The Ins and Outs of Agile Methods

Tom Smallwood

Lecture 28– CSCI 5828

04/22/2010

A little background

- Graduated with MESE 1995
- 25 years in development
- Variety of mostly small shops in Boulder area
- Agile development since late 90s
- Currently employed at Valtech
- Agile Transformation Coach
- tom@smallwood-software.com

Today's Purpose

- Free form and spontaneous
- Answer your questions about using Agile
- Supplement
- Typical challenges

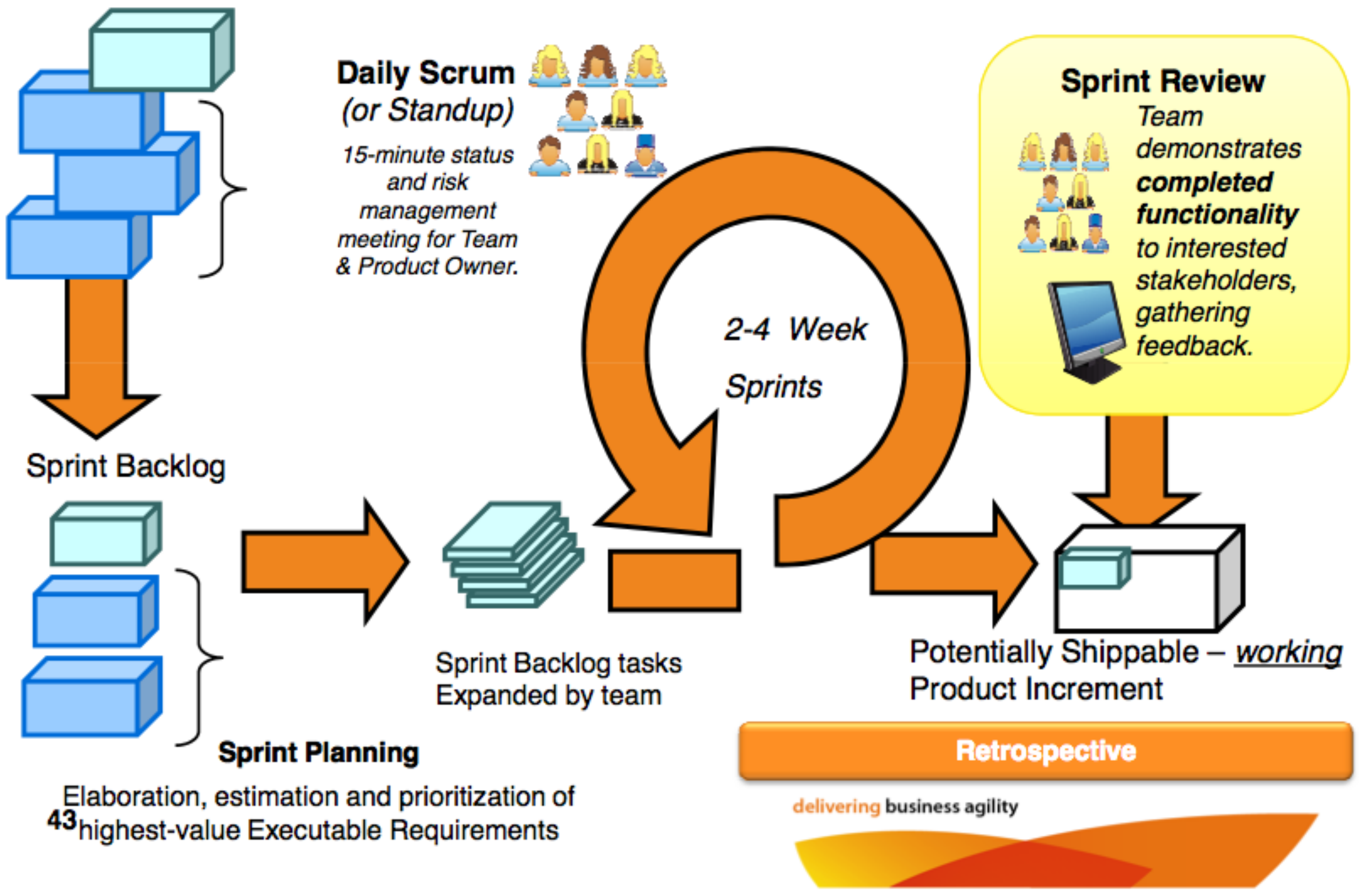
Your impressions

- How rigorous were you using agile?
- What worked well?
- Benefits that you experienced?
- Questions about Agile practices?

Agile Introduction

- The course provided a good foundation of Agile
- One of the course presentations provided a good description of Scrum

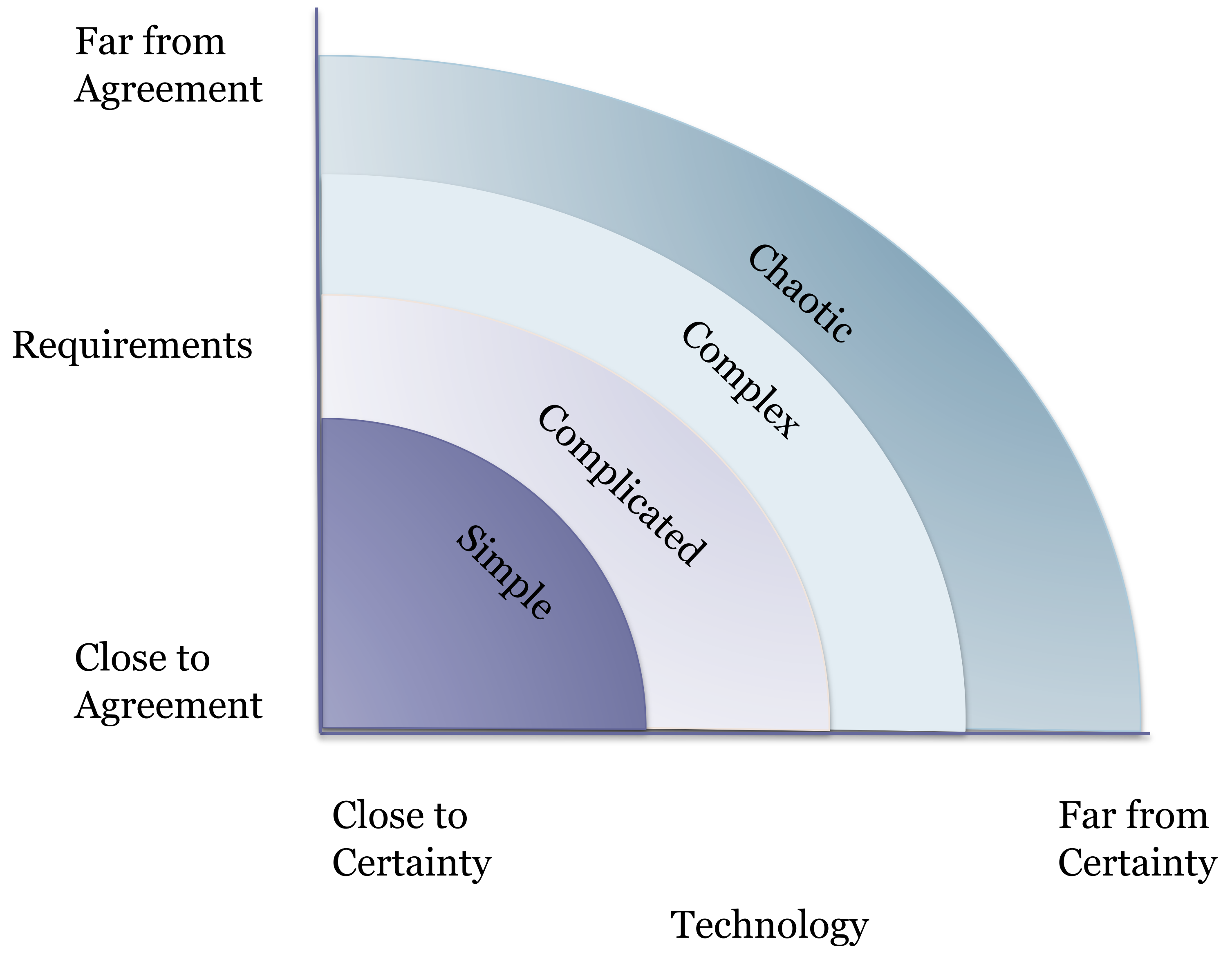
Scrum and Sprint Cycles



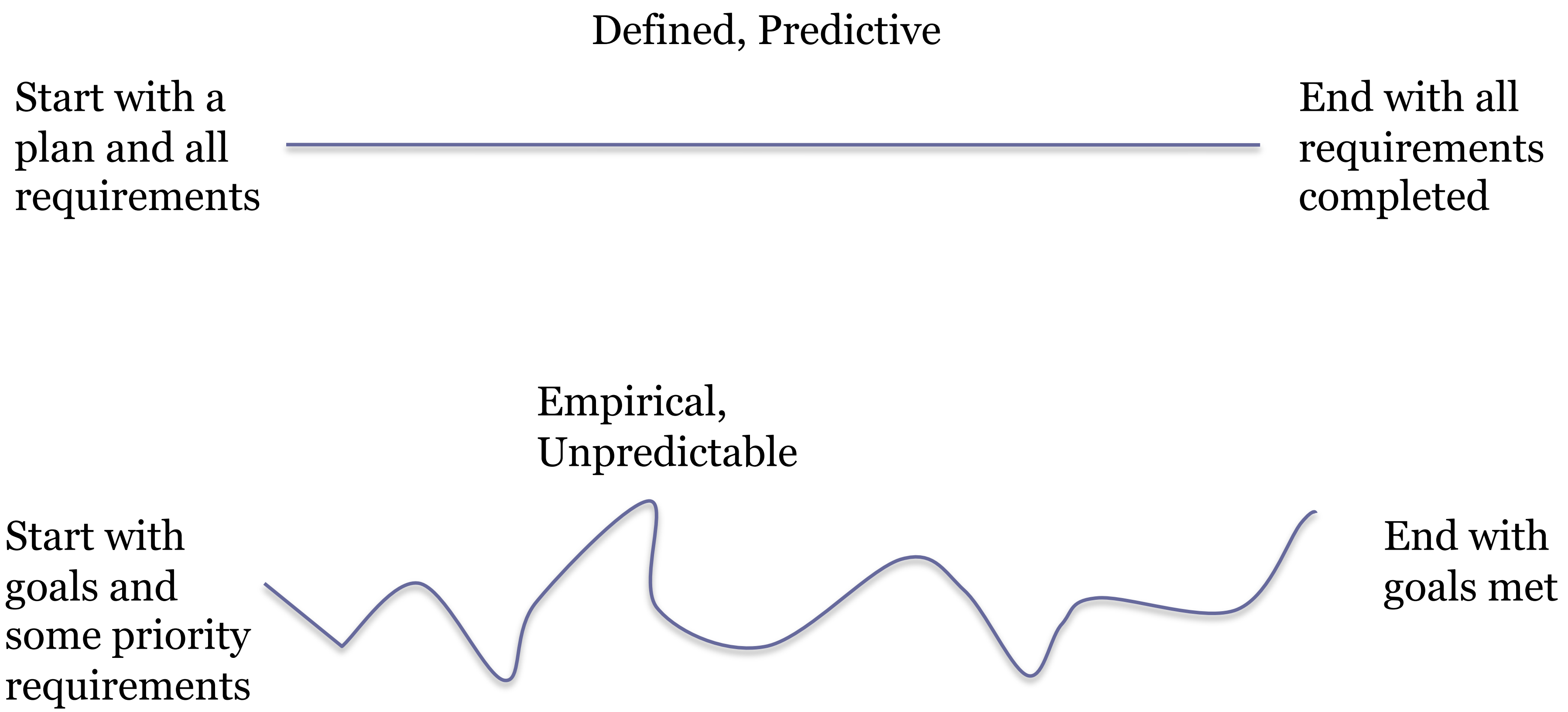
Product Backlog
As prioritized by *Informed* Product owner
Daily Scrum
(or Standup)
15-minute status and risk management meeting for Team & Product Owner.
Sprint Review
Team demonstrates **completed functionality** to interested stakeholders, gathering feedback.
2-4 Week
Sprints
Sprint Backlog
Sprint Backlog tasks
Expanded by team
Potentially Shippable – working
Product Increment
Sprint Planning
Retrospective
delivering business agility

