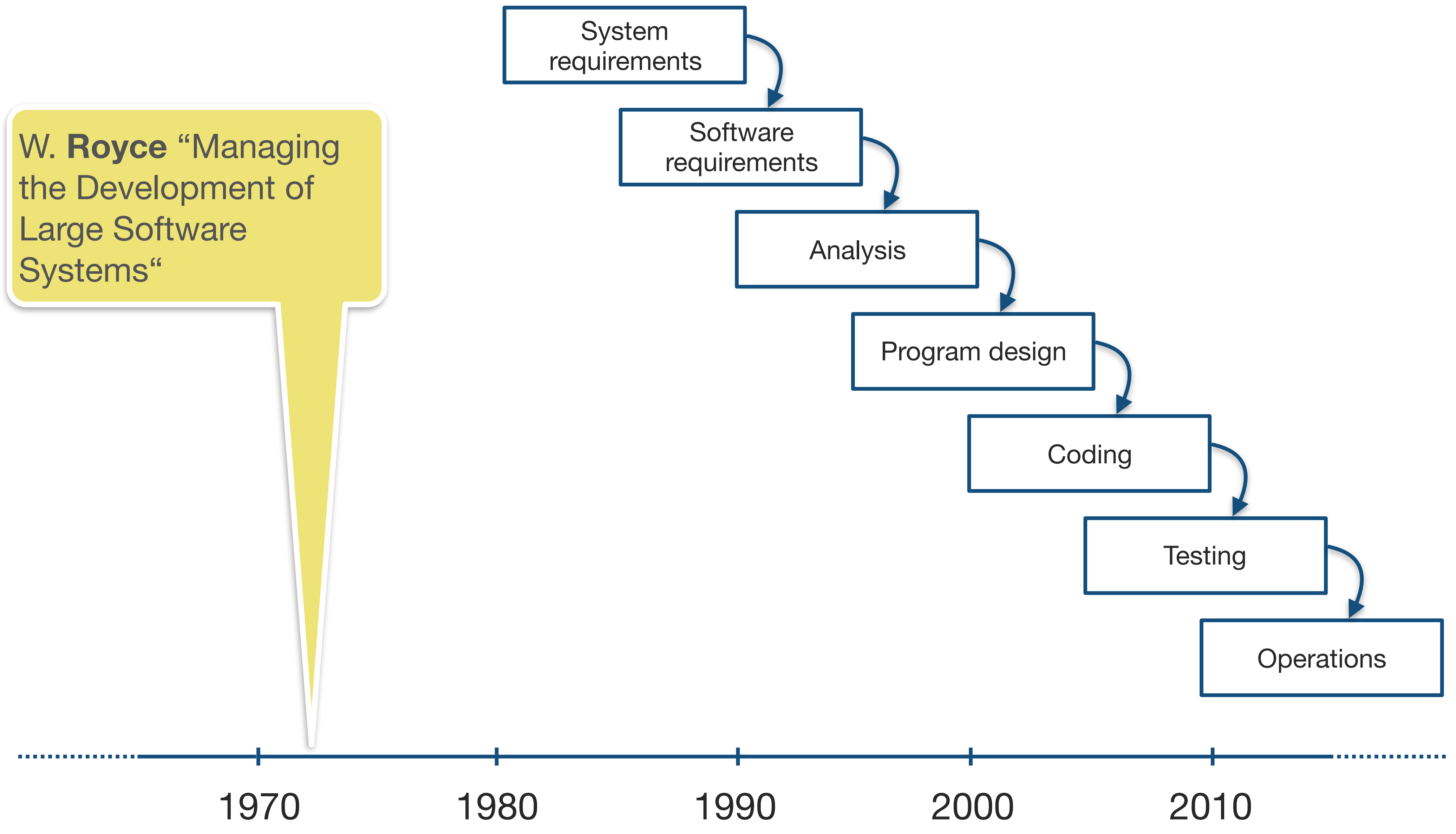
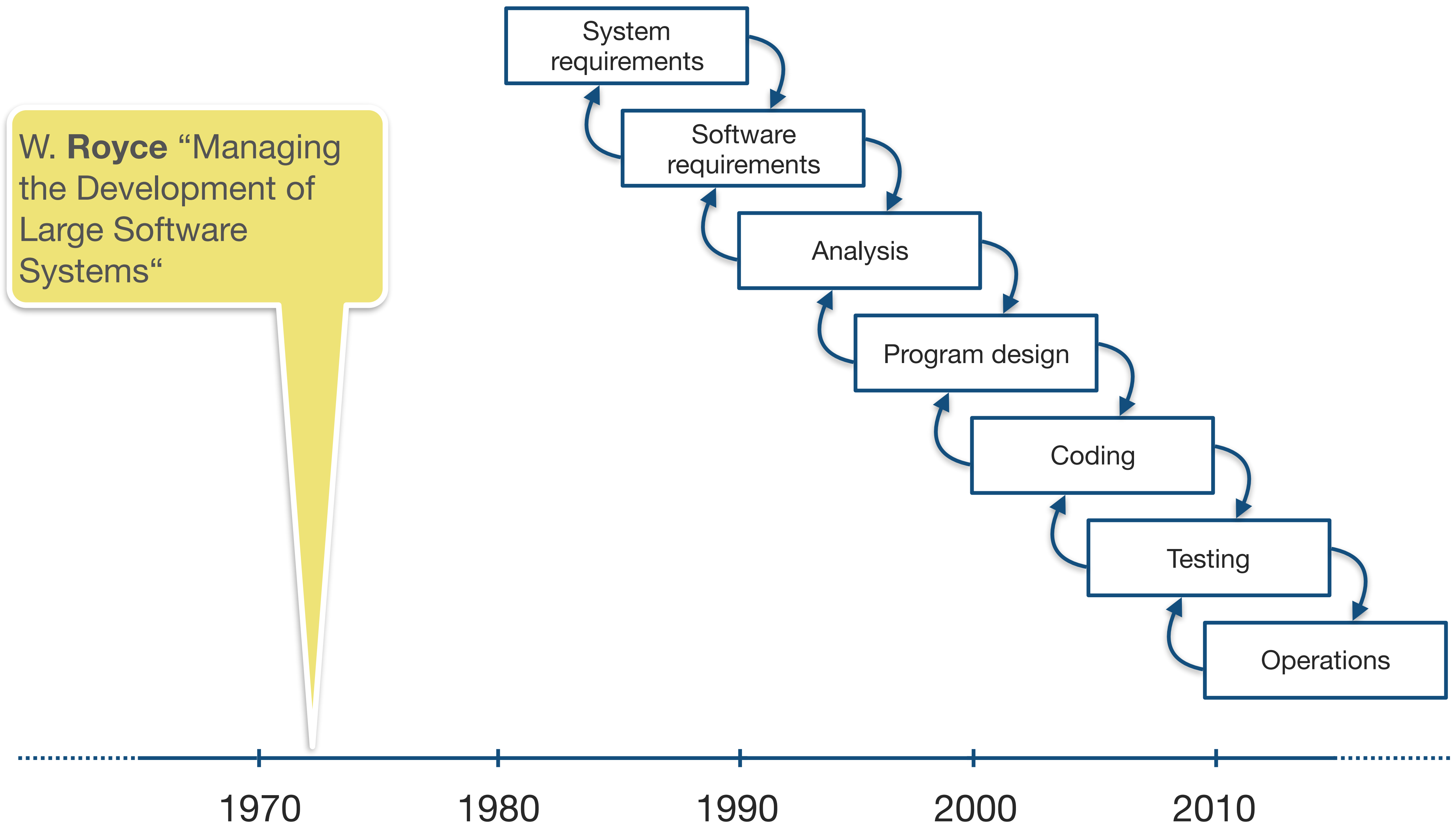
Agile, why?

Agile Software Development and its Manifesto



1970s - Waterfall model

Managing the Development of Large Software Systems by Winston W. Royce, IEEE 1970



1970s - Waterfall model

Managing the Development of Large Software Systems by Winston W. Royce, IEEE 1970

“risky and invites failure”

“testing phase occurs at the end of the development cycle”

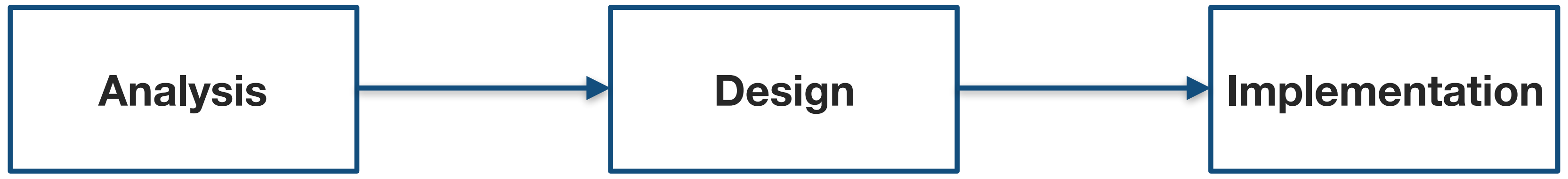
“one can expect up to 100% overrun in schedule and/or costs”

“design changes…so disruptive that…requirements…violated”

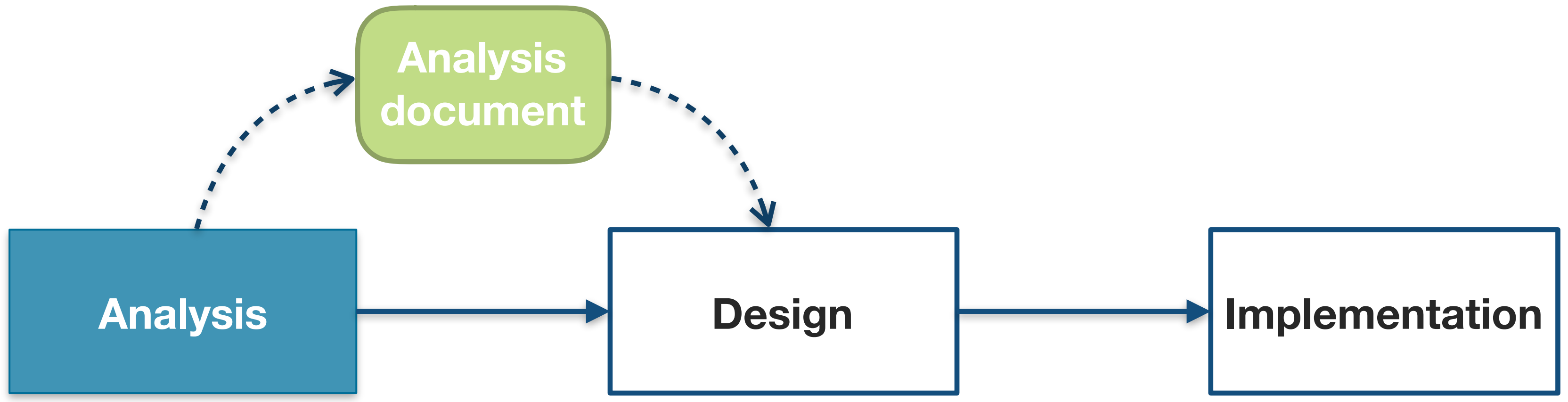
1970s - Waterfall model

Managing the Development of Large Software Systems by Winston W. Royce, IEEE 1970

