Lean Software Development

Speed – Quality – Low Cost

info@netobjectives.com
www.netobjectives.com

Alan Shalloway

- CEO, Net Objectives, CSM, Author
- Net Objectives offers training/coaching in all aspects of agile development
  - Lean, Scrum
  - Agile Analysis, Design Patterns, TDD, XP
- Training/Coaching Design Patterns, Lean Software Development, Test-Driven Development

The Context For This Seminar

Fast

Flexible

Flow

Lean Software Development

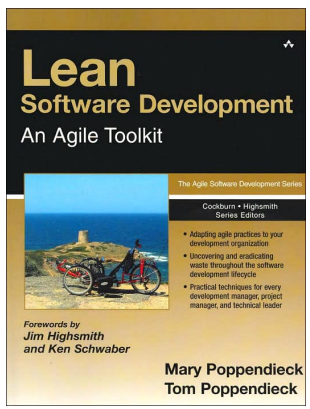
- Lean Software Development has the goal of delivering as much value to your customer as quickly as possible in the most efficient manner possible. The keywords here are speed and low cost. Accomplishing this requires your team to have a commitment to continuous process improvement. This requires management to both respect and empower the people involved. Only the team can create an effective process – management must facilitate this – not control them. Only by continuously improving your process, increasing quality of both your products and your process, can you sustain the high speed and resulting low cost.

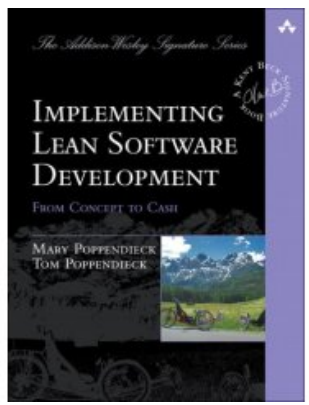
What Is Lean Software Development?

- A combination of:
  - Lean Thinking – Fast, Flexible Flow (Womack, Jones)
  - Toyota's Lean Principles applied in the software world (Poppendiecks)
  - Knowledge Based Software Development (Kennedy)
- Based on:
  - Toyota's Lean Production
  - Toyota's Lean Product Development
  - Deming
  - Theory of Constraints
- Includes Knowledge Management, Mindfulness

Special Thanks

- To Mary and Tom Poppendieck
- Course was originally based on their materials.
- Mary & Tom Poppendieck – *Lean Software Development*
- Mary & Tom Poppendieck – *Implementing Lean Software Development: From Concept to Cash*





Why Learn It?

- Complete foundation for agile.
- What if you don't?

What Is Different About Lean?

- Focused on delivering value quickly.
- A commitment to process improvement.
- A commitment to the people involved.
- Focusing on the whole – not decomposing into steps.
- Paying attention to the temporal nature of events.
- Being customer centric.

Overview

- Value to the customer in a timely manner.
  - Optimize the Whole
  - Deliver Fast
- Must continuously improve process
  - People drive this
  - Management facilitates it

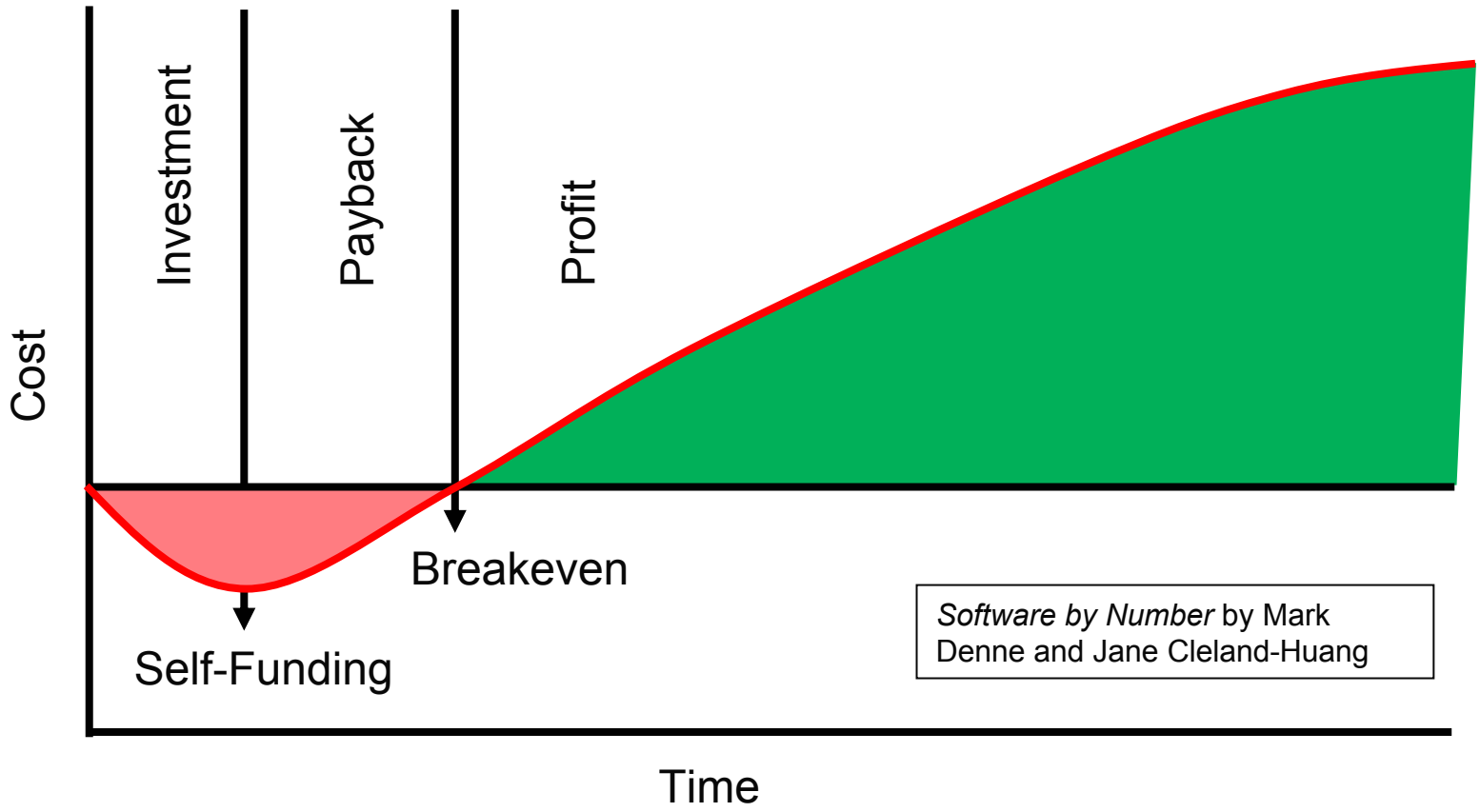
Value to the Customer

- Time is an important element.
- Net Present Value (NPV) of software is not based on interest rates.

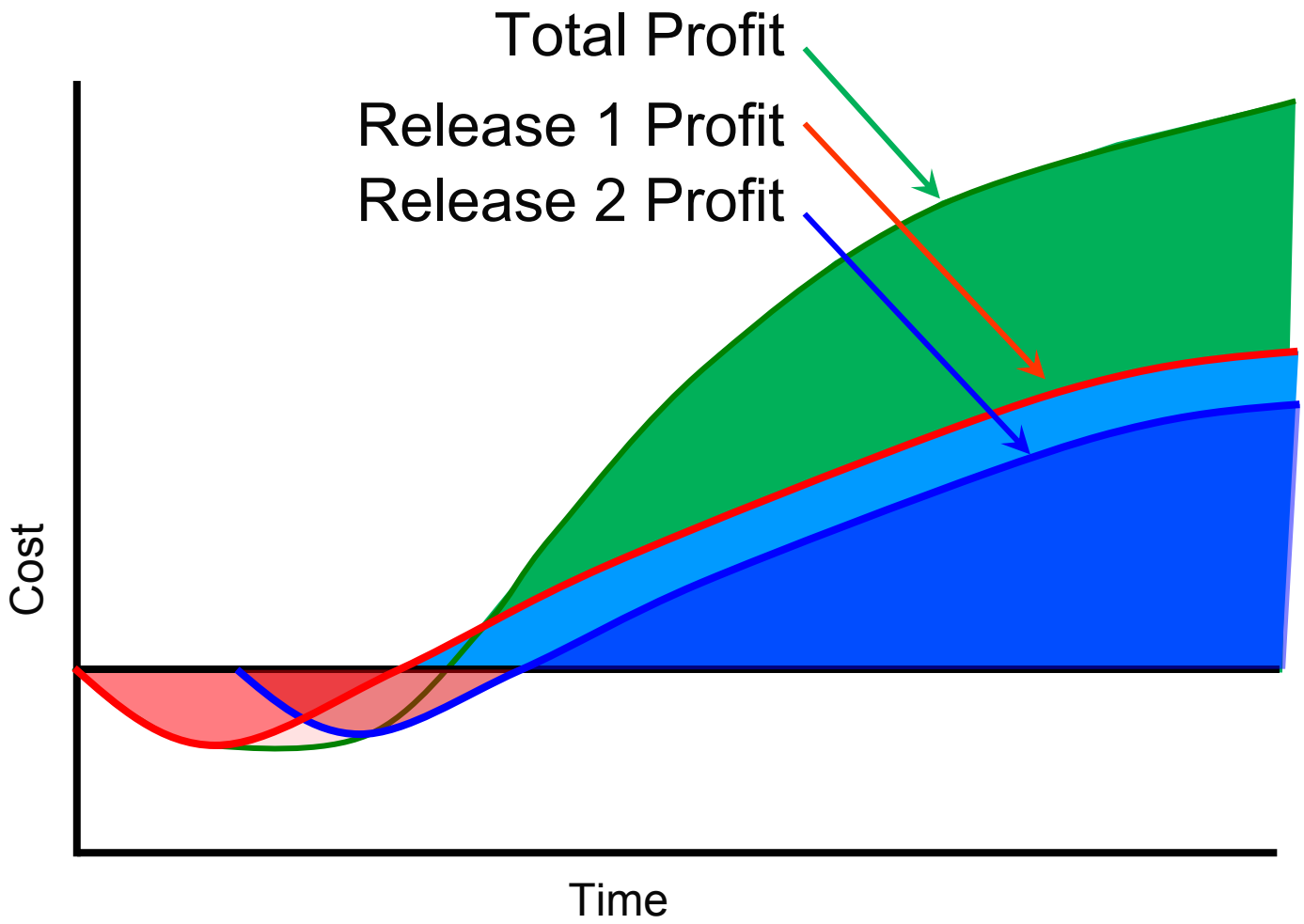
Deliver System in Stages (When Possible)

- Concentrate on the known, valuable, features
- Customers are more certain about the most valuable features
  - Gives value sooner
  - Creates clarity for what's next
  - Development team gains knowledge as they go
- Lowers risk
  - Of building what you don't need
  - Of overbuilding what you do need