Elaboration, estimation and prioritization of highest-value Executable Requirements

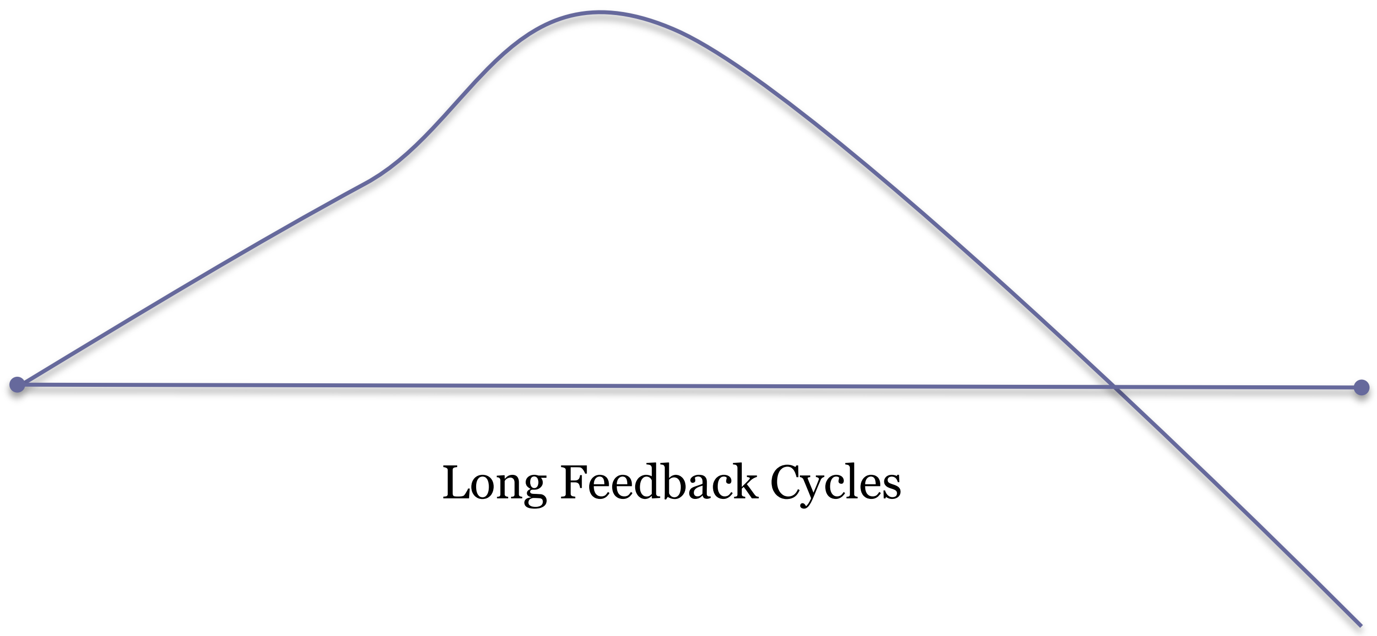
Why Iterative & Incremental


Far from Agreement
Requirements
Close to Agreement
Simple
Complicated
Complex
Chaotic
Close to Certainty
Far from Certainty
Technology

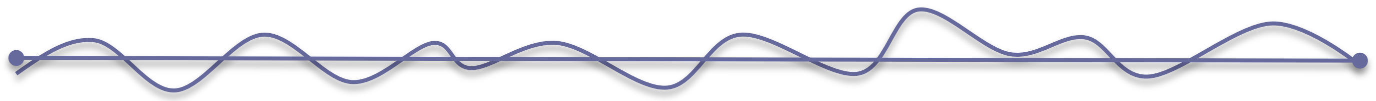
Chaotic	• High turbulence • No clear cause & effect • Unpredictable • Many decisions no time	• Immediate action to re-establish order • Prioritize and select actionable work • Look for what works rather than perfection • Act, sense, respond • Act on what is high priority and can be bounded
Complex	• More unpredictability than predictability • Emergent answers • Many competing ideas	• Create bounded environments • Increase level of interaction/ communication •Generate ideas • Probe, sense, respond • Servant leadership • Let people figure out the best way
Complicated	• More predictability than unpredictability • Fact-based management • Experts work out the wrinkles	• Utilize experts to gain insights • Use metrics to gain control • Sense, analyze, respond • Command & control
Simple	• Repeating patterns • Consistent events • Clear cause & effect • Well established knowns • Fact based management	• Use best practices • Extensive communication not necessary • Establish patterns and optimize them • Command & control


Defined, Predictive
Start with a plan and all requirements
End with all requirements completed
Empirical, Unpredictable
Start with goals and some priority requirements
End with goals met