Three Simple Phases

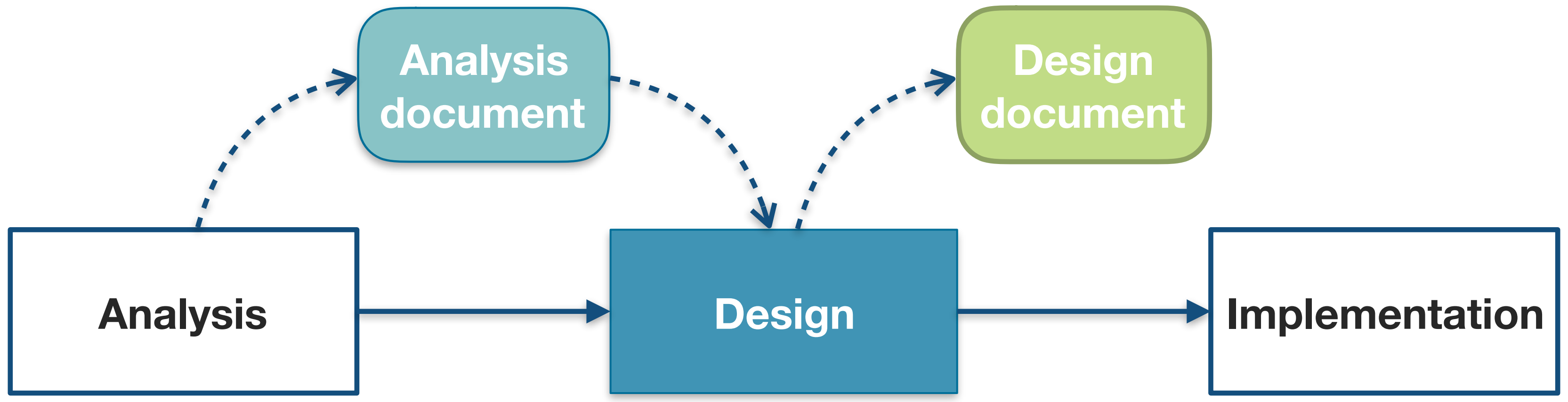


Three Simple Phases



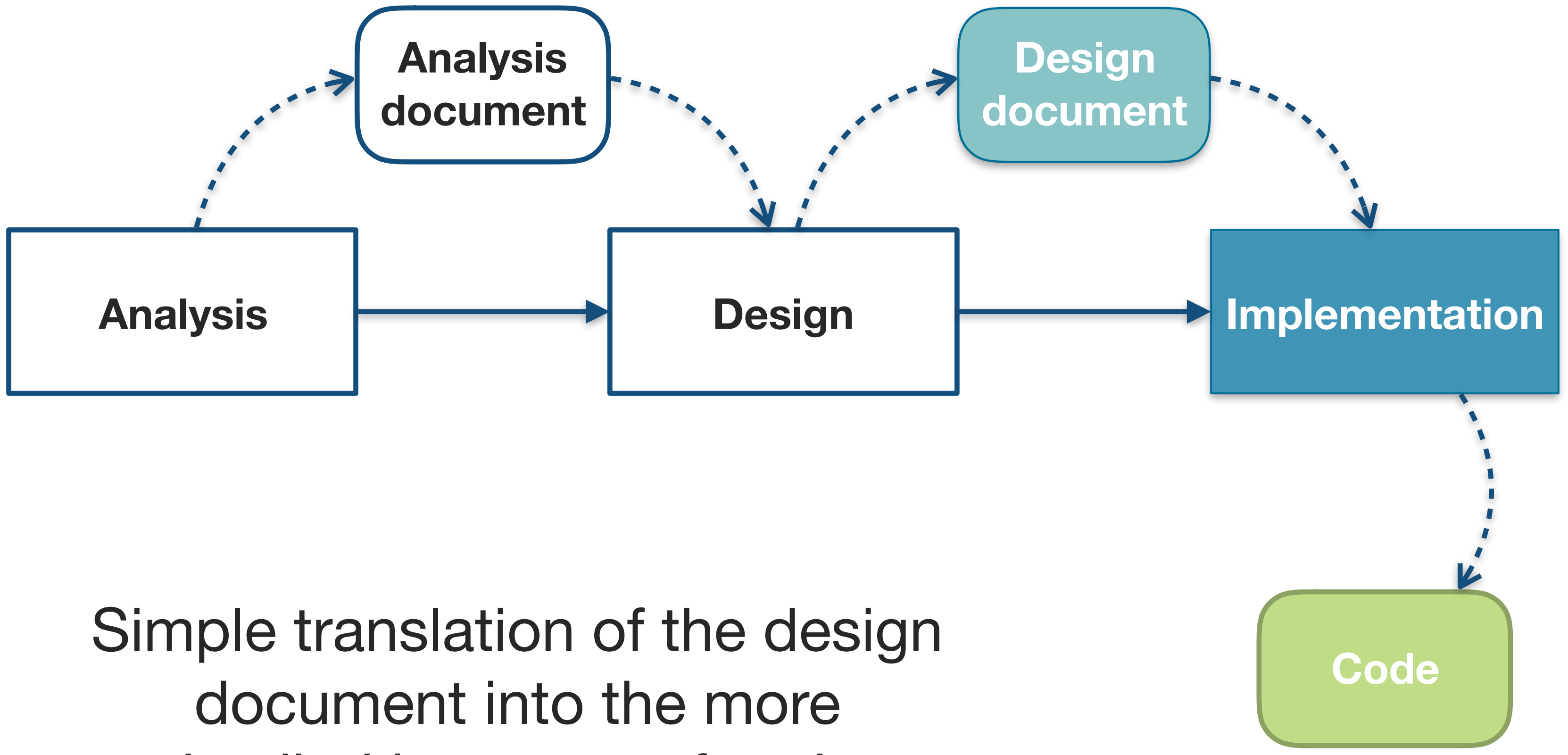
Produce an analysis document, specifying high level structure and goals.

Three Simple Phases



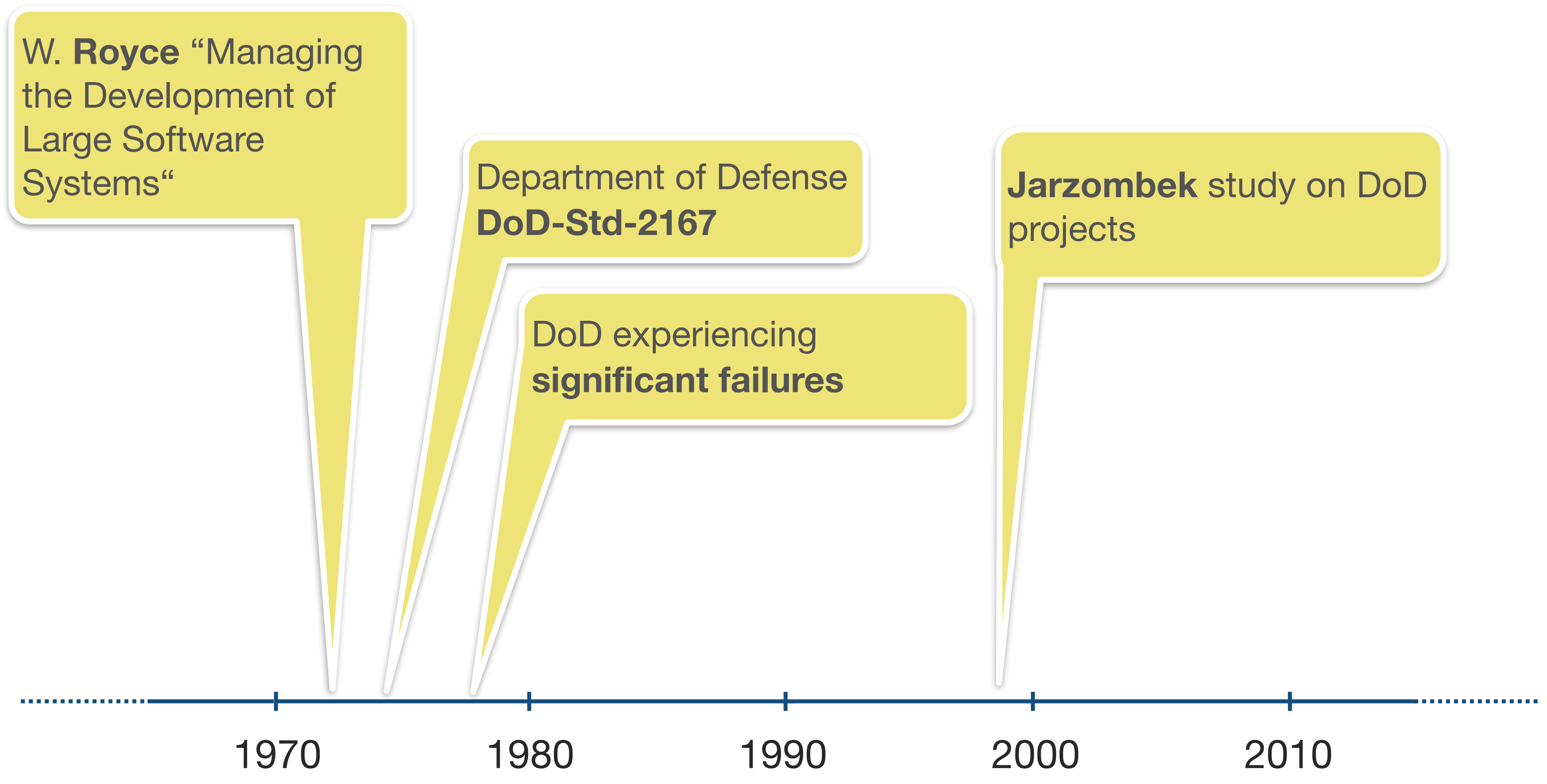
Translate the analysis document into a lower level document specifying functions and algorithms.

Three Simple Phases



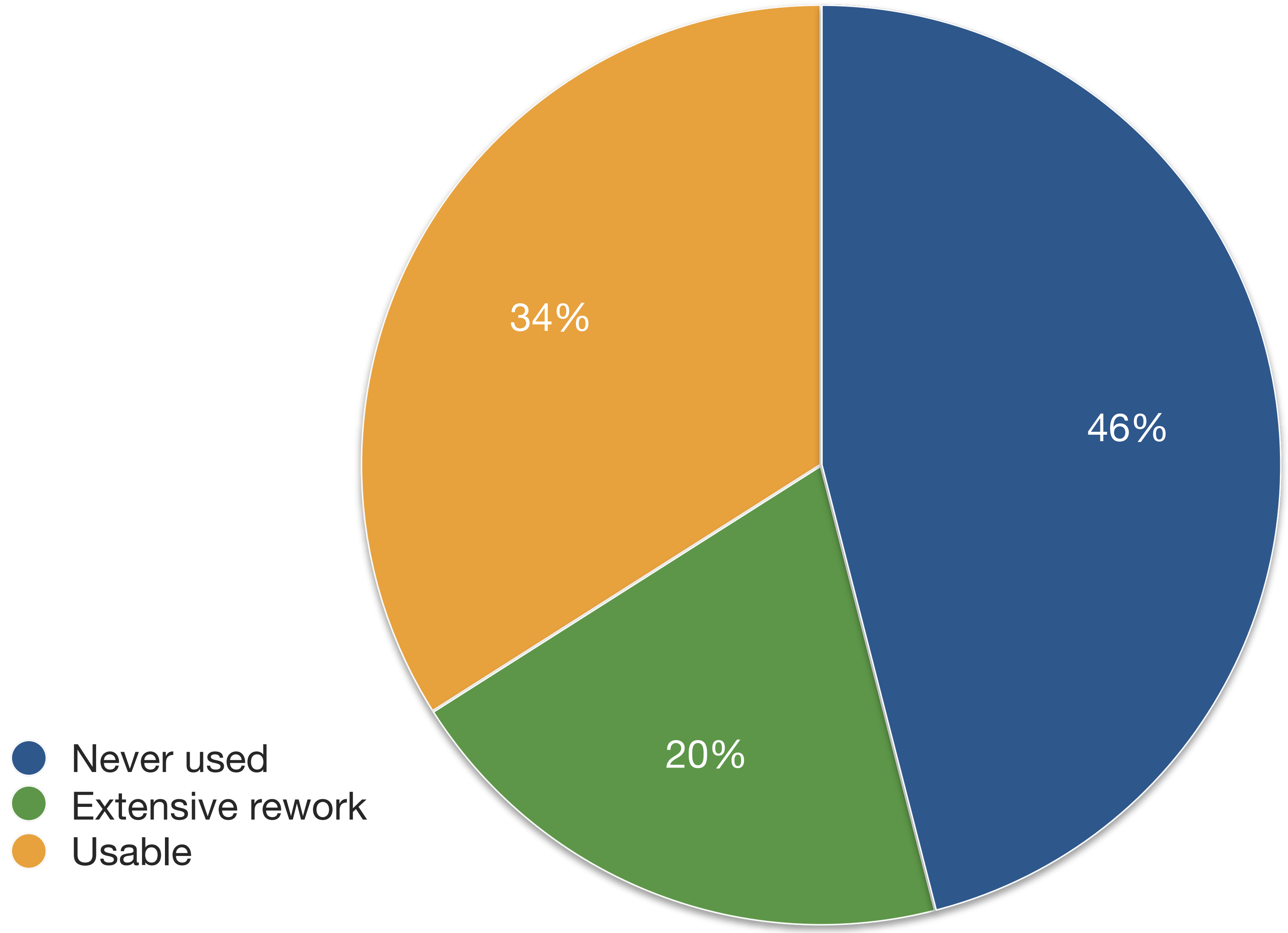
Simple translation of the design document into the more detailed language of code.

