

!!! All without sacrificing productivity

How to do it better?

- Have a well-defined task
- Define a goal at a time
- Rely, support and synchronize with your partner
- Pair with everyone in the team
- Be physically comfortable
- Give everyone a chance to be an expert

Important: communicate!!!

Proxemix

= the study of human use of space

Intimate distance (*for embracing, touching or whispering*)

15 cm - 46 cm

Personal distance (*for interactions among good friends or family*)

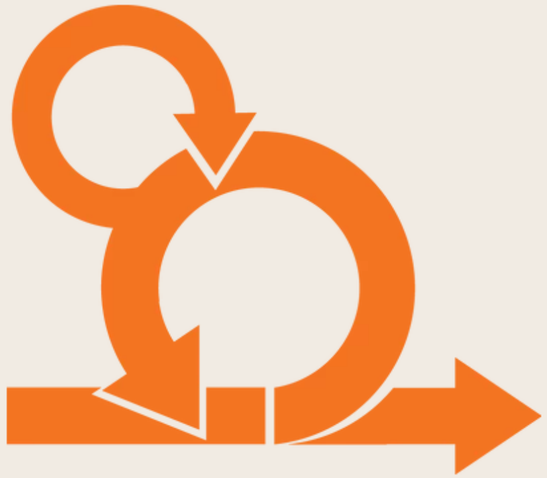
46 to 122 cm

Social distance (*for interactions among acquaintances*)

1.2 - 3.7 m

Public distance (*used for public speaking*)

3.7 to 7.6 m

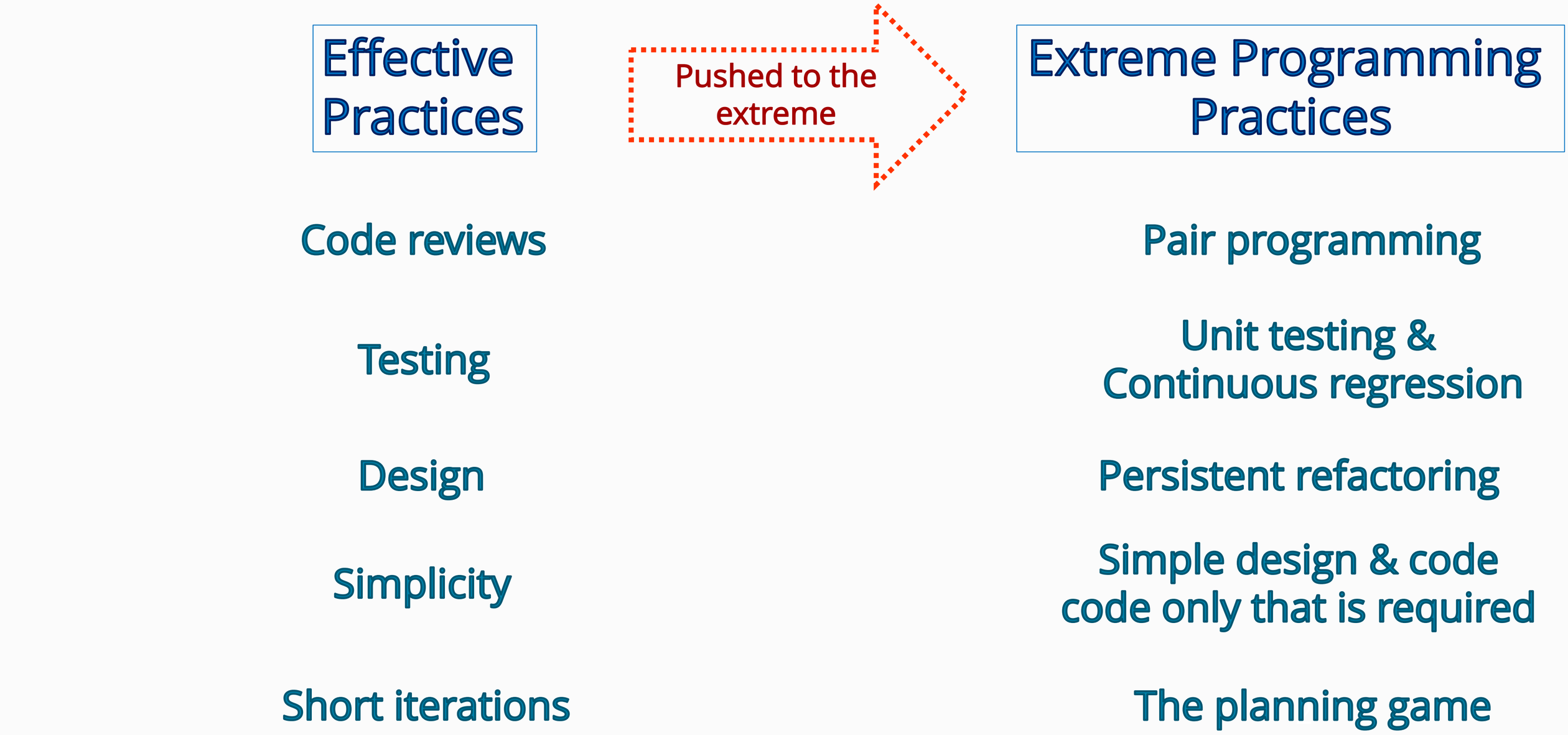


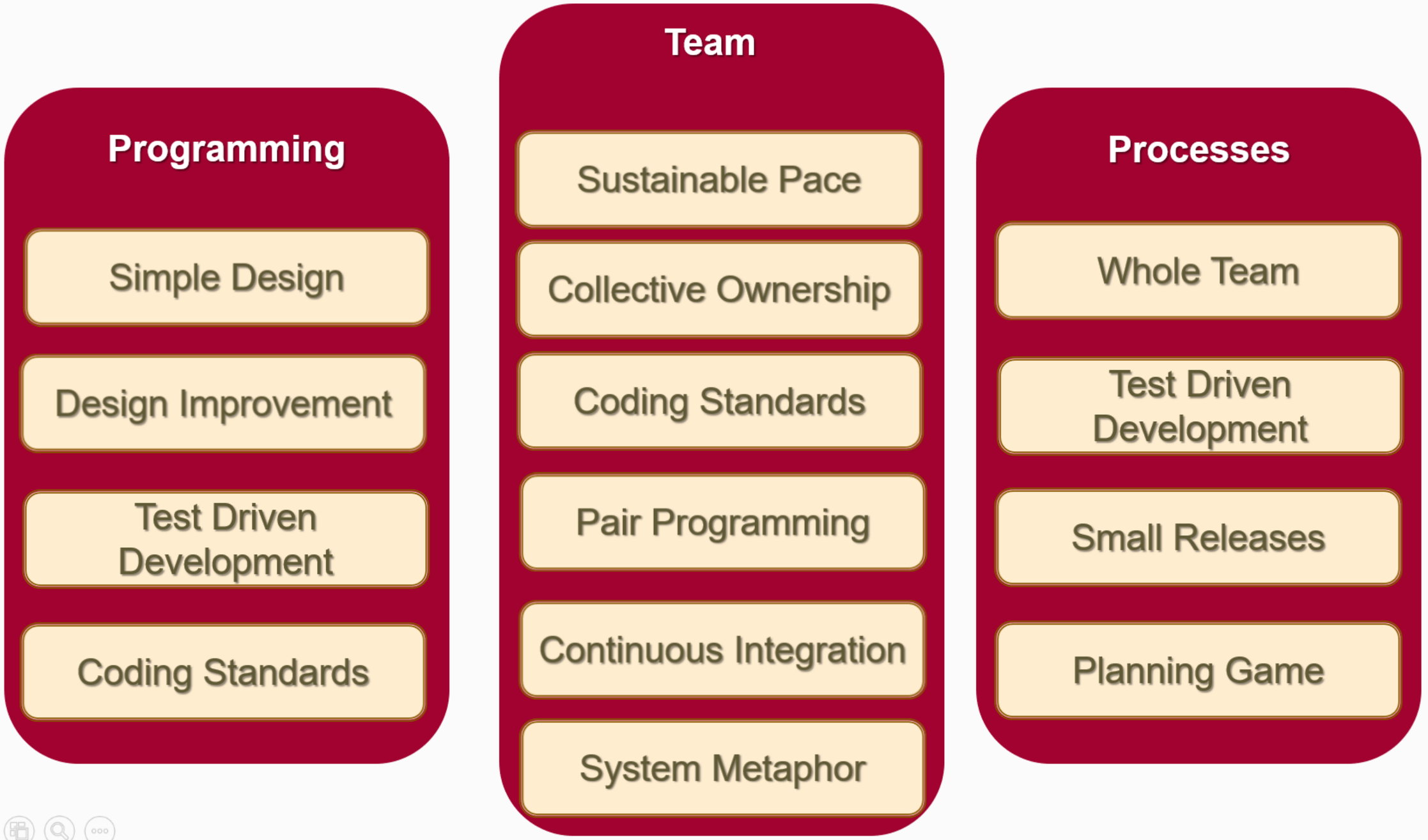
AGILE SOFTWARE DEVELOPMENT

eXtreme Programming

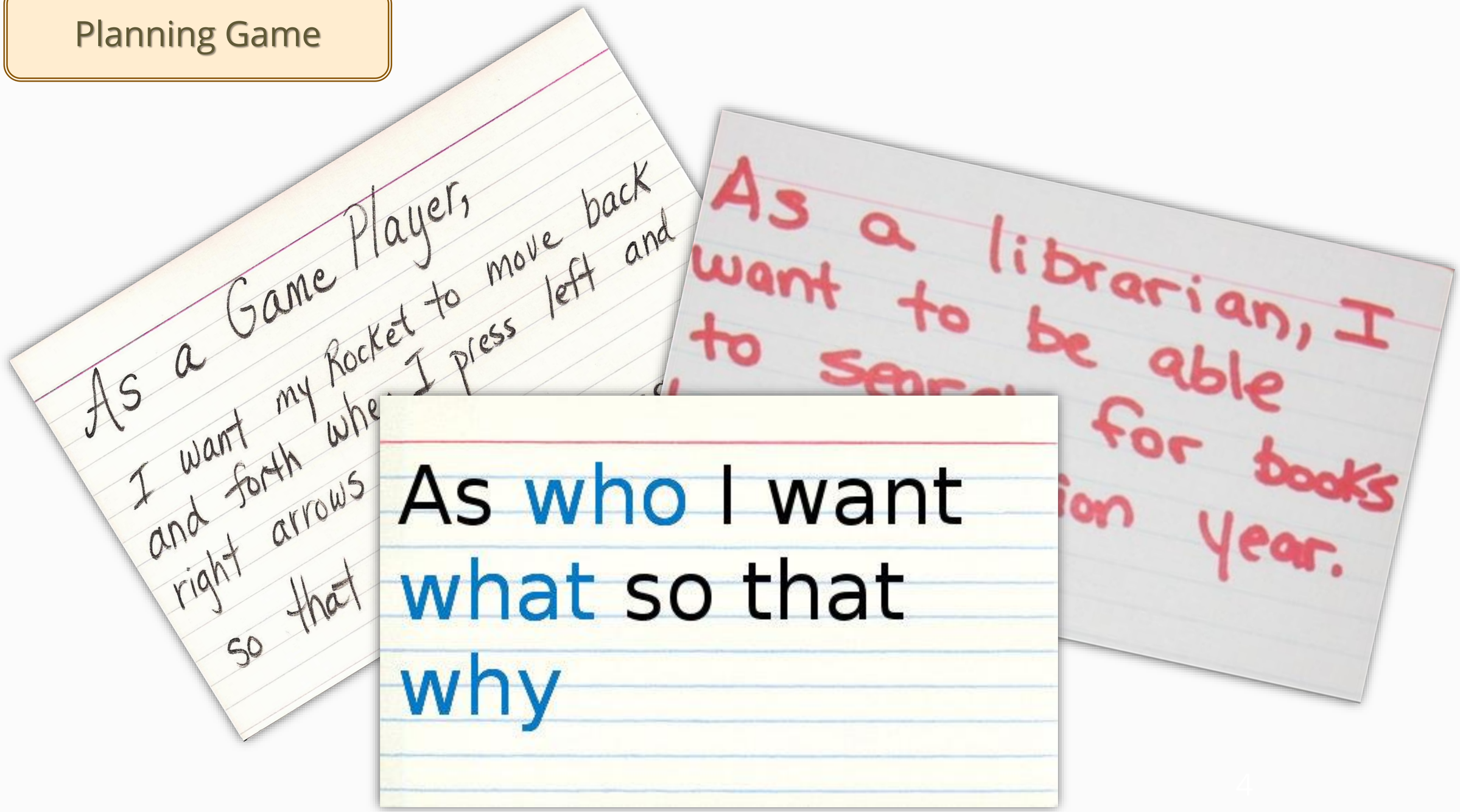
2

Extreme Programming




Programming
Simple Design
Design Improvement
Test Driven Development
Coding Standards
Team
Sustainable Pace
Collective Ownership
Coding Standards
Pair Programming
Continuous Integration
System Metaphor
Processes
Whole Team
Test Driven Development
Small Releases
Planning Game

Planning Game



Planning Game

Business and development need to make the decisions in tandem

Business people need to decide about

- **Scope**: How much of a problem must be solved for the system to be valuable in production?
- **Priority**: If you are given an option, which one do you want?
- **Composition of releases**: How much or how little needs to be done before the business is better off with the software than without it?
- **Dates of releases**: What are important dates at which the presence of the software (or some of the software) would make a big difference?

Planning Game

Technical people need to decide about

- **Estimates**: How long will a feature take to implement?
- **Consequences**: There are strategic business decisions that should be made only when informed about the technical consequences. Development needs to explain the consequences.
- **Process**: How will the work and the team be organized? The team needs to fit the culture in which it will operate. The software must be written well rather than preserve the irrationality of an enclosing culture.
- **Detailed scheduling/Risk priority**: Within a release, which stories should be done first?

Small Releases

The only way to ensure that you are developing the software the customer expects!

Every release could be used as a checkpoint to measure the estimation accuracy.

Small Releases

Fail Fast

Fail Often

Small Releases

- Product Roadmap =

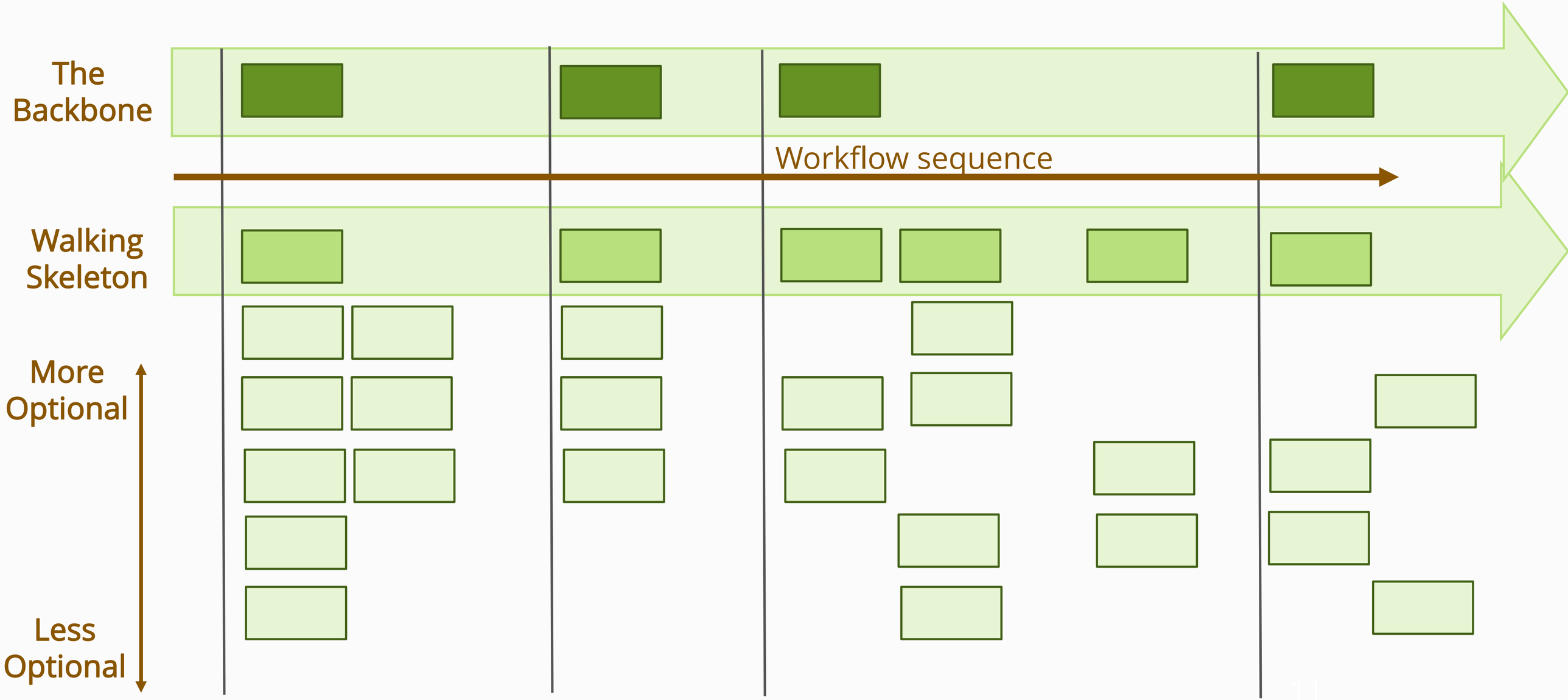
  visual overview of a product’s releases

  - Product roadmap = sequence of releases
  - Release = sequence of iterations
  - Iteration = set of user stories / features

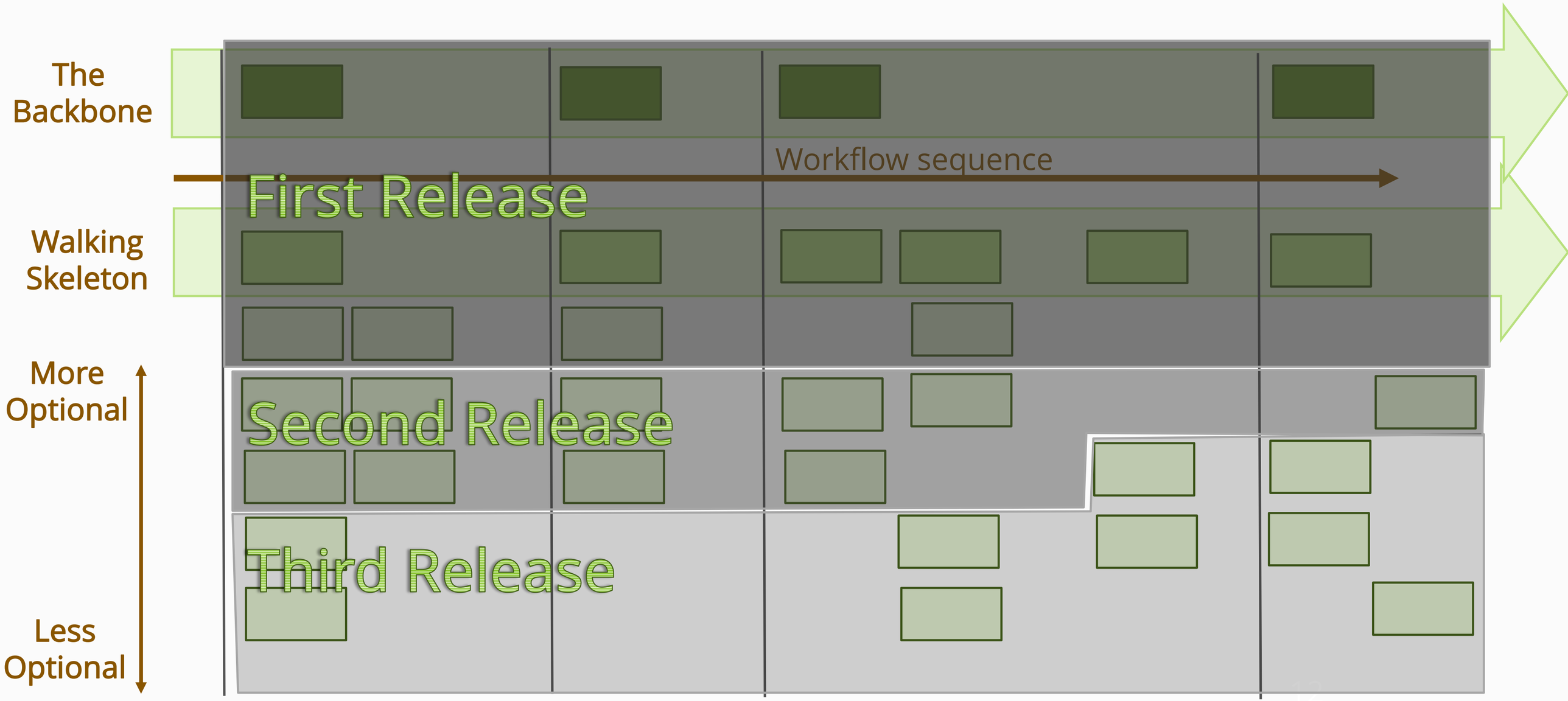
Small Releases

- Story Map (developed by Jeff Patton)
- Helps select and group features for a release
  - Backbone – essential functionality
  - Walking skeleton – smallest system that could possible work
  - Optional features

Small Releases



Small Releases



Metaphor

A good metaphor is a powerful aid in unifying the technical and business teams

The team evolves its own form of "*tribal language*", used to describe user stories and development

Metaphor

“. . . I still haven't got the hang of this metaphor thing. I saw it work, and work well, on the C3 project, but it doesn't mean I have any idea how to do it, let alone how to explain how to do it”

Martin Fowler

Metaphor

'Metaphor' seems to be one of the least understood precepts of XP although its supposed to be (one of) the most important.

Metaphor

Metaphor is something you start using when your mother asks what you are working on and you try to explain her the details.

How you find it is very project-specific.

Use your common sense or find the guy on your team who is good at explaining technical things to customers in a way that is easy to understand.

Simple Design

XP definition for *simplicity*:

- Runs all the tests
- Contains no duplicate code
- States the programmer's intent for all the code clearly
- Contains the fewest possible classes and methods

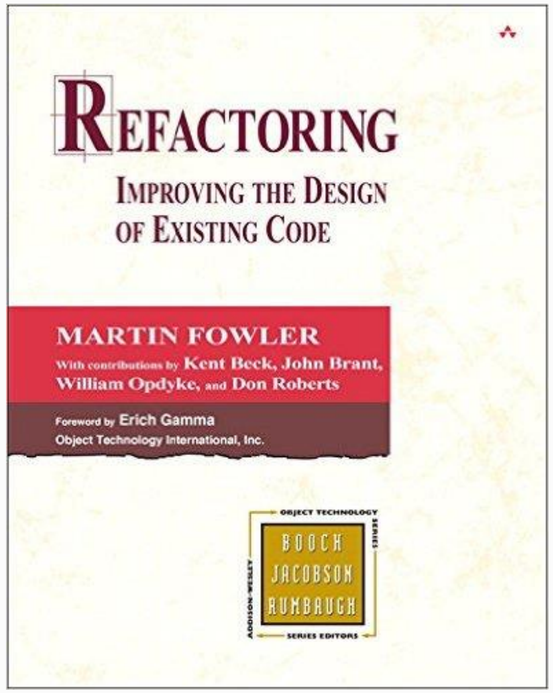
Simple Design

No big design upfront!

1. Only do what you need to do now!
2. Don't add anything because you think you *might* need it!

Refactoring

Refactoring is the technique of improving code without changing functionality



Why?

Because your code should be the simplest thing that could possibly work

Refactoring

1) Lack of tests
2) Name not from domain
3) Name not expressing intent
4) Unnecessary if
5) Unnecessary else
6) Duplication of constant
7) Method does more than one thing
8) Primitive obsession
9) Feature envy
10) Method too long (> 6 lines)
11) Too many parameters (> 3)
12) Test – not unitary
13) Test – setup too complex
14) Test – unclear Act
15) Test - more than one assert
16) Test – no assert
17) Test – too many paths

Source: Brutal Refactoring Game, Adi Bolboaca

Testing

Testing First Programming

Test Driven Development

Unit testing

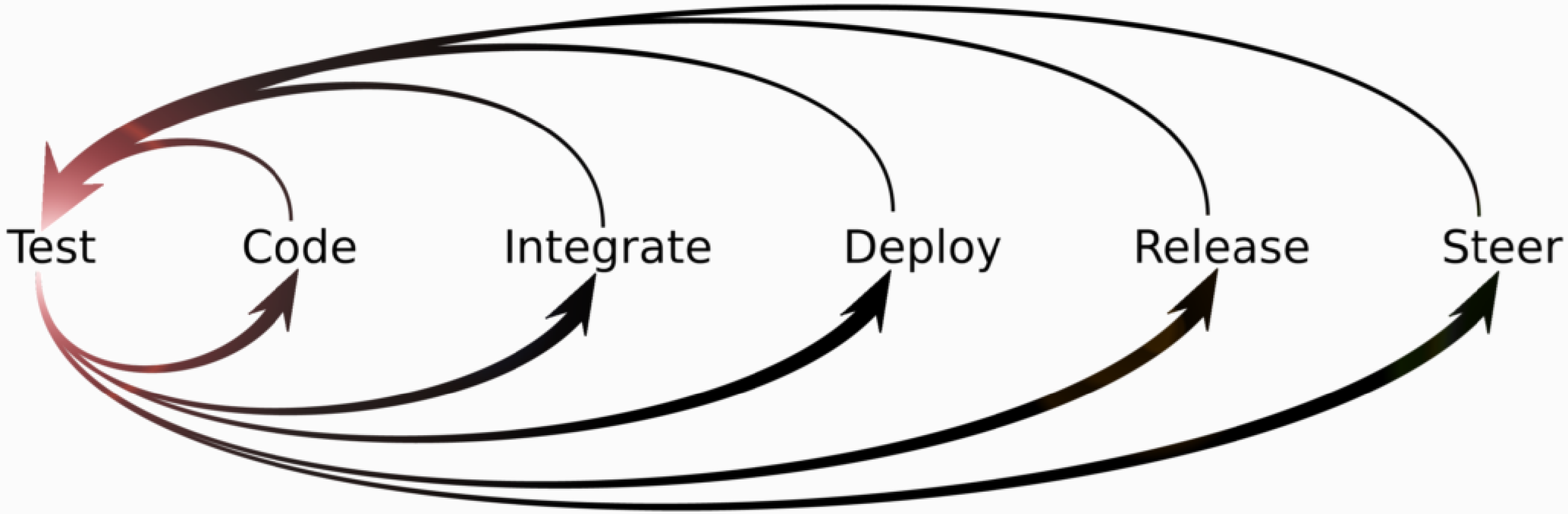
Acceptance testing

Test Driven Development

Testing

Testing First Programming

Test Driven Development



- Add a test, get it to fail, and write code to pass the test
- Remove duplication

Testing

Testing First Programming

Test Driven Development

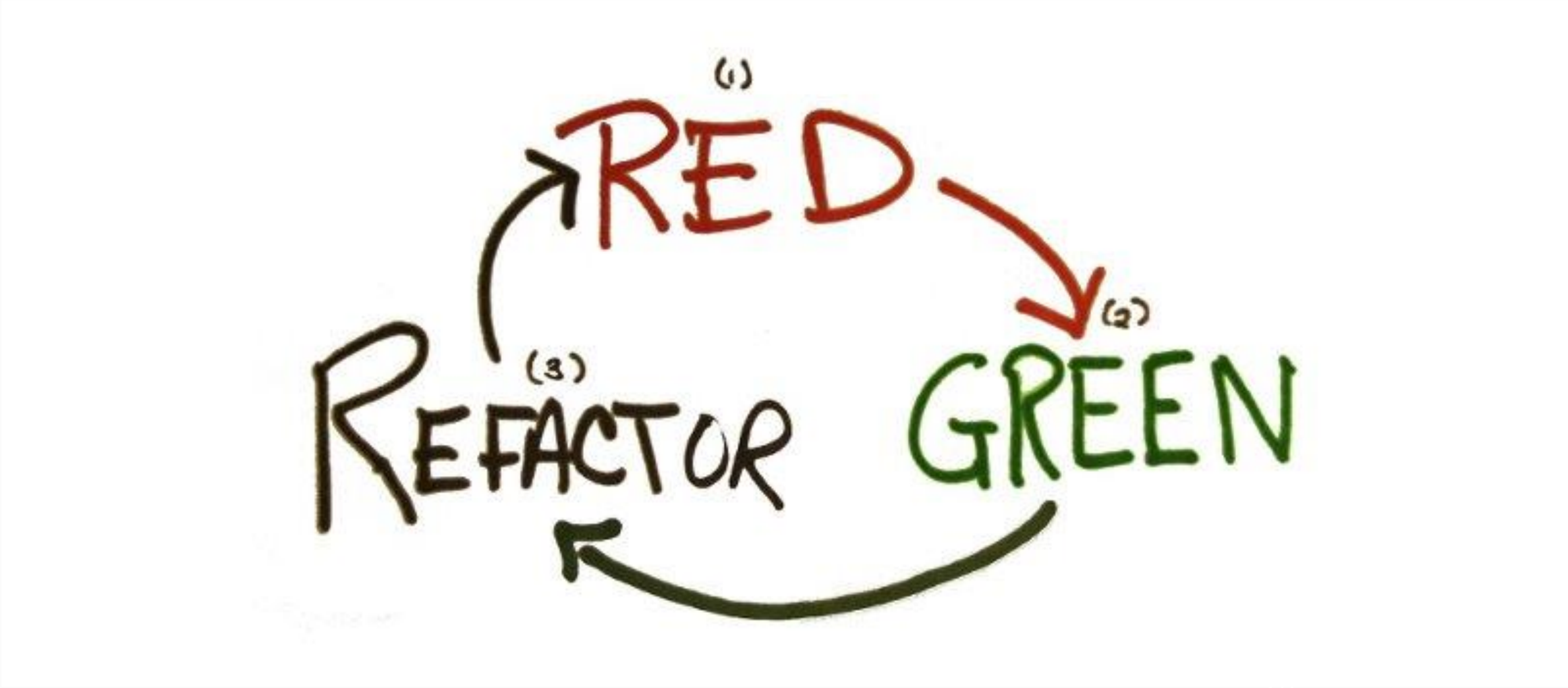
- Think about what you want to do.
- Think about how to test it.
- Write a small test. Think about the desired API.
- Write just enough code to fail the test.
- Run and watch the test fail. Now you know that your test is going to be executed.
- Write just enough code to pass the test (and pass all your previous tests).
- Run and watch all of the tests pass. If it doesn't pass, you did something wrong, fix it now since it's got to be something you just wrote.
- Refactor the code
- Run the tests again
- Repeat the steps above until you can't find any more tests that drive writing new code.

if you can't do these, you probably shouldn't start writing any code

Testing

Testing First Programming

Test Driven Development



Collective Ownership

Any person can change
the application code
at anytime

The catch: If you own all the code, you are responsible for all the code as well

Continuous Integration

10-minute Build

The longer the time period between integrations, the more conflicts you'll see, and the effort to integrate will increase.

The process: developers work on tasks until complete, integrate them, run tests, fix problems

40-Hour Work Week

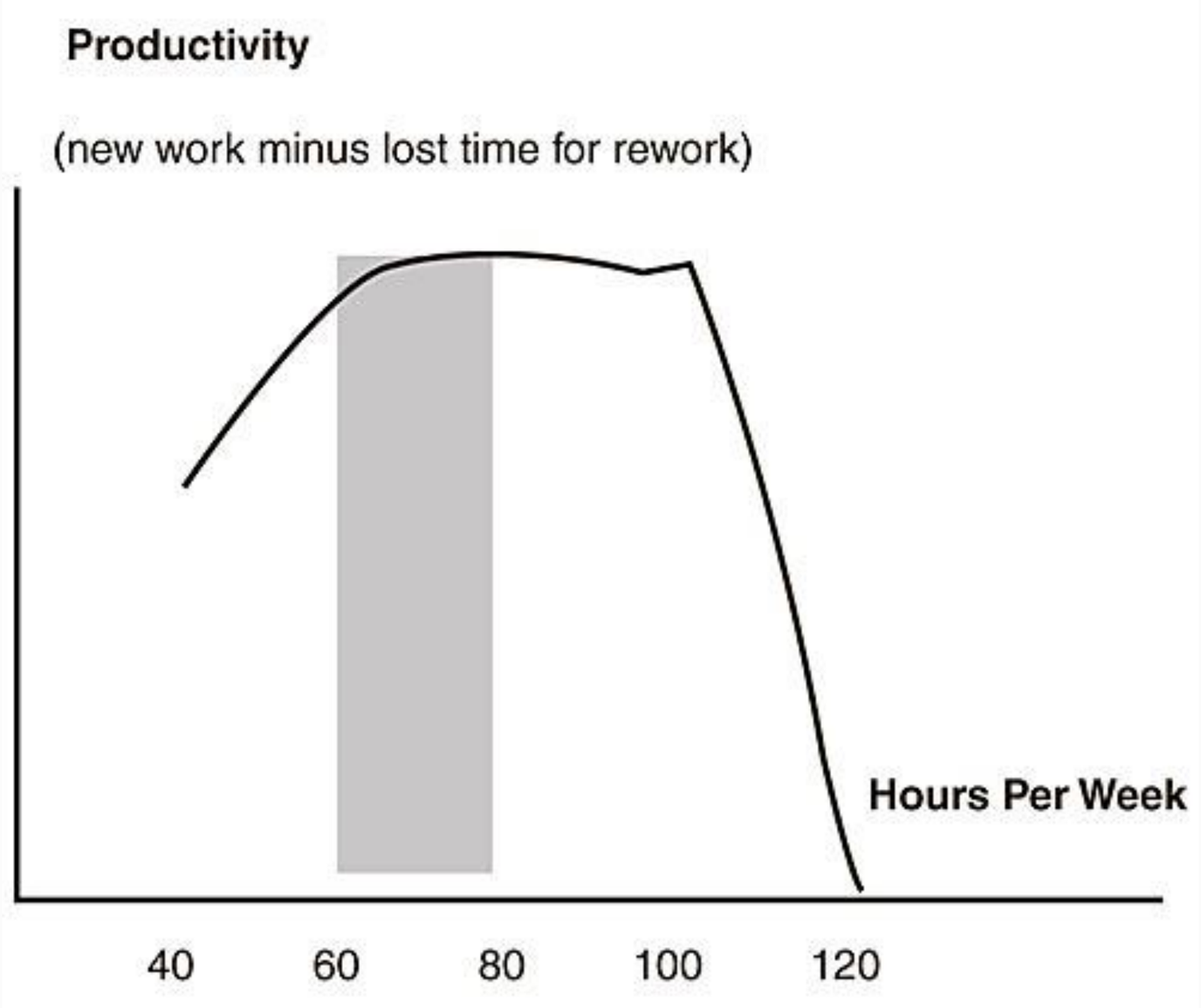
Sustainable Pace

You cannot maintain quality with overtime-heavy teams!