Effect of Feedback Length

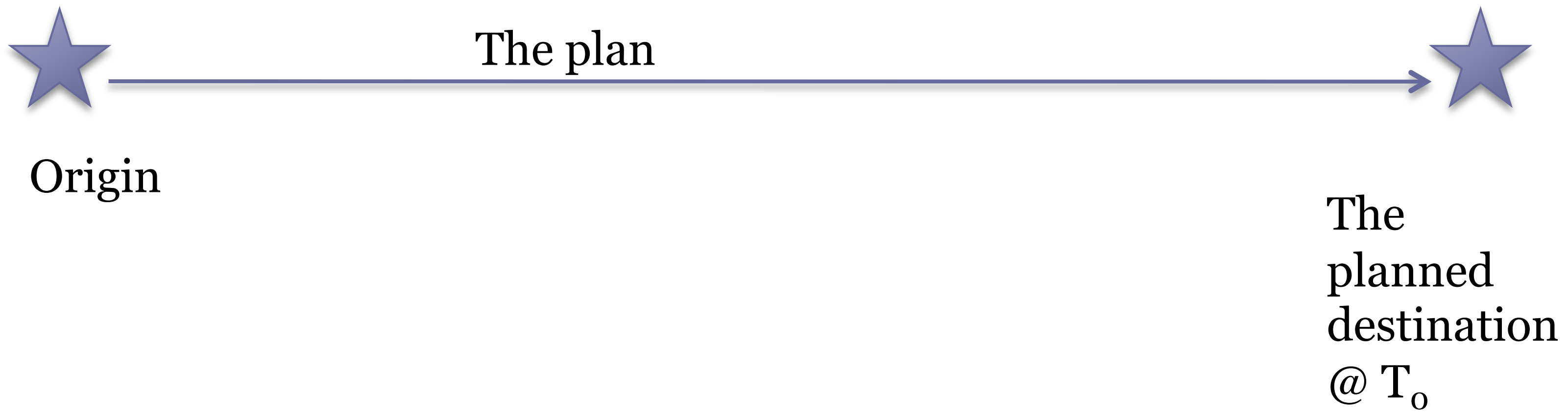


Long Feedback Cycles

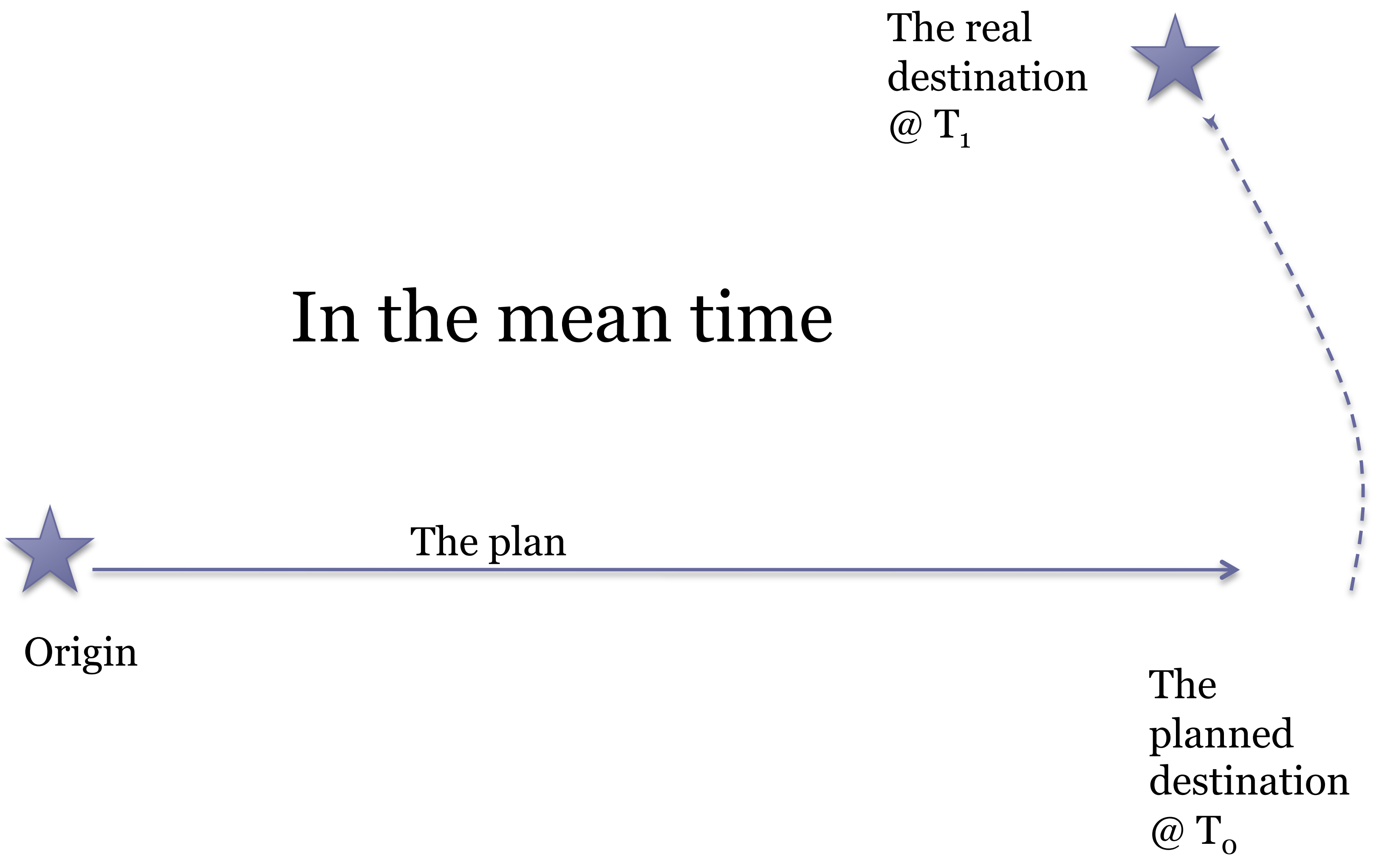


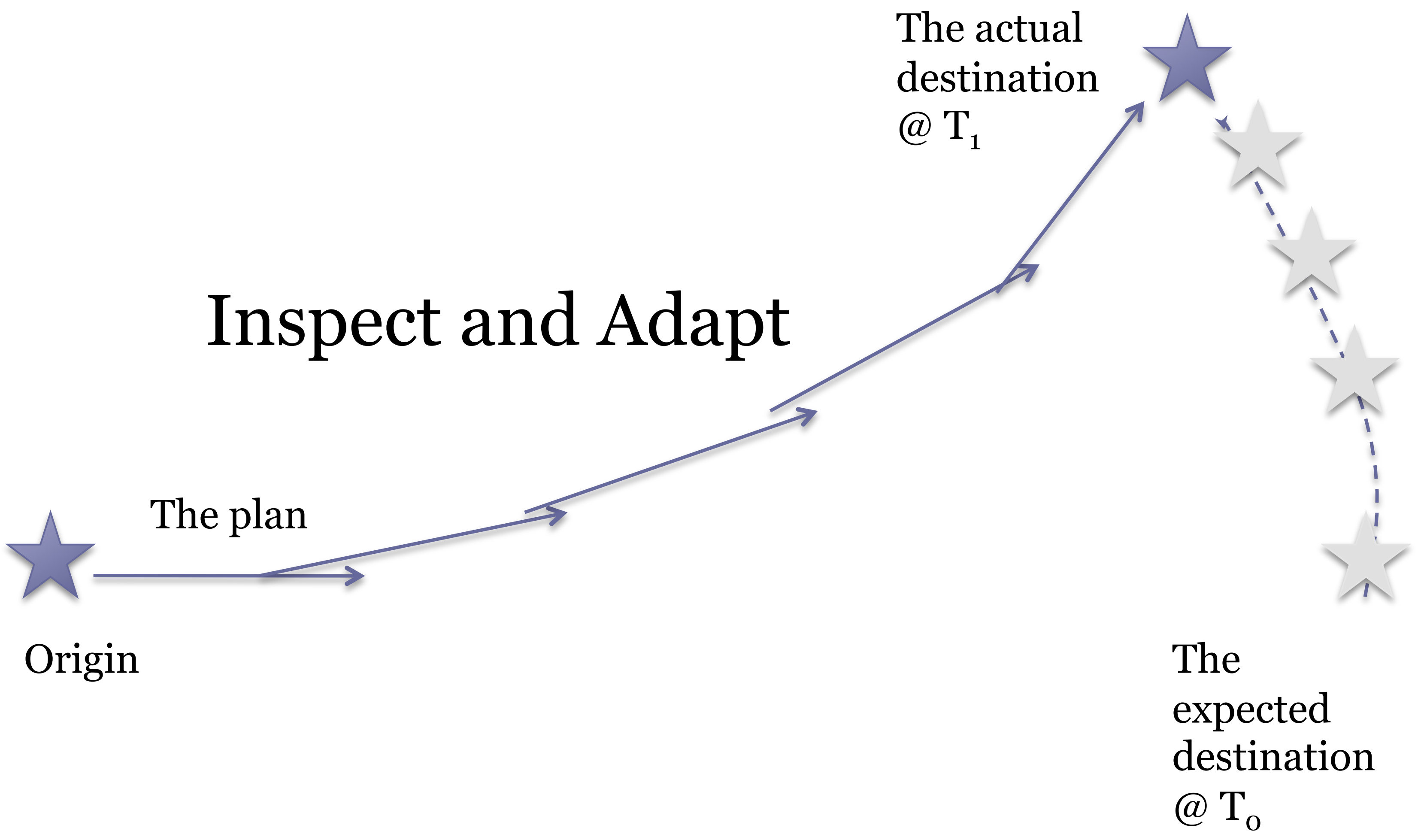
Short Feedback Cycles

Treating SW Dev as predictive



In the mean time

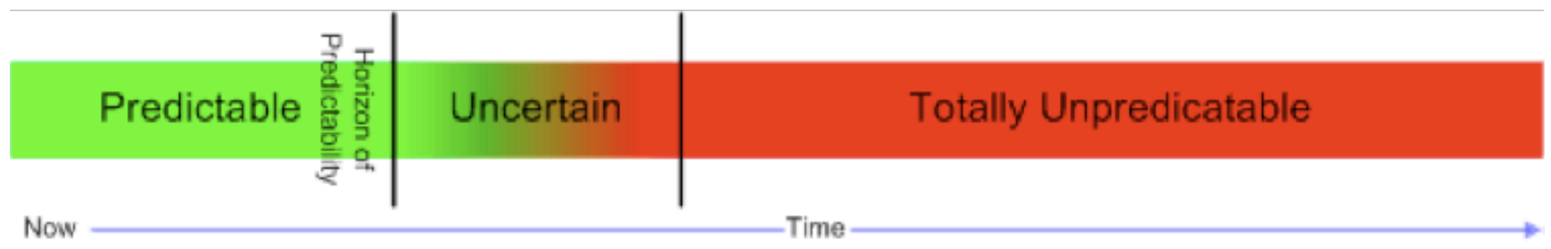



The actual destination @ T_1
Inspect and Adapt
The plan
Origin
The expected destination @ T_0

What we learn

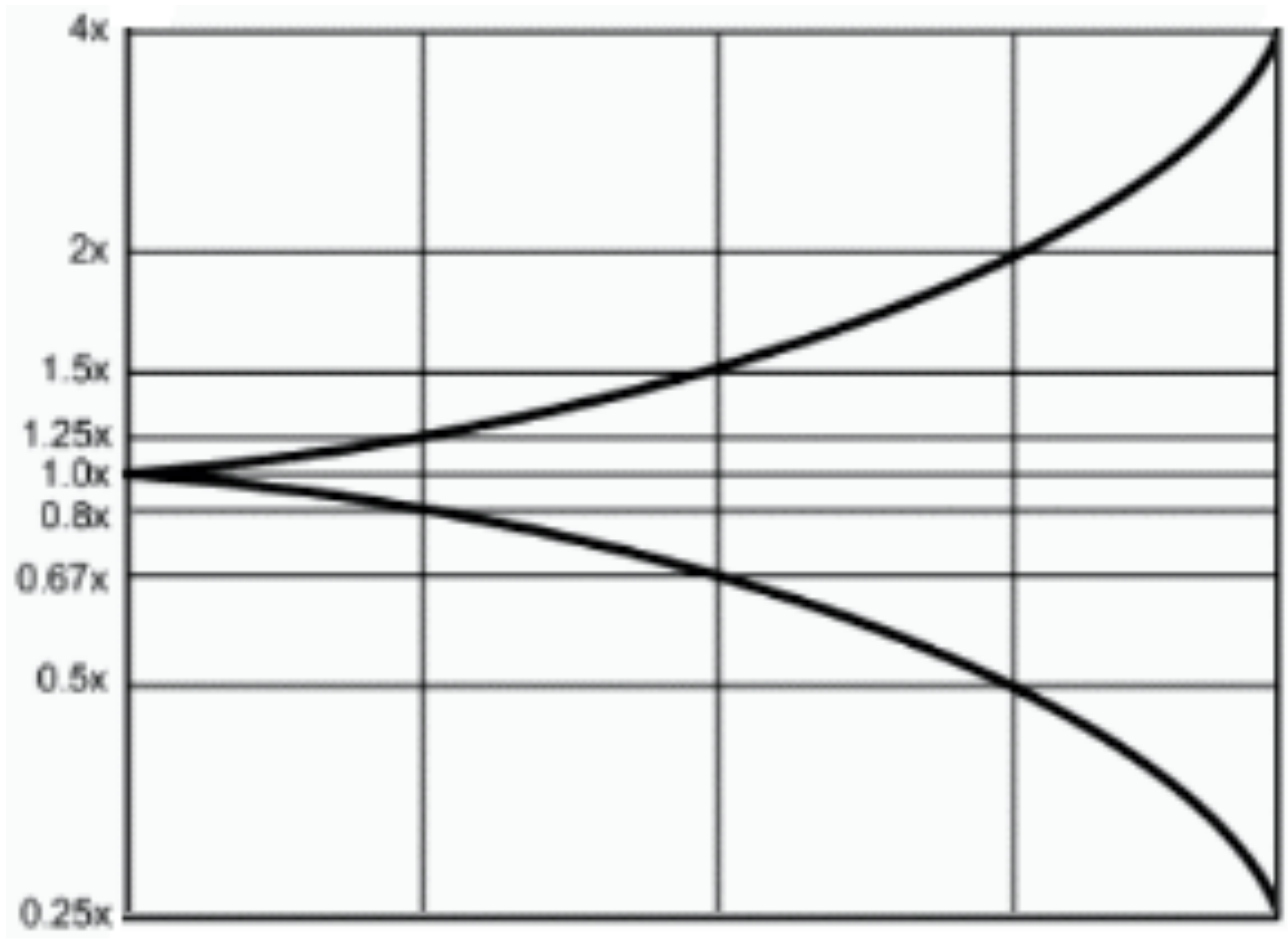
- Plans change for a variety of reasons
- Expect them to change
- SW is novel and complex
- Use a process that allows for change
- Planning is done at different levels
- Inspect and adapt

