www.odd-e.com | basv@odd-e.com

Agile & Scrum

A very short introduction

Who am I?

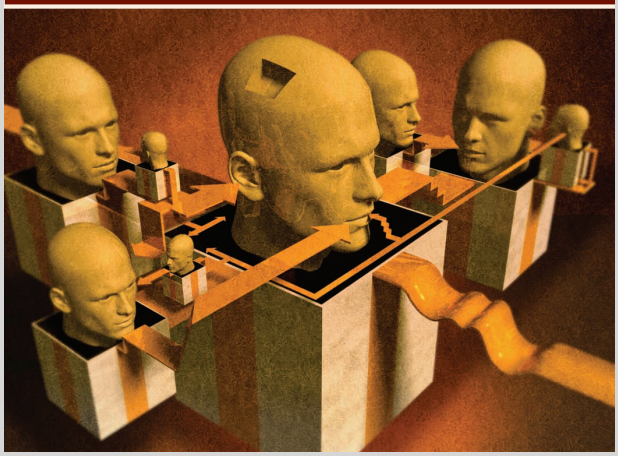
- Name: Bas Vodde
- Originally from Holland
- Lives in Singapore
  - Lived in China and Finland
- Works for Odd-e
- Agile coach, SW developer
- Led Agile transformation program in large company
- Experience with large embedded products

Odd-e

Scaling Lean & Agile Development

Thinking and Organizational Tools for Large-Scale Scrum

Craig Larman
Bas Vodde



Practices for Scaling Lean & Agile Development

Large, Multisite, and Offshore Products with Large-Scale Scrum

Craig Larman
Bas Vodde

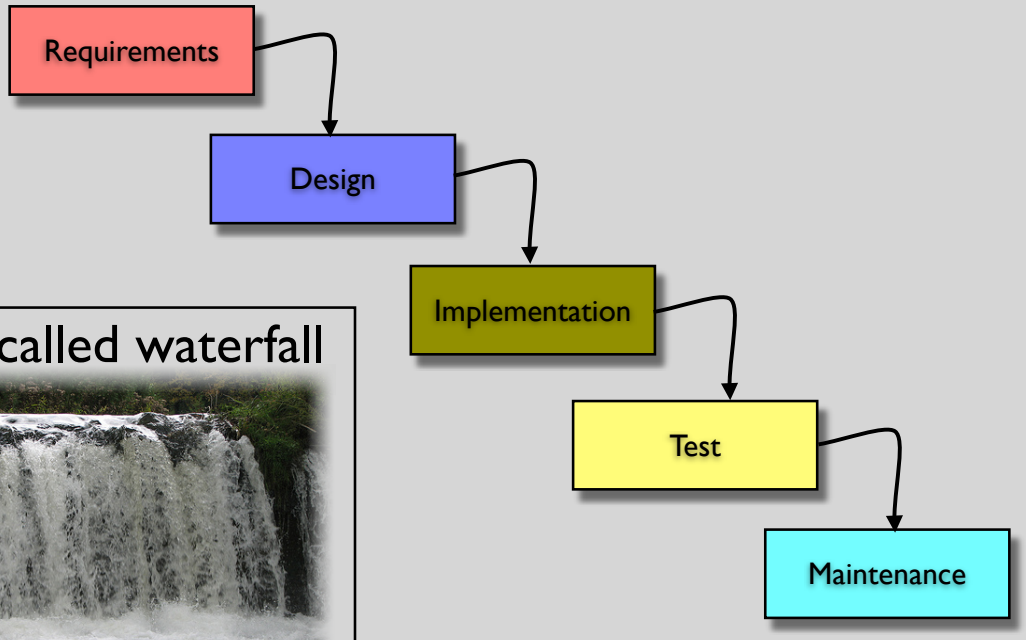
Odd-e

Short fuse!

Agenda

- Problems with Sequential Development
- Why Agile?
- Iterative development
- Scrum & Agile Overview
- Core Concepts
- More Scrum
- Misconceptions
- Additional backup material

Traditional Development



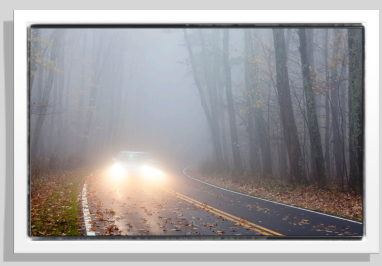
Also called waterfall

Problems with sequential development

Late "real" feedback

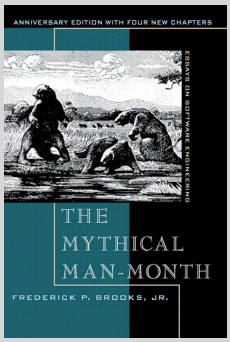
Difficult to deal with changes

Slow - Handovers

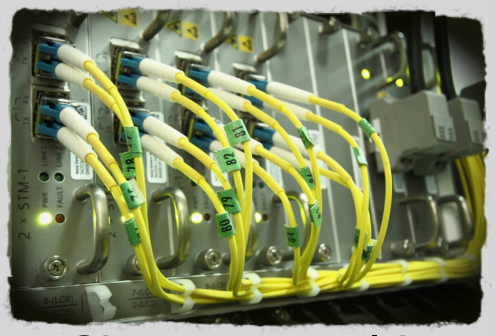


Lack of visibility

Known for years



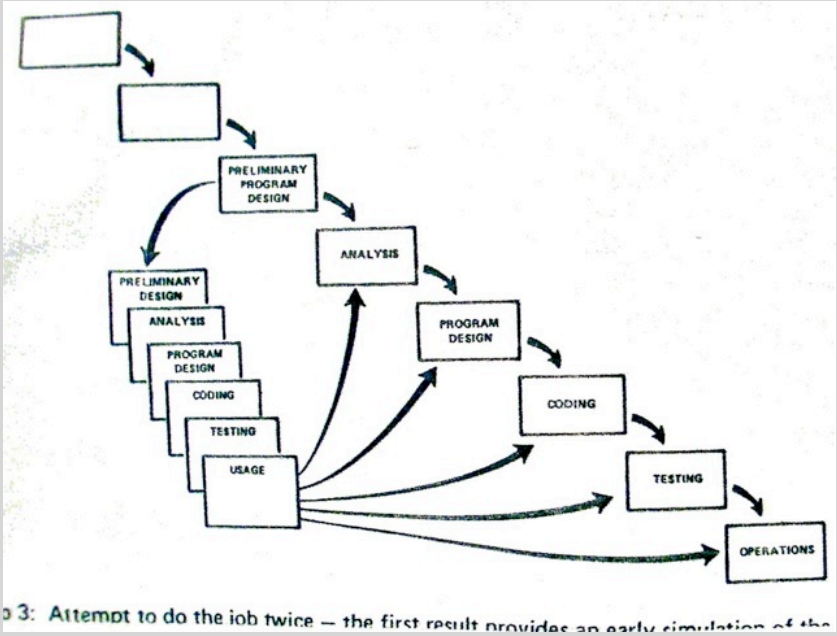
Mythical Man-month



Changing world

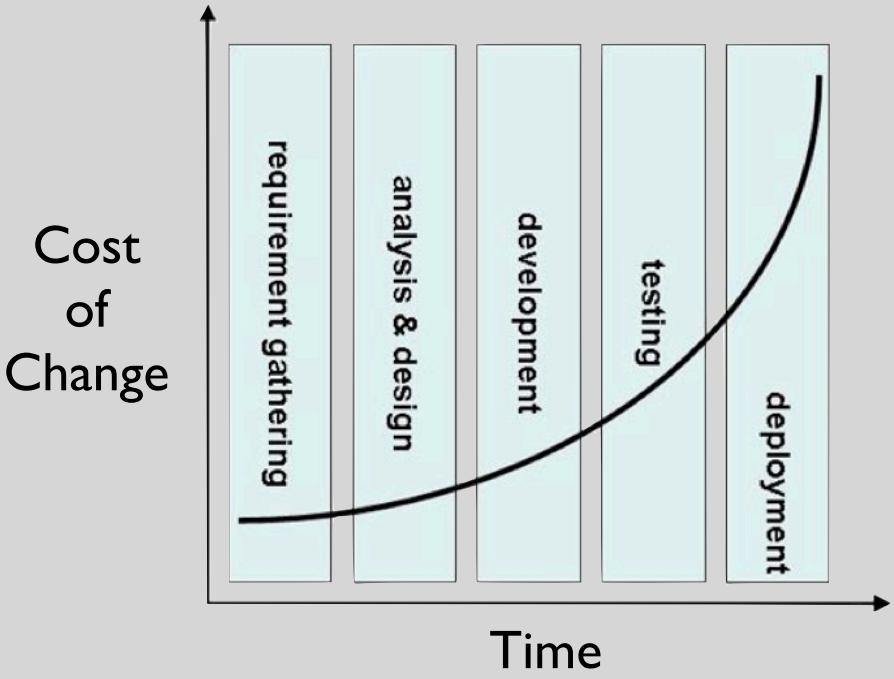
"software crisis"

Did it ever work?

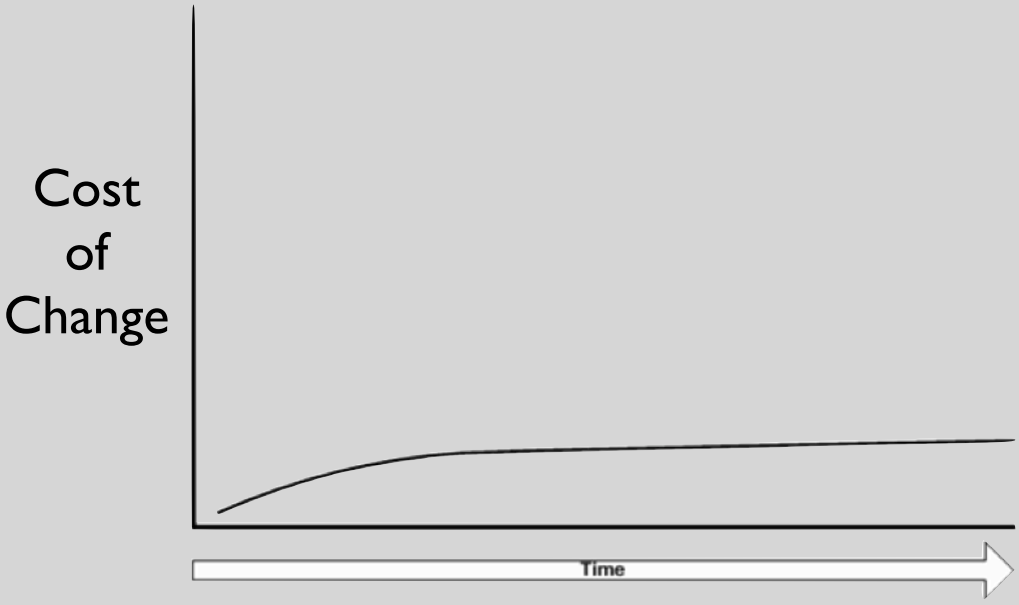


3: Attempt to do the job twice – the first result provides an early simulation of the