Strict, document-driven, single-pass waterfall model

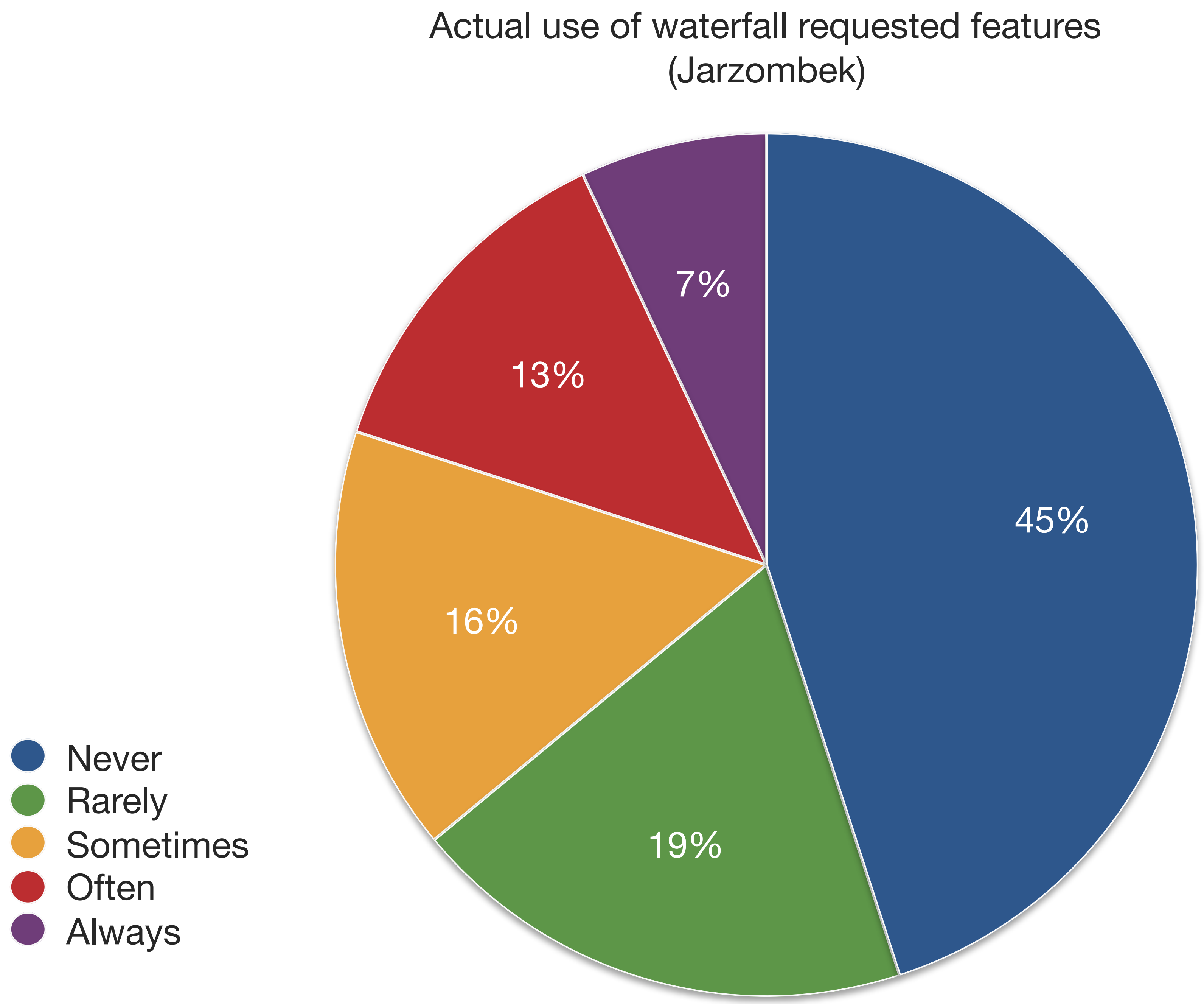

W. Royce "Managing the Development of Large Software Systems"
Department of Defense DoD-Std-2167
DoD experiencing significant failures
Jarzombek study on DoD projects
1970
1980
1990
2000
2010

Why were projects failing?

$37B worth of DOD projects using a waterfall approach (Jarzombek)



Why were projects failing?



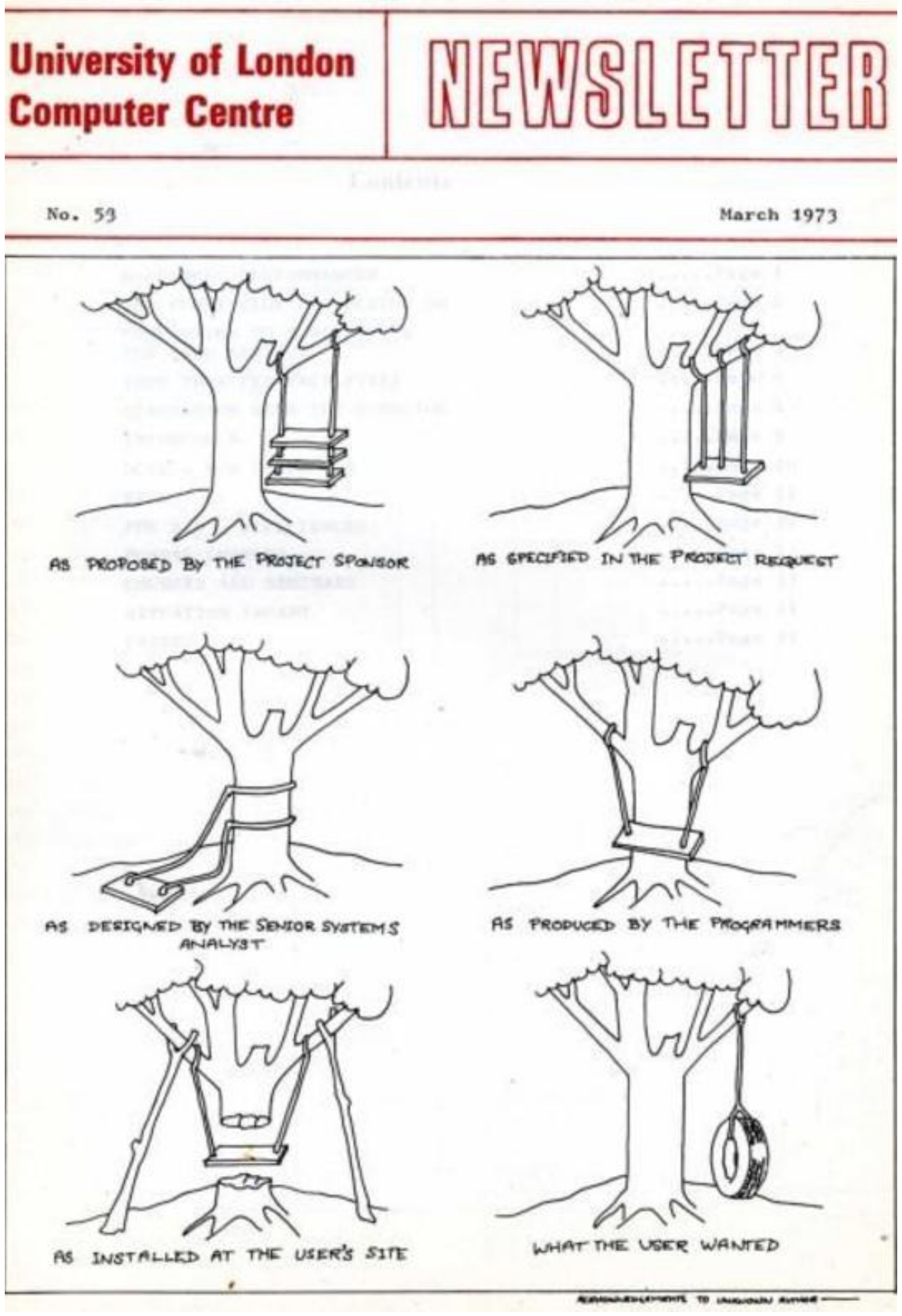
Why were projects failing?

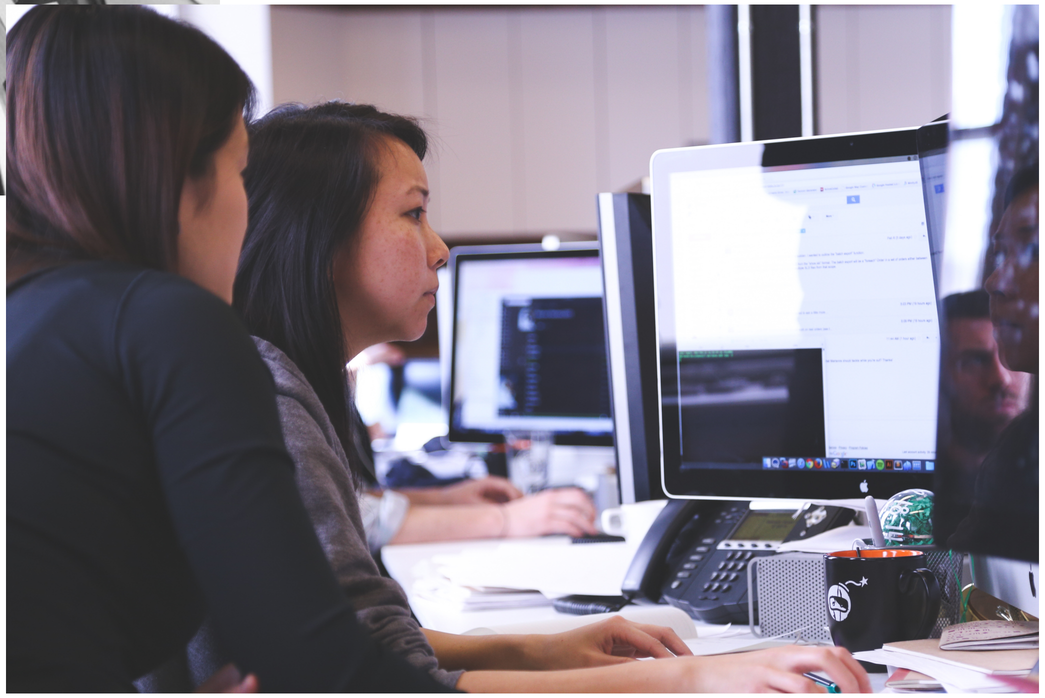
In **1987 Fred Brooks** led a task-force for the DOD to find out just what was going wrong.