Horizon of Predictability



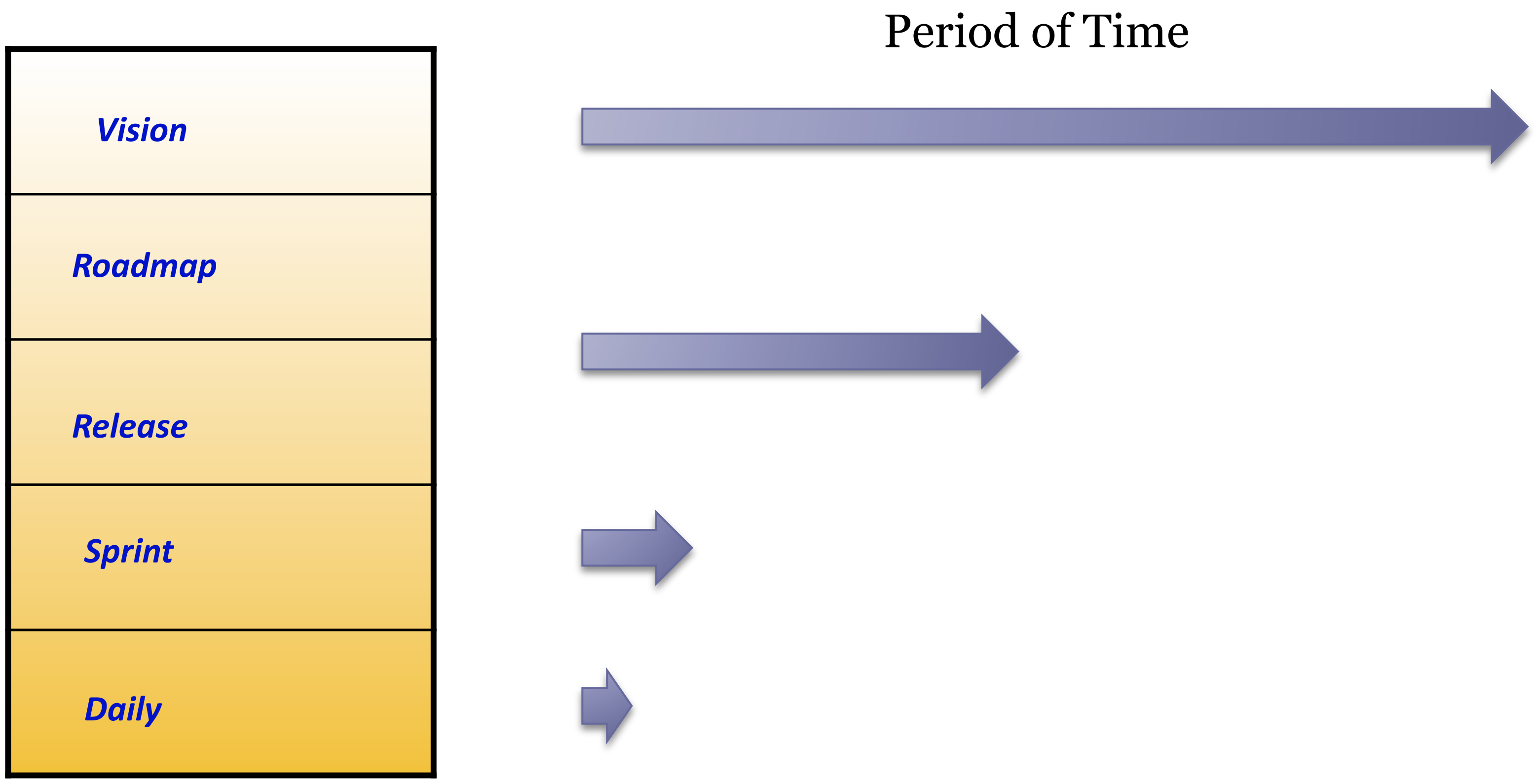
Cone of Uncertainty

Now

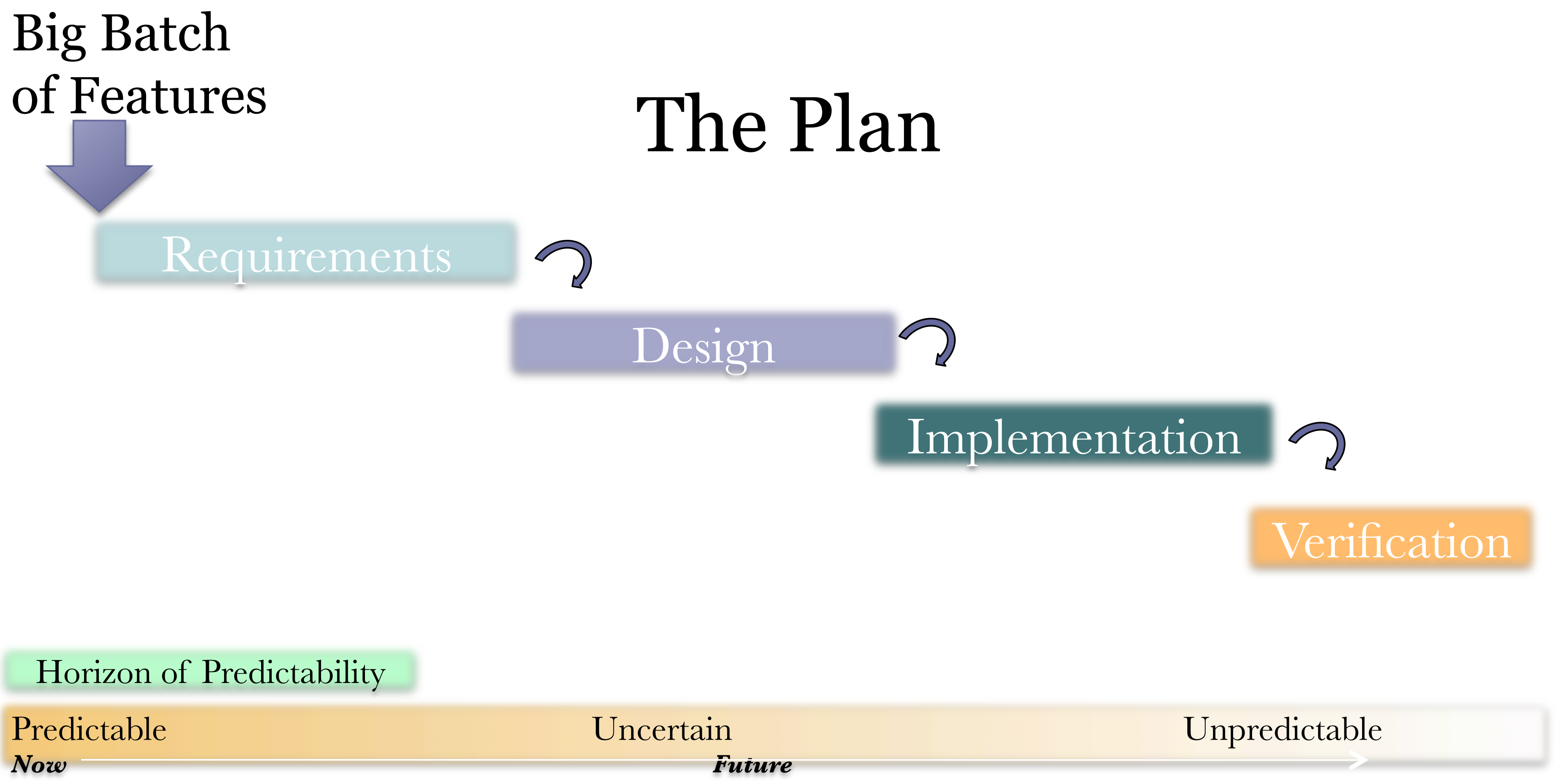


Future

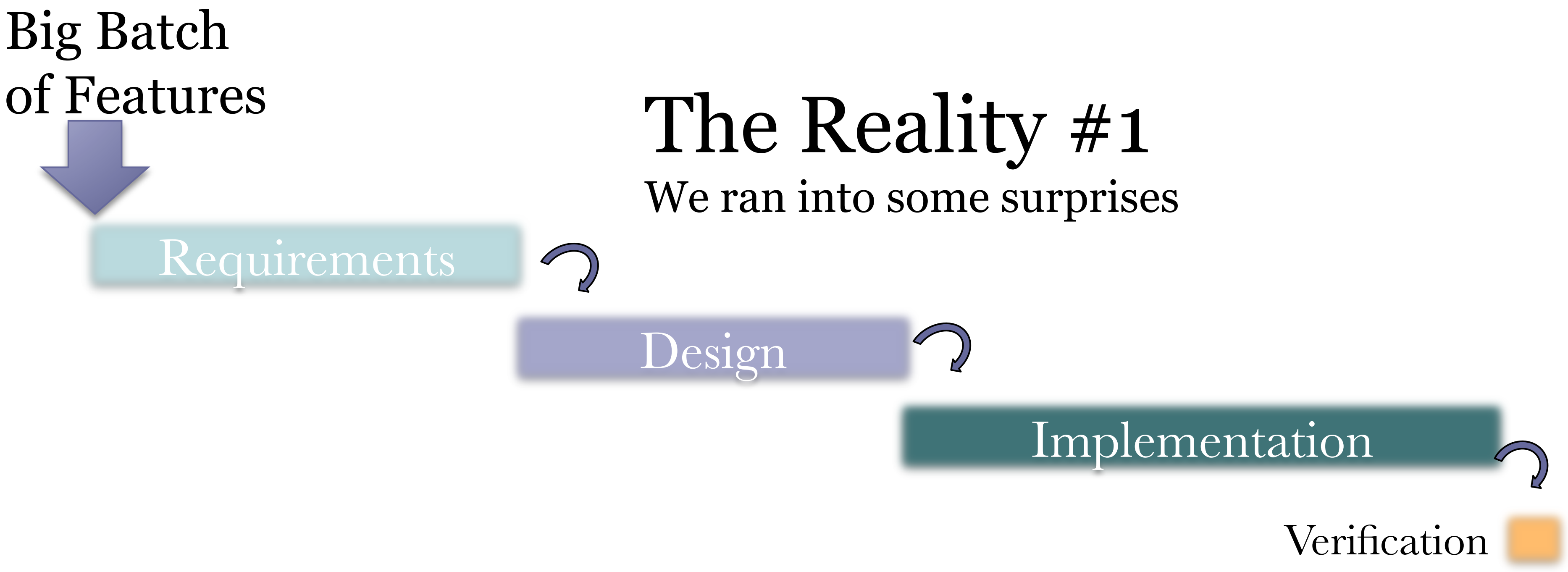
Levels of Planning



The Problem With “Waterfall”