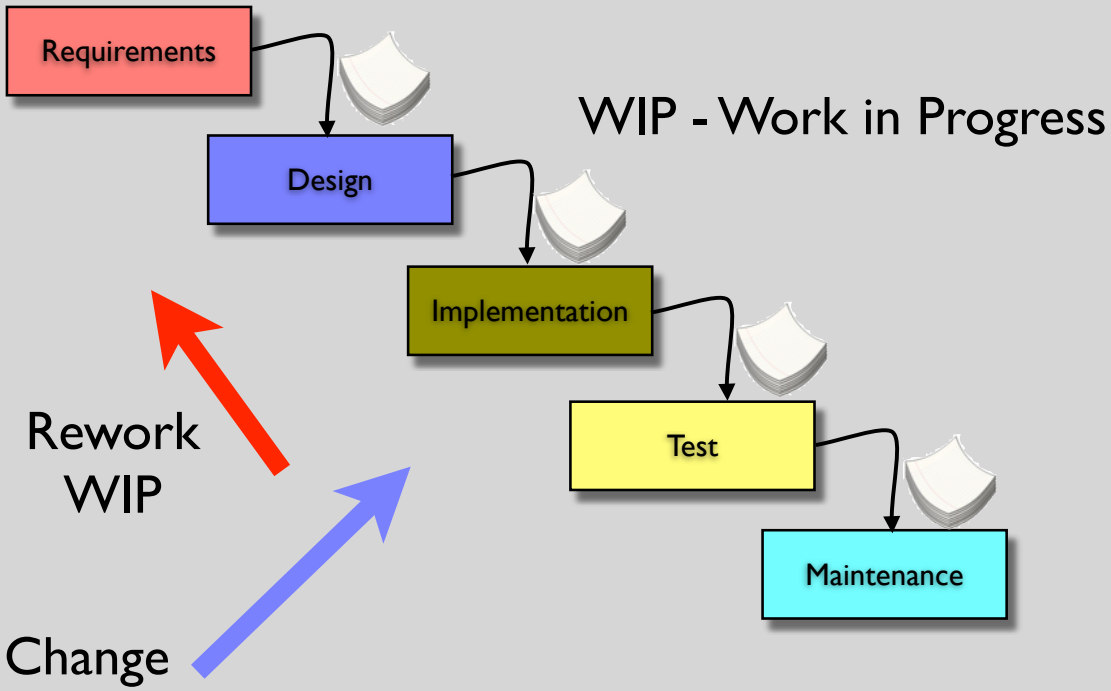
Original cost of change



Cost of Change - Kent Beck



Waterfall - Queues



Why agile?

Perfection Vision

Create the organizational ability
to respond to changes by being able to
to deliver or change direction
at any time
without additional cost

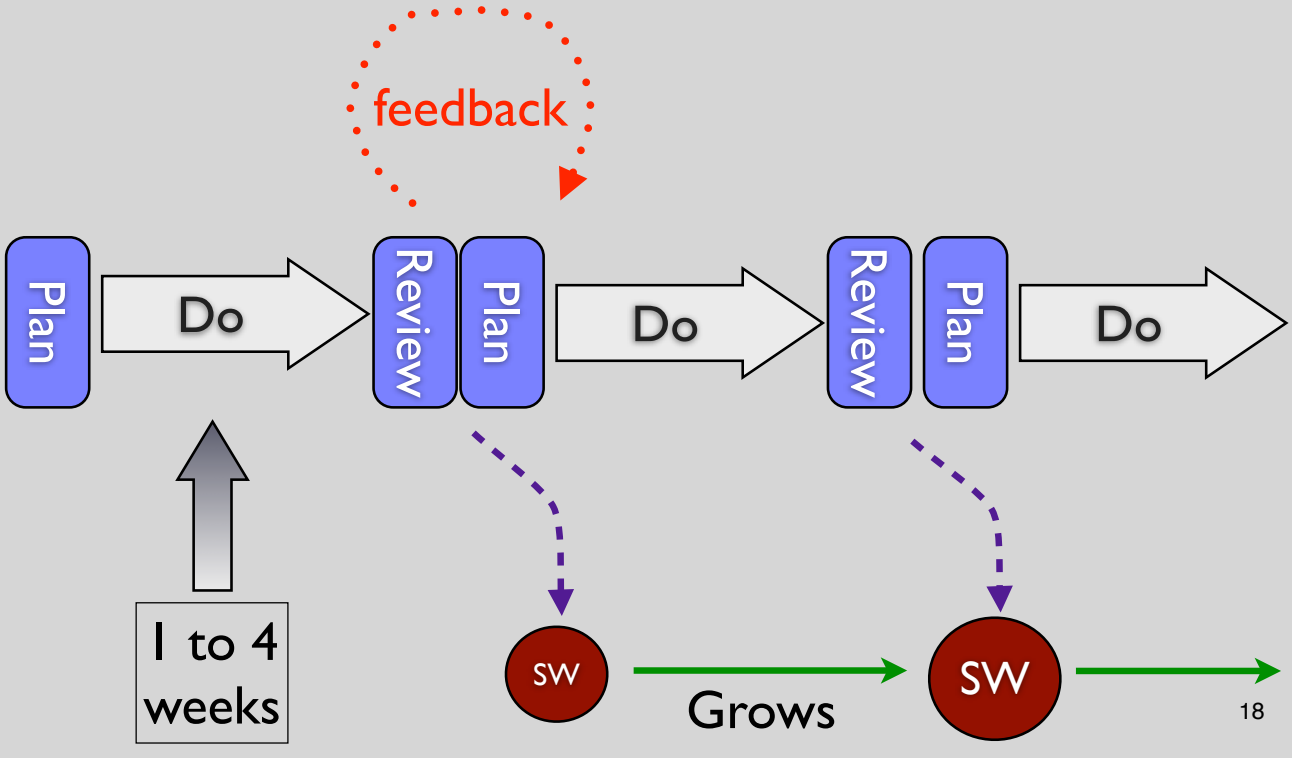
Perfection

Odd-e

Iterative Development

Odd-e

Iterations



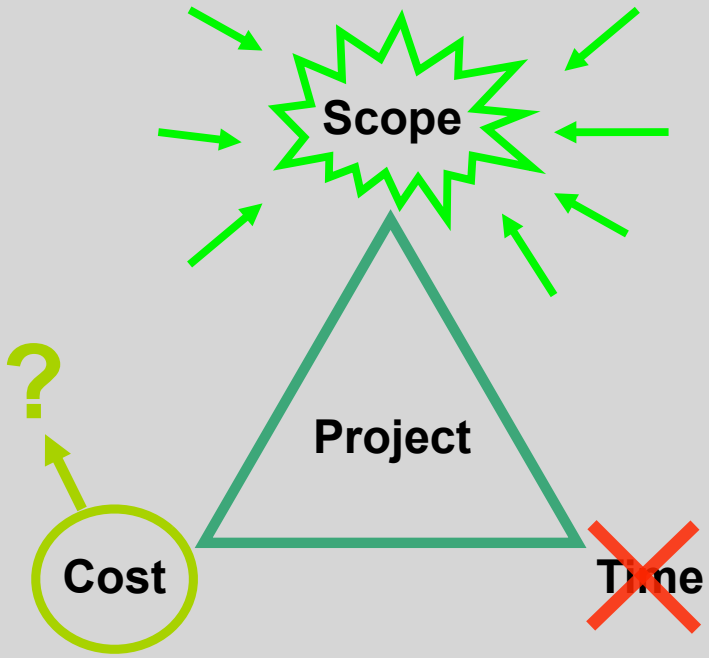
What is in the next iteration?

- Typical selection criteria:
  - Business value
  - Risk
  - Architectural importance

- Who decides:
  - In agile methods:
    - Scrum: Product owner
    - XP: Customer
    - In general: Customer, customer rep, product management

Agile methods are driven by the customer.
They require involvement of customers.

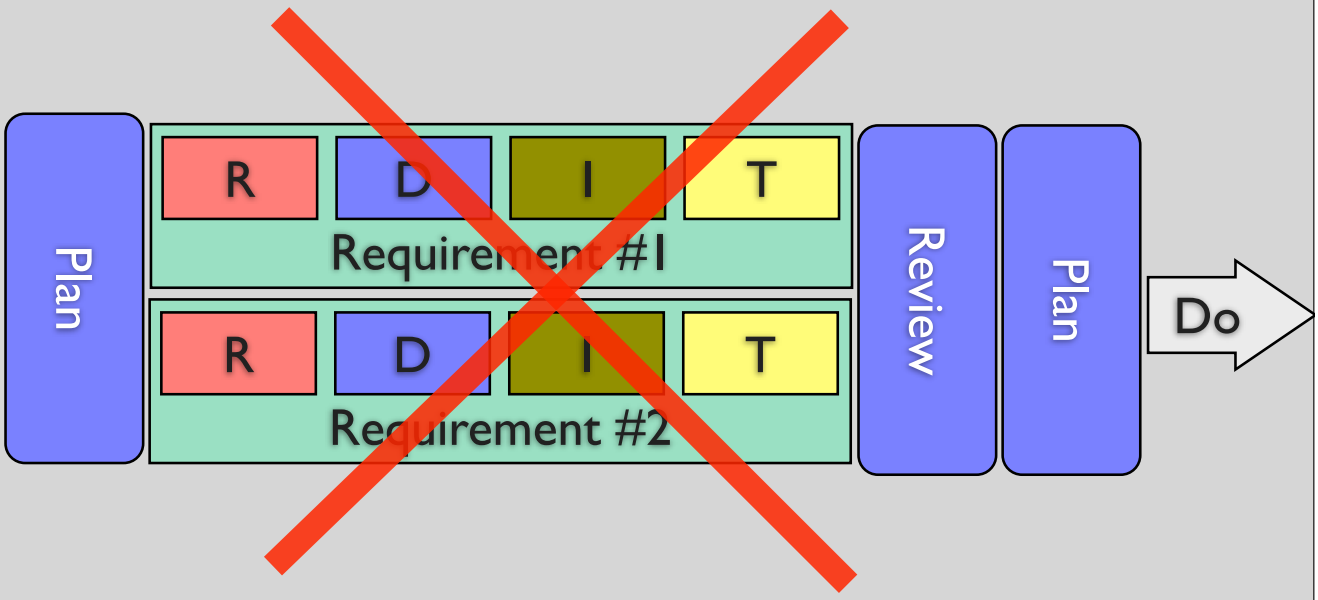
Timeboxing



Why time-box?

- Prioritization
- Keeps focus
- Avoids mini-waterfalls
- Better to slip scope than time

Mini-waterfalls?

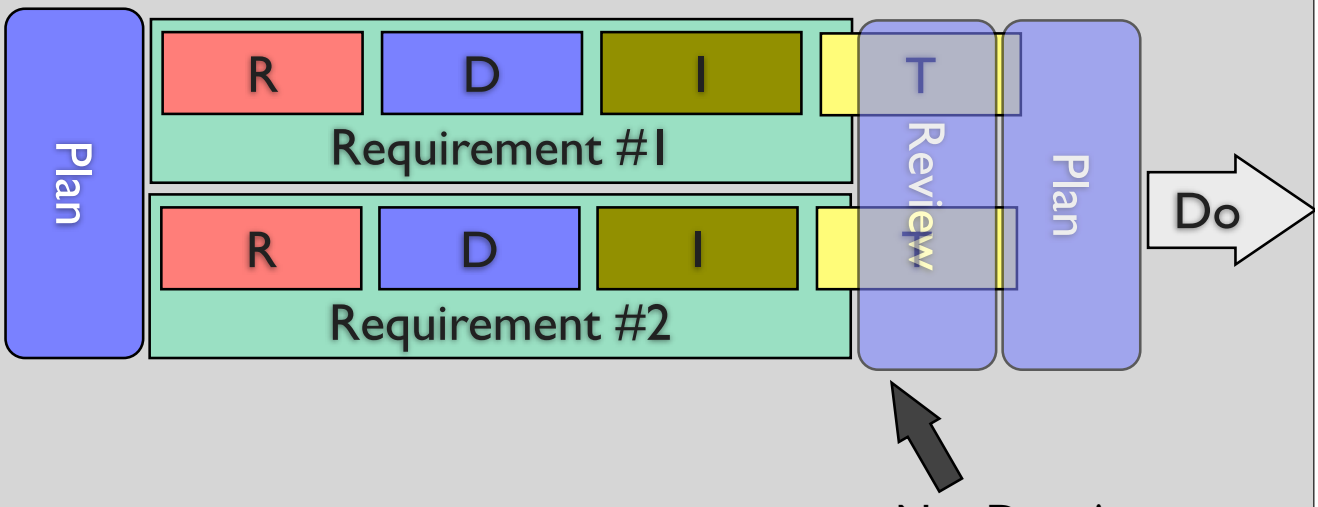


Why?

