



l e a n
software development

Lean Software Development

Tutorial



Lean Software Development

Introduction

- ✓ History

Eliminate Waste

- ✓ Value Stream Maps
- ✓ Find & Fit the Need

Build Quality In

- ✓ Mistake-Proofing

Defer Commitment

- ✓ Set-Based Design

Deliver Fast

- ✓ Queuing Theory

Respect People

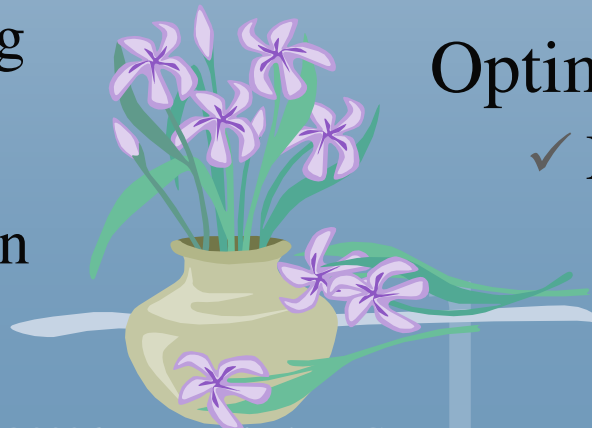
- ✓ Teams & Expertise

Focus on Learning

- ✓ Scientific Method

Optimize the Whole

- ✓ Measurements





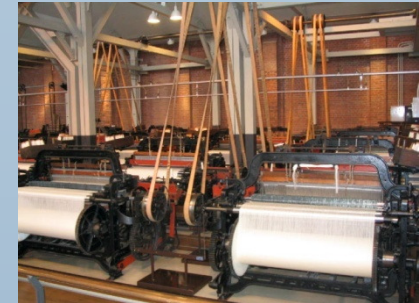
The Toyoda's



Sakichi Toyoda
(1867-1930)

Sakichi Toyoda (1867-1930)

- ✓ Extraordinary inventor of automated looms
- ✓ Crucial Idea: *Stop-the-Line*



Kiichiro Toyoda
(1894-1952)

Kiichiro Toyoda (1894-1952)

- ✓ Bet the family fortune on automotive manufacturing
- ✓ Crucial Idea: *Just-in-Time*



Eiji Toyoda
(1913-Present)

Eiji Toyoda (1913-Present)

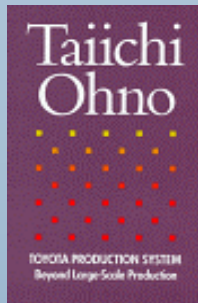
- ✓ 50 years of Toyota leadership
- ✓ Championed the development of *The Toyota Production System*





The Toyota Production System

Taiichi Ohno



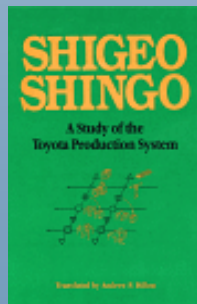
The Toyota Production System, 1988 (1978)

- ✓ Eliminate Waste
 - ✗ Just-in-Time Flow
- ✓ Expose Problems
 - ✗ Stop-the-Line Culture



Taiichi Ohno
(1912-1990)

Shigeo Shingo



Study Of 'Toyota' Production System, 1981

- ✓ Non-Stock Production
 - ✗ Single Digit Setup
- ✓ Zero Inspection
 - ✗ Mistake-Proof Every Step

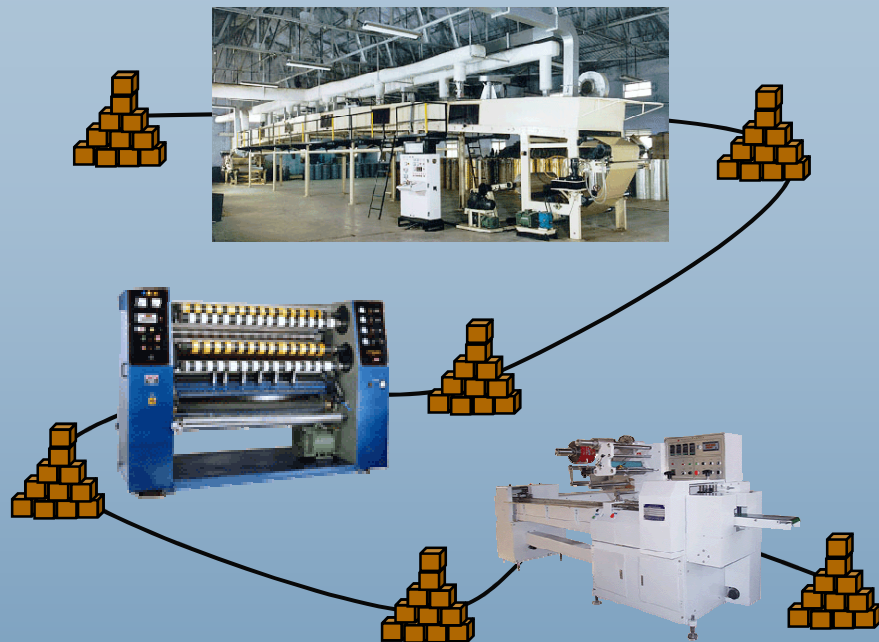


Shigeo Shingo
(1909 – 1990)



Pillars of Lean

Just-in-Time Flow



Stop trying to maximize local productivity.

Stop-the-Line Culture



Find and fix problems the moment they occur.



Don't Batch & Queue

Lists

- ✓ How long is your defect list?
- ✓ How far apart are your releases?
- ✓ How many weeks (years?) of work do you have in your backlog?



Churn

- ✓ If you have requirements churn, you are specifying too early.
- ✓ If you have test-and-fix cycles, you are testing too late.





Don't Tolerate Defects

There are Two Kinds of Inspection

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



The Role of QA

The job of QA is not to swat misquotes,
The job of QA is to put up screens.

A quality process builds quality into the code

- ✓ If you routinely find defects during verification
– your process is defective.





Lean Software Development

Introduction

- ✓ History

Eliminate Waste

- ✓ Value Stream Maps
- ✓ Find & Fit the Need

Build Quality In

- ✓ Mistake-Proofing

Defer Commitment

- ✓ Set-Based Design

Deliver Fast

- ✓ Queuing Theory

Respect People

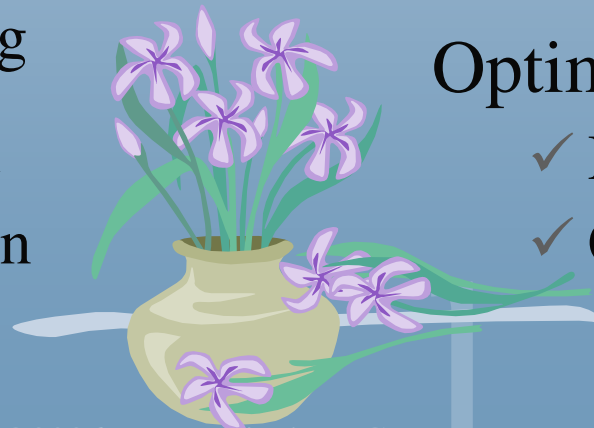
- ✓ Teams & Expertise

Focus on Learning

- ✓ Scientific Method

Optimize the Whole

- ✓ Measurements
- ✓ Contracts





Principle 1: Eliminate Waste

$$\text{Price} = \text{Cost} + \text{Profit}$$
$$\text{Profit} = \text{Price} - \text{Cost}$$



Taiichi Ohno

Waste is anything that has been started
but is not being used in production.

Extra features that are not
needed *now* are waste.

Waste is making the wrong thing
or making the thing wrong.



Complexity Kills

*The Three Faces of Complexity**

Waste

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

– *Keep it Simple*

Inconsistency

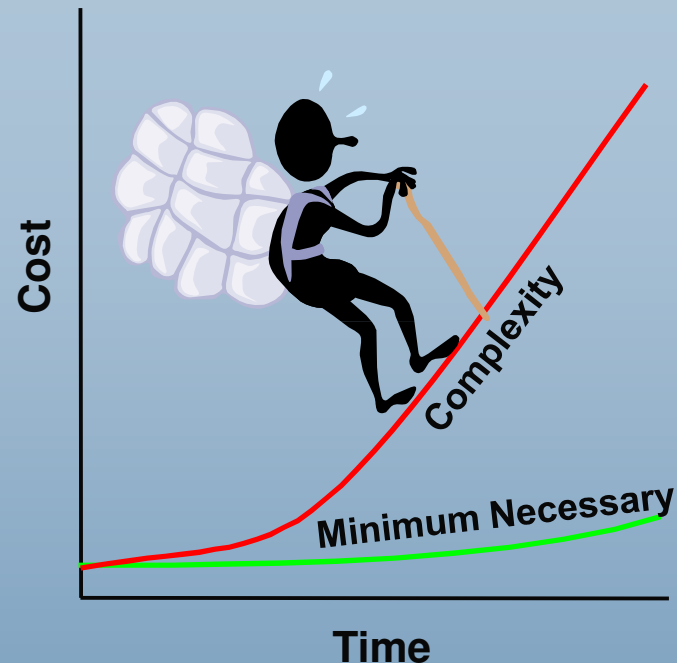
Uneven, unbalanced, or irregular

– *Make it Flawless*

Overload

Excessive or unreasonable burden

– *Let it Flow*



**The Elegant Solution: Toyota's Formula for Mastering Innovation* by Matthew May, 2006



The Seven Wastes

The Seven Wastes of Manufacturing - Shigeo Shingo

1. Inventory
2. Overproduction
3. Extra Processing
4. Motion
5. Transportation
6. Waiting
7. Defects

MUDA
anything that
does not add
VALUE



The Seven Wastes

The Seven Wastes of Manufacturing - Shigeo Shingo

1. Inventory
2. Overproduction
3. Extra Processing
4. Motion
5. Transportation
6. Waiting
7. Defects

The 7 Wastes of Software Development

1. Partially Done Work
2. Extra Features

"Every line of code costs money to write and more money to support. It is better for the developers to be surfing than writing code that won't be needed. If they write code that ultimately is not used, I will be paying for that code for the life of the system, which is typically longer than my professional life. If they went surfing, they would have fun, and I would have a less expensive system and fewer headaches to maintain."