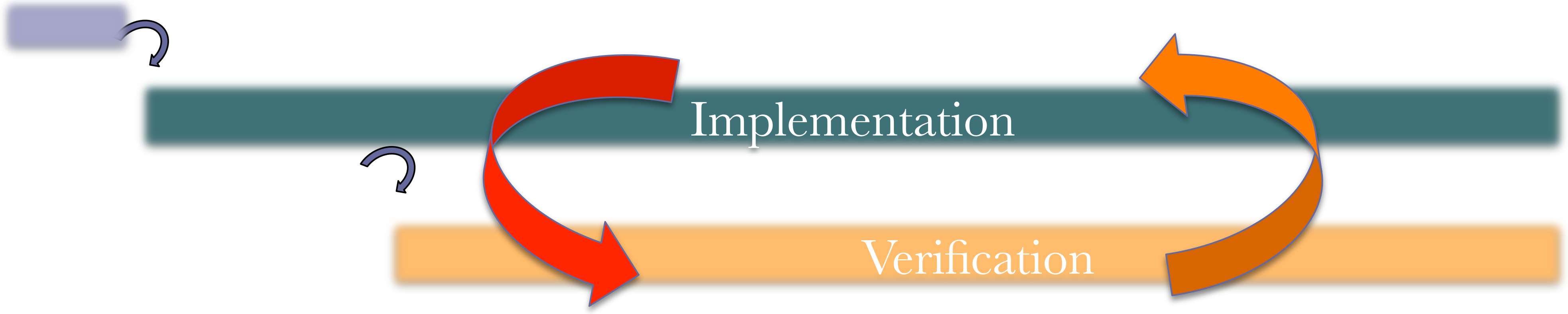
It never quite works out



The Problem With “Waterfall”

The Reality #2

We had to meet our schedule



Test and Fix
Aka “The circle of hell”

The Problem With “Waterfall”

The Reality #3

This took longer than expected

Implementation

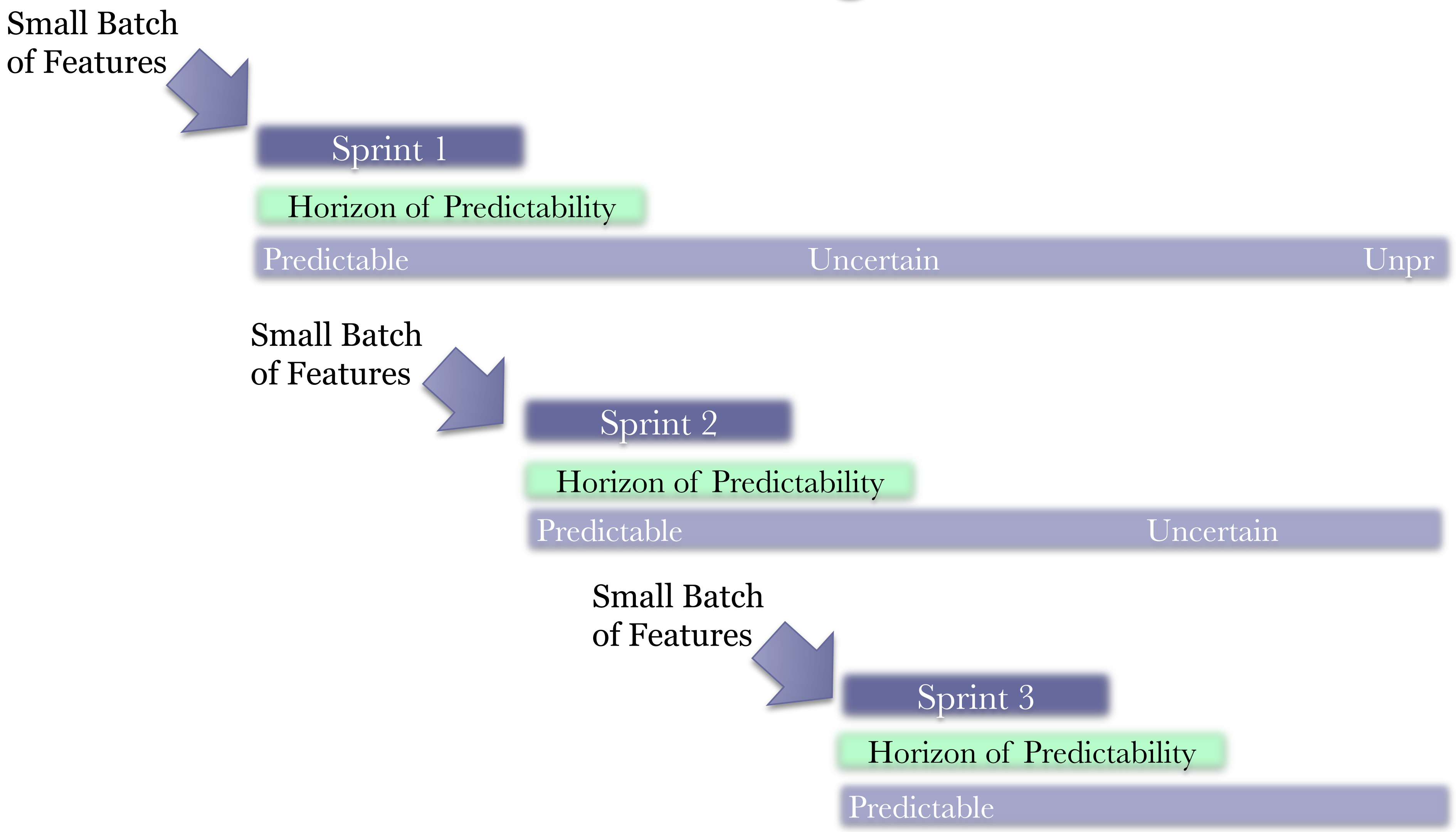
And other nightmares

- late integration of component and system
- untested deployments
- lack of production-like environments
- production find and fix
- manual regression testing
- death marches
- burnout
- divorce
- people quit

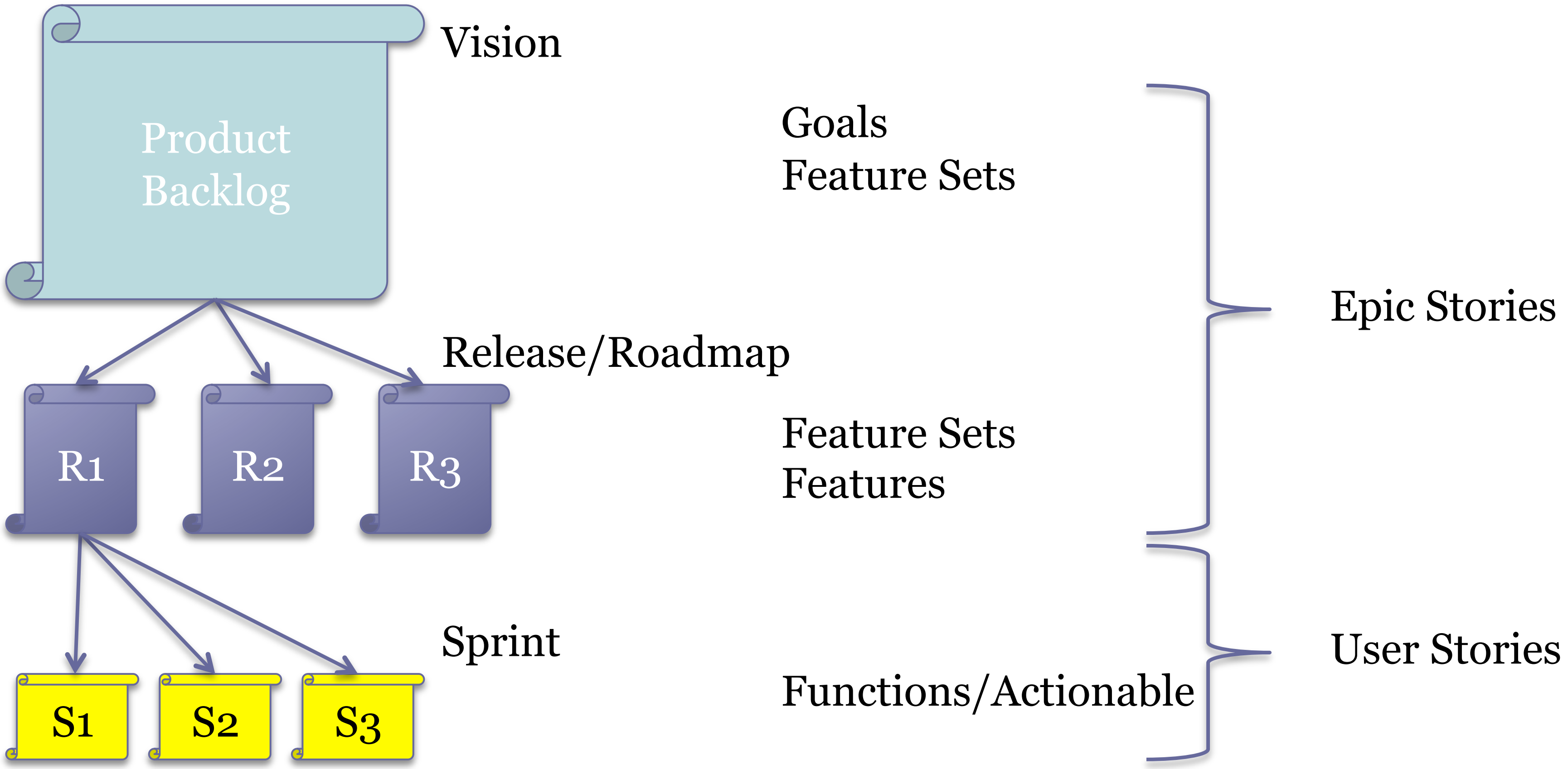
What we learn

- Small plans are less complex than big plans
- Variance exists no matter how well you plan
- Death marches are no fun
- Predict future progress by past progress
- Excessive designing results in bloat
- Building it proves it
- Complex systems emerge from simple systems