Each country or culture has differing acceptance of reasonable working hours

40-Hour Work Week

Sustainable Pace



On-Site Customer

Whole team

Sit together

Benefits

Simple Problem Solving

Avoid Misunderstanding

Immediate answers

Team Spirit

On-Site Customer

Whole team

Sit together

The customer must be on the project full-time for the duration and be located on-site with the team

The *customer* could include users, business experts, and any other customer-side resource

Coding Standards

Mandatory — Those standards to be adhered to by all team members.

Guidelines — Those considered best or good practice and often describe the general approach toward development.

Recommendations — These rules are considered good practice and should be used at all times unless there are exceptional circumstances where valid justification can be given.

Coding Standards

There are two types of people:

```
If (Condition)
{
        Statements
         /*
         …
         */
}
```

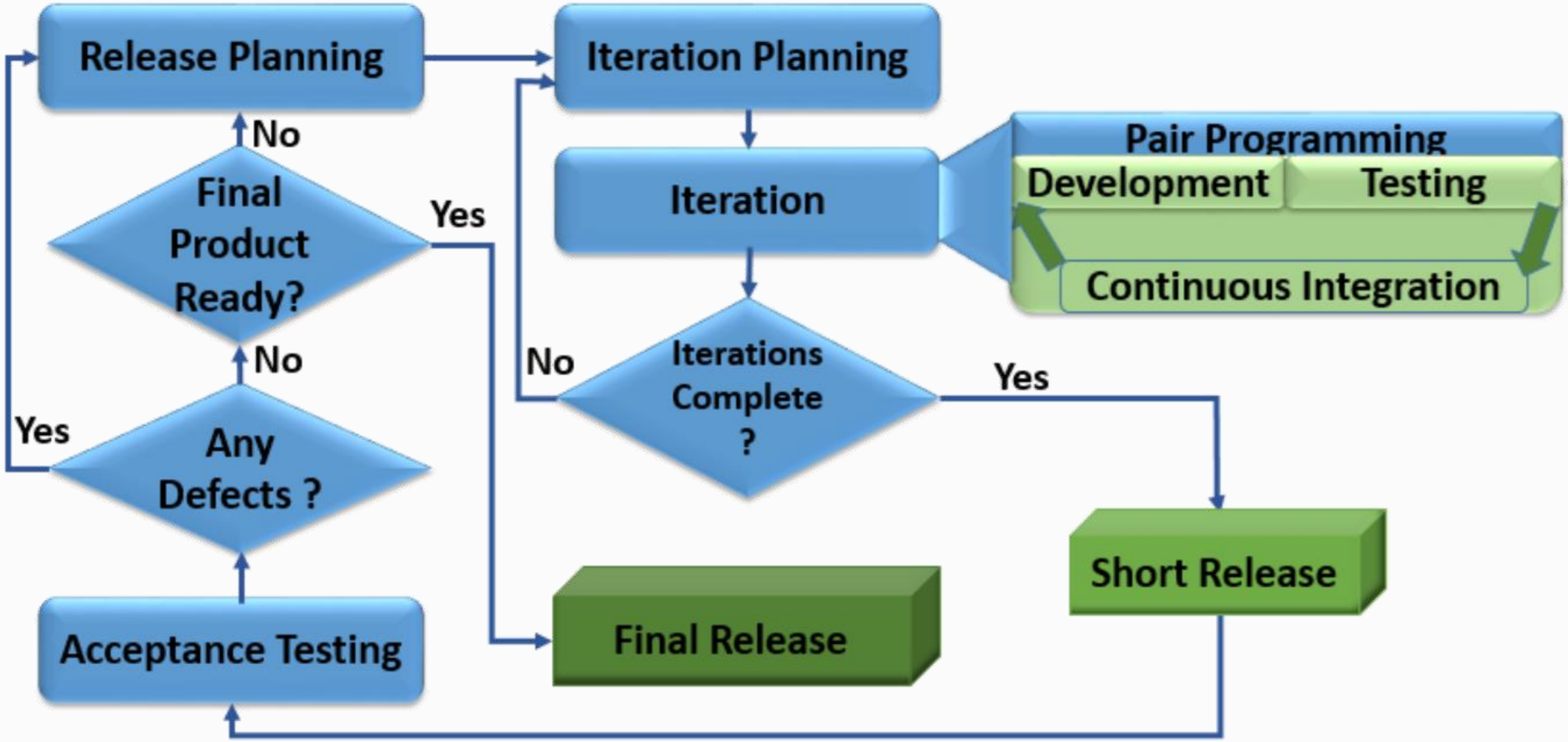
```
If (Condition){

     Statements
      /*
      …
      */
}
```

Coding Standards

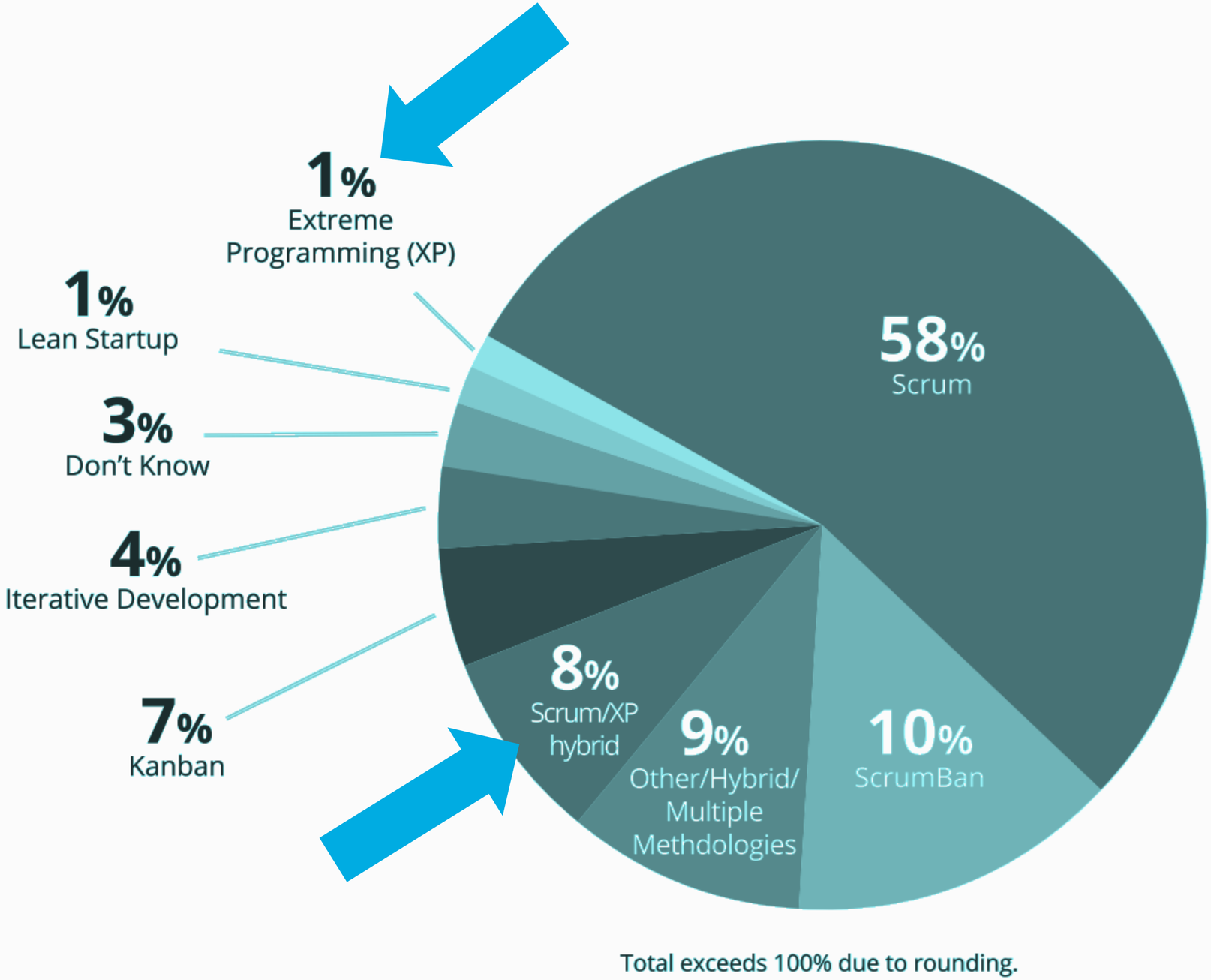
Types of coding standards:

- *Formatting*
- *Code structure*
- *Naming conventions*
- *Error handling*
- *Comments*

Extreme Programming Process Cycle



Source: Extreme Programming, tutorialspoint

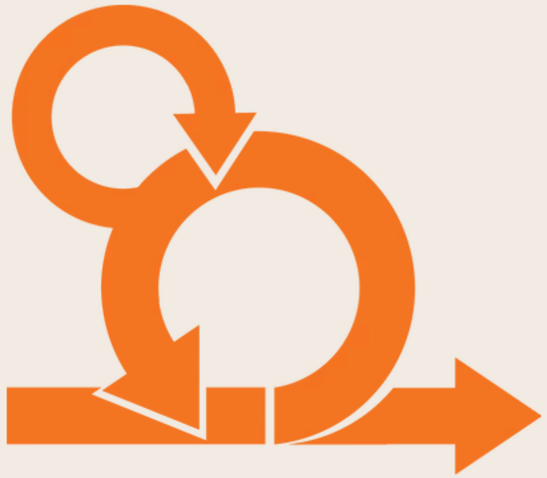


Total exceeds 100% due to rounding.

14th Annual State of Agile Report, May 26th 2020
https://stateofagile.com

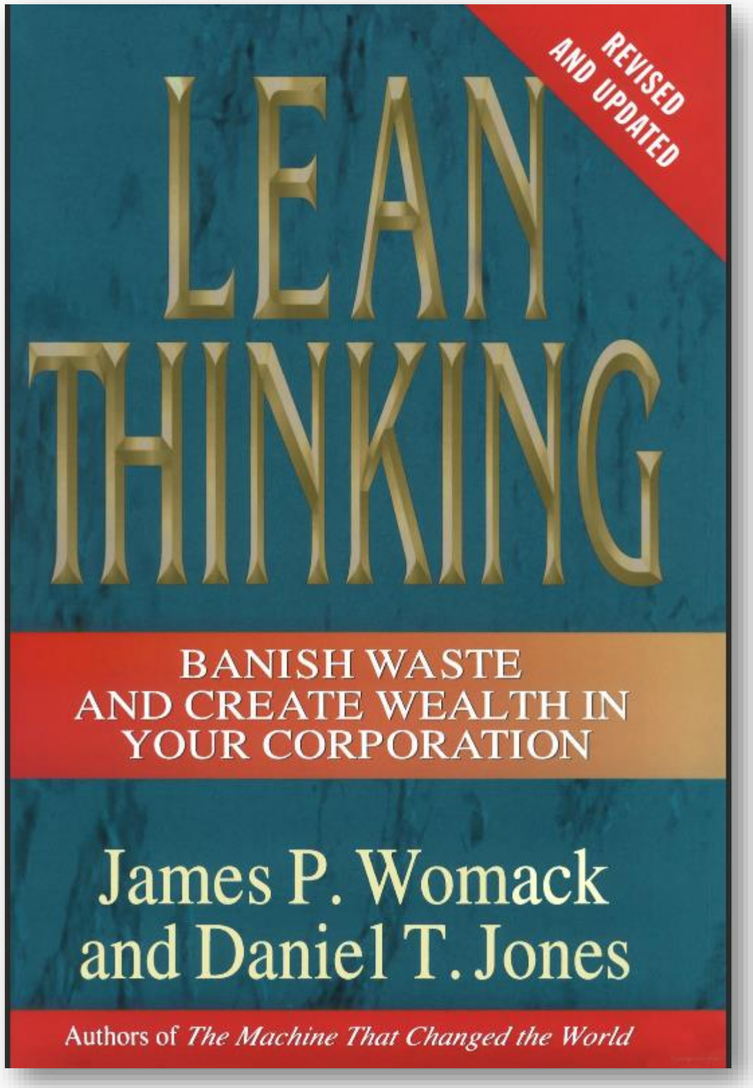
Why XP is not popular?

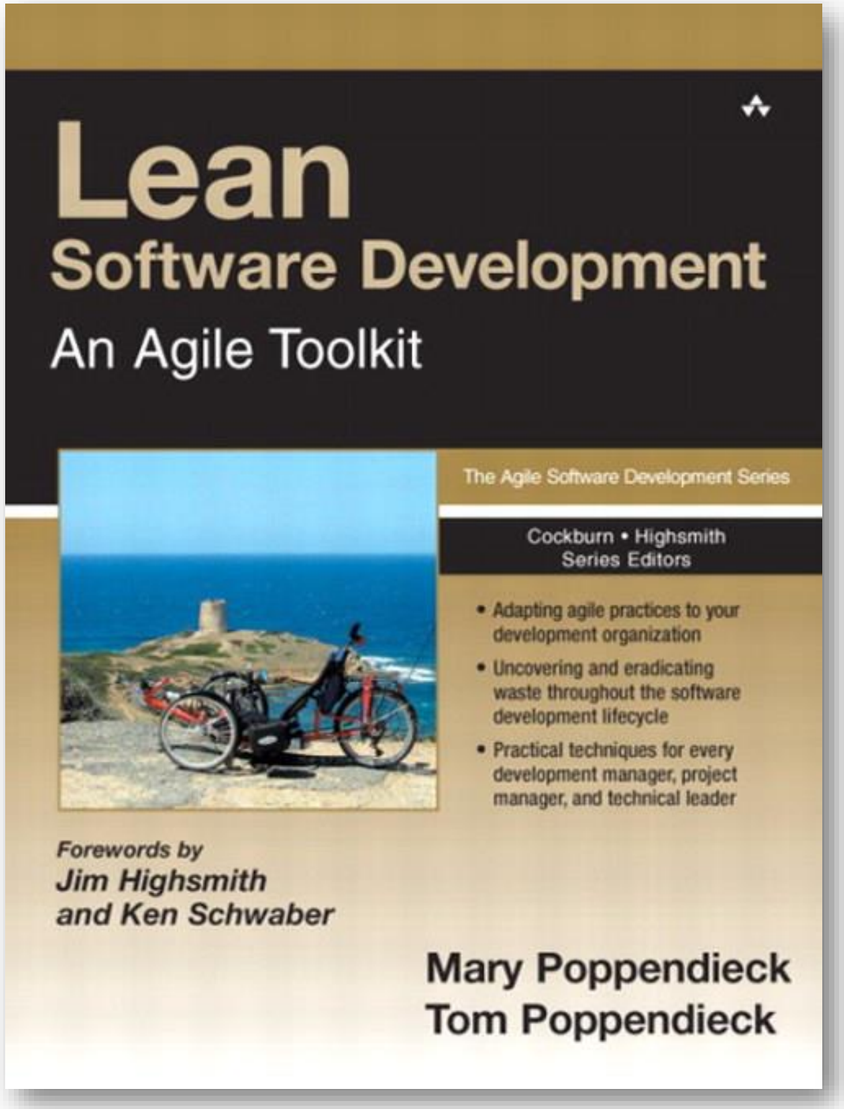
- *It is software engineering centric*
- *It requires high investment*
  - *Rockstar developers*
  - *Trainings*
  - *Infrastructure (automation solutions)*
  - *Culture*
- *It is irrational (to business people)*
  - *Unit tests, Test First Development, Story Points, Pair Programming*
- *It is too difficult*
  - *Test First Development, Refactoring, Simple & Emergent Design*

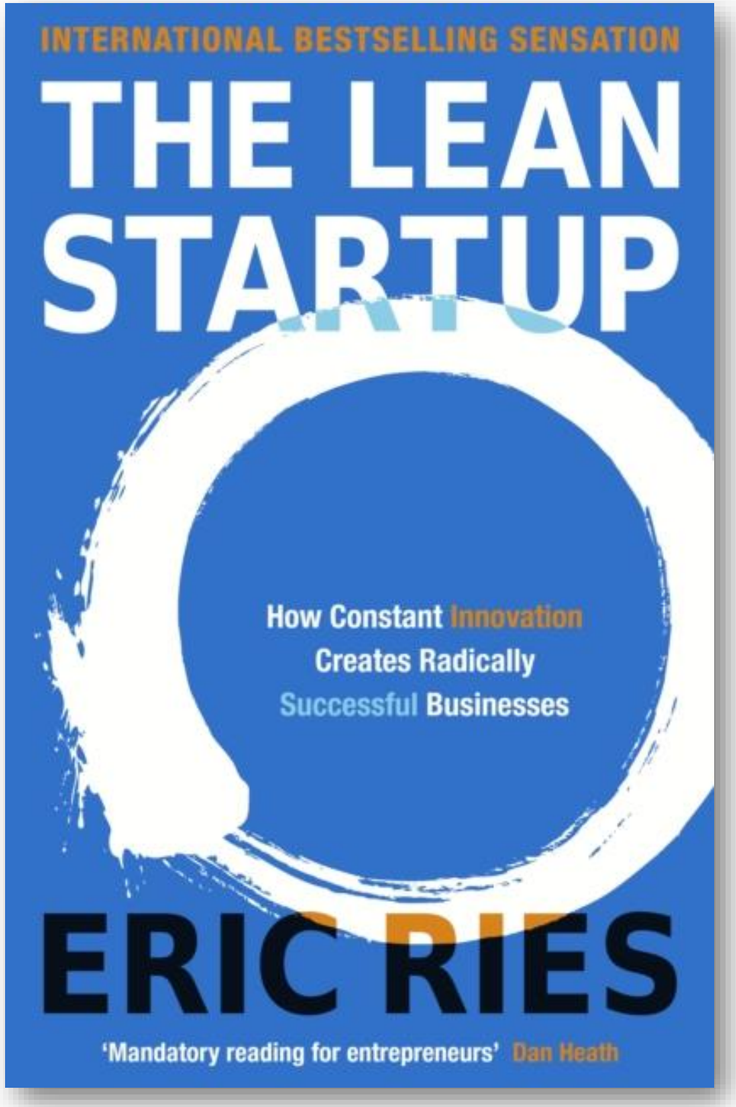


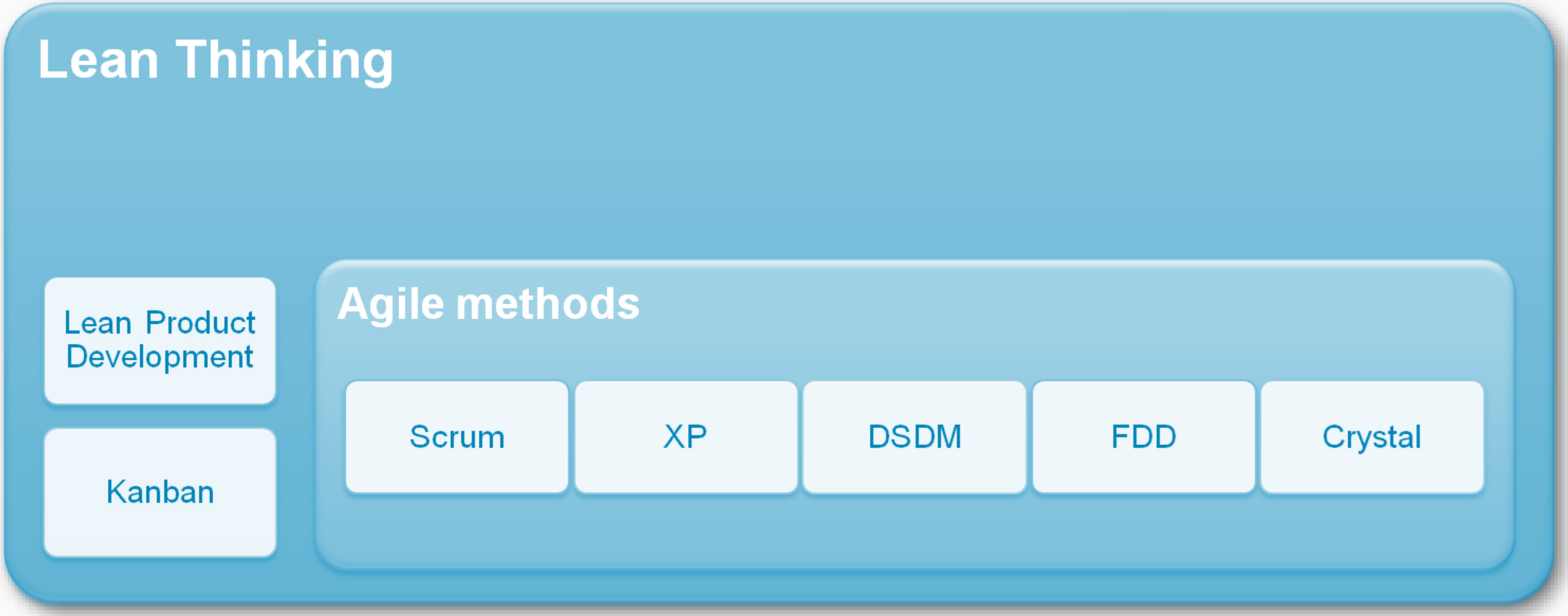
AGILE SOFTWARE DEVELOPMENT

Lean Software Development


REVISED AND UPDATED
LEAN THINKING
BANISH WASTE AND CREATE WEALTH IN YOUR CORPORATION
James P. Womack and Daniel T. Jones
Authors of The Machine That Changed the World


Lean
Software Development
An Agile Toolkit
The Agile Software Development Series
Cockburn • Highsmith
Series Editors
• Adapting agile practices to your development organization
• Uncovering and eradicating waste throughout the software development lifecycle
• Practical techniques for every development manager, project manager, and technical leader
Forewords by
Jim Highsmith
and Ken Schwaber
Mary Poppendieck
Tom Poppendieck


INTERNATIONAL BESTSELLING SENSATION
THE LEAN STARTUP
How Constant Innovation Creates Radically Successful Businesses
ERIC RIES
'Mandatory reading for entrepreneurs' Dan Heath


Lean Thinking
Lean Product Development
Kanban
Agile methods
Scrum
XP
DSDM
FDD
Crystal

Agile

- Adaptive to change
- Shorter planning and commitment cycles
- Focus on collaboration and interaction

Lean

- System view of value stream
- Identify ways to eliminate waste
- Limit work queues

Agile & Lean Commonalities

- Improve quality
- Amplify learning
- Continuously improve
- Empower people

"Think big, act small, fail fast; learn rapidly"

A Lean History

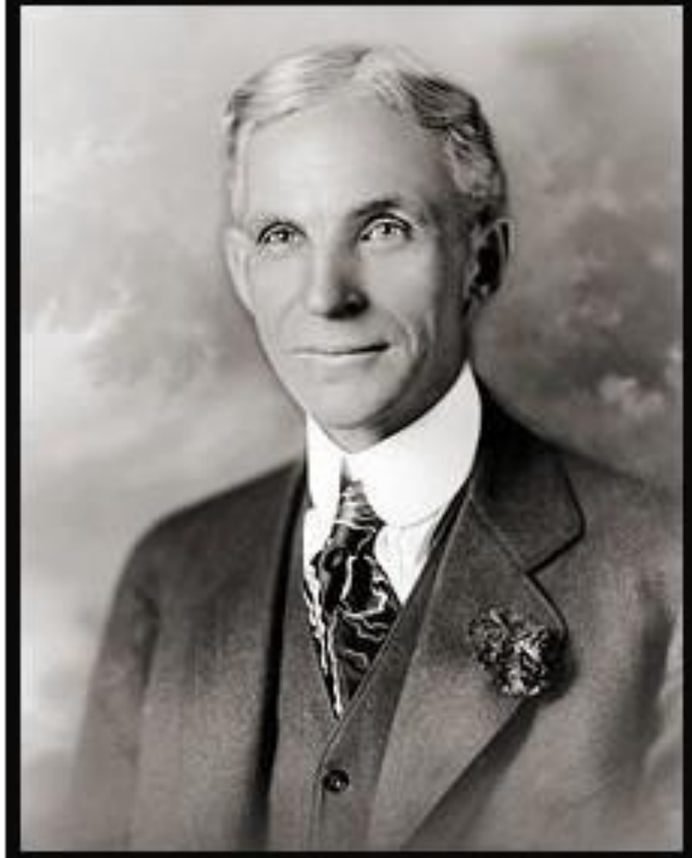
- Lean is a *manufacturing* & *production* practice that considers the expenditure of resources for any goal other than the creation of **value for the end customer** to be wasteful, and thus a target for elimination

- "**value**" is defined as any action or process that a customer would be willing to pay for

A Lean History

- Lean is centered around preserving value with less work
- Lean manufacturing is based on
  - optimizing flow,
  - increasing efficiency,
  - decreasing waste,
  - using empirical methods to decide what matters, rather than uncritically accepting pre-existing ideas
- Toyota was a leader in implementing lean practices in the 80s

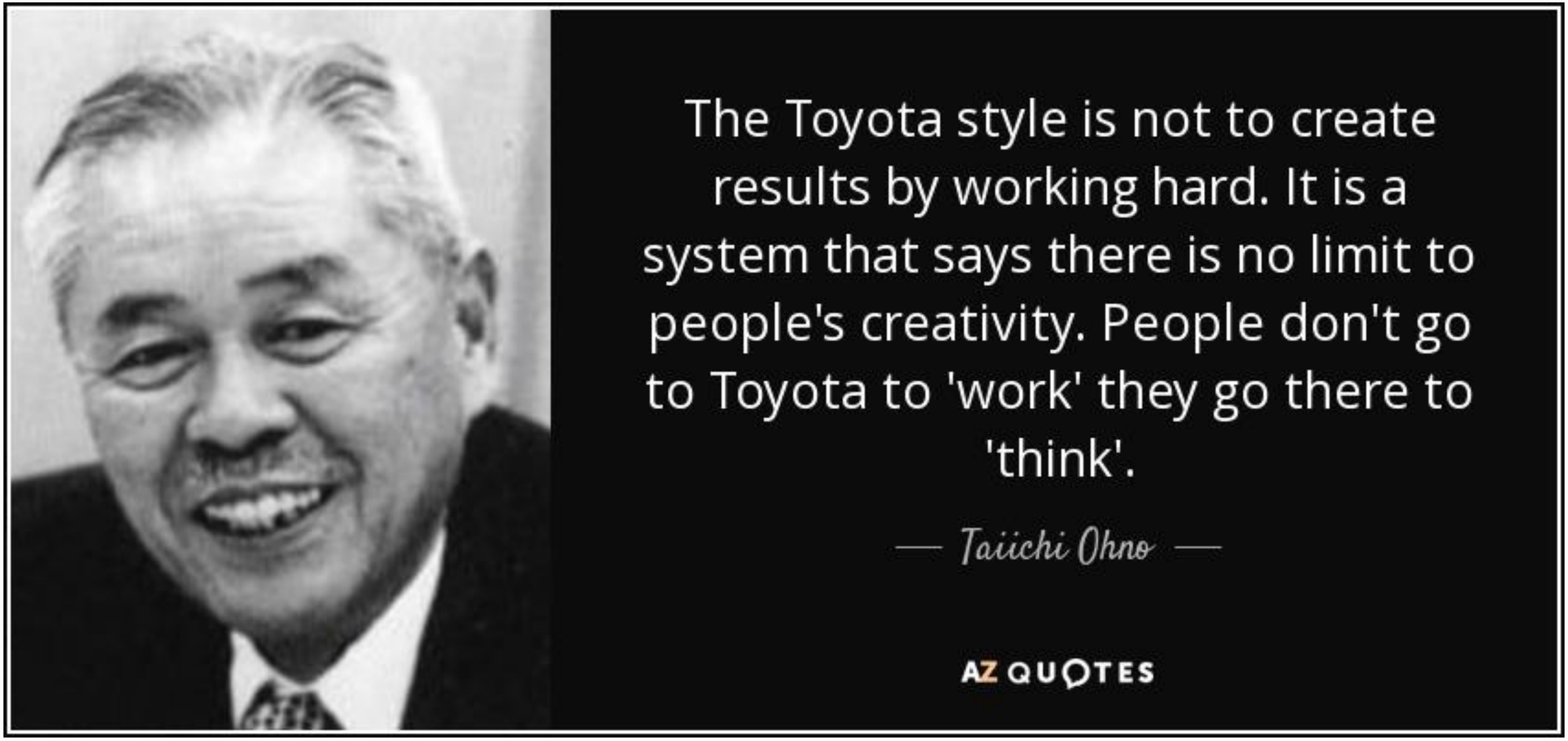

Taiichi Ohno
Toyota Production System



Any customer can have a car painted any colour that he wants so long as it is black.

(Henry Ford)

izquotes.com


The Toyota style is not to create results by working hard. It is a system that says there is no limit to people's creativity. People don't go to Toyota to 'work' they go there to 'think'.
— Taiichi Ohno —
AZ QUOTES

Toyota Production System :

How could Toyota make cars in small quantities but keep them as inexpensive as mass-produced cars?

“Just-in-time” manufacturing

“Don't decide what to manufacture until you have a customer order; then make it as fast as possible”

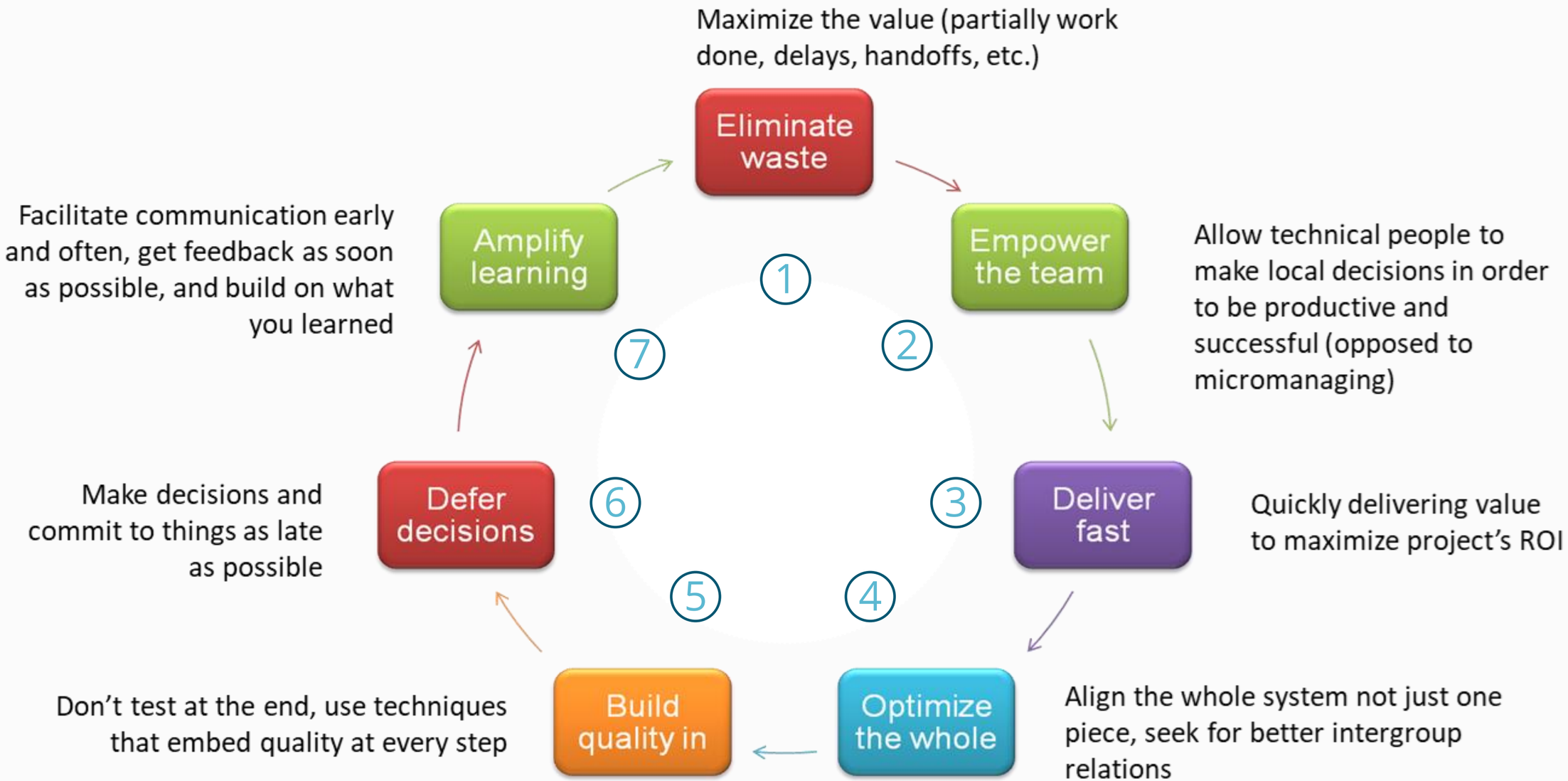
Case Study: Statewide Automated Child Welfare Information System (SACWIS)

- **Florida:** started in 1990, estimated 8 years and $32 million
  - In 2002 Florida spent $170 million and estimated to be completed in 2005 with $230 million
- **Minnesota:** started in 1999
  - completed in 2000 at cost of $1.1 million

- Why? Standardized infrastructure, minimized requirements, team of 8 capable people

Source: Standish Group

Lean Principles for Software Development



Lean Principles are… just Principles

- **Eliminate waste** does not mean throw away all documentation.
- **Empower the team** does not mean abandon leadership.
- **Deliver as fast as possible** does not mean rush and do sloppy work.
- **See the whole** does not mean ignore the details.
- **Build integrity in** does not mean big, upfront design.
- **Decide as late as possible** does not mean procrastinate.
- **Amplify learning** does not mean keep on changing your mind.

