



l e a n
software development

Lean Software Development

Workshop

Mary & Tom Poppendieck



Principles of Lean Software Development

1. Eliminate Waste

2. Embed Quality

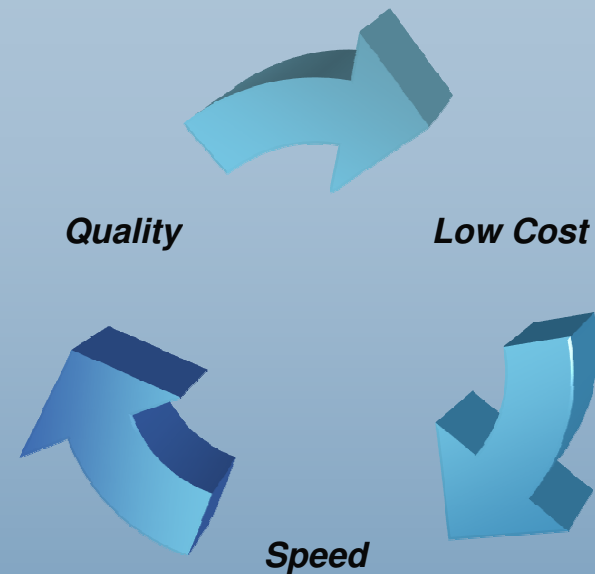
3. Learn First

4. Deliver Fast

5. Improve Forever

6. Respect People

7. Think System



l e a n



What is Waste?

Waste is...

Anything that depletes resources of time, effort, space, or money without adding customer value.

Value is...

Seen through the eyes of those who pay for, use, and derive value from the systems we create.

A System is...

A complete and useful product, along with whatever is needed to create and distribute the product.



