

lean  
software development

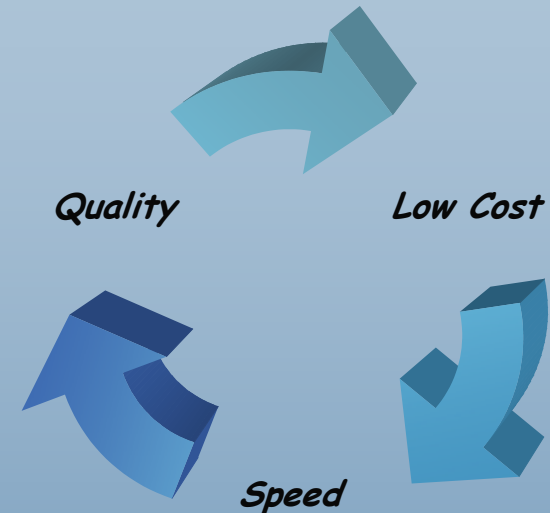
# Lean Software Development

*Speed – Quality – Low Cost*



# *Principles of Lean Software Development*

1. Eliminate Waste
2. Amplify Learning
3. Empower the Team
4. Build Integrity In
5. Delay Commitment
6. Deliver Fast
7. Optimize the Whole



l e a n



# *Principle 1: Eliminate Waste*

- *Waste is anything that does not add customer value*
  - Customers wouldn't choose to pay for it.
- 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.

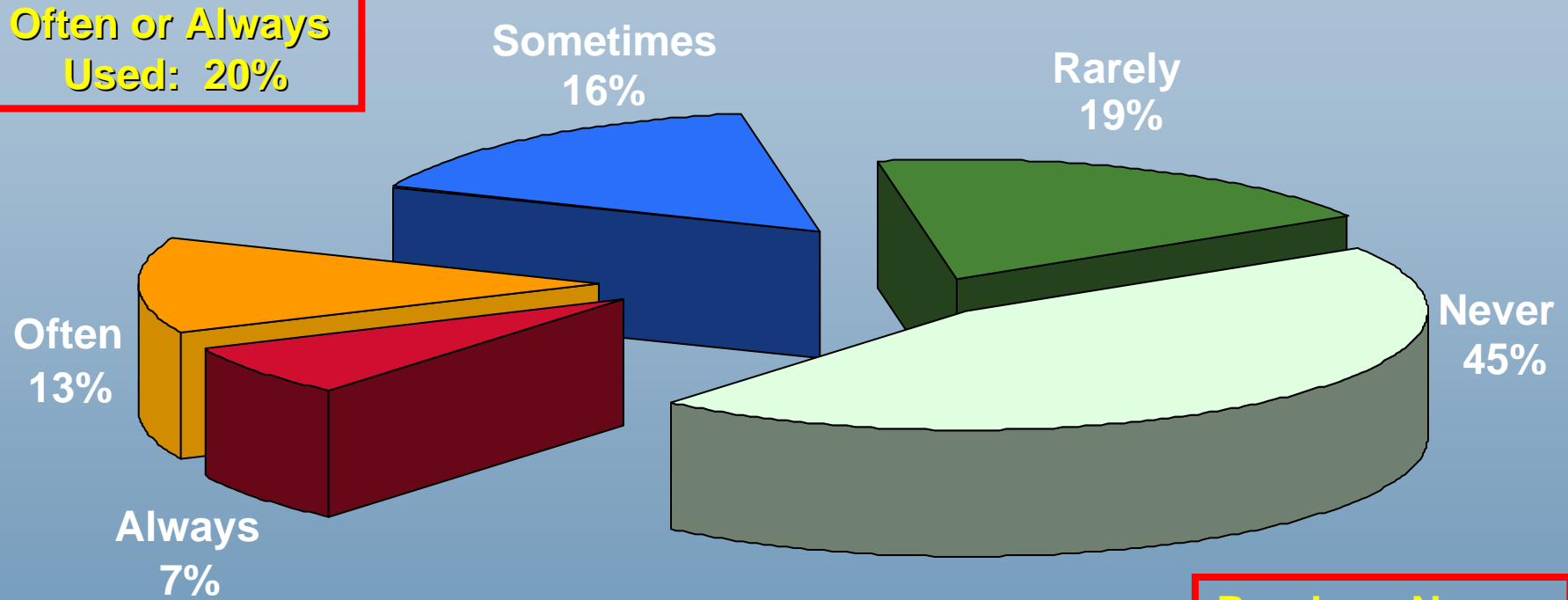




# *Myth: Early Specification Reduces Waste*

## Features and Functions Used in a Typical System

**Often or Always  
Used: 20%**



**Rarely or Never  
Used: 64%**

*Standish Group Study Reported at XP2002 by Jim Johnson, Chairman*



# *The Seven Wastes*

**MUDA**  
anything that  
does not add  
**VALUE**

## Software Development

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



# *The Seven Wastes*

## **Manufacturing**

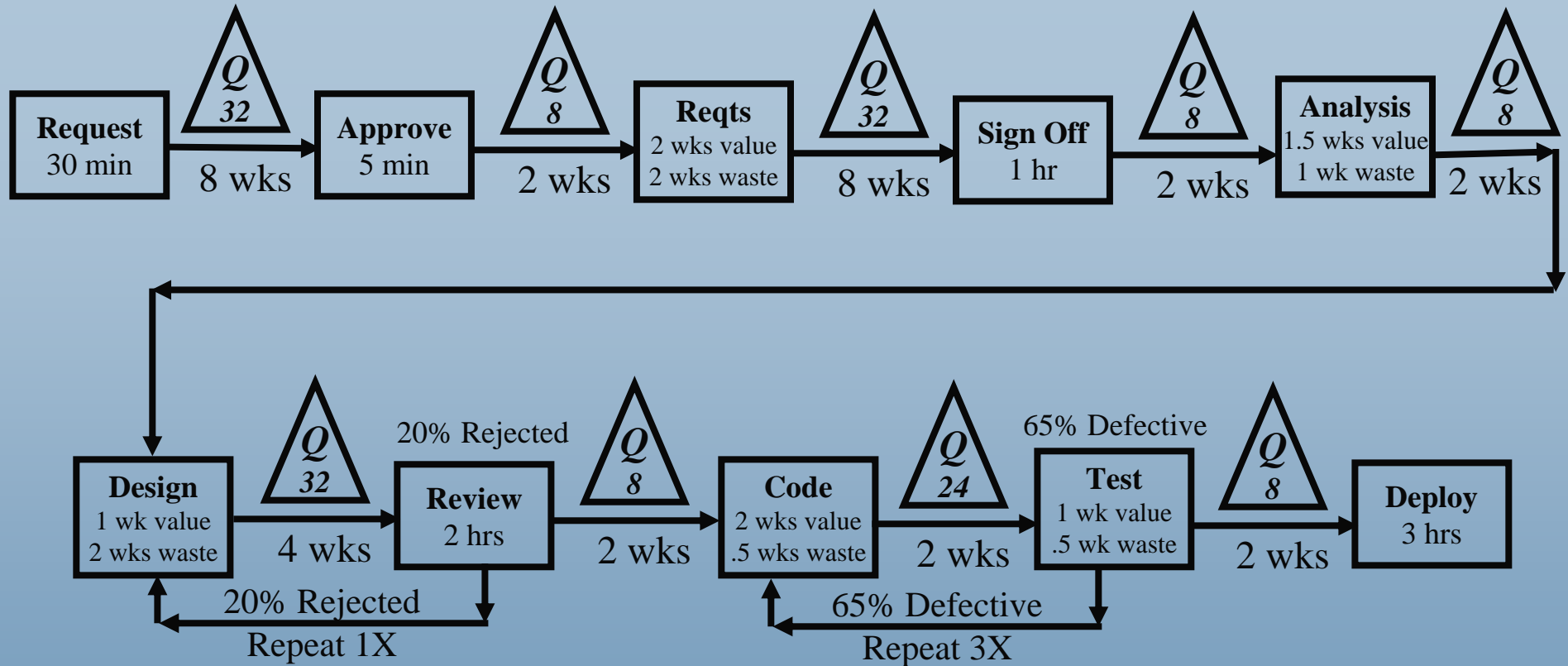
1. Inventory
2. Processing
3. Overproduction
4. Motion
5. Transportation
6. Waiting
7. Defects