The Plan

Not a good idea!

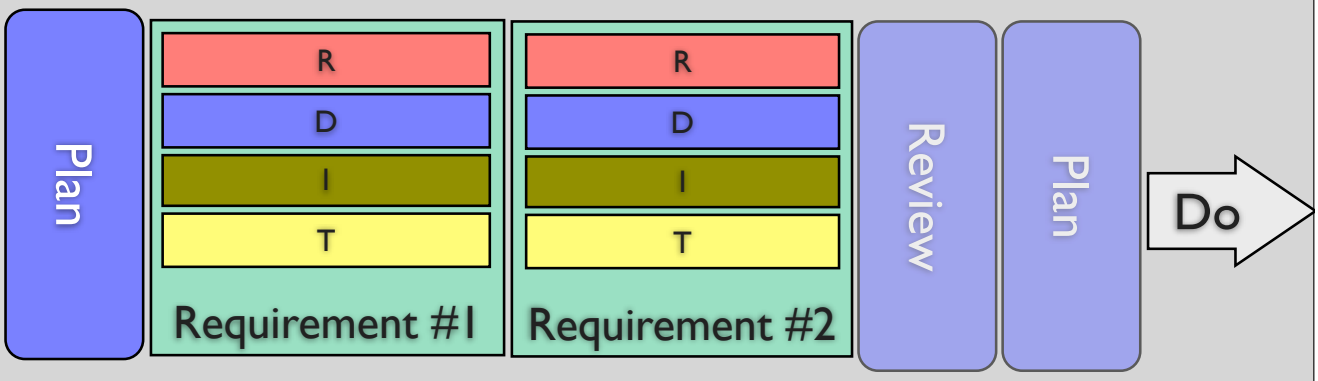
Mini-waterfalls?



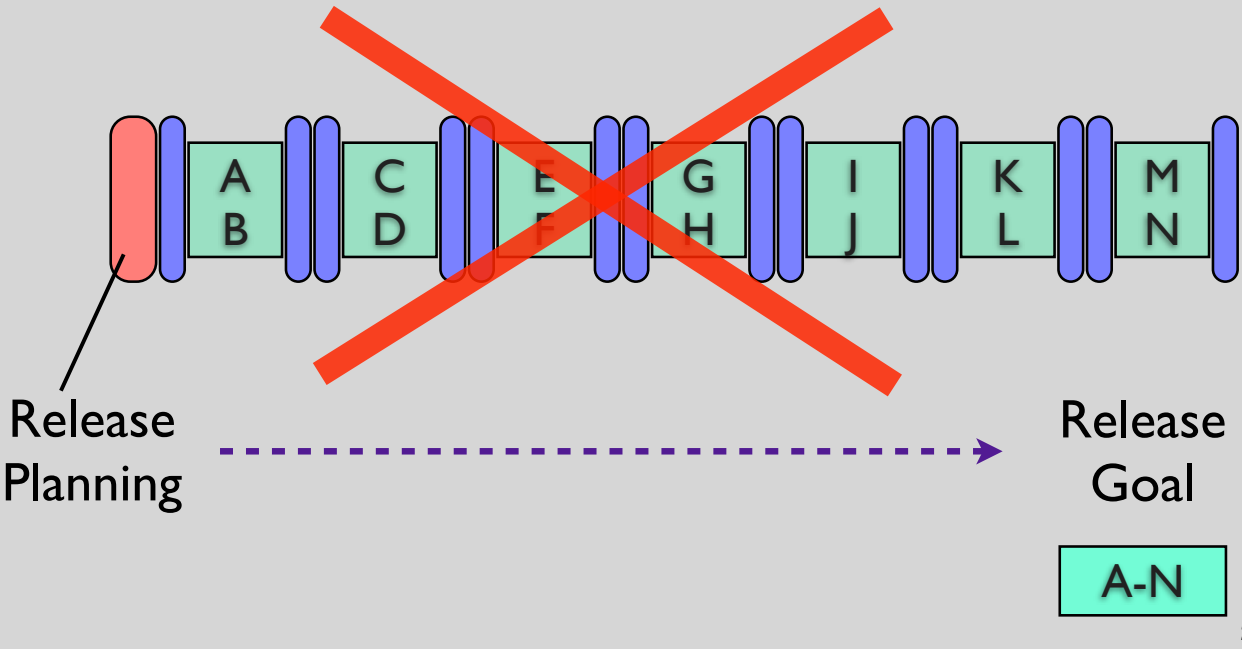
Not Done!
Should remove a requirement.
But **only** choice: Remove test