“...the document-driven, specify-then-build approach ... lies at the heart of so many ... software problems.”

The Problem with Requirements

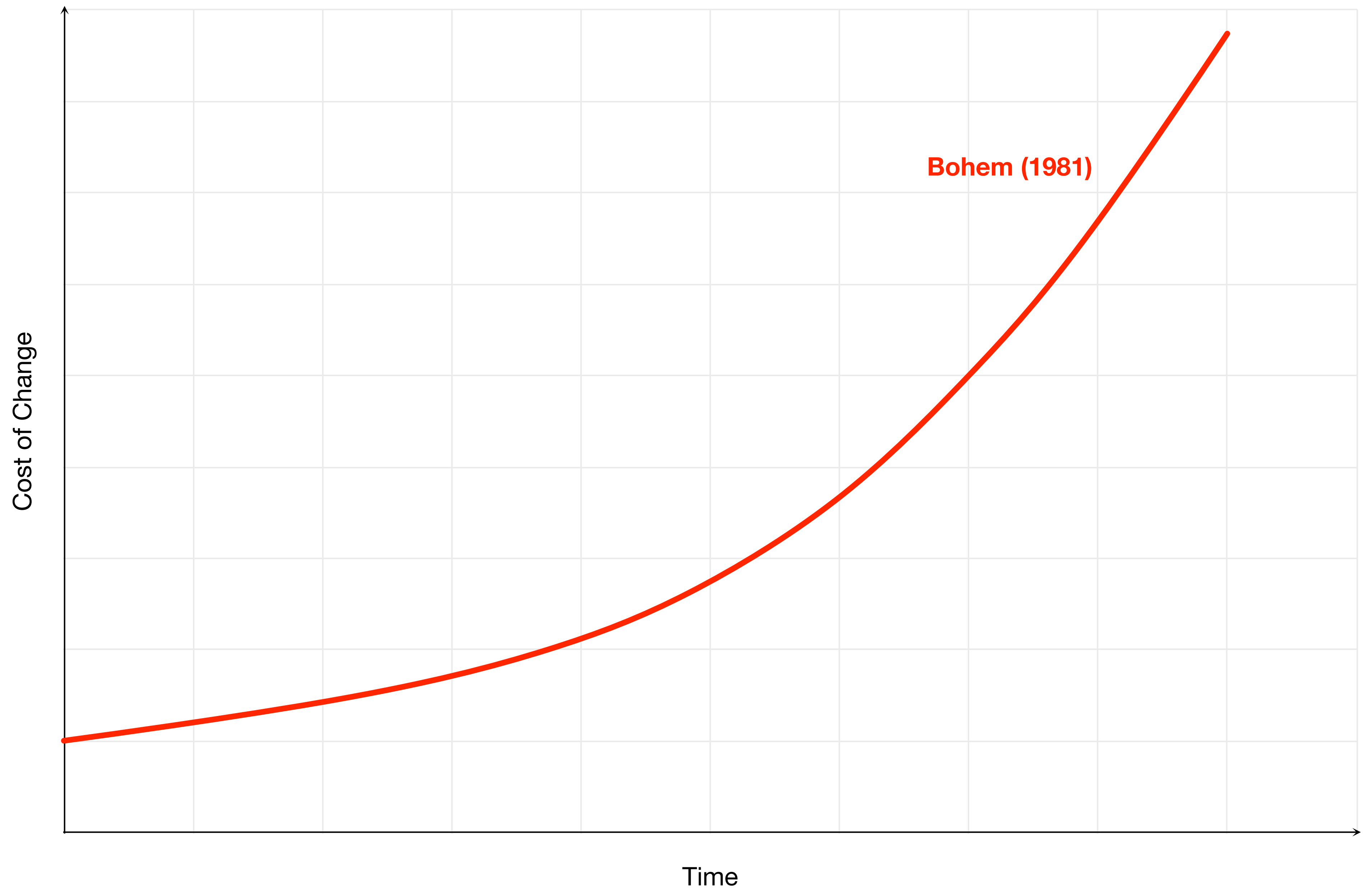
March 1973


University of London Computer Centre
NEWSLETTER
No. 53
March 1973
AS PROPOSED BY THE PROJECT SPONSOR
AS SPECIFIED IN THE PROJECT REQUEST
AS DESIGNED BY THE SENIOR SYSTEMS ANALYST
AS PRODUCED BY THE PROGRAMMERS
AS INSTALLED AT THE USER'S SITE
WHAT THE USER WANTED
ACKNOWLEDGEMENTS TO UNKNOWN AUTHOR



Charlie and Jane

Taylorism in software development



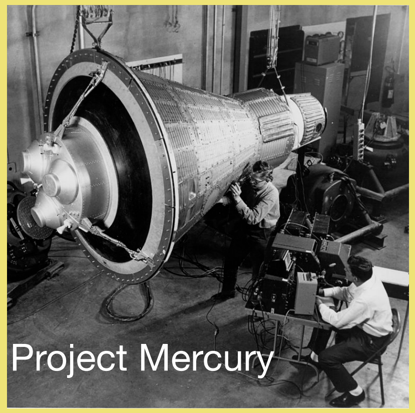
Cost of Change

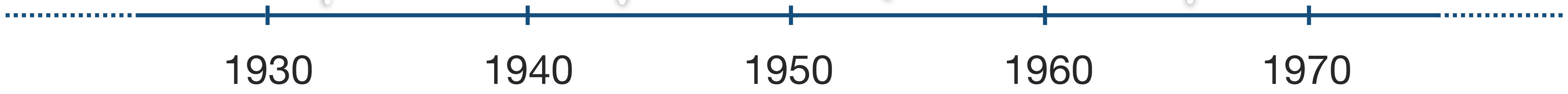
How to reduce it?

Iterative and incremental development

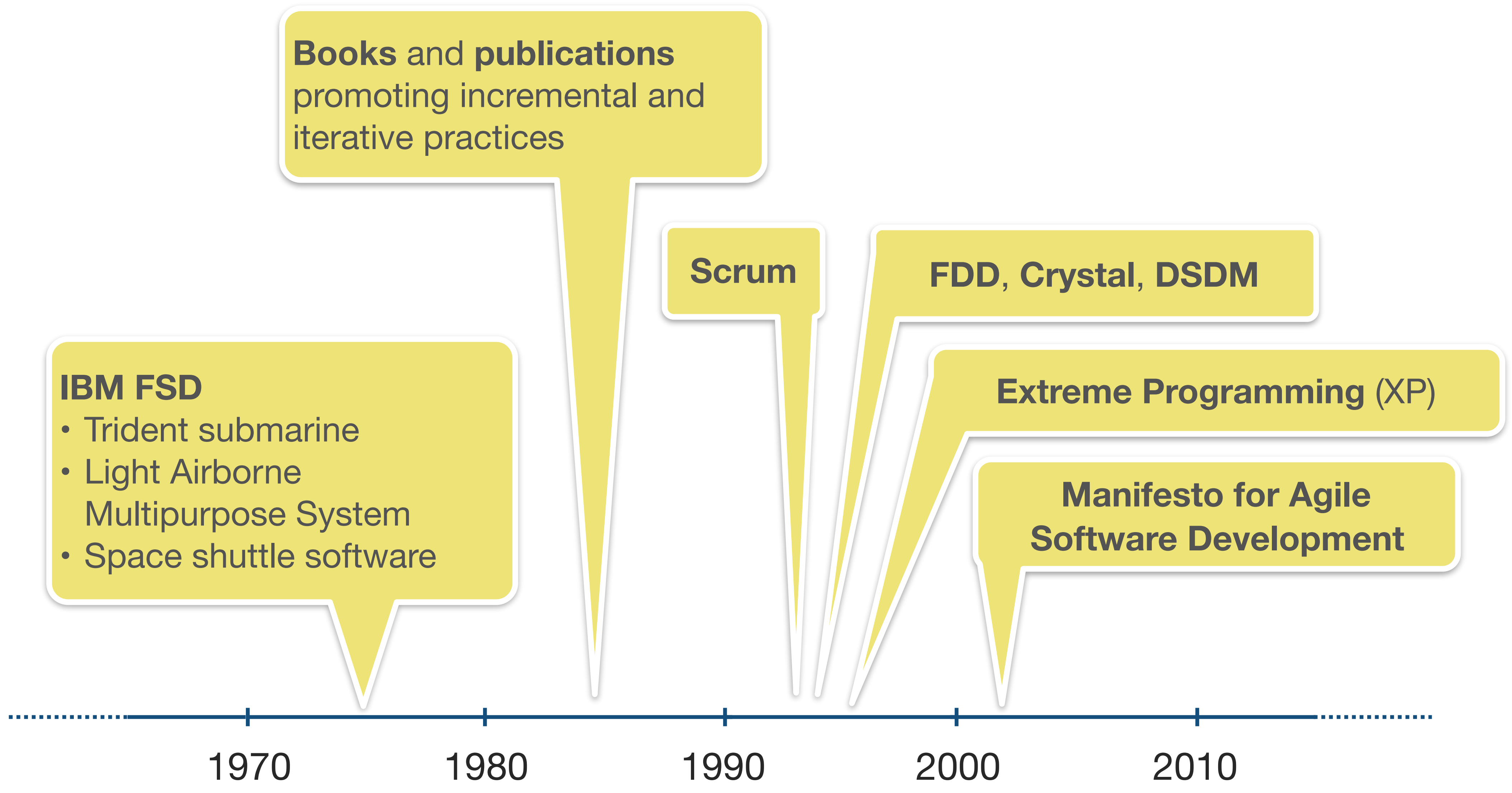
W. **Shewhart** "plan-do-study-act" (**PDSA**) cycles at Bell Labs