1. Eliminate waste

If a development cycle has collected requirements in a book gathering dust, that is waste

If developers code more features than are immediately needed, that is waste

Whatever gets in the way of rapidly satisfying a customer need is waste.

Handing off development from one group to another is waste

The Seven Wastes of ~~Manufacturing~~ Software Development

- ~~Inventory~~ Partially Done Work
- ~~Extra Processing~~ Extra Processes
- ~~Overproduction~~ Extra Features
- ~~Transportation~~ Task Switching
- Waiting
- Motion
- Defects

Shigeo Shingo, Toyota

Eliminate waste

WASTE

=

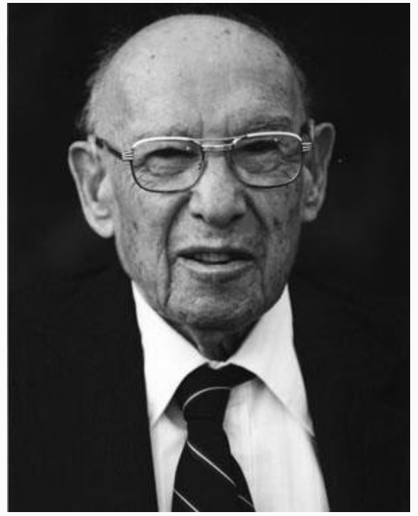
everything your organization does to develop software that is not analysis or

coding.

It is usually easier to see waste in a crisis

Eliminate waste

"There is nothing so useless as doing efficiently that which should not be done at all"



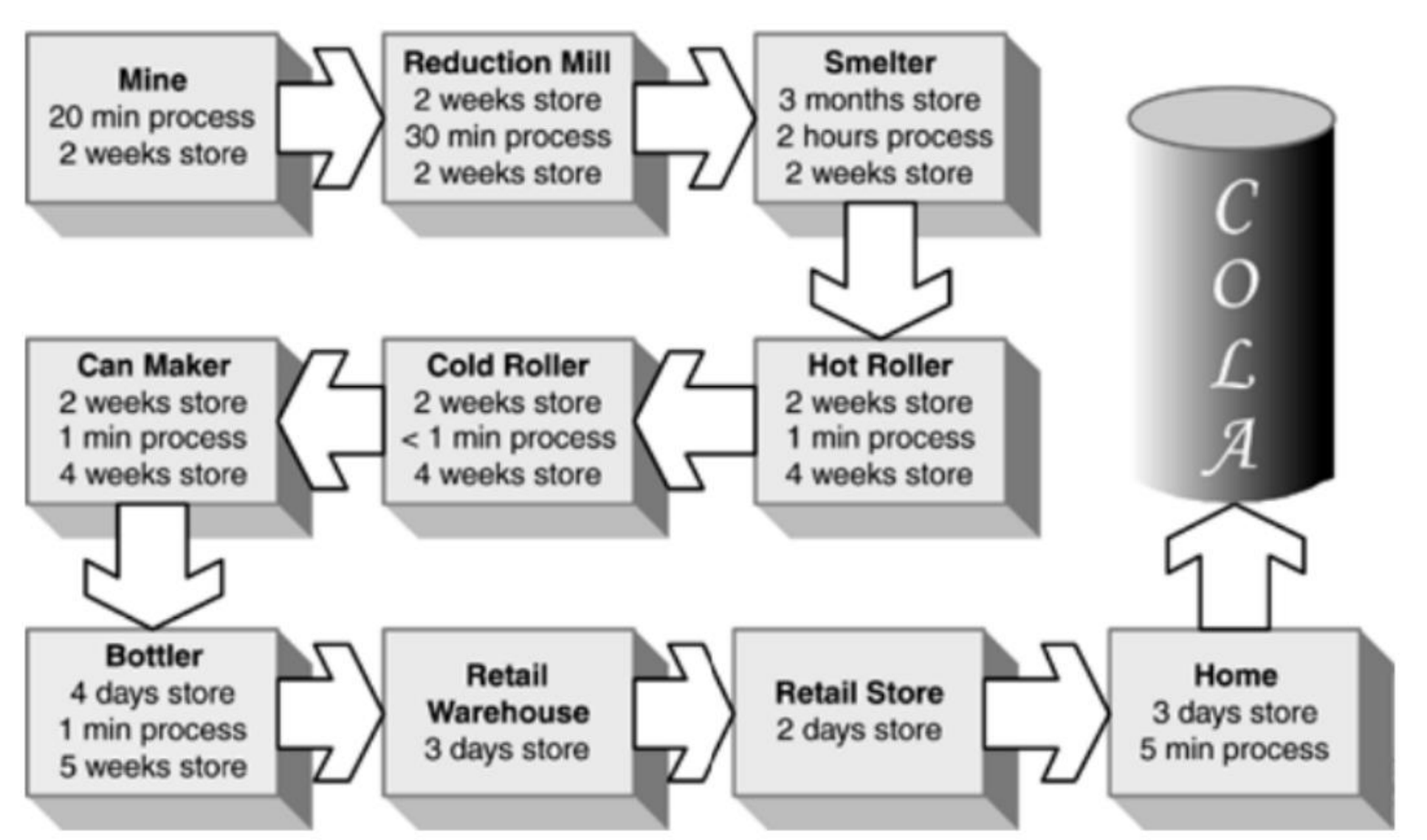
Peter Drucker

Eliminate waste

1. Implementing lean development is learning to see waste.
2. Uncover the biggest sources of waste and eliminate them.
3. Uncover the biggest remaining sources of waste and eliminate them.
4. Do it again.

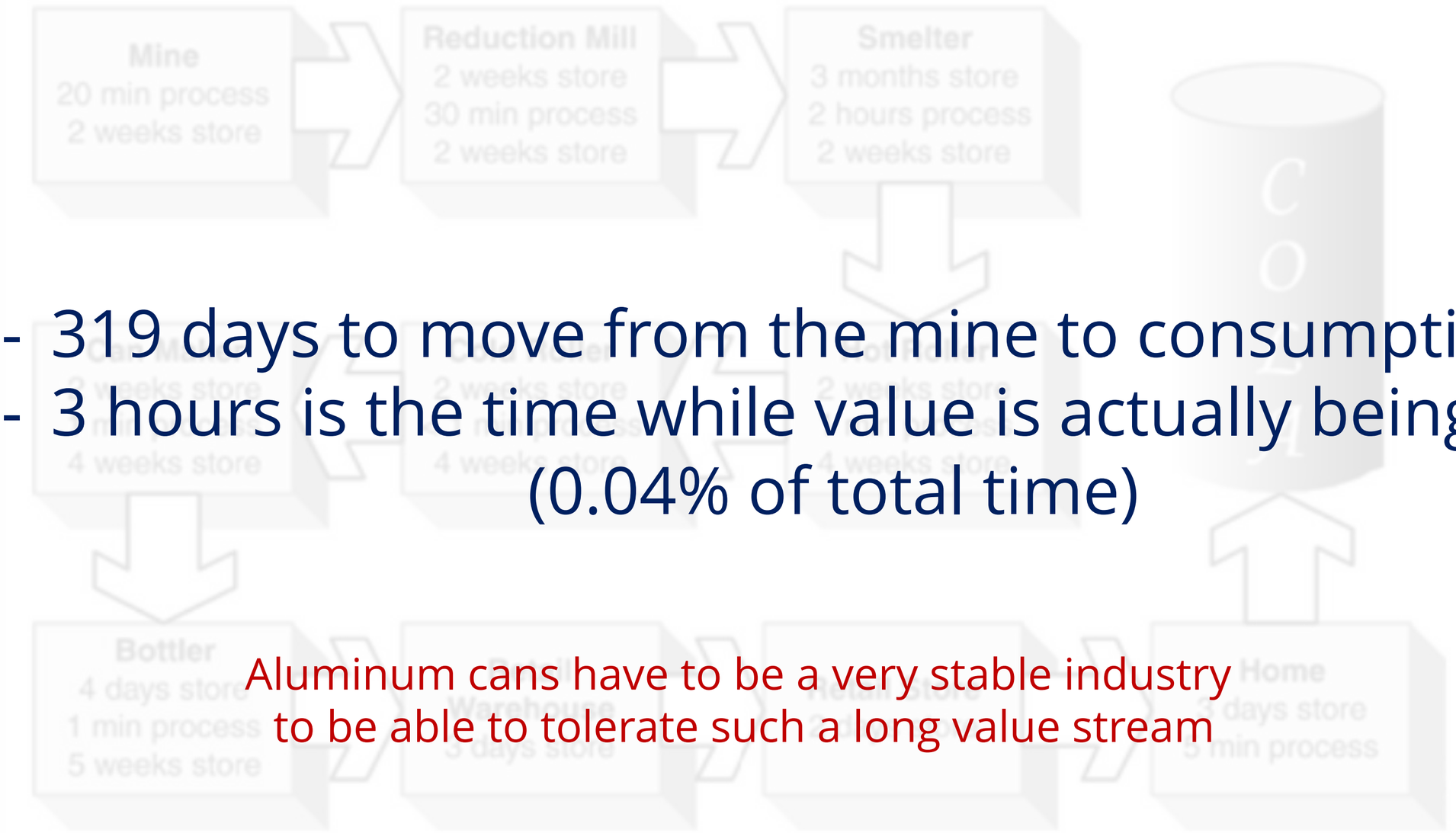
After a while, even things that seem essential can be gradually eliminated

Value Stream for Cola Cans



James P. Womack, Daniel T. Jones (1997)

Value Stream for Cola Cans



- 319 days to move from the mine to consumption
- 3 hours is the time while value is actually being added (0.04% of total time)

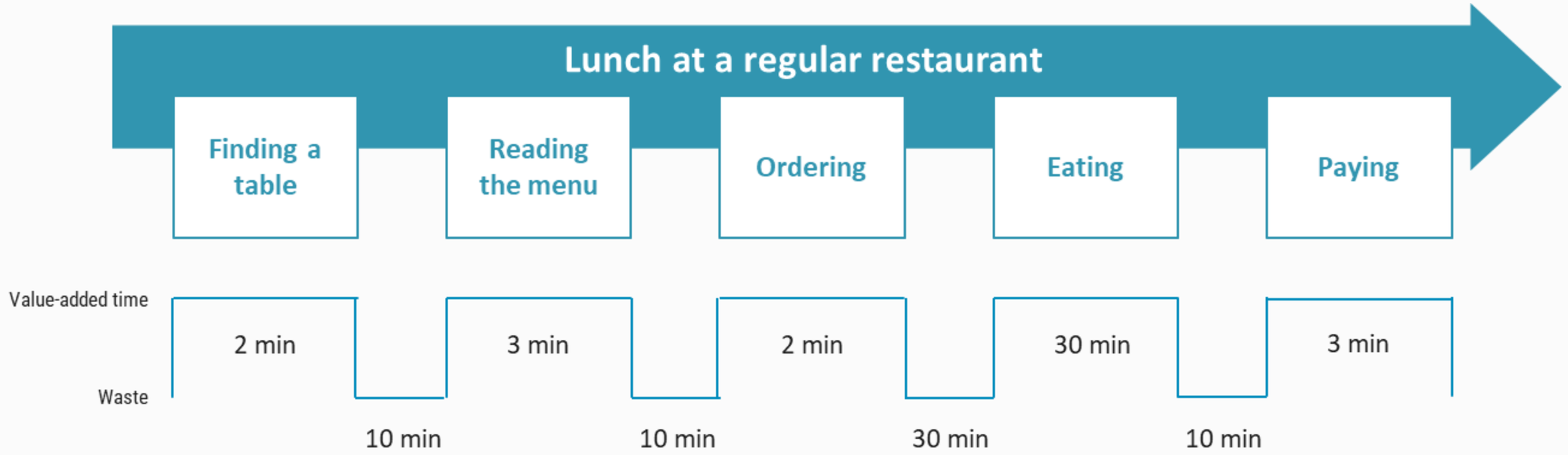
Aluminum cans have to be a very stable industry
to be able to tolerate such a long value stream

James P. Womack, Daniel T. Jones (1997)

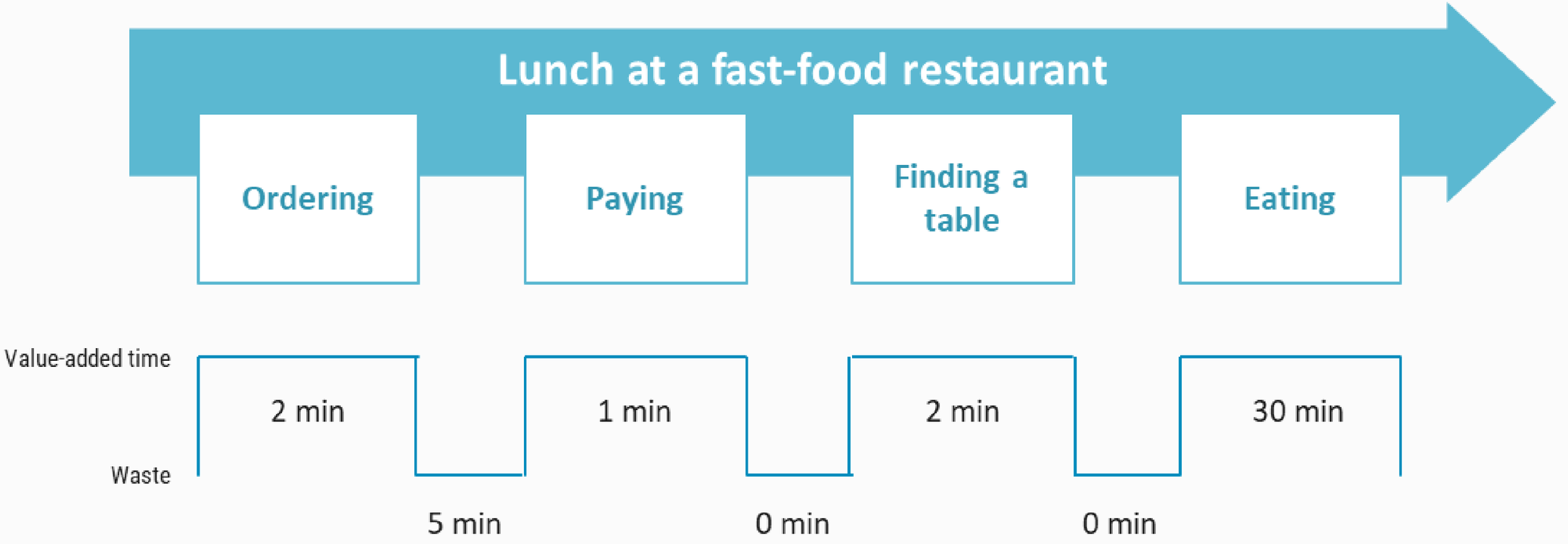
…not working for personal computers

Michael Dell

“8 days of inventory – competitors 40 days. If Intel comes out with a new chip, I am going to get that to the market 32 days sooner."

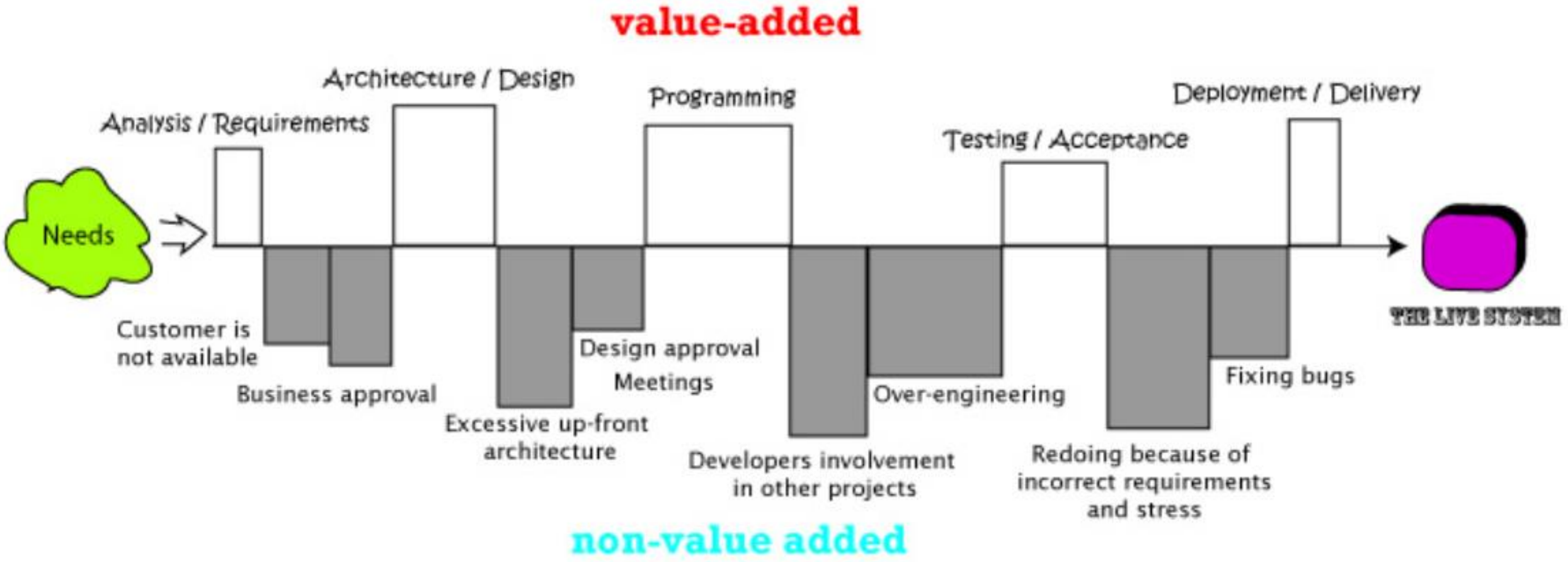


Total cycle time = 100 min
Value-added time = 40 min
Cycle efficiency = 40 / 100 = 40%



Total cycle time = 40 min
Value-added time = 35 min
Cycle efficiency = 35 / 40 = 88%

Eliminate waste



Eliminate waste

How to eliminate waste:

- Make a list of the 10 or 15 most important activities in your organization
- Rate 1-5 (1 customer do not care about , 5 customers value it highly)
- Develop a plan to cut those with 1 or 2 points

Eliminate waste

How to eliminate waste:

Develop a value stream map

Take the biggest cause of delay and plan to cut it in half

Eliminate waste

How to eliminate waste:

Seven meetings talk about the wastes in software development:

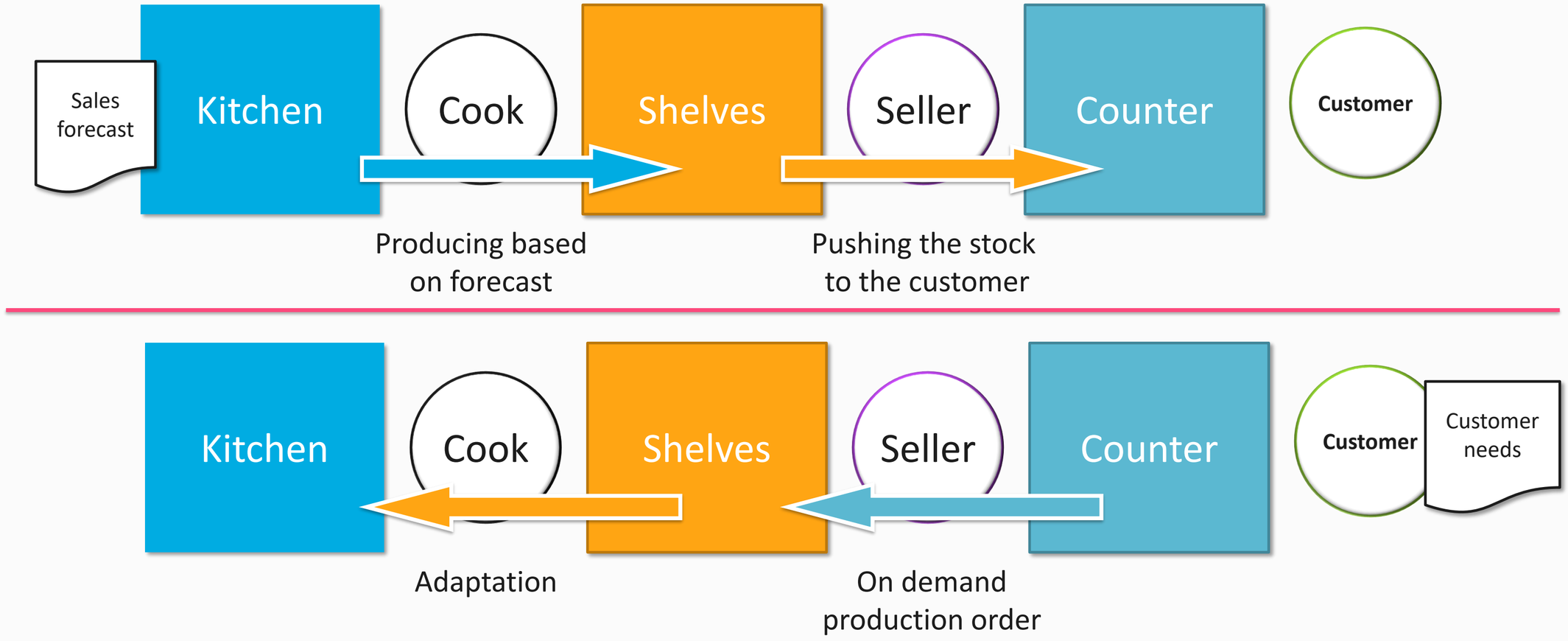
- Do you agree that this is waste? Why?
- How much time it consumes in avg / week
- What can we do to reduce that time

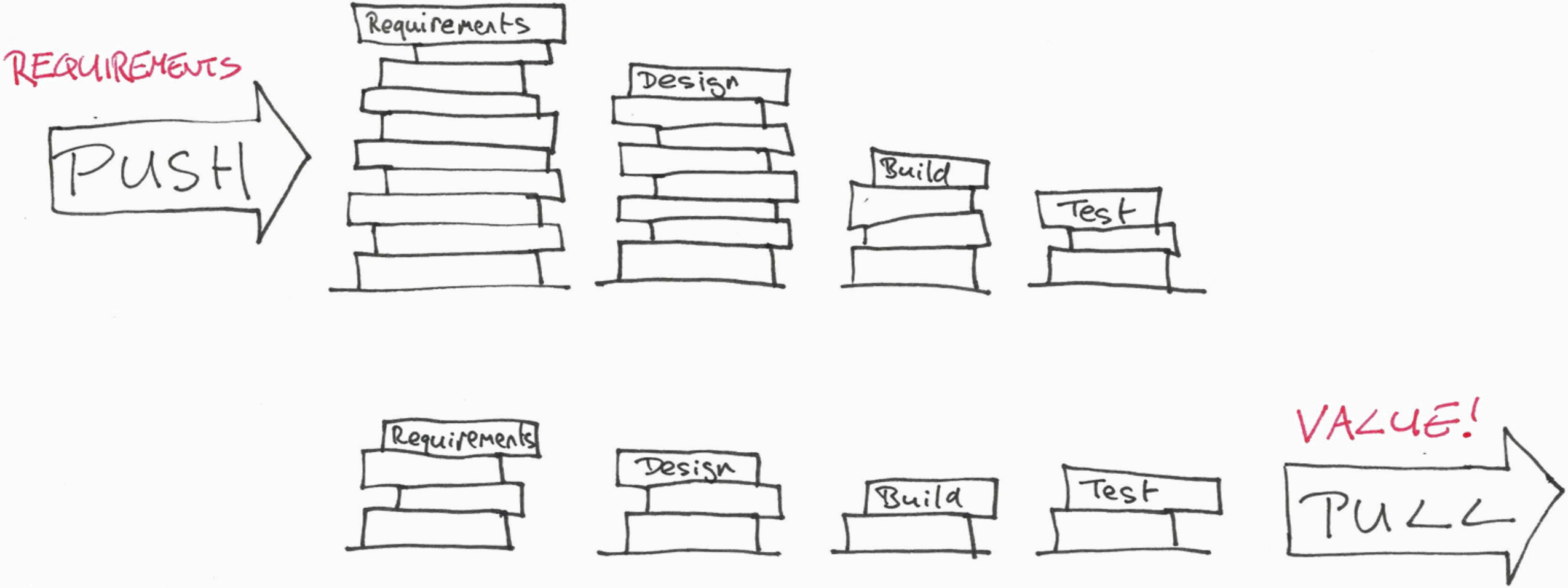
3. Deliver as fast as possible

- Customers like rapid delivery
- Rapid delivery means less time for customers to change their minds
- In-process, or partially done work can have undiscovered defects
- *Deliver as fast as possible* complements *decide as late as possible:* the faster you can deliver, the longer you can delay decisions.

3. Deliver as fast as possible

Push versus Pull




REQUIREMENTS
PUSH!
Requirements
Design
Build
Test
Requirements
Design
Build
Test
VALUE!
PULL

6. Decide as late as possible

In an evolving market, keeping design options open is more valuable than committing early.

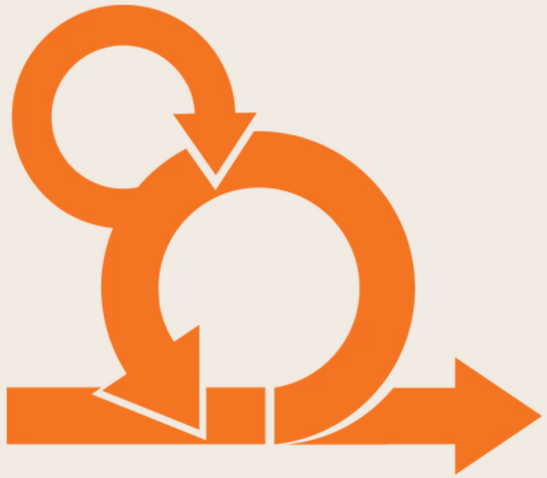
How to avoid change penalties?

- Traditional: make the right design decision in the first place and avoid the need to change later

- Lean: Don't make irreversible decisions in the first place; delay design decisions as long as possible, and when they are made, make them with the best available information to make them correctly

6. Decide as late as possible

- The last responsible moment:
  - delay commitment until the last responsible moment, that is, the moment at which failing to make a decision eliminates an important alternative.