-- Jeff Sutherland, CTO PatientKeeper



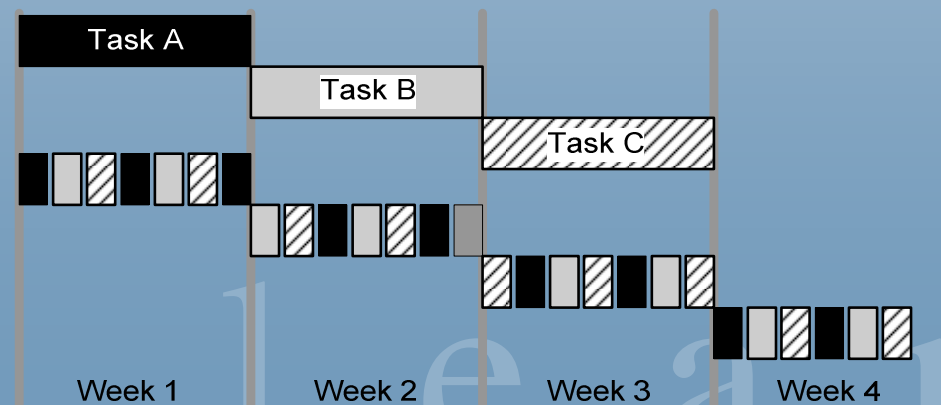
The Seven Wastes

The Seven Wastes of Manufacturing - Shigeo Shingo

1. Inventory
2. Overproduction
3. Extra Processing
4. Motion
5. Transportation
6. Waiting
7. Defects

The 7 Wastes of Software Development

1. Partially Done Work
2. Extra Features
3. Relearning
4. Task Switching





The Seven Wastes

The Seven Wastes of Manufacturing - Shigeo Shingo

1. Inventory
2. Overproduction
3. Extra Processing
4. Motion
5. Transportation
6. Waiting
7. Defects

The 7 Wastes of Software Development

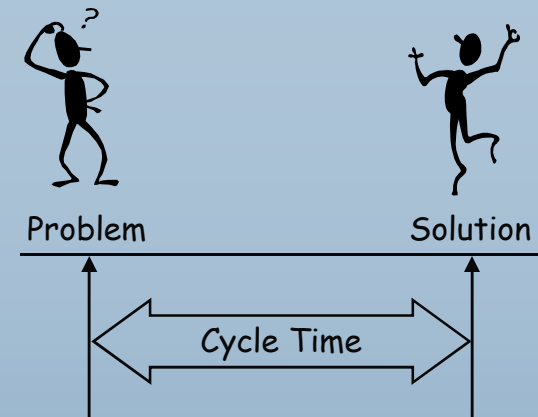
1. Partially Done Work
2. Extra Features
3. Relearning
4. Task Switching
5. Handoffs
6. Delays
7. Defects



Cycle Time

Cycle Time

- ✓ Average end-to-end process time
 - ✗ From problem detection
 - ✗ To problem solution



Begins and ends with the customer

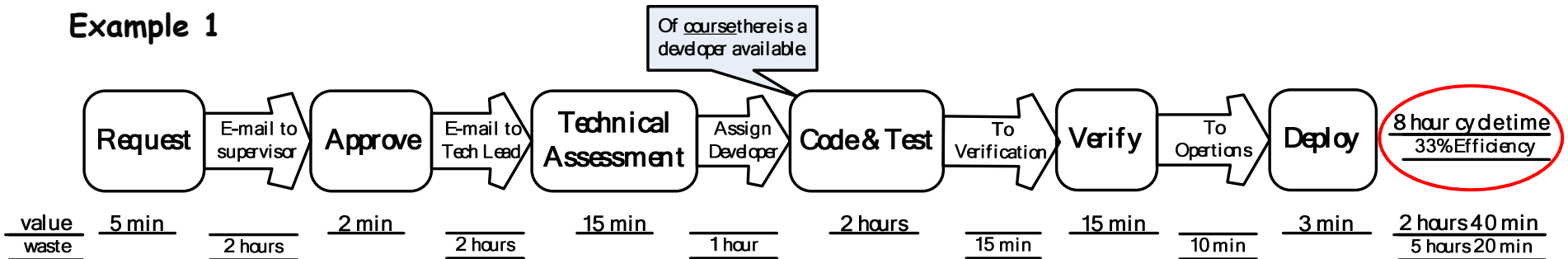
Software Development Cycle Time

- ✓ The speed at which a customer need is reliably and repeatedly translated into deployed software.

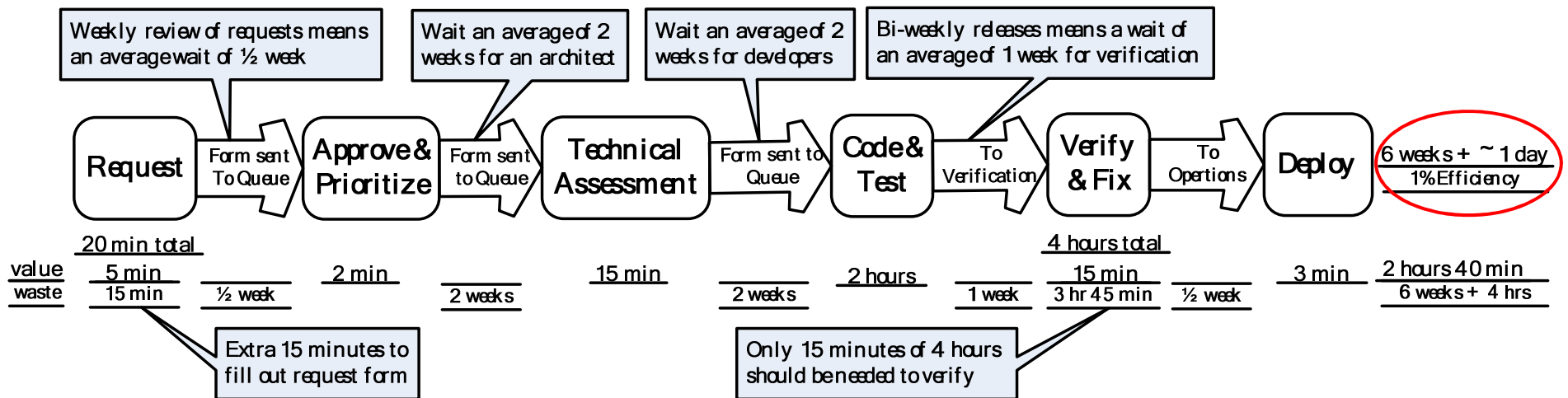


Value Stream Examples

Example 1



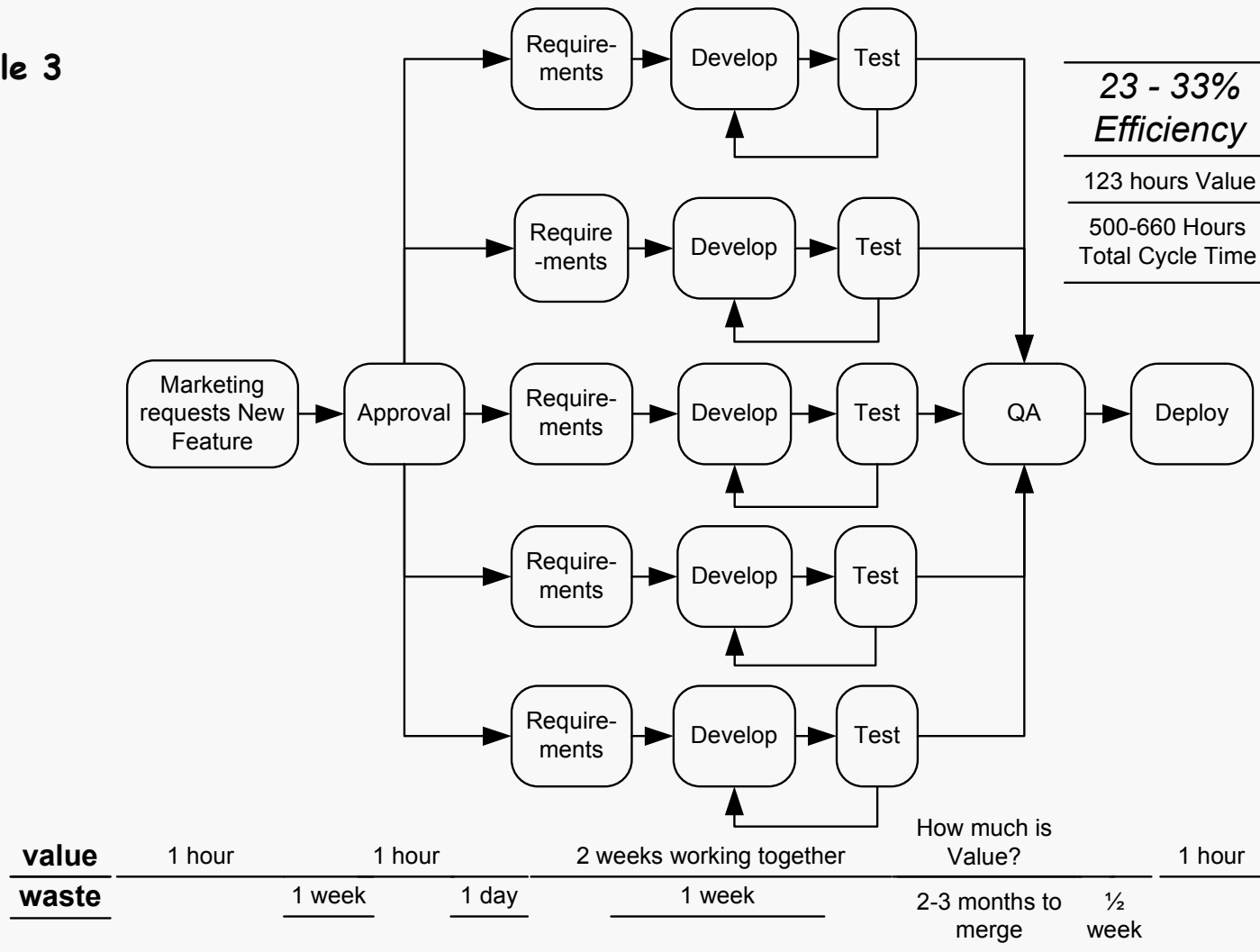
Example 2





Value Stream Examples

Example 3



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).

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

Calculate Process Cycle Efficiency*

Value Added Time

Total Cycle Time

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

Add up Value Added Time in each step

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

Report Back.





Lean Software Development

Introduction

- ✓ History

Eliminate Waste

- ✓ Value Stream Maps
- ✓ Find & Fit the Need

Build Quality In

- ✓ Mistake-Proofing

Defer Commitment

- ✓ Set-Based Design

Deliver Fast

- ✓ Queuing Theory

Respect People

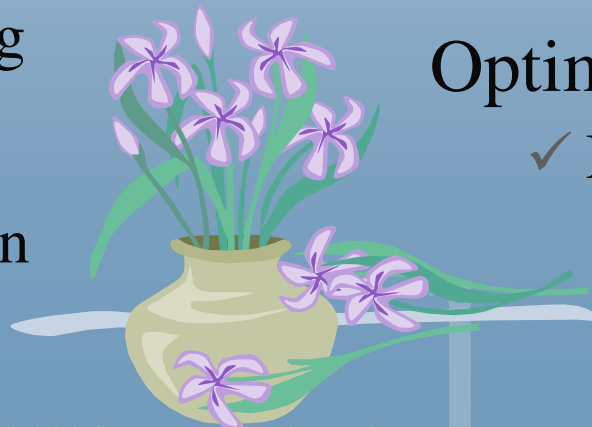
- ✓ Teams & Expertise

Focus on Learning

- ✓ Scientific Method

Optimize the Whole

- ✓ Measurements





Think Products, not Projects

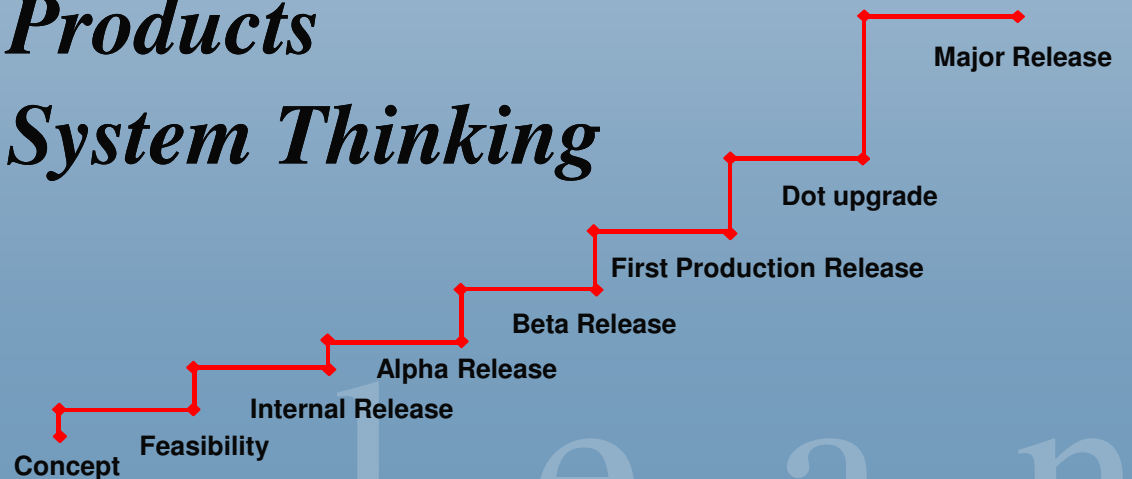
Up-front funding
Scope fixed at onset
Success = cost/schedule/scope
Team often disbands at completion