The Reality

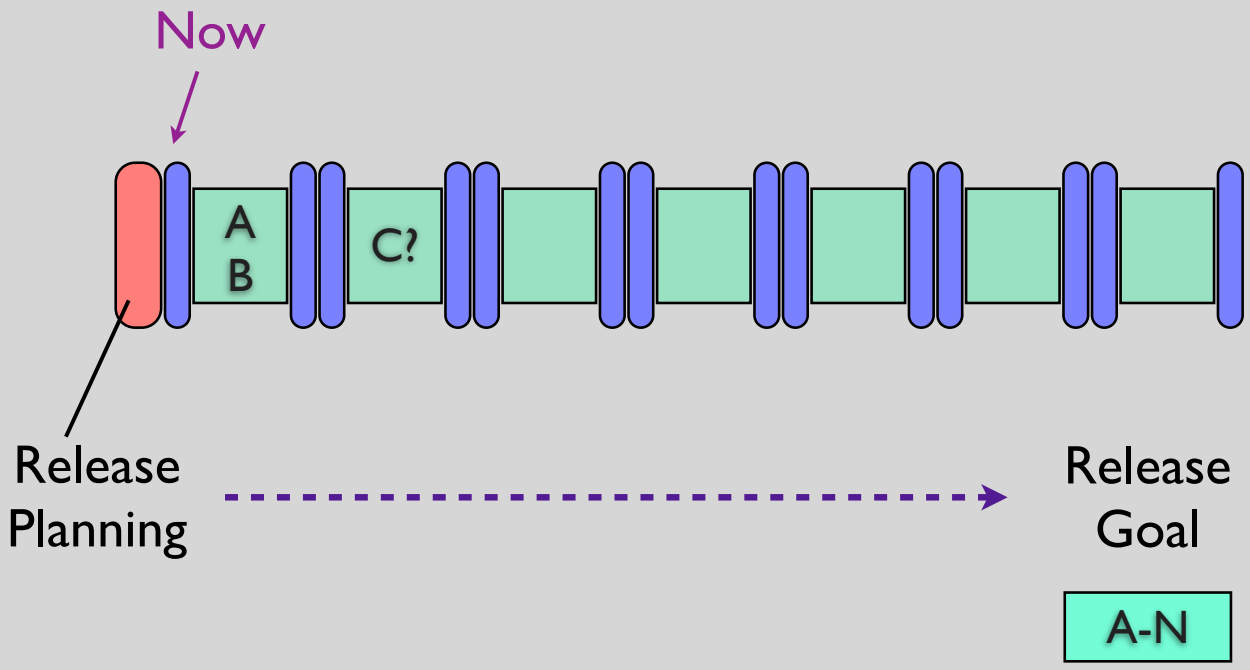
Alternatively - a good agile team



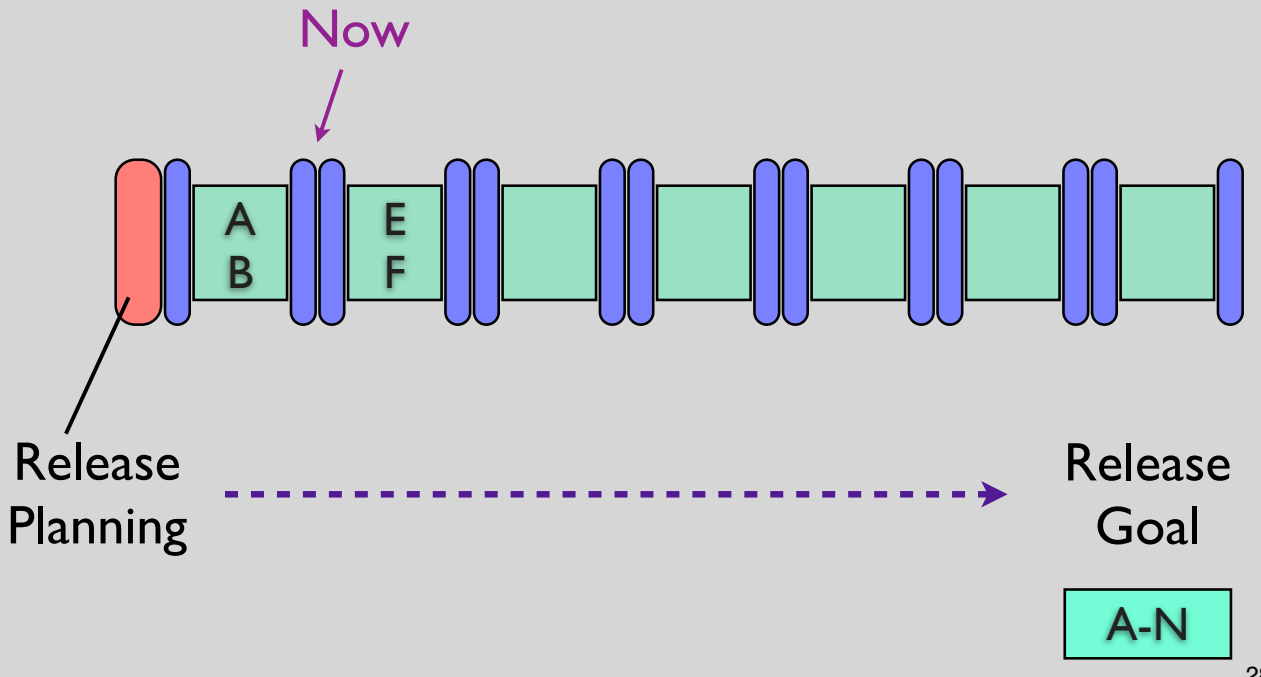
Predictive planning



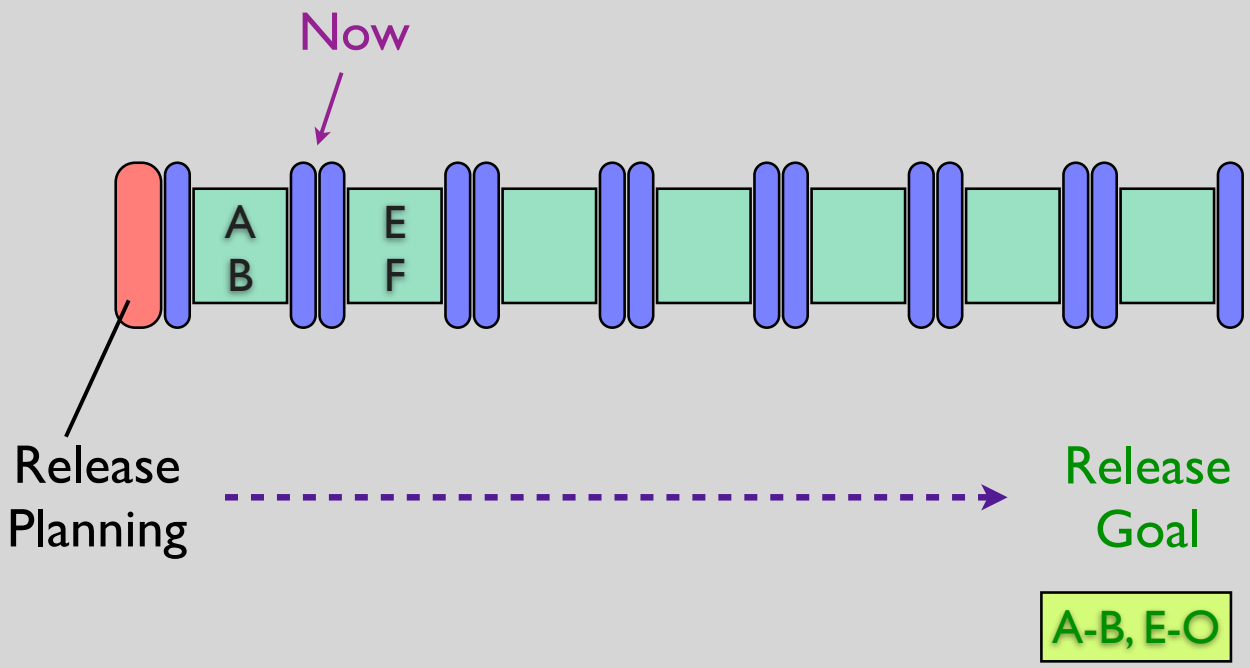
Adaptive planning



Adaptive planning



Or change in release goal



Not iterative when

- Iterations are longer than 4 weeks
- Iteration is not time-boxed
- Team completes specification before programming
- Iteration does not include testing
- Iteration does not produce workable code

Also known as "The Nokia-test"

Scrum

www.craiglarman.com
www.odd-e.com

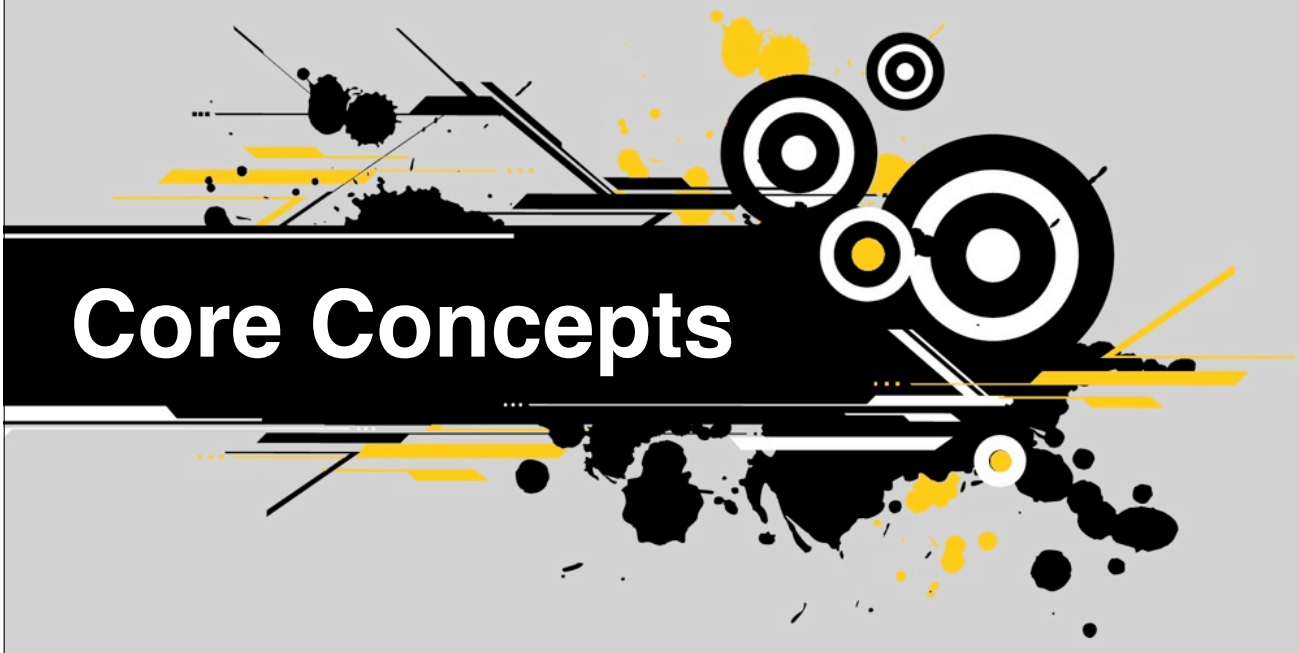
Copyright © C.Larman &
B. Vodde 2009.
All rights reserved.

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

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



Core Concepts

1. **Team** Teams

- Shared work product
- Interdependent work
- Shared responsibility
- Set of working agreements
- Responsibility for managing the outside-team relationships
- Distributed leadership

Shared Responsibility

2. Self-managing Teams

- The team together has the authority to:
  - Design, plan, and execute their task
  - Monitor and manage their progress
  - Monitor and manage their process

Empowerment

Text from: "Leading teams"
By Richard Hackman

Authority Matrix

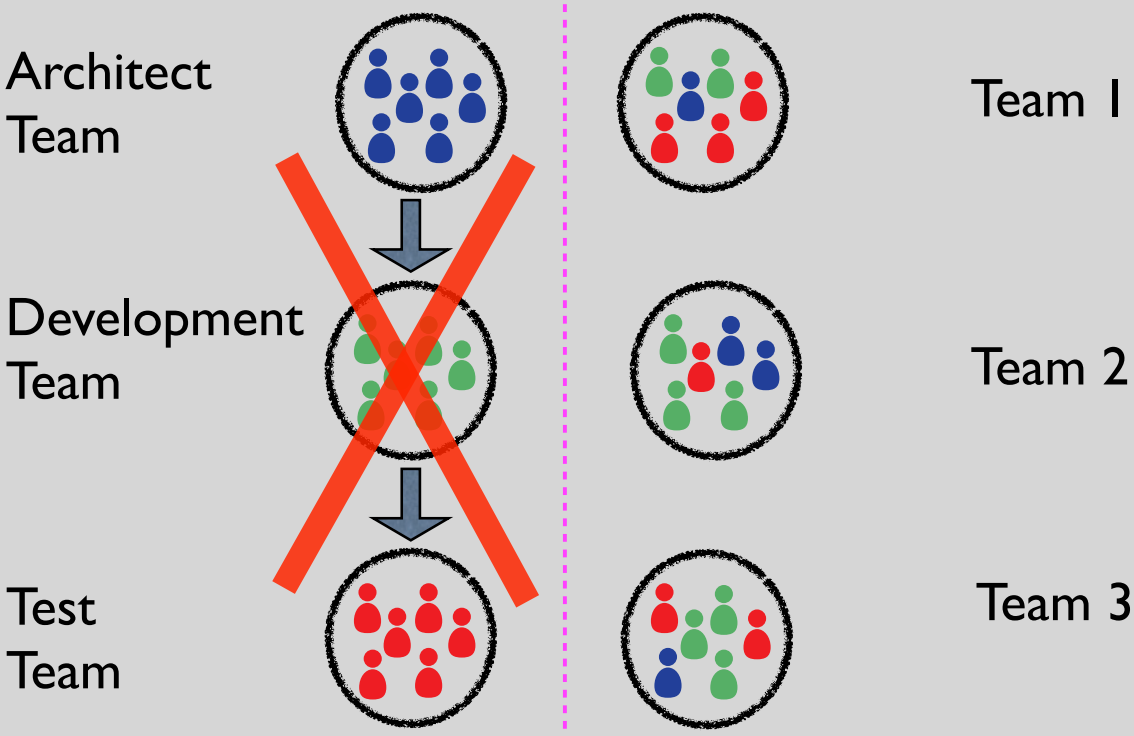
	Manager-led teams	Self-Managing teams	Self-Designing teams	Self-Governing teams
Setting overall direction				
Designing the team and its organizational context	Management Responsibility			
Monitoring and managing work process and progress		Team's Own Responsibility		
Executing the team task				

3. Cross-functional Teams

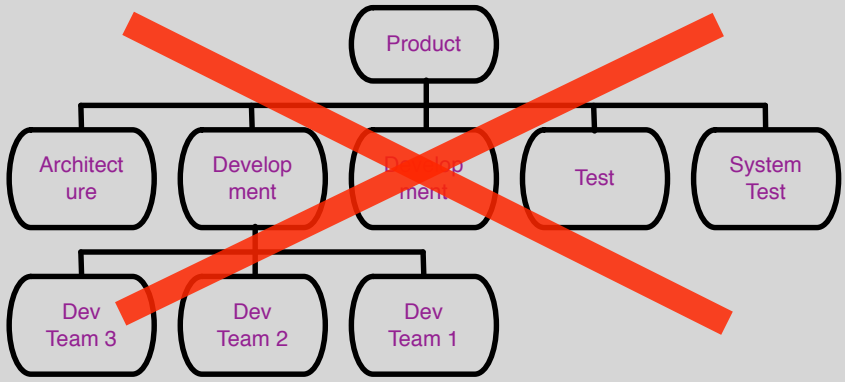
- All skills needed to build the product
- Balancing specialization with generalization
- Close cross-functional collaboration

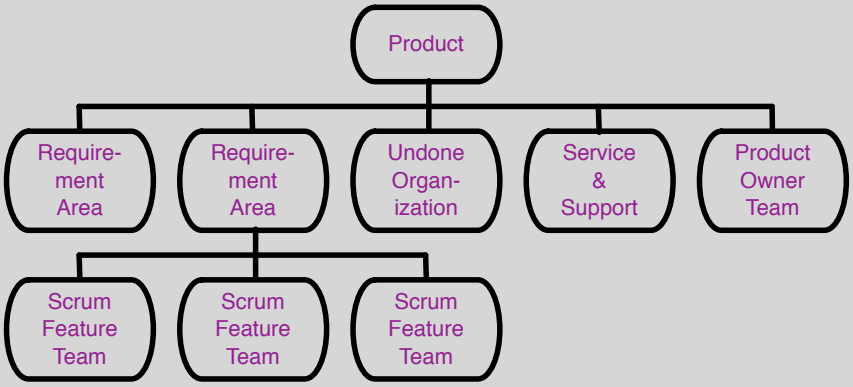
Multi-learning

Cross-functional teams



Organization





4. Short Iterative Full-Cycle Feedback

- Feedback
  - For improving product
  - For improving ways of working
- Iterative - repeating same activities
- Full-cycle - not phased
- Short - typically 2 weeks