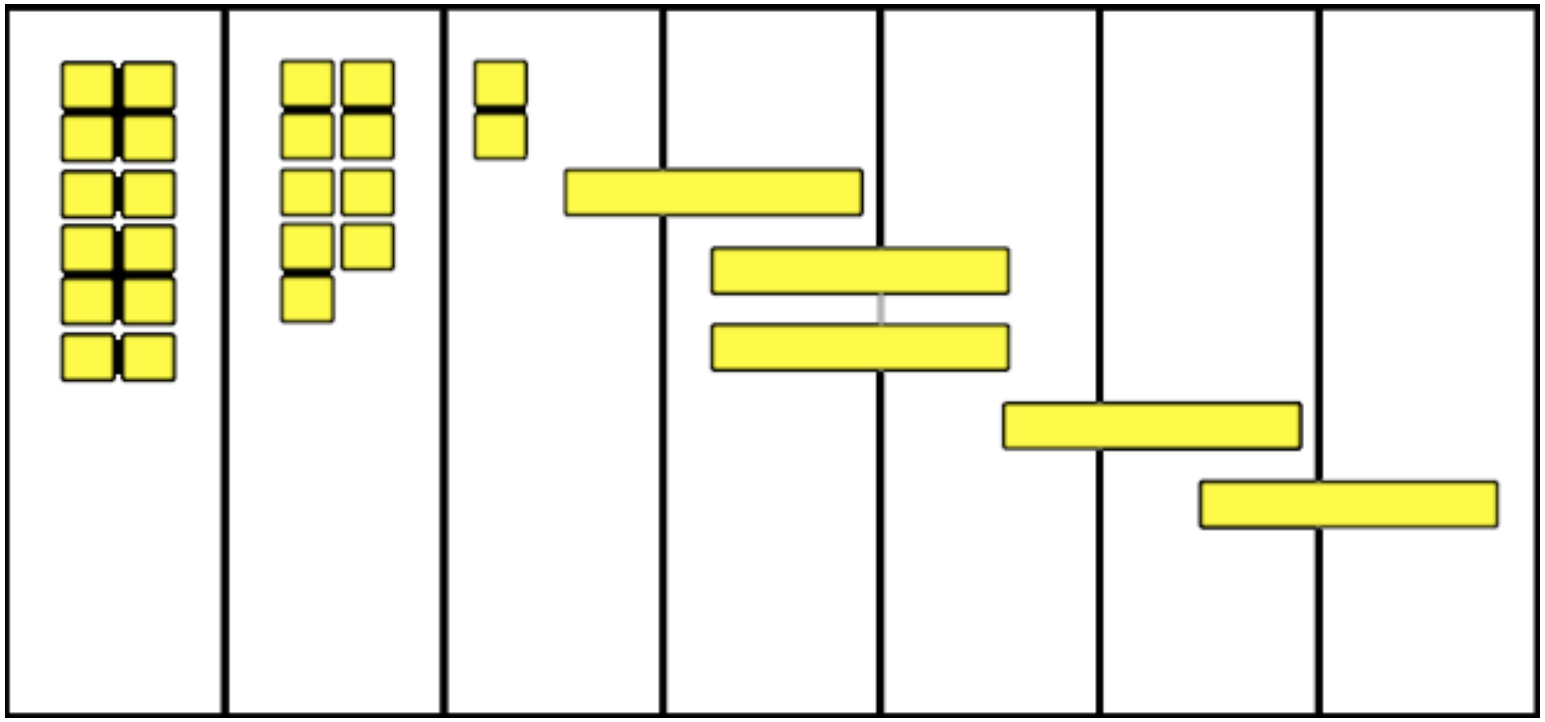
And How Does Agile Work?



Backlogs that Support Agile Planning



Abide by the horizon of predictability



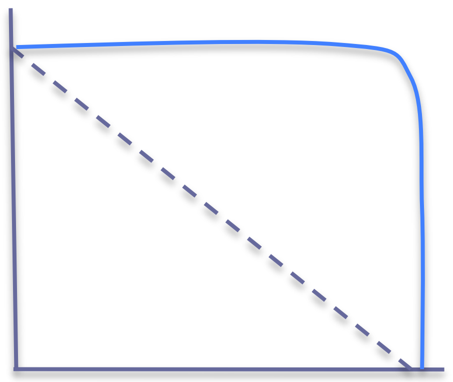
What we learn

- Uncover details to the level that is responsible
- Delay decisions until the last responsible moment
- Don’t do work until its needed (JIT)
- JIT Requirements

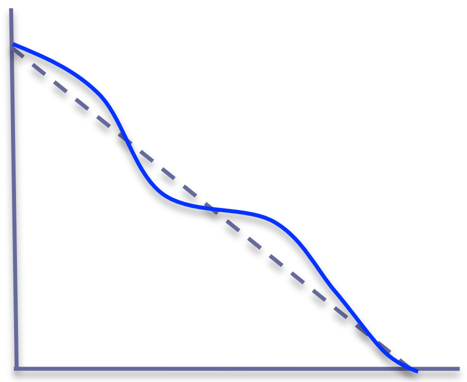
Right-sizing stories - INVEST

- I – independent
- N – negotionable
- V – valuable
- E – estimatable
- S – small
- T - testable

Right-sizing stories



Stories too big

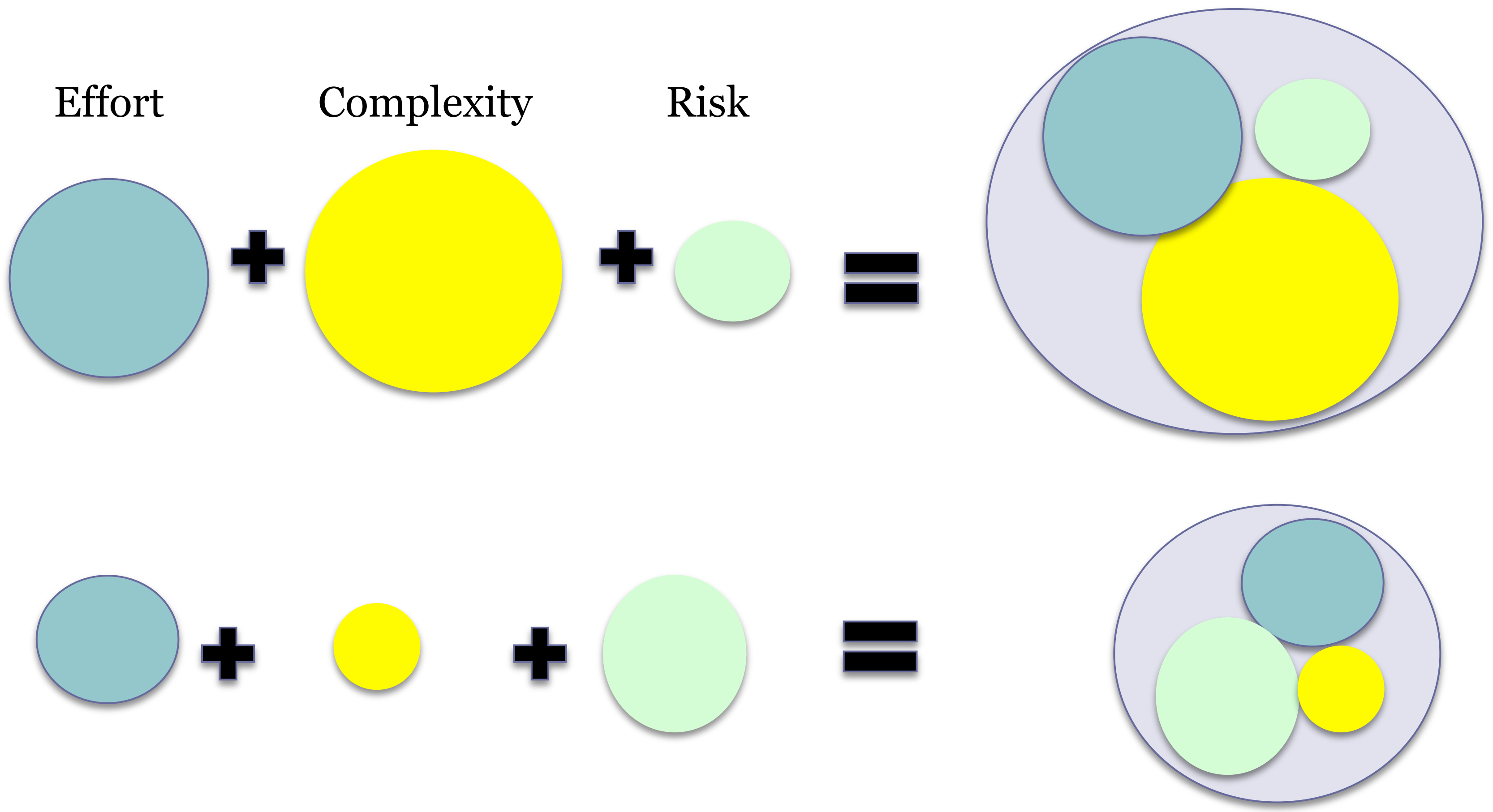


Stories small so steady progress is made

Estimating with story points

A Story Point is…

Effort + Complexity + Risk



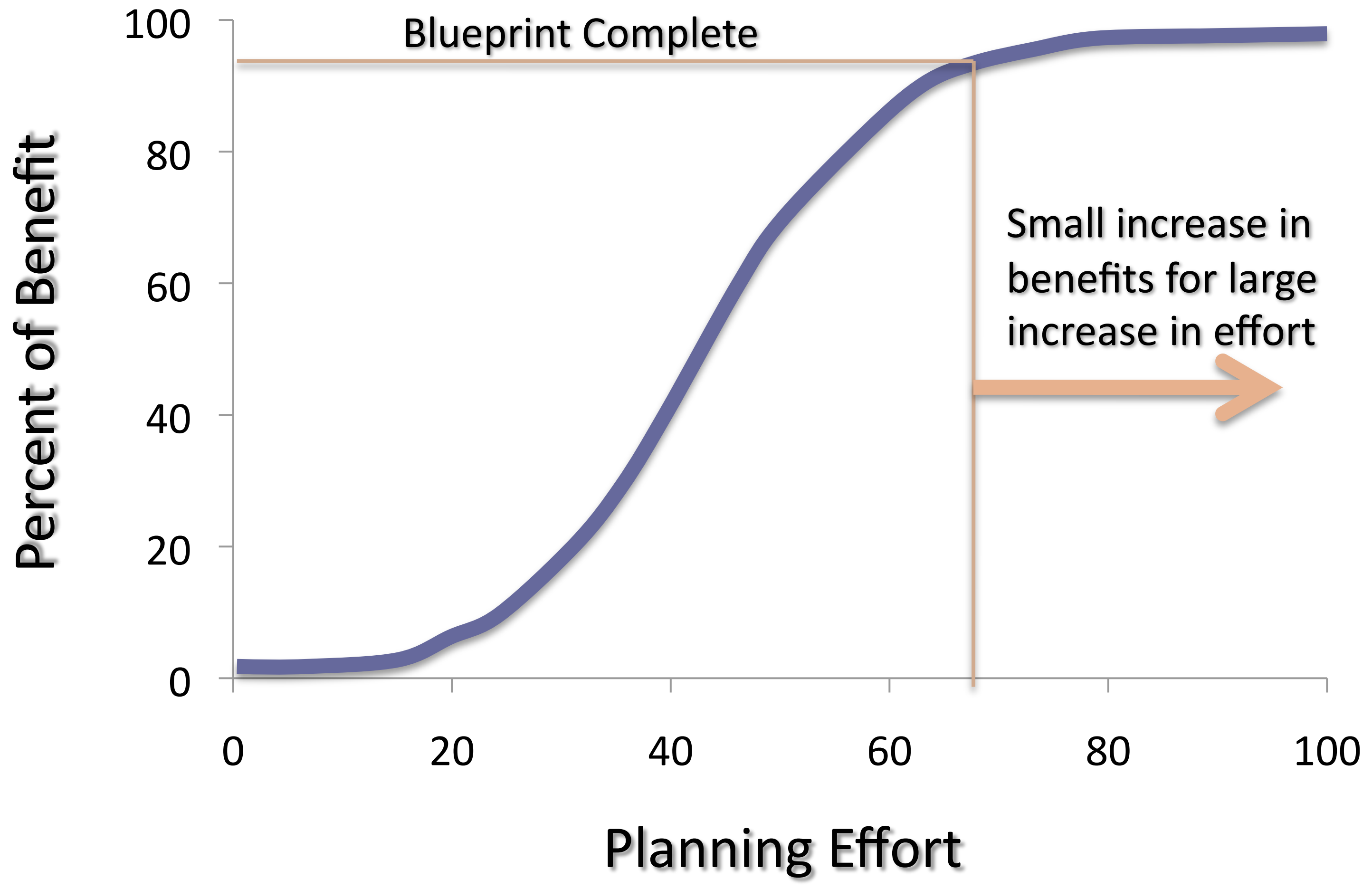
valtech

Planning Poker

Why Does Planning Poker Work?