Projects



Incremental funding
Scope expected to evolve
Success = profit/market share
Team often stays with product

Products *System Thinking*



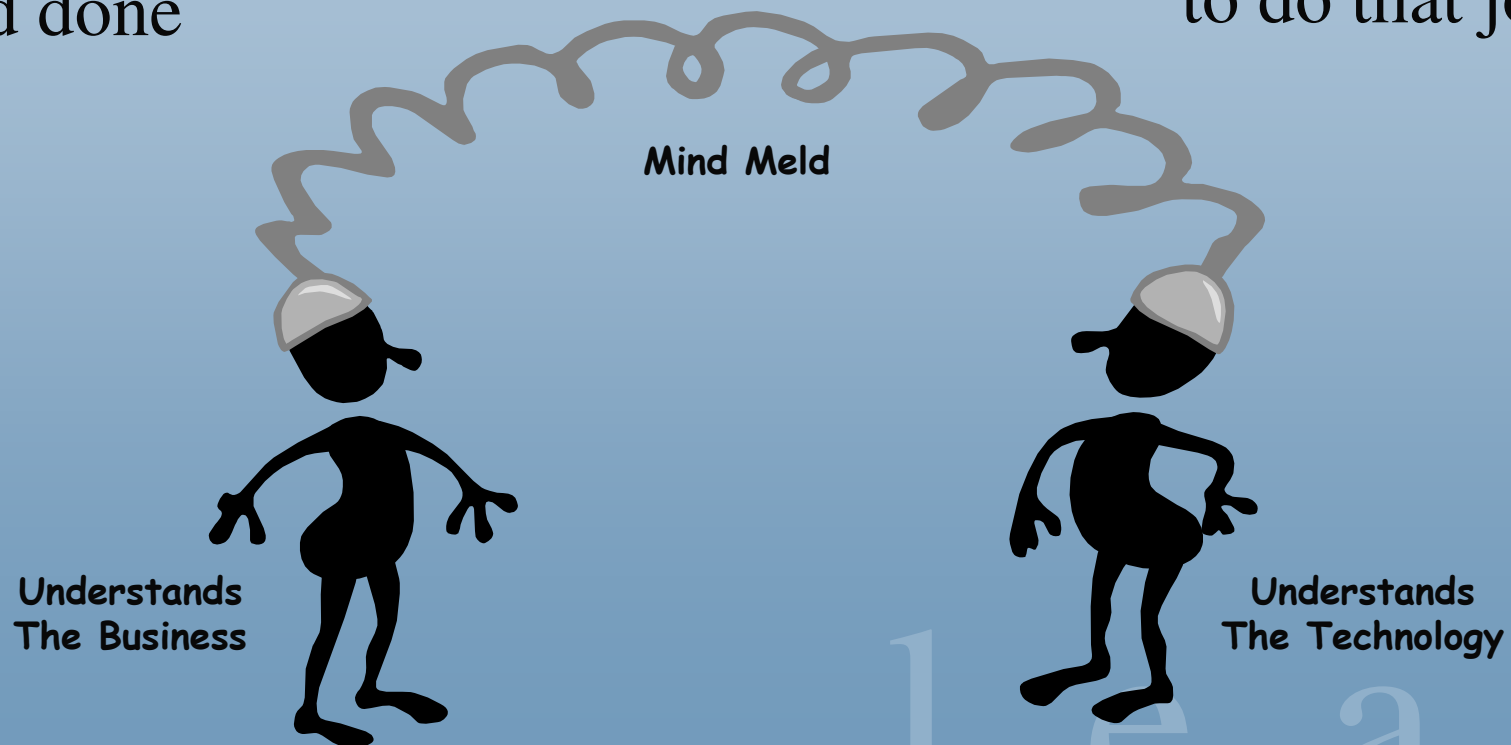


Brilliant Products

Embody a Deep Understanding of:

The job that customers
need done

The right technology
to do that job





Product Champion

Behind every great product is a person with:

- ✓ Great empathy for the customer
- ✓ Insight into what is technically possible
- ✓ The ability to see what is essential and what is incidental

Example: Chief Engineer at Toyota

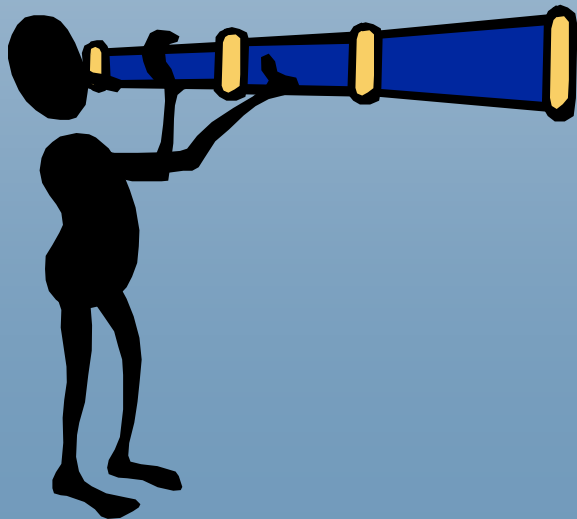
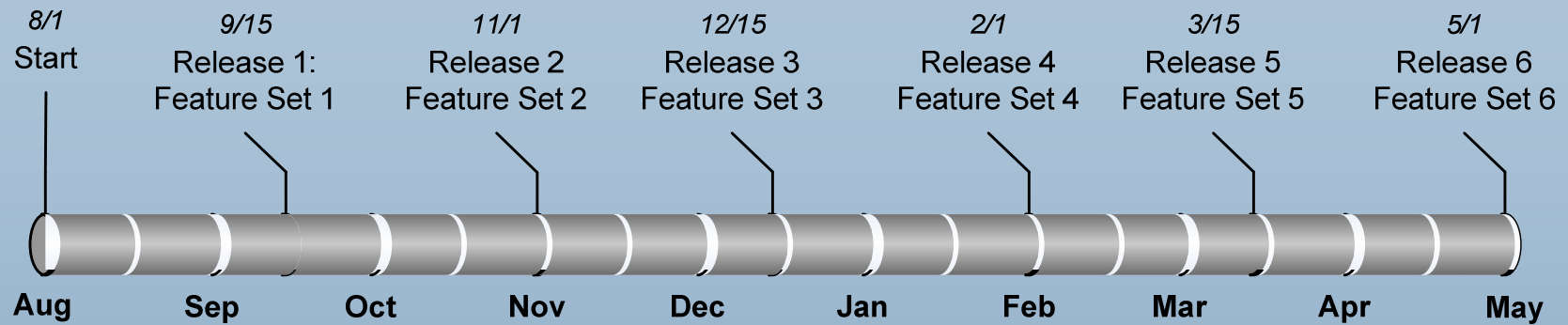
- ✓ Responsible for Business Success
- ✓ Develops Deep Customer Understanding
- ✓ Develops the Product Concept
- ✓ Creates The High Level System Design
- ✓ Sets the Schedule
- ✓ Understands what customers will value and conveys this to the engineers making day-to-day tradeoffs
- ✓ Arbitrates trade-offs when necessary
- ✓ Defends the Vision



l e a n



Marketing Leader Product Road Map



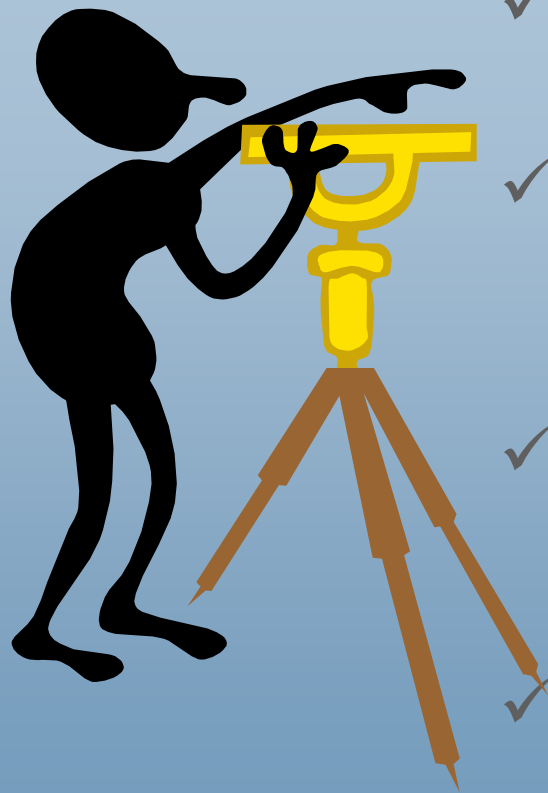
Release Planning

- ✓ Ballpark estimates
- ✓ Minimum useful feature sets
- ✓ Highest priority / greatest risk first



Technical Leader Architectural Vision

The Role of Systems Design (Architecture):



- ✓ Provide a foundation for growth
 - ✗ Create a common infrastructure
- ✓ Enable incremental development
 - ✗ Minimize dependencies
 - ✗ Modularize potential change
- ✓ Create space for teams to innovate
 - ✗ Design, code and test are different aspects of the same job and must be done concurrently
- ✓ Leave room for the future
 - ✗ Evolve the architecture over time



Leadership Roles

(How Many People?)

Marketing Leader

Business Responsibility
Customer Understanding
Release Planning
Tradeoffs

Technical Leader

System Architecture

- ✓ At a high level
- ✓ Work daily with those developing the details

Technical Guidance

- ✓ Integration
- ✓ Tradeoffs

Process Leader

Build Block Disciplines
Iterative Development
Visible Workspace

Project Leader

Funding
(Scheduling)
Tracking

Functional Leader

Staffing
Teaching
Standards

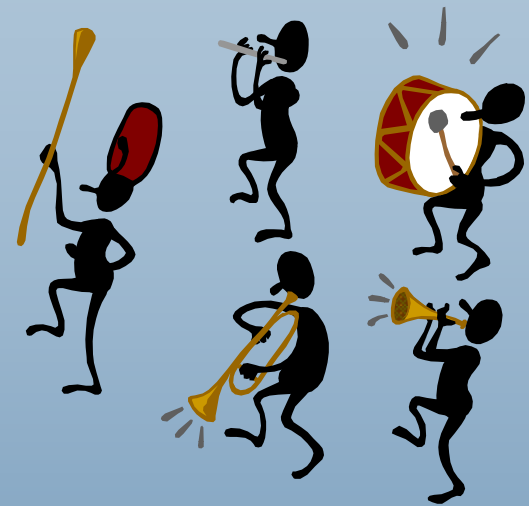


Complete Products

Complete products come from complete teams

Customers / Representatives
Business Analysts
User Interaction Designers
Testers/QA
Technical Writers
Operations
Support Desk

Product Champion
Marketing Leader
Technical Leader
Product Developers
Software Developers
DBA's
ETC.