Thanks to Jeff Patton

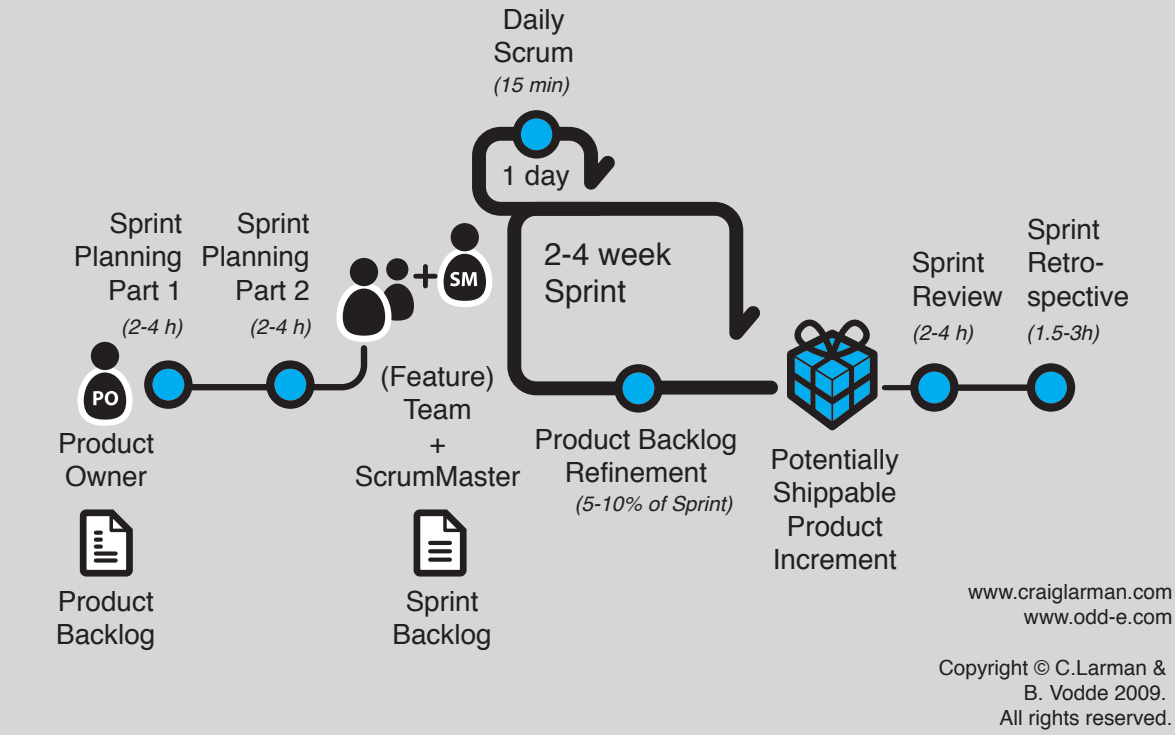
Inspect-adapt

5. Lowering Cost of Change

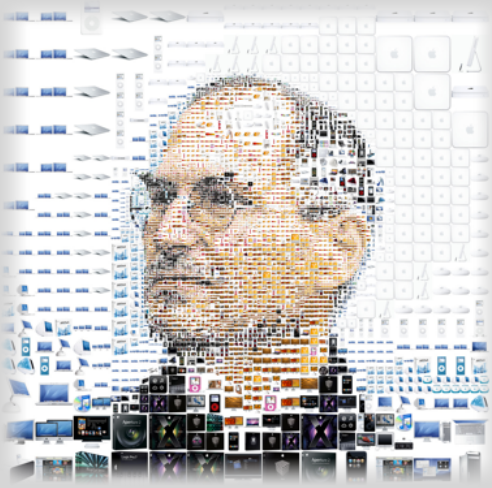
- Make responding to change economical
- Common strategies:
  - Lower work in progress
  - Remove duplication
  - Lowering complexity
  - Automation

Improvement

Scrum

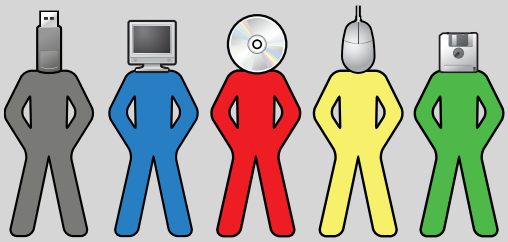


- ☑ Has product vision
- ☑ Responsible for profitability (ROI) and delivery



product owner

- Team team
- Cross-functional
- Self-managing
- Authority over tasks
- Shared responsibility

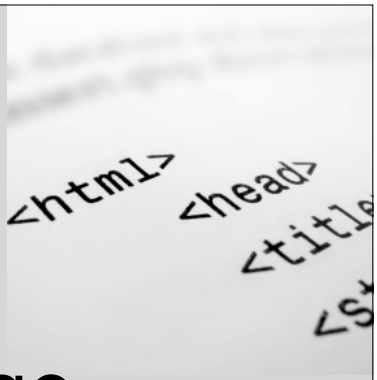


the team

- Leader & facilitator
- Agile coach
- Keeper of Scrum
- Not a project manager

scrum master

Misconceptions



Only an R&D change

Untrue, because:

- Majority of change impacts product management
- Major change in management style
- Roles and responsibilities across the organization will be affected

Traditional project management is compatible with Scrum

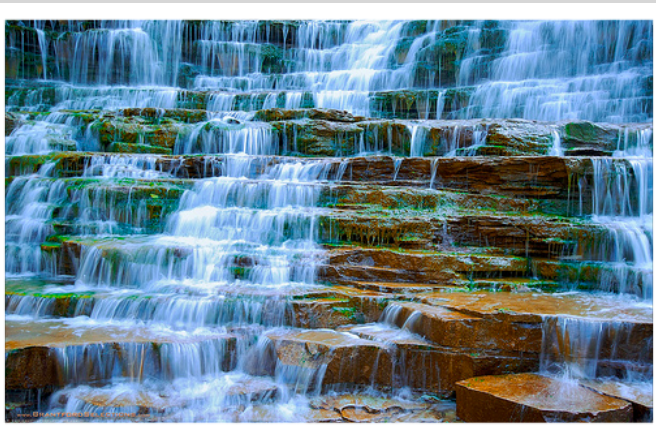
Untrue, because:

- In Scrum there is no project management role
- Usually the project management role ceases to exist!!
- When project managers are kept, it leads to problems.
- Projects are still managed by the three Scrum roles, not a PM

Significant improvement *happen* **without** significant changes in organizational structure or practices

Untrue, because:

- Significant changes are required, without it, no significant change will happen
- “We can’t solve problems by using the same kind of thinking we used when we created them.” (einstein)



Mini-waterfalls

Untrue, because:

- Agile development is **not** a series of mini-waterfalls
- Work within iterations is completely parallelized and often reversed.

Ad-hoc and ‘hacking’

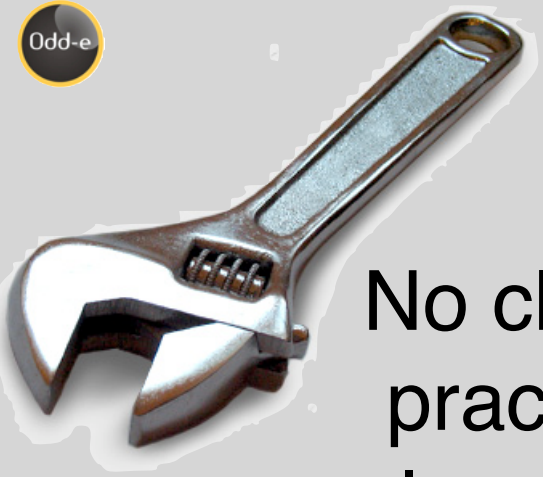
Untrue, because:

- Good agile development is much more disciplined than most traditional ways of working... but it is not waterfall development.

No visibility

Untrue, because:

- Good agile development **increases** visibility and predictability
- Yet at the same time acknowledges:
  - Some things are unpredictable
  - Change **will** happen!



No change to technical practices are needed, only management change

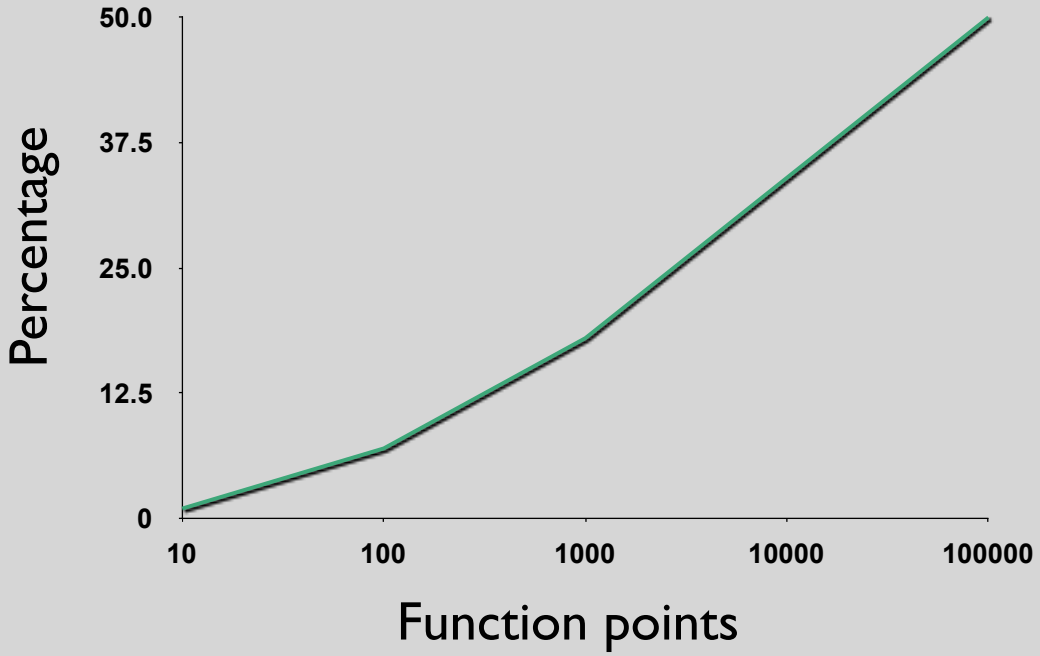
Untrue, because:

Organizational agility is constrained by technical agility

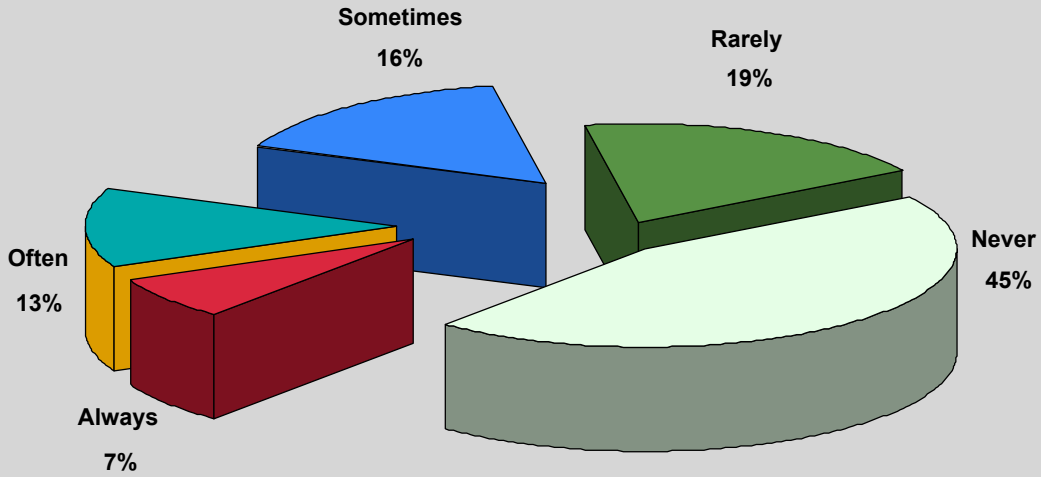

Odd-e
Additional Material


Odd-e
Statistics

Requirements change



Unused features



Programmer Productivity

Between fastest and slowest

28x

1

Original 1968 Sackman study

Between fastest and slowest

14x

1

Corrected Sackman study (2000)

Common difference

5x

1

Boehm study (1975)