Eliminate Waste

The Seven Wastes of Manufacturing - Taiichi Ohno

1. Overproduction
2. Work In Process
3. Overprocessing
4. Transportation
5. Motion
6. Waiting
7. Defects

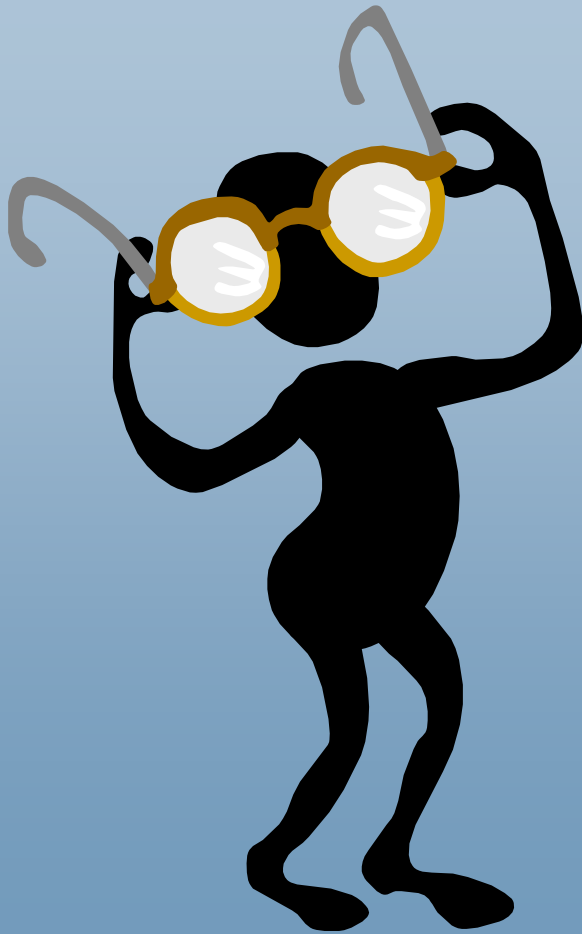
The 7 Wastes of Software Development

1. Extra Features
2. Partially Done Work
3. Relearning
4. Handoffs
5. Task Switching
6. Delays
7. Defects

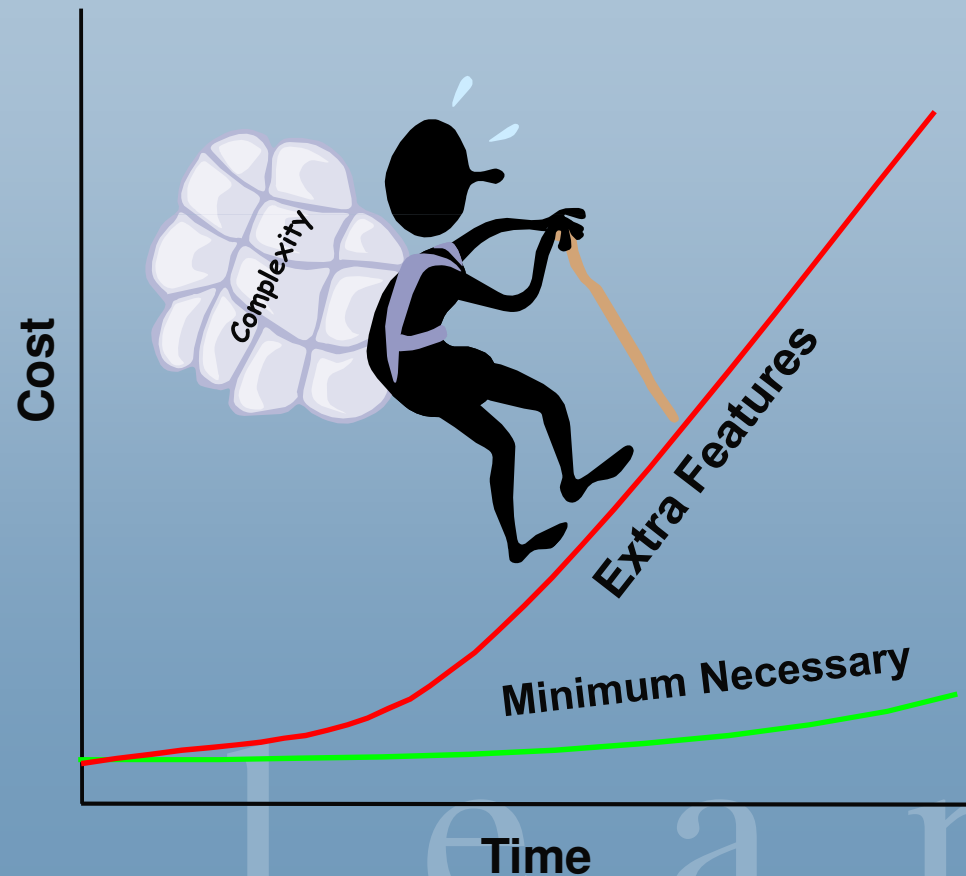


If It's Not Value, It's Waste

Put on Customer Glasses



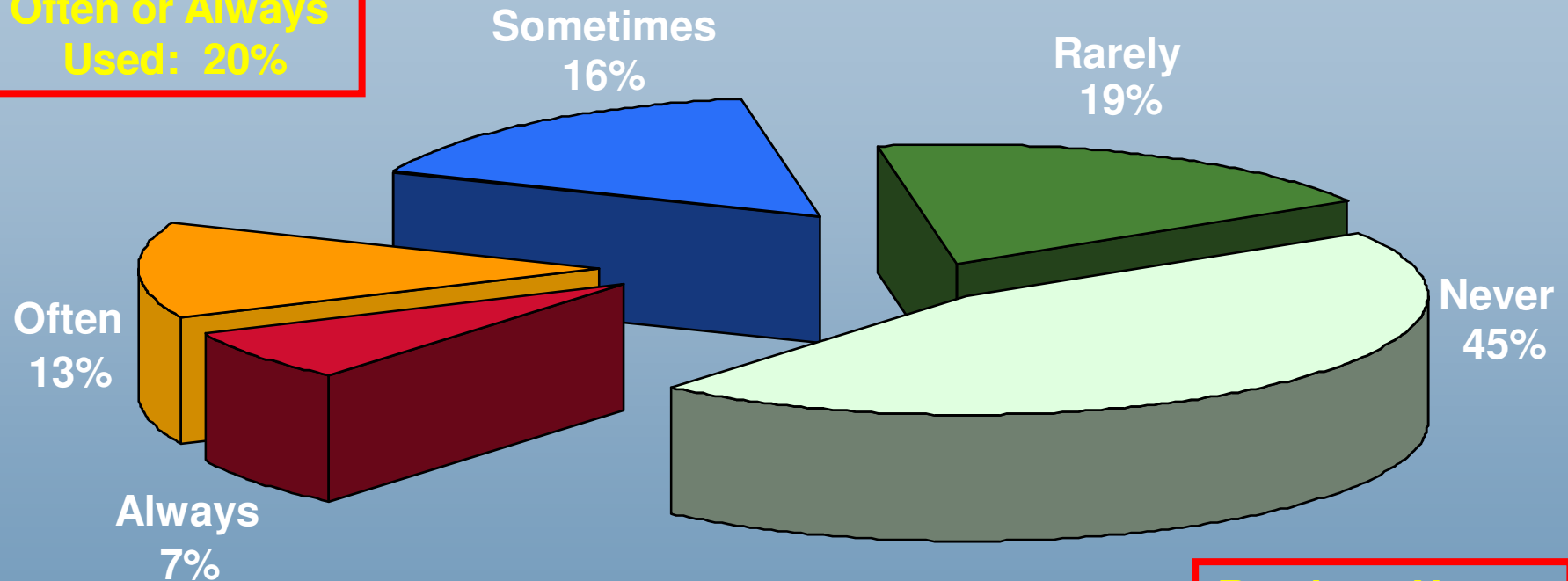
Extra Features



The 80-20 Rule

Features and Functions Used in a Typical System

**Often or Always
Used: 20%**



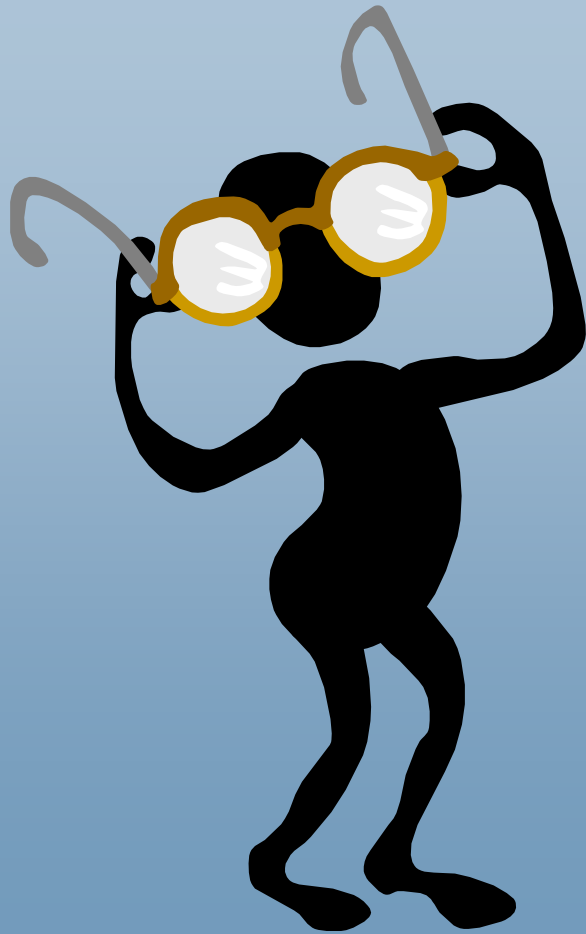
**Rarely or Never
Used: 64%**

Standish Group Study Reported at XP2002 by Jim Johnson, Chairman

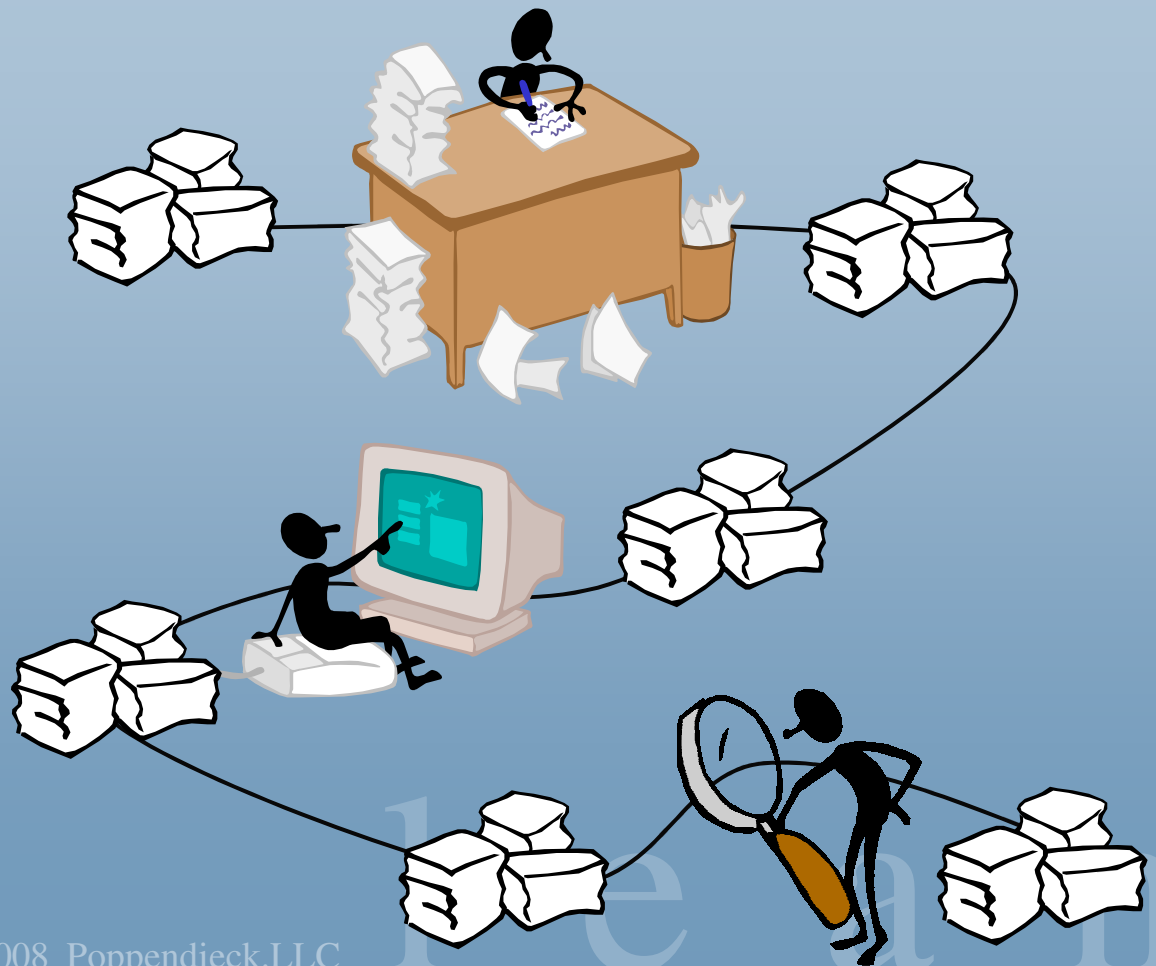


If It's Not Value, It's Waste

Put on Customer Glasses



Partially Done Work





Set-up Time

Manufacturing

Common Knowledge:

- ✓ Die changed have a huge overhead
- ✓ Don't change dies very often

Taiichi Ohno:

- ✓ Economics requires frequent die change
- ✓ *One Digit Exchange of Die*

Software Development

Common Knowledge:

- ✓ Releases have a huge overhead
- ✓ Don't release very often

Lean:

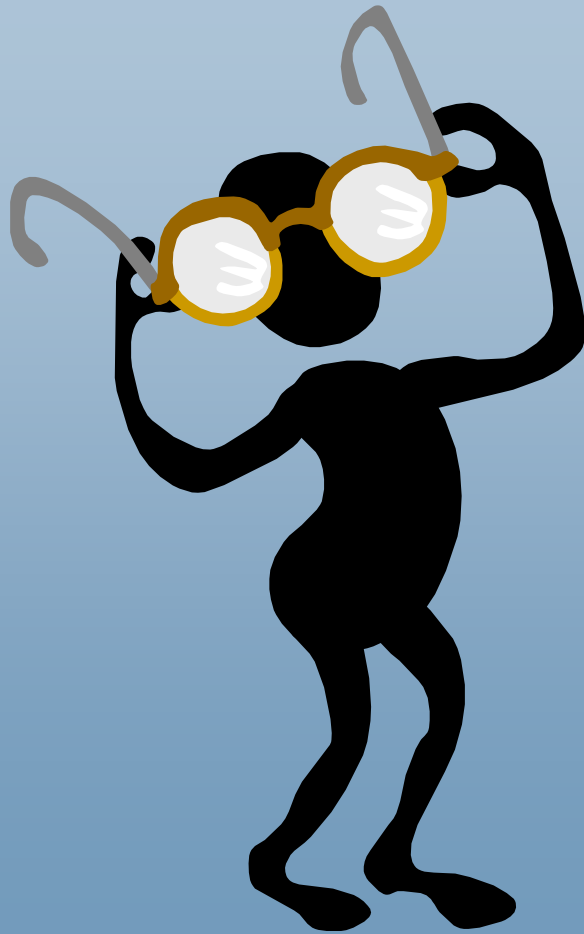
- ✓ Economics requires many frequent releases
- ✓ *One Digit Releases*