## **Software Development**

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



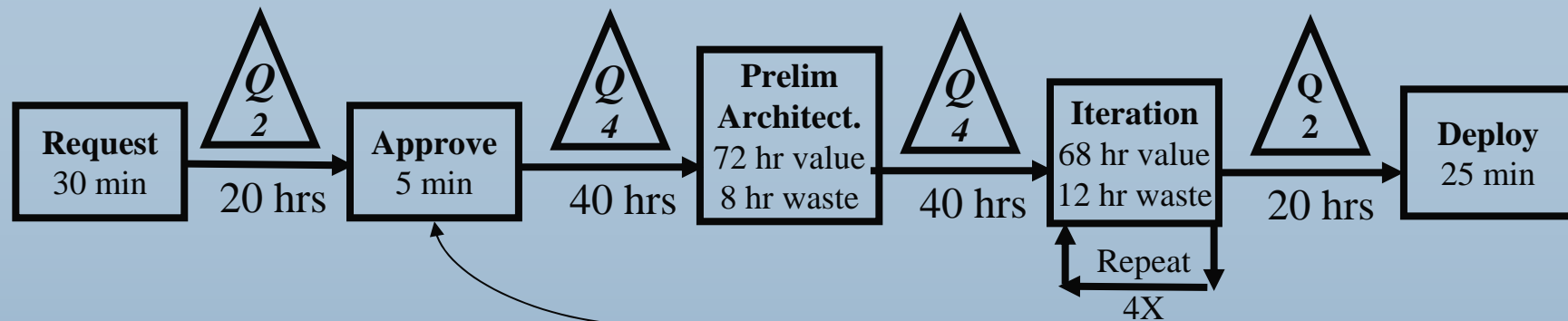
# Sequential Value Stream



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



# Lean Value Stream



## ■ Levers:

- Pipeline Management
- Development Workcell

$$\text{Process Cycle Efficiency} = \frac{\text{Value Added Time}}{\text{Total Cycle Time}} = \frac{345 \text{ hrs}}{521 \text{ hrs}} = 66\%$$



# Exercise:

## Current Value Stream Map

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

### 2. Current Value Stream Map

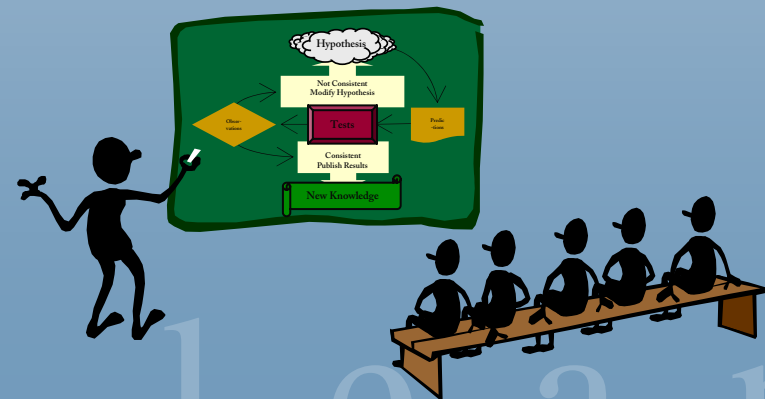
- List / diagram the key cycle steps
- List the average time of each step
  - Does the step add value full time?
  - Is the step ever repeated?
- List the average time between steps

3. Calculate Process Cycle Efficiency

$$\frac{\text{Value Added Time}}{\text{Total Cycle Time}}$$

- Add up time of each step plus time between steps = Total Cycle Time
- Add up Value Added Time in each step

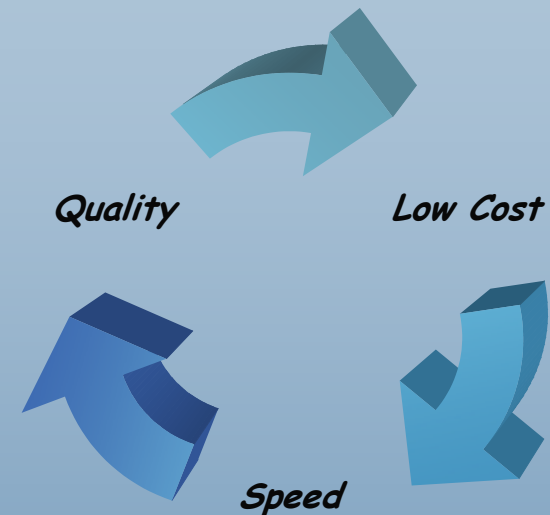
4. Report to the Class.





# *Principles of Lean Software Development*

1. Eliminate Waste
2. Amplify Learning
3. Empower the Team
4. Build Integrity In
5. Delay Commitment
6. Deliver Fast
7. Optimize the Whole