Between top quarter and bottom quarter

4x

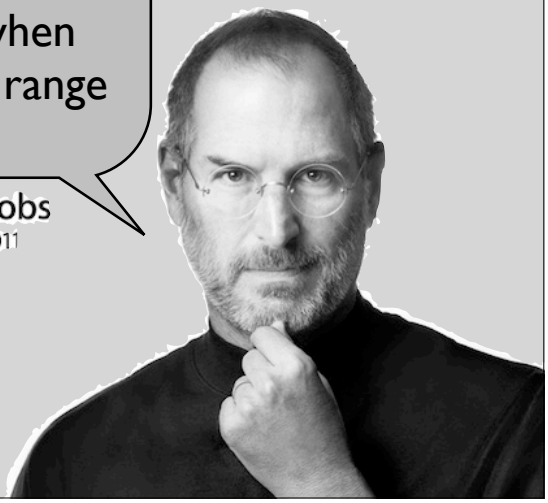
1

Prechelt (2000)

For Software teams. This difference is even bigger

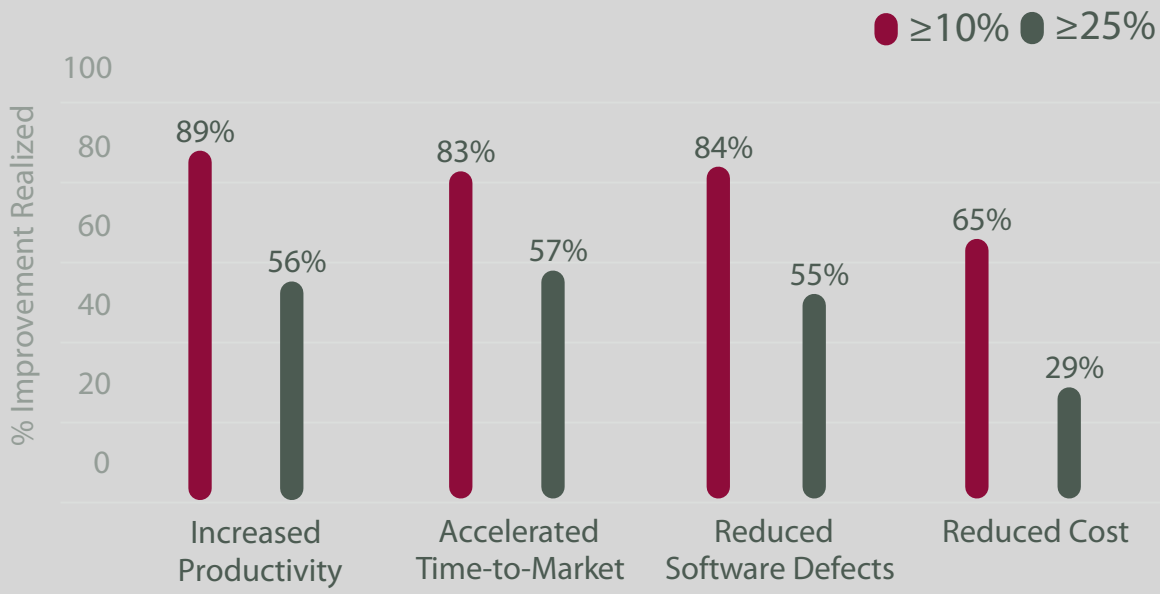
The difference between the best worker on computer hardware and the average may be 2 to 1, if you're lucky.
With automobiles, maybe 2 to 1.
But in software, it's at least 25 to 1.
The difference between the average programmer and a great one is at least that. The secret of my success is that we have gone to exceptional lengths to hire the best people in the world. And when you're in a field where the dynamic range is 25 to 1, boy, does it pay off.

Steve Jobs
1955-2011



Ref: gigaom.com/apple/steve-jobs-the-lost-years/

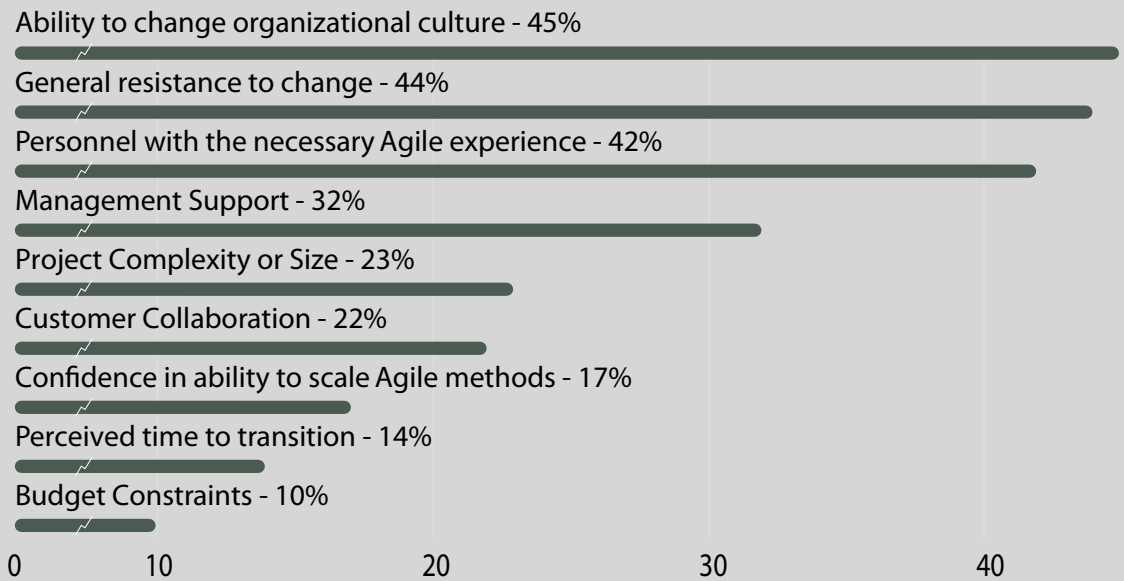
Improvements from adopting agile
(VersionOne) survey



Barriers to adopting agile
(VersionOne) survey

What are the barriers to further adoption of Agile in your current organization?

(select all that apply)



Organizational Impediments

10. Failure to remove organizational impediment
9. Misguided cost savings and synergy efforts
8. Lack of training
7. Single-function groups
6. Local vs. global optimization
5. Assumptions that book learning is enough
4. Individual performance evaluation and reward
3. Unrealistic promises
2. Assuming agile is all about developers
1. Silver bullet thinking and superficial adoption

Lean &
Toyota Production System

Produce according to demand and no more. To be able to respond to changes in the market

"At a time when all of us are struggling to implement lean production and lean management, often with complex programs on an organization-wide basis, it is helpful to learn that the creators of lean had no grand plan and no company-wide program to install it."

Text from: "Birth of Lean"

Kaizen

Improve for improvements sake

Everybody in the organization, not a specialized improvement group

Gemba

```
public CppUTestParserListenerEventListCreator createParserListenerEventListCreator() {
    return new CppUTestParserListenerEventListCreator();
}

public String getLineQualifier() {
    return escapeForRegex(LINE_QUALIFIER);
}

public final String getComprehensiveLinePattern() {
    return LINE_QUALIFIER + "(.*)(.*)(\\n)";
}
```


Go and See

"I urge you to make a special effort to see what's happening in the workplace. That's where the facts are. And the truth is hidden in the facts"

Text from: "Birth of Lean"

Managers as Teachers

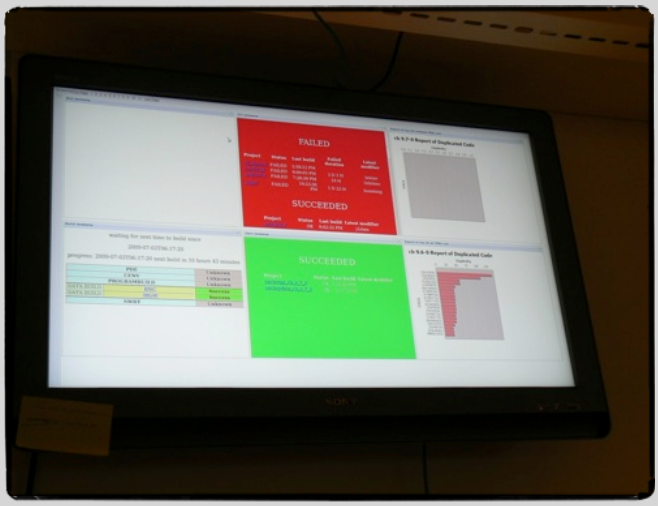
"Through education and training, subordinates become reliable, and the span of control becomes larger and larger. My ideal is to have one supervisor for every one hundred workers"

Text from: "Total Quality Control the Japanese Way" by Koaru Ishikawa

Avoid:

- Becoming stuck in bureaucracy
- Command people what to do

Stop & Fix



自働化

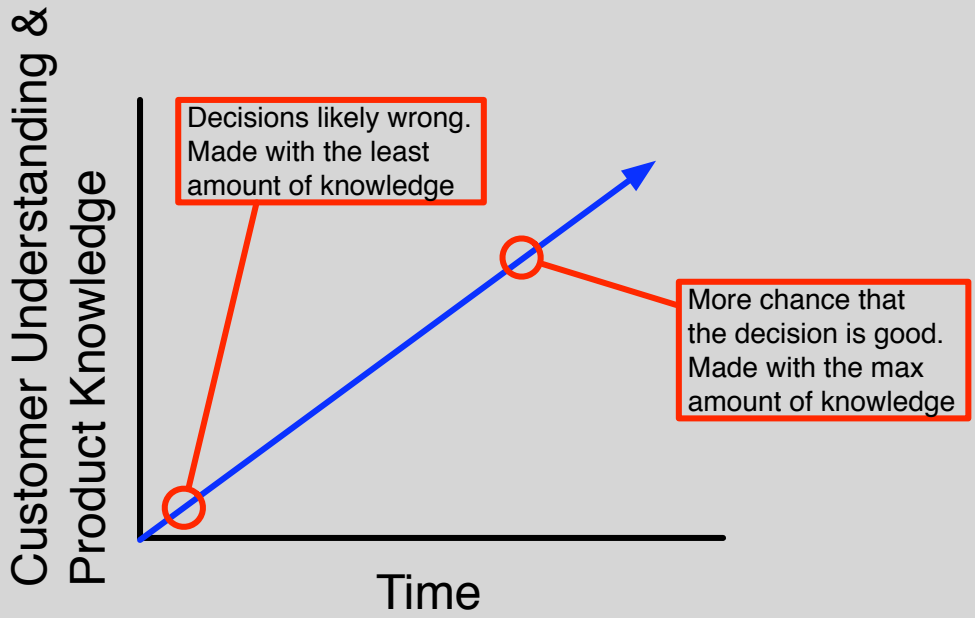
Seven Wastes

1. Over-production
2. Inventory
3. Motion
4. Waiting
5. Transportation
6. Over-processing
7. Defects

Ten Wastes - Software Development

1. Overproduction of features
2. Waiting, delay
3. Handoff
4. Extra process
5. Partially done work
6. Task Switching
7. Defects
8. Under-realizing people's potential
9. Knowledge scatter
10. Wishful thinking

Defer commitment



last **responsible** moment

Especially relevant for architectural decisions!

www.craiglarman.com
www.odd-e.com

Copyright © 2009
C.Larman & B. Vodde
All rights reserved.