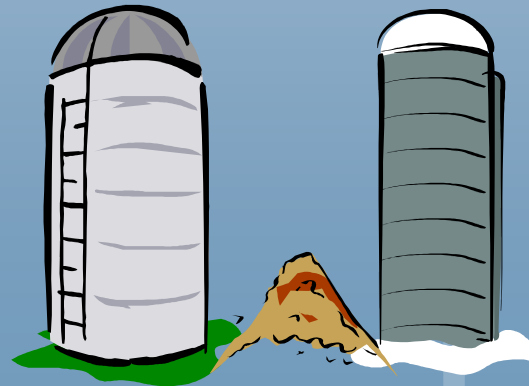


If It's Not Value, It's Waste

Put on Customer Glasses



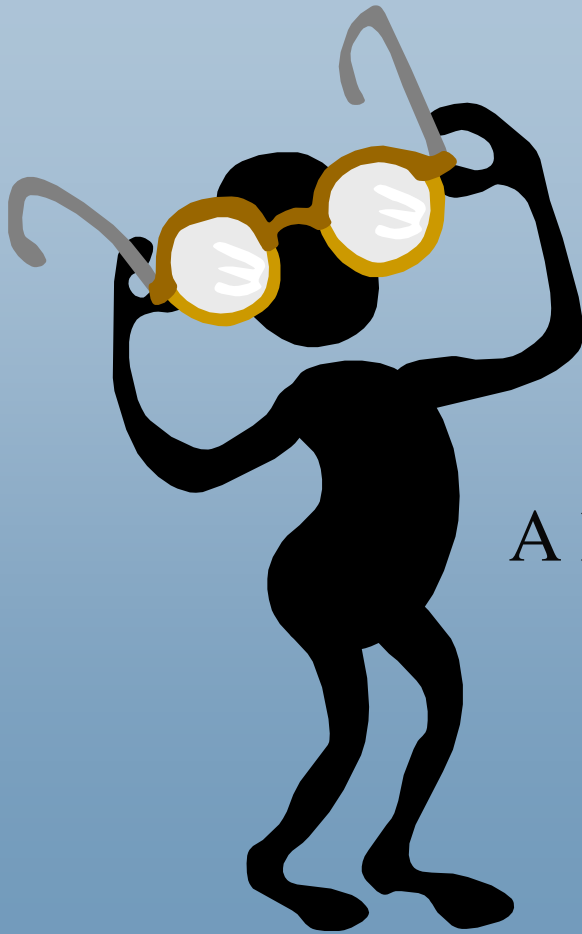
Relearning





If It's Not Value, It's Waste

Put on Customer Glasses



Handoffs



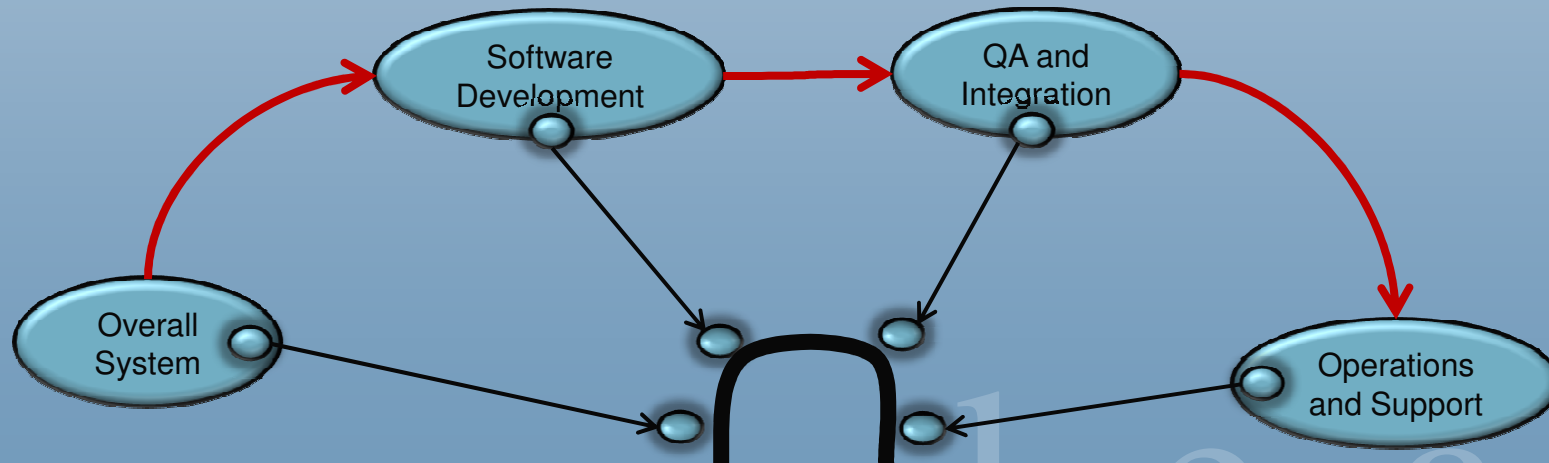
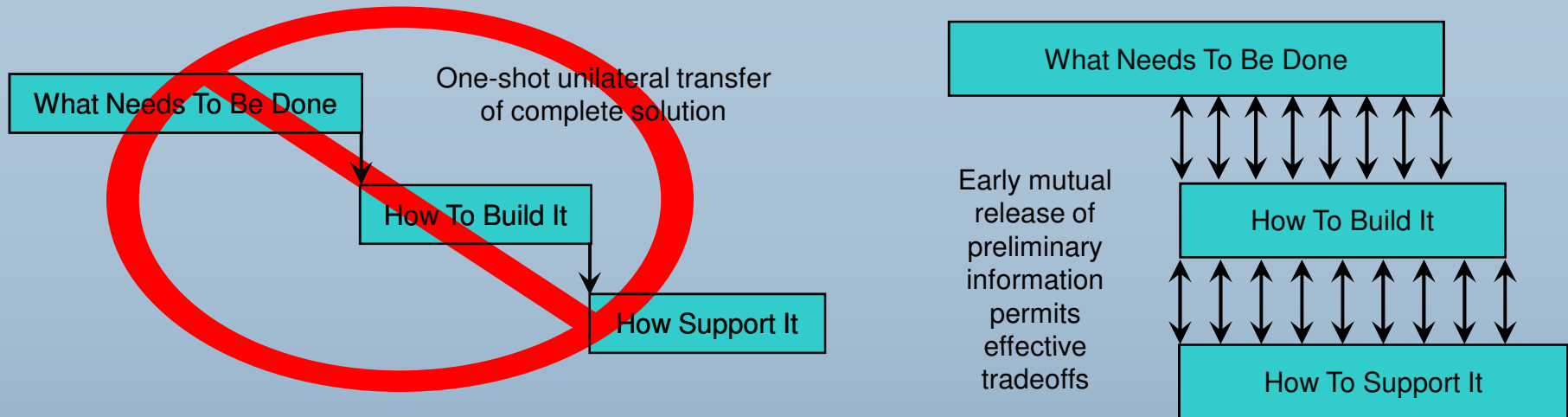
A hand-off occurs whenever we separate:*

- ✓ Responsibility – What to do
- ✓ Knowledge – How to do it
- ✓ Action – Actually doing it
- ✓ Feedback – Learning from results

*Allen Ward "Lean Product and Process Development"



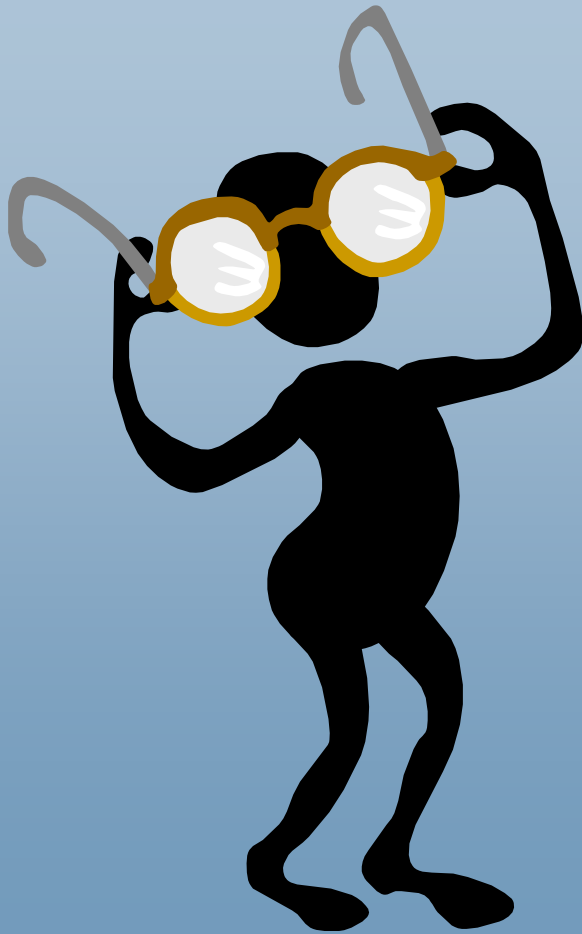
Cross-functional Teams



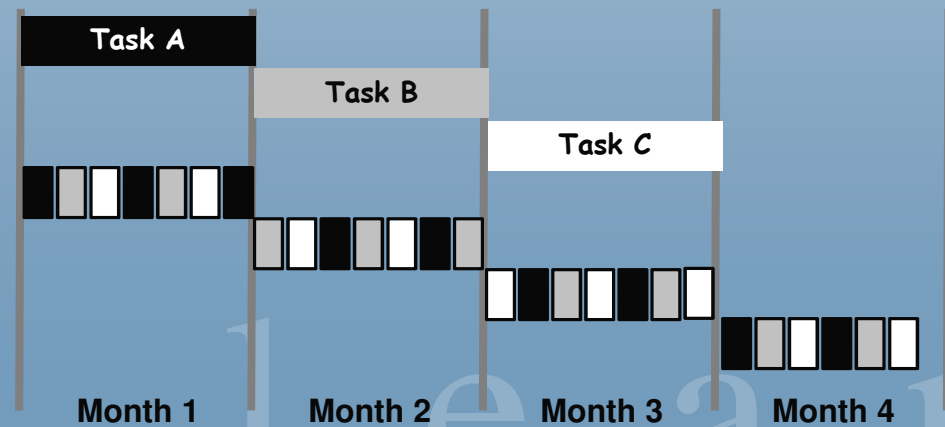


If It's Not Value, It's Waste

Put on Customer Glasses



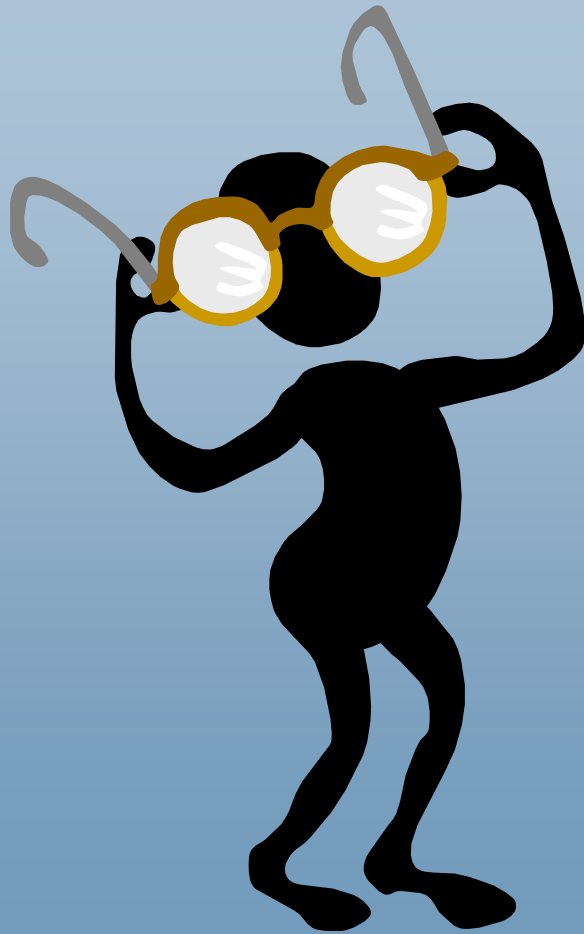
Task Switching



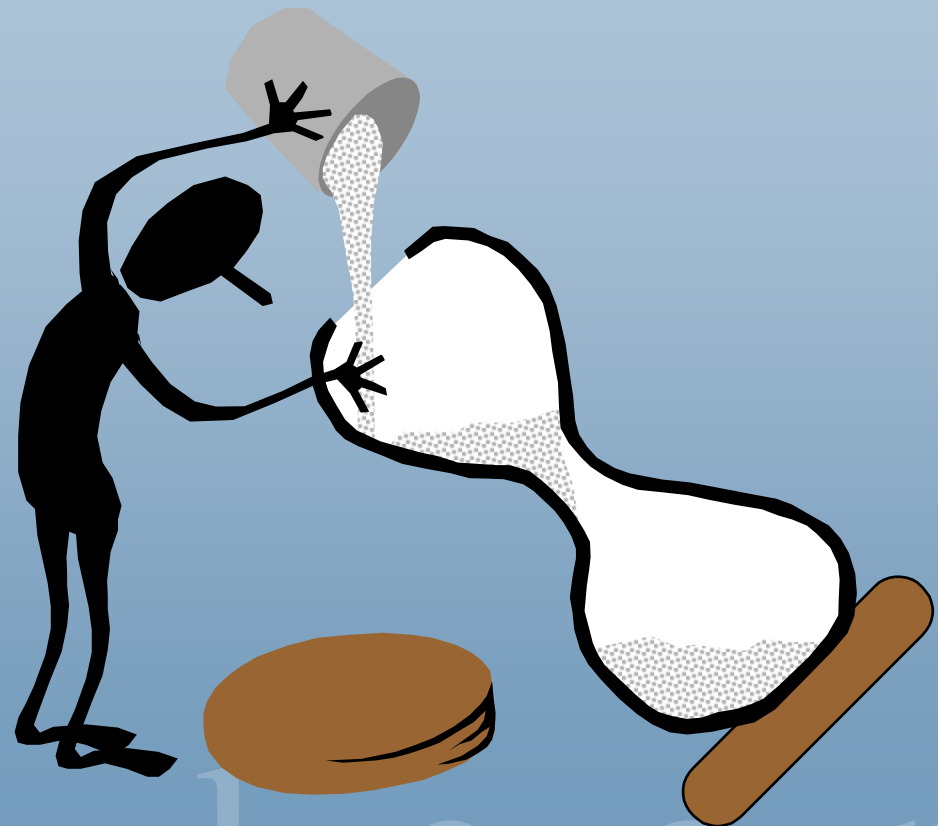


If It's Not Value, It's Waste

Put on Customer Glasses



Delays





Empire State Building

September 22, 1929
Demolition started
January 22, 1930
Excavation started
March 17, 1930
Construction started
November 13, 1930
Exterior completed
May 1, 1931
Building opened
Exactly on time
18% under budget



One Year Earlier:

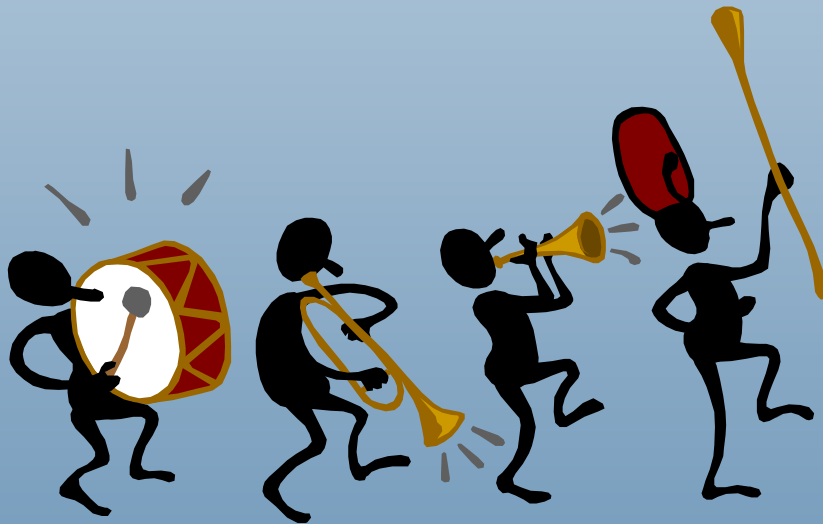


How did they do it? The key: Focus on FLOW.