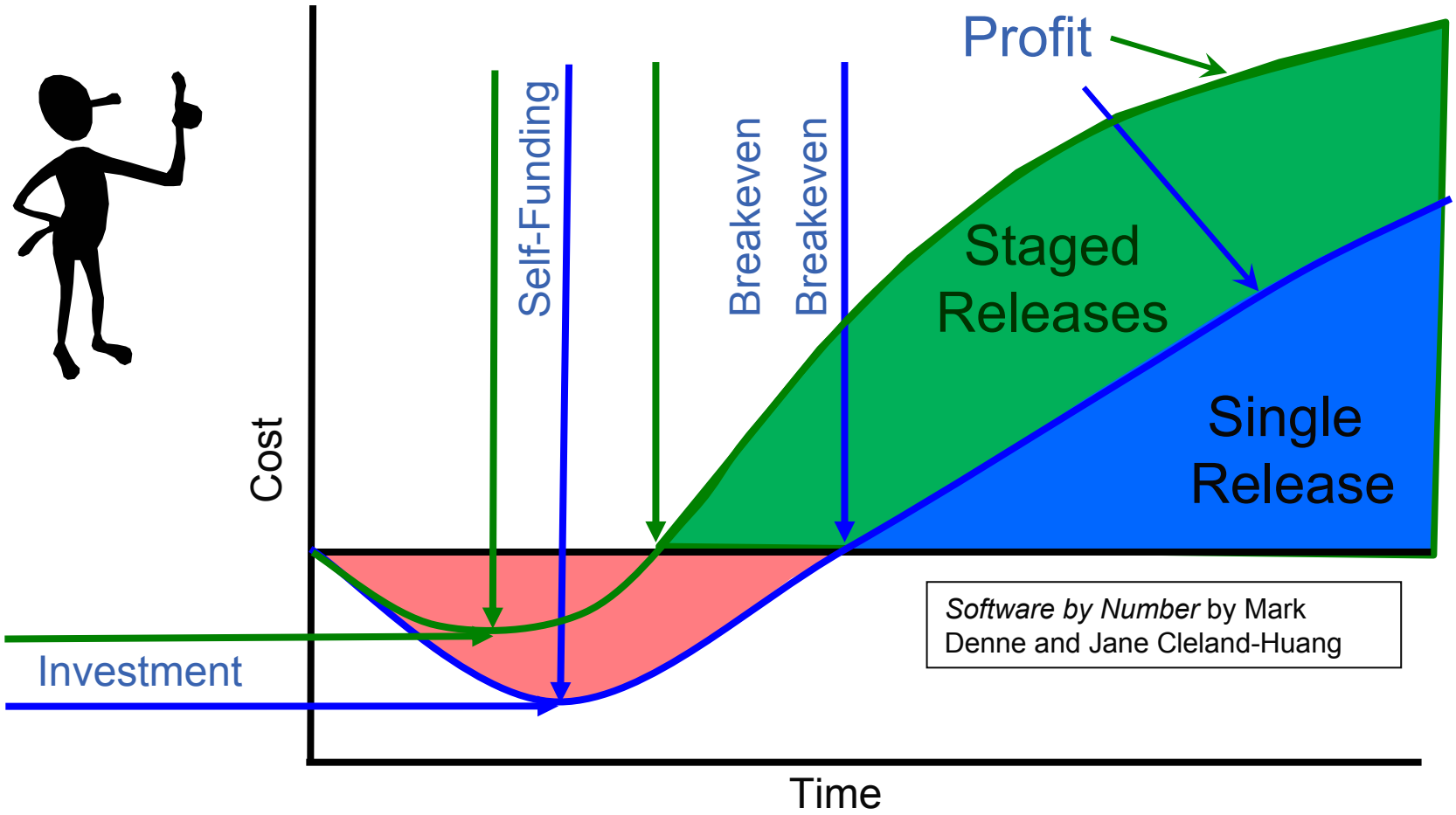
A Financial Model



Staged Releases



Increased Profit



Doing the Most Important Half

Standard development sequence

Suggested development sequence

Most important half of a feature

Less important half of a feature

Doing the Most Important 25%

Standard development sequence

Suggested development sequence

Most important quarter of a feature

Less important quarter of a feature

Paradigm of Lean

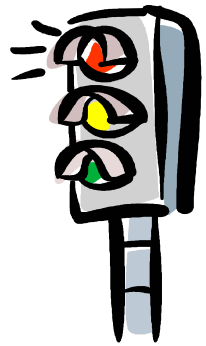
Fast

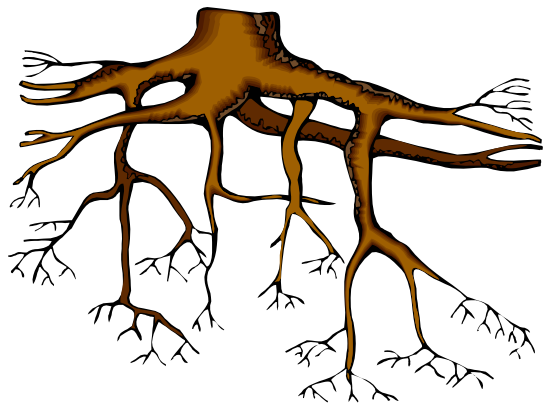
Flexible

Flow

Stop-the-Line Culture

1. Detect every ambiguity
   - Don't work-around even the smallest problem
2. Stop the line immediately
   - Correct the immediate condition if necessary
3. Look for the root cause
   - Why? … Why? … Why? … Why? … Why?
4. Determine the best countermeasure
   - Use many simple experiments
   - Measure results
5. Make the countermeasure permanent
   - Constant improvement = constant change
6. Do not allow for work arounds
   - Work arounds leave problems intact



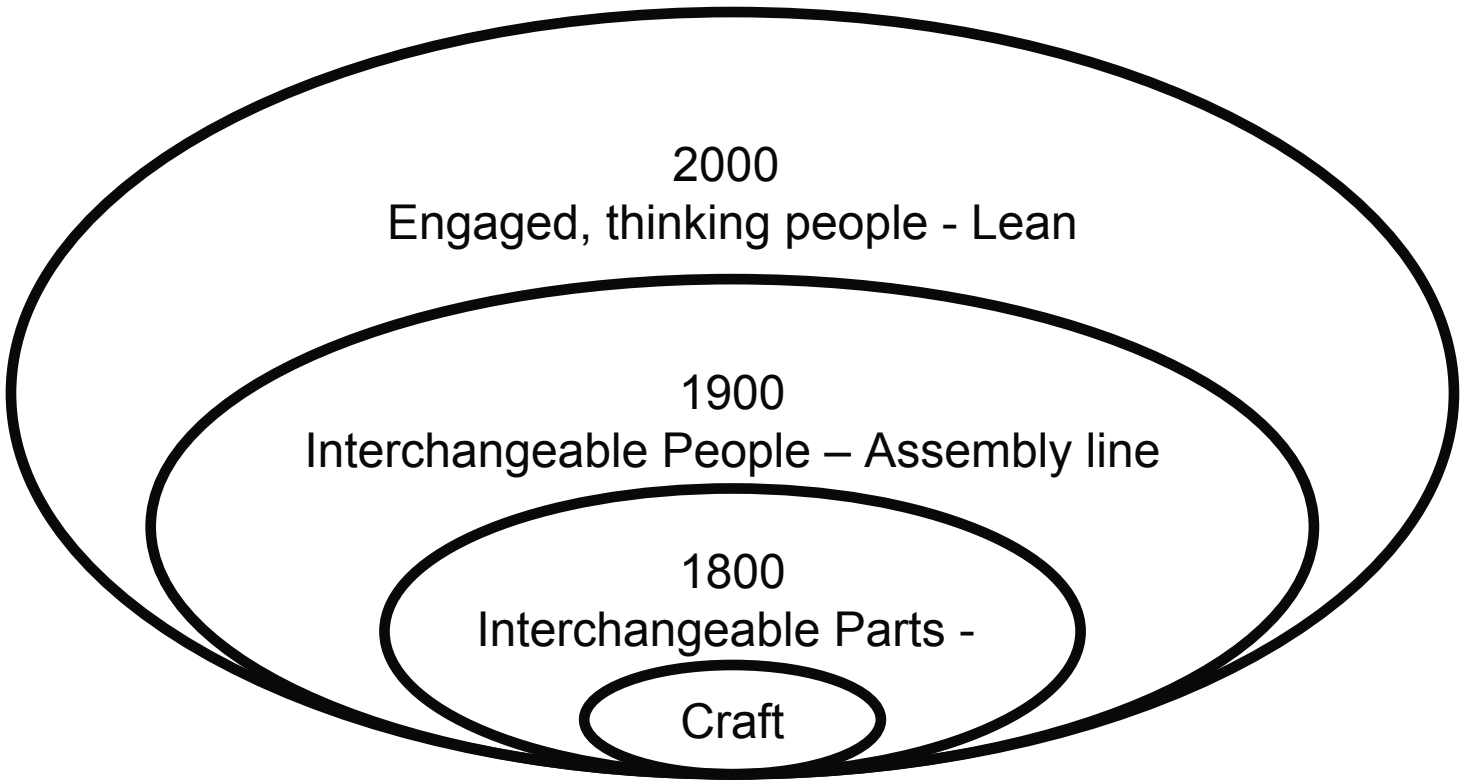


Develop Thinking People

- Software development requires people to solve problems.
- The team must understand the problem.
- They must be empowered to find the best solution.
- If you can't trust your people, you are in trouble.
- ***Trust local knowledge!***
- ***People build the process!***

Lean – A New Paradigm

- 4 industrial paradigms

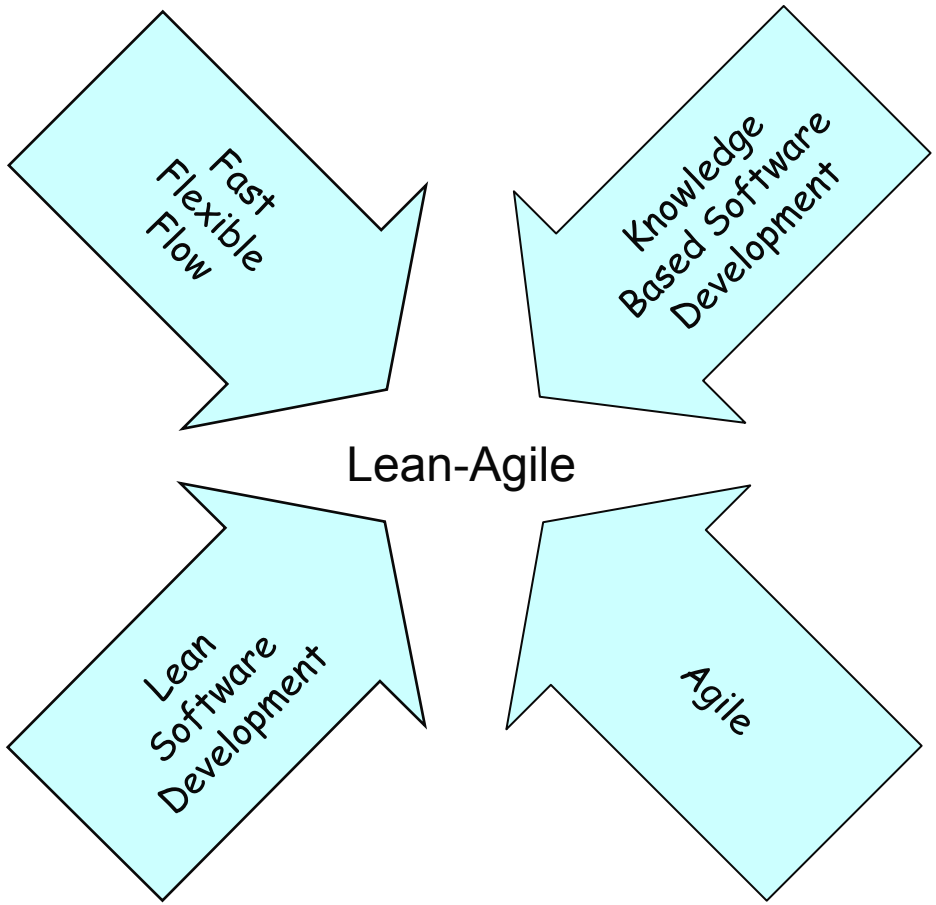


Lean Thinking – Lean Software Development

Speed – Quality – Low Cost

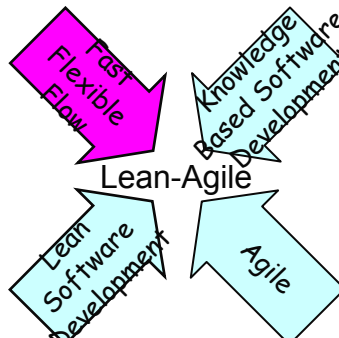
info@netobjectives.com
www.netobjectives.com

An Integration



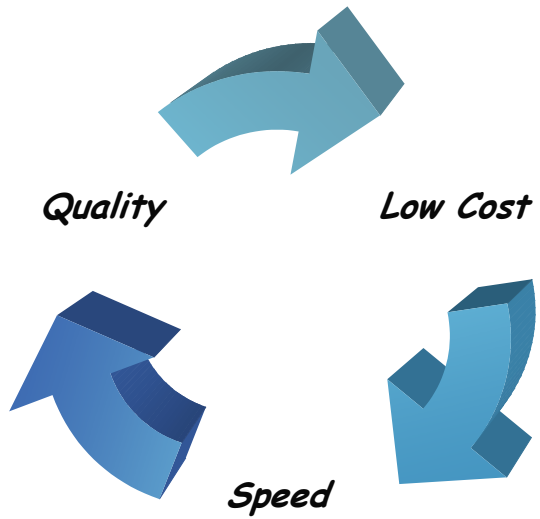
Lean Thinking: Banish Waste and Create Wealth in Your Corporation

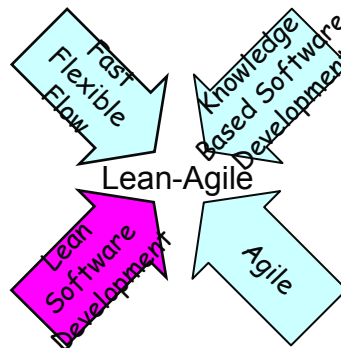
- By Womack and Jones. 1996, 2003.
- Another way to look at lean – includes both products and services.
- They define 5 principles:
  - Value
  - The Value Stream
  - Flow
  - Pull
  - Perfection



Principles of Lean Software Development

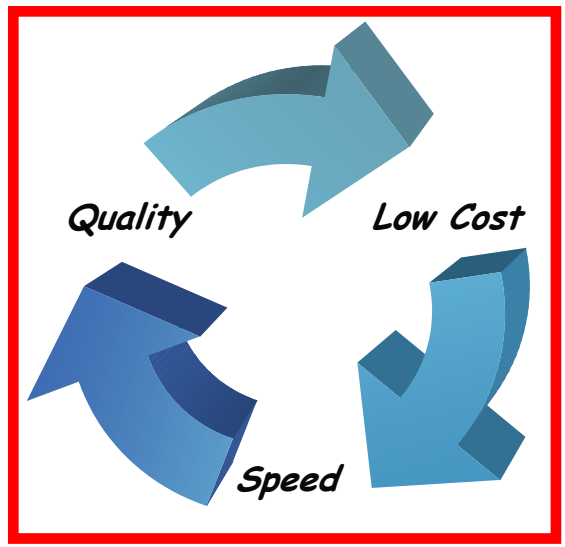
- Optimize the Whole
- Eliminate Waste
- Build Quality In
- Defer Commitment
- Respect People
- Create Knowledge
- Deliver Fast





A Reassessment

- All three are essential
- Starting with low cost:
  - Has limited value
  - Causes poor decisions
- Starting with speed gives insights
- Requires quality for sustainability
- Speed and quality results in lower cost



Value

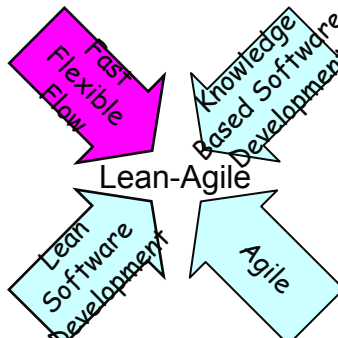
Fast

Flexible

Flow

Lean Thinking - VALUE

- Value is what the customer wants.
- What they are willing to pay for.
- What you are trying to produce.