Sustainable shortest lead time, best quality and value (to people and society), most customer delight, lowest cost, high morale, safety

Respect for People
- don't trouble your 'customer'
- "develop people, then build products"
- no wasteful work
- teams & individuals evolve their own practices and improvements
- build partners with stable relationships, trust, and coaching in lean thinking
- develop teams

Product Development
- long-term great engineers
- mentoring from manager-engineer-teacher
- cadence
- cross-functional
- team room + visual mgmt
- entrepreneurial chief engineer/product mgr
- set-based concurrent dev
- create more knowledge

14 Principles
long-term, flow, pull, less variability & overburden, Stop & Fix, master norms, simple visual mgmt, good tech, leader-teachers from within, develop exceptional people, help partners be lean, Go See, consensus, reflection & kaizen

Continuous Improvement
- Go See
- kaizen
  - spread knowledge
  - small, relentless
  - retrospectives
  - 5 Whys
  - eyes for waste
    * variability, over-burden, NVA ... (handoff, WIP, info scatter, delay, multi-tasking, defects, wishful thinking..)
- perfection challenge
- work toward flow (lower batch size, Q size, cycle time)

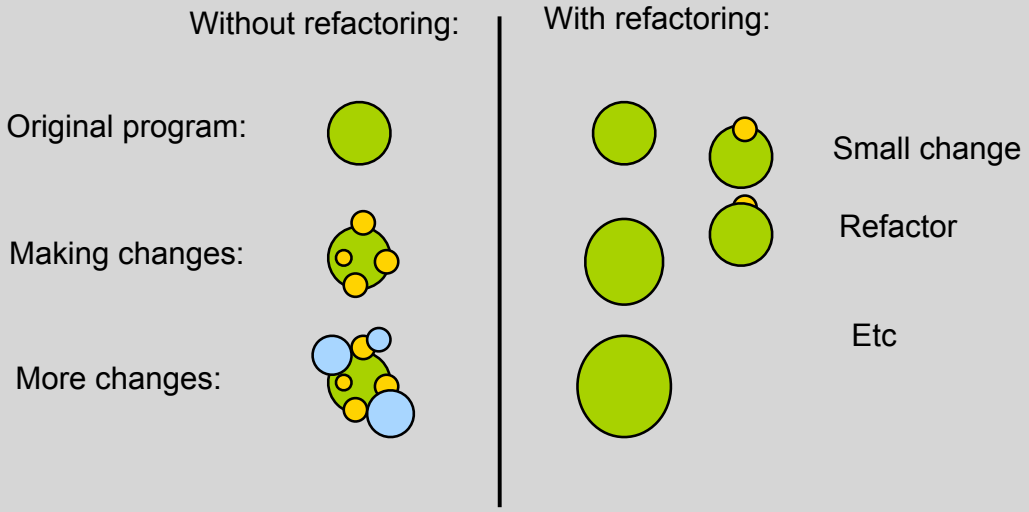
Management applies and teaches lean thinking, and bases decisions on this long-term philosophy

Technical Agility

How much education is there related to product development work?

+ Using your editor
+ Refactoring
+ Clean Code
+ Removing dependencies
+ Advance C++ design

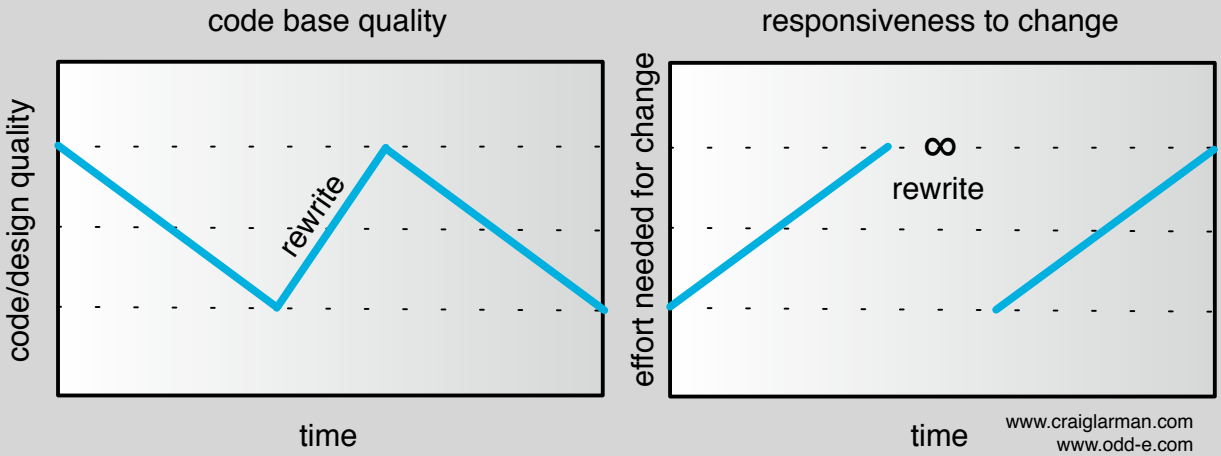
Codebase Quality



Cost of change increases rapidly!

Cost of change not increases

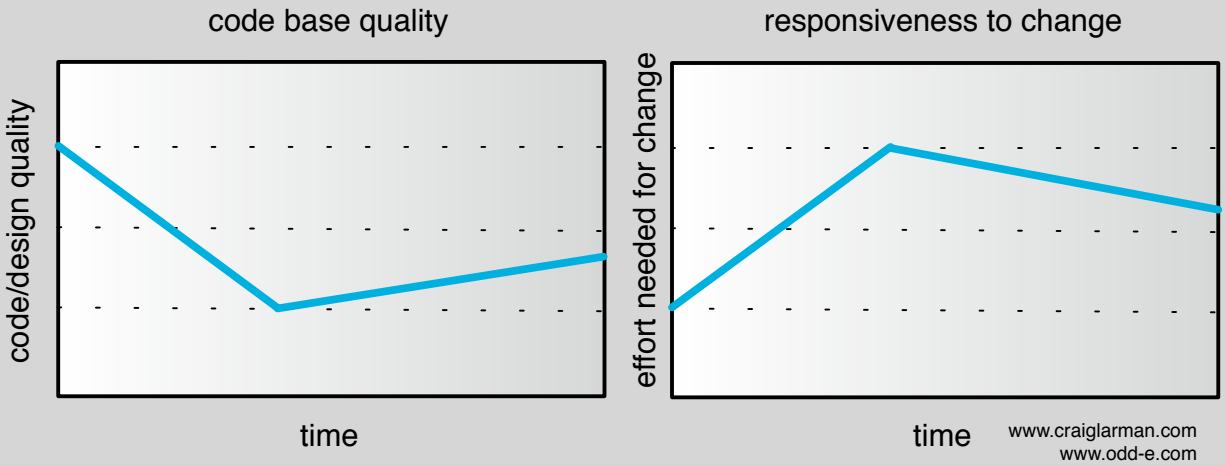
Creating Legacy Code



www.craiglarman.com
www.odd-e.com

Copyright © 2010
C.Larman & B. Vodde
All rights reserved.

Solve the root cause!



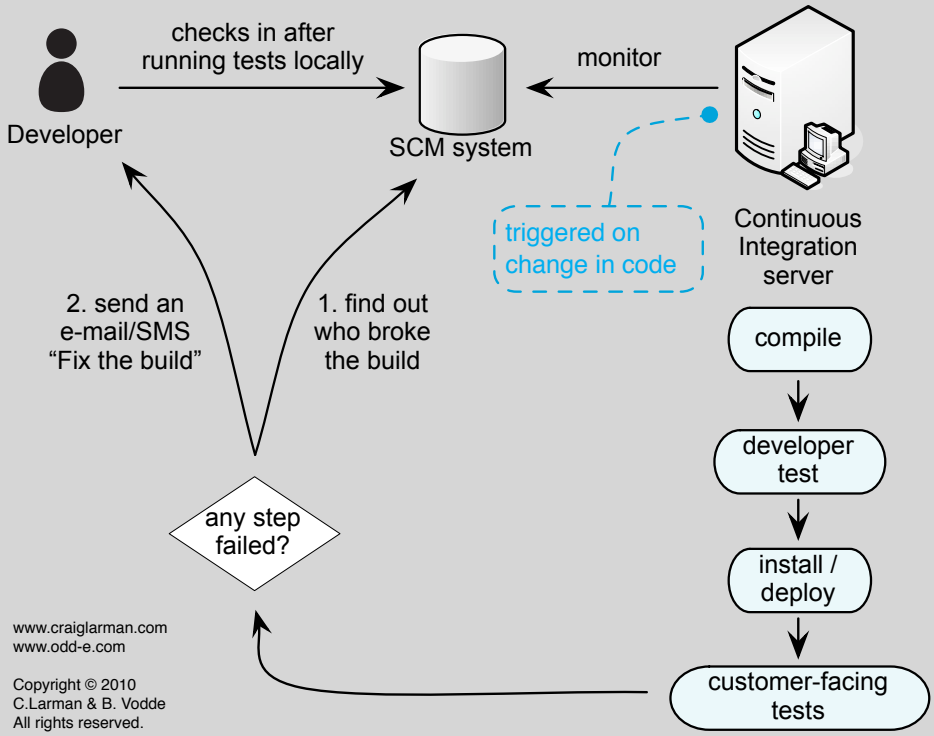
www.craiglarman.com
www.odd-e.com

Copyright © 2010
C.Larman & B. Vodde
All rights reserved.

Continuous Integration

Continuous Integration is a **developer practice**
with the goal to always keep a **working system**
by making **small changes**, slowly growing the system
and **integrating** them at least **daily**
on the **mainline**
typically supported by a **CI system**
with lots of **automated tests**

CI System

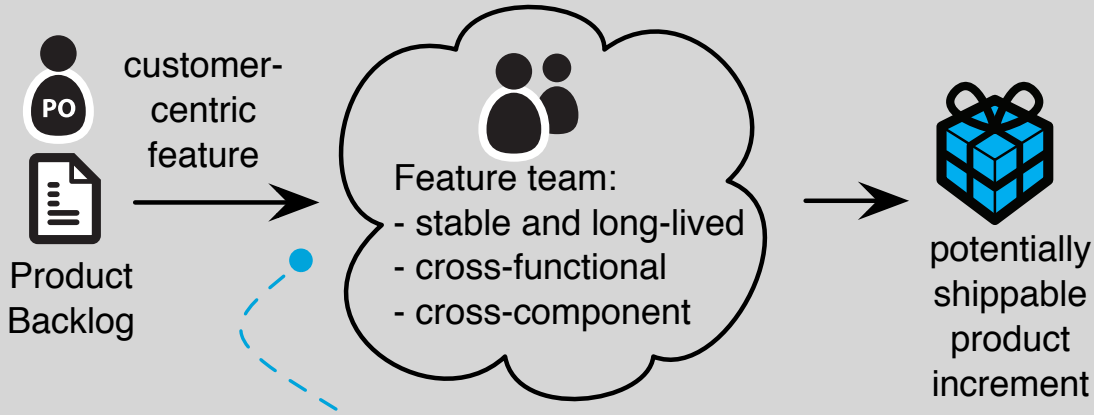


www.craiglarman.com
www.odd-e.com

Copyright © 2010
C.Larman & B. Vodde
All rights reserved.