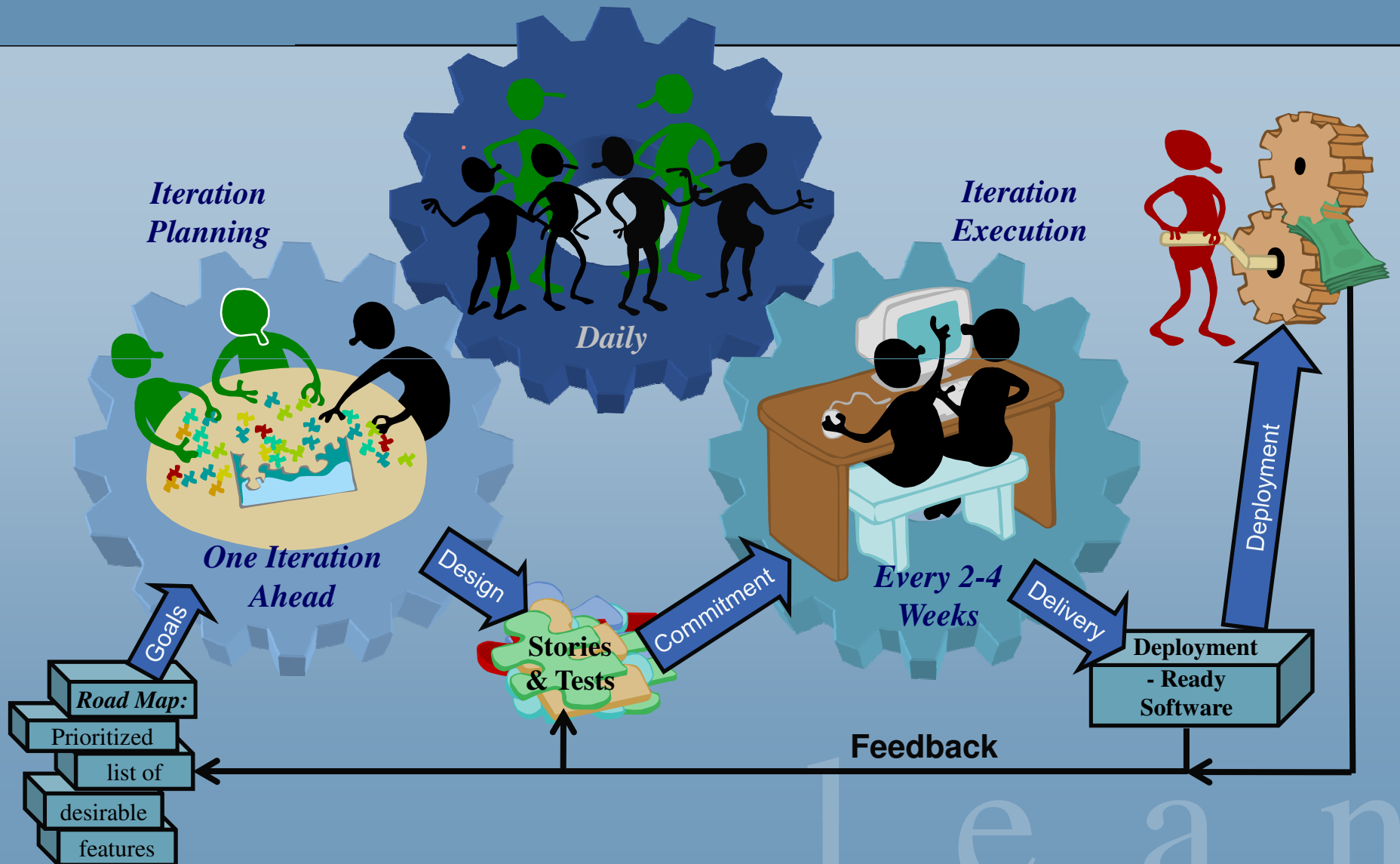
Deep Customer Understanding

- ✓ How does the current process really work?
- ✓ What problems does people struggle with every day?

Broad Technical Communication

- ✓ From Design to Operations

Iterative Development





Lean Software Development

Introduction

- ✓ History

Eliminate Waste

- ✓ Value Stream Maps
- ✓ Find & Fit the Need

Build Quality In

- ✓ Mistake-Proofing

Defer Commitment

- ✓ Set-Based Design

Deliver Fast

- ✓ Queuing Theory

Respect People

- ✓ Teams & Expertise

Focus on Learning


- ✓ Scientific Method

Optimize the Whole

- ✓ Measurements



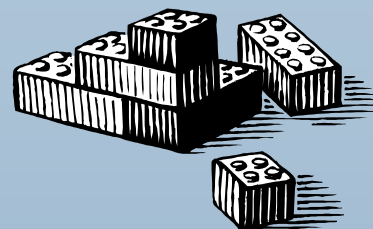
l e a n



Principle 2: Build Quality In

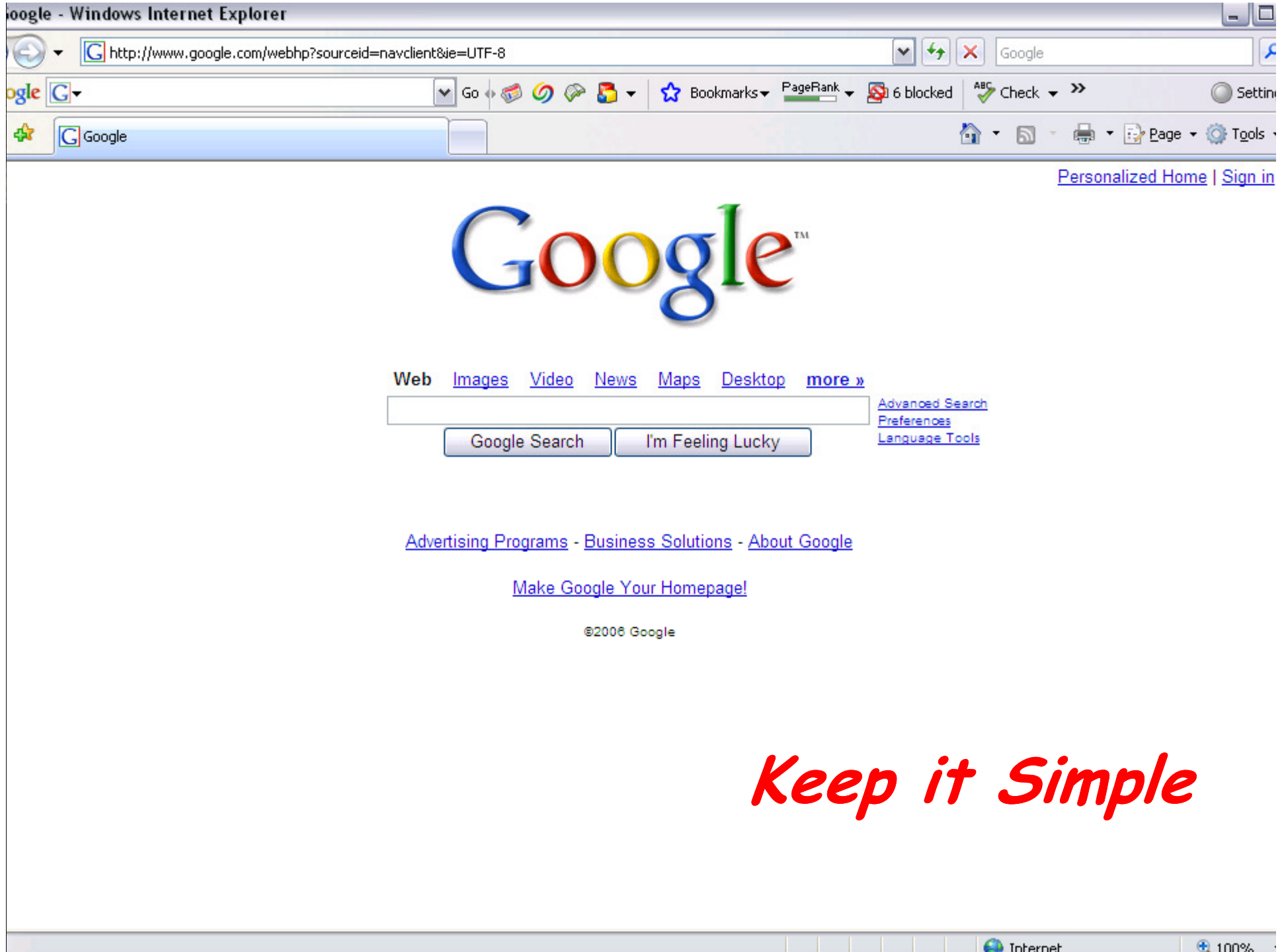
Development

- ✓ Coding Standards
- ✓ Configuration Management
 - ✗ Tool
 - ✗ Team Practices
- ✓ One Click Build
- ✓ Continuous Integration
- ✓ Automated Testing
 - ✗ Unit Tests
 - ✗ Acceptance Tests
 - ✗ **STOP** if the tests don't pass
- ✓ Nested Synchronization



Deployment

- ✓ Production-Hardy Code
- ✓ Automated Release Packages
- ✓ Automated Installation
- ✓ “Done” means the code is running live in production.

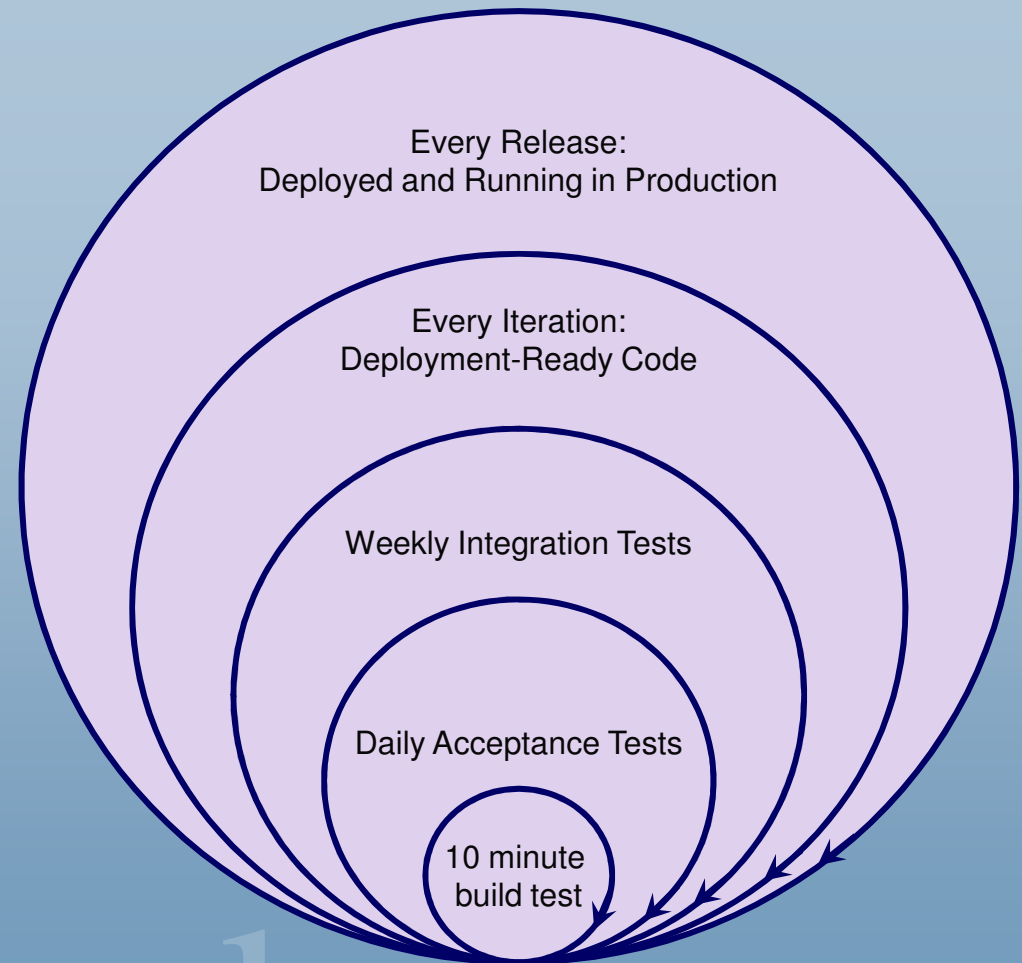


Keep it Simple



Nested Synchronization

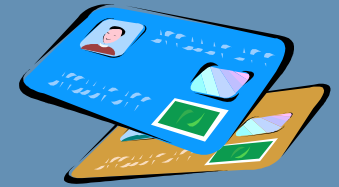
- ✓ Every few minutes
 - ✗ Build & run unit tests
 - ✗ **STOP** if the tests don't pass
- ✓ Every day
 - ✗ Run acceptance tests
 - ✗ **STOP** if the tests don't pass
- ✓ Every week
 - ✗ Run production testes
 - ✗ **STOP** if the tests don't pass
- ✓ Every iteration
 - ✗ Deployment-ready code
- ✓ Every Release
 - ✗ Deploy and run in production



Make it Flawless



Avoid Technical Debt



*Anything that makes code difficult to change
(The usual excuse for batches & queues)*

✓ Complexity

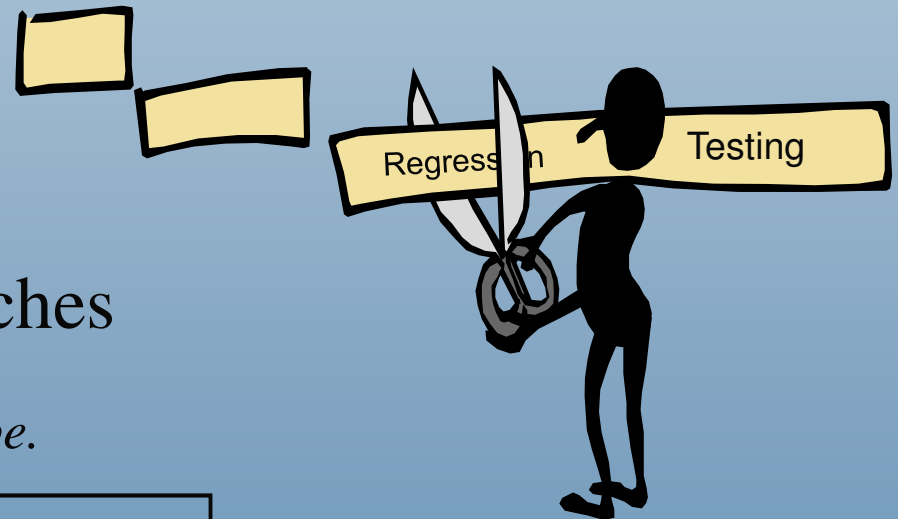
The cost of complexity is exponential.