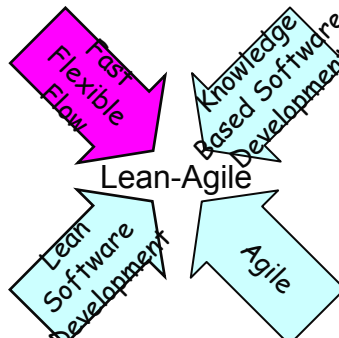


In Agile/Scrum

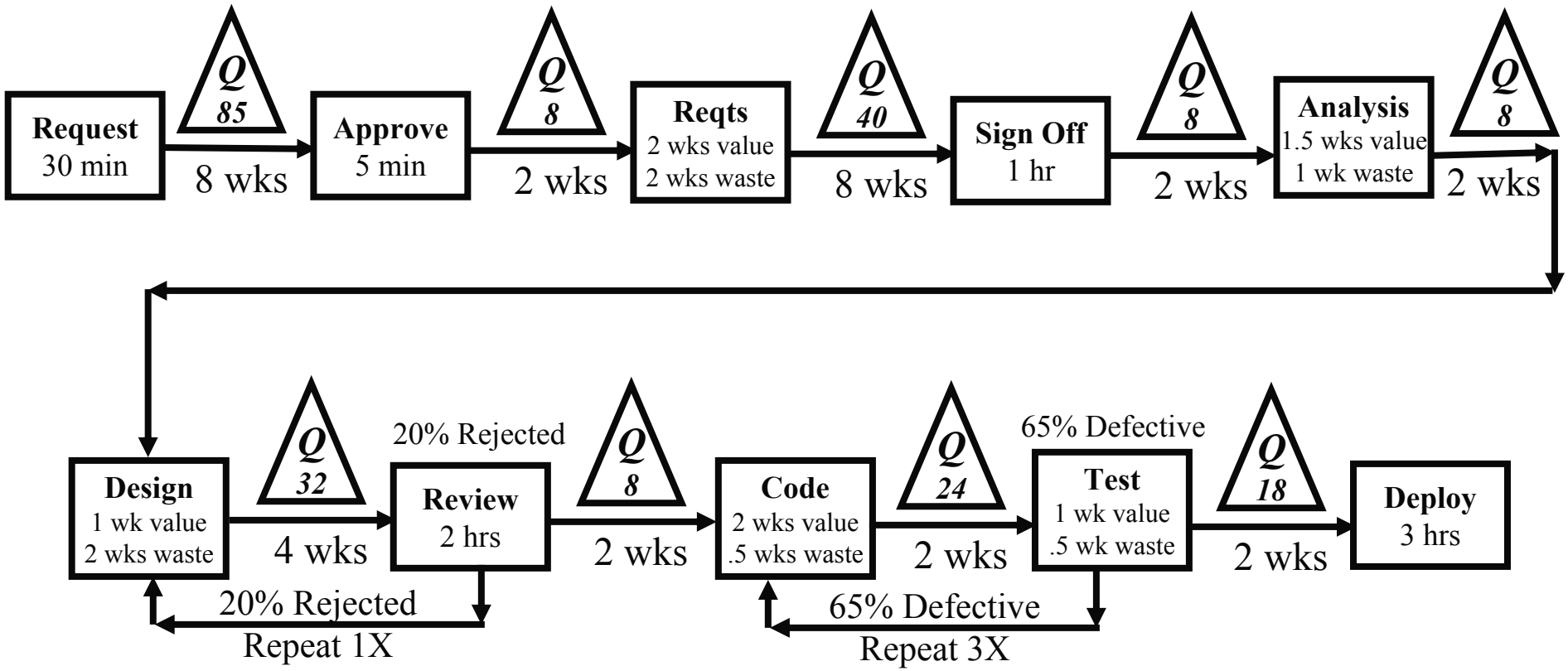
- Focus on customer
- Requires a product owner

Lean Thinking – THE VALUE STREAM

- The flow from beginning to end of creating the value.
- Often cuts across companies, virtually always cuts across organizations.



Sequential Value Stream



$$\text{Process Cycle Efficiency} = \frac{\text{Value Added Time}}{\text{Total Cycle Time}} = \frac{345 \text{ hrs}}{2000 \text{ hrs}} = 17\%$$

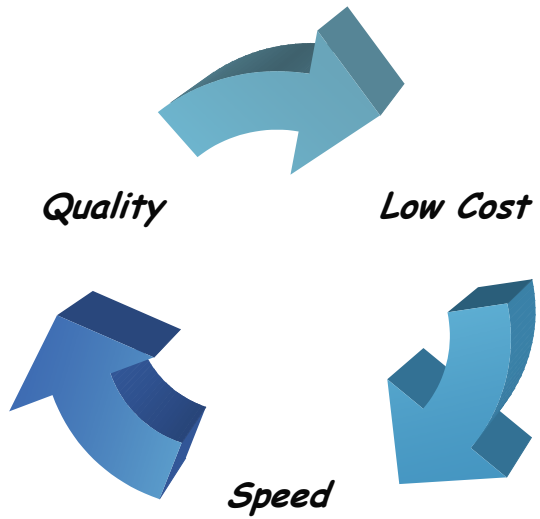
Temporal Focus

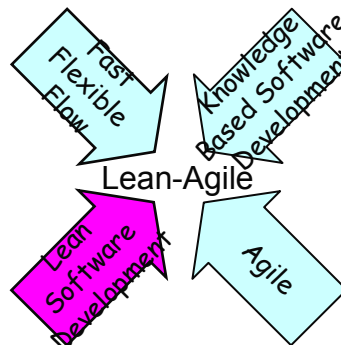
- Timing is important
- Focusing on delays uncovers problems
- Ultimately, we want to optimize the whole
- Focusing on:
  - Delays
  - Task switching (which cause delays)

  … gives insights into speeding up the process and eliminating waste.

Principles of Lean Software Development

- Optimize the Whole
- Eliminate Waste
- Build Quality In
- Defer Commitment
- Respect People
- Create Knowledge
- Deliver Fast





Principle: Optimize the Whole

Vicious Cycle #1:

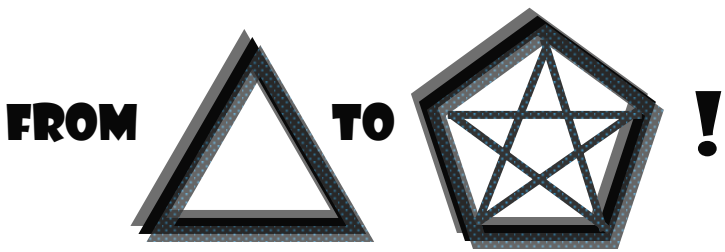
1. A customer wants some new features yesterday
2. Developers hear: Get it done fast, at all costs!
3. Result: Sloppy changes are made to the code base
4. Result: Complexity of code base increases
5. Result: Number of defects in code base increases
6. Result: Exponential increase in time to add features

Vicious Cycle #2:

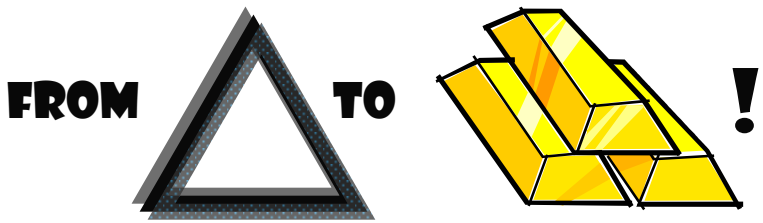
1. Testing is overloaded with work
2. Result: Testing occurs long after coding
3. Result: Developers don't get immediate feedback
4. Result: Developers create more defects
5. Result: Testing has more work. Systems have more defects.
6. Result: Feedback to developers if delayed further. Repeat cycle.

Myth: Optimize through Decomposition

- 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 Value with P&L or ROI instead.



Measure ***UP***

Span of Control

Hold people accountable for what they can ***control***

Measure at the individual level

Fosters competition

Span of Influence

Hold people accountable for what they can ***influence***

Measure at the team level

Fosters collaboration

Testing Manager

"We can't let testers write tests before developers write code. If we did that, the developers would simply write code to pass the tests!"

Product Manager

"Everyone in the company knows that their job depends upon delighting customers."

FLOW

Fast

Flexible

Flow

Lean Thinking - FLOW

- Improving flow increases value as well as decreasing waste.
- All impediments to flow are waste.

Just-in-Time Flow

Value should be added in a smooth flow

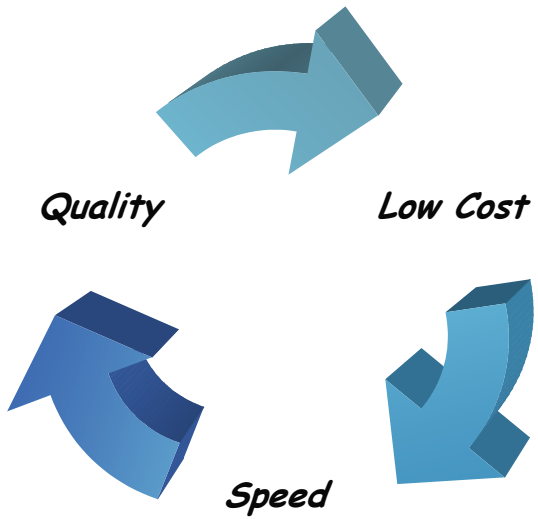
- There should be no impediments
- Steps should not be required to wait
- There should never be large inventories or queues

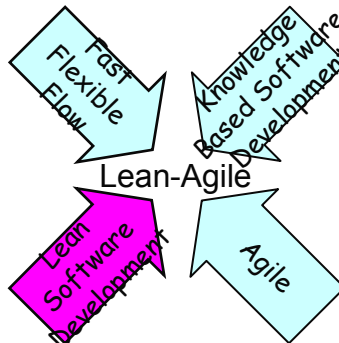
Large queues and inventories hide problems to flow

- Poor flow does not allow for quick delivery
- Poor flow does not allow for flexibility
- Hides quality problems

Principles of Lean Software Development

- Optimize the Whole
- Eliminate Waste
- Build Quality In
- Defer Commitment
- Respect People
- Create Knowledge
- Deliver Fast