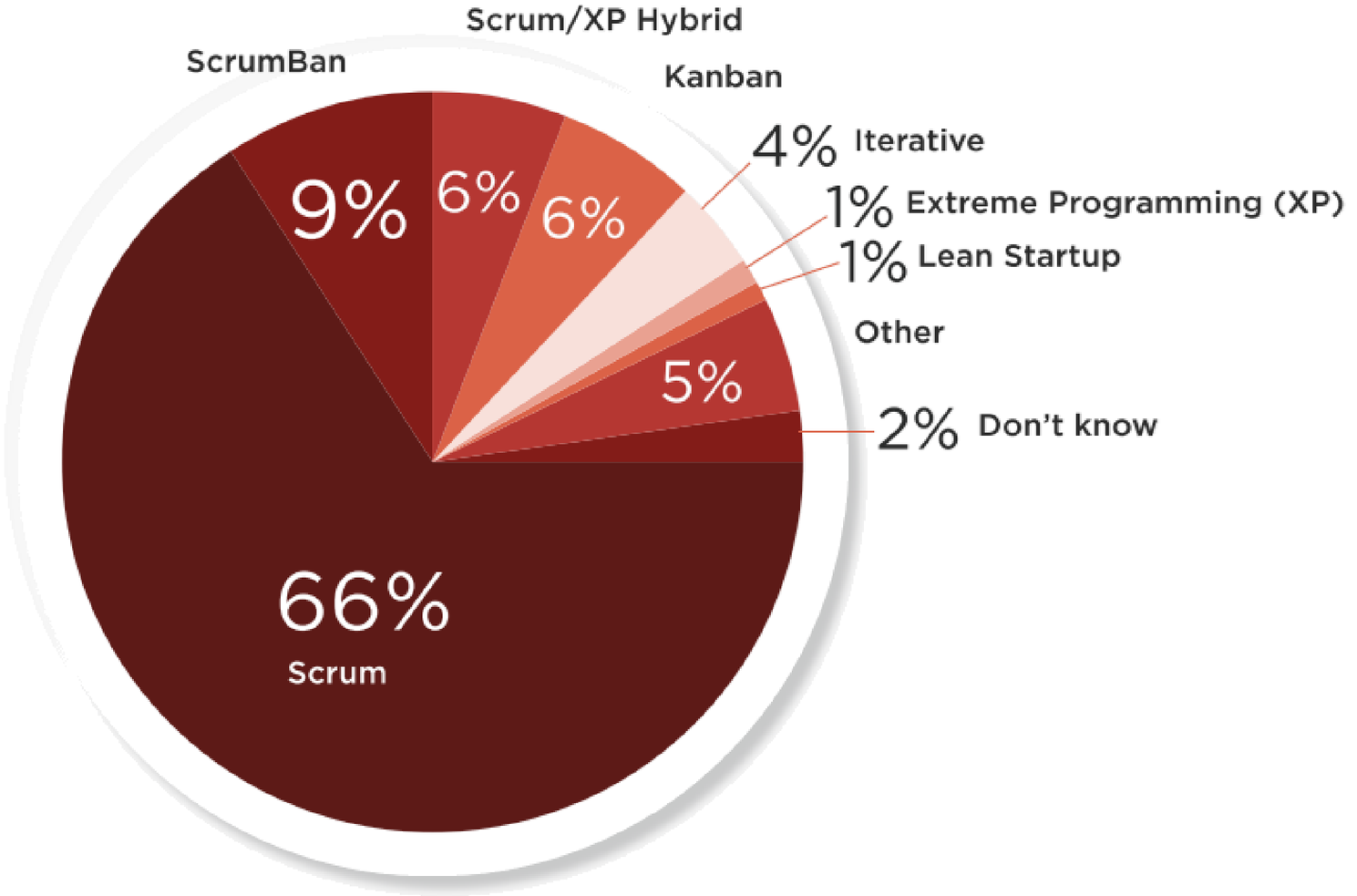
AGILE SOFTWARE DEVELOPMENT

Other Agile Methodologies & Practices

Most common Agile Methodologies

- Scrum
- XP
- Kanban

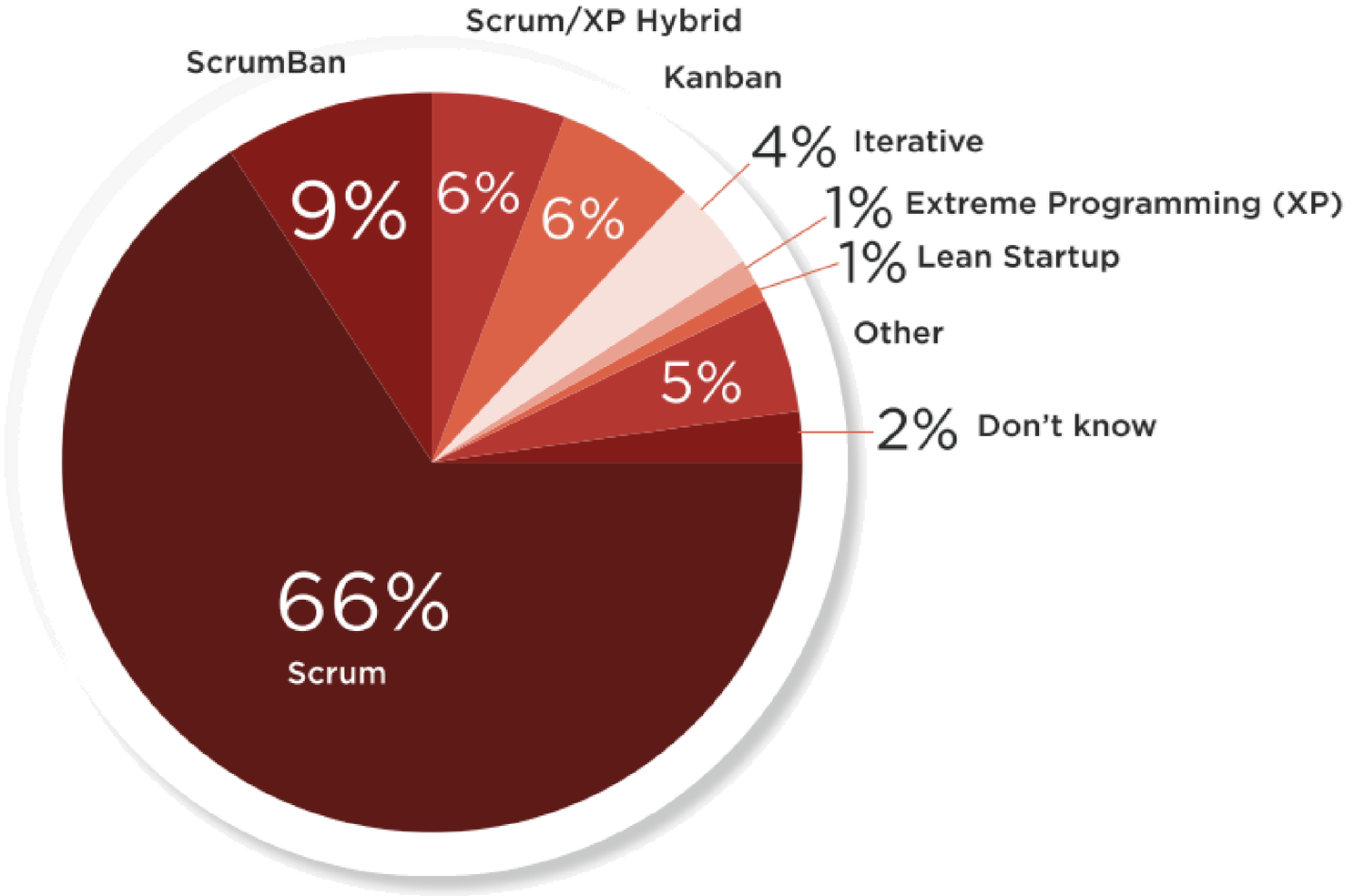
  => Lean



15th Annual State of Agile Report, 2021
https://stateofagile.com

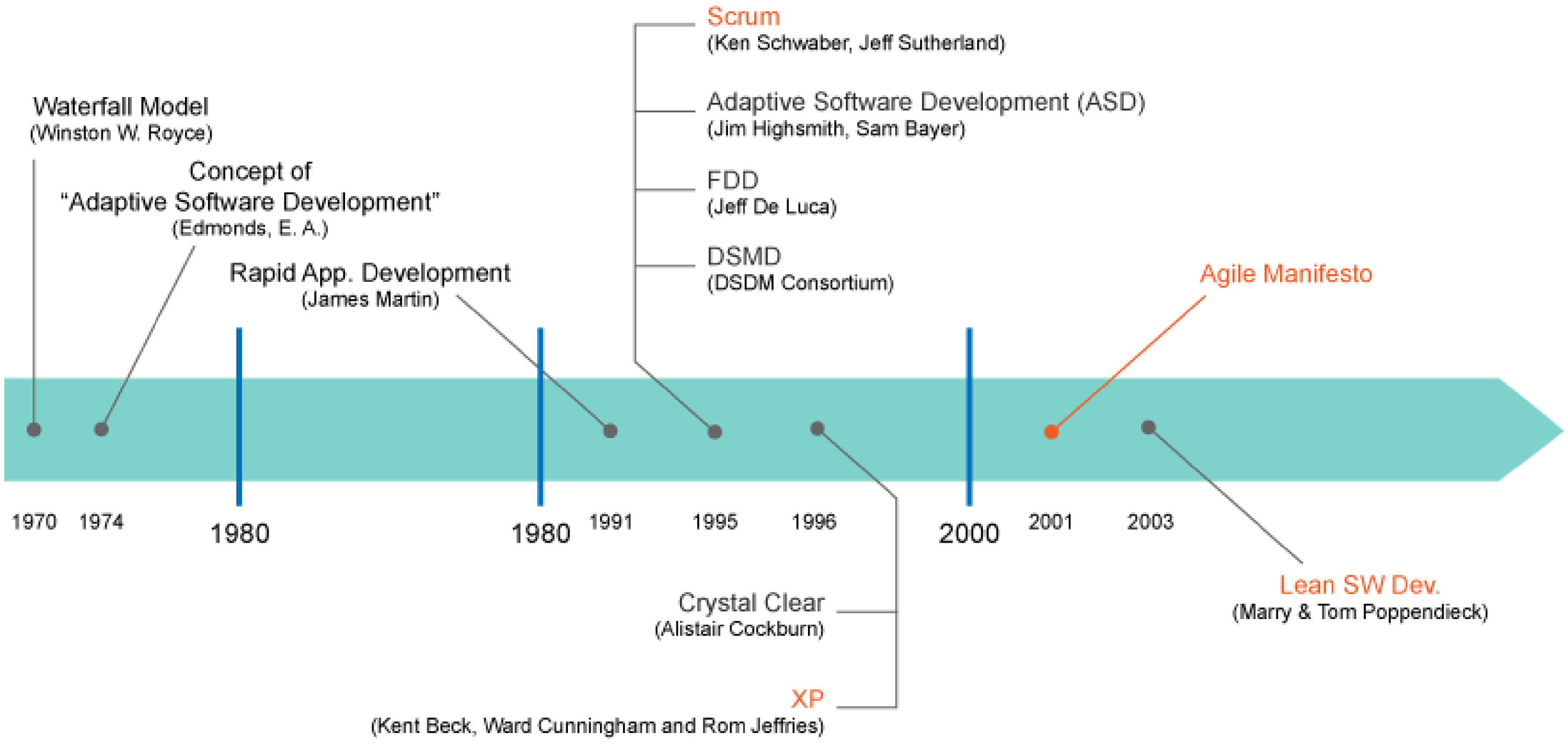
Most common Agile Methodologies

- Feature-Driven Development
- Agile Unified Process
- Crystal
- DSDM
- Disciplined Agile
- Scaled Agile Framework



15th Annual State of Agile Report, 2021
https://stateofagile.com

History of Agile



40 Agile Methods

- PRODUCT DEVELOPMENT
- FLOW
- KANBAN
- Modern Management Methods
- DEMING
  - PROFOUND KNOWLEDGE SYSTEM
- SCRUM
  - Scrumbut/ Scrumand
- and Lean Software Development
- LEAN
- XP
- CRYSTAL
- DSDM
- FDD
- ASD
- PERSONAL KANBAN
- SCRUM PLOP
  - PATTERN LANGUAGES OF PROGRAMS
- ETF
  - Enterprise Transition Framework
- LEAN STARTUP
- CERTIFICATIONS
- ACCELERATED AGILE
- HYBRID
  - SCRUMBAN
  - XANPAN
  - NONBAN
  - WATER SCRUM FALL
- AGILE UNIFIED PROCESS
- EXTREME (WIKISPEED) MANUFACTURING
- DISCIPLINED AGILE DELIVERY (DAD)
- TDD/ATD/BDD/SBE
  - Begin with the end in mind
- VANGUARD METHOD
- CONTEXT DRIVEN TESTING
- BEYOND BUDGETING
- HOLOCRACY
- RIGHTSHIFTING
- MANAGEMENT 3.0
- STOOS NETWORK
- RADICAL MANAGEMENT
- DRIVE
- CYNEFIN
- THEORY of CONSTRAINTS

CRAIG SMITH

OPEN YOUR EYES to OTHER METHODS

- ENTERPRISE UNIFIED PROCESS (EUP)
- LARGE SCALE SCRUM (LeSS)
- ENTERPRISE SCRUM
- SCALED AGILE FRAMEWORK (SaFe)
- XSCALE (AGILE TNG)
- DEVOPS
- SPOTIFY/SQUADIFICATION
- MIKADO METHOD
- PROGRAMMER ANARCHY
- MOB PROGRAMMING

by www.lynnecazaly.com

scrum australia

Feature Driven Development (FDD)

Feature Driven Development (FDD)

- Iterative/incremental software process
- Developed in 1997 by Jeff De Luca
- Domain Model is the core of FDD
  (no specific values / principles defined)

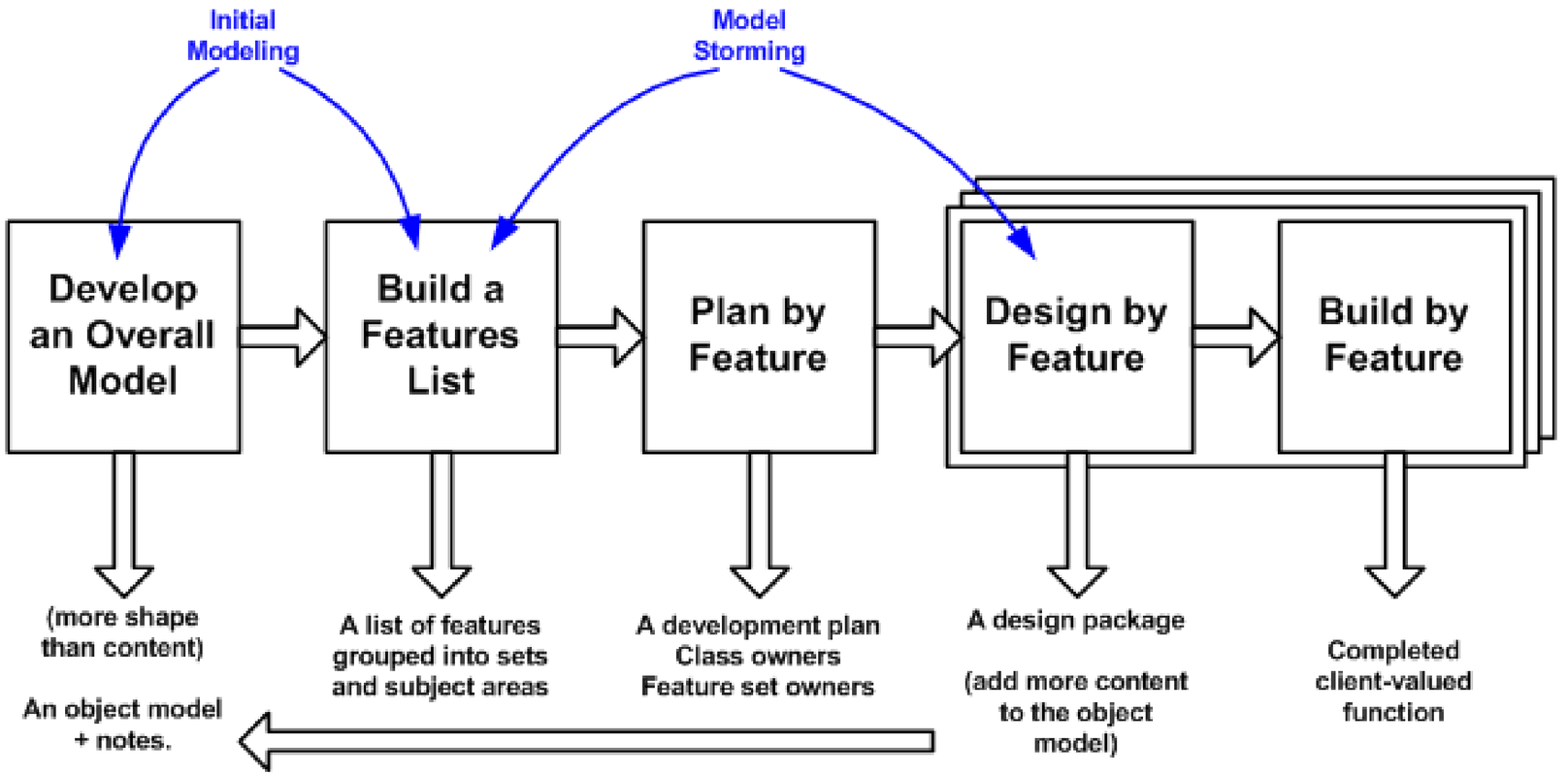
Feature Driven Development (FDD)

Requirements are gathered using a top-down approach

- Subject Areas (*general business practices*)
  - Feature Sets (*business activities*)
    - Features (*tasks*)

Typically 2 week iterations

Feature Driven Development (FDD)



Copyright 2002-2005 Scott W. Ambler
Original Copyright S. R. Palmer & J.M. Felsing

Feature Driven Development (FDD)

Everything is

planned,

designed,

built,

managed

at the feature level.

Feature Driven Development (FDD)

- Formula for defining features:

< action > < result > [of | to | for | from] < object >

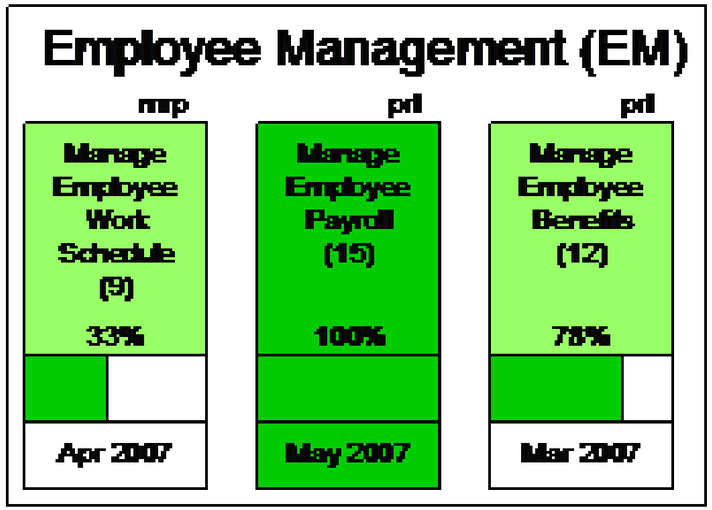
"Calculate monthly payment for car loan."

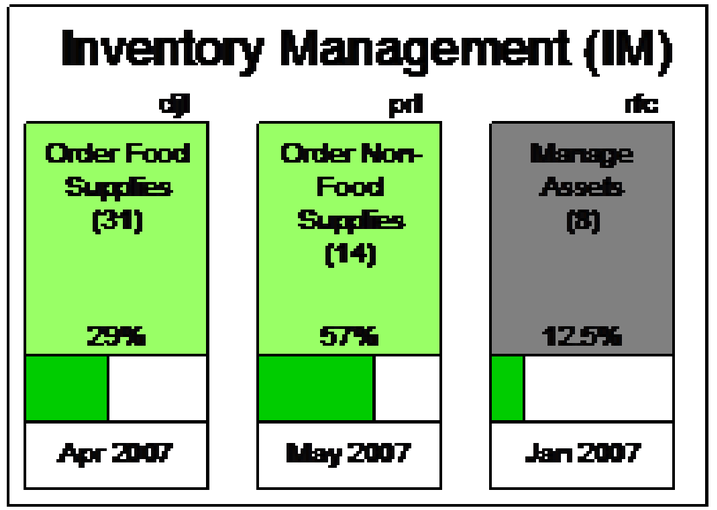
Feature Driven Development (FDD)

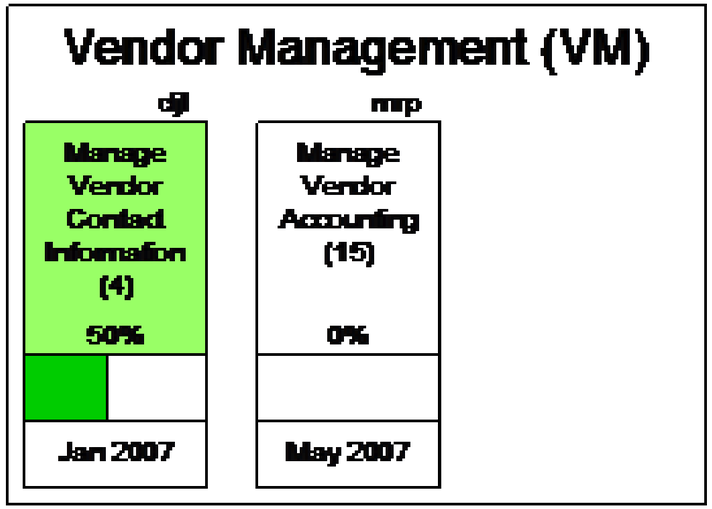
Roles:

- **Project Manager** –administrative, financial, reporting responsibilities
- **Chief Architect** –controls the design of the Domain Model manages the technical architecture, design sessions, and code reviews.
- **Development Manager** – manages daily development activities, coordinates the development team
- **Chief Programmer** –senior developer who is responsible for a specific Feature Set and manages their design and development activities.
- **Class Owner** –developer who reports to the CP and designs, codes, tests, and documents features
- **Domain Expert** – defines requirements as features that the solution must provide. Business analysts are the most common Des
- **Tester** – is responsible for validating that features perform as defined.
- **Deployer** – manages the data definitions and conversions and supports the deployment of code to the various platforms.
- **Technical Writer** – creates and maintains the documentation for users.

Feature Driven Development (FDD)







Parking Lot Chart

A visual, low maintenance way to report the progress of feature sets

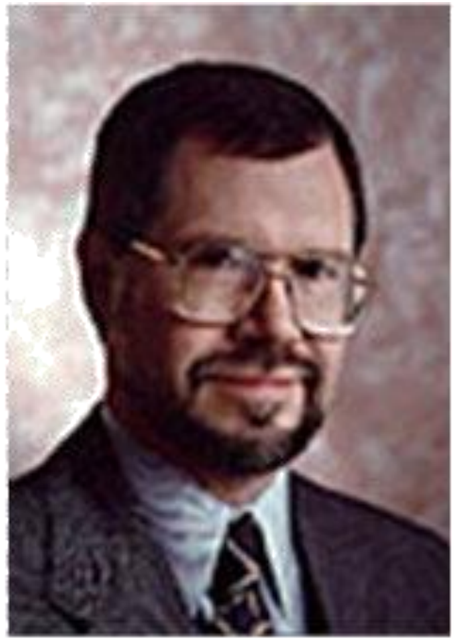
Feature Driven Development (FDD)

The exact state of each feature is documented in a table with six specific milestones

- Domain Walkthrough
- Design
- Design Inspection
- Code
- Code Inspection
- Promote to Build

Agile Unified Process

Agile Unified Process


"*Three Amigos*" ": Grady **Booch**, James **Rumbaugh**, Ivar **Jacobson**

Early 90's: Unified Modeling Language (UML)

Founders of Rational Software Corporation (today division of IBM)

Agile Unified Process

= simplified version of *Rational Unified Process*

(EssUP – *Essential Unified Process* – first attempt to simplify RUP by I. Jacobson)

- “high ceremony” framework
- based on integration of different agile concepts and techniques

Agile Unified Process

6 philosophies

- **Competence –** The team knows what it's doing. They won't read detailed process documentation, instead will apply high-level guidance and standards.
- **Simplicity –** Describe things concisely on a few pages, not reams of pages.
- **Agility –** Conforms to the values and principles of the Agile Alliance.
- **Activity –** Focus on only the high-value activities that count. Ignore the noise.
- **Tools –** Simple tools are often the best. Recommends using the tools best suited for the job.
- **Tailor –** AUP works best when tailored to the needs defined by the context.

Agile Unified Process

Phases

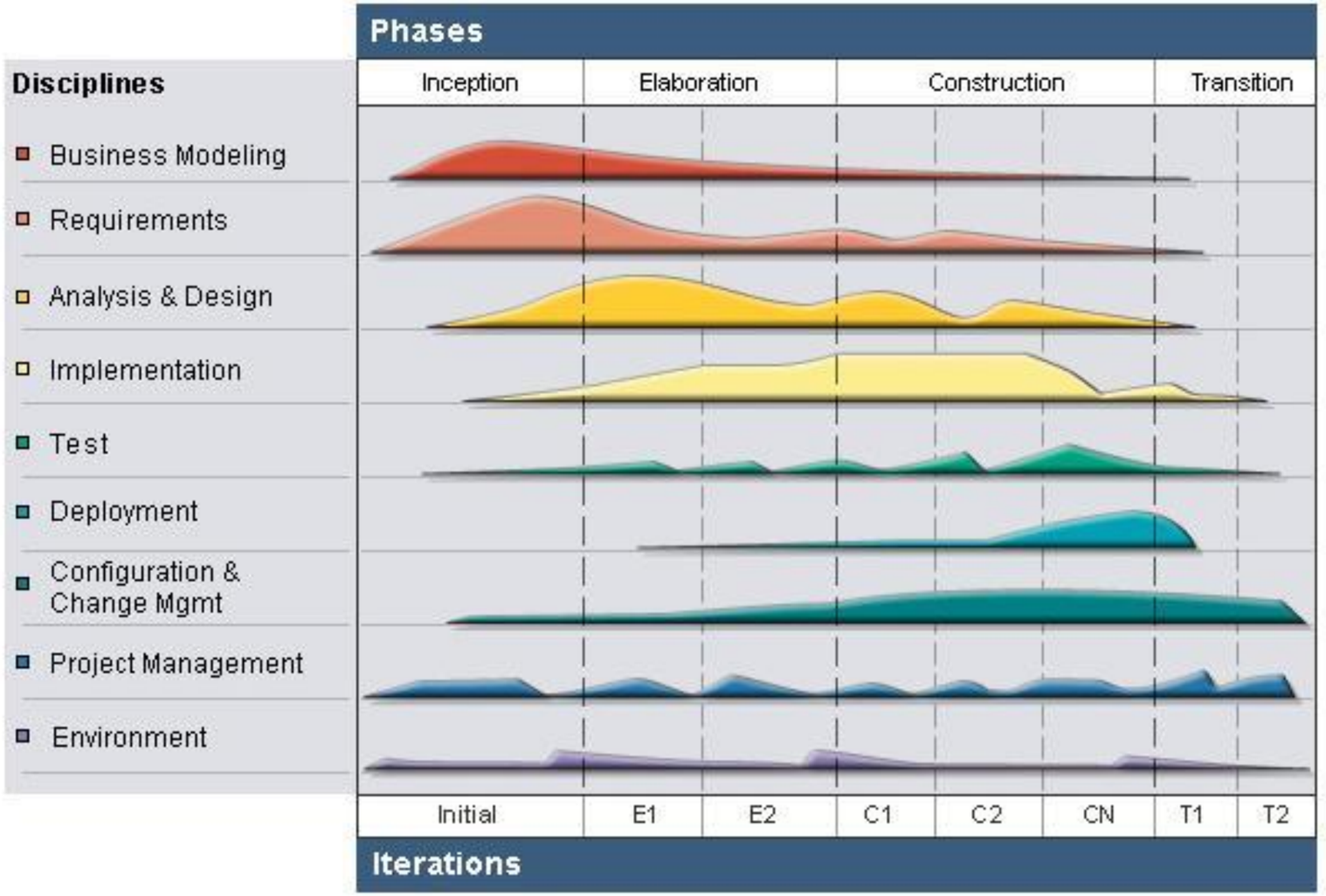
- **Inception** – cultivates a shared understanding of the project scope and defines architectural choices.
- **Elaboration** – develops the understanding of the system into requirements and validates architectural choices.
- **Construction** – occurs until system development is completed.
- **Transition** – all testing and system deployment to production.

Agile Unified Process

Disciplines

- **Model –** Use a model to represent the organization's business approach, the problem domain, and any viable solution to solve the problem.
- **Implement –** Code the model(s) into executable code and perform unit testing.
- **Test –** Apply additional tests to find defects, validate the system design works, verify the requirements are satisfied, and ensure code quality.
- **Deploy –** Plan and deliver the system for end users.
- **Configuration Management –** Control all project artifacts, including version tracking and change management.
- **Project Management –** Provide project management, including scope, resource, risk and progress management, and coordination of external interfaces, to achieve an on time, on budget completion.
- **Environment –** Provide process guidance standards and ensure needed tools are available for the team

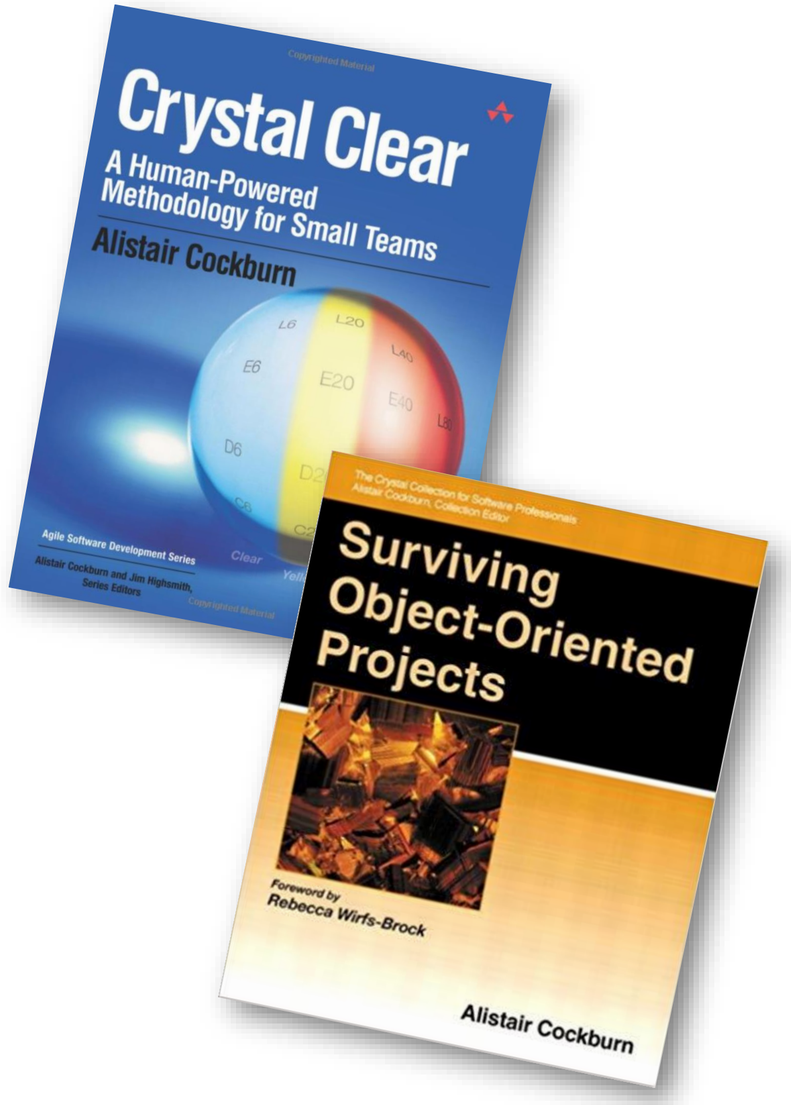
Agile Unified Process



Crystal

Crystal

- Alistair Cockburn, 2004
- Family of frameworks
  - based on size & criticality
  - not upward/downward compatible



Crystal

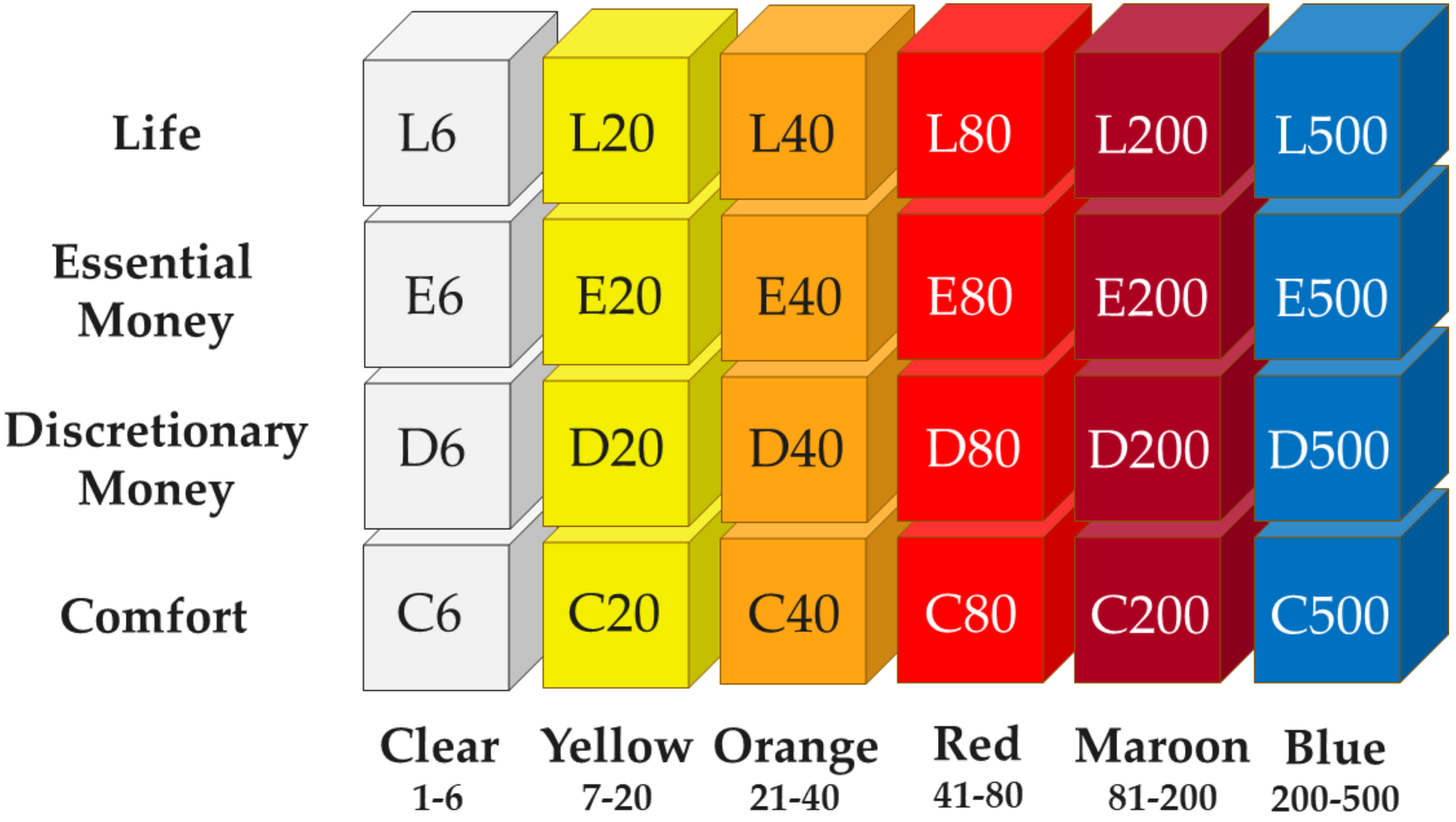
Key Principles

- **Frequent Delivery**: Project owners/customers can expect deliverables from the team(s) every couple of months.
- **Continual Feedback**: The entire project team & stakeholders meets on a regular basis to discuss project activities.
- **Constant Communication**: Teams co-located in the same room/facility. All projects expect to have frequent access to the person(s) defining the requirements.
- **Safety**: 1. The safe zone that team members must have to be effective and to communicate truth during the project.
  2. Evaluate how software projects affect the safety of their end-users.
- **Focus**: There should be enough time to complete priority items each without interruption.
- **Access to Users**: Project team will have access to one or more users of the system being built.
- **Automated Tests and Integration**: Controls must be put in place to support versioning, automated testing, and frequent integration of system components.