✓ Regression Deficit

*Every time you add new features
the regression test grows longer!*

✓ Unsynchronized Code Branches

*The longer two code branches remain
apart, the more difficult merging will be.*



Perfection is **One-Piece-Flow:**
Any useful feature set – at any time – in any order

Let it Flow



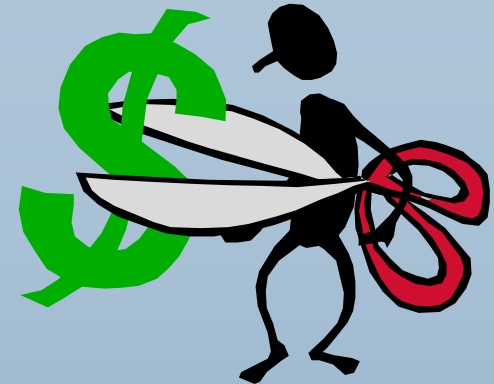
Test-Driven Development

Doesn't cost, it pays!

Example from Mike Cohn

- ✓ Team struggling with legacy java code
- ✓ 10 defects / 1000 NCSS*
- ✓ Affected company's reputation and threatened survival
- ✓ Adopted Test Driven Development
- ✓ Defects dropped to <3 / 1000 NCSS
 - ✗ Including all untouched legacy code
 - ✗ 80-90% improvement in quality
- ✓ Productivity more than tripled

* Non-comment Source Statements



lean



Types of Testing

Business Facing

Manual:
As Early as
Practical

Acceptance Tests

Business Intent
(Design of the Product)

Usability Testing

Exploratory Testing

Unit Tests

Developer Intent
(Design of the Code)

Property Testing

Response,
Security,
Scaling,
Resilience

Critique Product

Tool-Based:
As Early as
Possible

Automated:
Every Day

From Brian Marick

Support Programming

Automated:
Every Build

Technology Facing



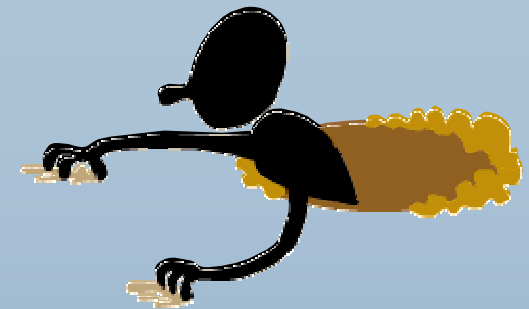
What About Legacy Code?

Stop Digging!

- ✓ Add unit tests whenever you touch the code.
- ✓ Automate acceptance tests for all new features.

Gradually Reduce the Regression Deficit

- ✓ Don't bother to retrofit unit tests
- ✓ Create a regression test harness for key API's
 - ✗ Prioritize by risk and time-to-test
 - ✗ Gradually increase code coverage
- ✓ If it's un-testable, gradually refactor the architecture
 - ✗ Example: Rally



References:

- ✓ Michael Feathers – *Working Effectively with Legacy Code*
- ✓ Mugridge & Cunningham – *Fit for Developing Software*

l e a n



Lean Software Development Agenda

Introduction

- ✓ History

Eliminate Waste

- ✓ Value Stream Maps

Build Quality In

- ✓ Mistake-Proofing

Defer Commitment

- ✓ Set-Based Design

Deliver Fast

- ✓ Queuing Theory

Respect People

- ✓ Teams & Expertise

Focus on Learning

- ✓ Scientific Method

Optimize the Whole

- ✓ Measurements



l e a n



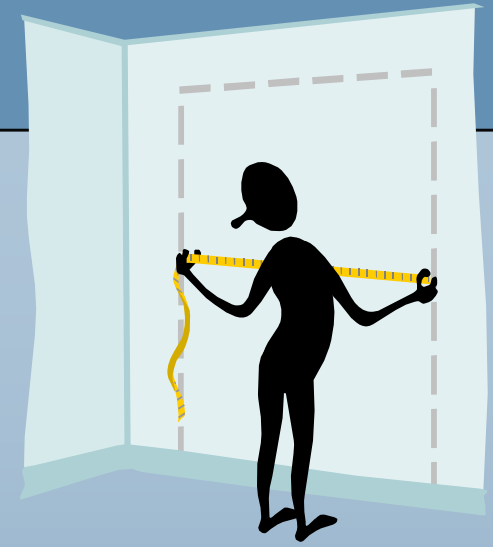
Principle 3: Defer Commitment

A Pilot from Innsbruck

- ✓ In Pilot training, we were taught how to make decisions in challenging situations:
 1. First decide when the decision will be made
 2. Don't make the decision until that time:
 - ❖ That is when you have the most information
 3. Don't make the decision after that time:
 - ❖ Because there are rocks in our clouds.



Maintain Options



The Goal: Change-Tolerant Software

- ✓ 60-80% of all software is developed after first release to production.
- ✓ A development process that anticipates change will result in software that tolerates change.

The Strategy: Maintain Options

- ✓ Make decisions *reversible* whenever possible.
- ✓ Make *irreversible* decisions as late as possible.



Case Study

Automatic Red-eye Reduction for printer line

- ✓ Can sell X,000 more printers with this feature
- ✓ Hard delivery date
- ✓ Which choice?
 - ✗ Simple algorithm, not very good but will hit date
 - ✗ Complex algorithm, very good, but date at risk
- ✓ Do both!
 - ✗ Guarantee delivery of at least the simple algorithm
 - ✗ Ship the better algorithm when it is ready
 - ❖ If it doesn't make the first release, it'll make another one.





Lean Software Development

Introduction

- ✓ History

Eliminate Waste

- ✓ Value Stream Maps
- ✓ Find & Fit the Need

Build Quality In

- ✓ Mistake-Proofing

Defer Commitment

- ✓ Set-Based Design

Deliver Fast

- ✓ Queuing Theory

Respect People

- ✓ Teams & Expertise

Focus on Learning

- ✓ Scientific Method

Optimize the Whole

- ✓ Measurements
- ✓ Contracts



l e a n



Principle 4: Deliver Fast



Manufacturing



Order Processing



Shipping

Companies that compete on the basis of speed:

- ✓ Enjoy a significant cost advantage relative to peers
 - ✗ A 25-30% cost advantage is typical.
- ✓ Have extremely low defect rates
 - ✗ It is impossible to go fast without superb quality.
- ✓ Acquire a deep customer understanding
 - ✗ Fast companies can take an experimental approach to product development.
- ✓ Have a sustainable competitive advantage.



Case Study

Speed to market

- ✓ 45 releases a year – for 4 years
 - Maintenance releases take a week
 - New feature releases take a month
 - New applications take a quarter
- ✓ Every iteration ends in a release
 - Live at all targeted customer sites



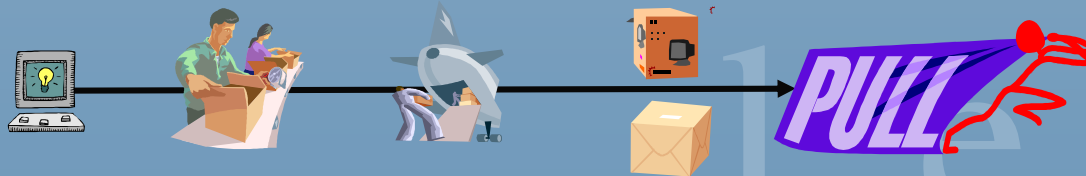
Jeff Sutherland
CTO PatientKeeper

Predictable Delivery

- ✓ Never a late release in 4 years
- ✓ Workload limited to capacity
- ✓ Priorities realigned weekly by CEO, sales, & account management

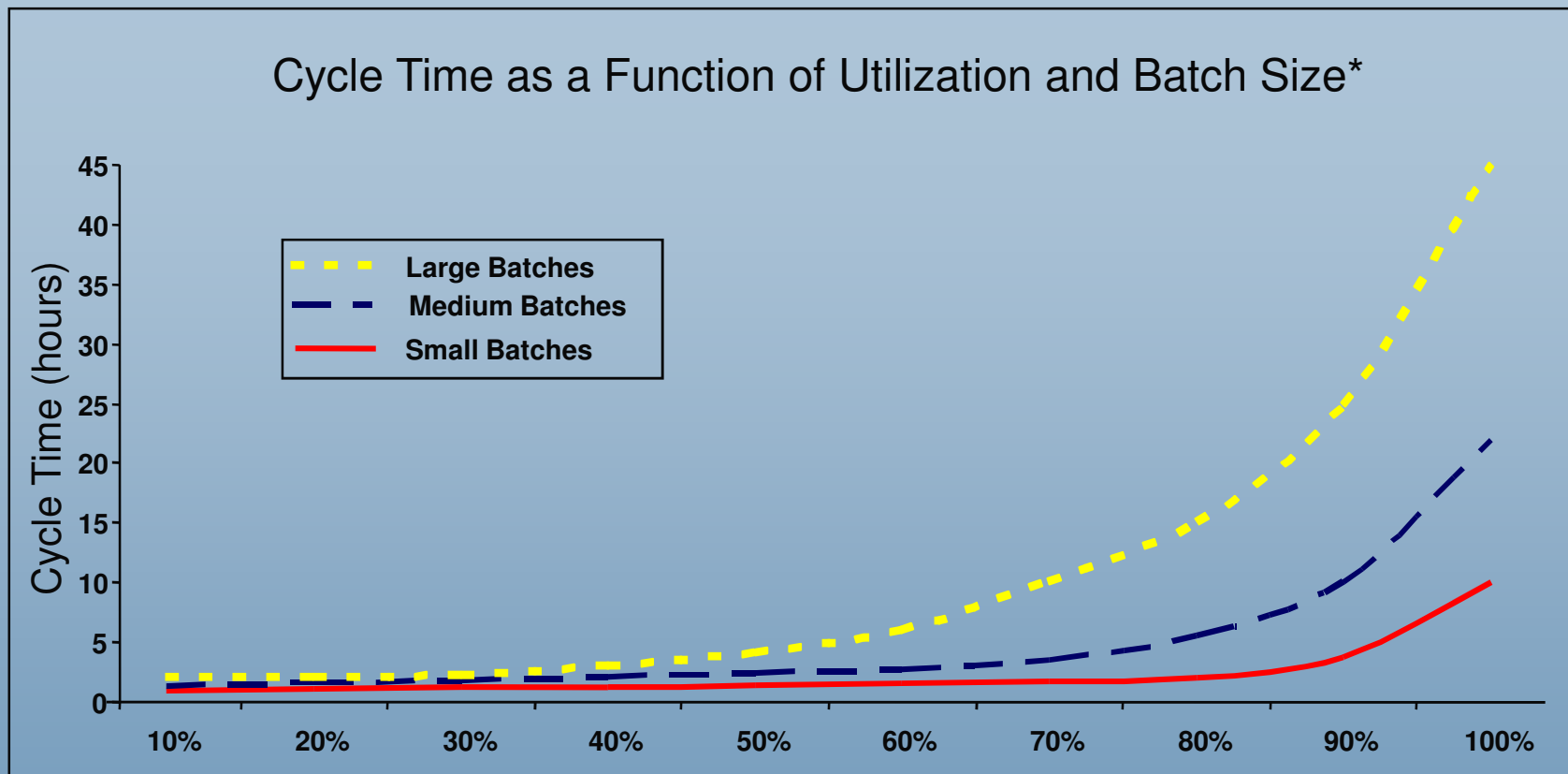
Stop-the-Line

- ✓ Eliminates buggy software because you die if you don't fix this
- ✓ Fixes the install process because you have to install 45 releases a year
- ✓ Improves the upgrade process because of frequent, mandatory upgrades
 - If it's hard, do it more often!


$$\text{Total Cycle Time} = \frac{\text{Number of Things in Process}}{\text{Average Completion Rate}}$$




The Utilization Paradox



Queuing Theory 101

*This assumes batch size is proportional to variability.