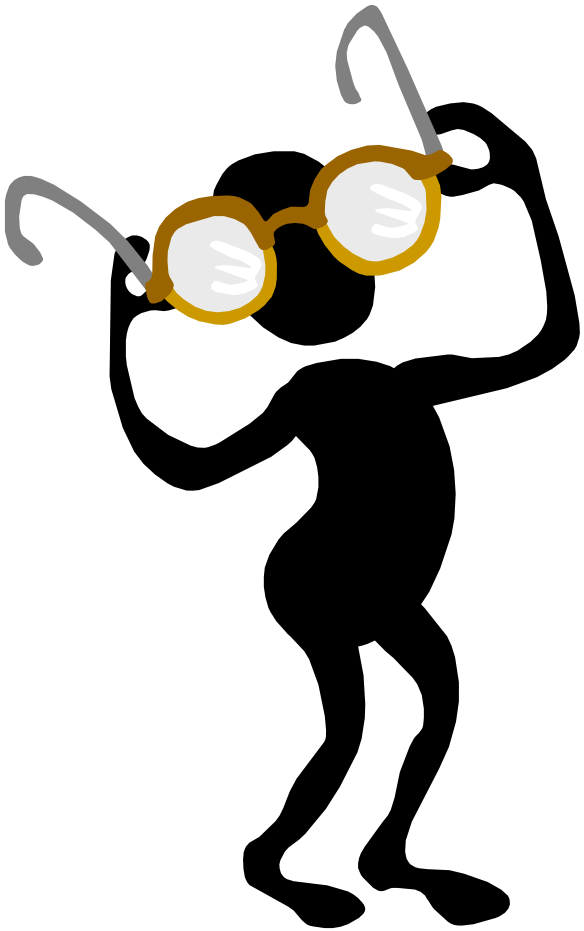


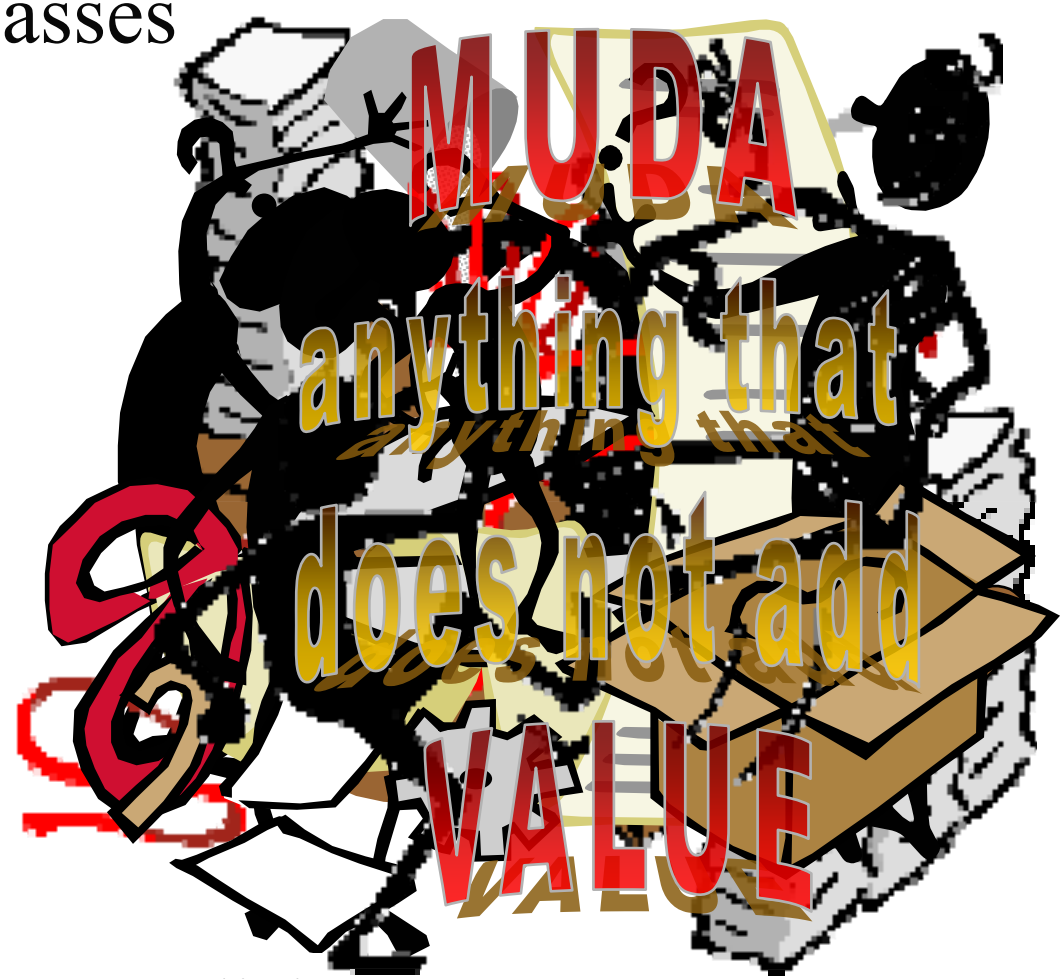
Principle: Eliminate Waste

- ***Waste is anything that does not add customer value***
  - Customers wouldn't choose to pay for it.
  - Doesn't add to the knowledge needed to develop a product.
- Waste is anything that has been started
  - but is not being used in production.
- Waste is anything that delays development
  - or keeps people waiting.
- Waste is any extra features
  - that are not needed now.
- Waste is making the wrong thing
  - or making the thing wrong.

Eliminate Waste

Put on Customer Glasses





The Seven Wastes

The Seven Wastes of Manufacturing
- Shigeo Shingo

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

The Seven Wastes

The Seven Wastes of Manufacturing - Shigeo Shingo

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

The 7 Wastes of Software Development

1. Partially Done Work
2. Paperwork
3. Extra Features

"The task then is to refine the code base to better meet customer need. If that is not clear, the programmers should not write a line of code. 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

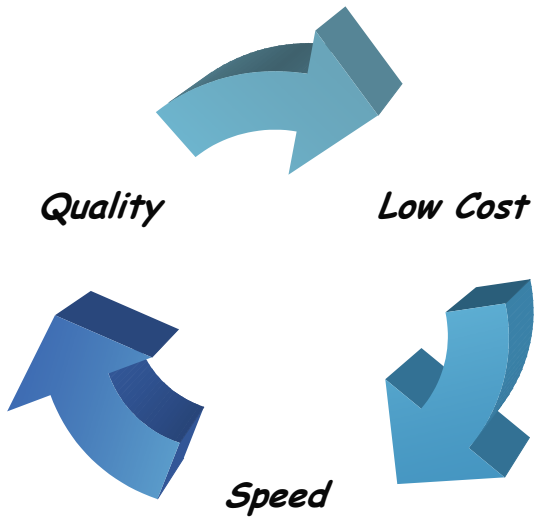
1. Inventory
2. Processing
3. Overproduction
4. Motion
5. Transportation
6. Waiting
7. Defects

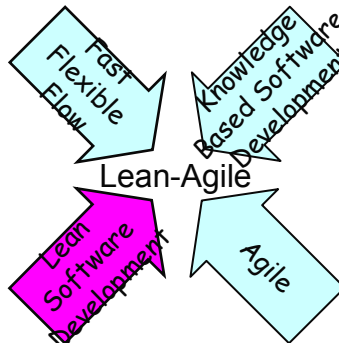
The 7 Wastes of Software Development

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

Principles of Lean Software Development

- Optimize the Whole
- Eliminate Waste
- Build Quality In
- Defer Commitment
- Respect People
- Create Knowledge
- Deliver Fast





Ask Any Development Team…

Q: Where do you spend most of your time?

A: Fixing bugs

Why?

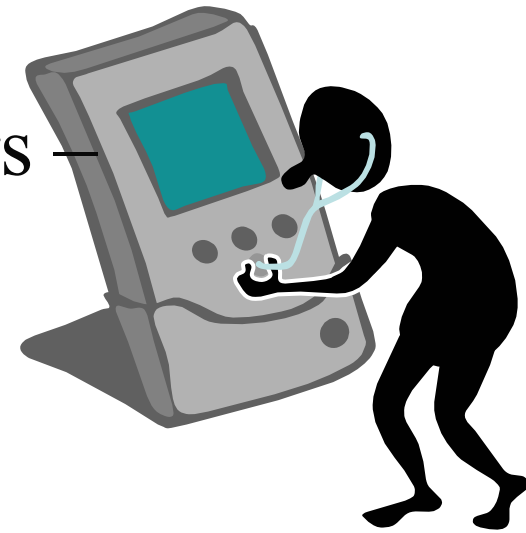
- Because the quality of their code is poor.
- Because 60-80% of the work done on a software product is in *maintenance*, yet we don't focus on this during its development
- Because Waterfall methods is not long-term thinking – it is short-term thinking: *Get it out as soon as possible without regard to the customer's opinion or the code's quality*

But Developers Are Wrong… (at least about this)

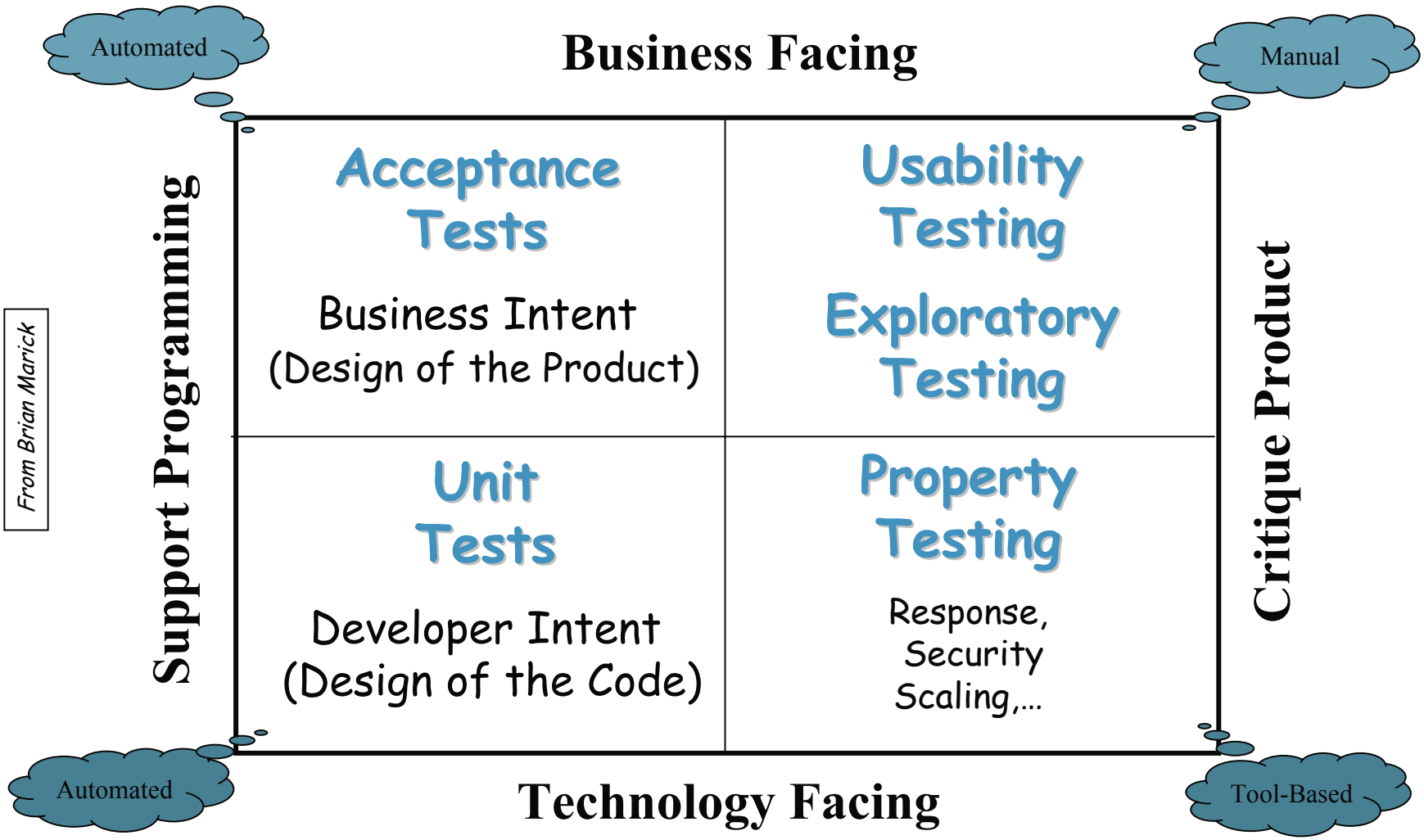
- They don't actually spend much time fixing bugs
- They spend their time in *finding* bugs
  - Because their code is hard to understand.
  - Their code is inter-connected and hard to modify
  - A lot of this happens because of the Emperor's Clothes mentality that they are doing it right the first time and won't have to change it.

Move Inspection Forward

- The job of testing is to prevent defects –
  - If you are focused on finding defects
    - – you are not doing your job.
- A quality process builds quality into the code –
  - If you routinely find defects during verification
    - – your process is defective.
- Defects are not caused by developers –
  - Defects are caused by a system which allows defects.
    - – Defects are a management problem.



Types of Testing



Testing Is Validation

We have to validate

- That we understand what is needed
- That we did what we wanted
- That the product is of sufficient quality

We must push testing up early.

- Tests improve the conversation between customers and developers.
- Tests become executable specifications.

Why Test?

- "The job of tests, and the people that develop and run tests, is to *prevent* defects, not to find them. A quality assurance organization should champion processes which build quality into code from the start, rather than test quality in later. This is not to say that verification is unnecessary. Final verification is a good idea; it's just that finding defects should be the exception, not the rule, during verification. If verification routinely triggers test-and-fix cycles, then the development process is defective.

- Implementing Lean Software Development: From Concept to Cash, Mary and Tom Poppendieck.

PULL

Fast

Flexible

Flow

Lean Thinking - PULL

- *"Pull* in simplest terms means that no one upstream should produce a good or service until the customer downstream asks for it, but actually following this rule in practice is a bit more complicated."

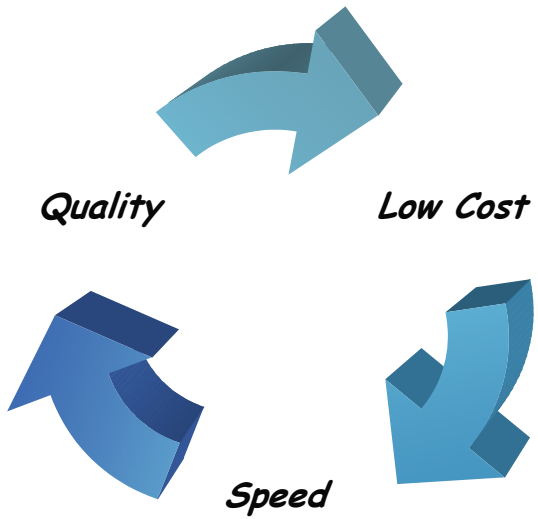
  - Lean Thinking, Womack and Jones.