W. E. **Deming** vigorously promoting PDSA

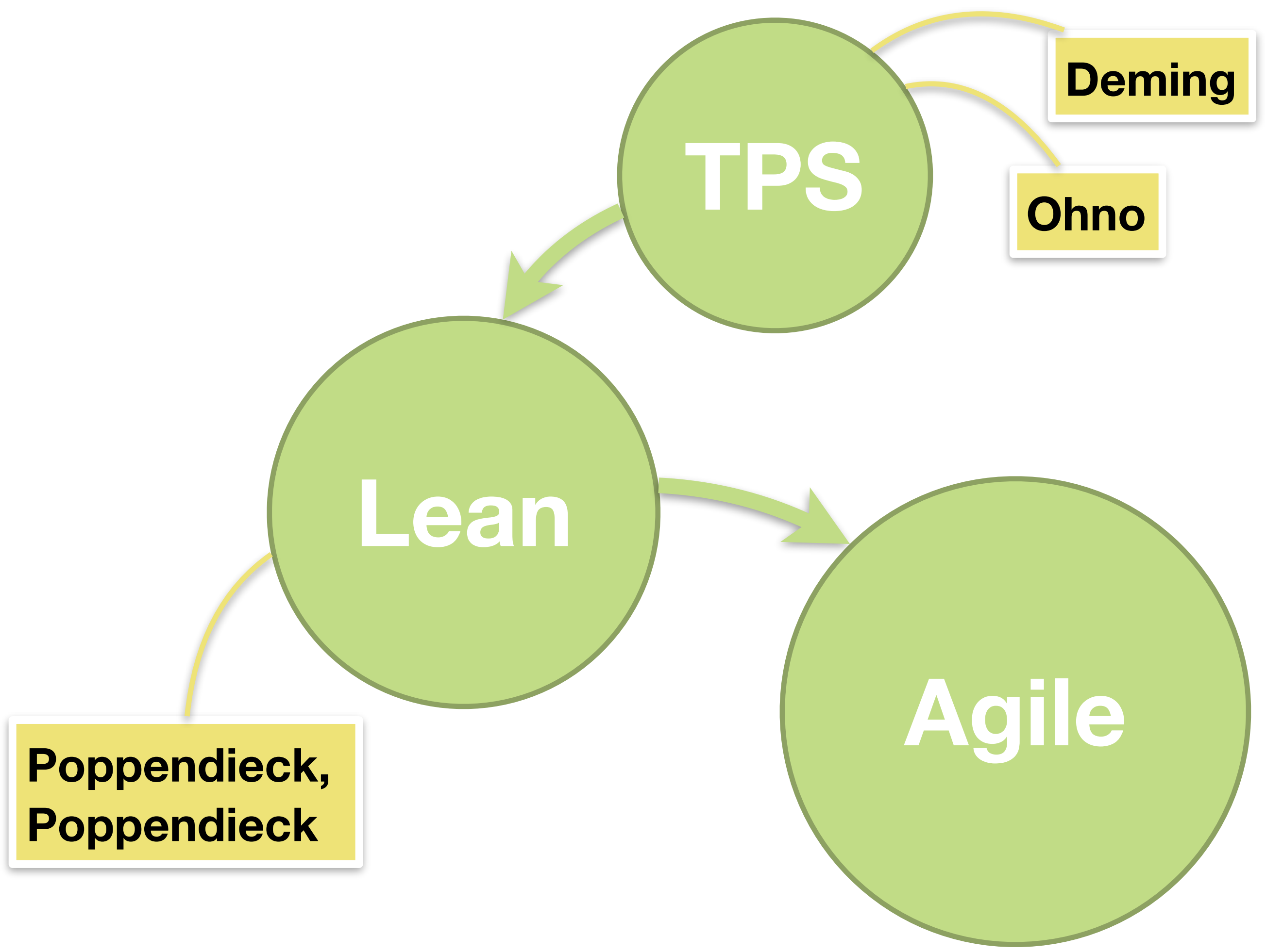




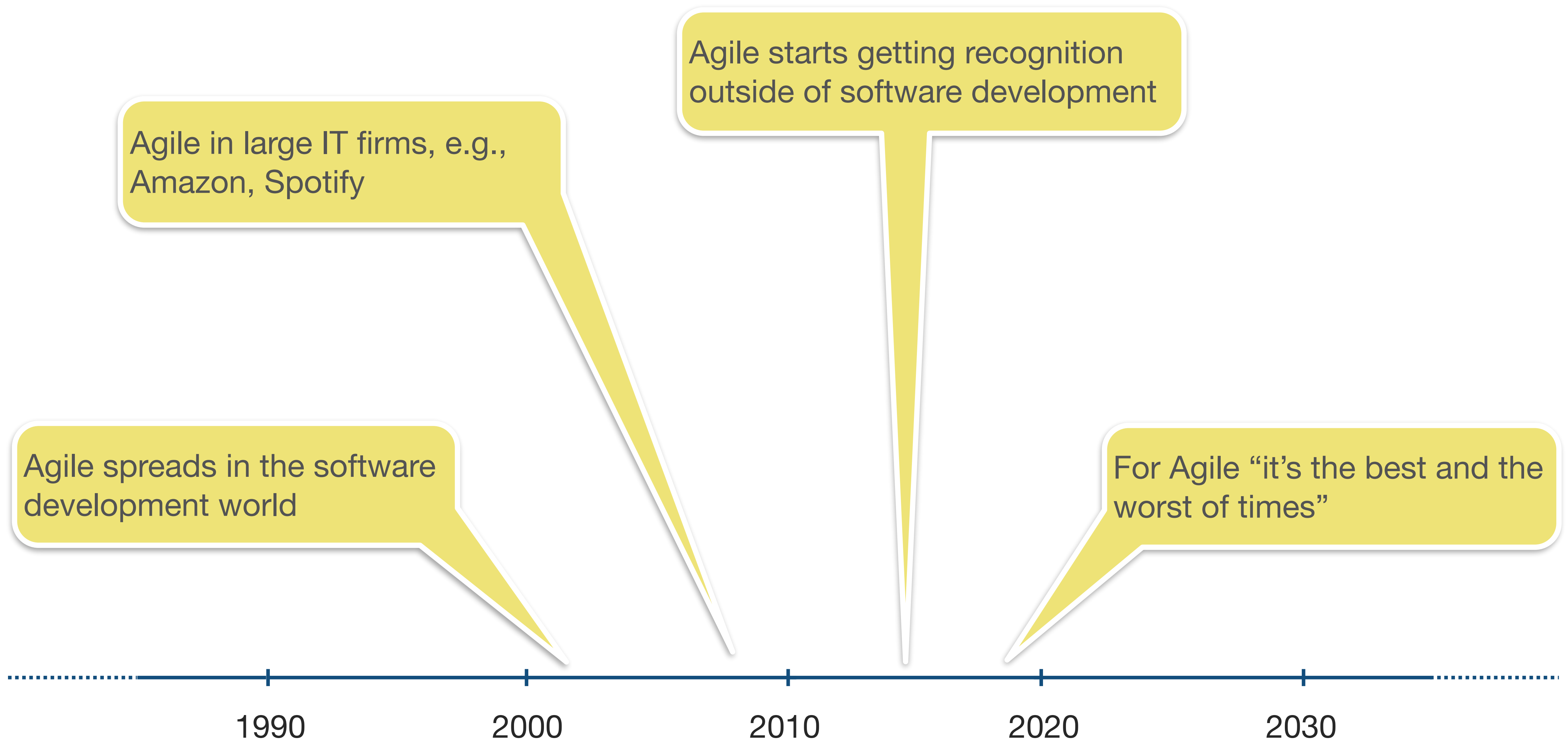
Iterative and incremental development


Books and **publications** promoting incremental and iterative practices
Scrum
FDD, Crystal, DSDM
Extreme Programming (XP)
Manifesto for Agile Software Development
IBM FSD
• Trident submarine
• Light Airborne Multipurpose System
• Space shuttle software
1970
1980
1990
2000
2010

From TPS to Lean and Agile



Agile in the last 17 years


Agile spreads in the software development world
Agile in large IT firms, e.g., Amazon, Spotify
Agile starts getting recognition outside of software development
For Agile "it's the best and the worst of times"
1990
2000
2010
2020
2030

Agile Today

- ⬆ Ways to deliver instant, intimate, incremental, risk-free value at scale
- ⬆ Spreading from IT Department to all parts, and all kinds, of organizations
- ⇩ Agile implemented as a superficial patch on traditional management
- ⇩ Huge amount of “fake Agile” going on