- Relative sizing has proven to be very accurate
- Minimal effort yields big results
- Multiple expert opinions during estimation
- People doing the estimates are the ones who will be doing the work
- Estimators justify estimates
- Averaging individual estimates leads to better estimates overall
- Extreme collaboration occurs
- Provides estimates that are accurate enough to get started.
- Estimates remain constant
- Team velocity (average story points per iteration)

How Much Estimating is Enough?



Challenges of Agile

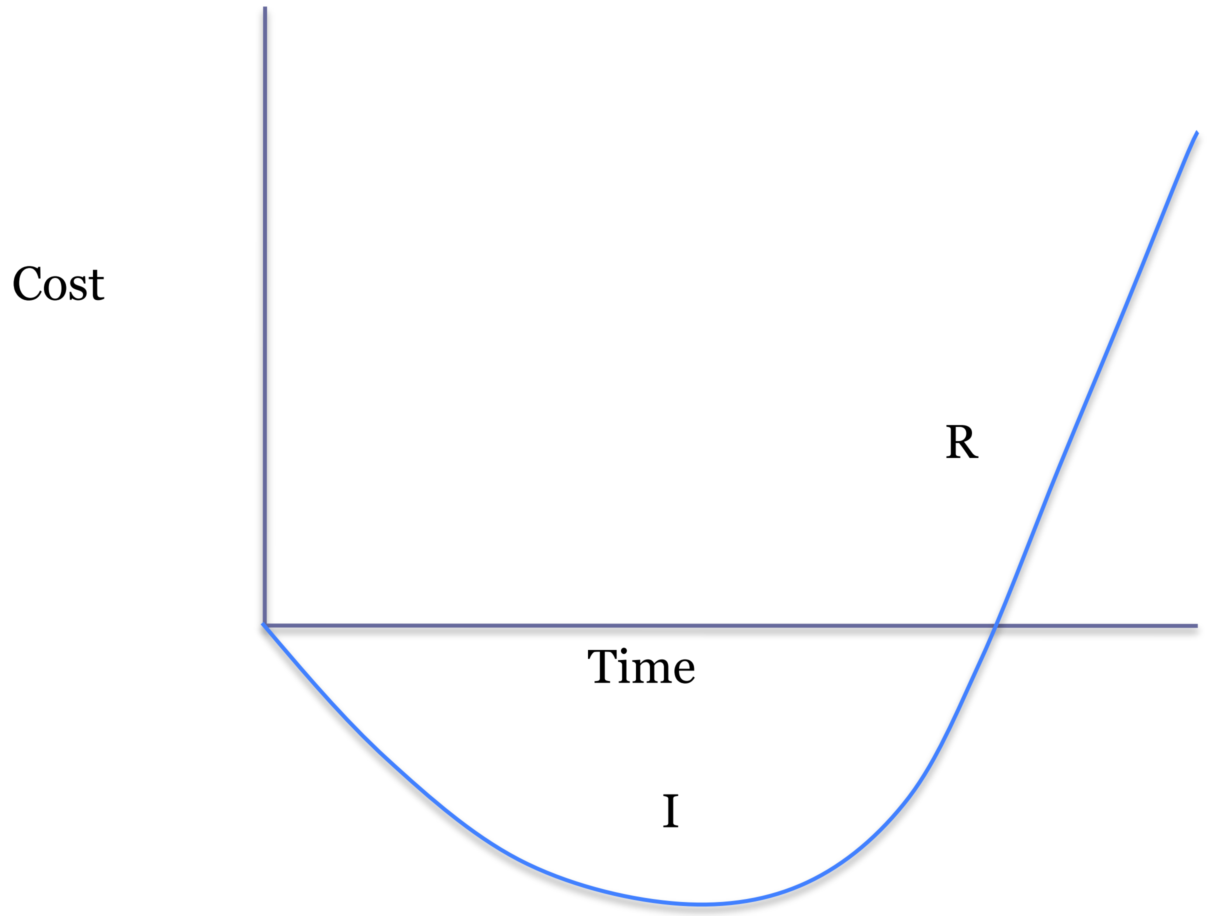
- Still considered new-fangled
- Requires discipline -- very few companies have it
- Agile failures blamed on Agile
- Most are looking for a process to follow not a new way of thinking
- Waterfall behaviors are difficult to overcome
- Agile requires cultural change – this is hard
- Teams are empowered, leadership serves
- Making the entire value stream agile

What Agile means to you

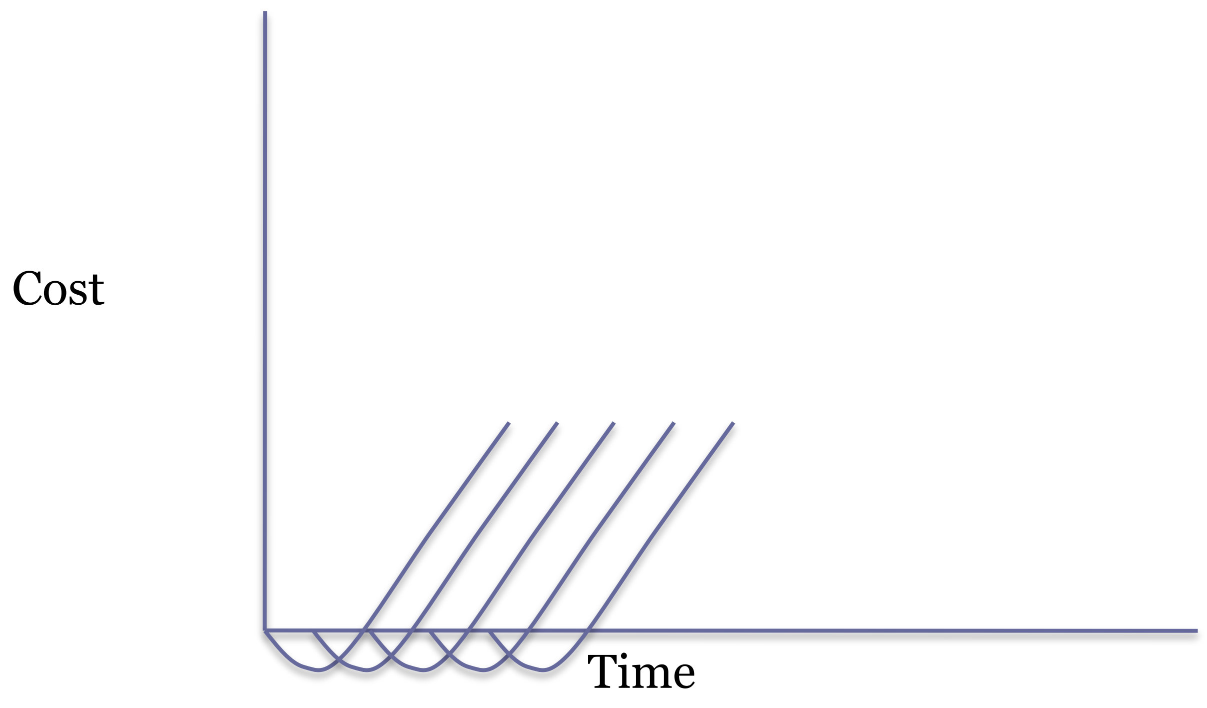
- Agile testing – very few companies do it. There is a big need for people that know how to do it.
- Make TDD your development methodology
- Be skilled in multiple disciplines
- Learn Agile/Lean – this is bound to be with us for a very long time. 2000s Agile was used in small shops. 2010s large companies are now adopting Agile.