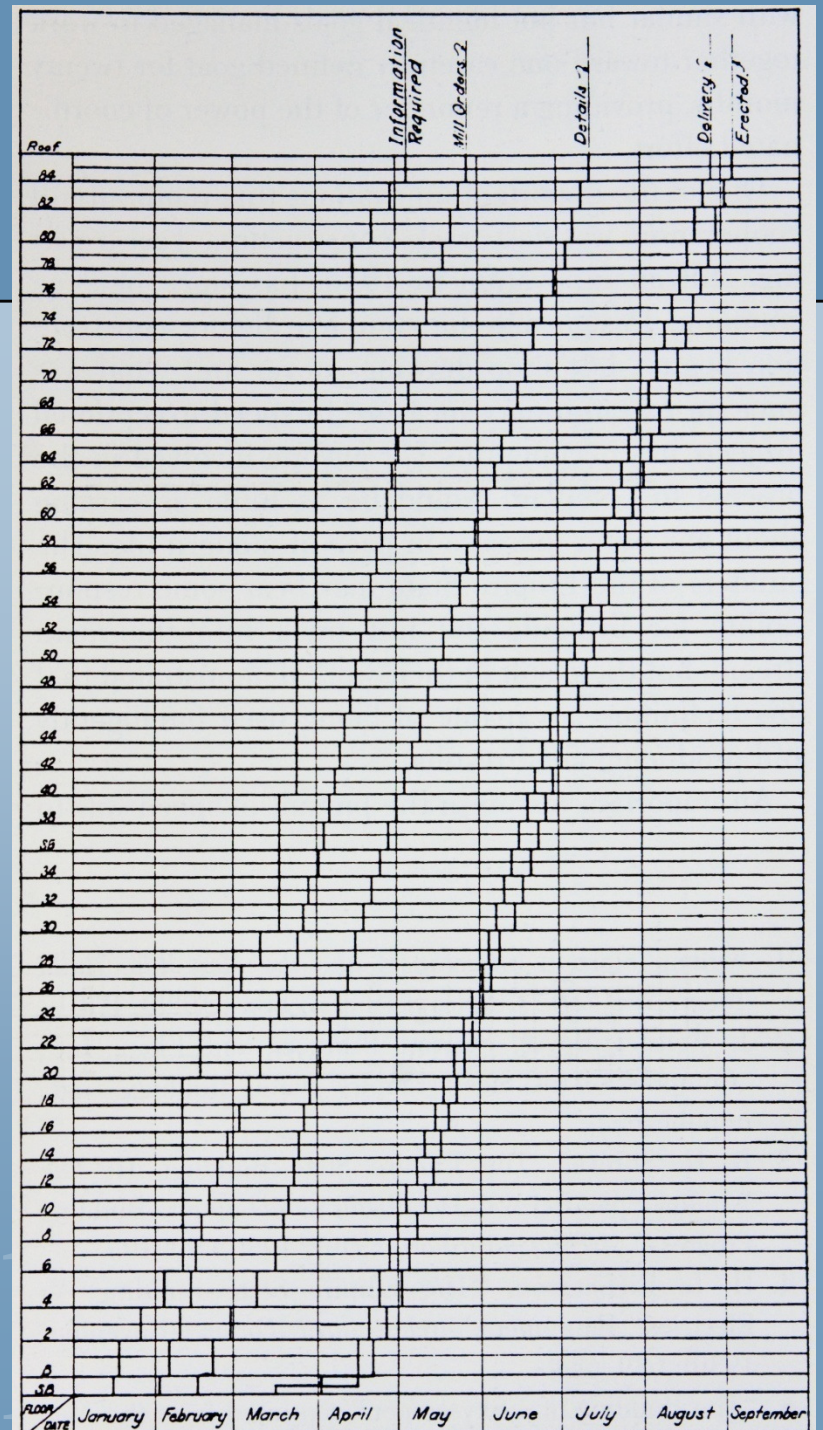


Steel Schedule

We thought of the work as if it were a band marching through the building and out the top.

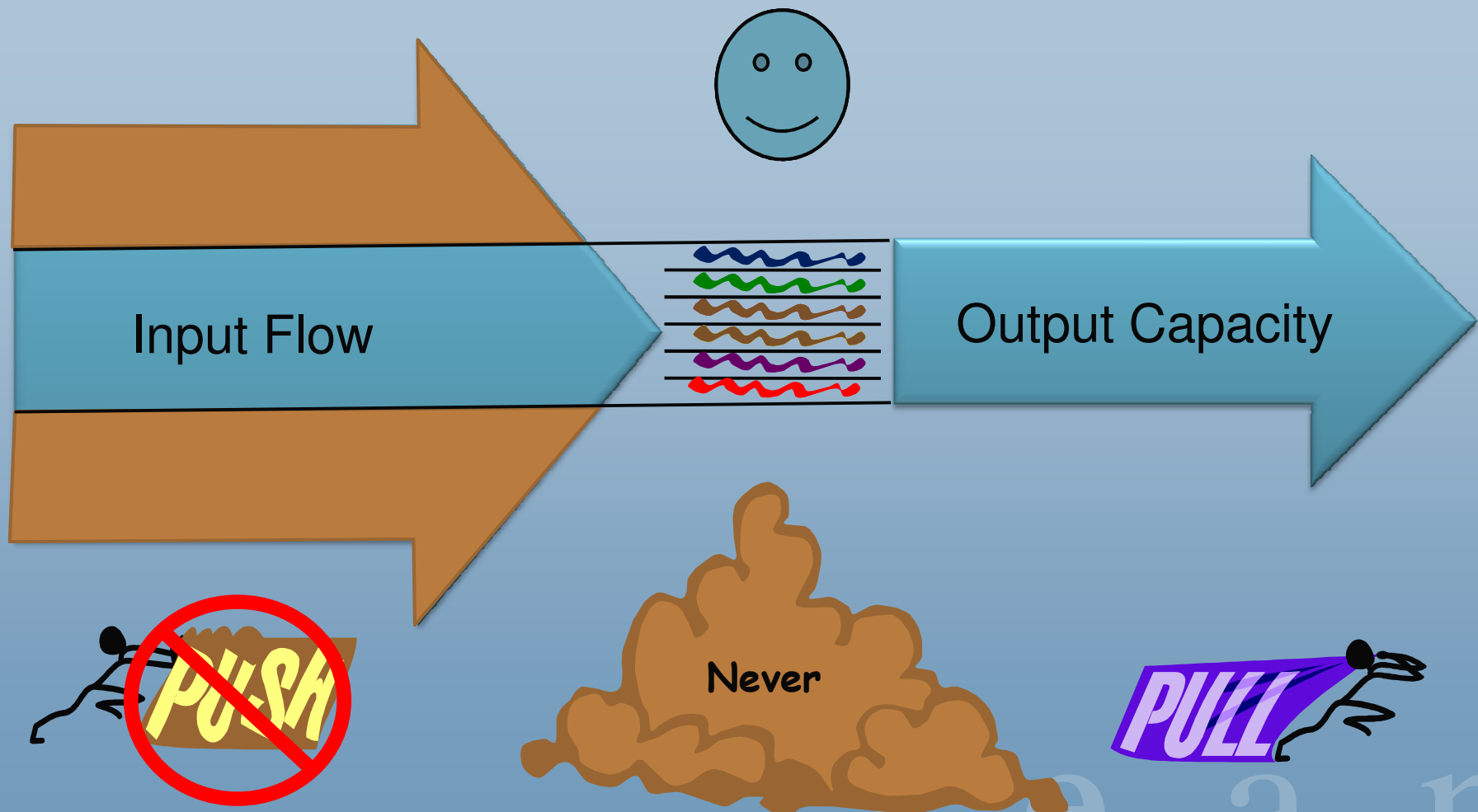


From: "Building the Empire State"
Builders Notebook: Edited by Carol Willis



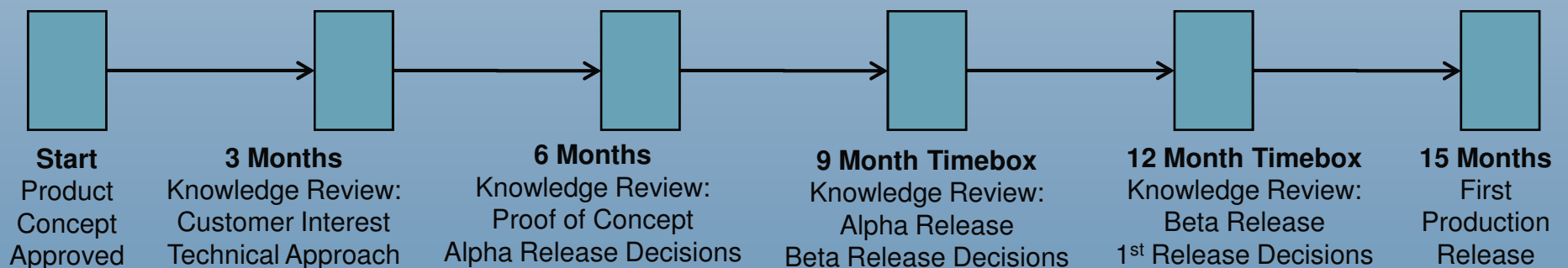
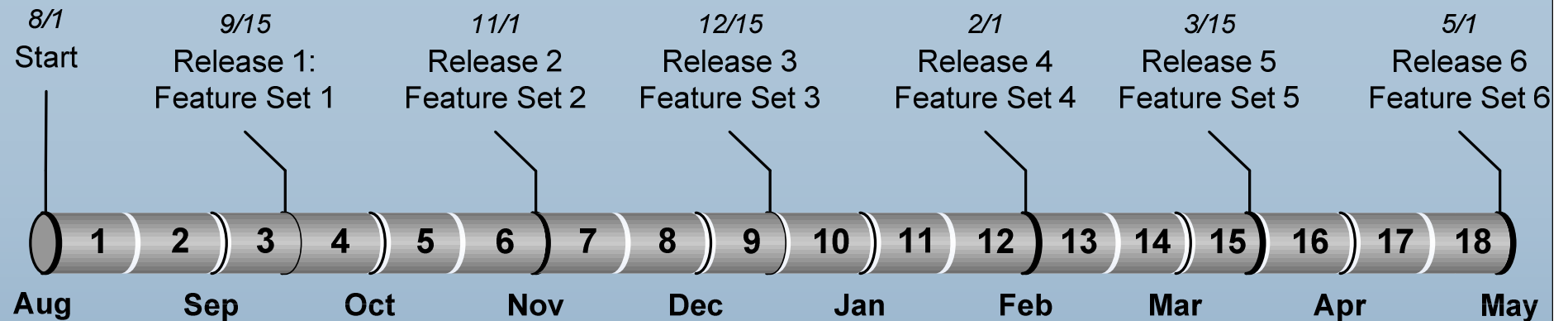