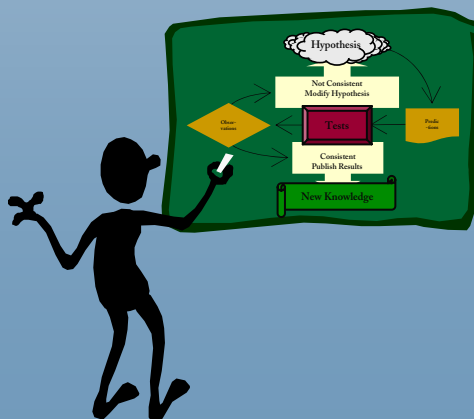


l e a n



# Principle 2: Amplify Learning

## A Tale of Two Projects



## One Year Later....

- Poor productivity
- Poor market acceptance
- Poor expert quality rating



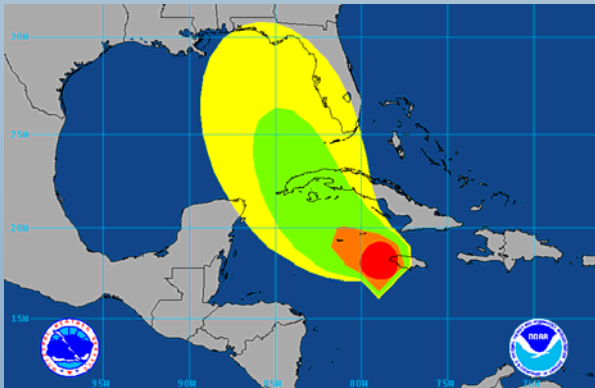
- Great market success

Alan MacCormack  
Harvard Business School

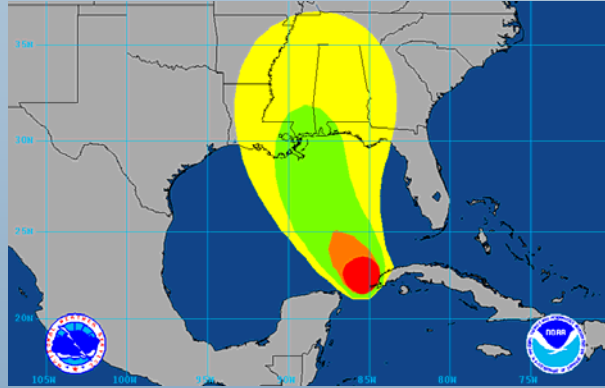


## Myth 2: Predictions Create Predictability

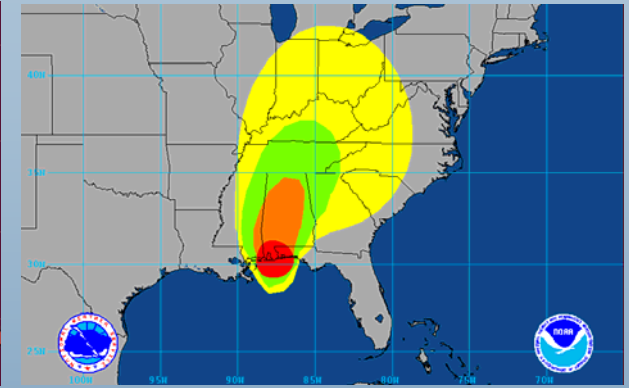
- Acting on predictions *as if they were fact* locks us into a course of action which will *probably* be wrong.



Ivan Sep 11, 2004



Ivan Sep 14, 2004



Ivan Sep 16, 2004

Probability that Eye will pass within 75 miles    **Yellow** <20%    **Green** <50%    **Orange** <100%    **Red** 100%

- Reduce response time so you can respond to events rather than predict them.
  - Consider how Dell handled the 10 day dock strike.

# Iterative Development



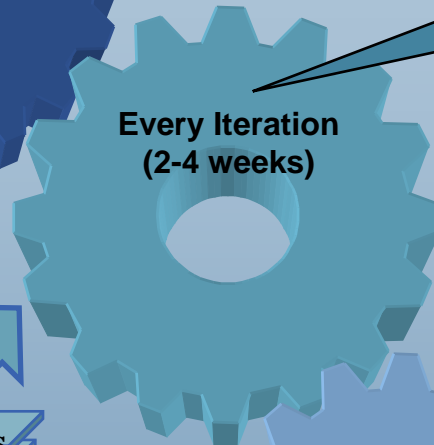
**Daily Meeting:** 15 minutes.  
Team members answer 3 questions:  
1) What did I do since last meeting?  
2) What will I do before next meeting?  
3) What obstacles are in my way?



**Iteration Execution:**  
Team designs and develops a potentially deployable increment of business value.



**Planning Meeting:**  
Team commits to iteration goal.



**Review Meeting:**  
Working software is demonstrated.

Deployment - Ready Software

**Iteration Planning:**  
Team designs next increment of business value and expands high priority features into small work packages (also called stories).



**Deployment:**  
Software delivers business value.



# What is a Workcell?



## Aircraft Sidewall Fastener

**Before: \$25-\$30**



**After: \$2.5 - \$3**



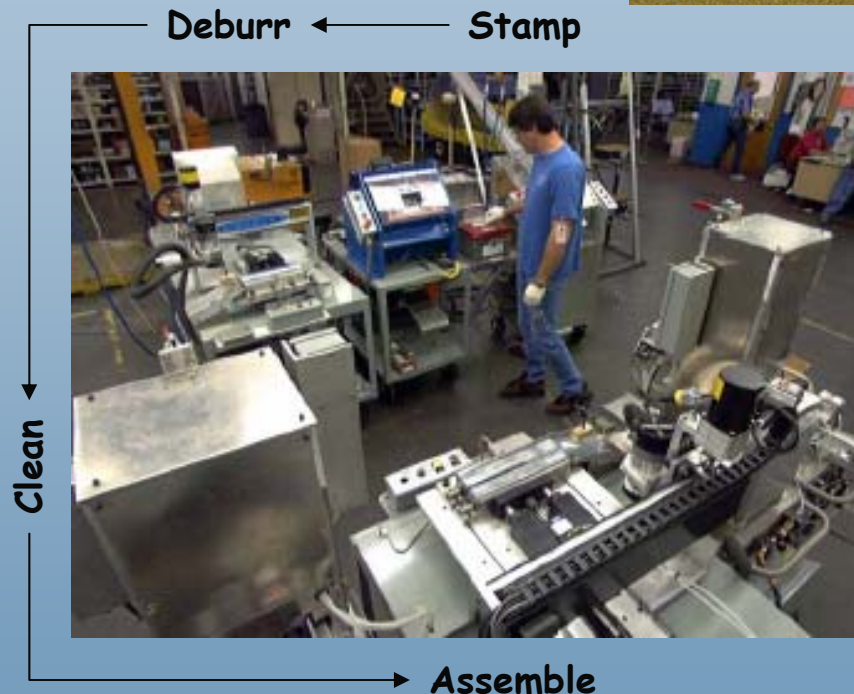
**Extrusion**



**Machining**



**Paint**



**Deburr**



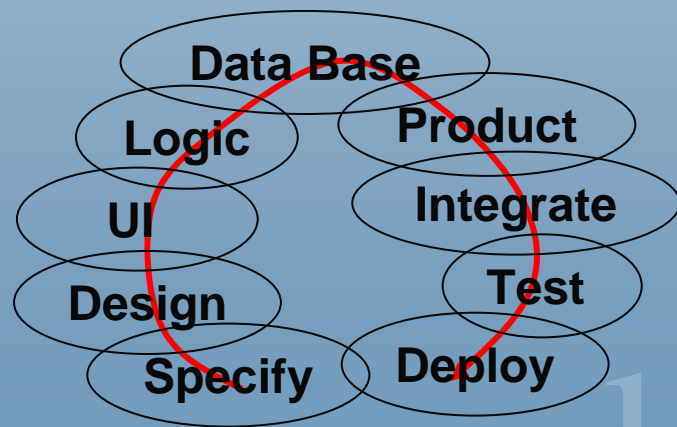
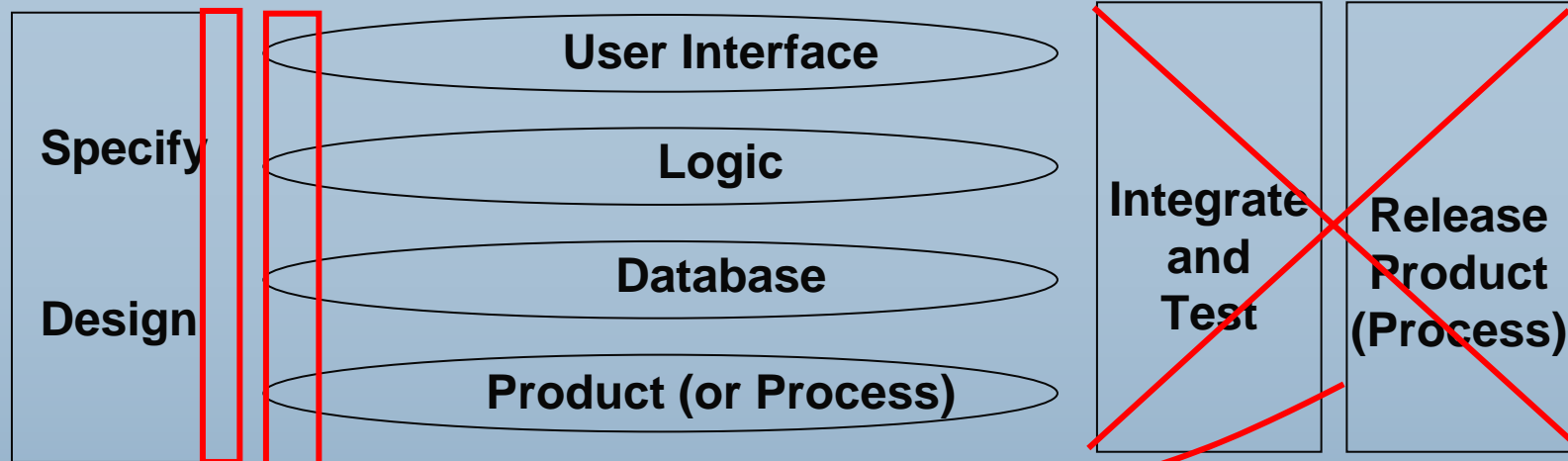
**Assemble**