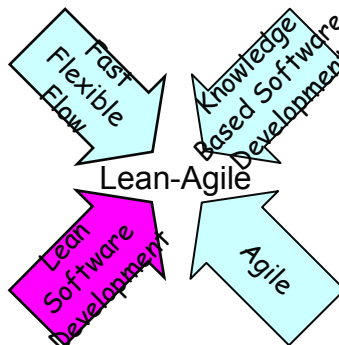
Empirical or Deterministic?

- Software Development is inherently a non-deterministic process.
- Doesn't mean can't be controlled.
- Means can't be managed completely through prediction.
- Must react and adapt.
- Basis for Scrum and other Agile methods.

Principles of Lean Software Development

- Optimize the Whole
- Eliminate Waste
- Build Quality In
- Defer Commitment
- Respect People
- Create Knowledge
- Deliver Fast





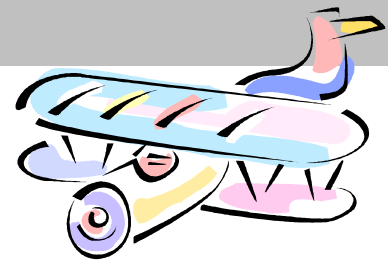
Principle 4: Defer Commitment

- Typically, decisions are made ***too early***.
- We don't want to make decisions that arbitrarily constrain our solution
  - Happens if don't have all the information
  - Defer decisions that we don't need to make now
- Don't wait too long
  - That wouldn't be responsible ***or safe!***

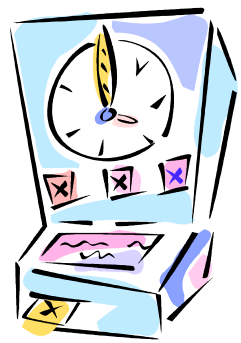
Early Decisions Cause Waste

- Premature decisions won't have all the information available later.
- Getting information before it's needed means:
  - You'll need to get it again later
  - You won't use the most up to date information later

Deciding When To Decide



- Pilots
  - "In Pilot training, we learned that when we had to make a decision, we should first decide when the decision should be made, then when the time comes, make the decision based on the available information."
- Military
  - "One of the most important thing I taught young recruits is that when they were threatened, they should decide on the timebox for a response, and not respond until the end of the timebox."



Lean Thinking - PERFECTION

- Provides the vision for continuously improving your value stream to increase flow, decrease waste and add value to the customer.

Improve Process

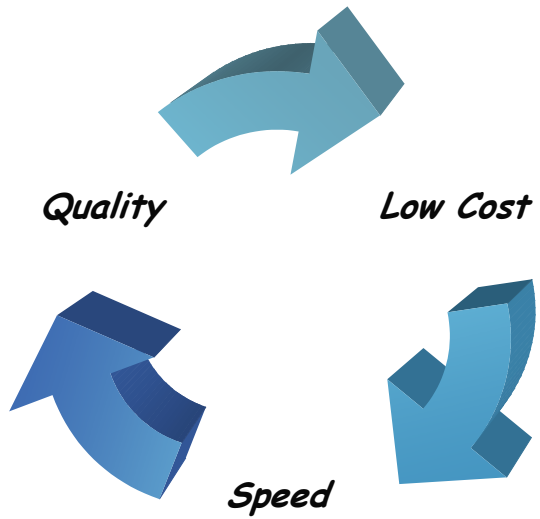
- Discover what customer needs.
- Go to root cause
- Always have a process
  - But improve it
  - Process become baseline for change
  - Must incorporate knowledge management techniques to both discover and retain knowledge

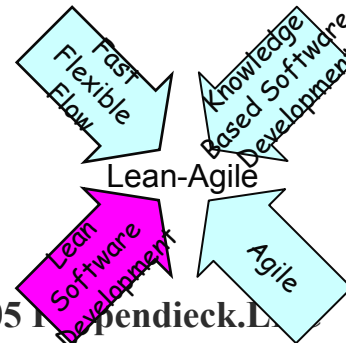
Applying Lean thinking

- Applying Lean thinking to manufacturing creates a Production *System* which creates many of the same type defect free, but which can accommodate many types
- Applying Lean Thinking to Software development creates a Production *Process* which accommodates and creates many types of software near defect free

Principles of Lean Software Development

- Optimize the Whole
- Eliminate Waste
- Build Quality In
- Defer Commitment
- Respect People
- Create Knowledge
- Deliver Fast





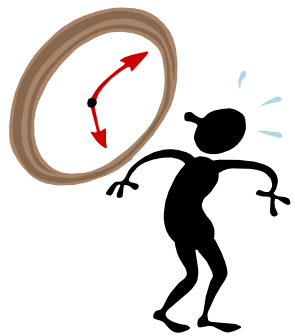
Principle: Respect People

"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



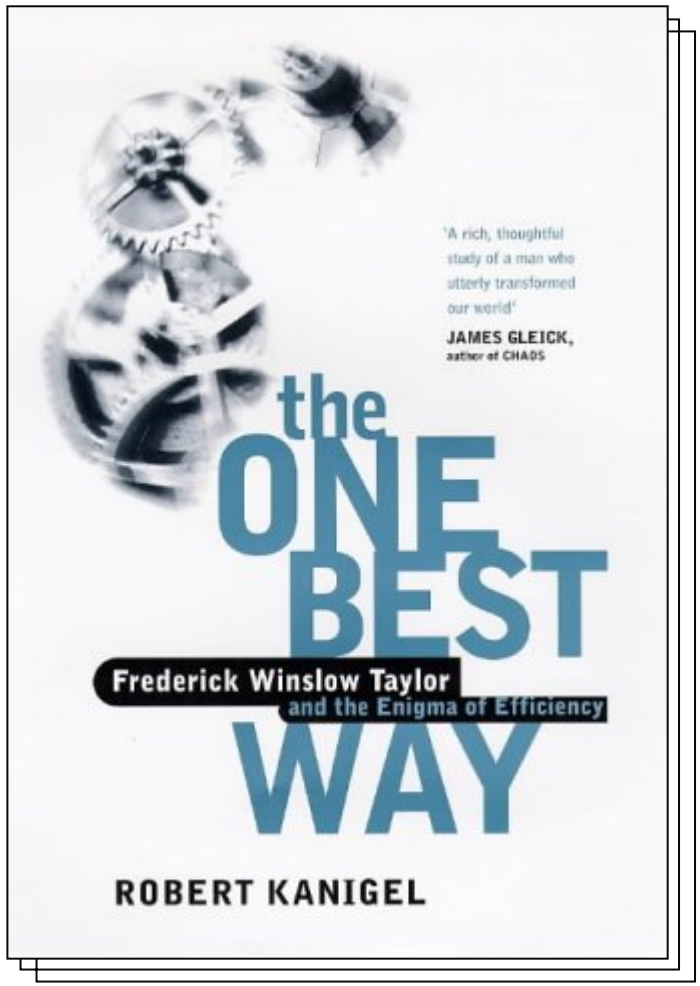
Enable Local Decisions

- When to Plant Maize
  - Local authority – Squanto
    - Advice – "plant corn when the oak leaves were the size of a squirrel's ear"
    - Actually, many factors involved – this just the trigger
  - Universal Measure – Fanner's Almanac
    - Typical local edition would be "planting corn in May after the first full moon or after May 20".
    - Requires adjustment, even by region.
    - Is safer – but loses planting time
- Seafaring Captains use Pilots when get close to a harbor
- Airlines
  - Flight Attendants

Myth: There is One Best Way

- Frederick Winslow Taylor:
- Grandfather of the Process Police

- Taylor's View of Efficiency
  - Employers get higher profits
  - Workers get higher pay
- But Scientific Management
  - Affronts human dignity
  - Discourages worker creativity

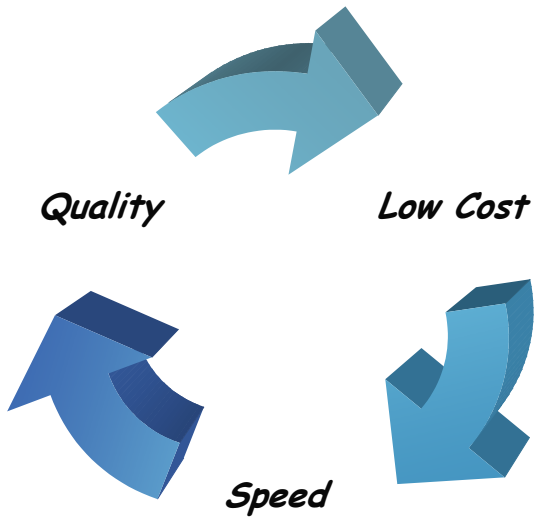


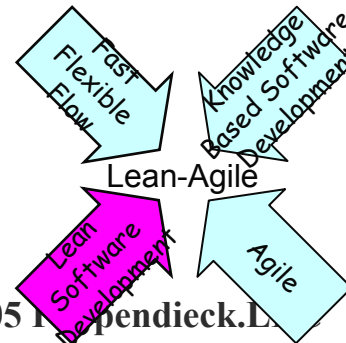
The Eighth Waste – Lost Knowledge

- Knowledge discovered but not shared
- Person who needed it didn't look
- Person who knew it wasn't consulted
- Knowledge forgotten

Principles of Lean Software Development

- Optimize the Whole
- Eliminate Waste
- Build Quality In
- Defer Commitment
- Respect People
- Create Knowledge
- Deliver Fast





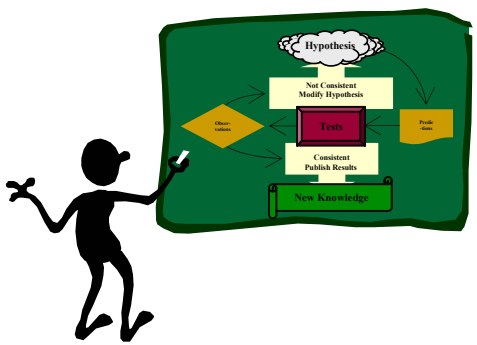
Software Development As Product Development

~C

- Software is more of a discovery process.
- Parallels the work of designing a product prior to manufacturing.
  - A useful definition of Product Development is that it is the collective activities, or system, that a company uses to convert its technology and ideas into a stream of products that meet the needs of customer and the strategic goals of the company – Product Development for the Lean Enterprise, Michael Kennedy.

Principle: Create Knowledge

A Tale of Two Projects



One Year Later….

- Get requirements
- Plan it all out

- Scientific Method
- Do something, see how it worked

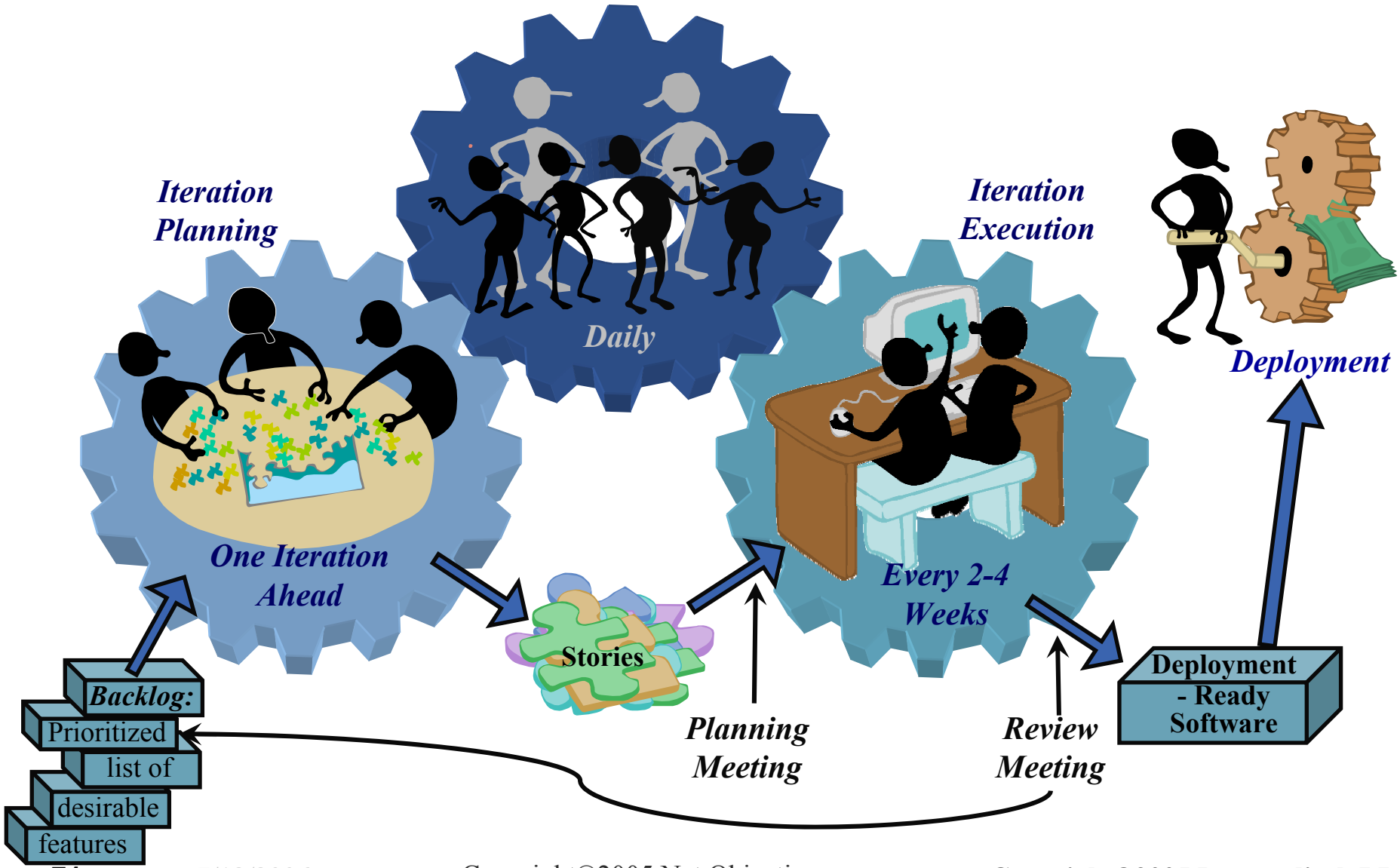
Alan MacCormack
Harvard Business School

One Year Later…

- **Team that figured it out**
  - Poor productivity
  - Poor market acceptance
  - Poor expert quality rating
- **Team that tried and learned**
  - Great market success