Pull Scheduling Small Requests





Pull Scheduling: Larger Systems



July 08

Copyright©2008 Poppendieck.LLC

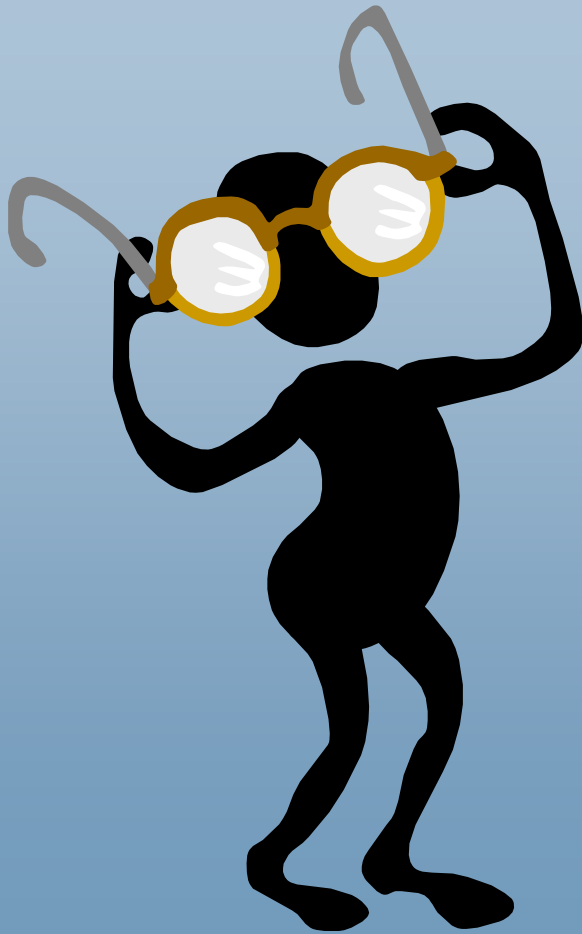
l e a n



If It's Not Value, It's Waste

Put on Customer Glasses

Defects

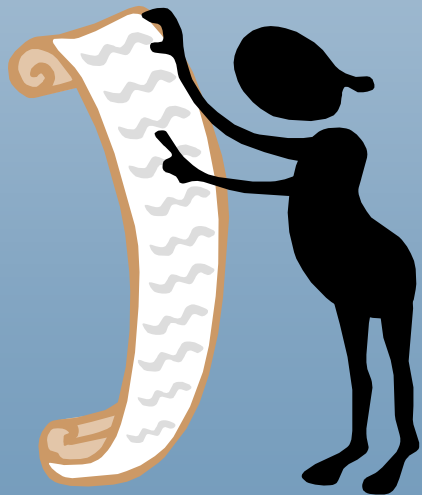




Don't Tolerate Defects

There are Two Kinds of Inspection

1. Inspection to Find Defects – WASTE
2. Inspection to Prevent Defects – Essential



Change The System

Mistake-Proof Every Step

- ✓ Detect defects the moment they occur

Don't track defects on a list

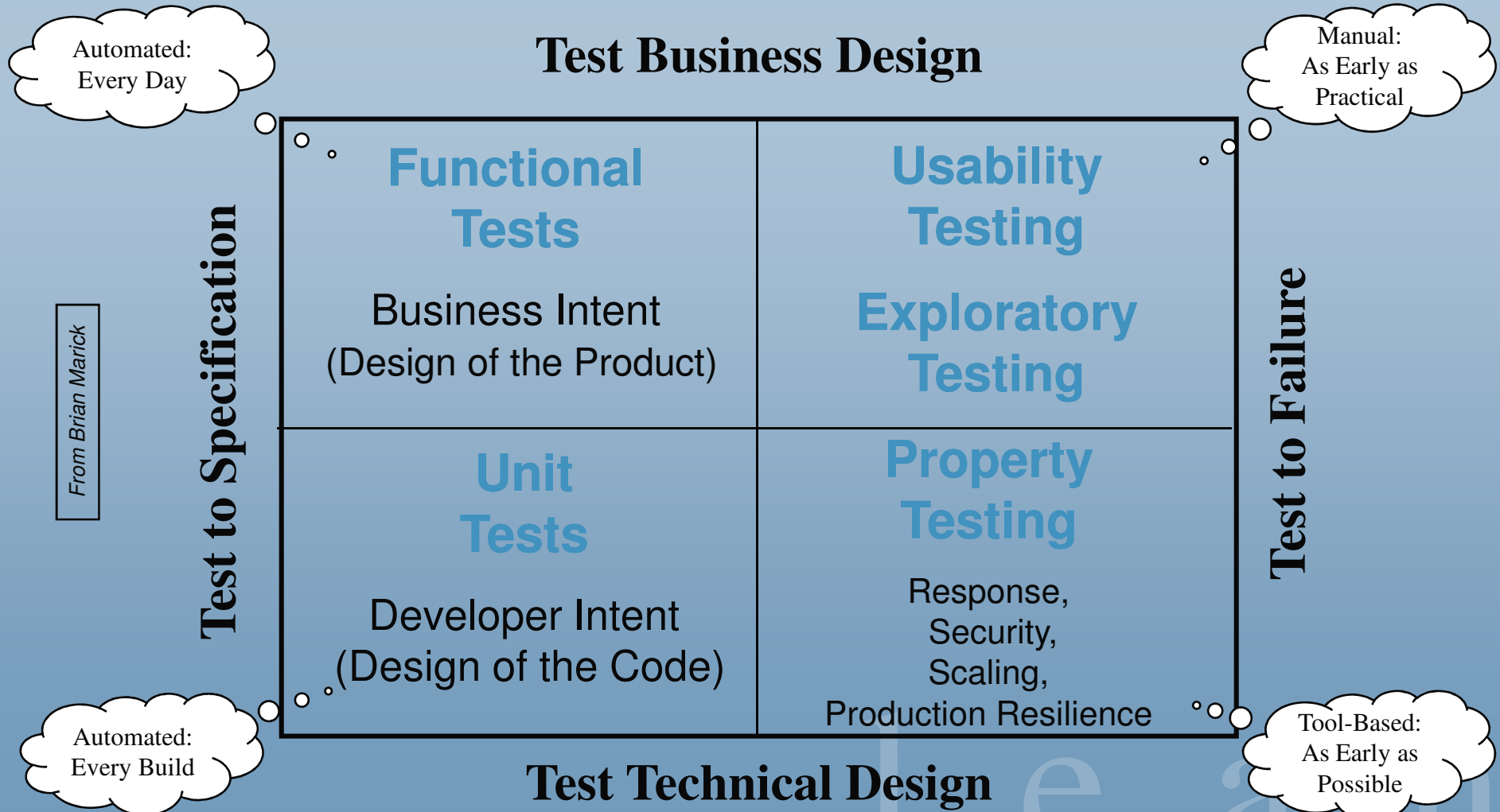
- ✓ Find them and fix them

Test FIRST

l e a n



Categories of Testing





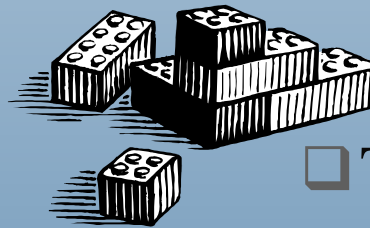
Building Block Disciplines

Mistake-Proofing

- ☐ Design/Code Reviews
- ☐ Configuration/Version Management
- ☐ One Click Build (Private & Public)
- ☐ Continuous Integration
- ☐ Automated Unit Tests
- ☐ Automated Functional Tests
- ☐ Production Test Harnesses
- ☐ **STOP** if the tests don't pass
- ☐ Automated Release Packages
- ☐ Frequent Releases

Simplicity

- ☐ Architecture
- ☐ Development
 - × Naming
 - × Coding
 - × Logging
 - × Security
 - × User Interaction
- ☐ Tools
 - × Code Checkers
 - × IDE's
- ☐ Refactoring
 - × Continuous Improvement of the code base.





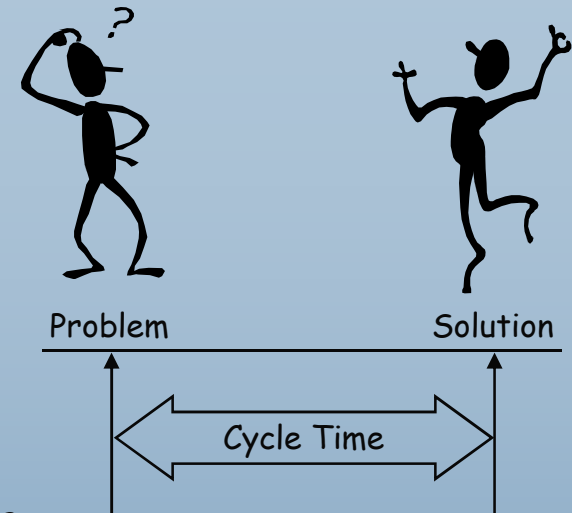
Seeing Waste

Value Stream Map

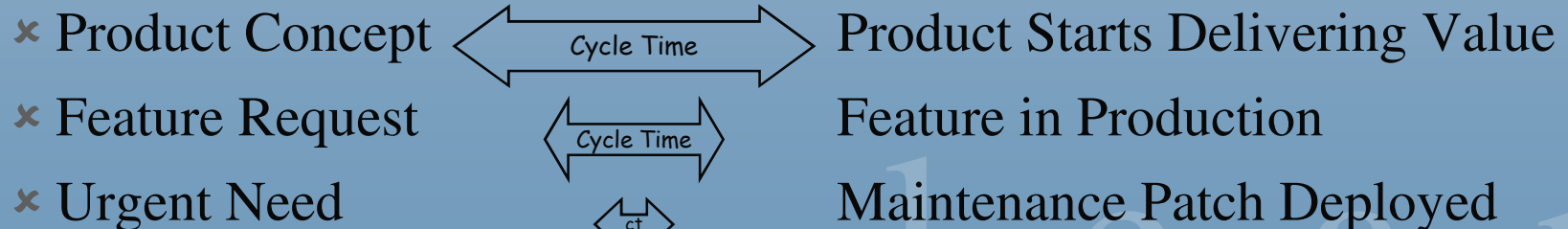
A diagnostic tool to find the biggest waste (opportunity for improvement) in a development process.