Design Techniques for Meeting Market Driven Target Costs  
Benny Leppert, *Lean Design & Development*, Chicago, 2005





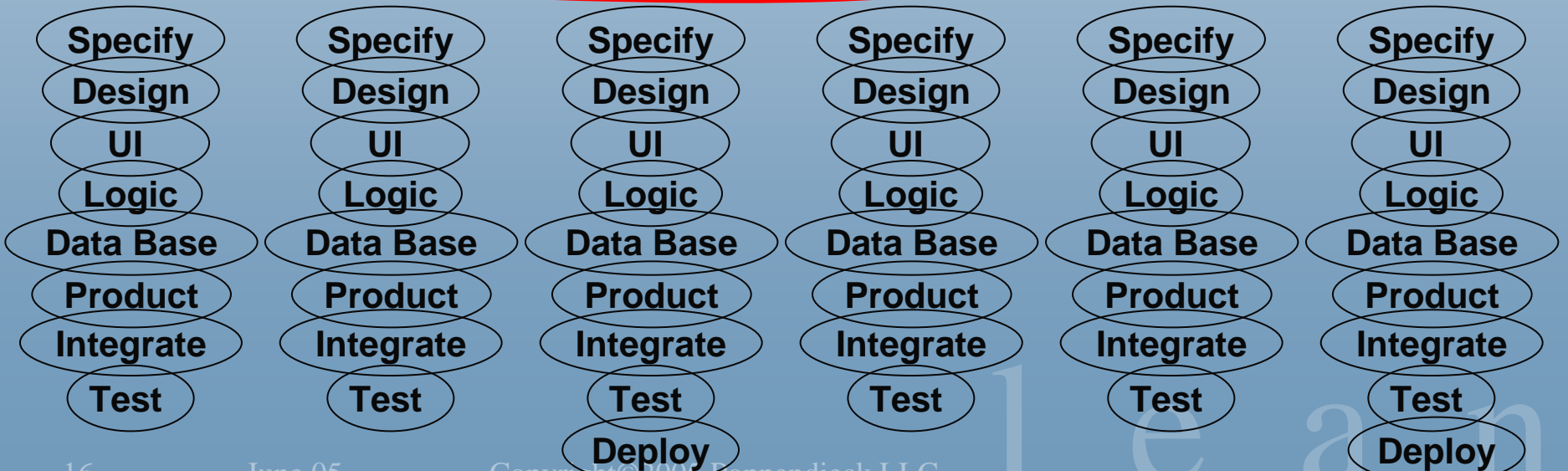
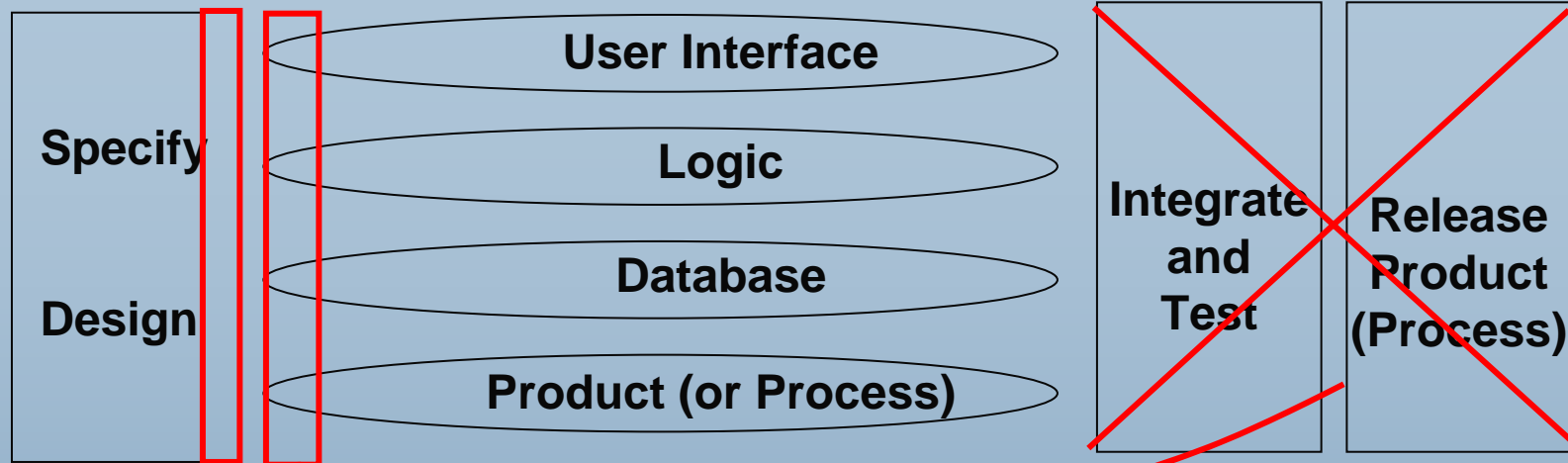
# *The Development Workcell*



lean



# The Development Workcell





## *Software Plays a Supporting Role*

- Software is rather useless
  - all by itself
- All software is embedded
  - In hardware
  - In a business process
  - In an activity (game, search)
- *The goal of software development is to make the product (process) the best it can be.*





## *The Whole Team*

- The workcell includes the whole team

Customers / Users

Product Manager

Technical Leader

Architects

Analysts

Help Desk

Operations

Maintenance

Product Designers

User Interaction Designers

Hardware Designers

Firmware Developers

Software Developers

Testers

DBA's

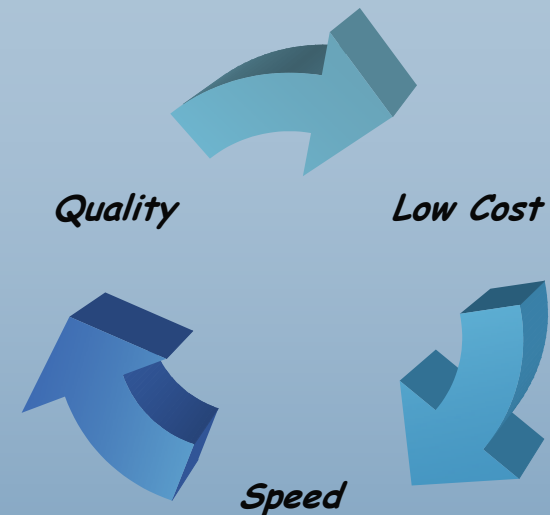
Technical Writers

- Everyone involved in the new product / business process



# *Principles of Lean Software Development*

1. Eliminate Waste
2. Amplify Learning
3. Empower the Team
4. Build Integrity In
5. Delay Commitment
6. Deliver Fast
7. Optimize the Whole