Crystal

- **Size**: number of people involved in the project.
  - Bigger size – more formality to the structure, artifacts and management
- **Criticality**: the potential for the system to cause damage
  - More critical: increase the rigidity of the project needs

Crystal


Life
Essential Money
Discretionary Money
Comfort
L6
L20
L40
L80
L200
L500
E6
E20
E40
E80
E200
E500
D6
D20
D40
D80
D200
D500
C6
C20
C40
C80
C200
C500
Clear
1-6
Yellow
7-20
Orange
21-40
Red
41-80
Maroon
81-200
Blue
200-500

Crystal Clear

- Has the fewest defined roles:
  - Sponsor
  - Senior Designer
  - Programmer
- Roles of *project manager*, *business analyst*, *tester*, etc. are shared among all team members.
- The expected release is every 60 or 90 days
- Minimal documentation (*project milestones*)

Crystal Orange

- Roles:
  - Sponsor
  - Project Manager
  - Business Analyst
  - Architect
  - Senior Designer
  - Programmer
  - Tester
- The expected release is every 90 or 120 days

Crystal Orange (cont)

- Specific Deliverables:
  - Requirements Document
  - Release Sequence (Schedule)
  - Project Schedule
  - Status Reports
  - UI Design Document (*if needed*)
  - Object Model
  - User Manual
  - Test Cases

Dynamic Systems Development Method DSDM

Dynamic Systems Development Method

- UK, 1990,
- DSDM Consortium (manages DSDM framework versions)
- Mot popular Agile methodology practice in UK
- Developed as an extension for RAD (Rapid Application Development)
- One of the *heavier* Agile approaches

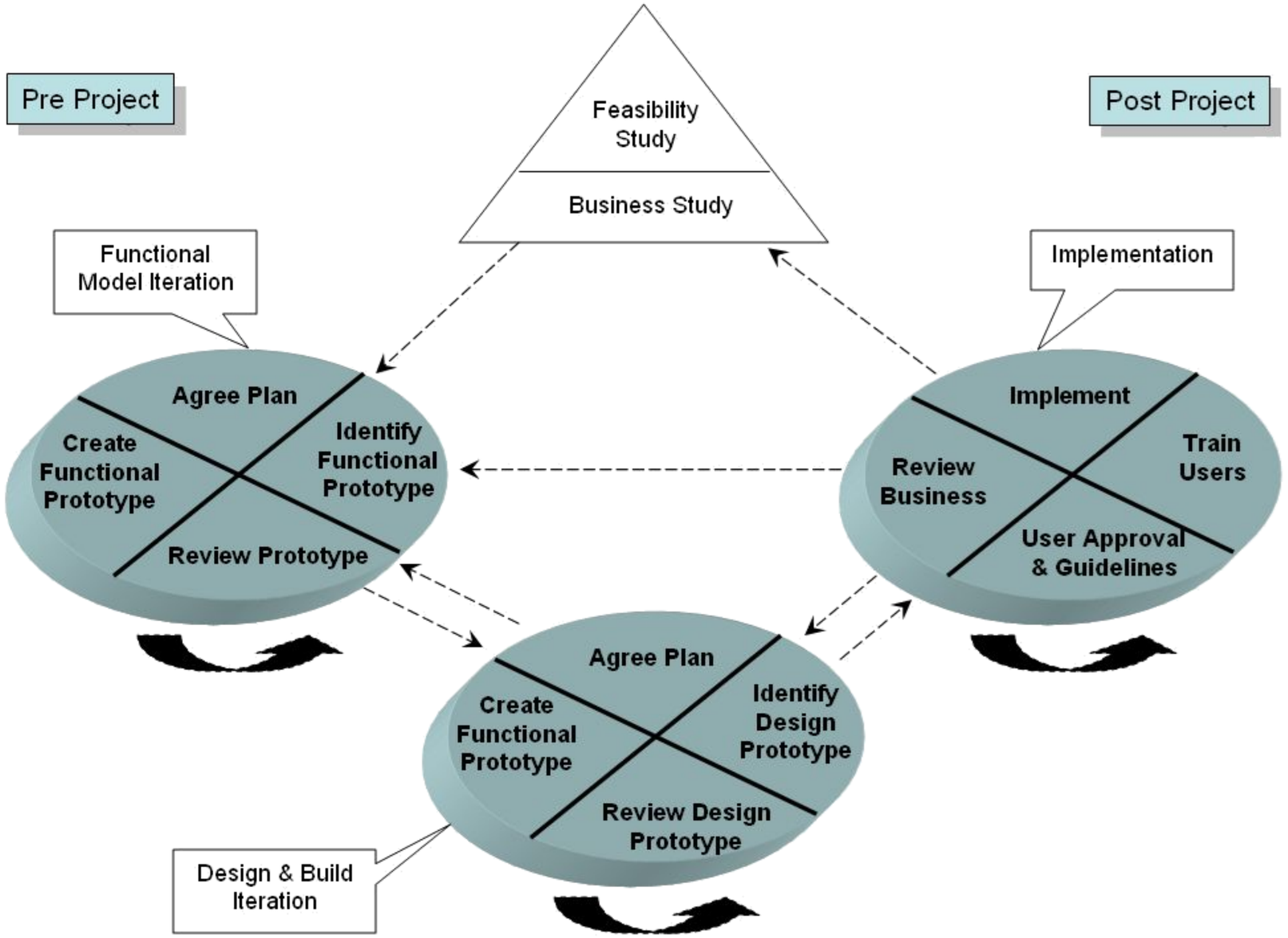
Dynamic Systems Development Method

- Principles
  - Focus on the business need
  - Deliver on time
  - Collaborate
  - Never compromise quality
  - Build incrementally from firm foundations
  - Develop iteratively
  - Communicate continuously and clearly
  - Demonstrate control

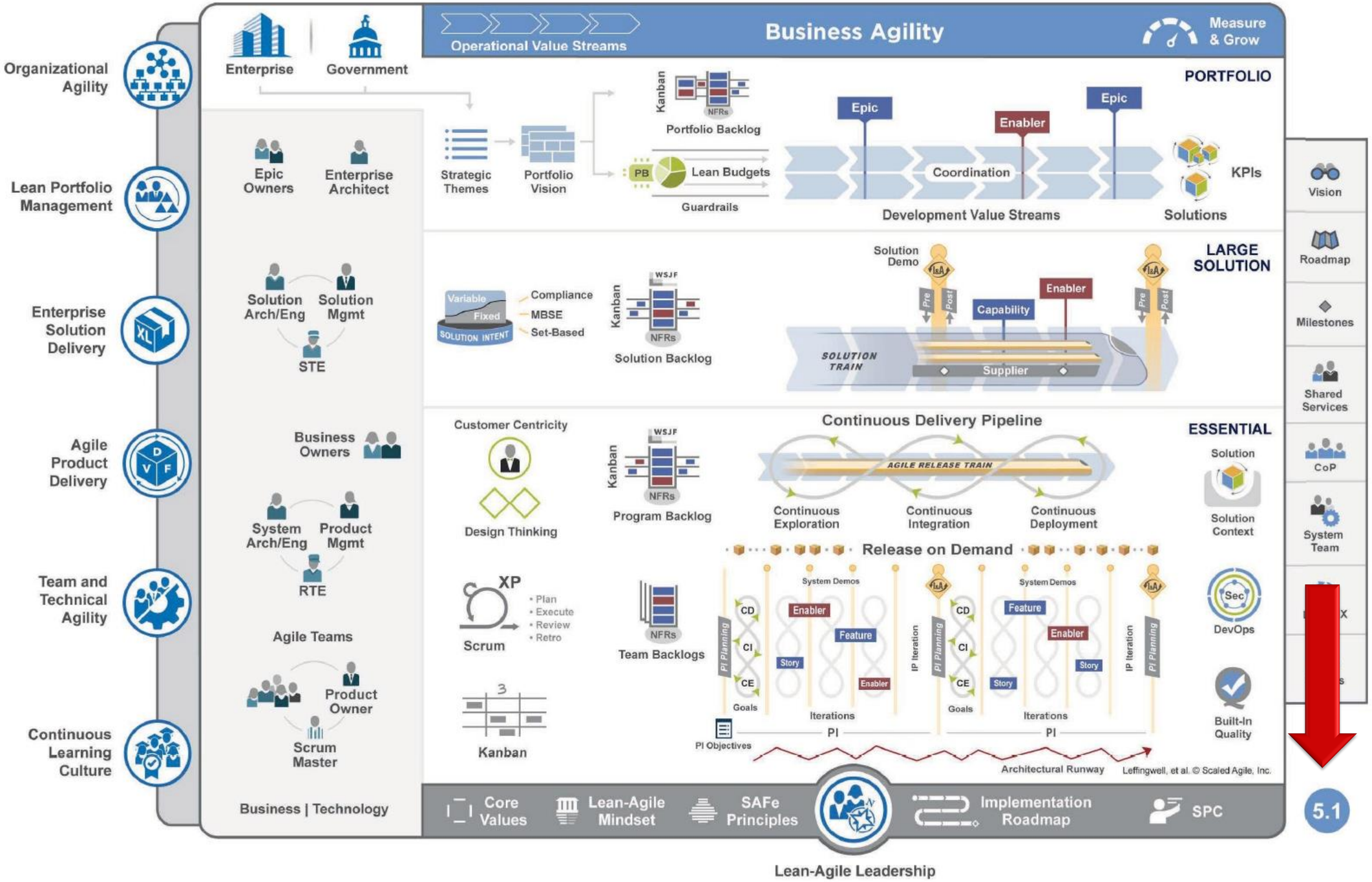
Dynamic Systems Development Method

- Phases:
  - **Pre Project**: Things that need to occur before the project begins.
  - **Project Lifecycle**: The actual project occurs. This phase is broken into five stages:
    - Feasibility Study
    - Business Study
    - Functional Model Iteration
    - Design and Build Iteration
    - Implementation
  - **Post Project**: Things that need to occur after the project has been completed.

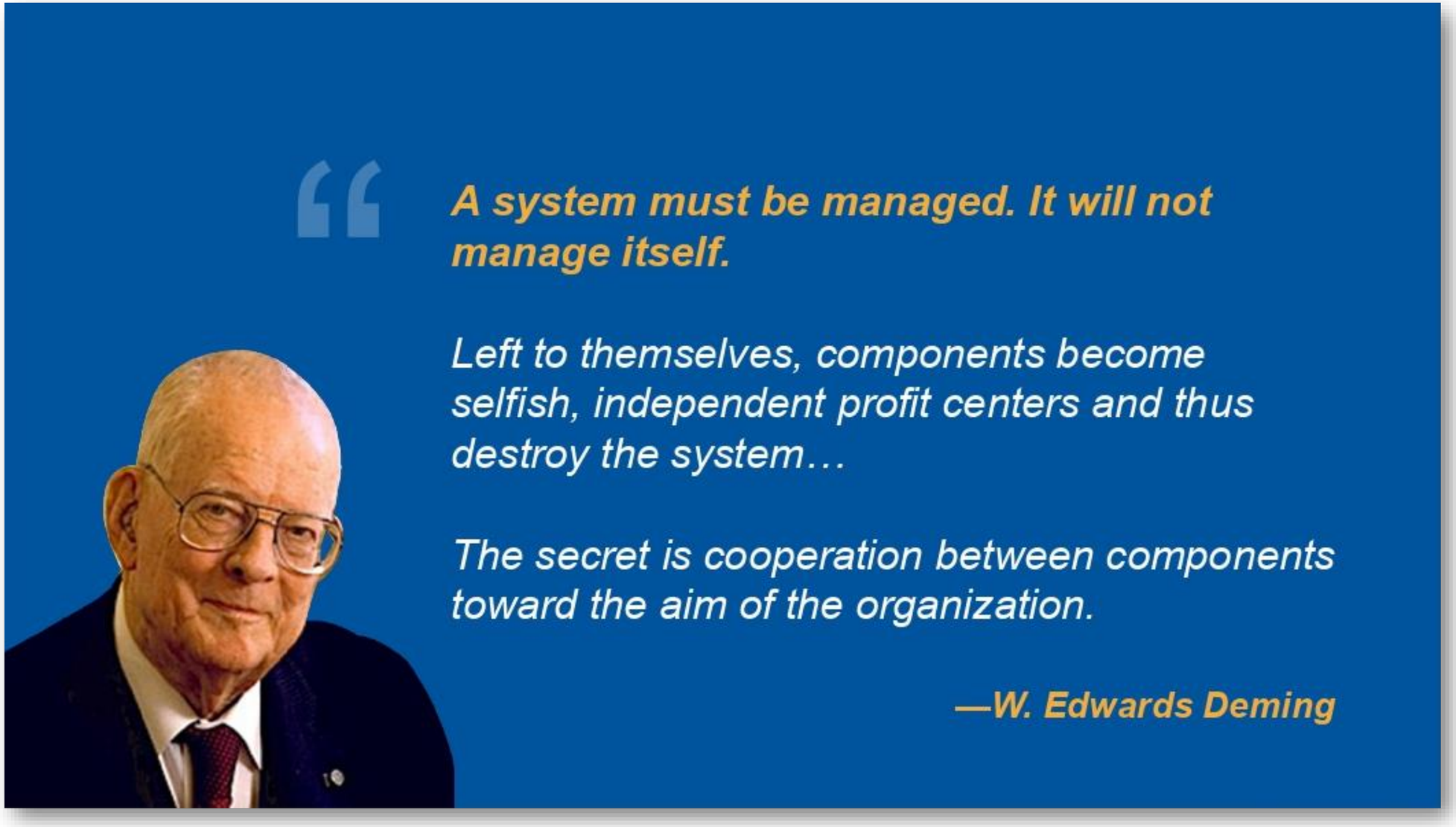
Dynamic Systems Development Method



Scaled Agile Framework
SAFe


Organizational Agility
Lean Portfolio Management
Enterprise Solution Delivery
Agile Product Delivery
Team and Technical Agility
Continuous Learning Culture
Enterprise
Government
Operational Value Streams
Business Agility
Measure & Grow
PORTFOLIO
Epic Owners
Enterprise Architect
Strategic Themes
Portfolio Vision
Kanban
NFRs
Portfolio Backlog
PB
Lean Budgets
Guardrails
Epic
Enabler
Epic
Coordination
KPIs
Development Value Streams
Solutions
LARGE SOLUTION
Solution Arch/Eng
Solution Mgmt
STE
Variable
Fixed
SOLUTION INTENT
Compliance
MBSE
Set-Based
WSJF
Kanban
NFRs
Solution Backlog
Solution Demo
Pre
Post
Capability
Enabler
SOLUTION TRAIN
Supplier
ESSENTIAL
Business Owners
Customer Centricity
Design Thinking
Continuous Delivery Pipeline
AGILE RELEASE TRAIN
Continuous Exploration
Continuous Integration
Continuous Deployment
Solution
Solution Context
System Arch/Eng
Product Mgmt
RTE
Agile Teams
XP
Plan
Execute
Review
Retro
Scrum
Program Backlog
Release on Demand
System Demos
CD
CI
CE
Enabler
Feature
Story
PI Planning
IP Iteration
Goals
Iterations
PI
PI Objectives
Team Backlogs
DevOps
Built-In Quality
Product Owner
Scrum Master
Kanban
Architectural Runway
Leffingwell, et al. © Scaled Agile, Inc.
Business | Technology
Core Values
Lean-Agile Mindset
SAFe Principles
Implementation Roadmap
SPC
Lean-Agile Leadership
Vision
Roadmap
Milestones
Shared Services
CoP
System Team
5.1

SAFe Specifics



SAFe Specifics

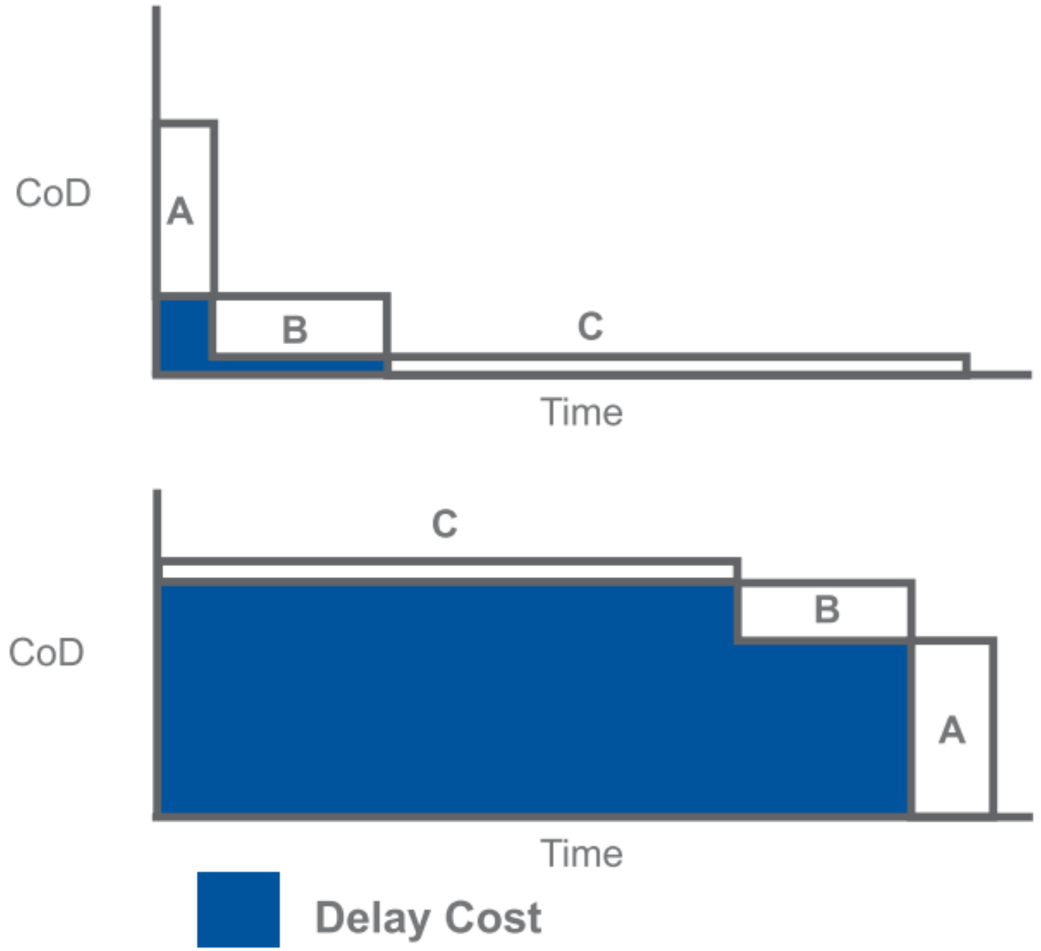
- ART = a team of (cross-functional) teams
- Common cadence
- Agreement inside ART on the meaning of story points

"*Your Customer is whoever consumes your work.*"

SAFe Specifics

- *Program Increment* planning: each 5 iterations
- Prioritization is done based on *Cost of Delay* & *Weighted Shortest Job First*

SAFe Specifics



Feature	Duration	CoD
A	1	10
B	3	3
C	10	1

$$\text{WSJF} = \frac{\text{Cost of delay}}{\text{Job duration (Job size)}}$$

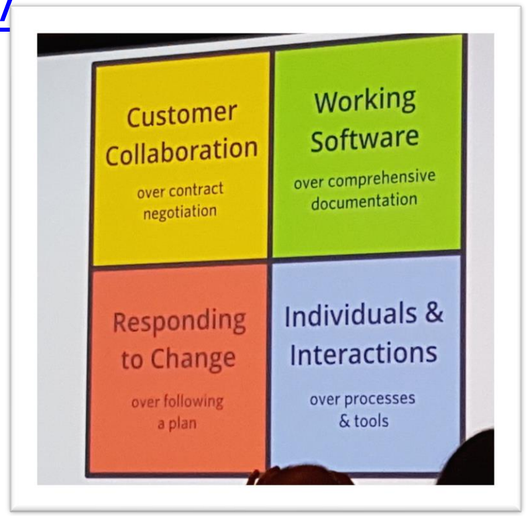
Modern Agile

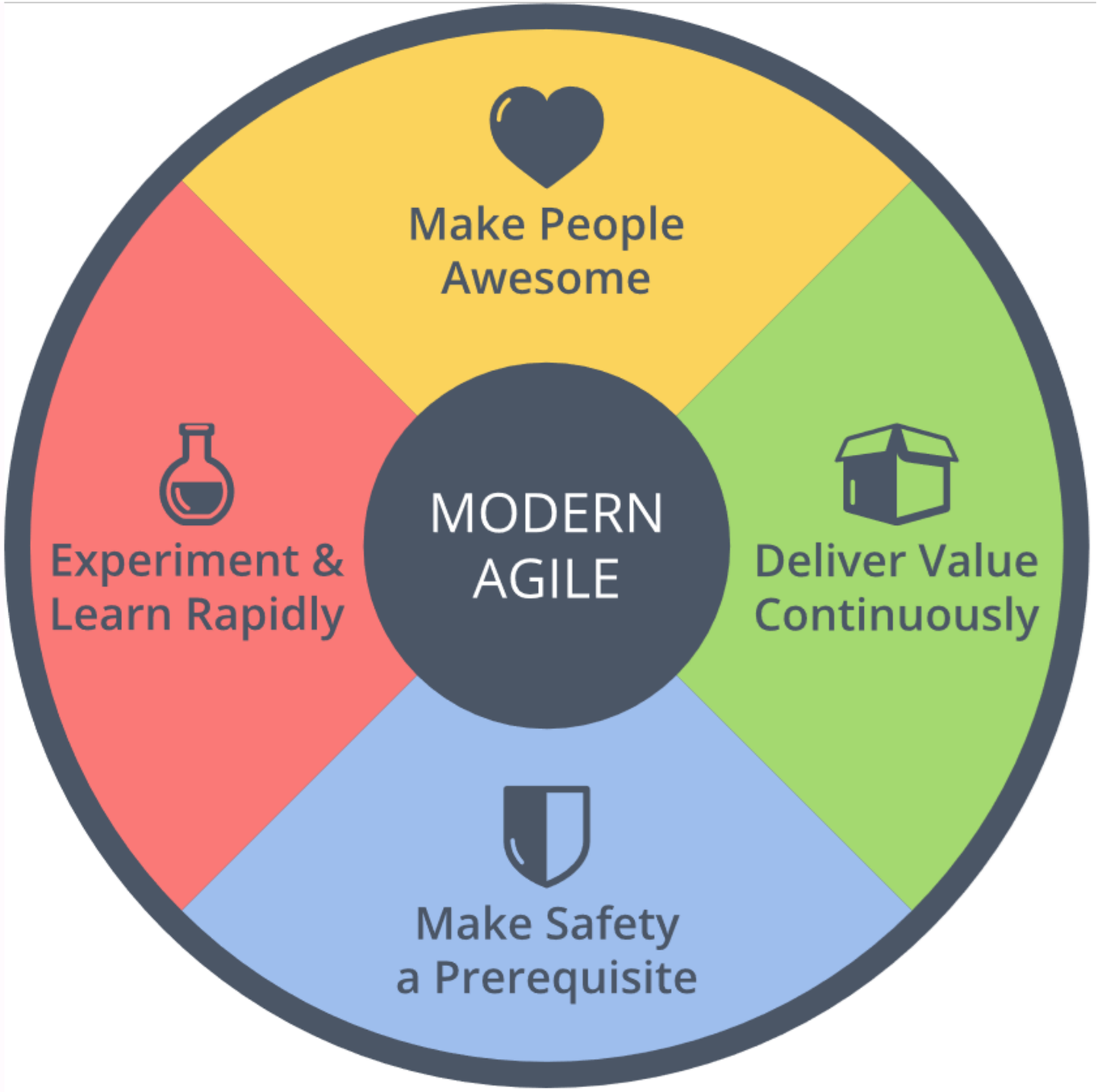
Modern Agile *is a community for people interested in uncovering better ways of getting awesome results. It leverages wisdom from many industries, is principle driven and framework free.*

Joshua Kerievsky, *CEO, Industrial Logic*

2015 – Joshua Kerievsky

modernagile.org/

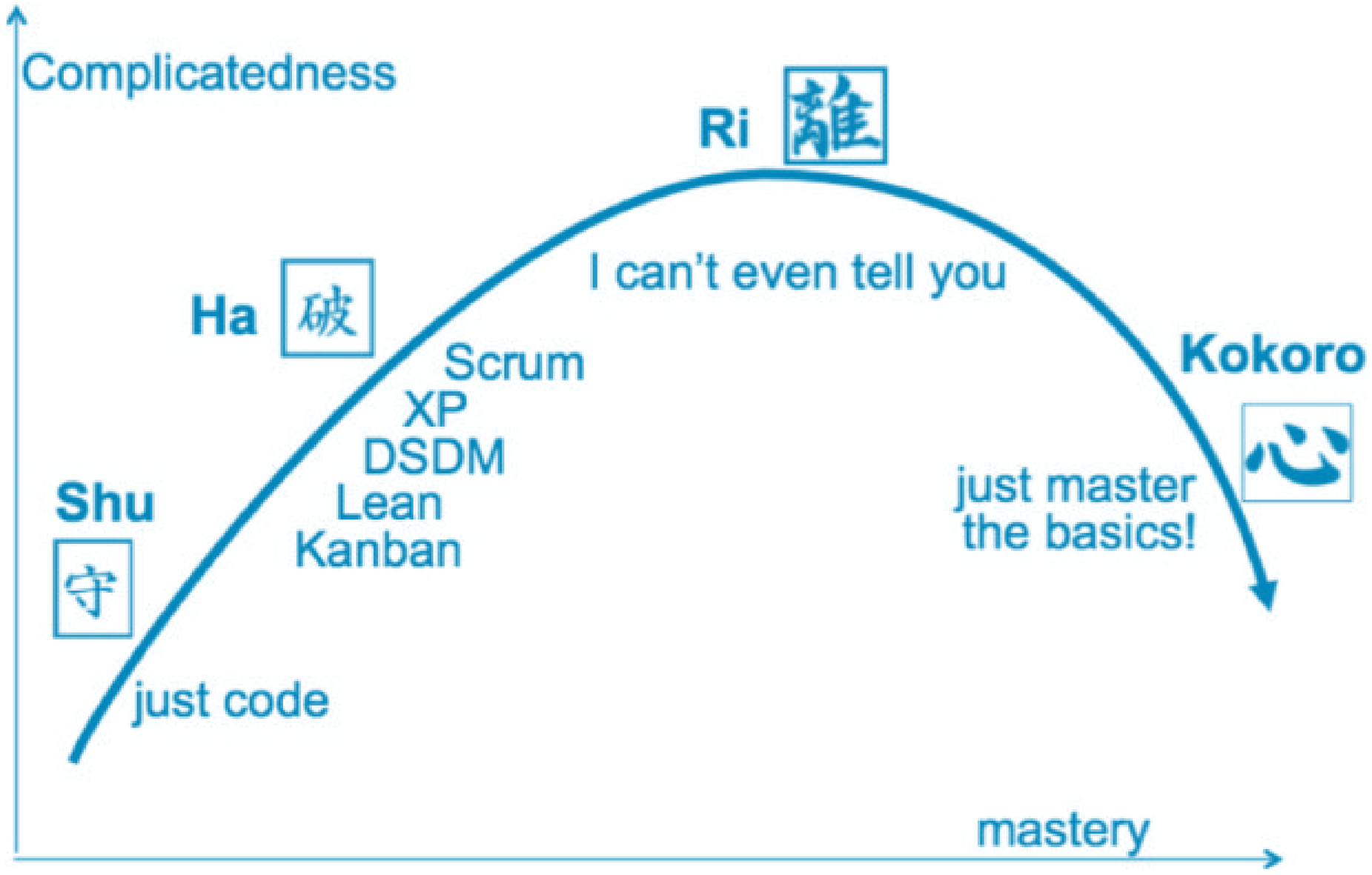




The Heart of Agile

The "*Oath of Non-Allegiance*"

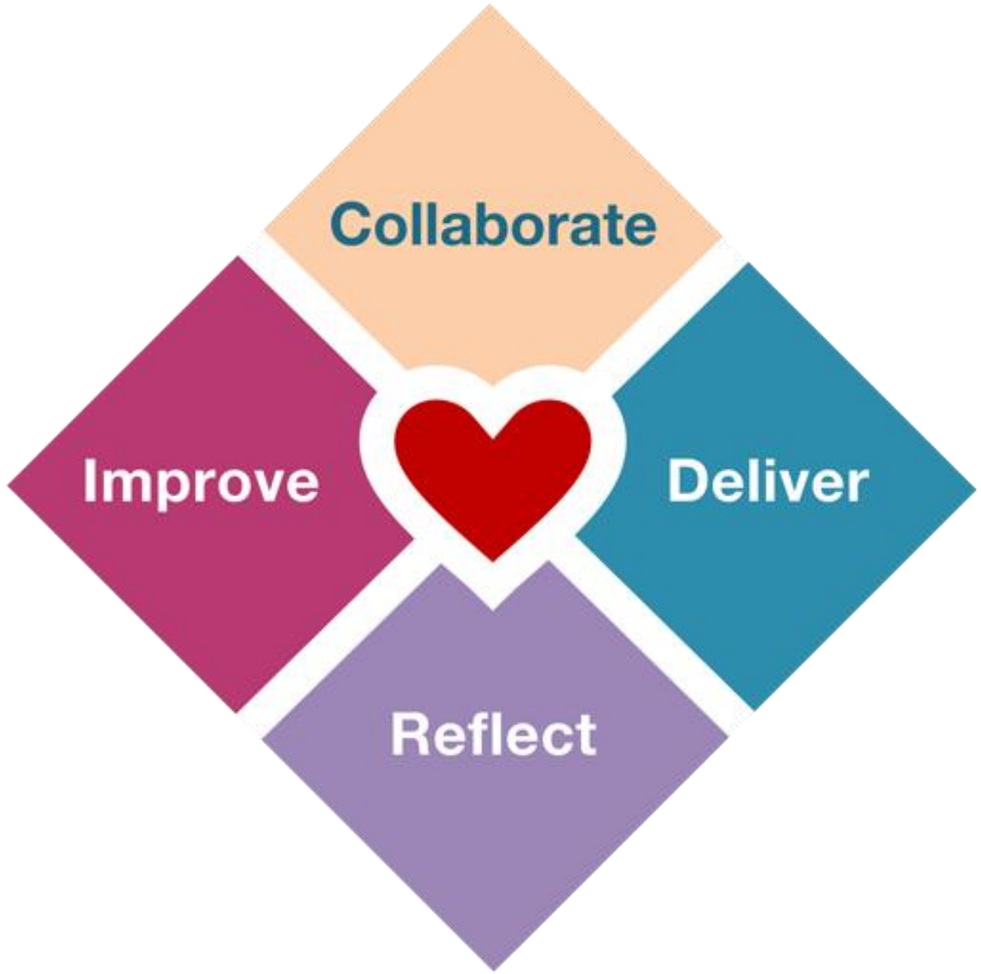
"I promise not to exclude from consideration any idea based on its source, but to consider ideas across schools and heritages in order to find the ones that best suit the current situation."