Characteristics of Agile

Adaptability

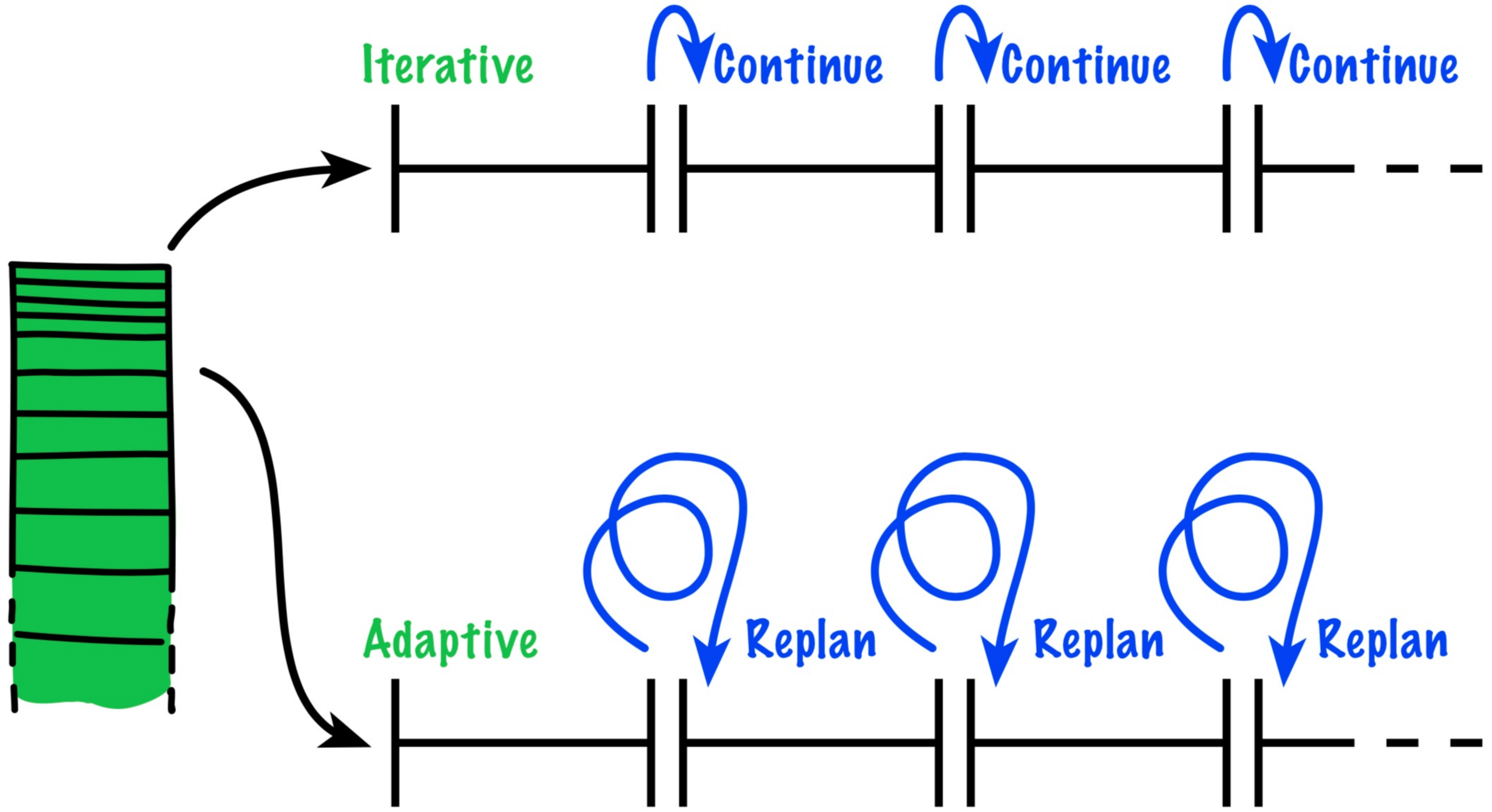
- Adaptability as a driver

⬆Agility/Adaptability ⇒ ⬆Value

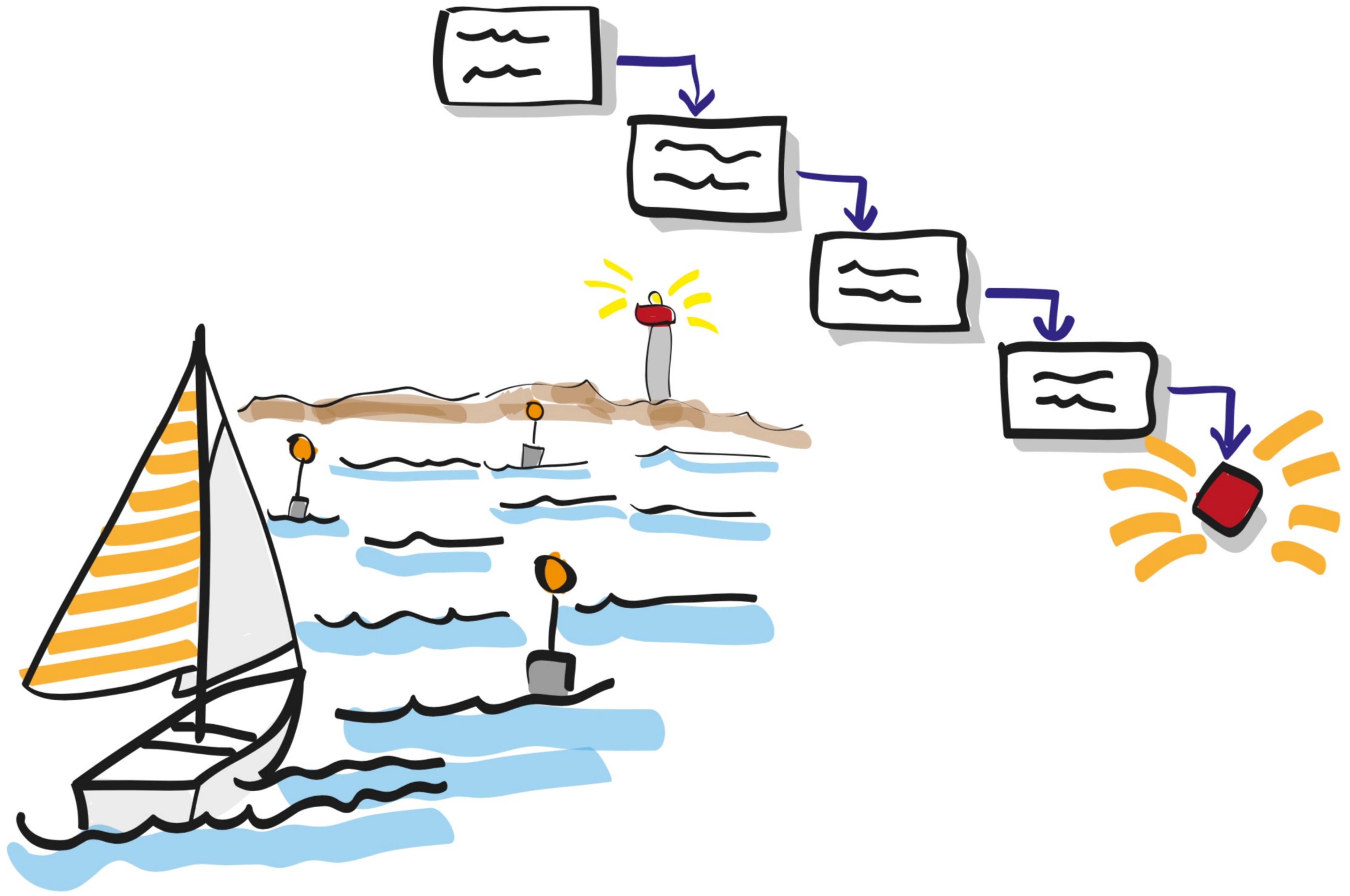
- Two common misunderstandings

Agility ≠ Fast

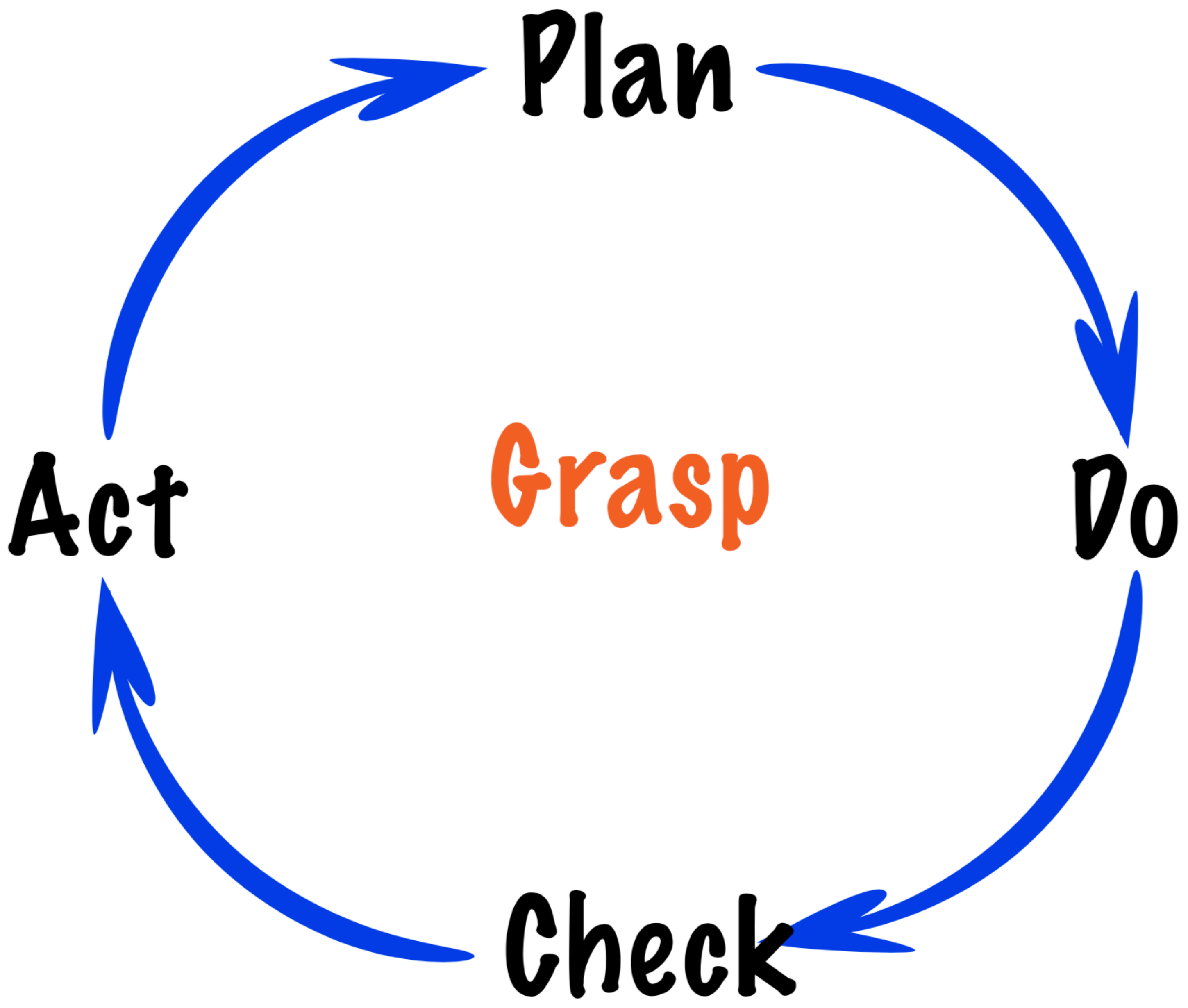
Agility ≠ Cheap



Adaptive Vs Iterative

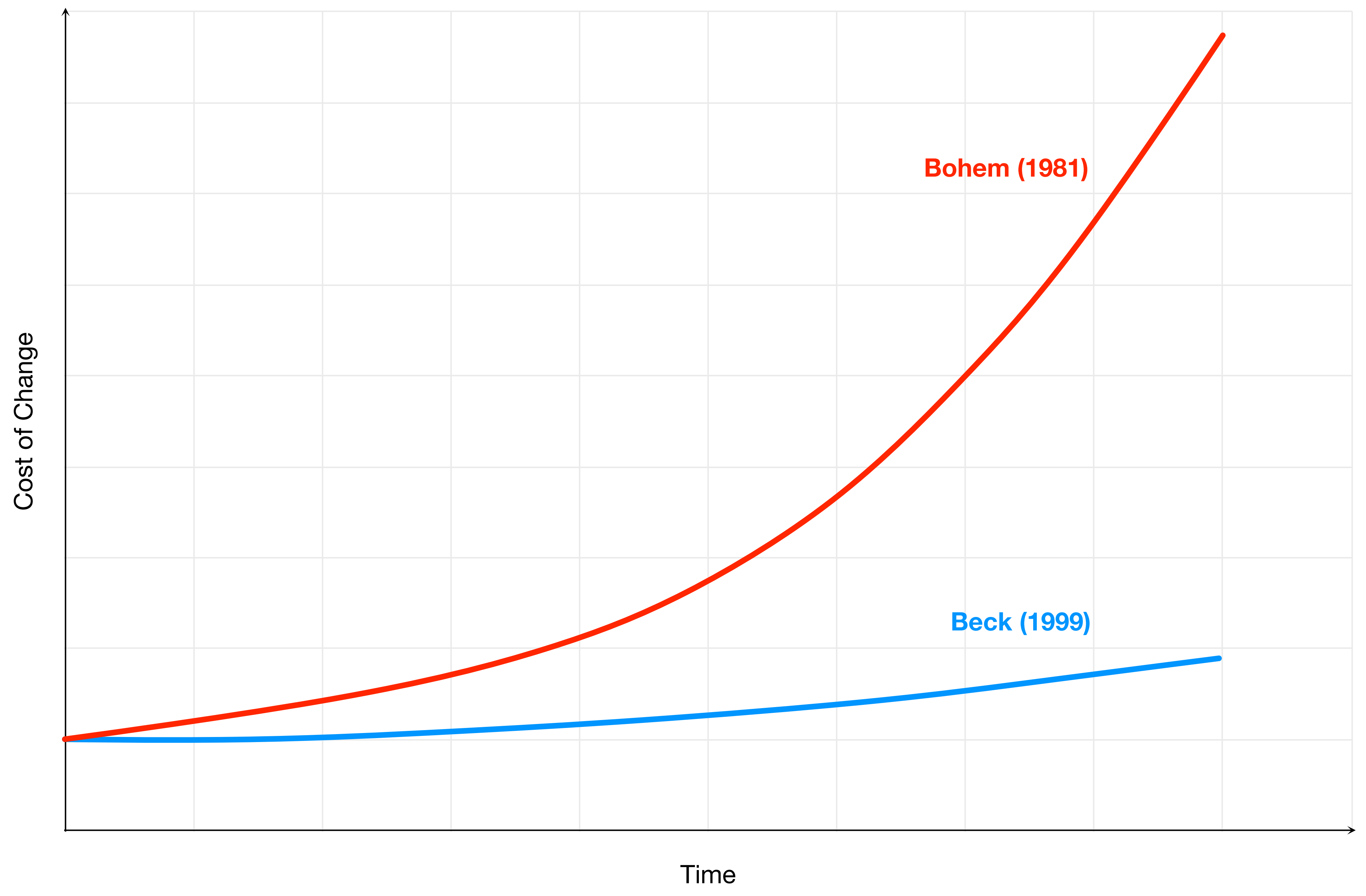


Empirical Vs Defined Process



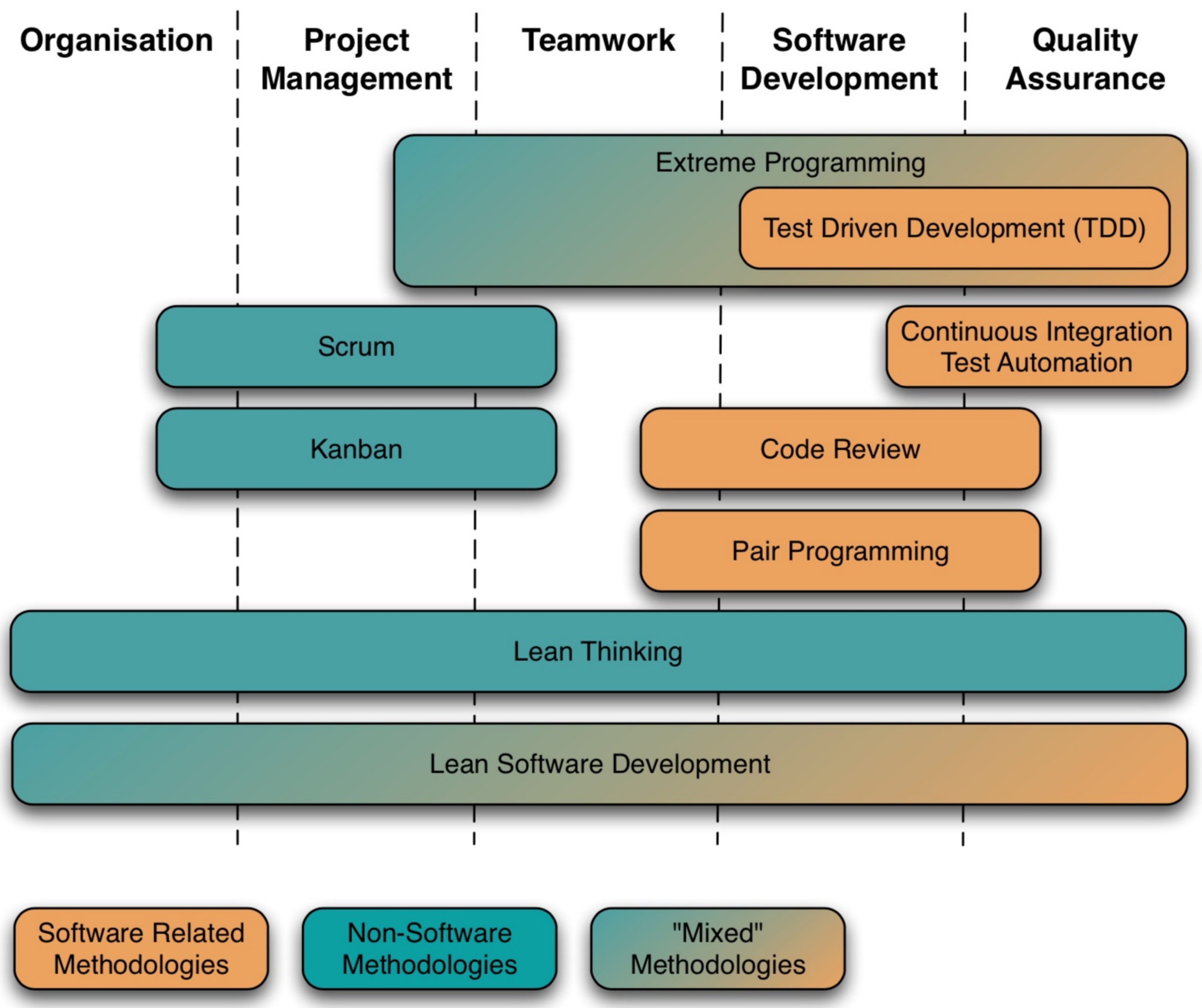
Continuous Improvement

Deming Cycle

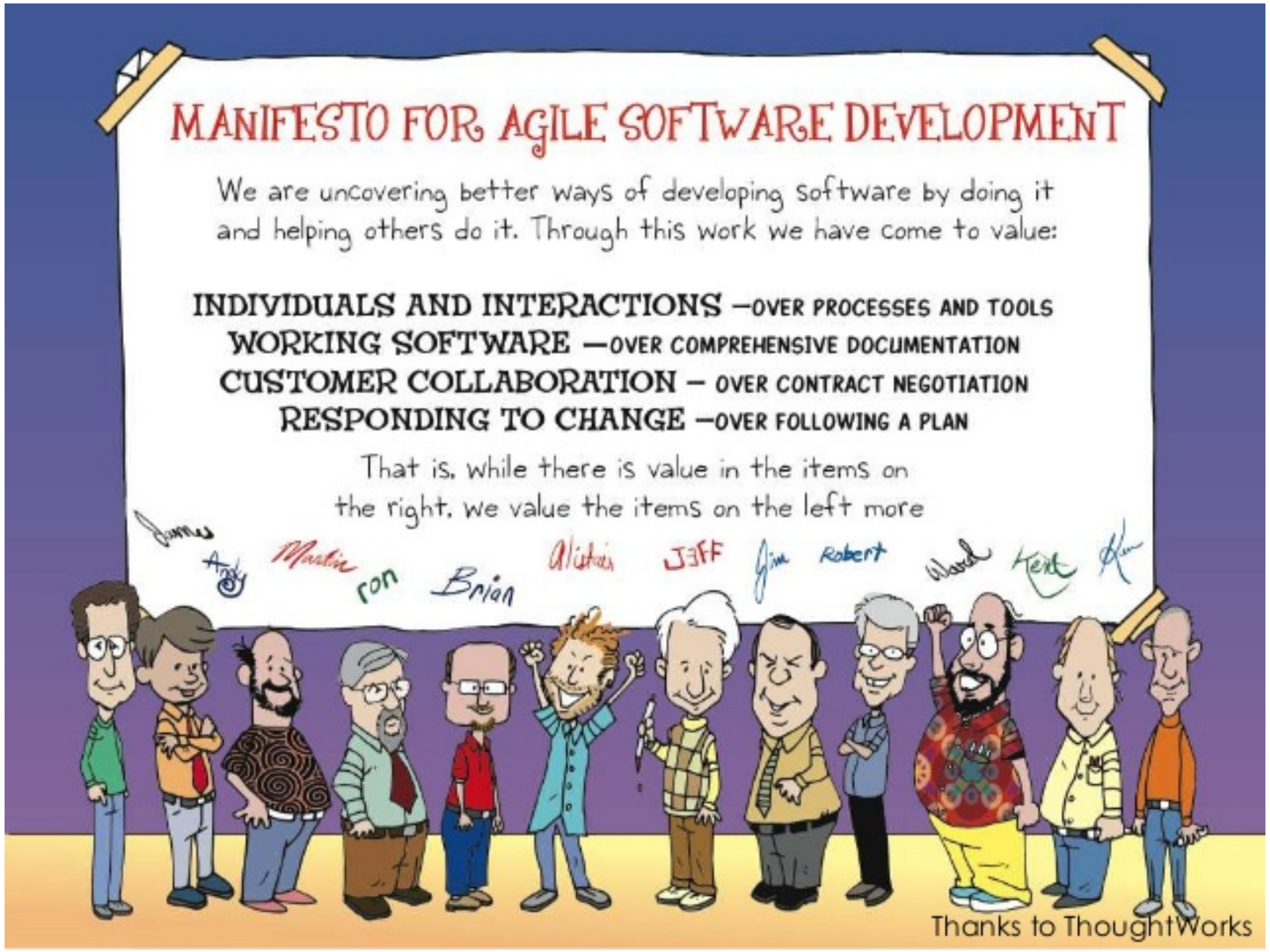


Cost of Change

Can be reduced by increasing quality



What is Agile?