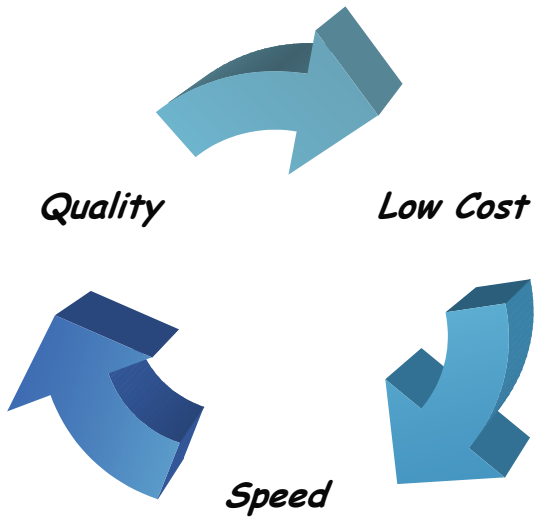
Alan MacCormack
Harvard Business School

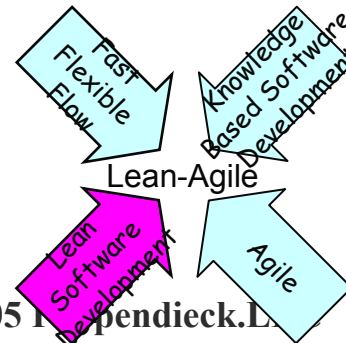
Iterative Development



Principles of Lean Software Development

- Optimize the Whole
- Eliminate Waste
- Build Quality In
- Defer Commitment
- Respect People
- Create Knowledge
- Deliver Fast





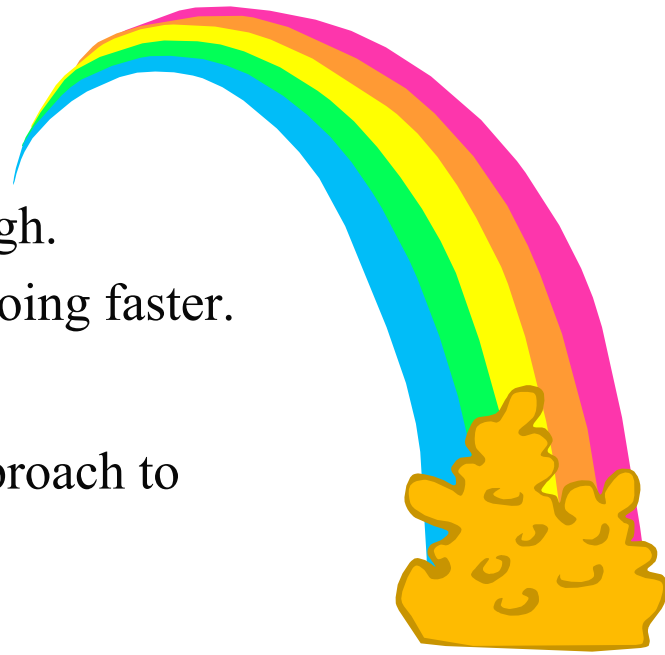
Case Study - PatientKeeper

- **Speed to market**
  - 45 cycles while competition does one
    - Maintenance releases once or twice a week
    - New feature releases every month
    - New applications released every quarter
- **Predictable Delivery**
  - Never a late release
  - Problems are seen long before the release date
  - The company self-organizes around the problems
- **Pull from Demand**
  - Priorities reorganized on a weekly basis by CEO, sales, and account management
  - Customer impact and schedule impacts are dealt with at the time of the decision
- **No Abnormalities because rapid cycle time:**
  - 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 a constant flow of mandatory upgrades
  - It forces standardization of software via new features rather than customization

Jeff Sutherland
CTO PatientKeeper

Myth: Haste Makes Waste

- Companies that compete on the basis of speed:
  - Enjoy a significant cost advantage relative to peers
    - A 25-30% cost advantage is typical.
    - Dell claims a 50% cost advantage.
  - Have extremely low defect rates
    - It is impossible to go fast unless quality is high.
    - Attacking quality issues is the first step for going faster.
  - Develop a deep customer understanding
    - Fast companies can take an experimental approach to product development
    - Fast companies fail fast and learn quickly
  - Have a sustainable competitive advantage.



Time-to-Market

- Decreased Time-to-Market
  - Greater market share
  - Greater profit margins
  - Lower investment
  - Faster return on investment
  - Lower risk
  - Longer product lifetime
  - Increased customer satisfaction
  - Increased bandwidth to pursue more markets

- Disclaimer: Assumes the right product

Lean-Agile Connection

Agility In The Context of Lean

info@netobjectives.com
www.netobjectives.com

Lean Process

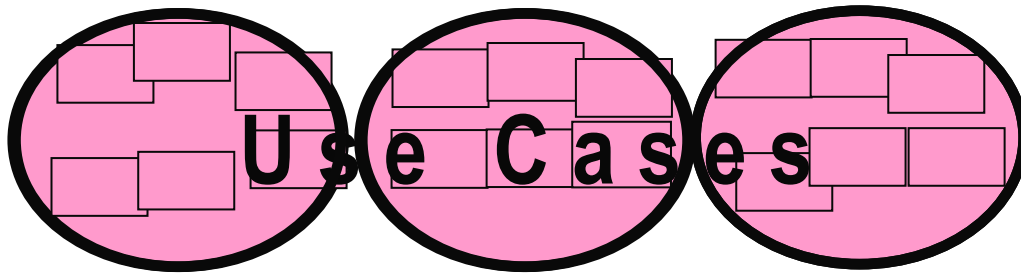
- Let's contrast a software development process based on the notion you can plan ahead (classic push) …
- with one based on Lean (a classic pull system).

Building Functionality With The Waterfall Methodology

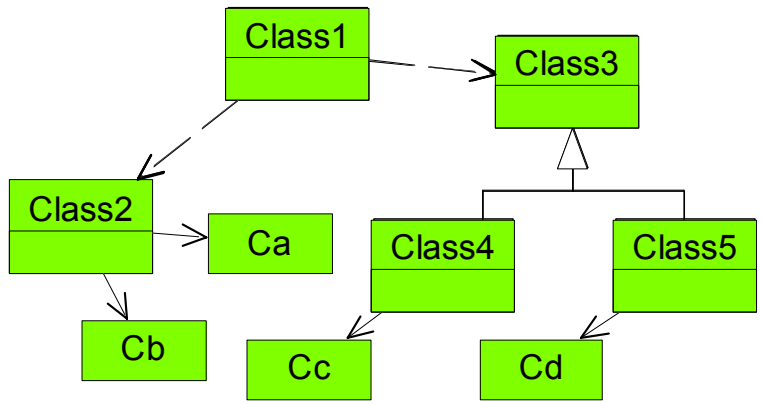
- Waterfall method has us do the following:
  - Significant requirements conversations
  - Virtually complete analysis of the use cases
  - Develop an architecture for the application
  - Create local designs
  - Implement the designs
  - Test the code (if there is time remaining)

Waterfall: Use Cases, Stories and Architecture

Define our use cases
Complete our analysis
Break out into stories
Architect the System
Refine the design



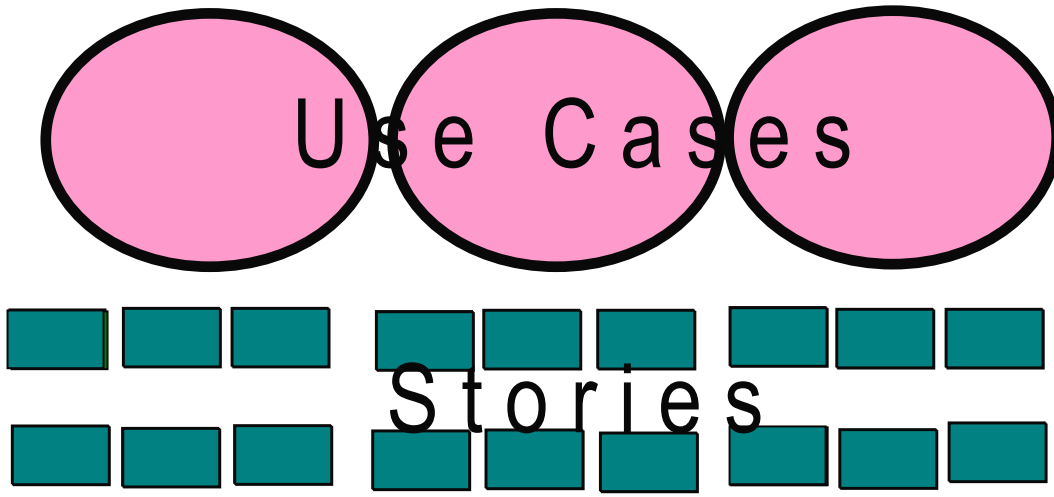
Stories



Analyzed | Architected | Designed

Waterfall: From Architecture Through Test

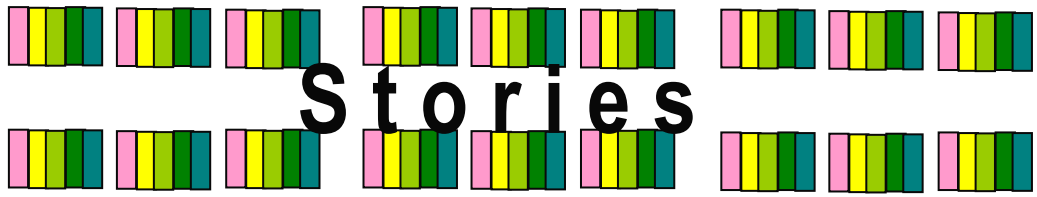
Define our use cases
Complete our analysis
Break out into stories
Architect the System
Refine the design
Finish Coding
Finish Testing

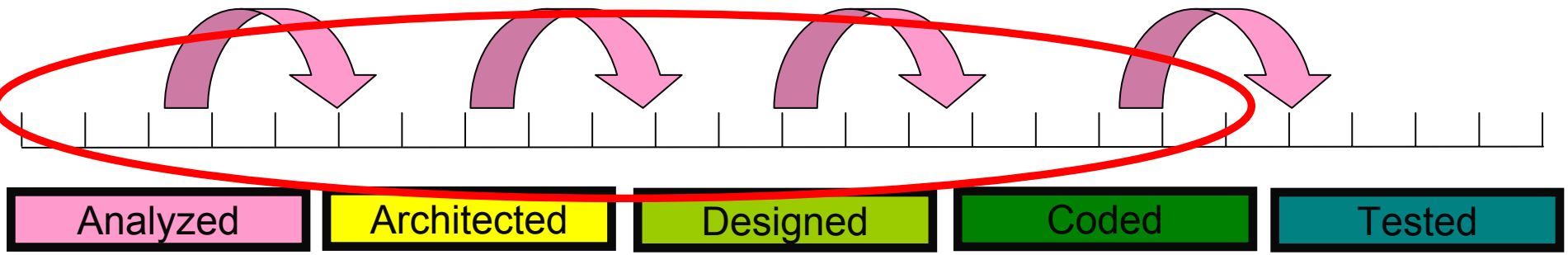


Analyzed | Architected | Designed | Coded | Tested

Waterfall: Where's The Problem?

- Stories by work type
- Timeline of work
- Where's the waste?
  - Delays
  - Task Switching
  - Work in progress





Predictability

- Future product development tasks are driven primarily by the results of the prior tasks, and therefore can't be predetermined. A better paradigm is to distribute the planning and control to those who understand and can react to the interim results.