# *Principle 3: Empower the Team*

- Crisis at our Video Cassette Plant
  - Competition selling cassettes for less than we could make them
- Our Response
  - Just-in-Time (Lean) Production
- The Great Coffee Cup Simulation
  - Plant Manager, Materials Control Manager, IT Manager (me)
  - Add Production General Managers
  - Add General Supervisors
  - Add Shift Supervisors
  - Add Shift Workers
- The Result
  - Entire plant switched to pull scheduling over one weekend
  - Pack-out went from 60% to 95% the first week
  - Employees said this was the best thing that happened to the plant





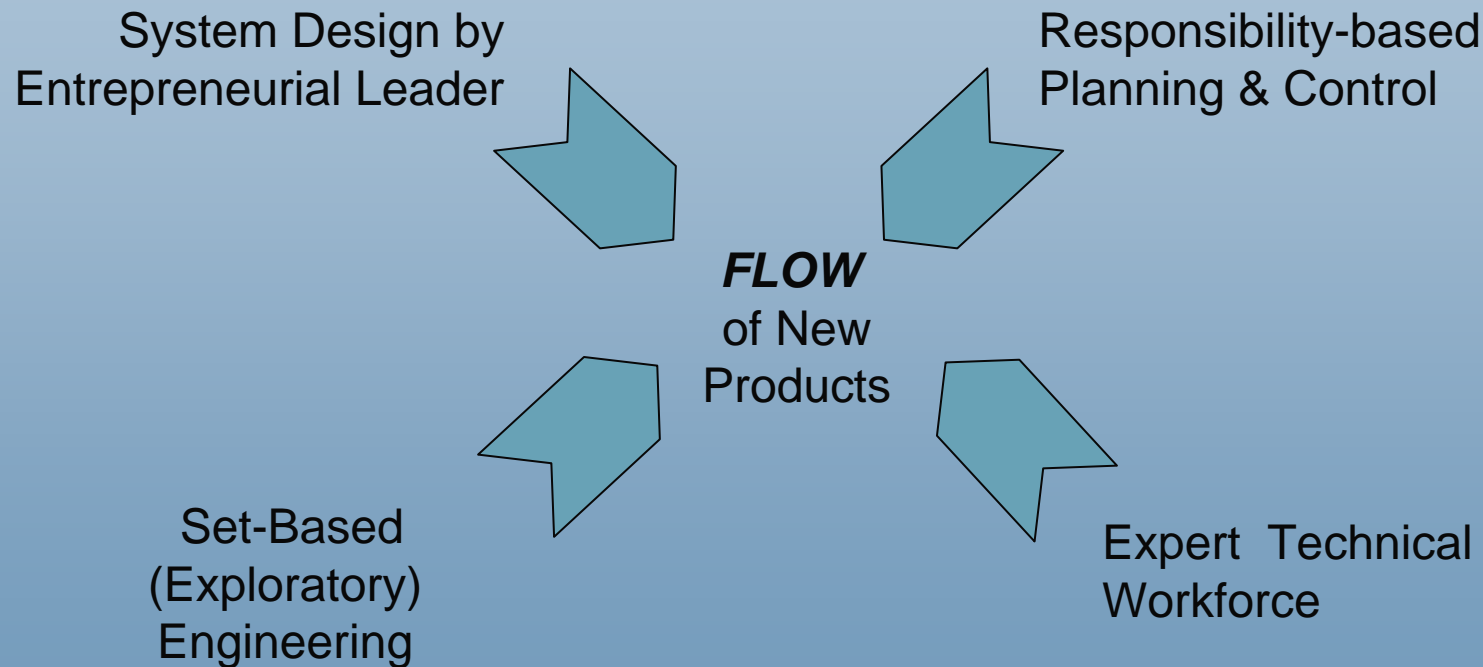
# *Toyota: The Benchmark for Product Development Excellence*

- **Faster to Market**
  - Average vehicle development time one-half that of competitors'
    - *Never* misses milestones
- **Higher Quality**
  - J. D. Power and Associates 2003 Vehicle Dependability Study:
    - Lexus is the top ranked nameplate for ninth consecutive year
    - Toyota leads with 9 models with top segment rankings – 3 times more than nearest competitor
- **Less Waste**
  - 80% of Engineers time is spend adding value, vs. 20% at competitors
- **More Innovative**
  - A leader in introducing game-changing technology
  - Licensed Prius hybrid technology to Ford
- **More Profitable**
  - Profit \$10 billion in 2004/5
    - ~ twice the *combined* profit of its four nearest competitors



# *The Toyota Product Development System*

“The real difference between Toyota and other vehicle manufacturers is not the Toyota Production System, it is the Toyota Product Development System.” Kosaku Yamada, Chief Engineer of Toyota’s Lexus line

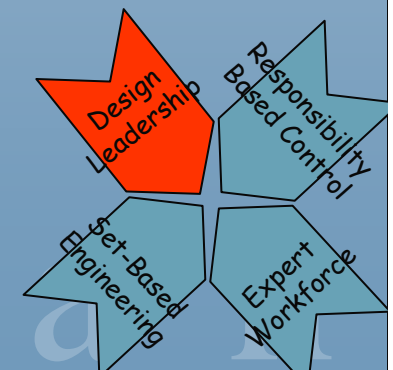


Product Development For the Lean Enterprise  
by Michael Kennedy Oaklea Press, 2003



# *Chief Engineer*

- Responsible for Business Success
- Develops Deep Customer Understanding
- Creates High Level System Design
- Sets the Schedule
- Conveys customer value proposition to engineers making day-to-day tradeoffs
- Arbitrates trade-offs when necessary
- Defends the Vision





# *Leadership Roles*

## *(How Many People?)*

### *Marketing Leader*

- Customer Understanding
- Release Planning

### *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

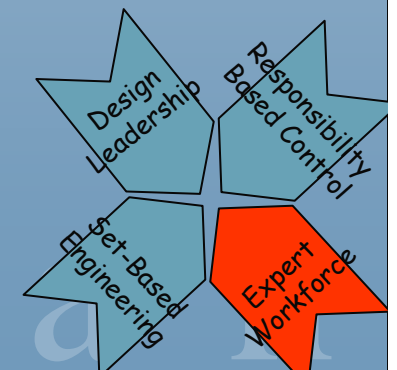
### *Project Leader*

- Funding
- Staffing
- (Scheduling)
- Tracking
- Run Interference for the Team



# *Expert Workforce*

- Early rotation for broad understanding
- Increasingly demanding assignments
- Functional manager acts as teacher
- Compensation system does not force good engineers to move into management
- 80% of the time spent “doing engineering”
- Specialists available to help with technical challenges





# *Self-Organizing Team*

- Small team
- Clear mission
- Short timeframe
- Proper leadership
- Necessary skills available
  - Technology expertise
  - Domain experience
  - All relevant functions
- Good understanding of customer needs
- Assured of getting needed resources
- Standards and disciplines in place
- Freedom to make decisions