Reducing Software Development Cycle Time

- 🕒 Even out the Arrival of Work
- 🕒 Minimize the Number of Things In Process
- 🕒 Minimize the Size of Things In Process
- 🕒 Establish a Regular Cadence
- 🕒 Limit Work to Capacity
- 🕒 Use Pull Scheduling

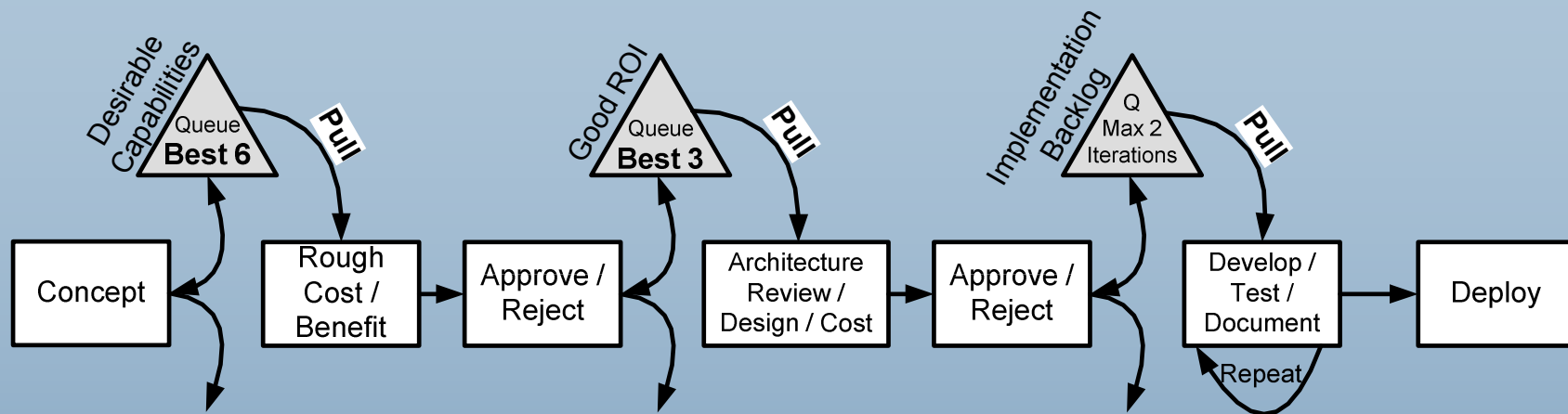


Queuing Theory 101

l e a n



Pull Scheduling Example



Manage the Queues

- ✓ Pull from Queues to limit work to capacity
- ✓ Keep Queues small to decrease cycle time



Lean Software Development

Introduction

- ✓ History

Eliminate Waste

- ✓ Value Stream Maps
- ✓ Find & Fit the Need

Build Quality In

- ✓ Mistake-Proofing

Defer Commitment

- ✓ Set-Based Design

Deliver Fast

- ✓ Queuing Theory

Respect People

- ✓ Teams & Expertise

Focus on Learning

- ✓ Scientific Method

Optimize the Whole

- ✓ Measurements



l e a n



Principle 5: Respect People

*Move responsibility and
decision-making to the
lowest possible level.*

People thrive on

- ✓ **Pride**
- ✓ **Commitment**
- ✓ **Trust**
- ✓ **Applause**





The Key to Toyota's Success

“Only after American carmakers had exhausted every other explanation for Toyota's success – an undervalued yen, a docile workforce, Japanese culture, superior automation – were they finally able to admit that Toyota's real advantage was its ability to harness the intellect of ‘ordinary’ employees.”

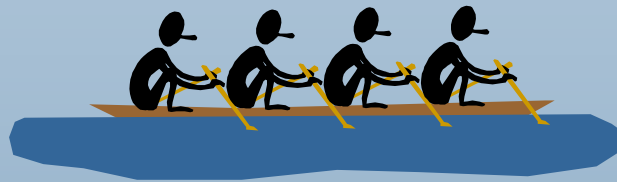
“Management Innovation” by Gary Hamel,
Harvard Business Review, February, 2006



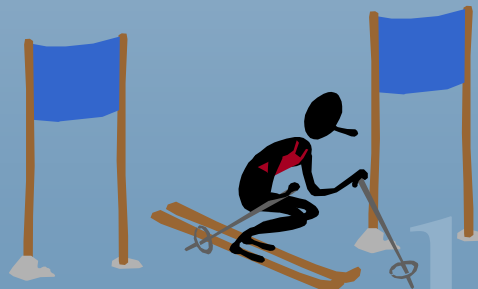


What is a Team?

A team is a group of people who have committed to each other to work together to achieve a common purpose.



A group of people who sum up their individual efforts to meet a goal may be a work group, but they aren't a team.





Reliable Commitment

If a team doesn't meet its commitments:

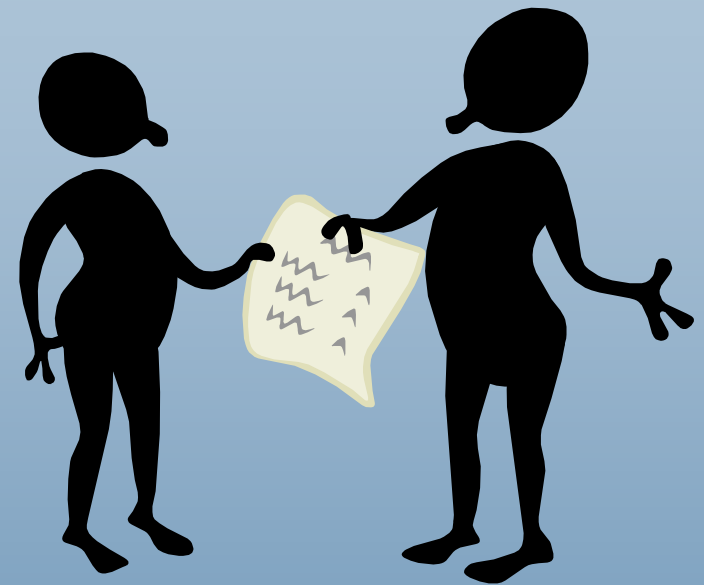
- ✓ Is it a team or a work group?
- ✓ Are tasks self-selected or assigned?
- ✓ Is the team *expected* to deliver?
- ✓ *Who gets Partial Credit?*

An intact team working in a known domain establishes a *Reliable Velocity*

- ✓ Story points per iteration

This establishes the *capacity* of the team

- ✓ And (across stable teams) the capacity of the organization



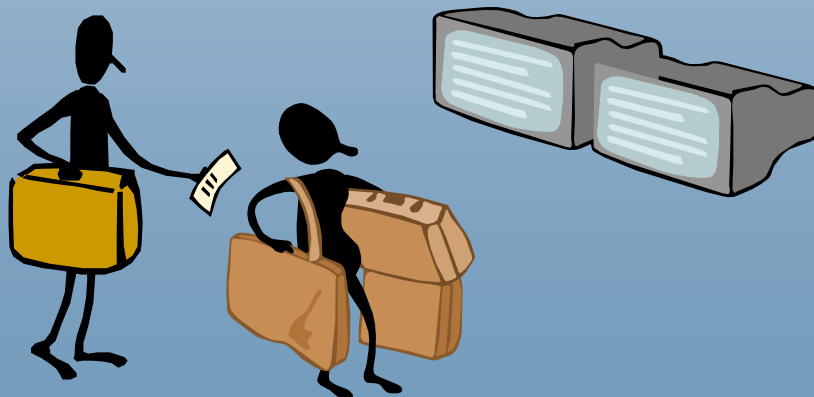


Make Work Self-Directing

Dispatched



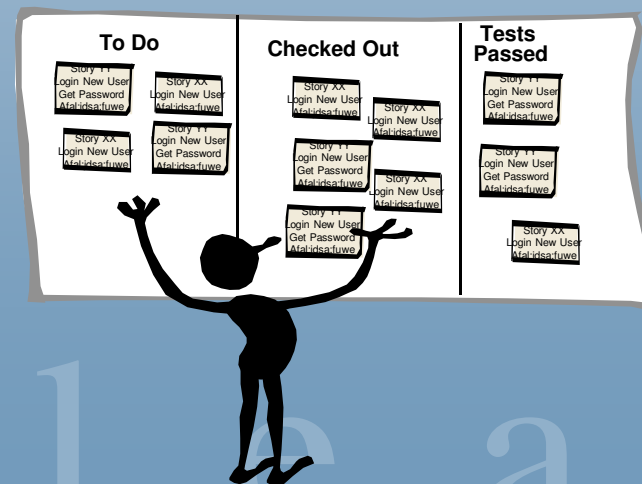
Self-Directing



Dispatched



Self-Directing

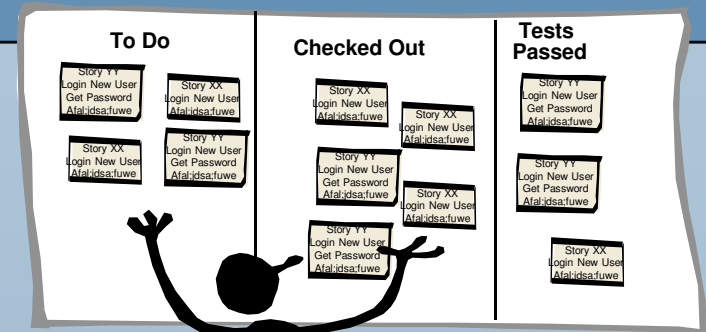




The Visual Workplace

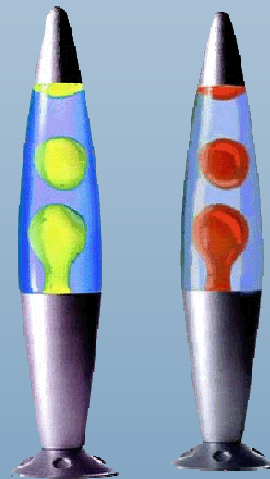
Kanban

✓ What to do next?



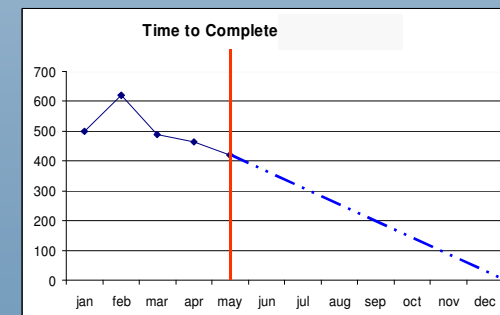
Status Lights

✓ Stop the Line!