Development Process Capability:

- ✓ The reliable, repeatable cycle time from customer need until that need is satisfied.



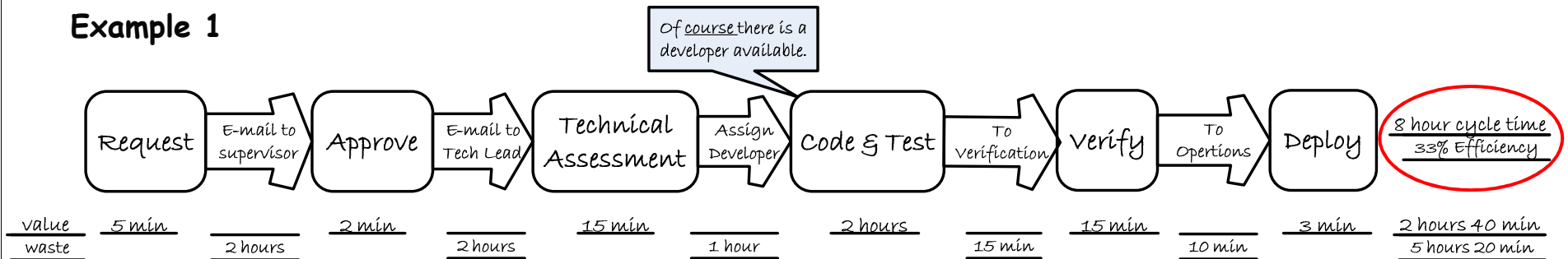
Begins and ends with the customer



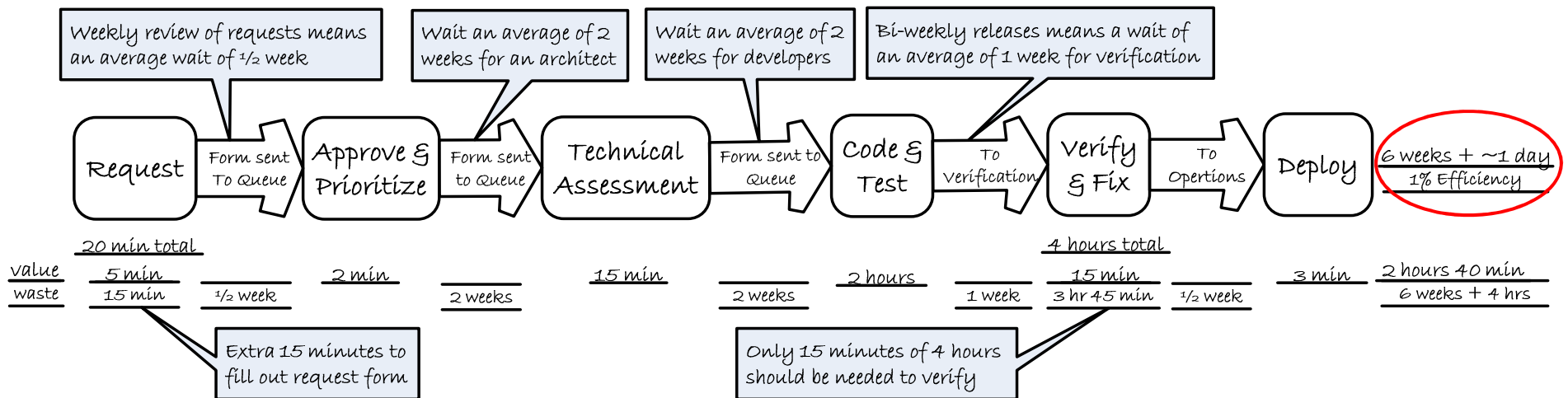


Value Stream Examples

Example 1



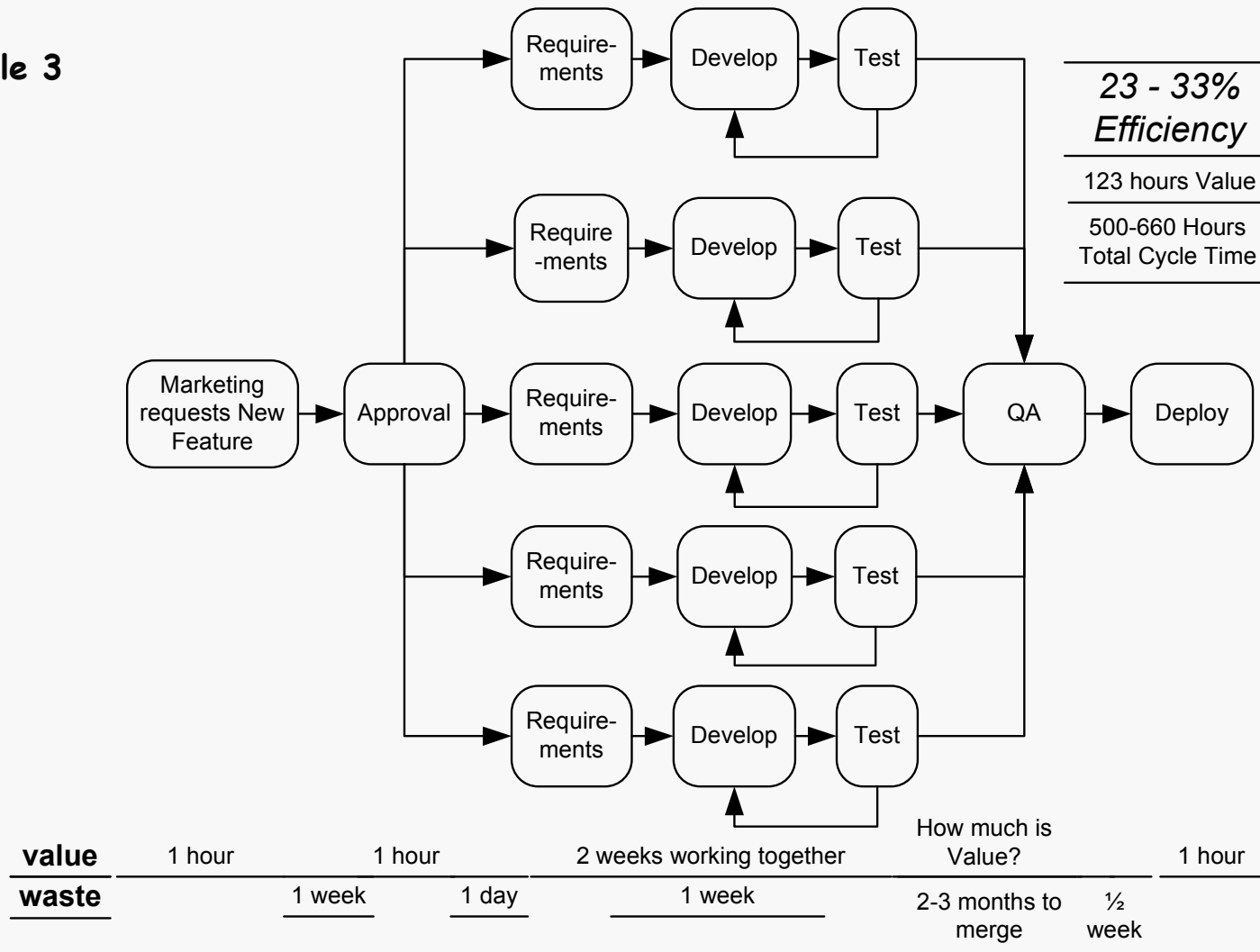
Example 2





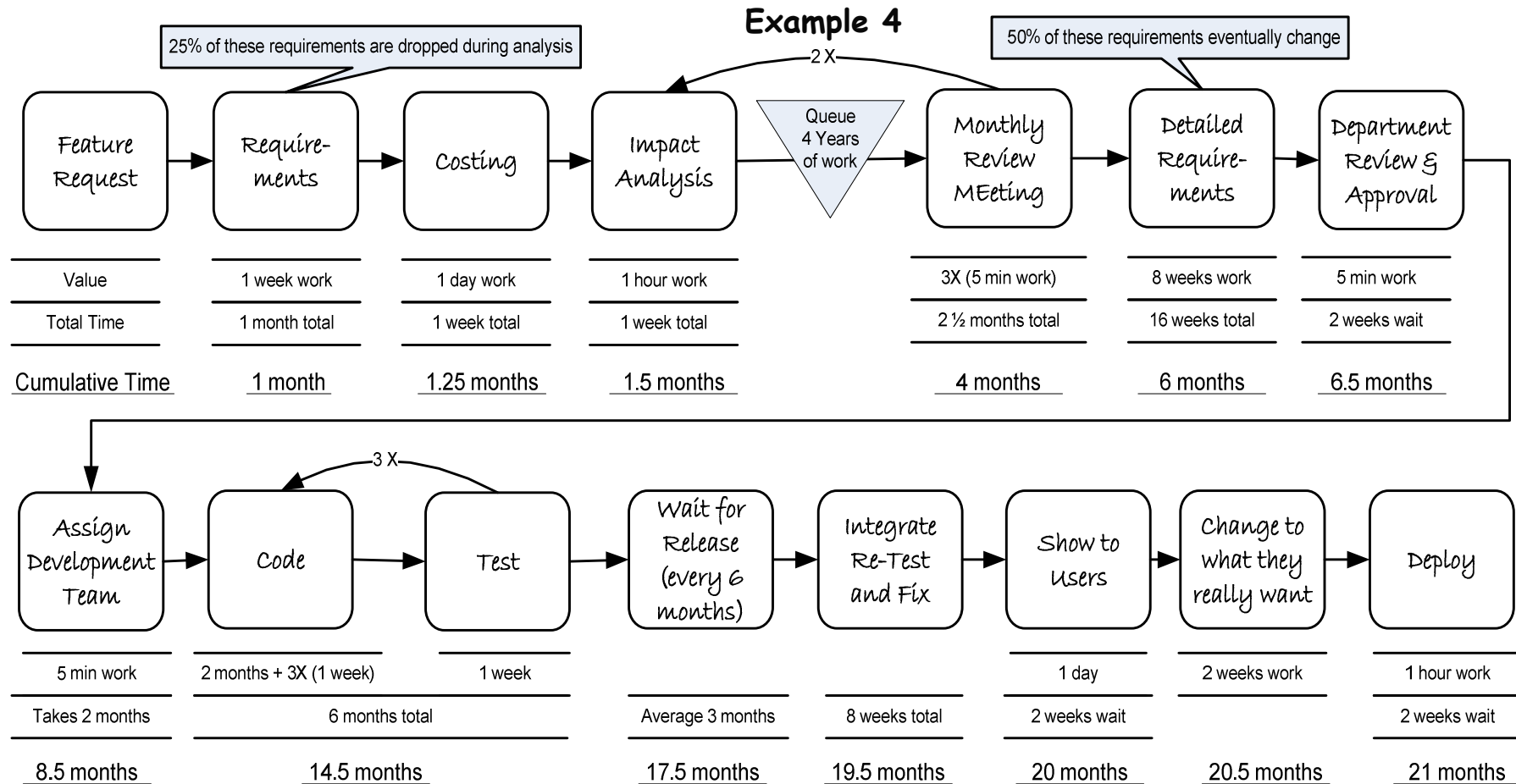
Value Stream Examples

Example 3





Value Stream Examples

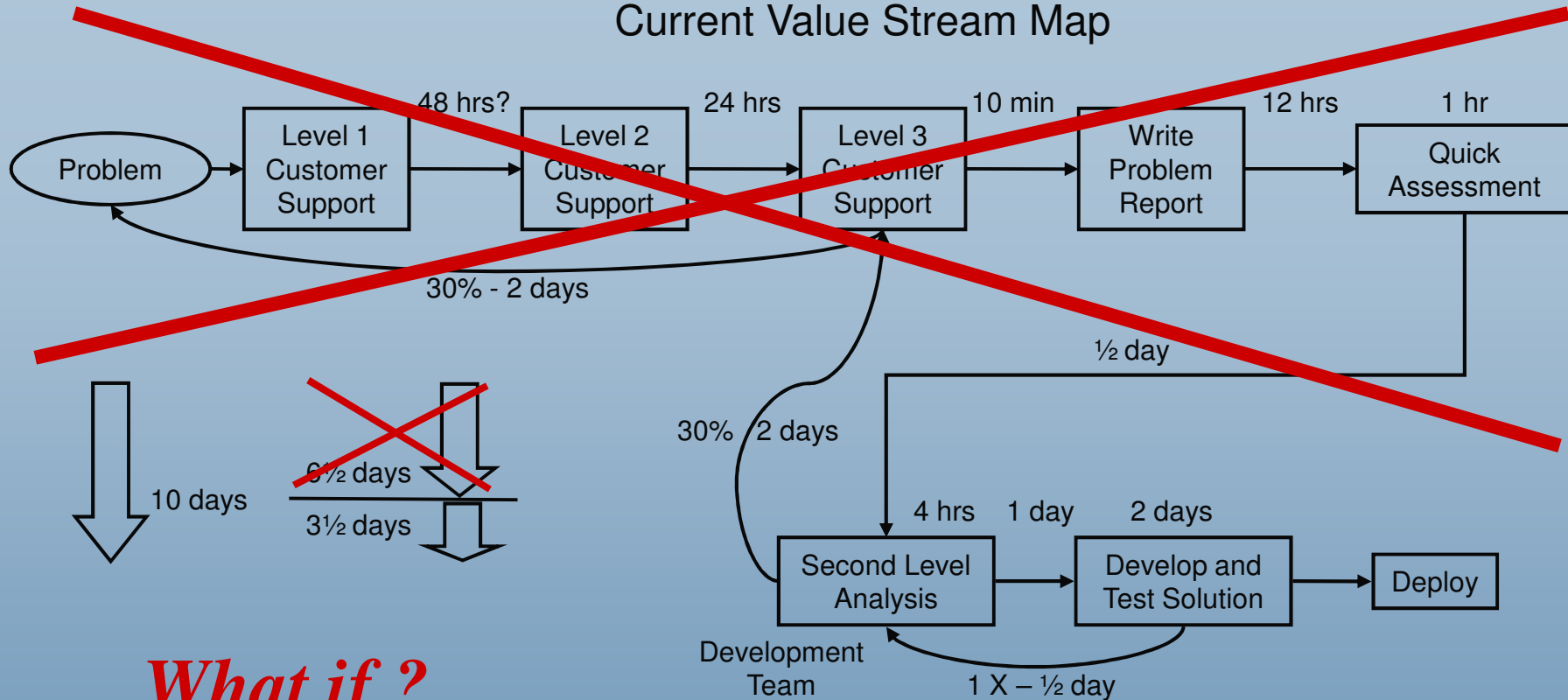


Question: What is really value added? What is the % efficiency?



Case Study: Critical Defects

Current Value Stream Map



l e a n



Case Study: Critical Defects

Future Value Stream Map

Questions:

Who will staff the phones?

- ✓ Developers – in rotation

How many will we need?

- ✓ Experiment – find out

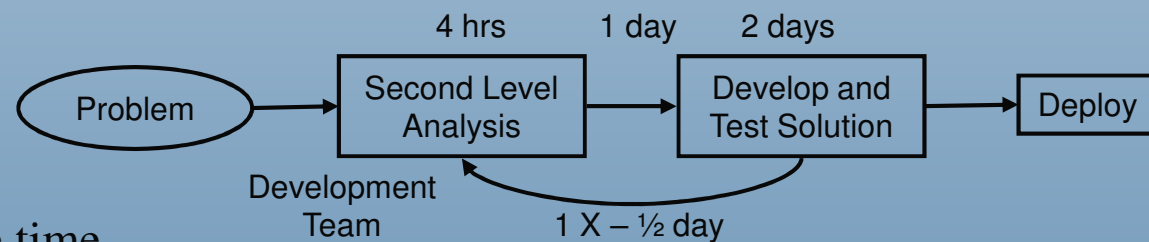
Two Rules:

1. Immediately after a release, responsible team takes calls.
2. Learning from each call **MUST** be recorded in knowledge base which is available to customers.