Other musings

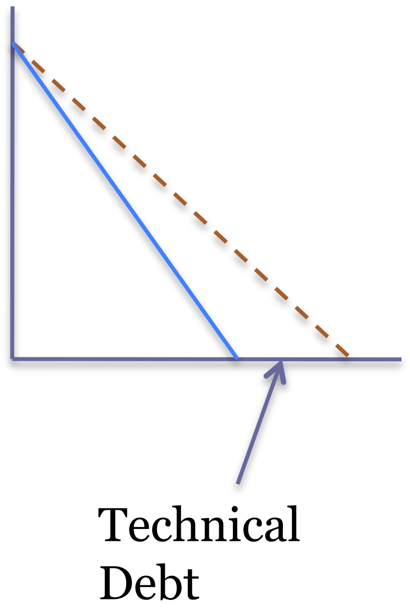
ROI - Waterfall

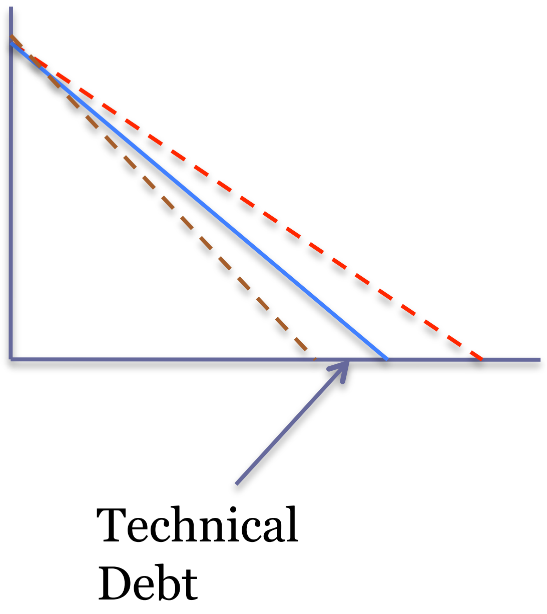


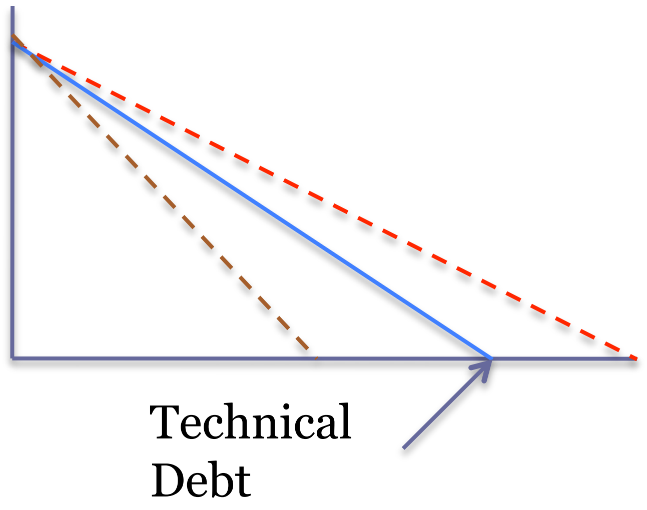
ROI - Iterative



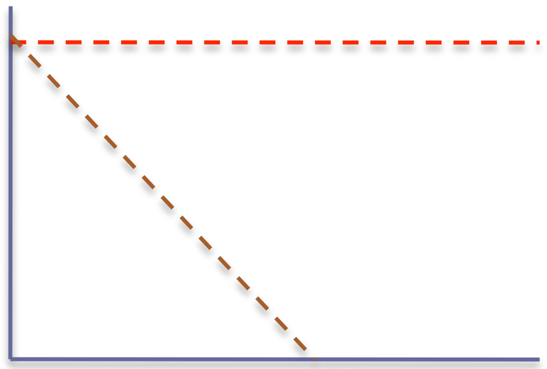
Technical Debt







Over time, technical debt accumulates, System must be rewritten



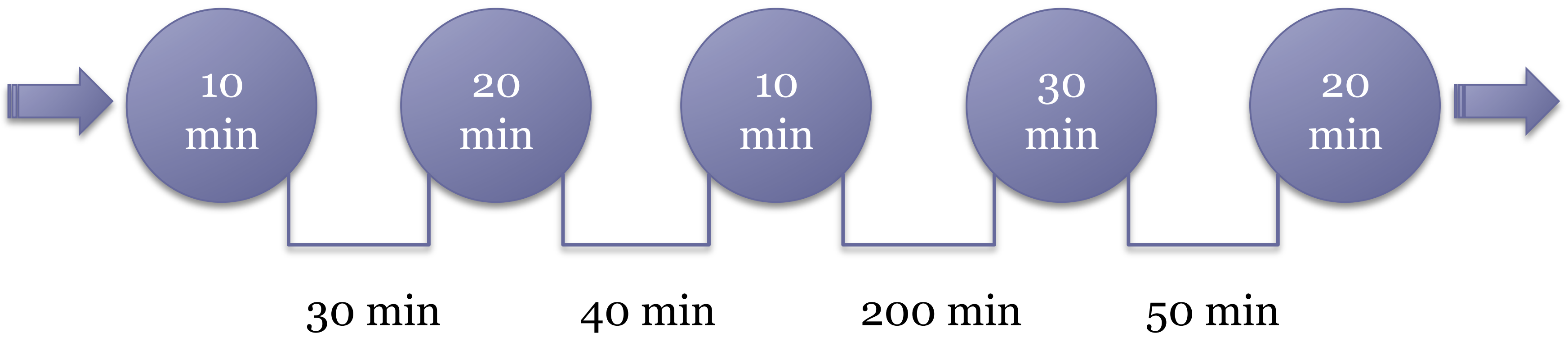
Lean SW Development

Lean Says…

- Add nothing but value
  - Center on people who add value
  - Flow value from demand
  - Optimize across organizations

Add nothing but value

The Value Stream

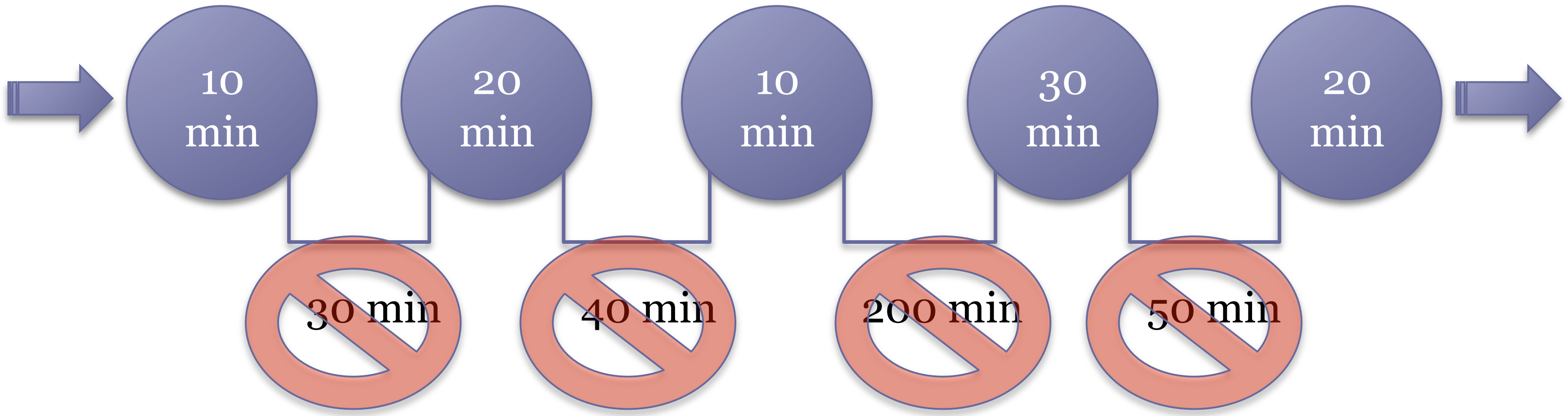


Time Worked		90	
-------------- =		-------------- =	20%
Efficiency		Efficient	
Cycle Time		410	

Drive out waste

The first step in Lean thinking is learning to see waste and remove it.

Don’t try to improve the “value add” steps – at least initially



Seven Wastes of Manufacturing

Overproduction
Inventory
Extra Processing Steps
Motion
Defects
Waiting
Transportation

Seven Wastes of SW Development

Overproduction
Inventory
Extra Processing Steps
Motion
Defects
Waiting
Transportation

Overproduction

Extra (Unused) Features
Gold Plating
Un-integrated code
Untested Code
Un-deployed Code

Develop only for today's stories
Don't build "for the ages"
YAGNI

Inventory

Work in Progress
(All work in progress is potential waste)

Prematurely specified details
Partially completed stories
Un-integrated code
Untested code
Un-deployed code

Delay work until it is needed and can be completed (i.e., JIT)
Minimize WIP

Extra Steps

Inefficient Process
Manual Operations
Excessive Formality
Unnecessary Paperwork
Handoffs
Complex communication methods
Doing more than is necessary

Face-to-face communication
Do the simplest thing possible

Motion

Finding and Re-finding Information
Relearning
Long feed-back loops
Distributed teams
High-effort communication
Handoffs
Jerky and interrupted flow

Keep communication costs (effort) cheap
Cross-functional and co-located teams
Smooth flow

Defects

Defects not caught by tests
Unclear acceptance criteria
Handoffs
Long feed-back loops
No Product Owner

Keep defects out of the code!
Use TDD, SDD, Executable Requirements
Automated testing of all types
“Stop the line” mentality
Mistake-proof anything and everything

Waiting

Waiting
Distributed teams
Multi-tasking
Organizational Silos
Product Owner not available
Long feed-back loops
Handoffs

Teams make critical decisions every 15 minutes
Cross-functional teams
Co-located teams
Highly available Product Owner

Transportation

Handoffs
Managing/Maintaining Premature Details
Managing large backlogs and bug lists
Product Owner (customer) not available to team

Every time information is transferred to another group or person knowledge is usually lost (and waiting is usually introduced)
Follow JIT Principles
Clean House

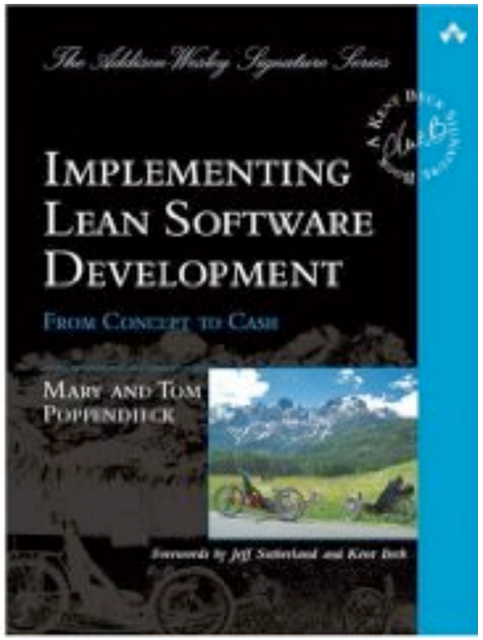
Thinking Lean

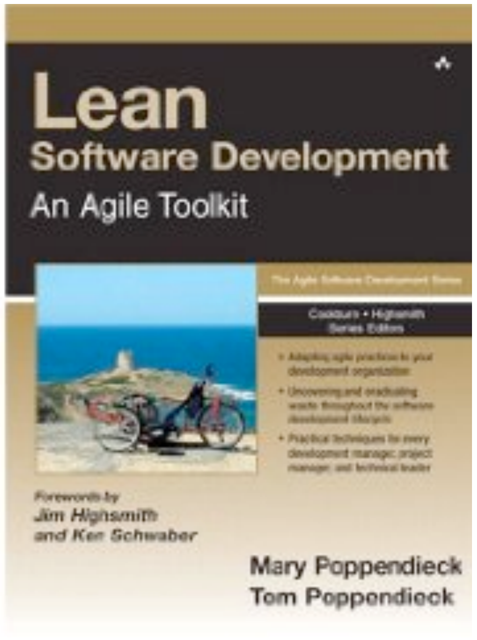
If something does not directly
add value, it is waste.

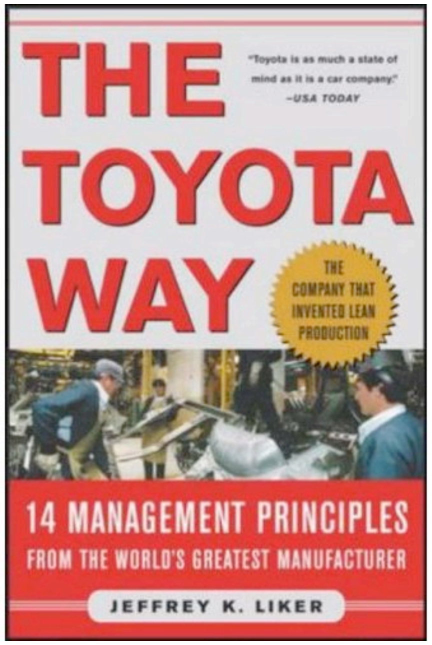
If there is a way to do without it,
it is waste.

Speed is the absence of waste

Resources


The Addison-Wesley Signature Series
A Kent Beck Signature Book
Implementing Lean Software Development
From Concept to Cash
Mary and Tom Poppendieck
Forewords by Jeff Sutherland and Kent Beck


Lean
Software Development
An Agile Toolkit
The Agile Software Development Series
Cockburn • Highsmith
Series Editors
Forewords by
Jim Highsmith
and Ken Schwaber
Mary Poppendieck
Tom Poppendieck


The Toyota Way
"Toyota is as much a state of mind as it is a car company." —USA TODAY
The company that invented lean production
14 Management Principles
from the World's Greatest Manufacturer
Jeffrey K. Liker