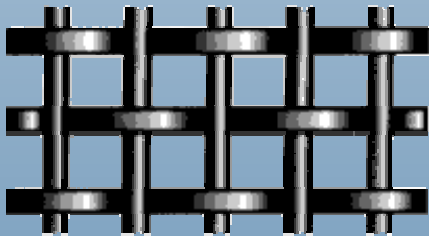
# *Communities of Expertise*

- Matrix

- Value Adding Teams
- Communities of Expertise

*Value Adding Teams*

*Communities of Expertise*



- Functional Managers

- Teacher

- Hire
- Mentor
- Set Standards
- Establish Communities

- Team Leaders

- Conductor

- Assemble the Team
- Clarify the Purpose
- Make Work Self Organizing
- See to Individual Motivation

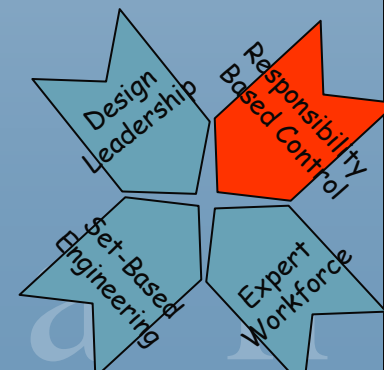


# *Responsibility-Based Planning & Control*

- Chief Engineer publishes schedule
- Engineers decide how to meet the schedule
  - Are responsible to make dependent tasks happen
  - 'Pull' knowledge as needed
- The schedule is **ALWAYS** met
  - Impossible?

## *Consider Your Daily Newspaper*

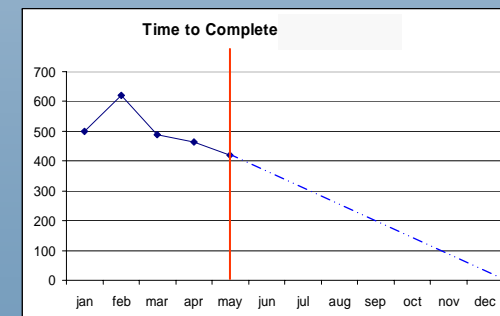
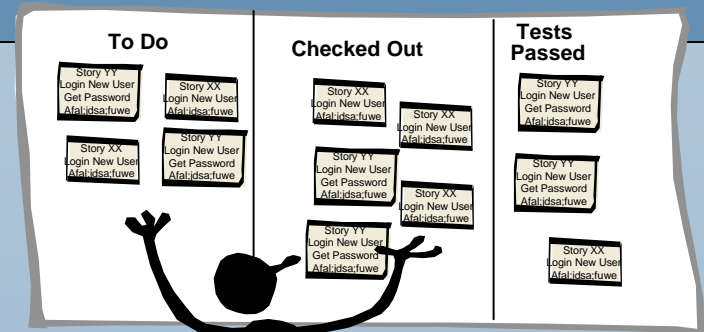
- Comes together every night no matter what
- Page Editors decide how to meet the schedule
  - Are responsible to fill their pages
  - 'Pull' stories as needed





# The Visual Workplace

- Kanban
  - What to do next?
- Stop Lights
  - Stop the Line!
- Big Visible Charts
  - How are we doing?





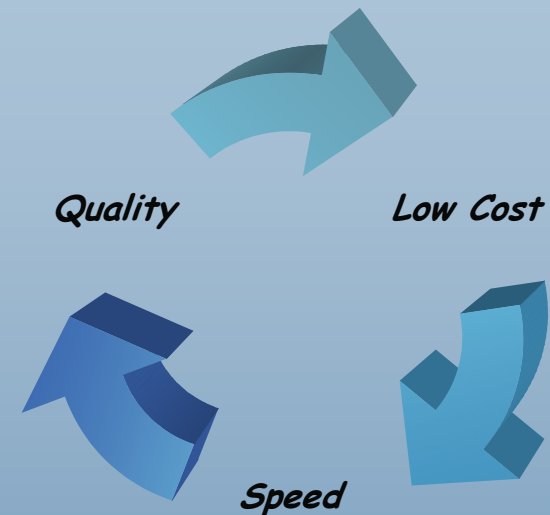
# *Visual Workspace Assessment*

	<b>Current Practice</b>	<b># of handoffs</b>	<b>Level of self-direction</b>
How do people know what customers really want?			
How do people get technical questions answered?			
How do people know what features to work on next?			
How do people know what defects to work on next?			
How do people know if tests are passing?			
How do people know their progress toward meeting the overall goal of their work?			



# *Principles of Lean Software Development*

1. Eliminate Waste
2. Amplify Learning
3. Empower the Team
4. Build Integrity In
5. Delay Commitment
6. Deliver Fast
7. Optimize the Whole



l e a n



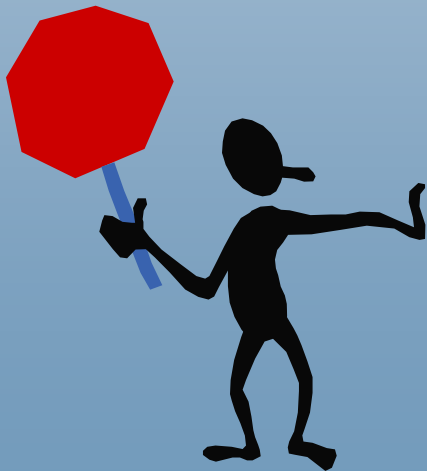
# *Principle 4: Build Integrity In*

Software Integrity

AND

- Superb Quality
  - Stop the Line

- Exceptional Value
  - The whole system works together to delight users.





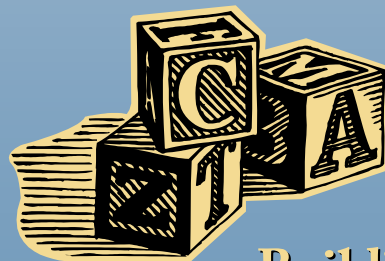
# *Building Block Disciplines*

## 1. Organized Workspace

- Clean
- Organize
- Systemize
- Maintain
- Sustain

## 2. Standardized Infrastructure

- Architecture
- Conventions
  - Naming
  - Coding
  - GUI
  - Security
  - Etc.



**Building Block  
Disciplines**

## 3. Configuration Management

## 4. Automated Tests

## 5. One Click Build

## 6. Continuous Integration

**STOP** if the tests don't pass!

## 7. Automated Release

- Mistake Proof Installation

## 8. Refactoring

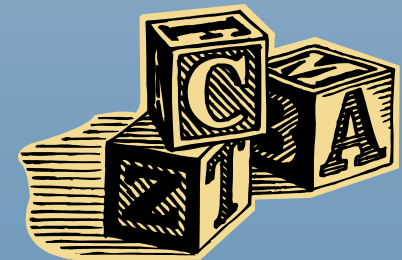
- Continuous improvement
- Pay down technical debt



# Organized Workspace

## The Five S's

- Clean (Seiso)
  - Clear out the clutter from computers and work areas.
  - Get rid of everything that is not needed.
- Organize (Seiri)
  - Use a standard filing system on servers and workstations.
  - Have a place for everything and everything in its place.
- Systematize (Seiton)
  - Standardize on software tools and configurations.
  - Agree on check-in procedures and what to do if the build breaks.
- Maintain (Seiketsu)
  - Keep computers and software up to date.
  - Backup systems and data regularly.
- Sustain (Shitsuke)
  - Make these disciplines a habit.