Lean: Story Centric Process
Iteration Planning

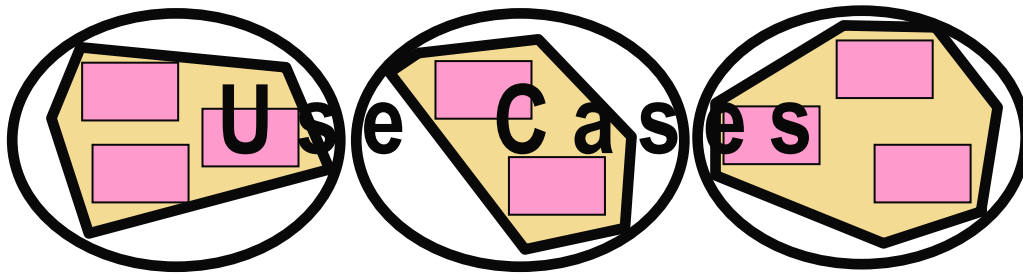
Define our use cases

Initial analysis

Identify some stories

Select stories to work on

Do Commonality-Variability Analysis



Stories

Lean: How To Pick Stories

- Pick Stories to:
  - Mitigate risk
    - Help select system architecture
    - To build application architecture
    - Customer feedback required
  - Build most valuable software for customers

Lean: Iteration – Build The Stories

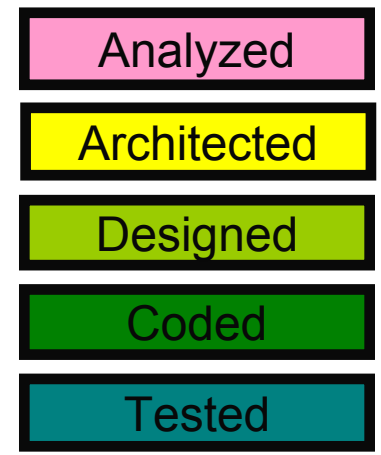
Stories by work type

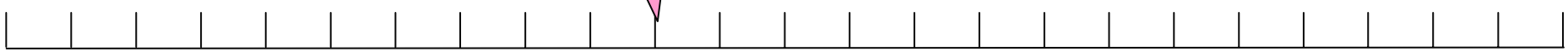
Timeline of work

- **No delays**
- **No task-switching**
- **Little work in progress**

Opportunity to Adjust

- **New stories become available**
- **Stories become re-prioritized**





Lean Thinking In Agile

- Incremental product (stories) built a la “team swarm” – work cells.
- Use TOC to improve the process as you move along the project.
- People have no flexibility in following the process, but they have complete flexibility in defining the process
  - at least within the context of management goals and guidelines.

The Lean-Agile Connection

- Use Lean Thinking to:
  - Help decide/speed the process in which projects to build
    - Value Stream Maps
  - Create cross-functional teams
    - Think Work-Cells
    - Optimizing the whole
- Use Lean Software Development:
  - Build stories in a low-cost, high quality manner
- Use agile methods to:
  - Iterations with replanning
  - Synchronize stories
  - Create Management Visibility

The Purpose of Iterations

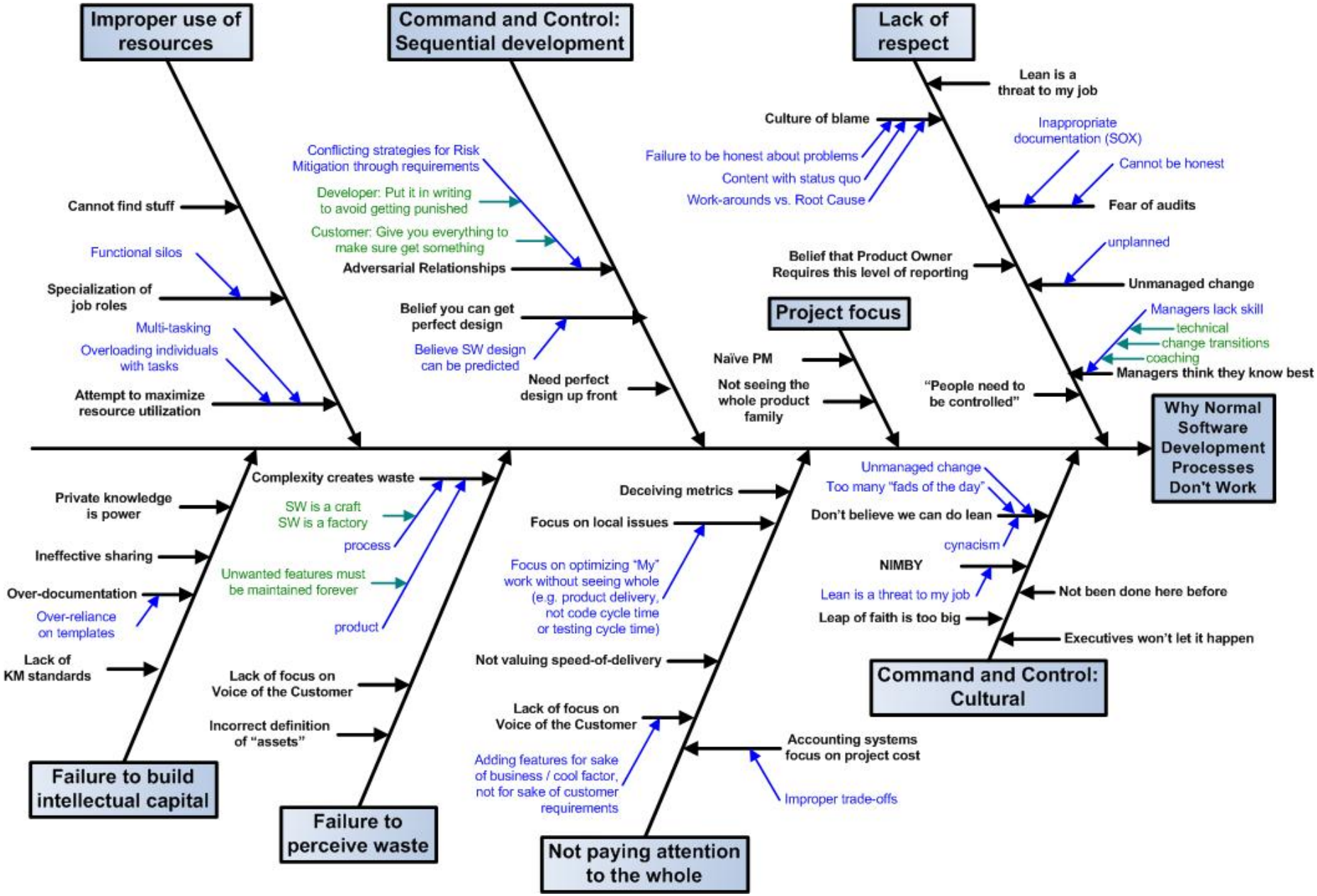
- Iterations allow us to:
  - Coordinate separate stories
  - Coordinate planning sessions
  - Get feedback to customers at predictable times
  - Synchronize story production
  - Create visibility to management.
- Iterations give the opportunity to adjust between them.

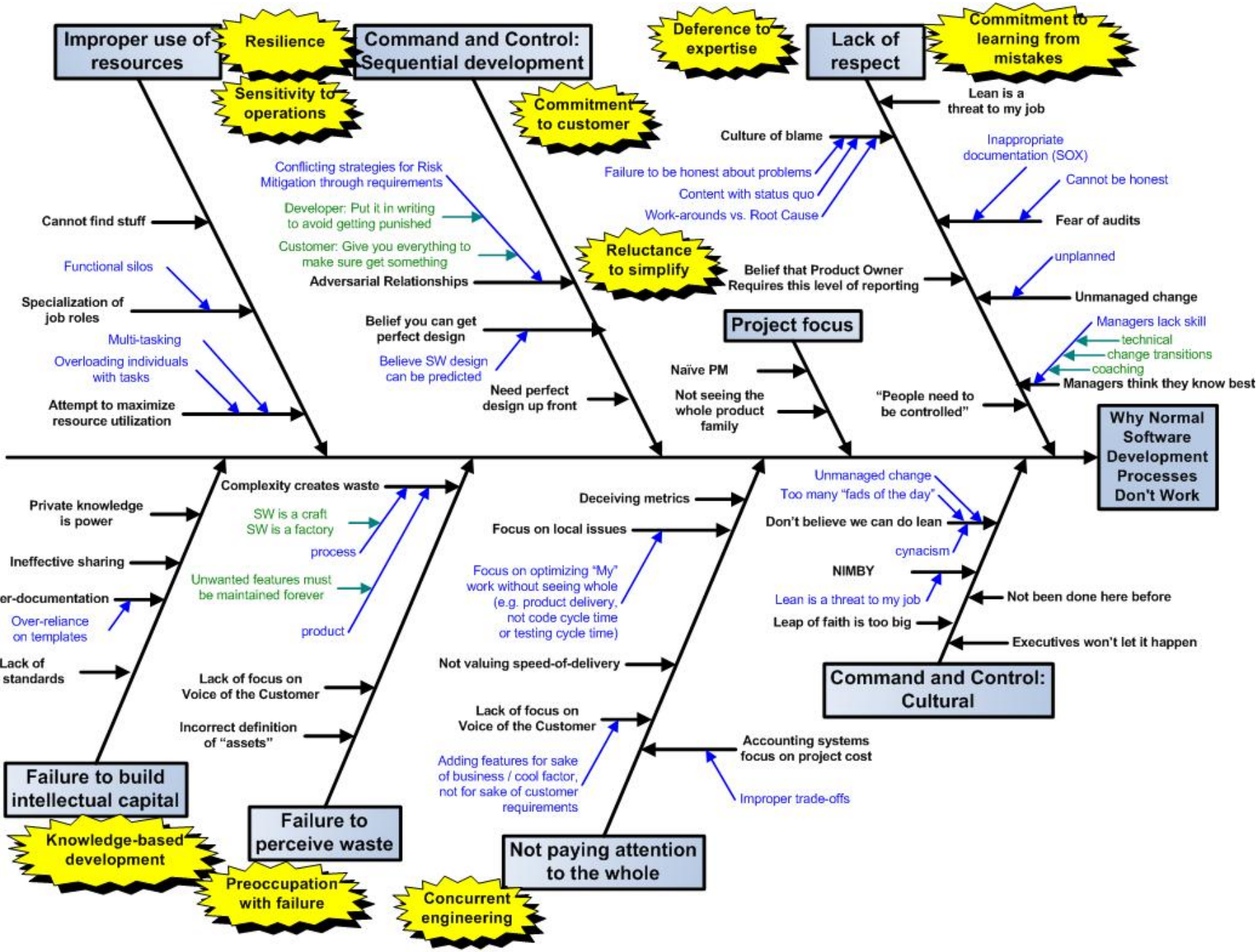
Flow in Software

- Get a feature described
  - Define test for its acceptance
  - Build it
  - Test it
  - Validate it
- Remove impediments from this flow
- Requires system of small, prioritized, stories.

Impediments to Lean

info@netobjectives.com
www.netobjectives.com





Command And Control

- Violates
  - Respect people
  - Create knowledge
  - Defer commitment
  - Eliminate waste (causes waste)
- Ignores reality that software development is not a deterministic process

Work-Arounds Vs. Fixing Root Causes

- Continues wasteful practices
- Disrespects people's time
- Creates an attitude that eliminating waste isn't important
- Can't improve ongoingly

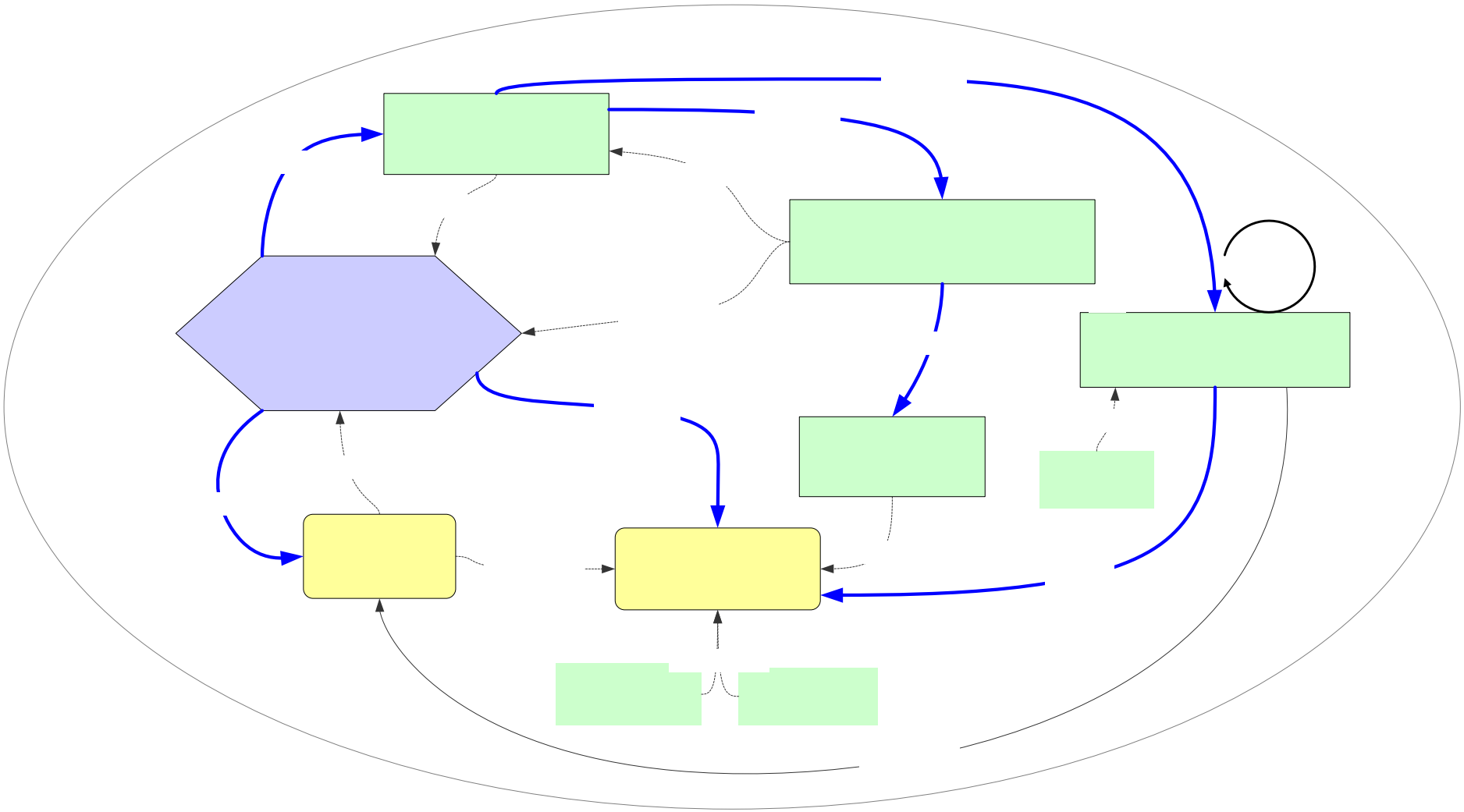
Perhaps the Biggest

- Software is different from other industries
  - We therefore believe we can't learn from other industries
  - We therefore can't learn about lean because even though it works elsewhere, it won't work here

Imagine

- Back in the 80s, when it was clear Toyota was doing something different, instead of this attitude:
  - Oh, that's because they are Japanese, they work harder and listen to what their managers tell them besides …
  - We therefore can't use what they have developed
- We said:
  - Oh, they are doing something correctly. Maybe we should learn their techniques, thoughts, process, paradigm, … and see if we can improve our own.
- What would the automotive industry look like now?
- Now imagine, what if we in software development say – "What can we learn from Lean, now?"
- Where will we be in 20 years?