Complicatedness
Ri
離
I can't even tell you
Ha
破
Scrum
XP
DSDM
Lean
Kanban
Kokoro
心
Shu
守
just master
the basics!
just code
mastery

The Heart of Agile

2015 – Alistair Cockburn

heartofagile.com/

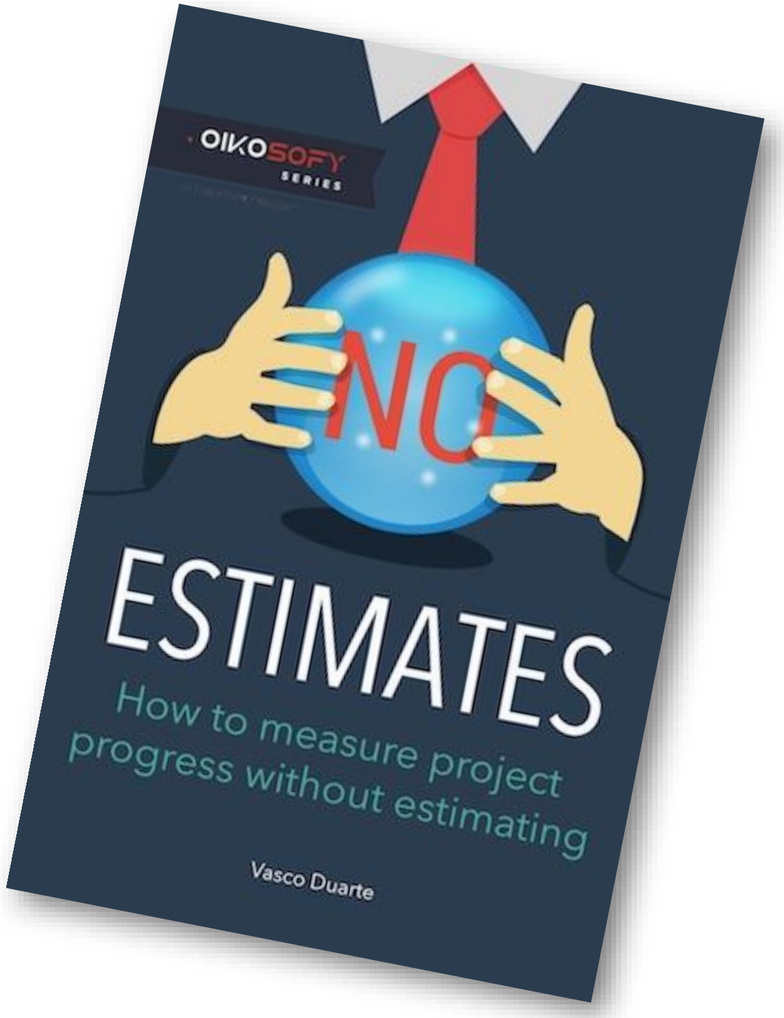


Other (*extreme*) Agile practices

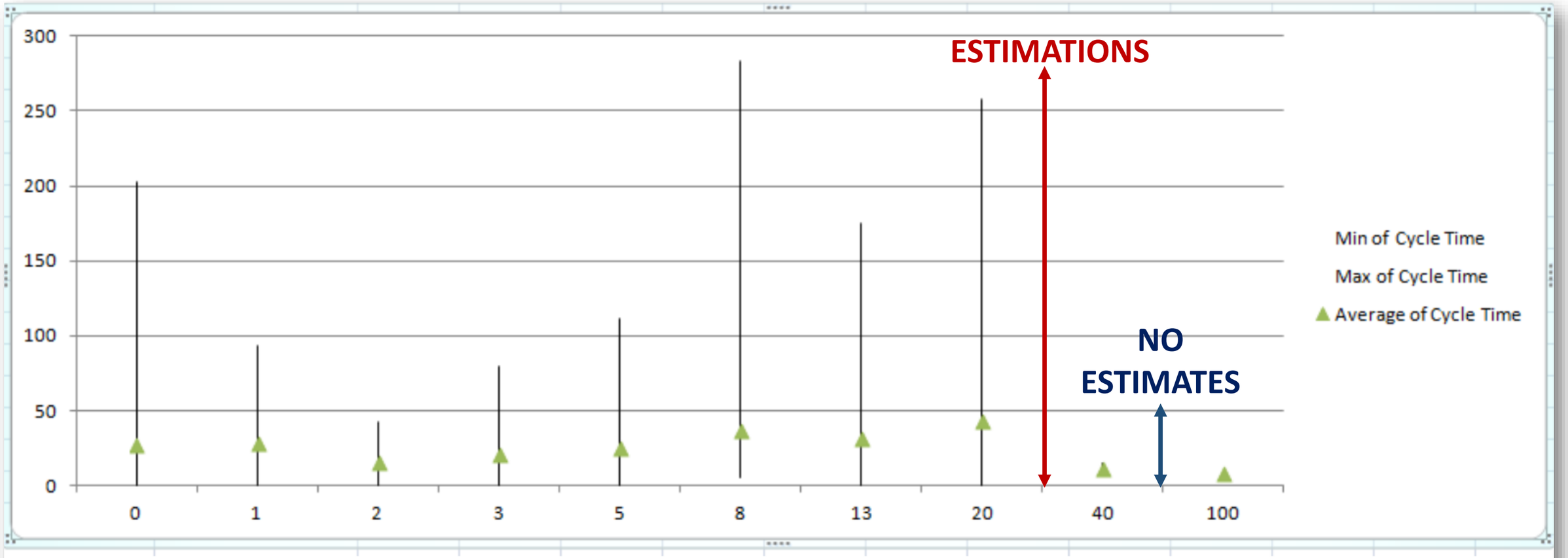
#NoEstimates

Woody Zuill

Vasco Duarte



#NoEstimates



Cory Foy @cory_foy · 27 Jan 2014

A chart I just ran for a team. Bottom numbers are the story point estimates, left numbers are cycle time in *days* pic.twitter.com/hiJQDtVOLK

10 5 9

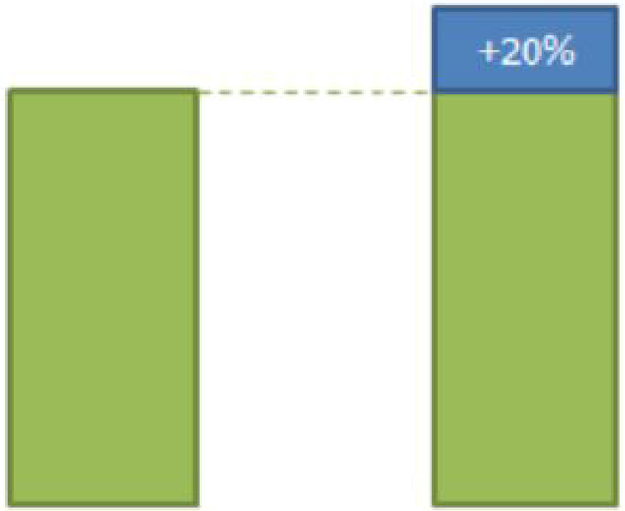
#NoEstimates

After just 3 sprints

Story Points predictive power

The true output: 349,5 SPs completed

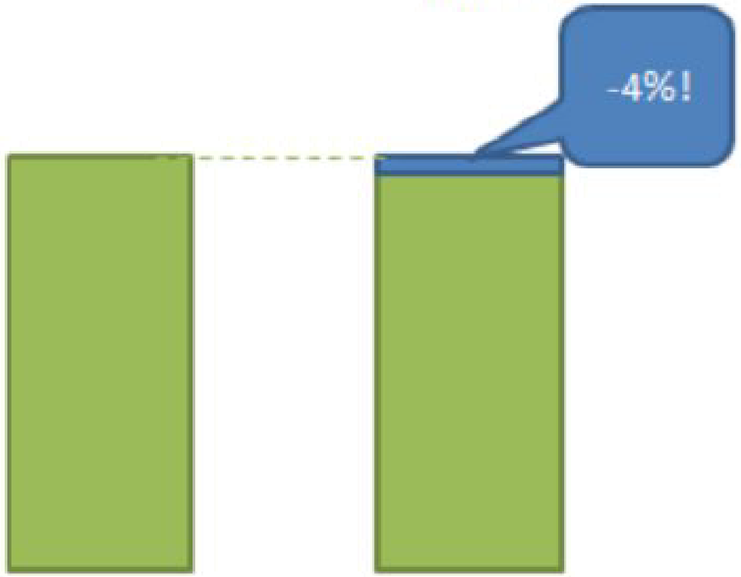
The predicted output: 418 SPs completed



of Stories predictive power

The true output: 228 Stories

The predicted output: 220 Stories



#NoEstimates Whitepaper by Vasco Duarte

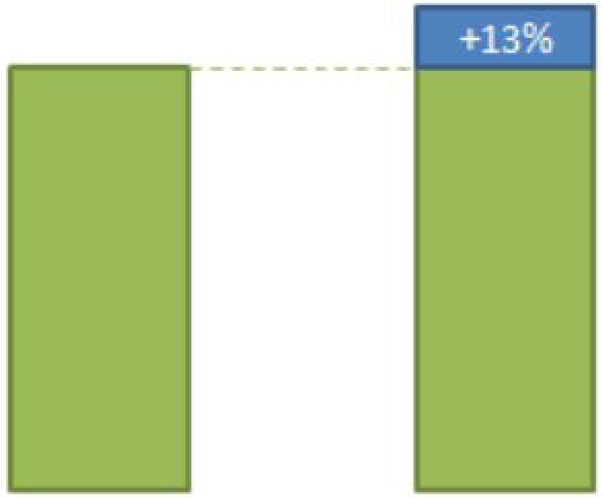
#NoEstimates

After just 5 sprints

Story Points predictive power

The true output: 349,5 SPs completed

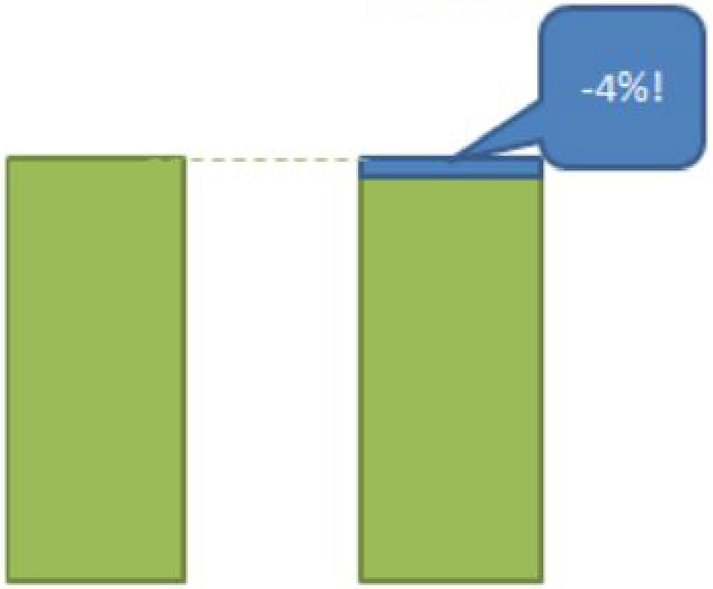
The predicted output: 396 SPs completed



of Stories predictive power

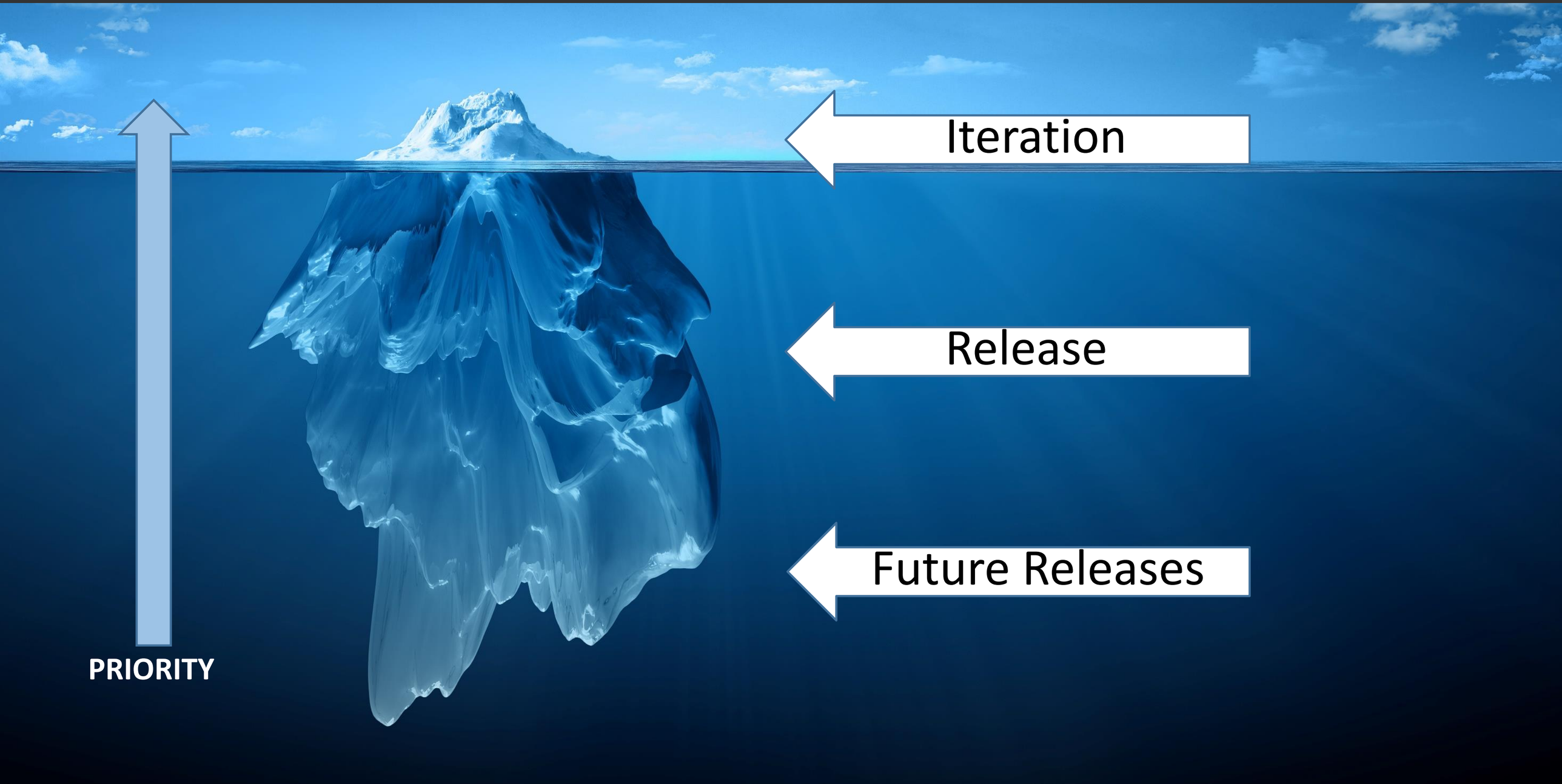
The true output: 228 Stories

The predicted output: 220 Stories



#NoEstimates Whitepaper by Vasco Duarte

#NoBacklog



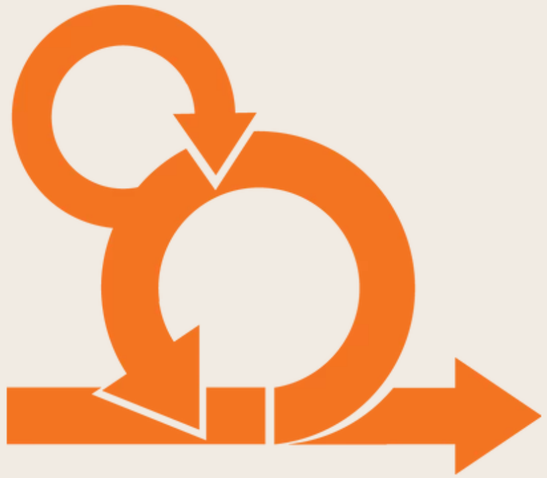
#NoBacklog

"Most backlogs are waste.

Estimating backlog items is therefore super-waste.

Revisiting backlog estimates are in
mentally-deranged territory"

Paul Klipp



AGILE SOFTWARE DEVELOPMENT

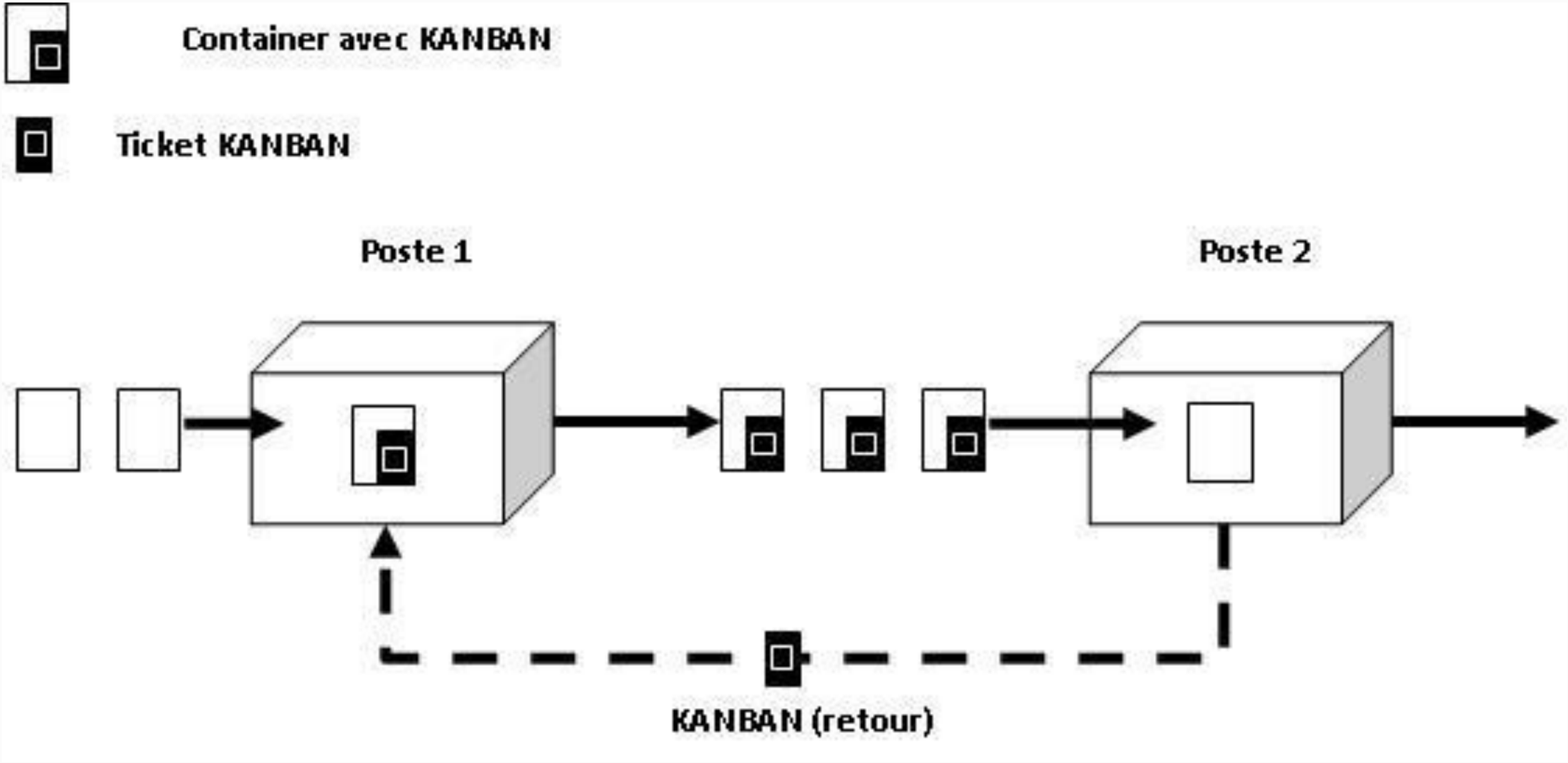
Kanban

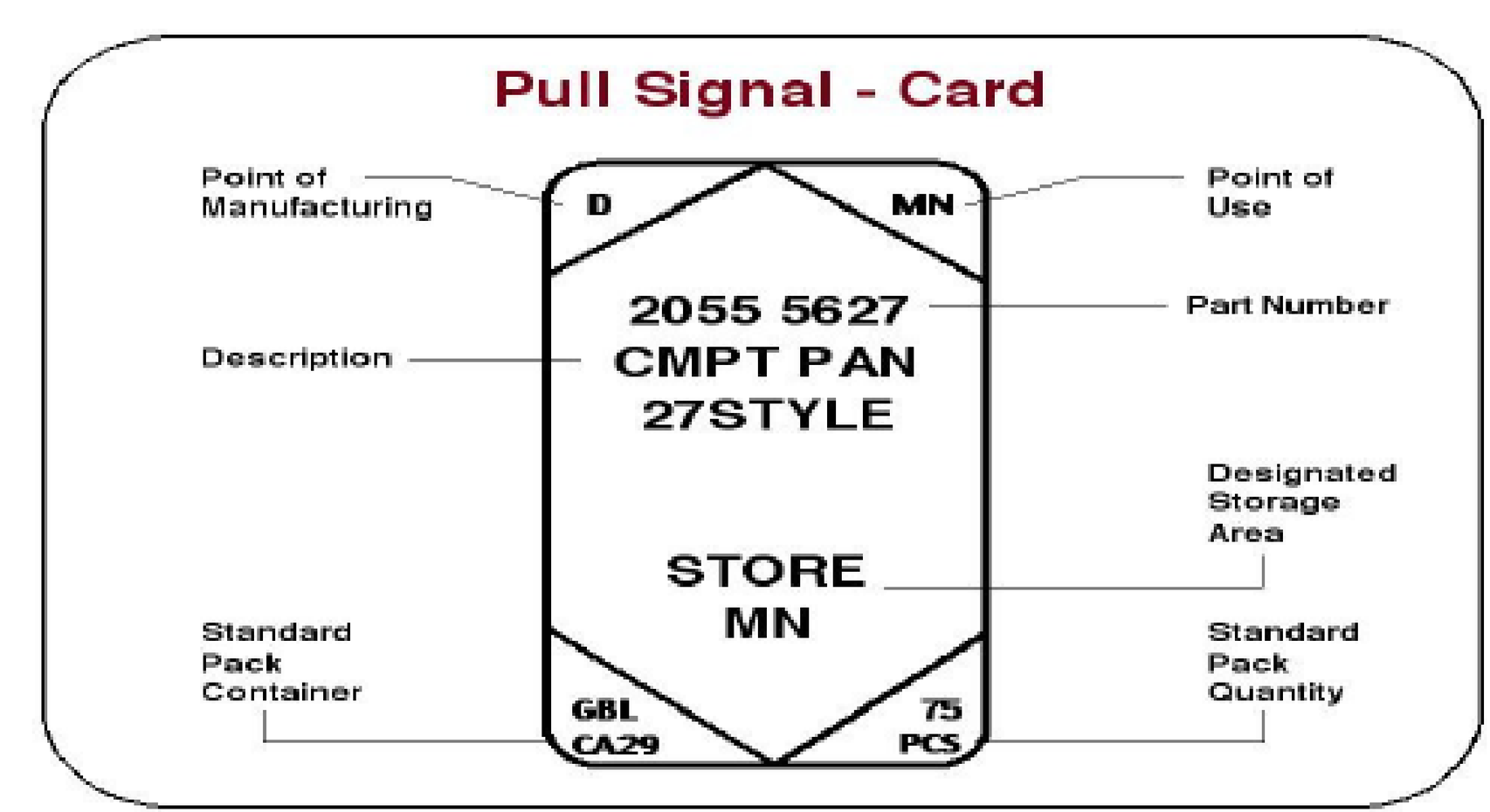
part 1

Kanban

=

“signal card”

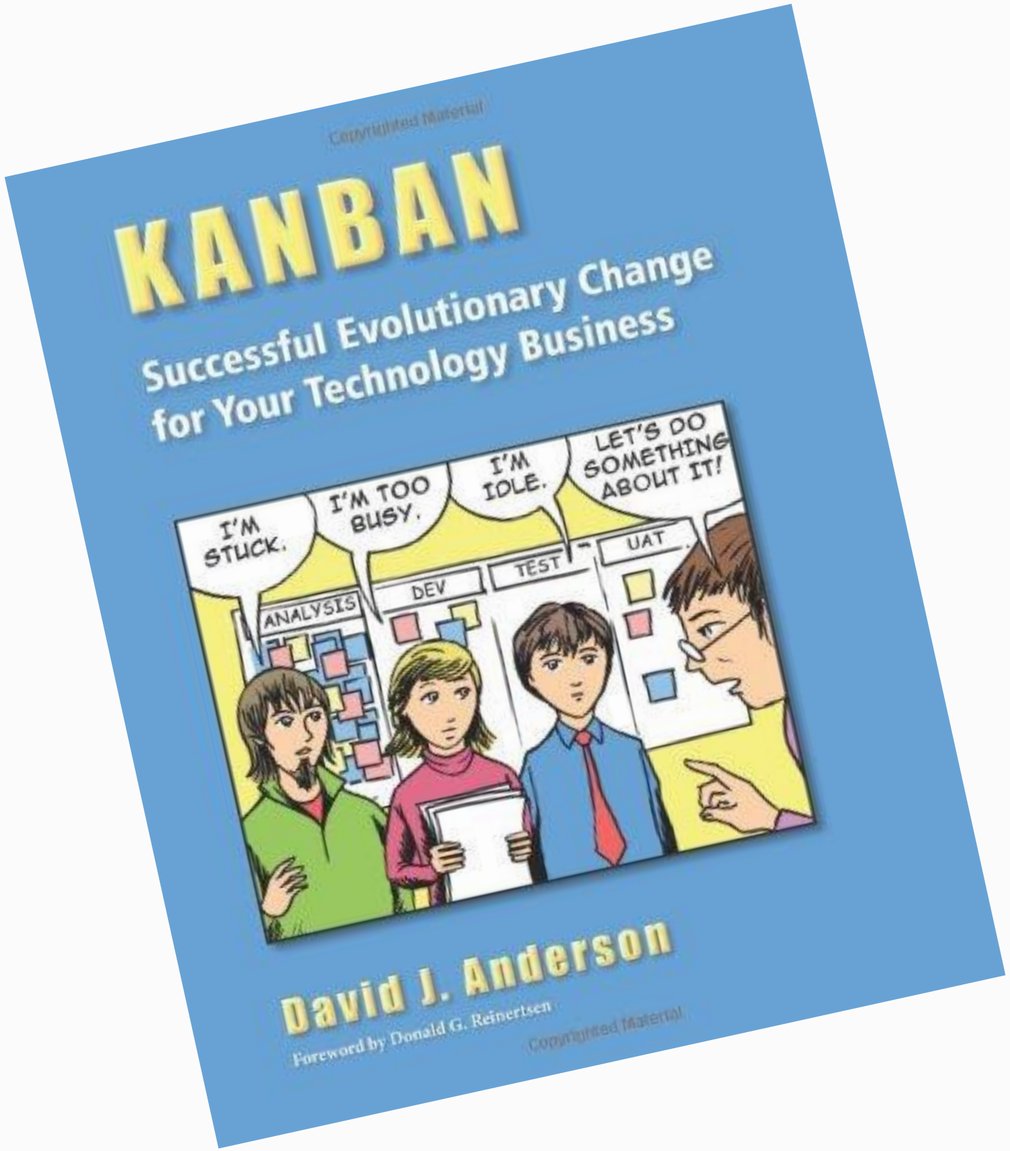

Container avec KANBAN
Ticket KANBAN
Poste 1
Poste 2
KANBAN (retour)


Pull Signal - Card
Point of Manufacturing
D
MN
Point of Use
2055 5627
Part Number
Description
CMPT PAN
27STYLE
Designated Storage Area
STORE
MN
Standard Pack Container
GBL
CA29
75
PCS
Standard Pack Quantity

Limits
work in progress
(WIP)


PART NO.
K5487
PART NO.
PH467
PART NO.
R64
PART NO.
C597
DO
DONE
DO
DONE
DO
DONE
DO
DONE

- Kanban systems ⊂ Pull systems
  - Systematic way to achieve a sustainable pace of work
  - An approach to introducing process changes that would meet with minimal resistance
- Kanban requires that process policies are defined explicitly
- First virtual Kanban system for software engineering: 2004, Microsoft

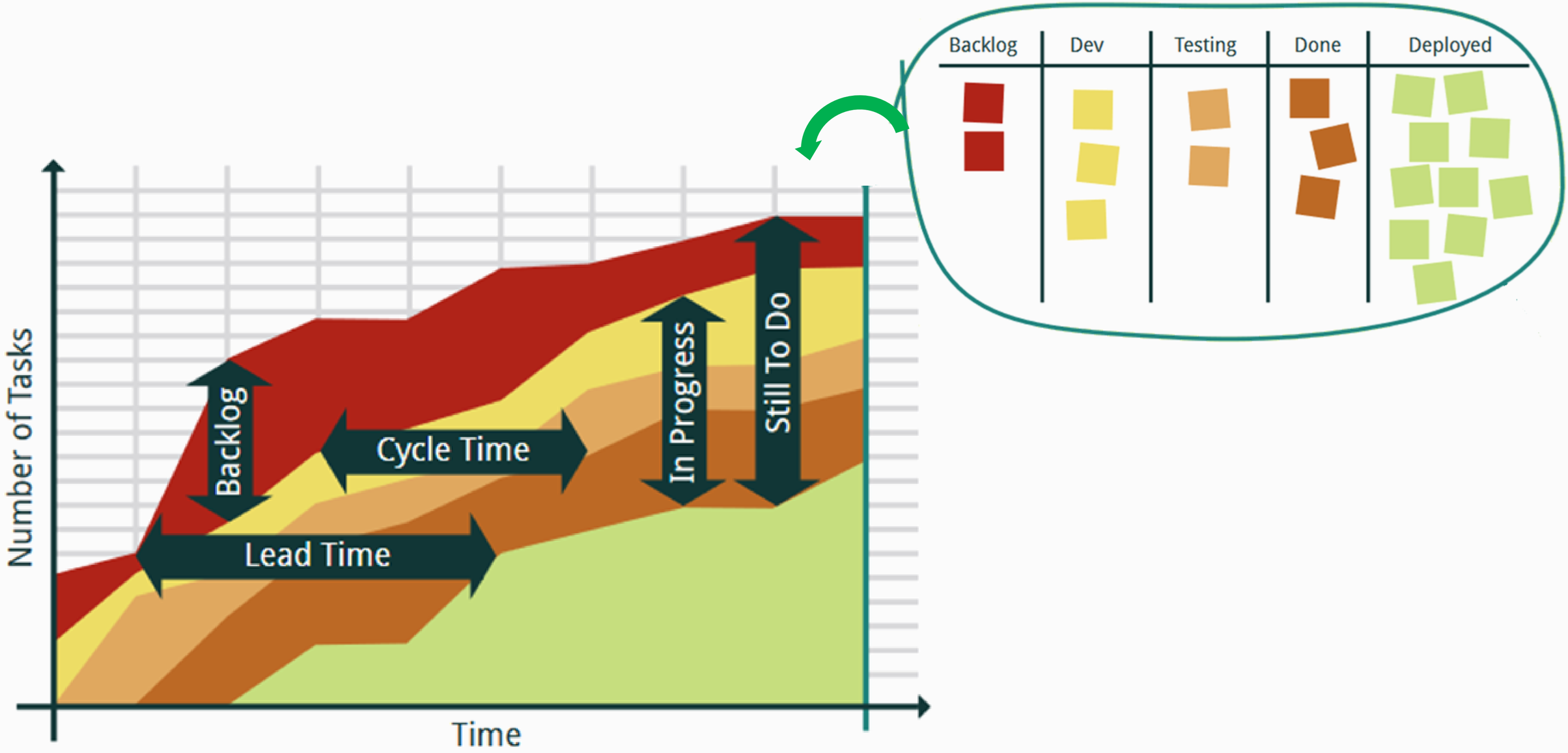


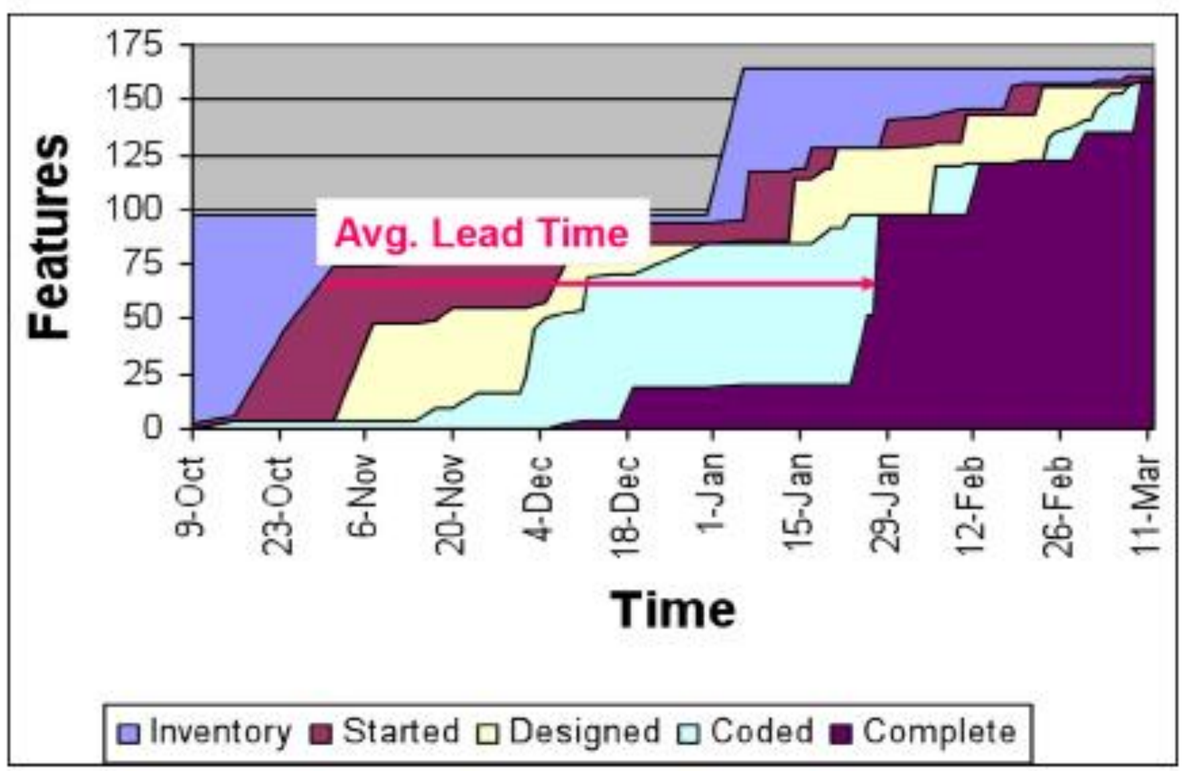
Recipe for success

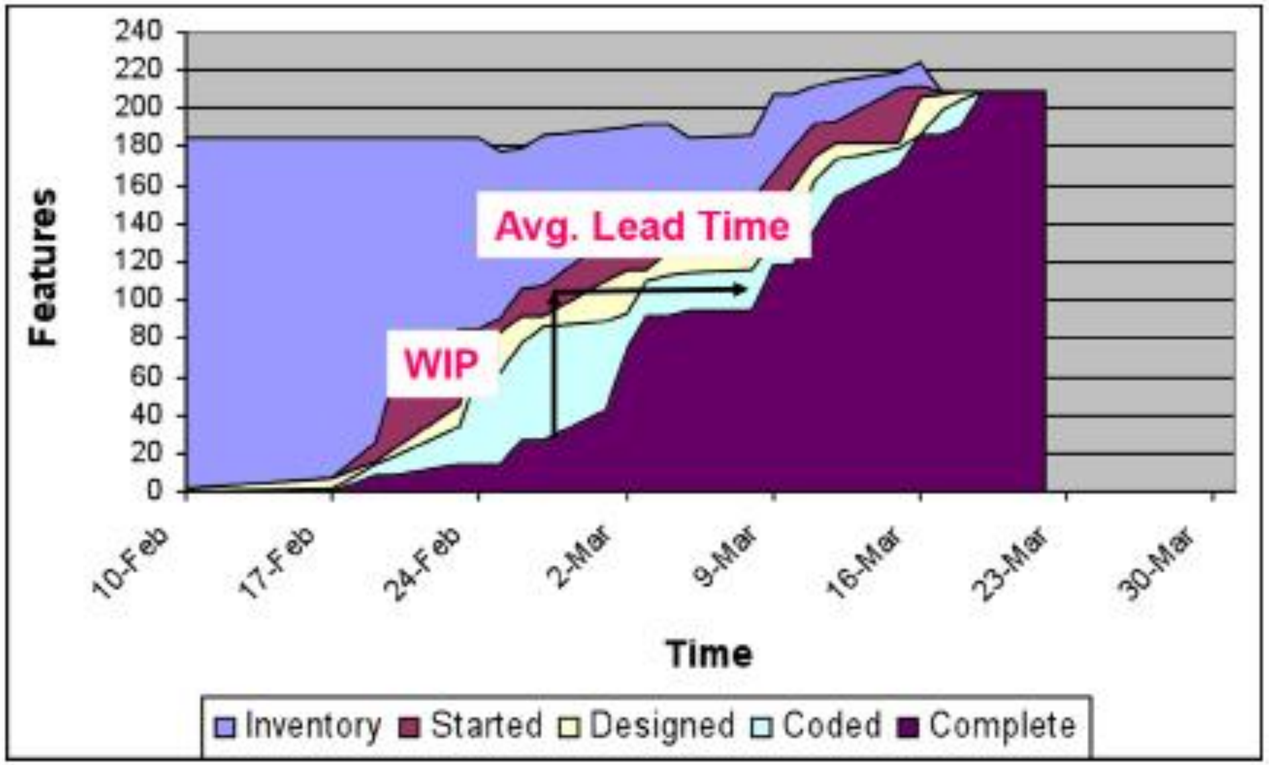
- Focus on Quality
- Reduce WIP
- Deliver Often
- Prioritize
- Attack sources of variability to improve predictability

Kanban delivers all of them!