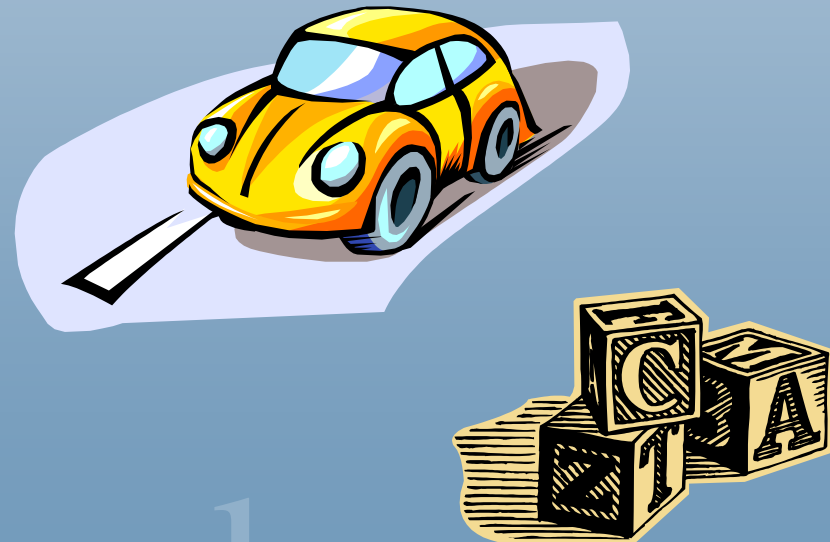




# *Standardized Infrastructure*

- Architecture
  - Vision
  - Evolution
- Standards
  - Tools
  - Conventions
    - Naming Standards
    - Coding Standards
    - GUI Standards
    - Etc.

*Good standards free people to move fast.  
Bad standards cause confusion and slow people down.*



# Types of Testing



Automated

Manual

## Business Facing

**Acceptance Tests**

Business Intent  
(Design of the Product)

**Usability Testing**

**Exploratory Testing**

**Support Programming**

**Critique Product**

**Unit Tests**

Developer Intent  
(Design of the Code)

**Property Testing**

Response,  
Security  
Scaling,...

Automated

## Technology Facing

Tool-Based

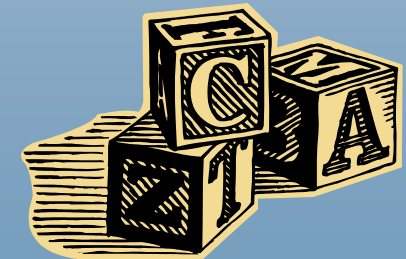
From Brian Marick



# *Automation*

## *Reliable – Repeatable – Traceable*

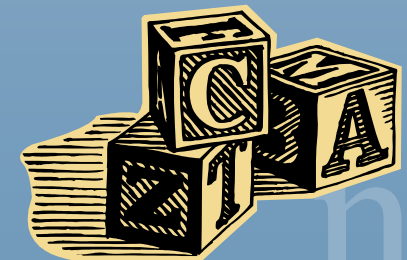
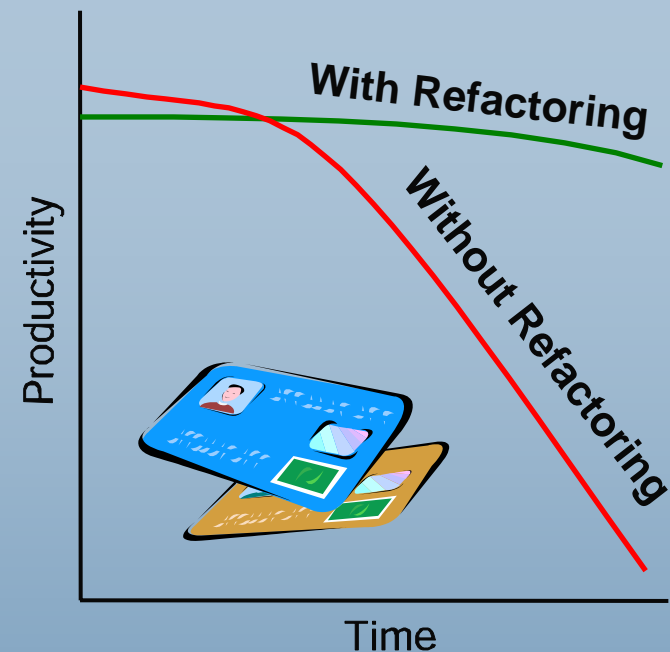
- Configuration Management
  - Subversion (<http://subversion.tigris.org>)
- Automated Build
  - ANT ([www.ant.apache.org](http://www.ant.apache.org))
- Continuous Integration
  - Cruise Control (<http://cruisecontrol.sourceforge.net>)
- Unit Testing
  - X-Unit (J-Unit for Java, etc.)
  - ([www.xprogramming.com/software.htm](http://www.xprogramming.com/software.htm))
- Acceptance Testing
  - FIT (<http://fit.c2.com>) & ([www.fitnessse.org](http://www.fitnessse.org))





# Refactoring Conquering Complexity

- Start with a minimalist design
  - Add only what is necessary
  - No features ahead of their time
  - No features after their time
- Keep improving the design
  - Simple, Clean Design
  - No Extra Features
  - No Repetition
- Technical debt
  - Incurred when you don't refactor
  - Has to be paid off sometime
  - Heavy interest rates





# Assessment: Basic Disciplines

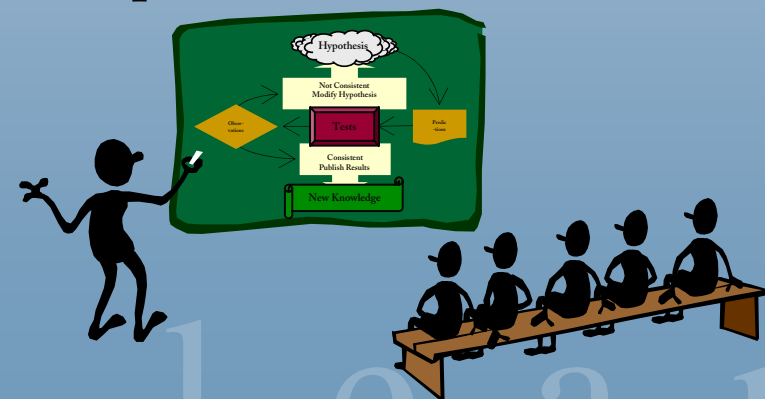
1. On a scale of 1 – 5 (high), rate these Basic Disciplines for your organization
  - Five S's
  - Standards
  - Configuration Management
  - Automated Unit Tests
  - Automated Acceptance Tests
  - One Click Build
  - Continuous Integration
  - Automated Release
  - Refactoring

Total your score.

2. Discuss:

- Who is responsible for setting standards? Who should be?
- What 'centers of expertise' exist – or should exist?

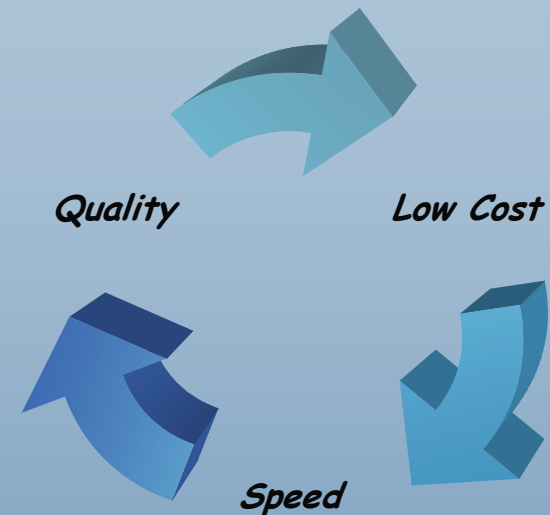
3. Report to the Class.