MANIFESTO FOR AGILE SOFTWARE DEVELOPMENT

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

INDIVIDUALS AND INTERACTIONS —OVER PROCESSES AND TOOLS
WORKING SOFTWARE —OVER COMPREHENSIVE DOCUMENTATION
CUSTOMER COLLABORATION — OVER CONTRACT NEGOTIATION
RESPONDING TO CHANGE —OVER FOLLOWING A PLAN

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

Manifesto for Agile Software Development

Manifesto for Agile Software Development

We are uncovering better ways of developing

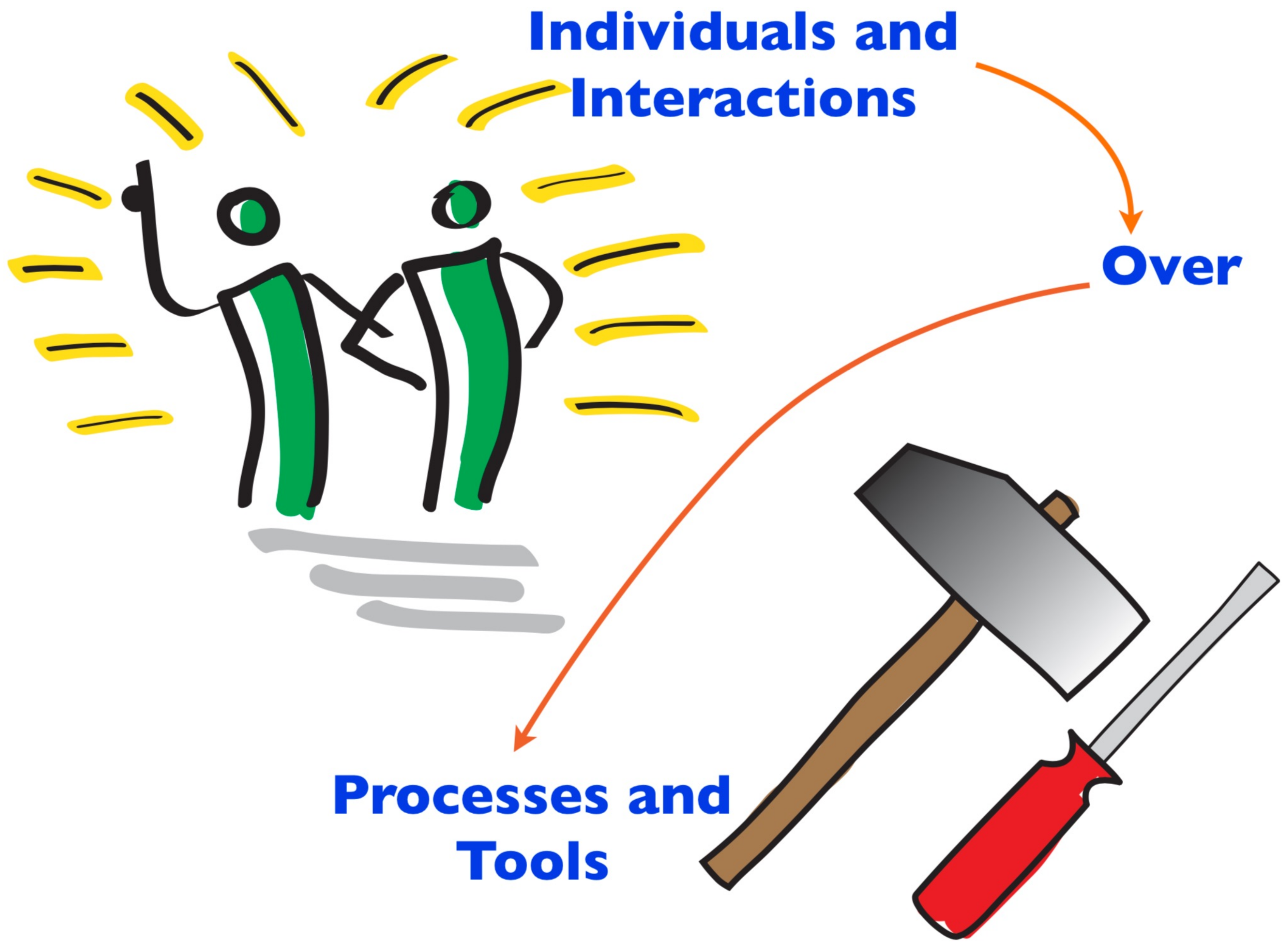
software by doing it and helping others do it.