Cumulative Flow Diagram







1. Longer lead times seem to be associated with significantly poorer quality!
2. Great amounts of WIP -> Longer lead times

Conclusion

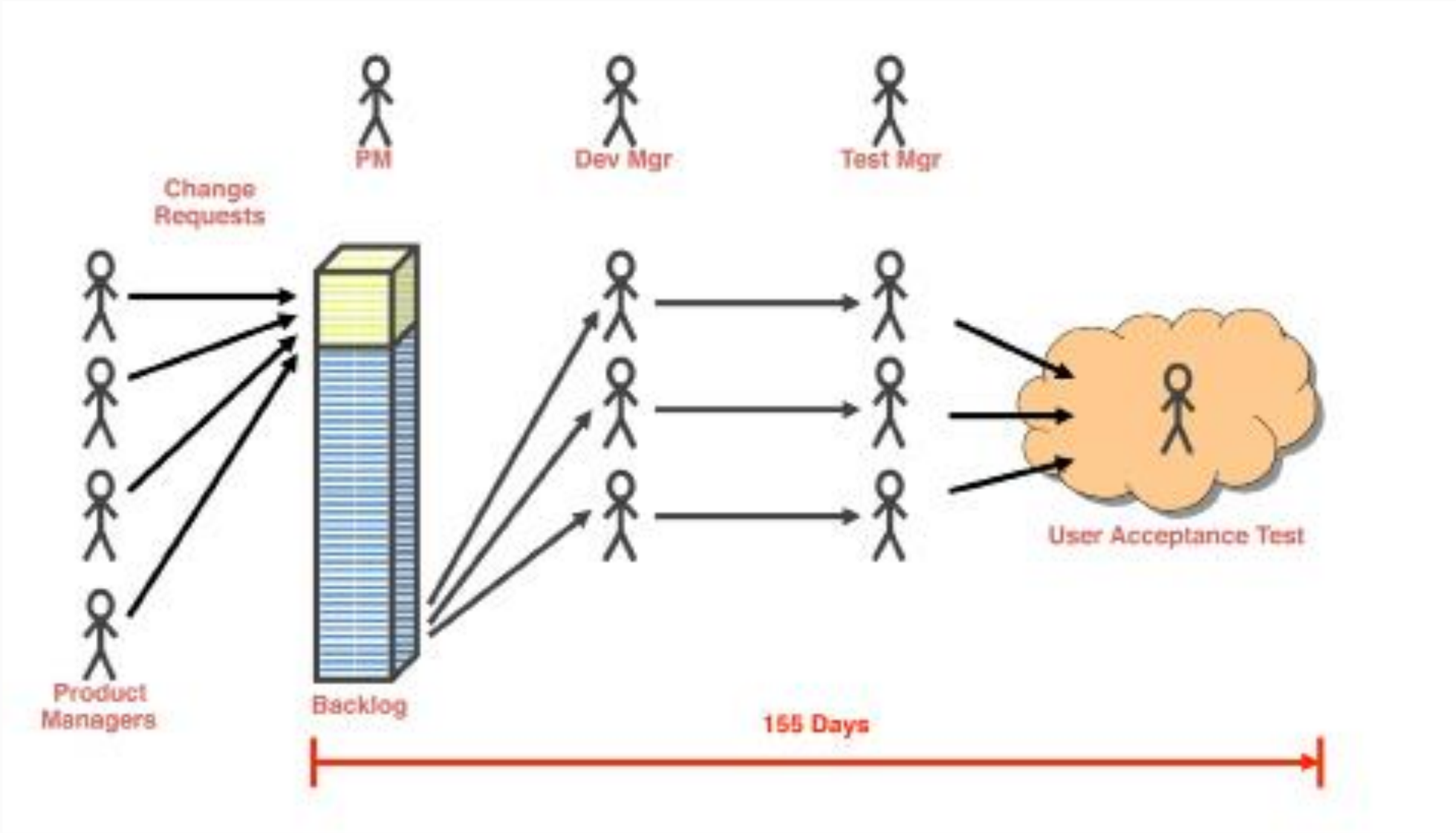
- Reducing work-in-progress, or shortening the length of an iteration, will have a significant impact on initial quality.

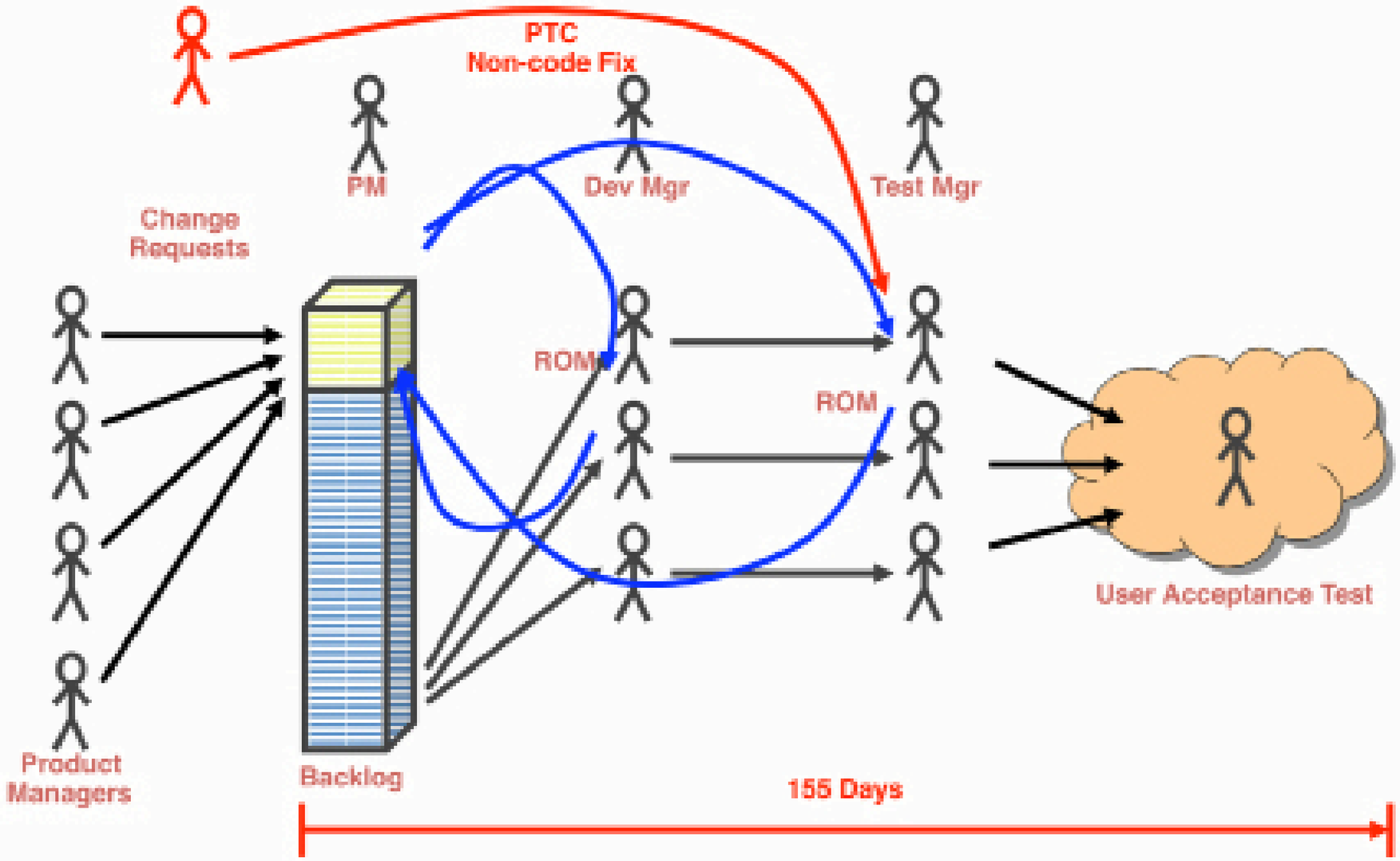
Also…

- Frequent releases build trust

- The throughput of a process is constrained by a bottleneck.
- It’s unlikely we know where that bottleneck is. (all claim to be completely overloaded)
- When limiting the work-in-progress within => only the bottleneck resources will remain fully loaded.
- The other workers in the value stream will find they have slack capacity.

2004 - developed upgrades & fixed production bugs
for about 80 cross-functional IT applications used by Microsoft

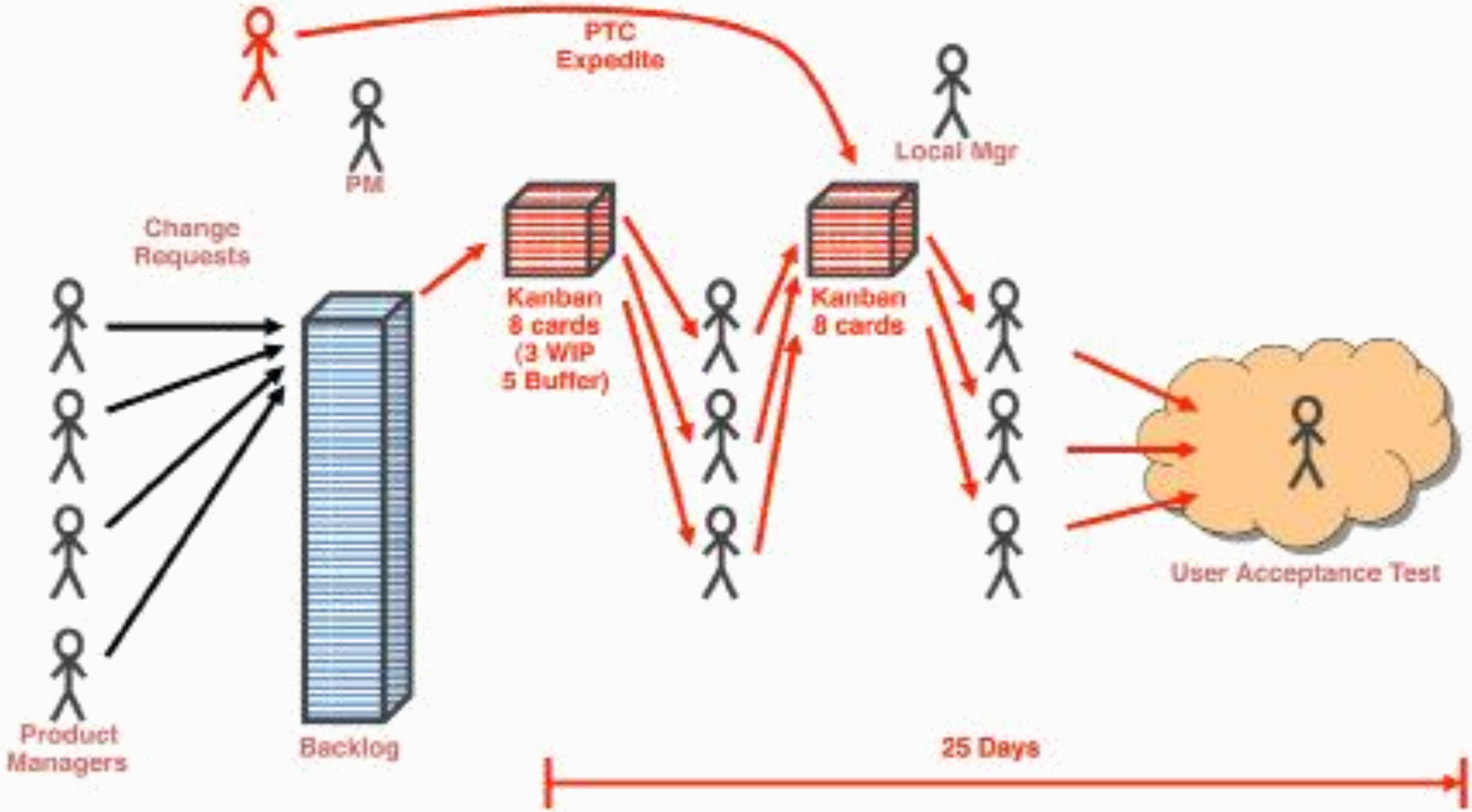

PM
Dev Mgr
Test Mgr
Change Requests
Product Managers
Backlog
User Acceptance Test
155 Days

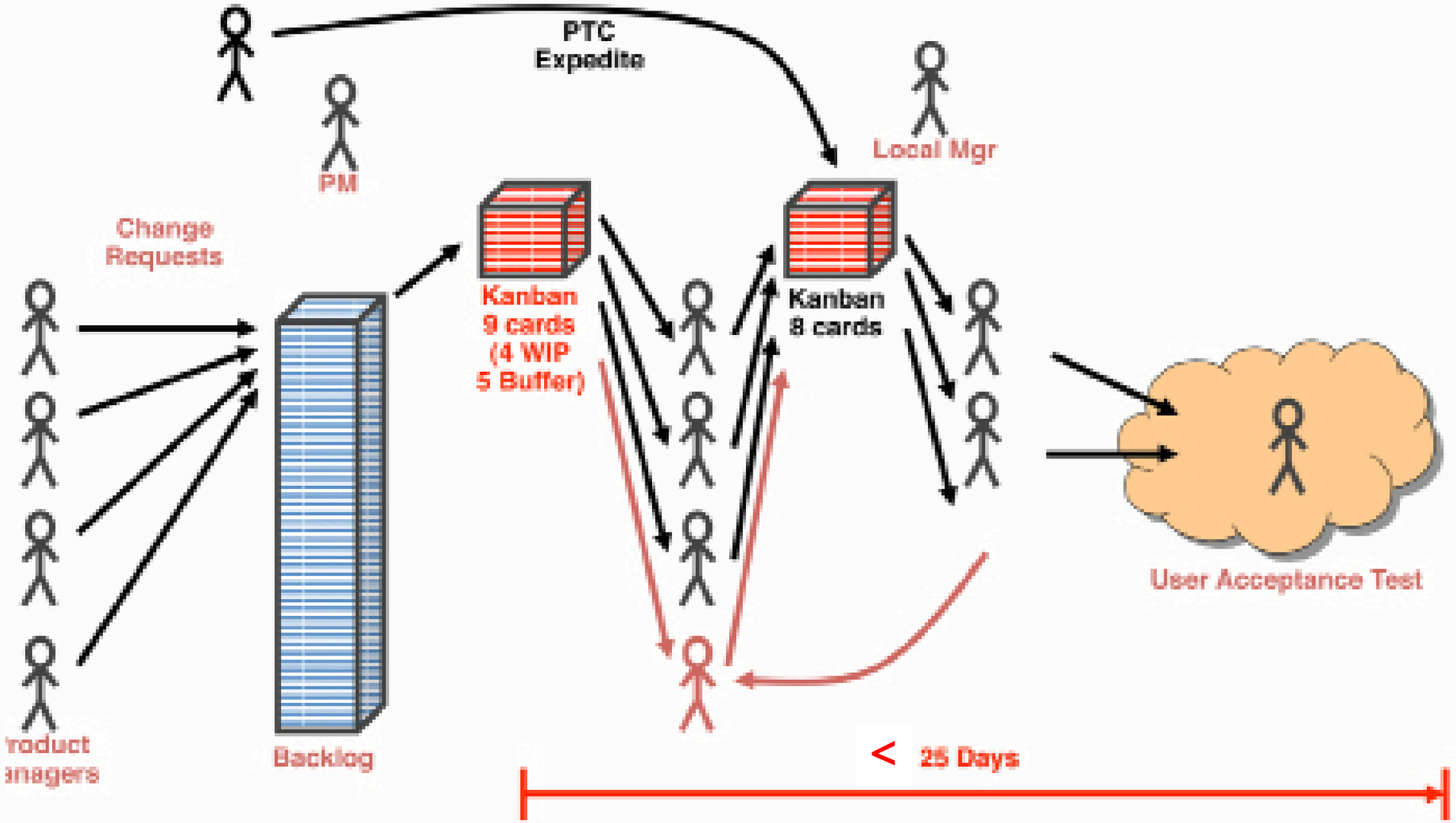

PTC
Non-code Fix
PM
Dev Mgr
Test Mgr
Change
Requests
ROM
ROM
Product
Managers
Backlog
155 Days
User Acceptance Test

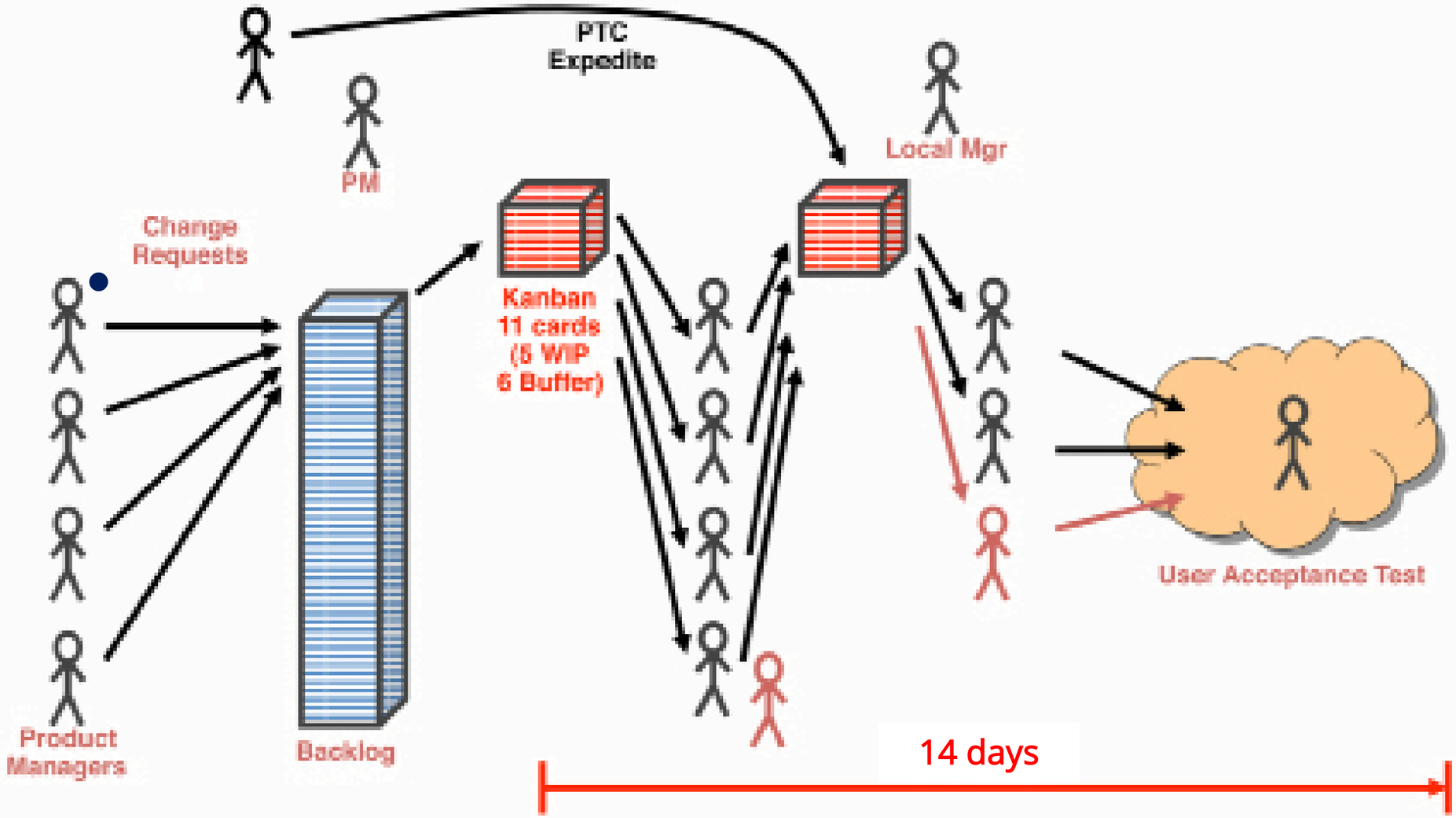
An average request took 11 days of engineering!!!

⇒ More than 90 percent of the lead time was queuing, or other forms of waste.

⇒ The estimation effort was consuming 33-40% of capacity


PTC
Expedite
PM
Local Mgr
Change
Requests
Kanban
8 cards
(3 WIP
5 Buffer)
Kanban
8 cards
User Acceptance Test
Product
Managers
Backlog
25 Days


PTC
Expedite
PM
Local Mgr
Change
Requests
Kanban
9 cards
(4 WIP
5 Buffer)
Kanban
8 cards
User Acceptance Test
roduct
anagers
Backlog
< 25 Days



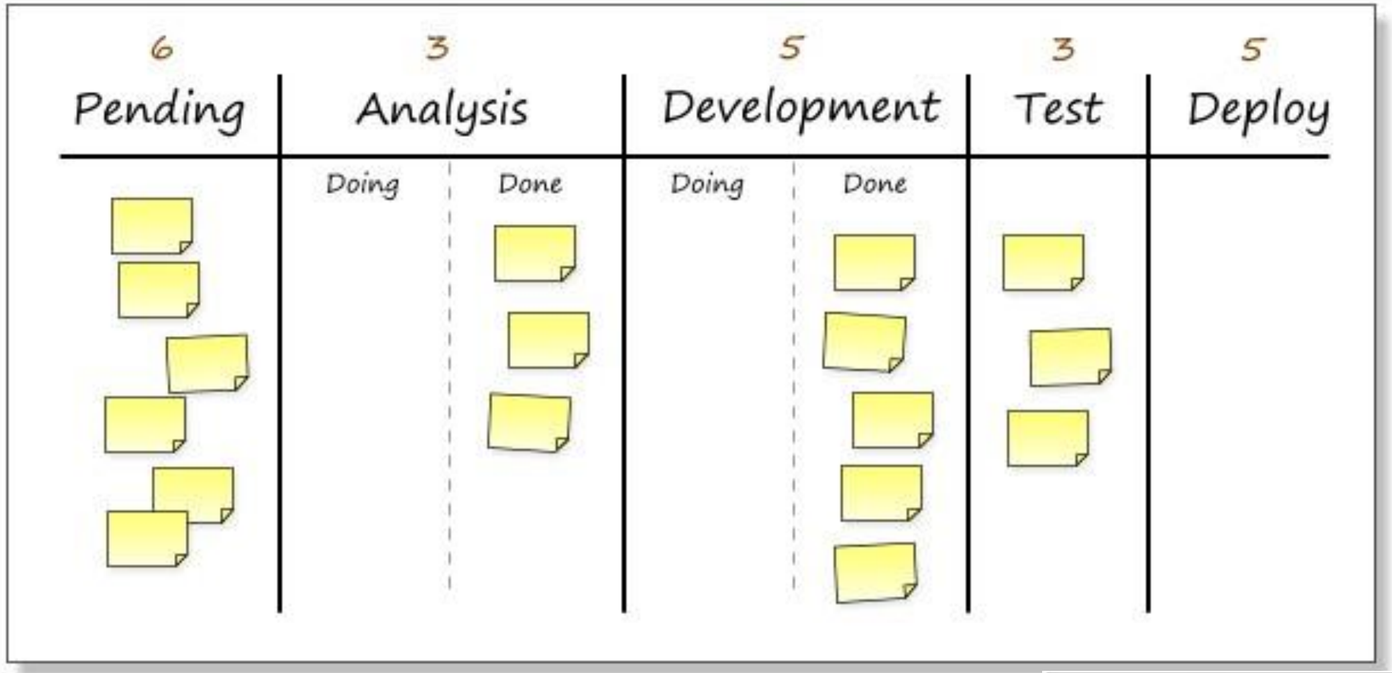
The backlog was eliminated entirely on November 22, 2005!

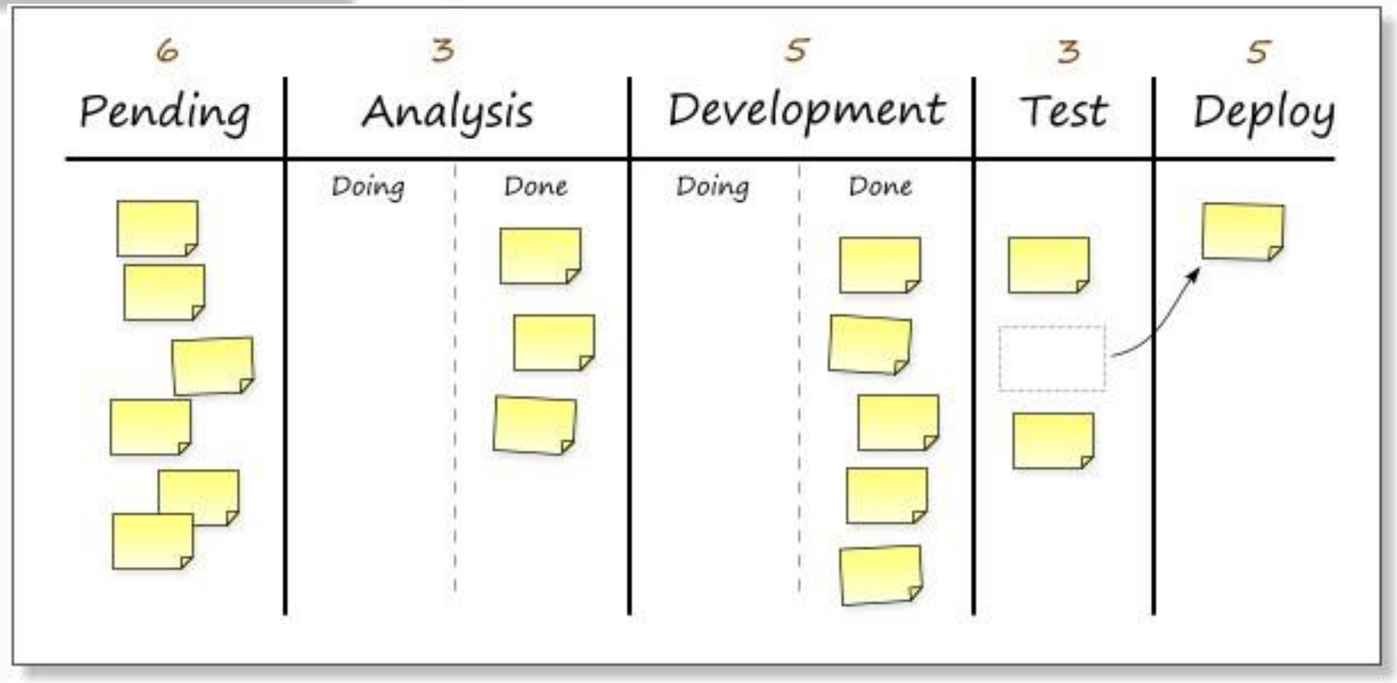
Conclusions after implementing first Kanban System

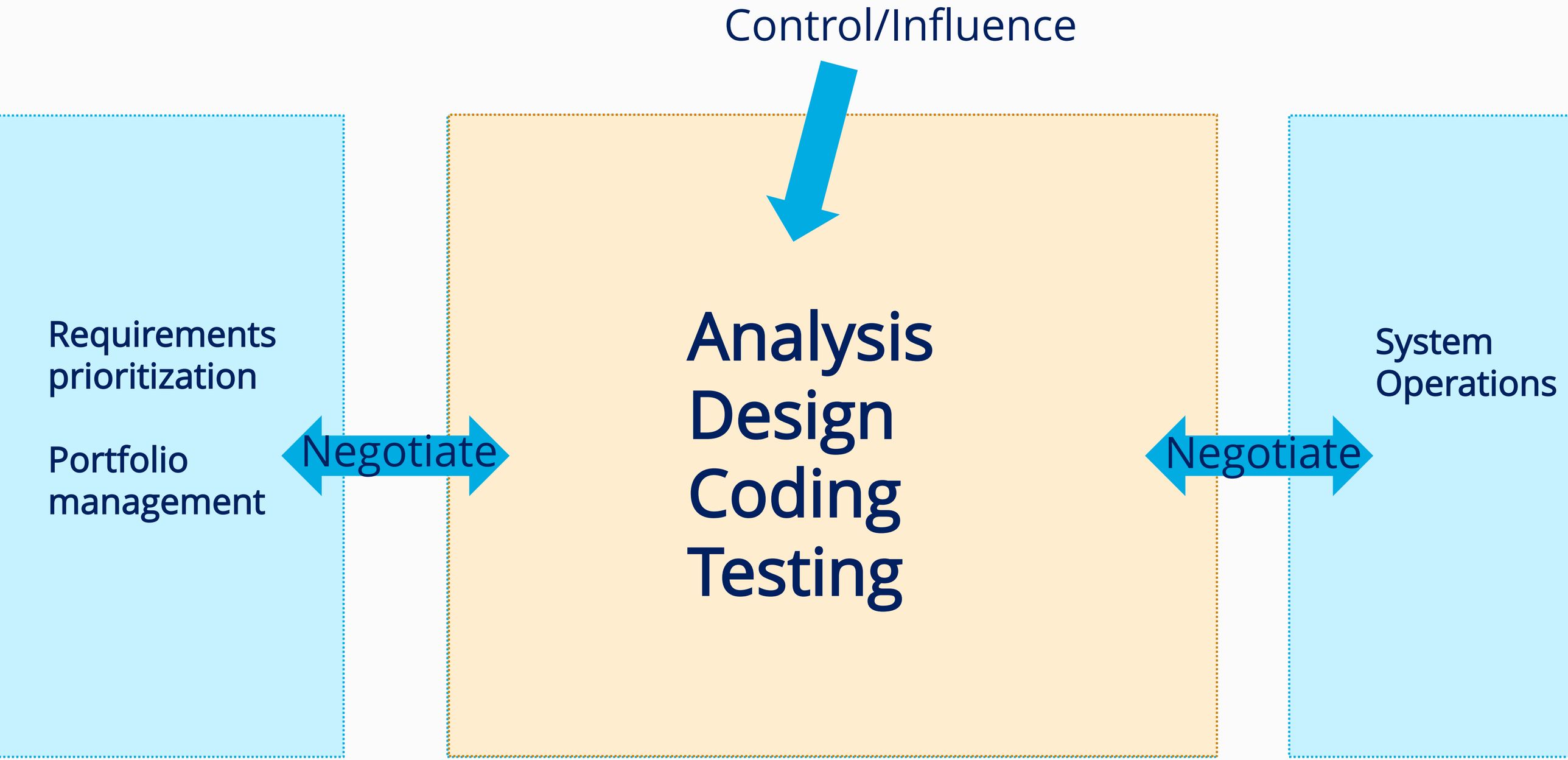
Kanban:

- enables incremental changes
- enables change with reduced political risk
- enables change with minimal resistance
- will reveal opportunities for improvement that do not involve complex changes to engineering methods

Changes can take time to take full effect!