Influence Diagram



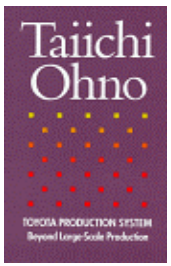
Thank You!

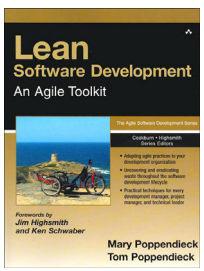
More Information: www.netobjectives.com and www.poppendieck.com

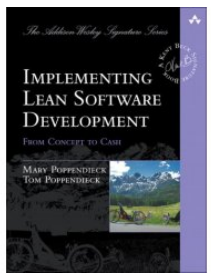
info@netobjectives.com
www.netobjectives.com

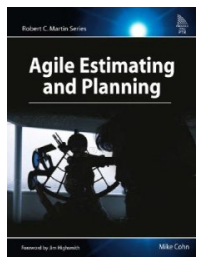
A Short List of Books

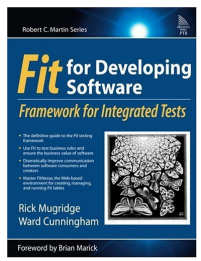
1. Taiichi Ohno – *Toyota Production System*
2. Mary & Tom Poppendieck – *Lean Software Development*
3. Mary & Tom Poppendieck – *Implementing Lean Software Development: From Concept to Cash*
4. Michael Kennedy – *Product Development in the Lean Enterprise*
5. Mike Cohn – *Agile Estimating and Planning*
6. Mugridge & Cunningham – *Fit for Developing Software*
7. William Bridges – *Managing Transitions*
8. Geoffrey Moore – *Dealing with Darwin*

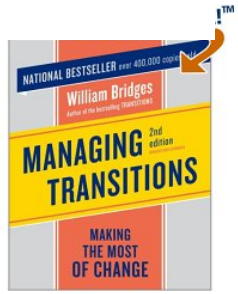


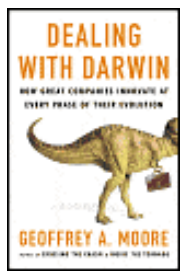












Net Objectives – Who We Are

- **Vision**: Effective software development without suffering.
- **Mission**: To assist companies in maximizing the business value returned from their efforts in software development and maintenance. We do this by providing training, coaching, and consulting that both directly assists and empowers our customers to create and sustain this ability.
- **Our staff:** Includes several recognized experts in the area of Scrum, Use Cases and Design Patterns.
- Our expertise includes:
  - Lean Software Development
  - Agile methods (Scrum, XP, RUP)
  - Agile Analysis
  - Design Patterns
  - Test-Driven Development

Net Objectives' Community of Practice

- We provide a public forum for discussion of topics including:
  - Design patterns
  - Agile methods (including XP, Scrum, RUP)
  - Use cases
  - Emergent design

- We also have several books under development on-line for review by the community.
- Go to http://www.netobjectivesgroups.com for more information.
- To subscribe to our e-zine, go to http://www.netobjectives.com/subscribe.htm and fill out our online subscription form.

Upcoming Free Seminars

- Executive Briefing On Agile Software Development – June 7, Seattle

See http://www.netobjectives.com/events/pr_main.htm for more information

Upcoming Public Courses
in California

- ScrumMaster Certification – Aug 7-8
- The Product Owner: Managing Agile Requirements – Aug 9-10
- Design Patterns Explained – Aug 15-17
- ScrumMaster Certification – Oct 10-11
- The Product Owner: Managing Agile Requirements – Oct 12-13

- All courses 9am-5pm

See http://www.netobjectives.com/events/pr_main.htm
for more information

Overview of Training

- Lean Software Development
  - Lean Software Development: A Practitioners Course
  - Lean Software For Managers
- Agile/Scrum
  - Certified ScrumMaster Training
  - Certified Product Owner: Managing Agile Requirements
  - Introduction to Scrum
- Agile Analysis
  - Effective Use Case Analysis
  - Agile Use Case Analysis
- Design Patterns
  - Design Patterns Explained
- Test-Driven Development
  - Test-Driven Development: Iterative Development With Refactoring and Unit-Testing
- Effective Programming
  - Effective Object-Oriented Analysis and Design
  - Effective Agile Programming
- ASP.NET
  - Effective ASP.NET
  - Test-Driven ASP.NET
- Integrated Custom
  - Agile Software Development with Design Patterns

Business/Corporate Perspective
Lean Software Development

- **Lean Software Development: A Practitioners Course.** Of the many methods that have arisen to improve software development, Lean is emerging as one that is grounded in decades of work understanding how to make processes better. Lean thinking focuses on giving customers what they want, when and where the want it, without a wasted motion or wasted minute. This 2-day course teaches how to apply lean principles such as: Rapid Response, Constant Learning, Built-in Quality, Local Responsibility and Global Optimization to software development by working through real problems with your peers . This course is based on the work of Mary and Tom Poppendieck.

- **Lean Software For Managers** What do PatientKeeper, a hospital data management system, and Zara, a high fashion clothing chain, have in common with Dell Computer and Toyota Motor Corporation? All four companies are overwhelming their competition with a constant flood of new products that seem to be exactly what customers want, even as they set the standard for quality and value. How do they do it? To these companies, Lean Thinking is a way of life: Rapid Response, Constant Learning, Built-in Quality, Local Responsibility and Global Optimization are part of the culture.

Project Management Perspective
Agile/Scrum

- **Certified ScrumMaster Training.** Agile project management is as radically different from traditional project management as agile processes are different from traditional methodologies. One of the most popular agile methods is called Scrum, and this course is about managing Scrum projects. Rather than plan, instruct and direct, the agile project manager (called the ScrumMaster) facilitates, coaches and leads. In this course you are certified as a ScrumMaster and learn how to make a development team, a project, or an organization agile. The course consists of lecture, hands-on discussions and exercises, case studies, and examples used to educate you in the way of the ScrumMaster.
- **Certified Product Owner: Managing Agile Requirements.** Iterative development has proven itself to be successful but, managing business expectations and driving development is still hard. We demonstrate how to strike this balance. Further, we explore the benefits and responsibilities of the Product Owner, arguably the hardest job in software development. The existence of a qualified, empowered, Product Owner with analytic support is one of the major success factors for processes.
- **Introduction to Scrum.** Learn why Scrum is an easy to apply agile software development method that can apply to large and small organizations alike. All students who successfully complete this course become Certified Scrum Masters.

Technical Perspective
Agile Analysis

- **Effective Use Case Analysis.** Use Cases are the best tool known for capturing, documenting, and validating the functional requirements for a system. Unfortunately, trying to capture a system's use cases all at once often leads to a severe case of "analysis paralysis". Our approach to use case development is an incremental one that allows us to develop, refine, and validate use cases as we move to more detailed software requirements. We call this technique the Ever-Unfolding Story, and it is the subject of this course.
- **Agile Use Case Analysis.** Agile software development is based on a simple concept: work on the most important things first, and iteratively look for what is 'now' the most important thing. By doing this a development team is able to adapt to changes as they occur, and maximize effectiveness of the development effort. We use the term "Agile Use Cases" to refer to an incremental technique for iteratively developing Use Cases and refining them into software requirements using the Ever-Unfolding Story. Agile Use Cases provide for frequent validation of the requirements, thus supporting agile development.

These courses are similar in content, but with different focus, the first is for larger, more traditional, projects; while the second is designed to support agile developments. Each of these 3-day courses consists of lecture and hands-on exercises, and the use case portion is largely based on Alistair Cockburn's book "Writing Effective Use Cases" - winner of the Jolt Productivity Award for 2001.

Technical Perspective
Design Patterns,

Design Patterns Explained: A New Perspective on Object-Oriented Design. This 3-day course goes beyond merely teaching several design patterns. It also teaches the principles and strategies that make design patterns good designs. This enables students to use these advanced design techniques in their problems whether design patterns are even present. After teaching several patterns and the principles underneath them, the course goes further by showing how patterns can work together to create robust, flexible, maintainable designs.

- **Test-Driven Development: Iterative Development With Refactoring and Unit-Testing.** 3 days. The practice of Agile Software Development requires, among other things, a high degree of flexibility in the coding process. As we get feedback from clients, stakeholders, and end users, we want to be able to evolve our design and functionality to meet their needs and expectations. This implies an incremental process, with frequent (almost constant) change to the code we're working on. Refactoring, the discipline of changing code without harming it, is an essential technique to enable this process. Unit testing, which ensures that a given change has not caused an unforeseen ripple effect in the system, is another.

Technical Perspective
Effective Programming

- **Effective Object-Oriented Analysis and Design.** This 5-day course is taught in C++, C#, and/or Java. C++, C#, and Java are powerful programming languages. Their true value emerges only if they are applied with good object-oriented practices. Without these, they can be difficult languages to maintain and extend. Although many people understand the basics of object-oriented programming (polymorphism, interfaces, encapsulation, ...) they don't understand how to use these effectively. This course starts with a better way to do analysis than is traditionally taught. It follows up with 14 practices that both enable better object-oriented coding as well as create a better understanding of object-oriented design. These practices come from both design patterns and new agile coding methods. (This course is about 40% lab)

- **Effective Agile Programming.** Requirements change. This is the new (or not so new) mantra of software development. Changing requirements mandate changeable code. This 3-day course deals with how we write changeable code. It covers issues of code quality, core coding practices, basic object-orientation, refactoring and unit-testing. It then discusses the issue of how to deal with existing legacy code.

Language Support
ASP.NET Training

- **Effective ASP.NET.** Microsoft's ASP.NET is a powerful new technology for developing web applications. Its very power of being a simple tool for building robust applications also gives it the tendency to generate maintenance nightmares, if not used appropriately. The problems begin when the application grows beyond a few simple pages. Solid design techniques must be used to keep such applications flexible and maintainable. We take the time to clearly reveal how web applications work, and how they differ from traditional GUI or console-based applications.
- **Test-Driven ASP.NET.** Test-Driven Development (TDD) is a powerful tool for combining software design, testing, and coding to increase reliability and productivity. With ASP.NET, however, Microsoft has created a tool that doesn't easily lend itself to test-driven development. Furthermore, the ease of creating ASP and ASP.NET applications has led to established applications that don't have any tests, making changes difficult and risky. Savvy developers have been looking for ways to apply test-driven development to both new and existing code so they can make changes quickly and safely. In this course, we teach you how to use NUnitAsp to perform test-driven ASP.NET. We show you "best practices" for performing test-driven development of new ASP.NET and then dive into the challenges and solutions of adding tests to existing code. We provide guidance in test-driven development, NUnitAsp, and throw in lots of pithy comments derived from years of experience with architecting web applications.

- **Agile Software Development with Design Patterns.** This 5-day course analyzes what it means to be an agile project, and provides a number of best practices that enhance agility (focusing on XP). After teaching several patterns and the principles underneath them, the course goes further by showing how patterns can work together with agile development strategies to create robust, flexible, maintainable designs.

Bibliography

Best search engine for buying books: www.bestbookbuys.com

Full bibliography at http://www.netobjectives.com/bib. Includes:

- Articles by us
- Streamzines by us
- Recommended reading list

Toyota Production System

- http://www.toyota.co.jp/en/vision/production_system/video.html