Why not?

Results:

- ✓ 65% faster response time
- ✓ 40% increase in available development time (for 800 developers)!
 - ✗ **Before:** 60% of development time spent on critical defects
 - ✗ **After:** 20% of development time spent on critical defects



Exercise: Current Value Stream Map



Select a process for creating a Value Stream Map. Decide when the clock starts (eg. customer has a need) and when it stops (need is filled).

Add up time of each step plus time between steps = Total Cycle Time

Add up Value Added Time in each step

Calculate Process Cycle Efficiency*

Value Added Time

Total Cycle Time

* George & Wilson, *Conquering Complexity in Your Business*

Current Value Stream Map

List / diagram the key steps

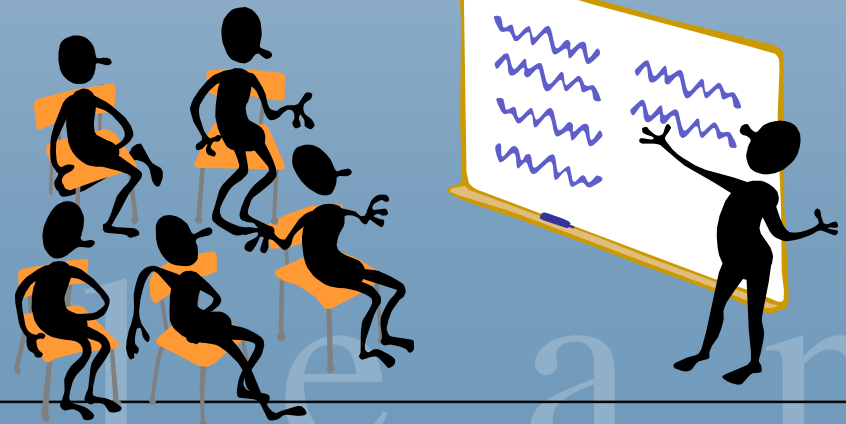
List the average time of each step

✓ Does the step add value full time?

✓ Is the step ever repeated?

List the average time between steps

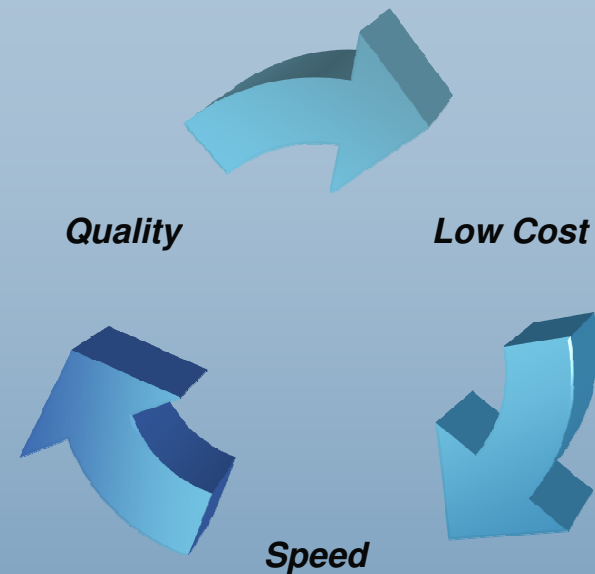
Report Back.