Big Visible Charts

✓ How are we doing?



l e a n



Lean Software Development

Introduction

- ✓ History

Eliminate Waste

- ✓ Value Stream Maps
- ✓ Find & Fit the Need

Build Quality In

- ✓ Mistake-Proofing

Defer Commitment

- ✓ Set-Based Design

Deliver Fast

- ✓ Queuing Theory

Respect People

- ✓ Teams & Expertise

Focus on Learning

- ✓ Scientific Method

Optimize the Whole

- ✓ Measurements

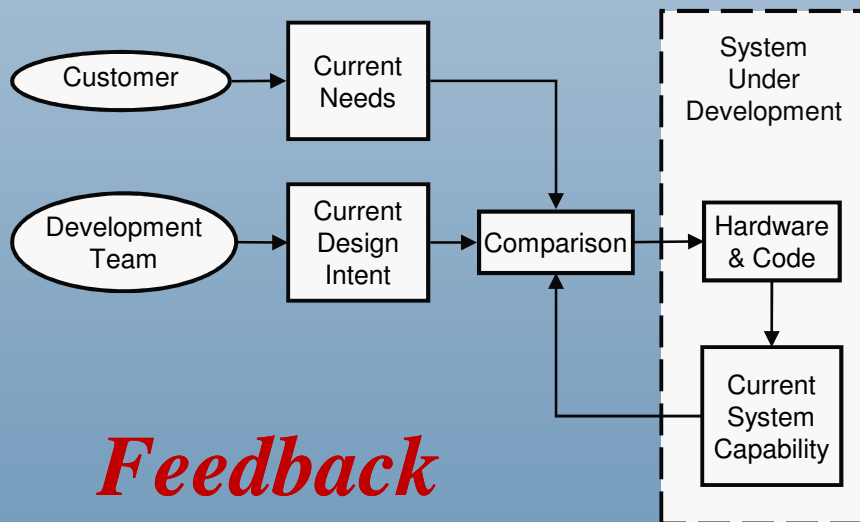




Principle 6: Focus on Learning

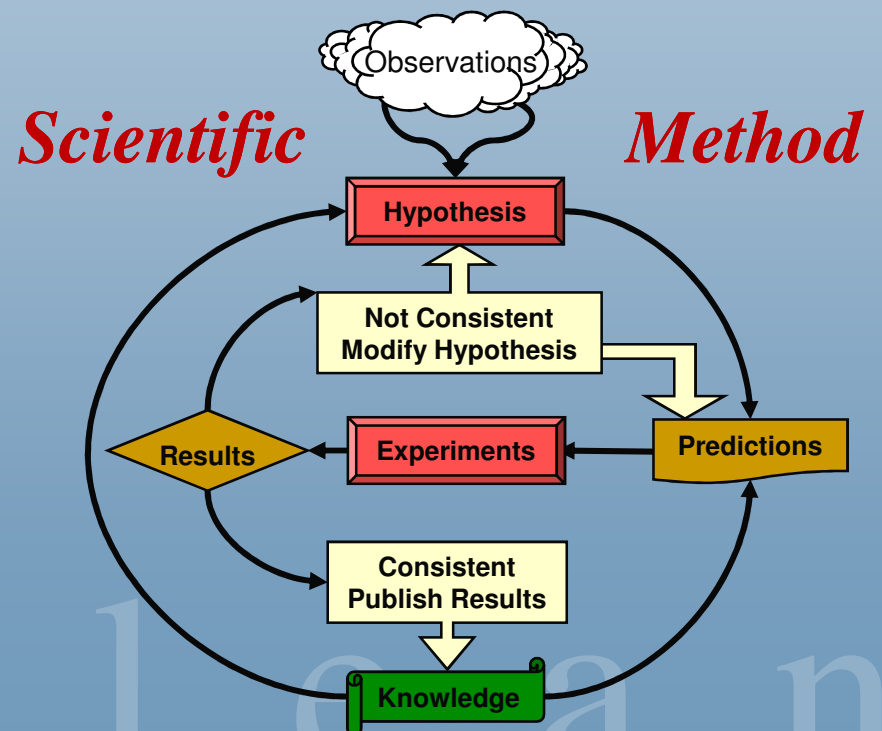
Cycles of Discovery

Products emerge through iterations of learning.



Relentless Improvement

Engaged people design and improve their own processes.





Standards

The current best known way to do a job

- ✗ So everyone follows the standard

The baseline for change

- ✗ If you don't know where you are
you don't know when you've improved.



- ✓ Everybody's job is to take their work to the next level.
- ✓ Workers are expected to challenge and change things that annoy them or keep them from doing a great job.
- ✓ People are proud of their work and their processes.



Capturing Knowledge

The A3 Report

Two sheets of letter paper

A3 reports are concise, useful summaries of knowledge.

They condense findings and capture important results.



Ground Rules

Everything fits on one side of an A3 sheet.

If it doesn't fit, condense it to smaller sheet!

Use as few words as possible.

- ✓ A picture is worth a thousand words.

A3's are dynamic documents.

- ✓ Obtain feedback and update.



Capture in an A3 Document

A Use Case

Coding Standards

Logging Approach

Security Standards

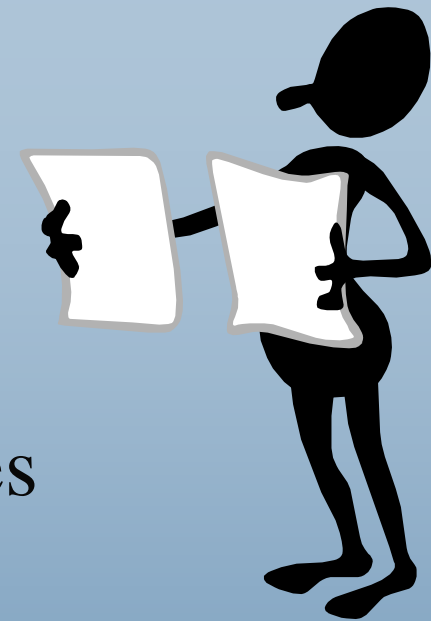
Configuration Management Practices

Problem Under Investigation

High Level Architecture

Business Goals of System Under Development

Iteration Goal / Release Goal





Lean Software Development

Introduction

- ✓ History

Eliminate Waste

- ✓ Value Stream Maps
- ✓ Find & Fit the Need

Build Quality In

- ✓ Mistake-Proofing

Defer Commitment

- ✓ Set-Based Design

Deliver Fast

- ✓ Queuing Theory

Respect People

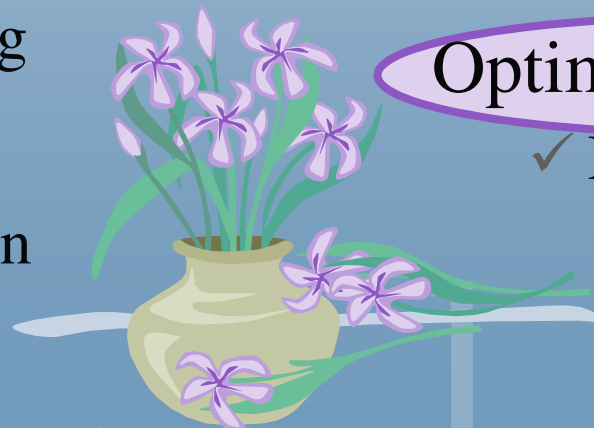
- ✓ Teams & Expertise

Focus on Learning

- ✓ Scientific Method

Optimize the Whole

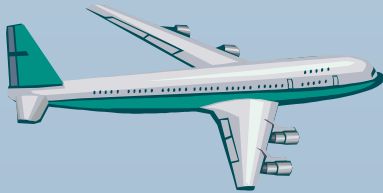
- ✓ Measurements



l e a n



Principle 7: *Optimize the Whole*



Consider the Airplane

An engine won't fly at 10,000 m.

A wing or tail will fall from the sky.

*An airplane is a system,
no part can fly alone.*

*Make the engine larger without
changing anything else, and the
plane will likely fall apart.*

*An airplane must be changed
and managed as a system.*

Zara: Women's fashion clothing

- ✓ Design-to-Store in 2 weeks.
- ✓ Twice-weekly orders.
 - ✗ Delivers globally 2 days after order
 - On hangers, priced, ready to sell
 - Shipping prices are not optimized!
- ✓ Manufactures in small lots
 - ✗ Mostly at co-ops in Western Spain
 - At Western European labor rates...

RESULTS	Zara	Industry
New Items introduced / year	11,000	3,000
Items sold at full price	85%	60-70%
Unsold Items	<10%	17-20%
% sales spent on advertising	0.3%	3-4%
% sales spent on IT	0.5%	2%

This is Systems Thinking.



Systems Thinking is Counterintuitive

Drive cost out of
each department

- ✓ Easy
- ✓ Often interferes with overall cost reduction

Eliminate waste
between departments

- ✓ Difficult
- ✓ May not result in the lowest department costs



In order for organizations to perform brilliantly, there are two prerequisites: First, everyone has to agree on *what they want*, and second everyone has to agree on *cause and effect*.*



*"The Tools of Cooperation and Change," by Clayton Christensen and others, *Harvard Business Review*, Oct 2006



Financial Perspectives

Balance Sheet Thinking

- 💣 What is the break-up value of the company?

“I look at the bottom line. It tells me what to do.” Roger B. Smith

*“This metric guided GM into the most catastrophic loss of market share in business history.”**

- ✓ Delay doesn't matter
- ✓ Just-in-case is wise
- ✓ Work-in-process has value
- ✓ Queues support better decisions

**“Conquering Complexity in your Business,”
by Michael George & Stephen Wilson, p 53*

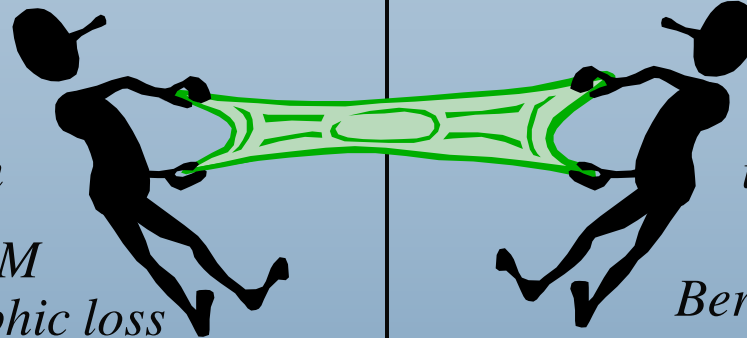
Cash Flow Thinking

- 🕒 How long does it take to convert capital into cash?

“The value of any stock, bond, or business today is determined by the cash inflows and outflows...”