6
Pending
3
Analysis
Doing
Done
5
Development
Doing
Done
3
Test
5
Deploy


6
Pending
3
Analysis
Doing
Done
5
Development
Doing
Done
3
Test
5
Deploy

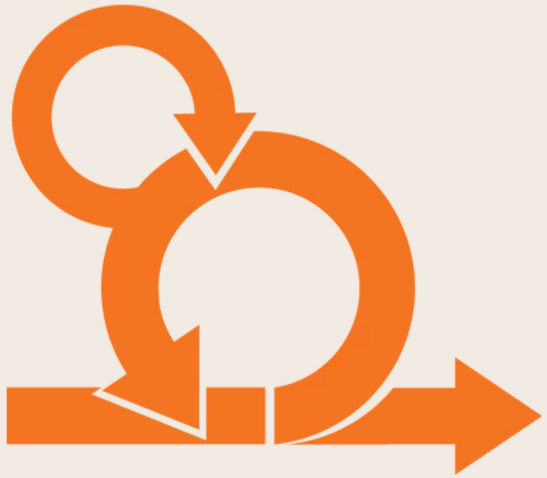

Control/Influence
Requirements prioritization
Portfolio management
Negotiate
Analysis
Design
Coding
Testing
Negotiate
System Operations

Kanban

is an approach that drives change

by optimizing

your existing process.



AGILE SOFTWARE DEVELOPMENT

Performance Management in Agile Teams

Project performance

Favorable conditions

Interesting project
Involved customer
Mature team

Unfavorable conditions

Unappealing project
Disengaged customer
Junior team
New technology
High-risk domain

CHALLENGES

Because of ….condition (cause)……
It will/might happen that ….trigger……
Leading to ….effect……

Types of challenges

Risks

Probability of condition < 100%

Probability of trigger = 100%

Strategy: mitigation, contingency, transfer

Issues

Probability of condition = 100%

Probability of trigger = 100%

Strategy: solve

Assumptions

Probability of condition < 100%

Probability of trigger < 100%

Strategy: constant checking

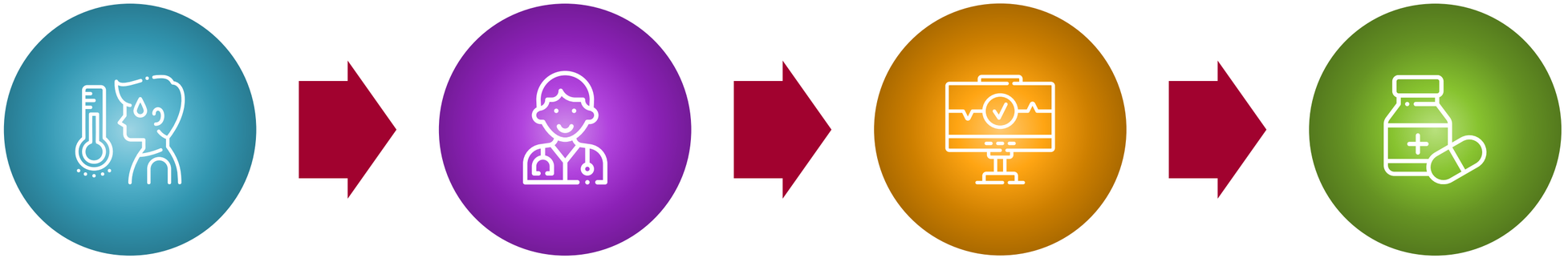
Constraints

Probability of condition = 100%

Probability of trigger < 100%

Strategy: adapt

Approach



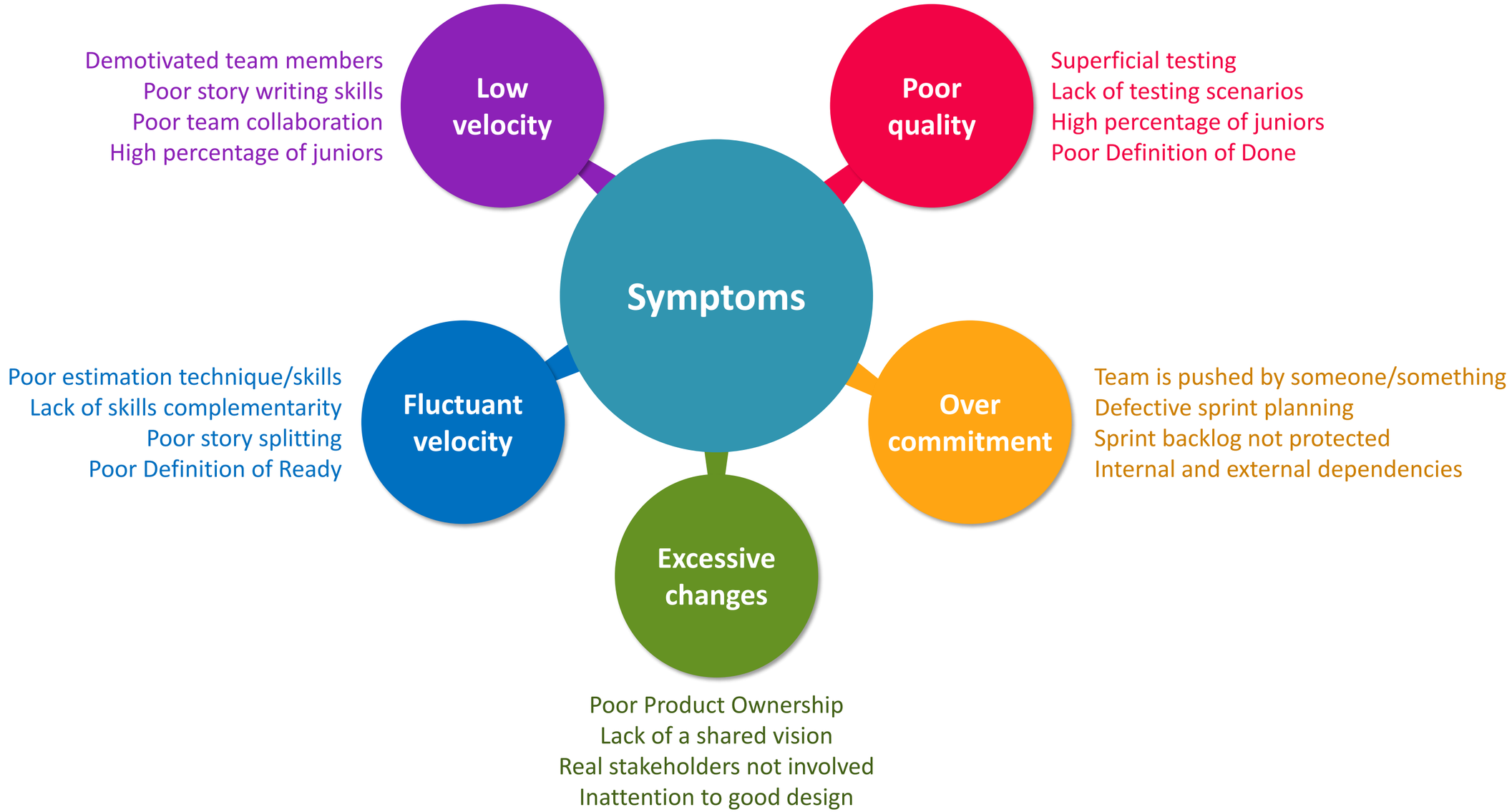
SYMPTOMS
How it manifests , what are the main perceivable effects

CAUSES
What are the most probable root causes for the symptoms

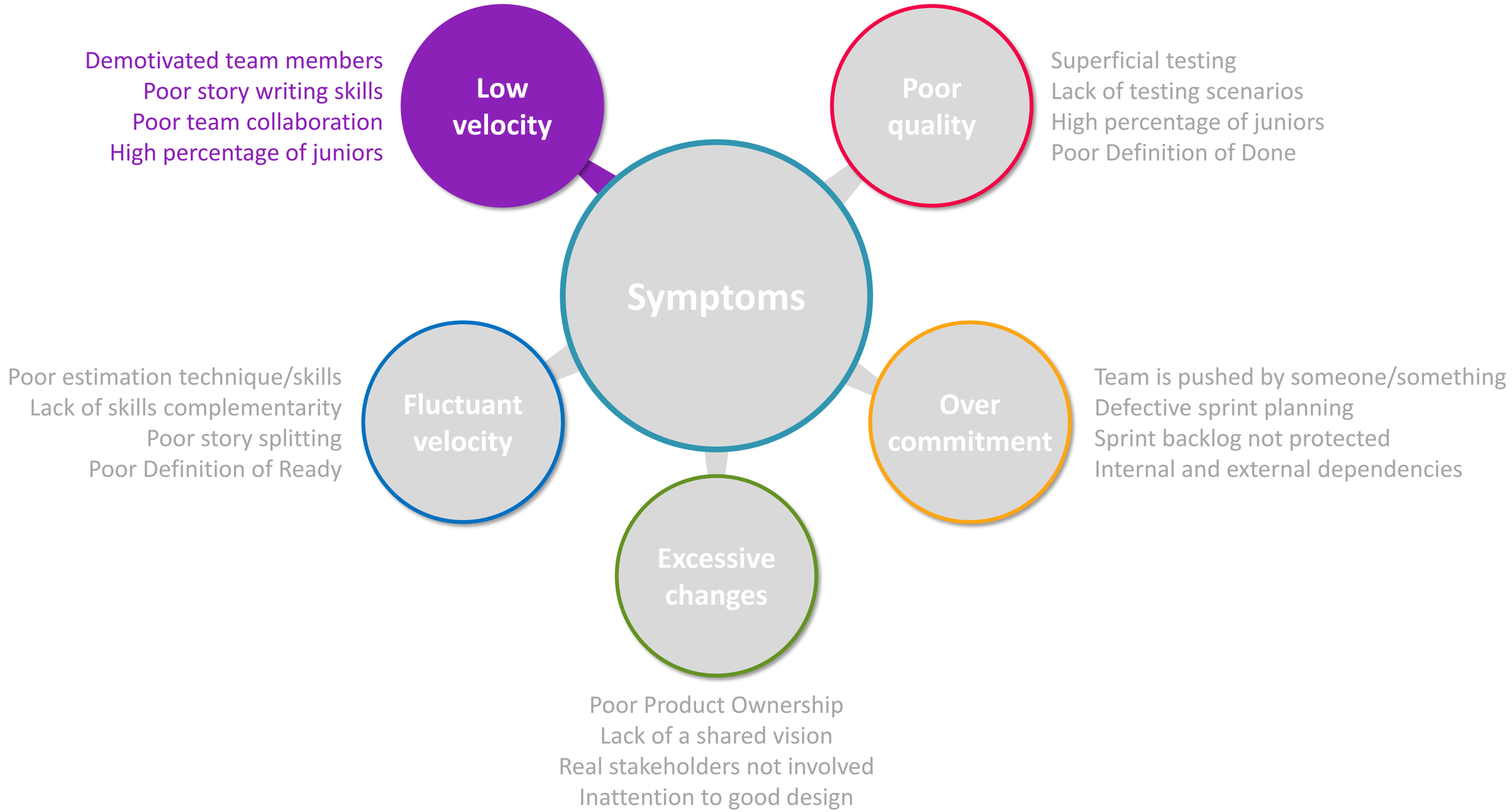
DIAGNOSTIC
How we may diagnose the nature and severity of the challenge

SOLUTIONS
What can be done to address the challenge or remove the cause

Most frequent symptoms



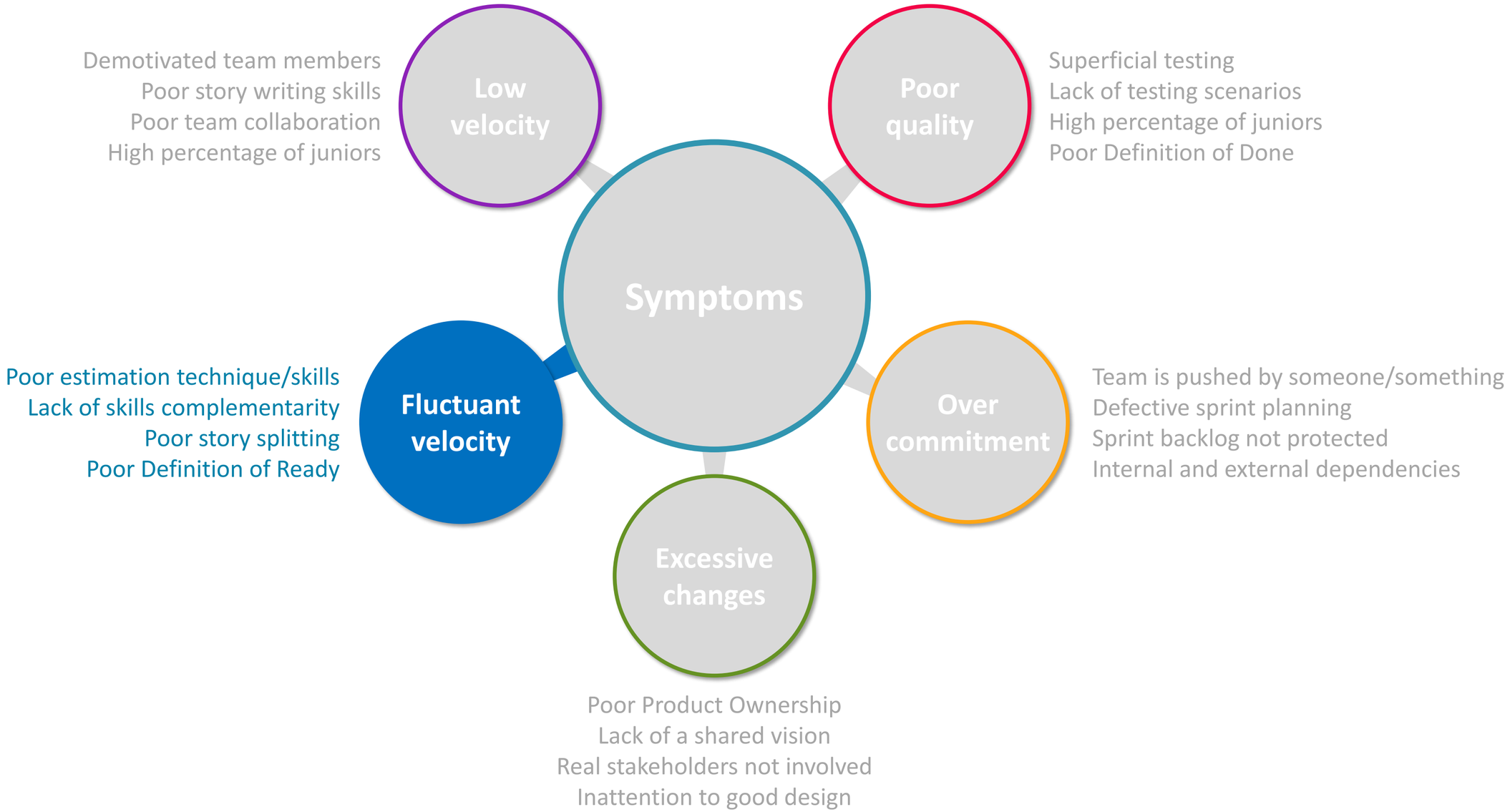
Most frequent symptoms



Low velocity (compared to project complexity)

	Demotivated team members	Poor story writing skills	Poor team collaboration	High percentage of juniors
CLASSIFYING	Sometimes an issue or risk, sometimes a constraint	Always an issue	Always an issue	Usually a constraint, sometimes an issue
DIAGNOSING	Face-to-face talking Direct observation	Check INVEST rules Check acceptance criteria	Apply Gemba (mingle) Attend daily standups	Examine team CVs Direct observation
SOLVING	Seek for deeper cause Align project & team goals	Story writing meetings Use business analysis skills	Apply value stream mapping Maintain a stable team core	Get external mentoring support Replace some team members
ADAPTING	Manage stakeholders expectations	Don’t adapt, solve it!	Don’t adapt, solve it!	Manage stakeholders expectations

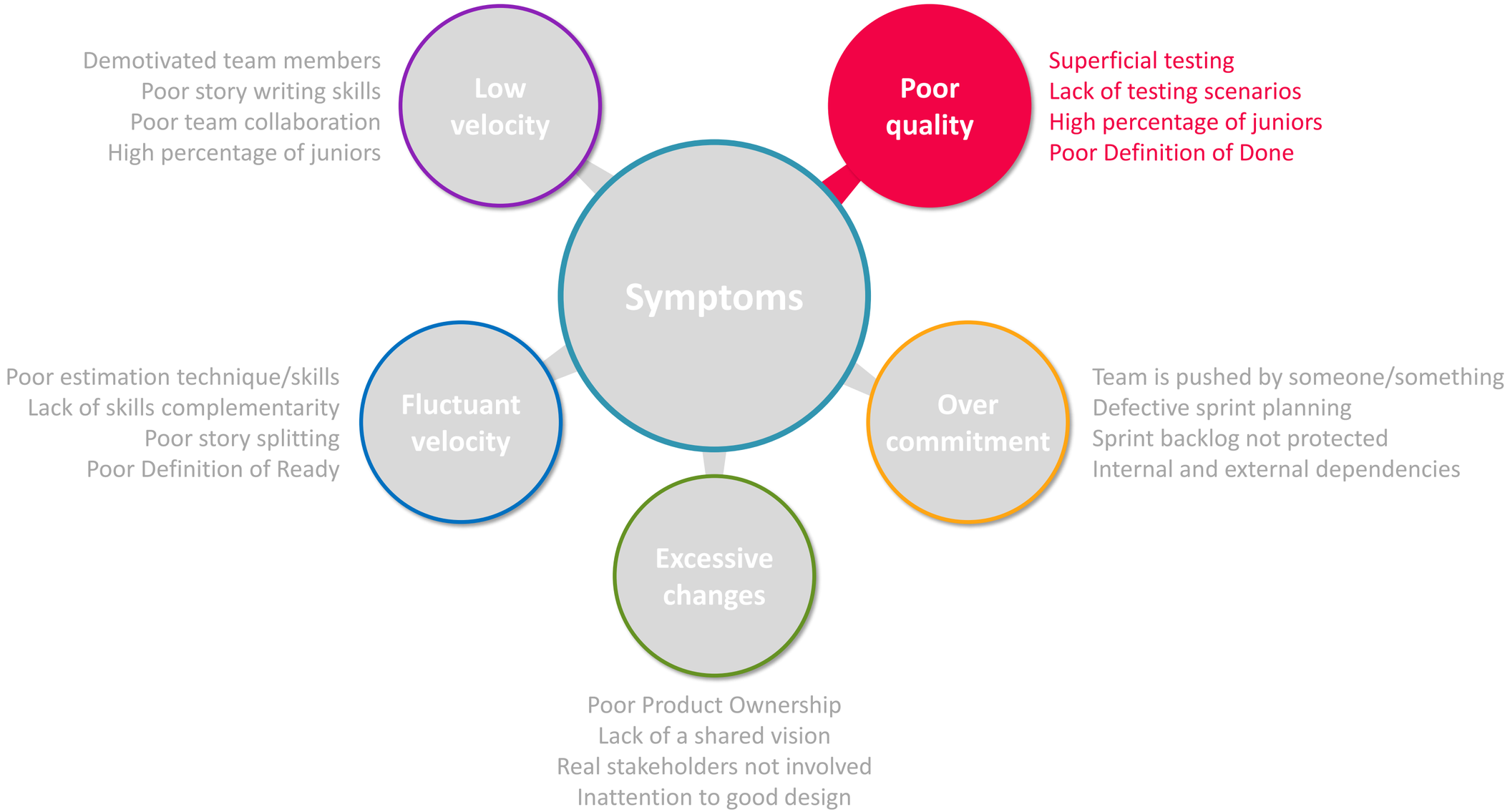
Most frequent symptoms



Fluctuant velocity

	Poor estimation technique / skills	Lack of skill complementarity	Poor story splitting	Unclear Definition of Ready
CLASSIFYING	Always an issue	Sometimes an issue or risk, sometimes a constraint	Always an issue	Always an issue
DIAGNOSING	Analyze effort / SP Test previous estimations	Analyze effort / team member Look for bottlenecks	Monitor unfinished stories	Monitor sprint plannings Ask team which is the DoR
SOLVING	Review current SP system Move to a different technique	Pair working Knowledge sharing strategy	Apply splitting techniques Adopt a SP threshold	Run a clarification session Review periodically DoR
ADAPTING	Don’t adapt, solve it!	Match stories to skills	Don’t adapt, solve it!	Don’t adapt, solve it!

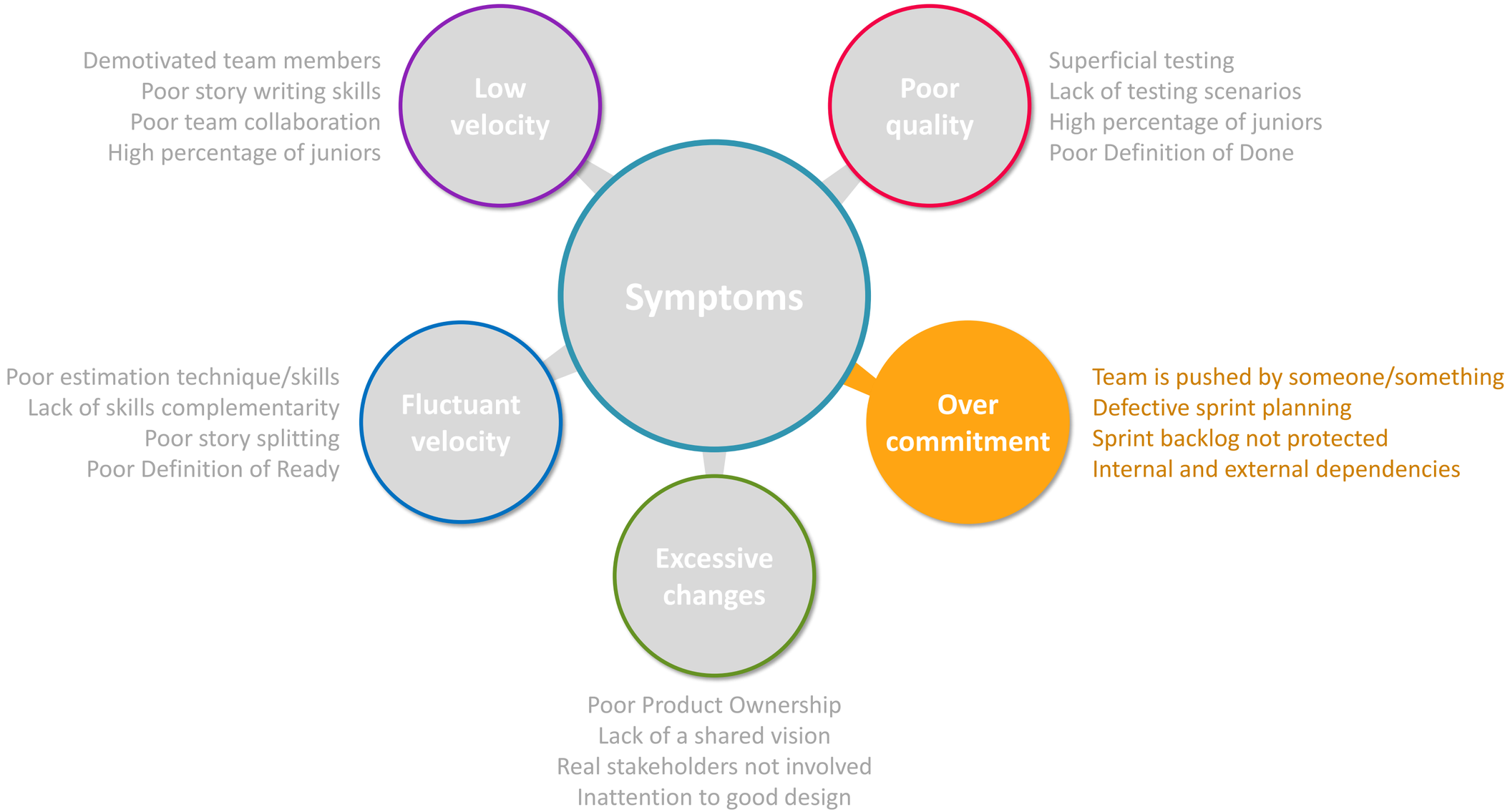
Most frequent symptoms



Poor quality of deliverables

	Superficial testing	Lack of testing scenarios	High percentage of juniors	Poor Definition of Done
CLASSIFYING	Always an issue	Always an issue	Usually a constraint, sometimes an issue	Always an issue
DIAGNOSING	Analyze QA effort / SP Monitor escaped defects	Inspect testing practice Examine acceptance criteria (*Given… When… Then…*)	Monitor bugs by seniority	Monitor sprint reviews Ask team which is the DoD
SOLVING	Increase test automation Introduce QA metrics	Adopt AC format Include test scenarios in DoD	Implement code review Implement unit testing	Run a clarification session Review periodically DoD
ADAPTING	Don’t adapt, solve it!	Don’t adapt, solve it!	Create a bug fixing squad Accept workarounds	Don’t adapt, solve it!

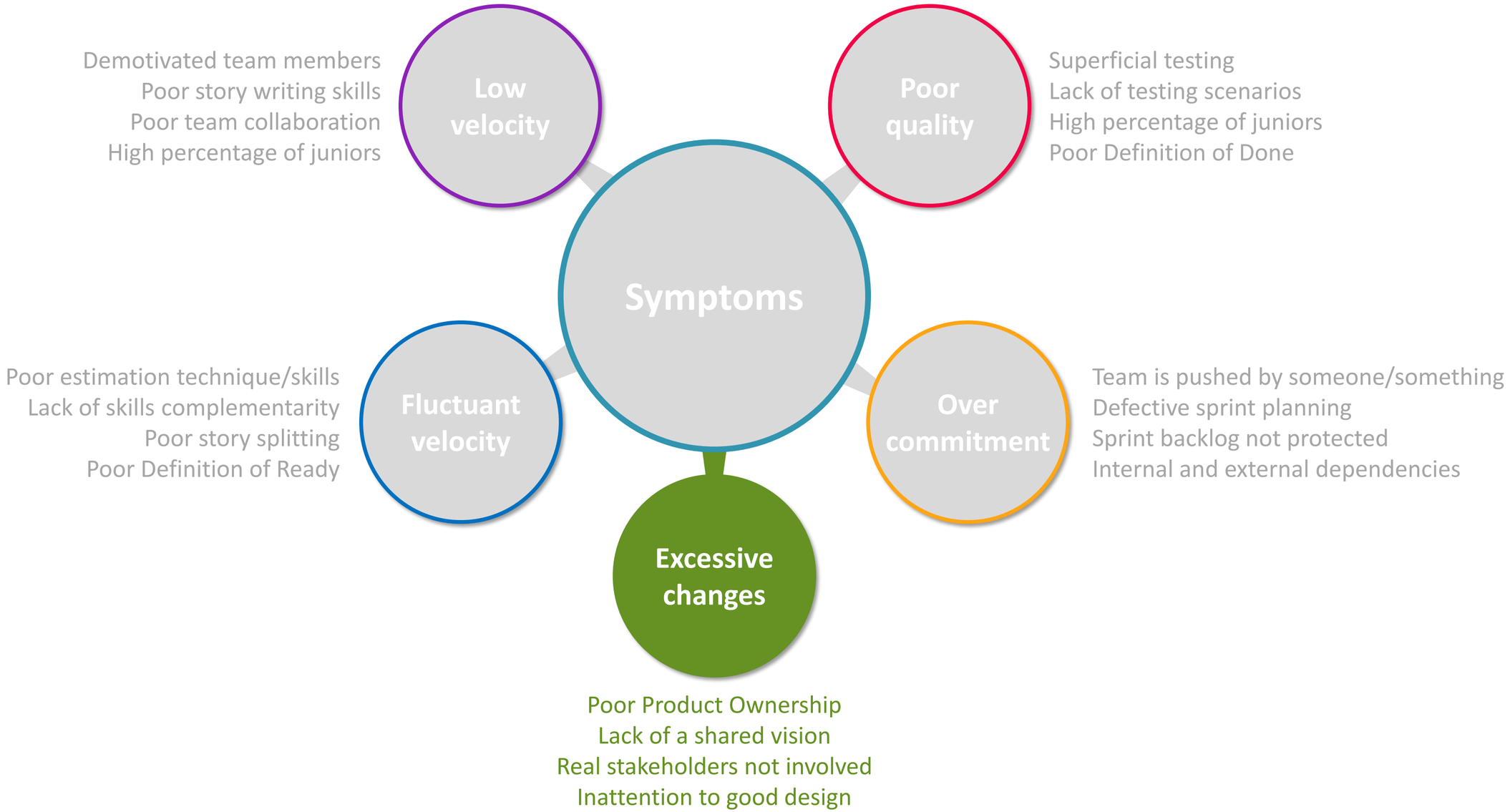
Most frequent symptoms



Over commitement (constant or frequent)

	Team is pushed by someone/something	Defective sprint planning	Sprint backlog not protected	Internal & external dependencies
CLASSIFYING	Always an issue	Always an issue	Always an issue	Usually an issue, sometimes a constraint
DIAGNOSING	Monitor communication Discuss with informal leaders	Inspect planning practice Examine task allocation	Monitor changes of sprint backlog Daily Scrum/Standup	Monitor for waitings & approvals (process waste)
SOLVING	Coach the pushing person Coach team to commit	Split stories in subtasks Introduce WIP limits in sprint	Coach PO/stakeholders Coach team to discipline	Include dependency in DoR Remove dependency from DoD
ADAPTING	Don’t adapt, solve it!	Don’t adapt, solve it!	Don’t adapt, solve it!	Improve availability of external resources

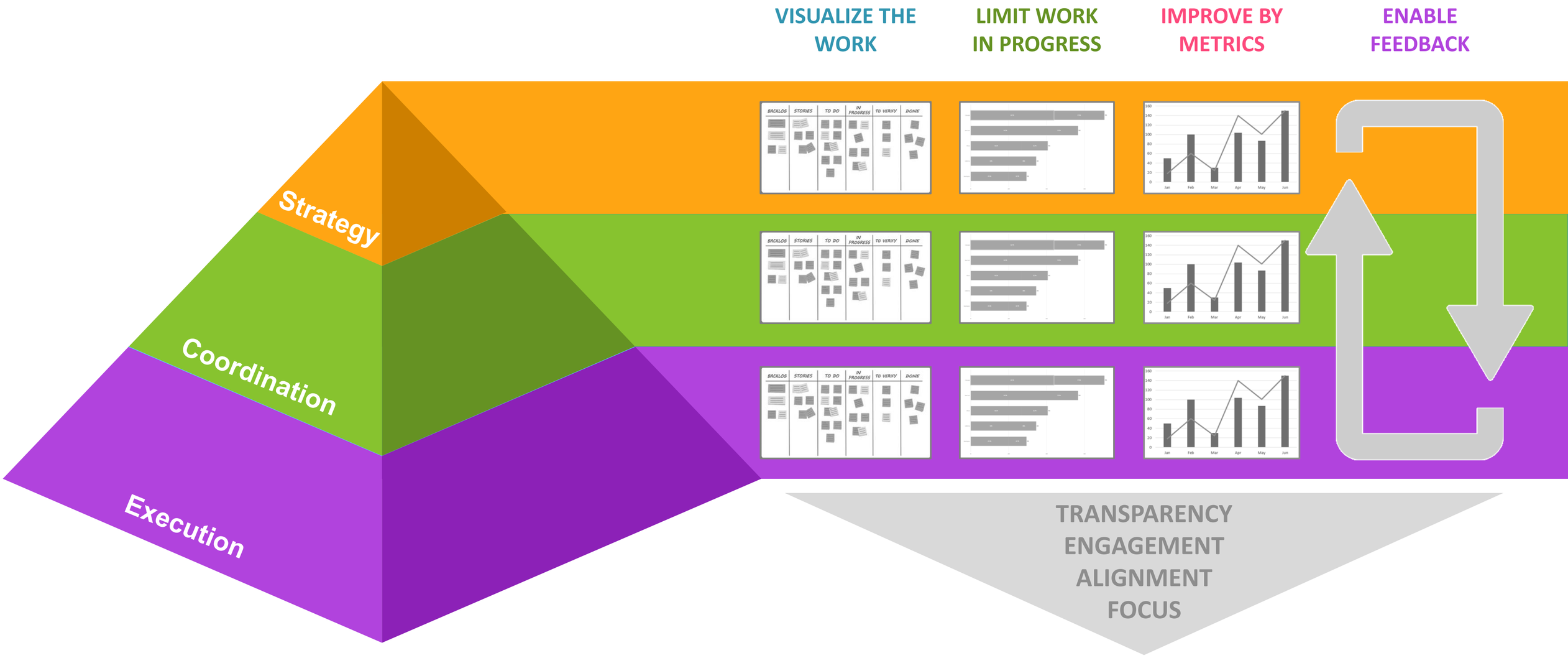
Most frequent symptoms



Excessive changes (affecting budget and time)

	Poor product ownership	Lack of a shared vision	Real stakeholders not involved	Inattention to good design
CLASSIFYING	Always an issue	Always an issue	Usually an issue, sometimes a constraint	Always an issue
DIAGNOSING	Inspect project backlog Discuss with stakeholders	Inquire team members Examine PO-team alignment	Monitor decision making process	Create a refactoring backlog Monitor refactoring needs
SOLVING	Coach the PO Get support for PO	Reiterate project goals Create project visual maps	Get real decision makers on board	Get support from architects Create solution architecture
ADAPTING	Don’t adapt, solve it!	Don’t adapt, solve it!	Implement pseudo dual track (prototype-develop)	Don’t adapt, solve it!

Project governance



Coordination performance