Principles of Lean Software Development

1. Eliminate Waste
2. Embed Quality
3. Learn First
4. Deliver Fast
5. Improve Forever
6. Respect People
7. Think System



l e a n



Queuing Theory



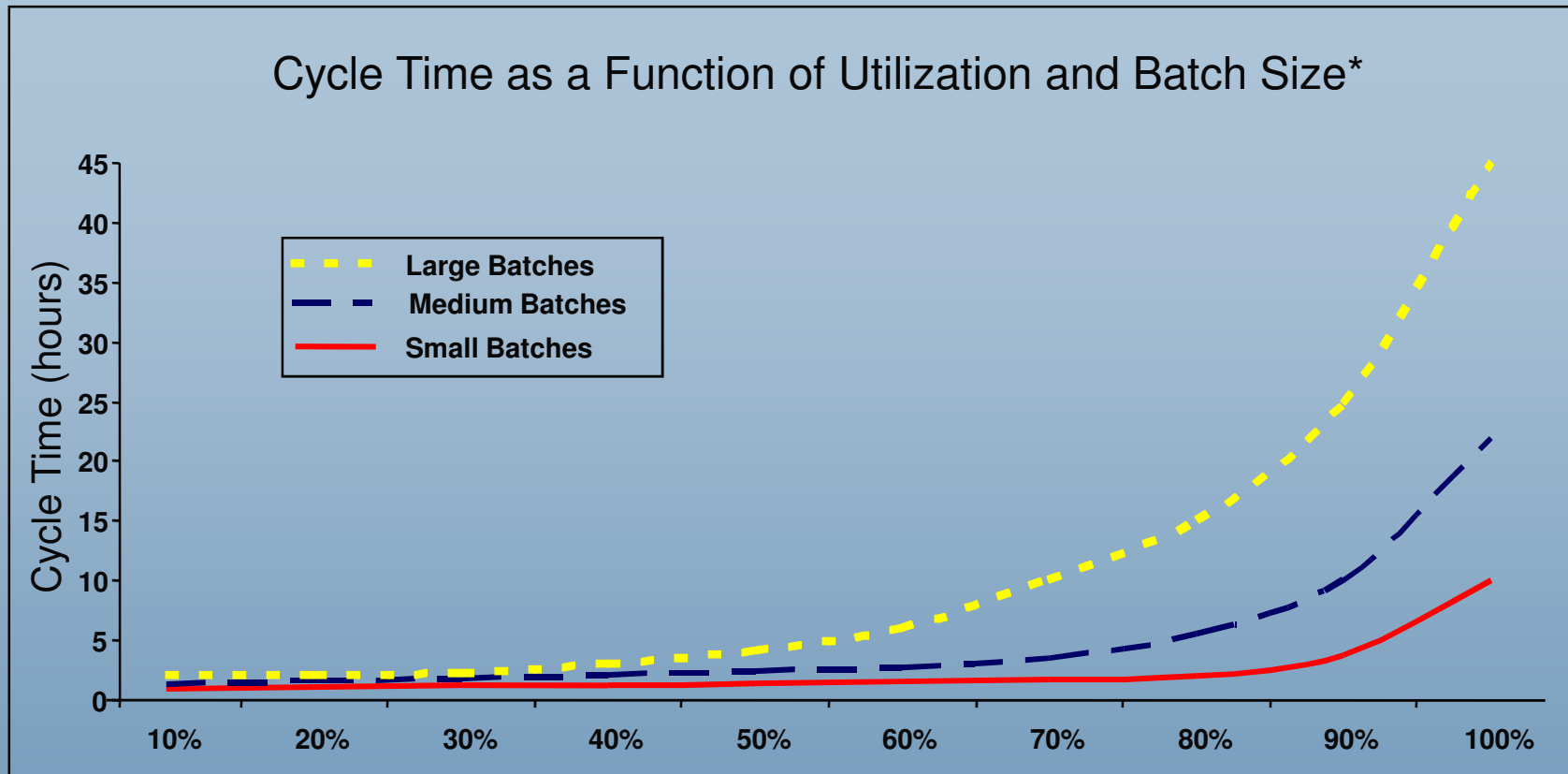
Little's Law

$$\text{Time Through the System} = \frac{\text{Number of Things in Process}}{\text{Average Completion Rate}}$$

l e a n



The Utilization Paradox



High Performance

Thrashing

*This assumes batch size is proportional to variability.



Reducing Cycle Time

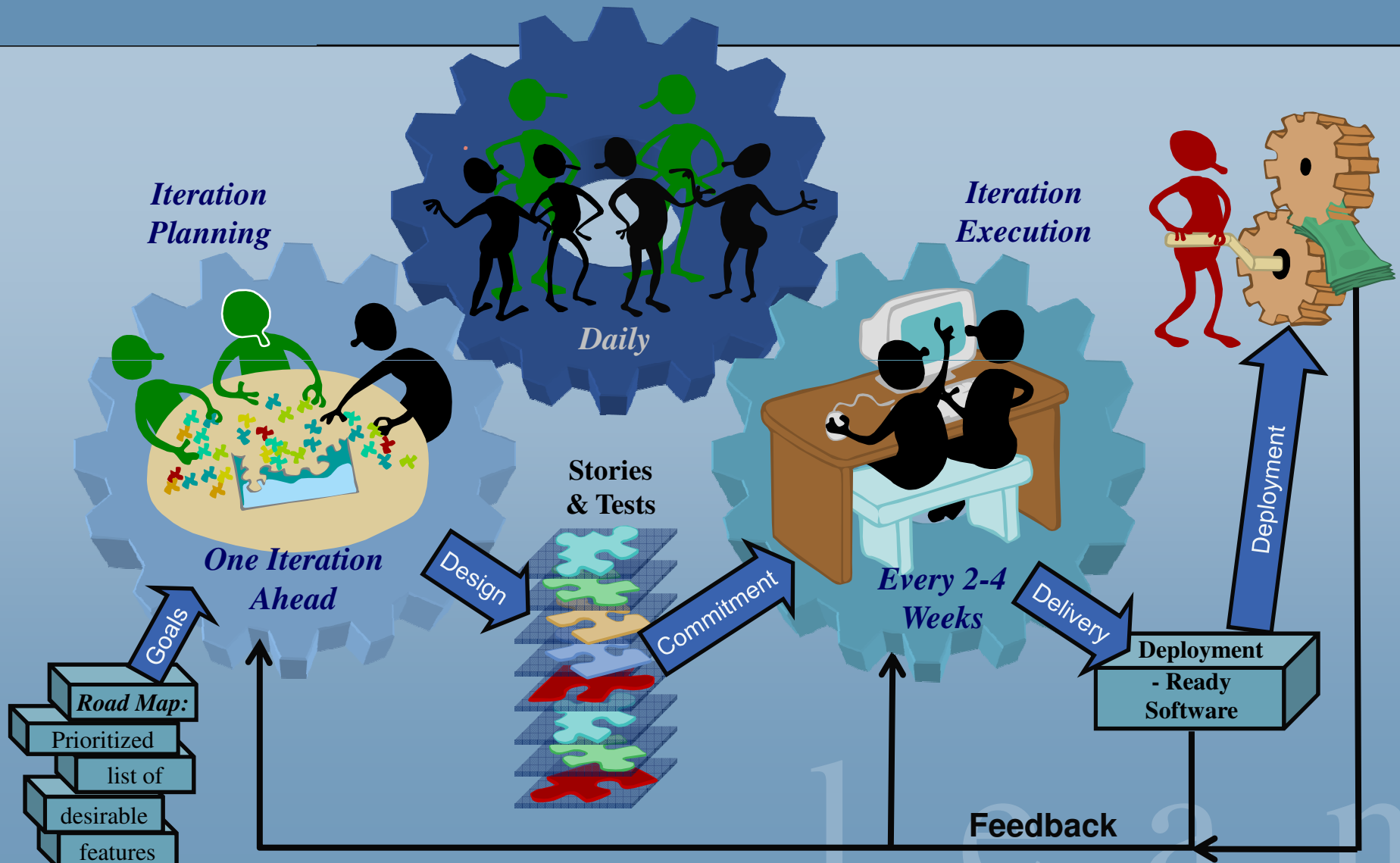
Level the Workload

- ✓ Even out the Arrival of Work
- ✓ Establish a Regular Cadence



Queuing Theory 101

Establish a Regular Cadence





Reducing Cycle Time

Level the Workload

- ✓ Even out the Arrival of Work
- ✓ Establish a Regular Cadence

Limit Work To Capacity

- ✓ Timebox, Don't Scopebox
- ✓ Pull – Don't Push



Queuing Theory 101



Timebox, Don't Scopebox



Ask NOT: How long will this take?

Ask instead: What can be done by this date?





Reducing Cycle Time

Level the Workload

- ✓ Even out the Arrival of Work
- ✓ Establish a Regular Cadence

Limit Work To Capacity

- ✓ Timebox, Don't Scopebox
- ✓ Pull – Don't Push

Optimize Throughput – Not Utilization

- ✓ Minimize the Size of Things In Process
- ✓ Minimize the Number of Things In Process

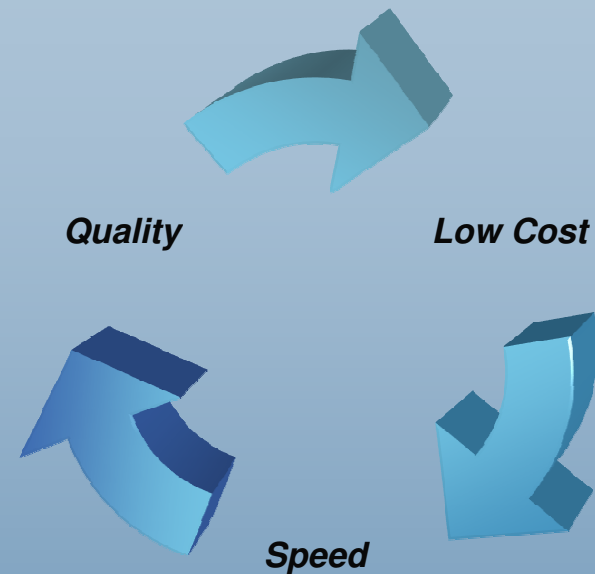


Queuing Theory 101



Principles of Lean Software Development

1. Eliminate Waste
2. Embed Quality
3. Learn First
4. Deliver Fast
5. Improve Forever
6. Respect People
7. Think System

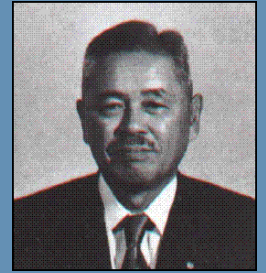


l e a n



Taiichi Ohno

Standard Work



When creating Standard Work, it will be difficult to establish a standard if you are trying to achieve ‘the best way.’ This is a big mistake. Document exactly what you are doing now. If you make it better than it is now, it is kaizen. If not, and you establish the best possible way, the motivation for kaizen will be gone.

That is why one way of motivating people to do kaizen is to create a poor standard. But don’t make it too bad. Without some standard, you can’t say ‘We made it better’ because there is nothing to compare it to, so you must create a standard for comparison. Take that standard, and if the work is not easy to perform, give many suggestions and do kaizen.

From *Workplace Management*, by Taiichi Ohno, originally published in 1982, from translation by Jon Miller, Gemba Press, 2007.



Relentless Improvement

KAIZEN

Kai

改

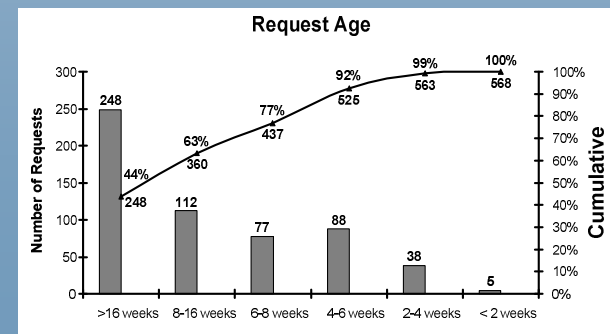
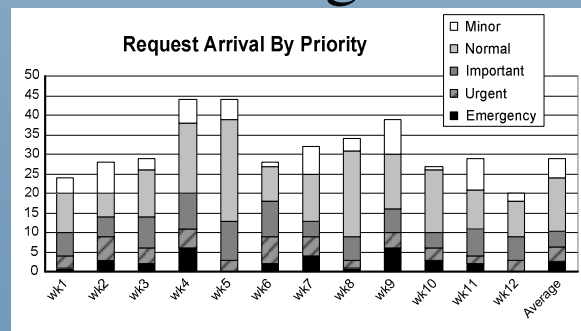
Change

Zen

善

Good

1. Solve One Problem at a Time
2. Use Data-Based Problem Analysis
 - ✓ Pareto Analysis
 - ✗ Find the most important problem
 - ✓ Root Cause Analysis
 - ✗ Cause & Effect Diagram
 - ✗ Five Why's
3. Try Many Quick Experiments
4. Measure & Change Based on Results
5. Do it Again





lean

software development

Thank You!

More Information: www.poppendieck.com

Mary & Tom Poppendieck