Berkshire Hathaway Annual Report, 1992 (Warren Buffett)

- ✓ Delay creates waste
- ✓ Just-in-time is wiser
- ✓ Work-in-process is waste
- ✓ Queues gum up the works and slow things down

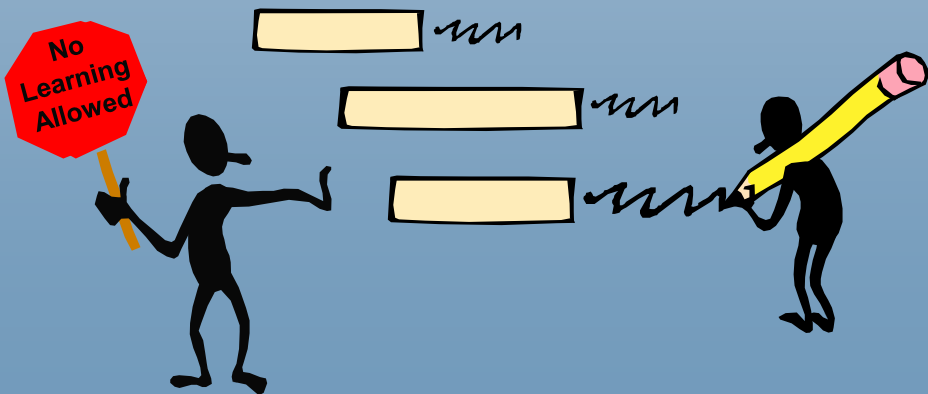




Conformance to Plan

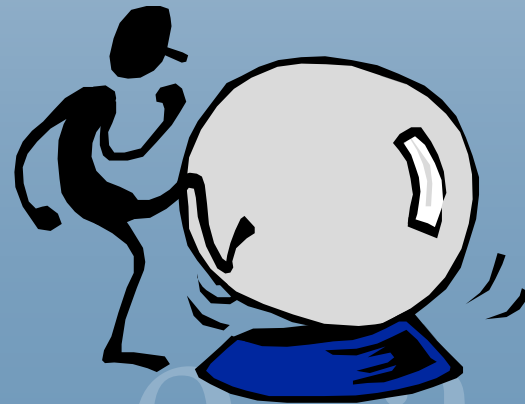
A Plan is a Commitment

- ✓ Predictability comes from conformance to plan.
- ✓ The plan is always right, even though it was made when we had the least information.



Planning is indispensable, but plans are useless. *

- ✓ The most predictable performance comes from maintaining options until we have the most information.





Utilization

We need full utilization of expensive resources.

- ✓ It is impossible to have intact teams because this decreases utilization.
- ✓ Large queues of work help keep everyone busy.



It is impossible to move rapidly without slack.

- ✓ Intact teams increase overall productivity by preserving team learning.
- ✓ Batch and queue mentality is the biggest detriment to system-wide performance.

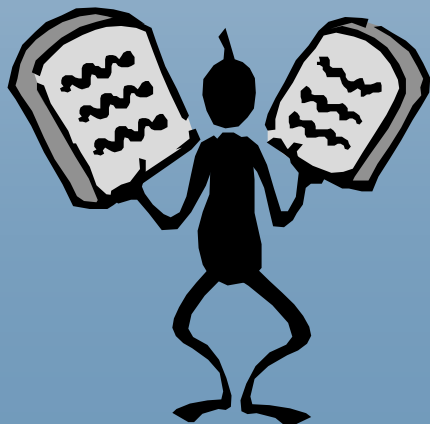




Work Standards

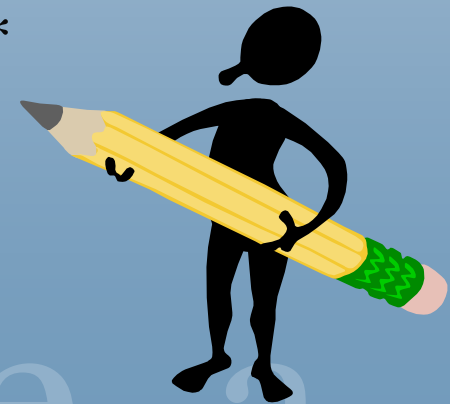
The purpose of standards is to make it possible for any one to do any job.

- ✓ Standards are initiated by process groups.
- ✓ Written standards are to be followed, not changed.



The purpose of standards is to provide a baseline for the team to change.

- ✓ *If you believe that standards are writ in stone, you will fail. You have to believe that standards are there to be changed.**





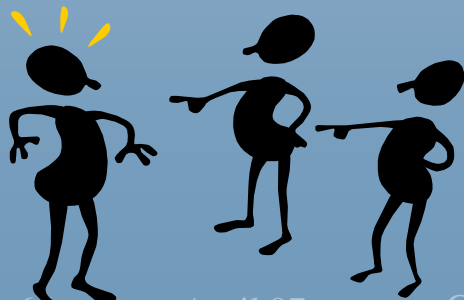
Accountability

Span of Control

Hold people accountable for what they can control
Measure at the individual level
Fosters competition

Example

The development team should be responsible for technical success
The product manager should be responsible for business success



“There is no such thing as “Technical Success”

Kent Beck, XP 2004

Span of Influence

Hold people accountable for what they can influence
Measure at the team level
Fosters collaboration

Example

The team includes technical and business people, and the whole team assumes responsibility for business success





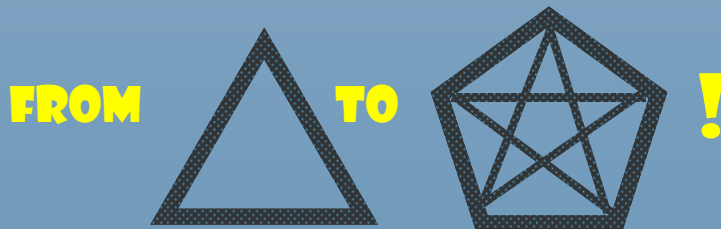
Measure UP

Decomposition

- ✓ You get what you measure
- ✓ You can't measure everything
- ✓ Stuff falls between the cracks
- ✓ You add more measurements
- ✓ You get local sub-optimization

Example

- ✓ Measure Cost, Schedule, & Scope
 - ✗ Quality & Customer Satisfaction fall between the cracks
 - ✗ Measure these too!



Aggregation

- ✓ You get what you measure
- ✓ You can't measure everything
- ✓ Stuff falls between the cracks
- ✓ You measure UP one level
- ✓ You get global optimization

Example

- ✓ Measure Cost, Schedule, & Scope
 - ✗ Quality & Customer Satisfaction fall between the cracks
 - ✗ Measure Business Case Realization instead!





Three System Measurements

Average Cycle Time

- ✓ From Product Concept
- ✓ To First Release
- or
- ✓ From Feature Request
- ✓ To Feature Deployment
- or
- ✓ From Defect
- ✓ To Patch



The Business Case

- ✓ P&L or
- ✓ ROI or
- ✓ Goal of the Investment



Customer Satisfaction

- ✓ A measure of sustainability





1900 – Industrial Training

Charles R. Allen – New Bedford, Massachusetts

- ✓ Four Step method of Industrial Training
 - ✕ Preparation, Presentation, Application, and Testing
- ✓ On-the job training is best
- ✓ Supervisors know how to do the job
- ✓ Supervisors need training in how to train



1917 – War Ships were needed

- ✓ Allen developed programs for shipbuilding
- ✓ 88,000 people were trained in 2 years
- ✓ Wrote the book *“The Instructor, the Man and the Job”*

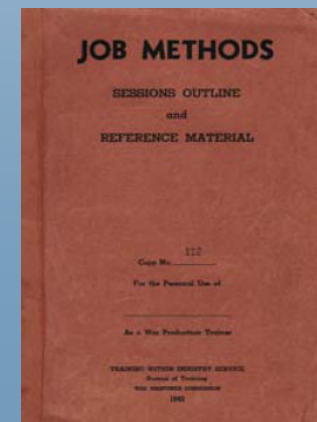
“If the learner hasn’t learned, the teacher hasn’t taught.”



1940 – Training Within Industry

Wartime Production

- ✓ Inexperienced Workforce
- ✓ Used Allen's Approach
 - ✗ Training for first line supervisors
 - ❖ Job Instruction – how to train workers
 - ❖ Job Methods – how to improve the way work is done
 - ❖ Job Relations – how to treat workers with respect
- ✓ Program canceled after the war
- ✓ Taught in Europe and Japan
 - ✗ Well received, especially in Japan
 - ✗ Modified and still used today in Toyota





1980 – Dr. W. Edwards Deming System of Profound Knowledge

Appreciation for the system

- ✓ A systems view was fundamental; never sub-optimize.
- ✓ Manage the relationship between suppliers, producers, and customers

Knowledge of Variation

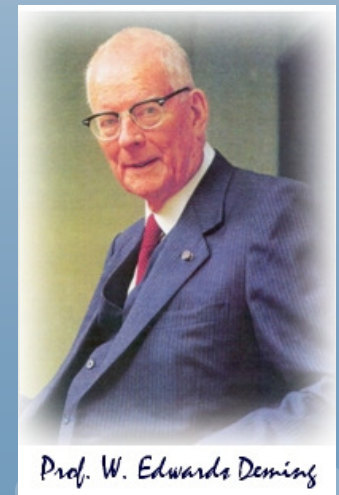
- ✓ Most variation is “common cause variation” – inherent in the system.
 - ✗ Trying to eliminate this variation only makes things worse.
 - ✗ Systemic problems lie beyond the power of the individual worker.
 - ✗ Deadlines and slogans do nothing to address systemic problems.
- ✓ Provide leadership in changing the way the system works.

Theory of Knowledge

- ✓ Use the Scientific Method to improve systems
 - ✗ Hypothesis, Experiment, Learn, Incorporate Learning (PDCA)

Psychology

- ✓ When it comes to people, the things that make a difference are skill, pride, expertise, confidence, and cooperation.



Prof. W. Edwards Deming



Provide Leadership

John Shook
Lean Enterprise Institute
www.lean.org

Three Models of Leadership

“Dictator” Style: “Do it my way...”

“Empowerment” Style: “Do it your way...”

“Lean” Style: “Follow me...and let’s figure this out together”

The leader’s job at Toyota:

First, get each person to take initiative to solve problems and improve his or her job.

Second, ensure that each person’s job is aligned to provide value for the customer and prosperity for the company

Principle:

- ✓ First Line Supervisors

- ✓ Natural Team Leads

Should know how to do the job

Focus training on these leaders

Train How to train workers

- ✓ How to help workers improve their work processes

- ✓ How to respect people



Lean Roadmap

1. Begin where you are

How do you create value and make a profit?

2. Find your biggest constraint

What is the biggest problem limiting your ability to create value and make a profit?

3. Envision your biggest threat

What is the biggest threat to your ability to continue creating value and making a profit over the long-term?

4. Evaluate your culture

Establish and reinforce a culture of deep respect for front-line workers and for partners. Remove barriers that get in the way of pride in workmanship.

5. Train

Train team leads, supervisors and managers how to lead, how to teach, and how to help workers use a disciplined approach to improving work processes.

6. Solve the Biggest Problem

Turn the biggest constraint over to work teams.

Expect many quick experiments that will eventually uncover a path to a solution.

7. Remove accommodations

Uncover the rules that made it possible to live with the constraint. Decide what the new rules should be.

8. Measure UP

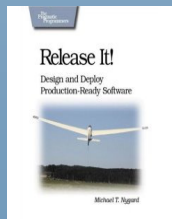
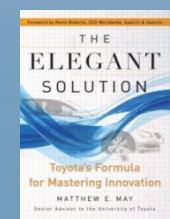
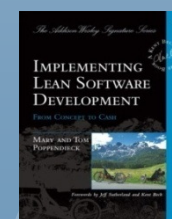
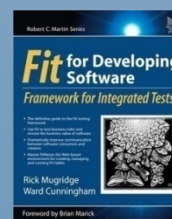
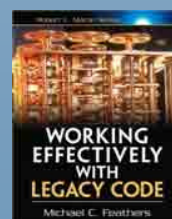
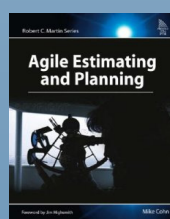
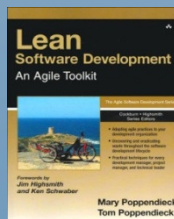
See if end-to-end cycle time, true profitability, and real customer satisfaction have improved.





A Short List of Books

1. Taiichi Ohno – *Toyota Production System*
2. Mary & Tom Poppendieck – *Lean Software Development*
3. Michael Kennedy – *Product Development in the Lean Enterprise*
4. Mike Cohn – *Agile Estimating and Planning*
5. Michael Feathers – *Working Effectively with Legacy Code*
6. Mugridge & Cunningham – *Fit for Developing Software*
7. Poppendieck – *Implementing Lean Software Development*
8. Matthew May - *The Elegant Solution*
9. Michael Nygard – *Release It!*





lean

software development

Thank You!

More Information: www.poppendieck.com