Through this work we have come to value:

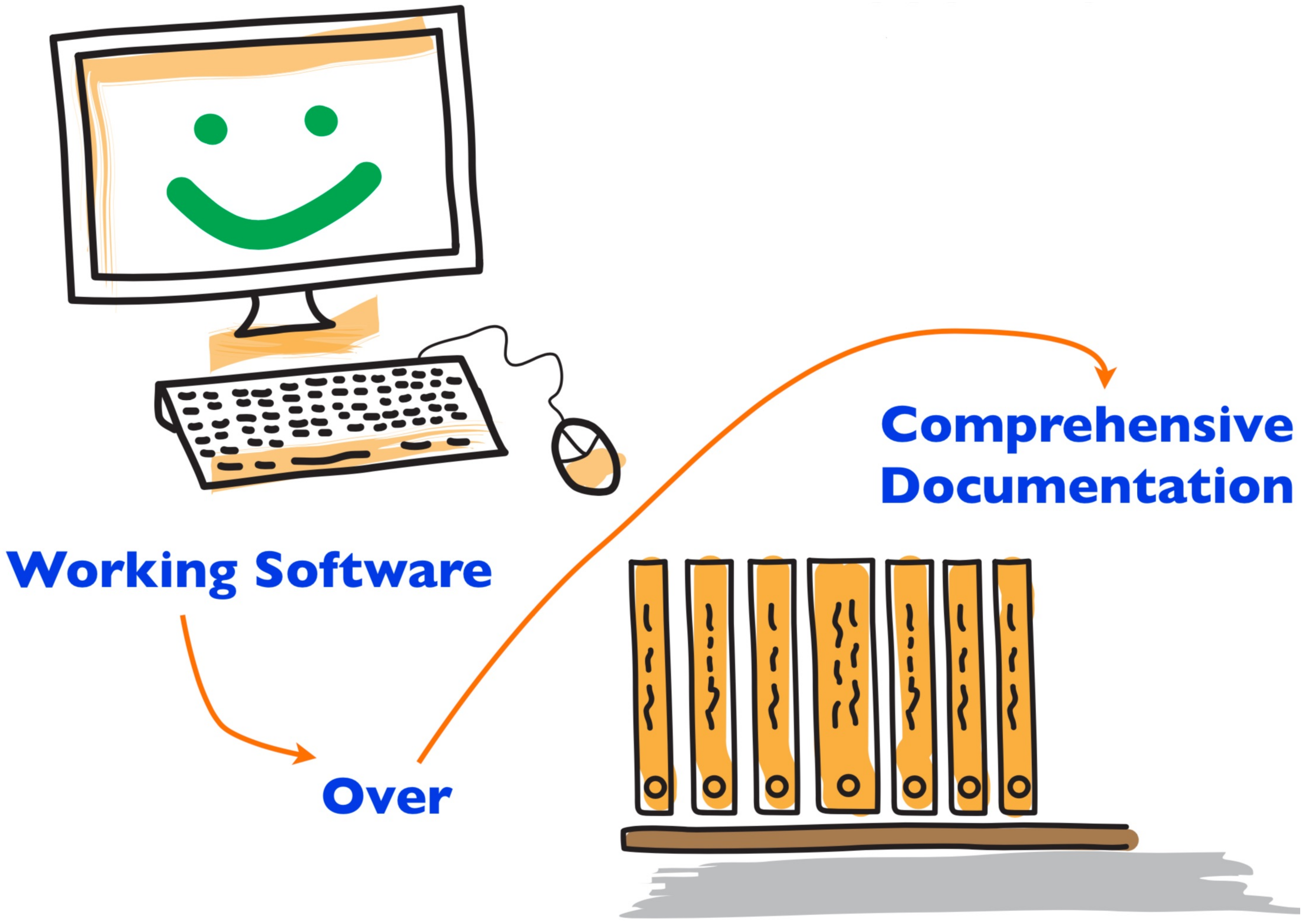
[…]

That is, while there is value in the items on

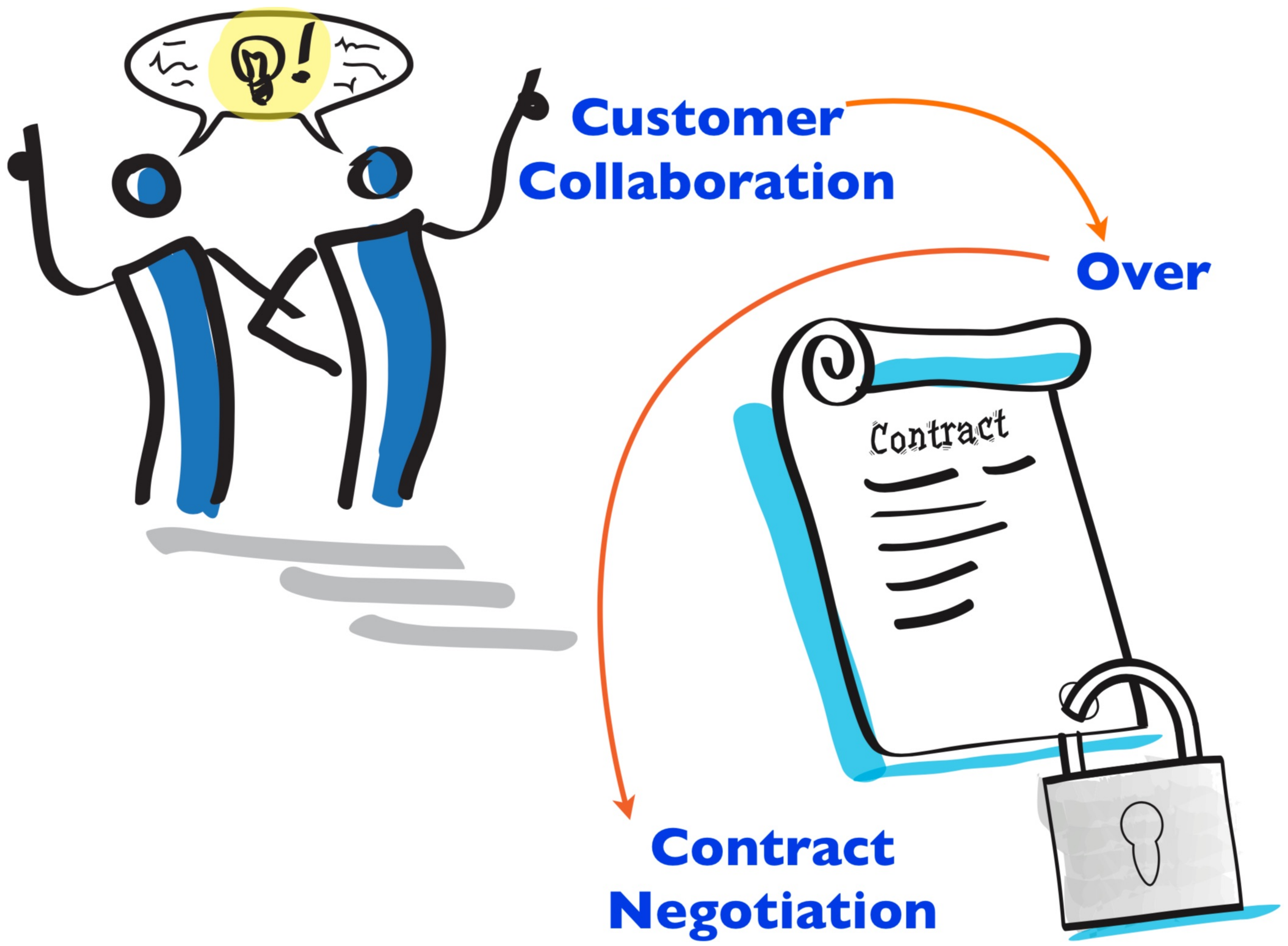
the right, we value the items on the left more.



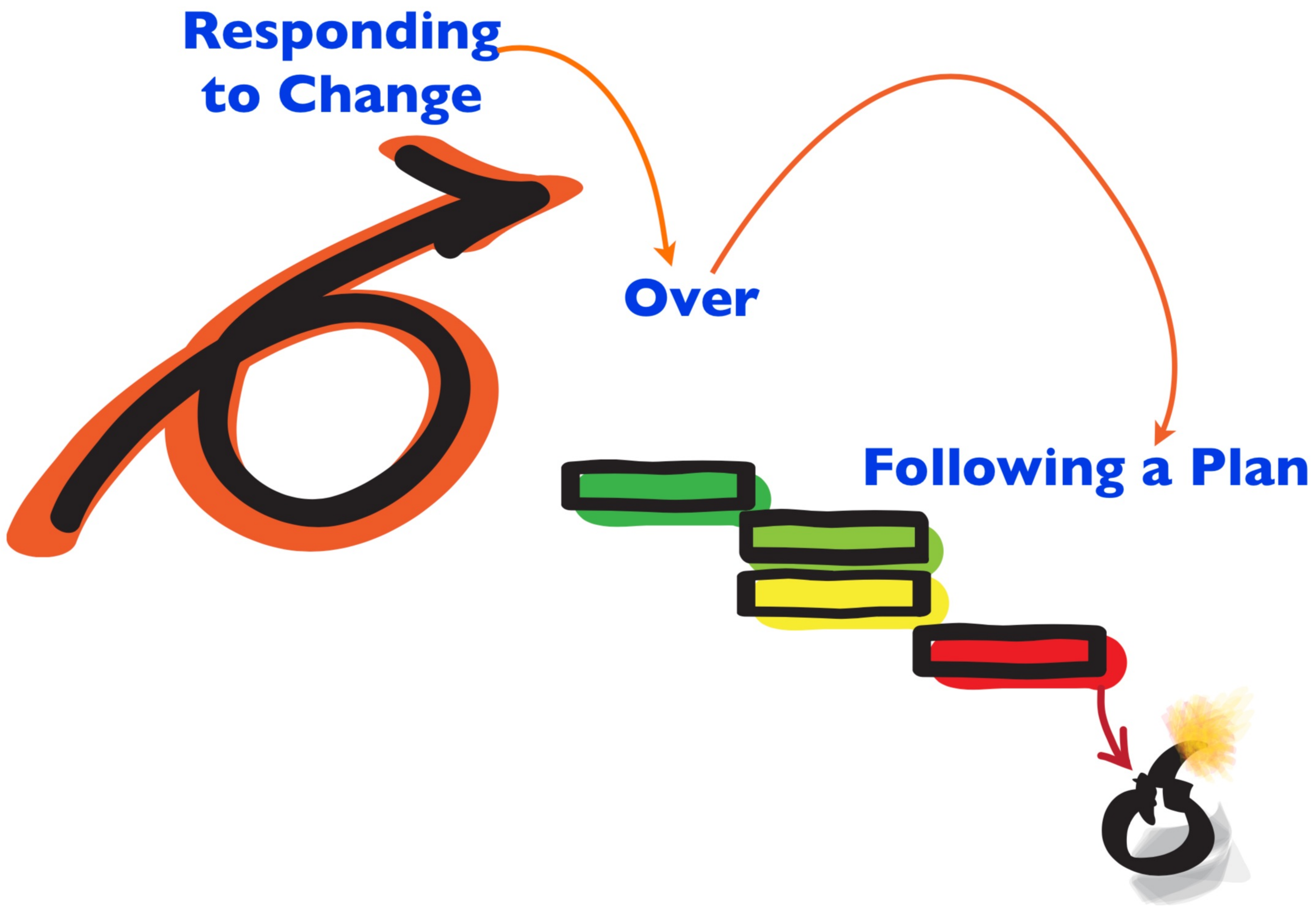
We value…



We value…


Customer Collaboration
Over
Contract
Contract Negotiation

We value…



We value…

We follow these principles:

1.

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

We follow these principles:

2.

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

We follow these principles:

3.

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

We follow these principles:

4.

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

We follow these principles:

5.

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

We follow these principles:

6.

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

We follow these principles:

7.

Working software is the primary measure of progress.

We follow these principles:

8.

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

We follow these principles:

9.

Continuous attention to technical excellence and good design enhances agility.

We follow these principles:

10.

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

We follow these principles:

11.

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

We follow these principles:

12.

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