Large

Feature teams



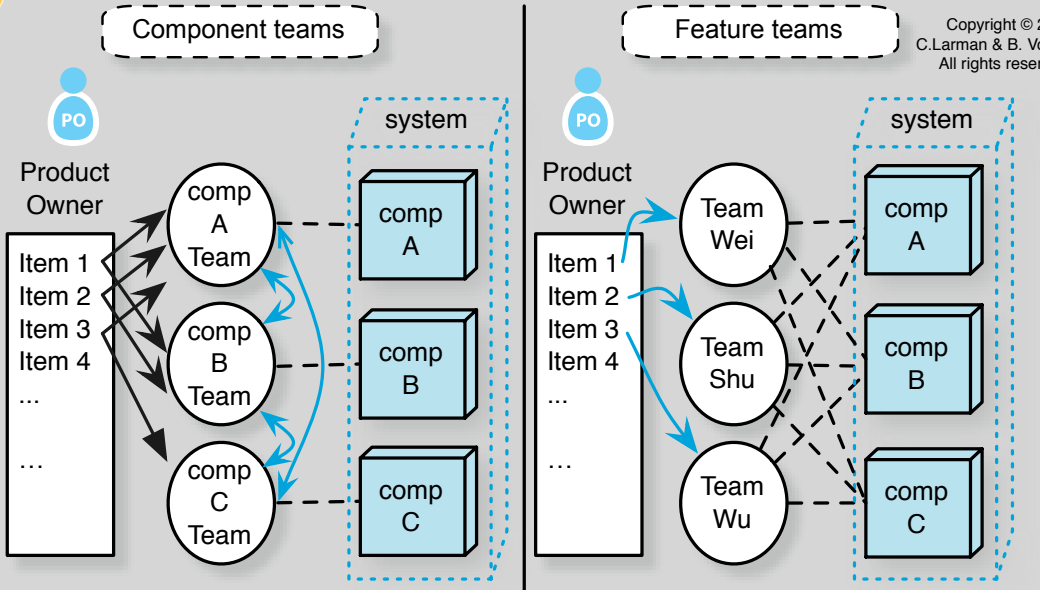
Team has the necessary knowledge and skills to complete an end-to-end customer-centric feature. If not, the team is expected to learn or acquire the needed knowledge and skill.

www.craiglarman.com
www.odd-e.com

Copyright © 2010
C.Larman & B. Vodde
All rights reserved.

www.craiglarman.com
www.odd-e.com

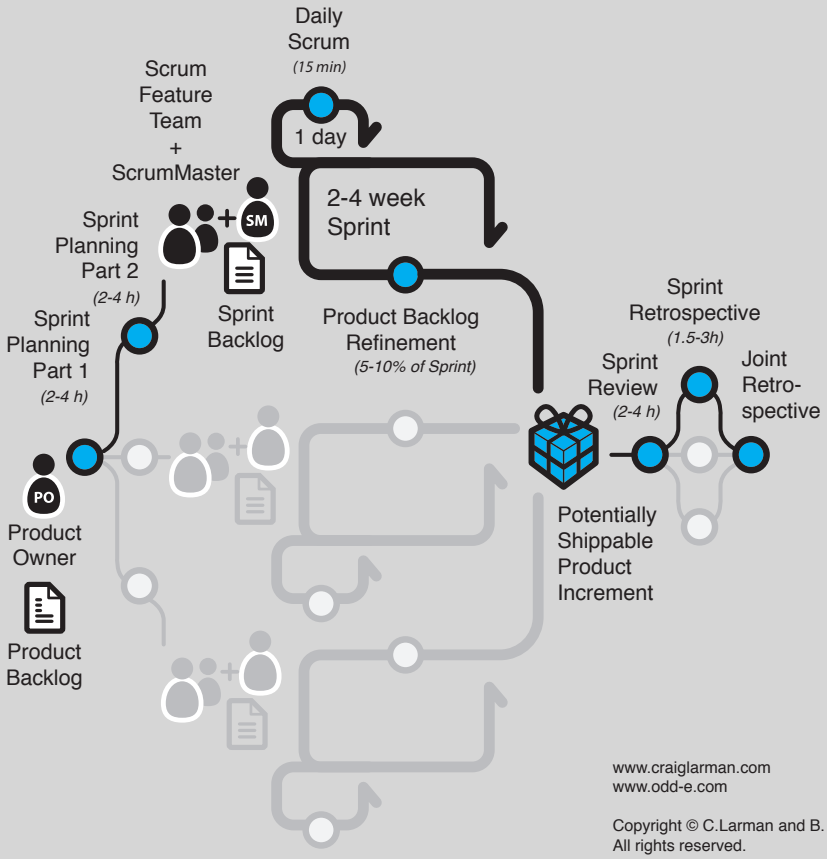
Copyright © 2010
C.Larman & B. Vodde
All rights reserved.



Work from multiple teams is required to finish a customer-centric feature. These dependencies cause waste such as additional planning and coordination work, hand-offs between teams, and delivery of low-value items.
Work scope is narrow.

Every team completes customer-centric items. The dependencies between teams are related to shared code. This simplifies planning but causes a need for frequent integration, modern engineering practices, and additional learning.
Work scope is broad.

Large-scale Scrum for up to 10 teams with one Product Owner



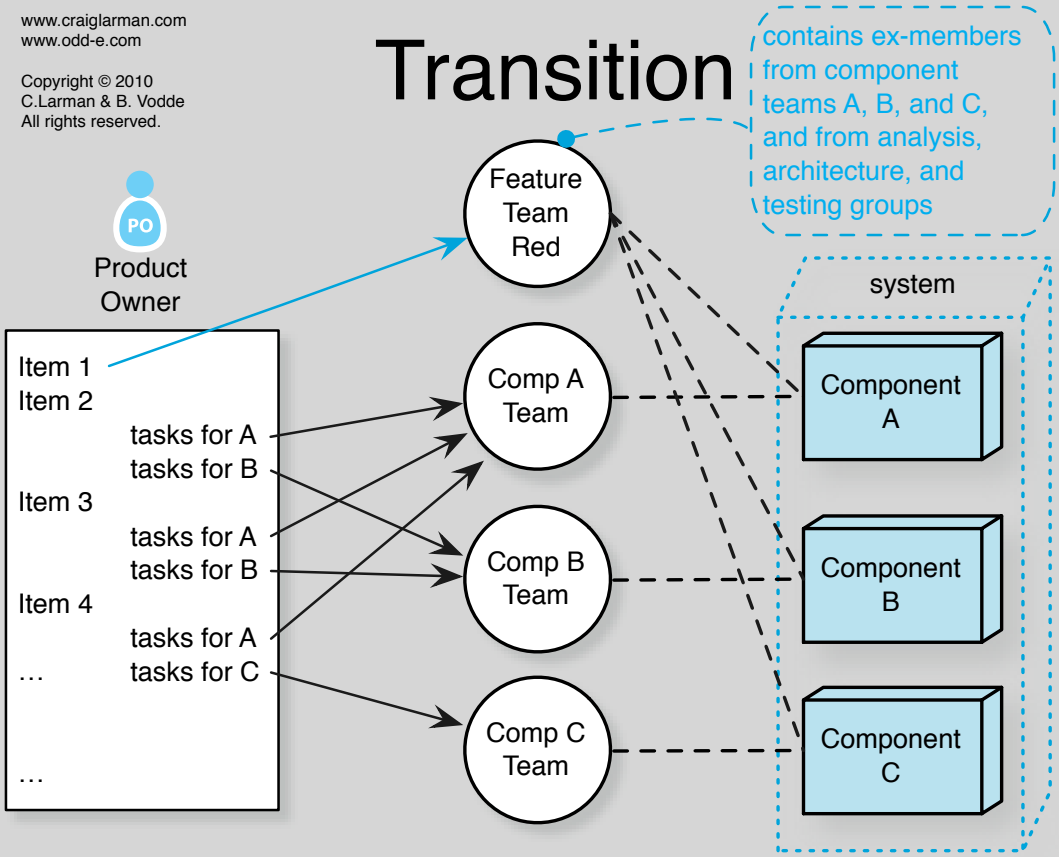
www.craiglarman.com
www.odd-e.com

Copyright © C.Larman and B. Vodde 2008.
All rights reserved.

www.craiglarman.com
www.odd-e.com

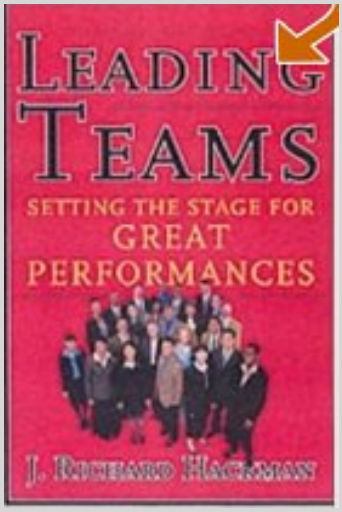
Copyright © 2010
C.Larman & B. Vodde
All rights reserved.

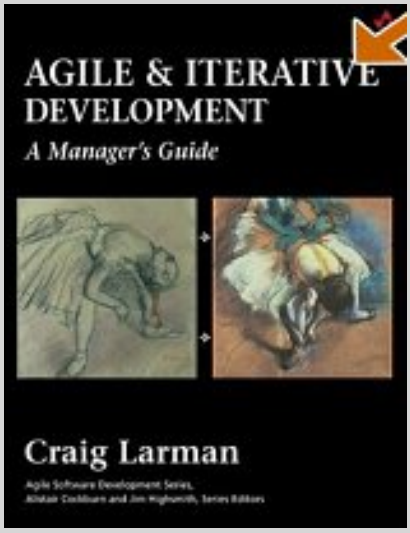
Transition

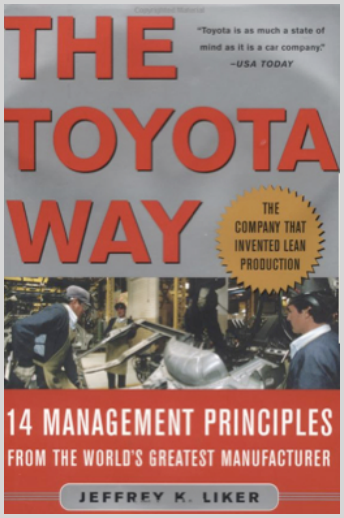


Resources

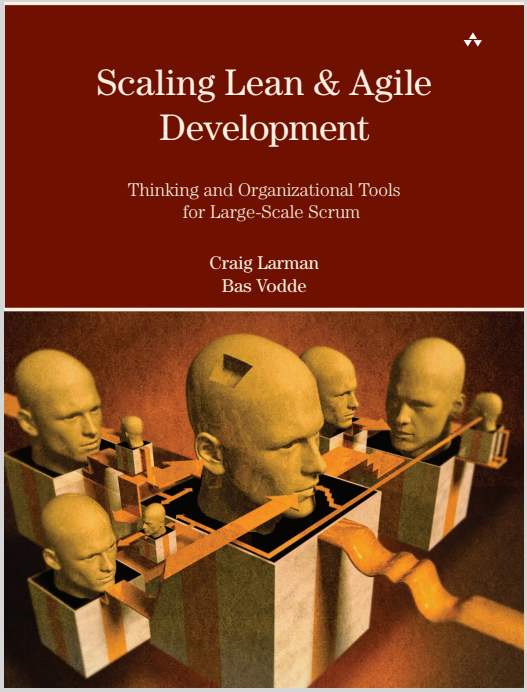
Books

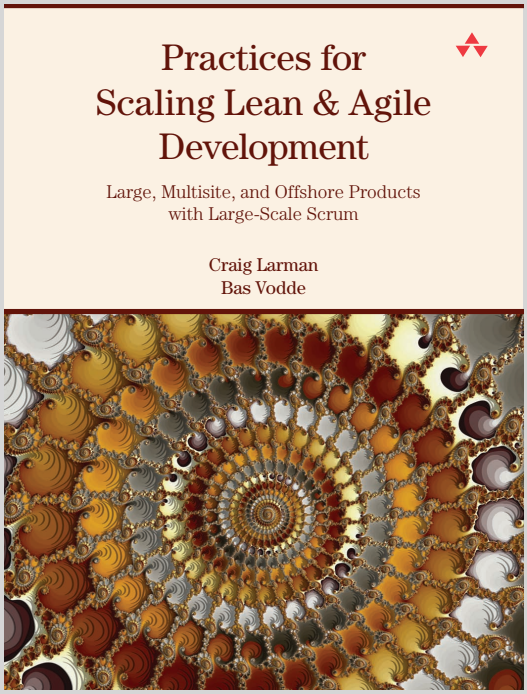






Books - Scaling






Q&A