# *Principles of Lean Software Development*

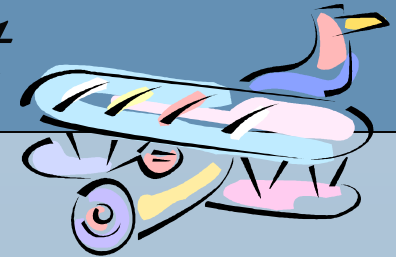
1. Eliminate Waste
2. Amplify Learning
3. Empower the Team
4. Build Integrity In
5. Delay Commitment
6. Deliver Fast
7. Optimize the Whole



l e a n



# Principle 5: Delay Commitment

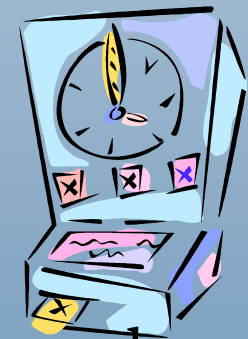


## ■ Pilots

- “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.”





# *Myth: Get it Right the First Time*

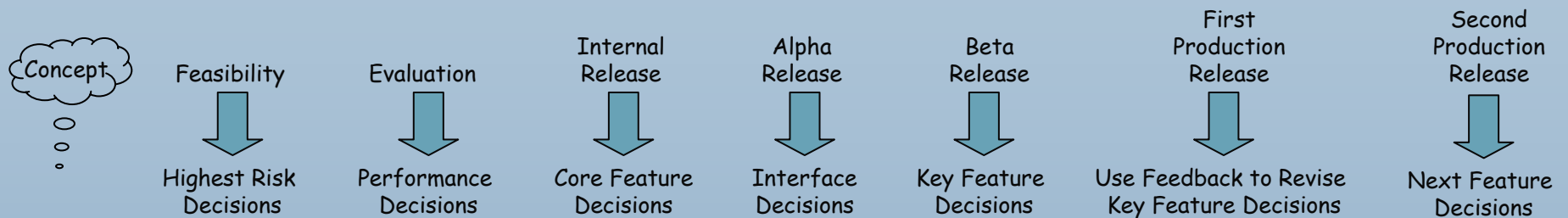
- The Software I Use:
  - Intuit Quicken ➤ 1983
  - Microsoft Word ➤ 1983
  - Microsoft Excel ➤ 1984
  - Intuit QuickBooks ➤ 1992
  - Adobe Acrobat ➤ 1992
  - Internet Explorer ➤ 1995
  - Norton AntiVirus ➤ 1995
  - Microsoft Outlook ➤ 1997
  - Google ➤ 1997

*Reality:  
Make it Better  
Every Time*

*The Goal:  
Lasting Value*

# *Schedule Decisions*

- Decisions points are scheduled ahead of time:



- Decisions are made as late as possible
- Multiple options are evaluated
- The design of the software evolves
- Decision points are hard dates
- They may be set at the prior decision point
- There is no such thing as a 'final' decision point



# 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.



# *Change-Tolerant Tactics*

## Encapsulate Variation

- Group what is likely to change together inside one module
- Know the domain!

## Simplicity

- Easy to understand

## No Repetition

- Instead of copy & paste
- Refactor!

## Separate Concerns

- A module should have one and only one responsibility
- And only one reason to change

## Delay Commitment

- Add features Just-in-Time
  - Not Just-in-Case
- Maintain Options
  - Abstract interfaces
  - Multiple prototypes



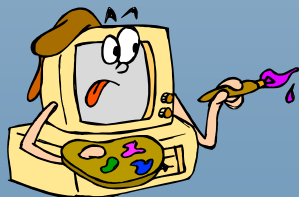
# *More Case Studies*

## Multiple Prototypes

- Medical Devices

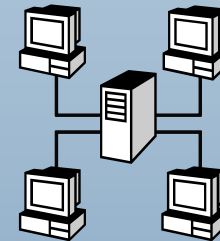


- Web Design

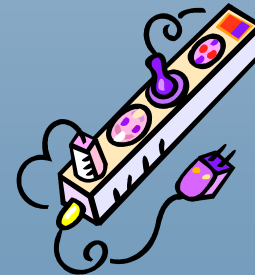


## Pluggable Interfaces

- Middleware



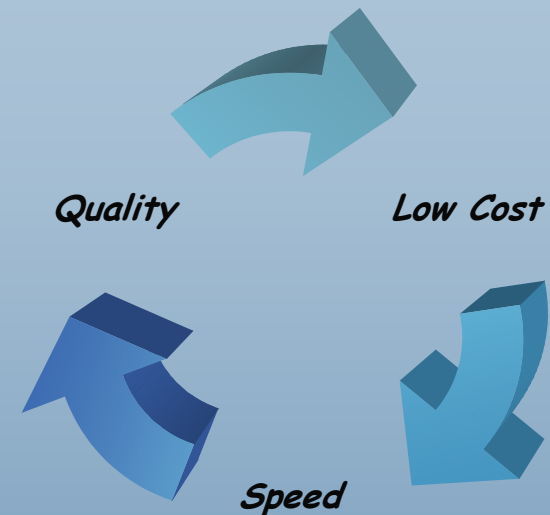
- Build-Time Options





# *Principles of Lean Software Development*

1. Eliminate Waste
2. Amplify Learning
3. Build Integrity In
4. Empower the Team
5. Delay Commitment
6. Deliver Fast
7. Optimize the Whole



l e a n



## *Principle 6: Deliver Fast*

- Rapid Delivery is the Competitive Advantage of:



Manufacturing

**L.L.Bean**

Order Processing



Shipping



Supply Chain



Aircraft Turnaround



Product Development

- Competing on the basis of time provides
  - Significant competitive advantage
  - Large barrier to entry



## *Myth: Haste Makes Waste*

- Companies that compete on the basis of speed:
  - Have 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.
    - Attaching quality issues is the first step for going faster.
  - Develop a deep customer understanding
    - Fast companies can take an experimental (data-based) approach to product development
    - Fast companies fail fast and learn quickly



# Cycle Time

- Cycle Time

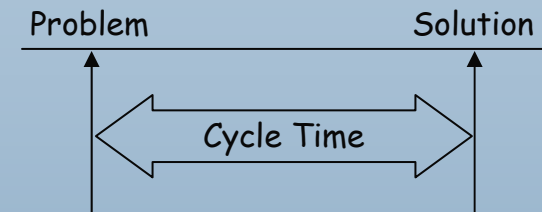
- Average end-to-end process time

- From problem detection
    - To problem solution

- Also known as: Time-to-Market

- Software Development Cycle Time

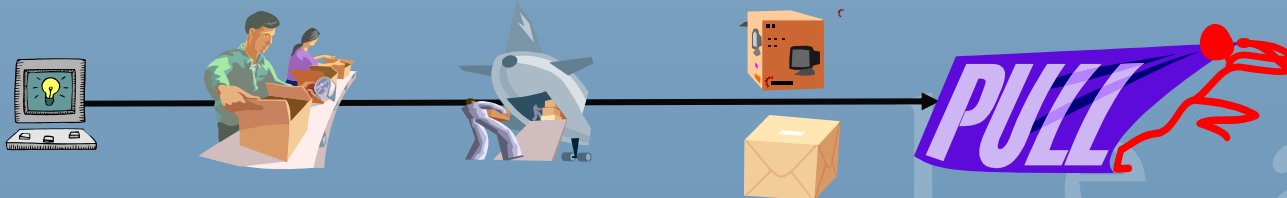
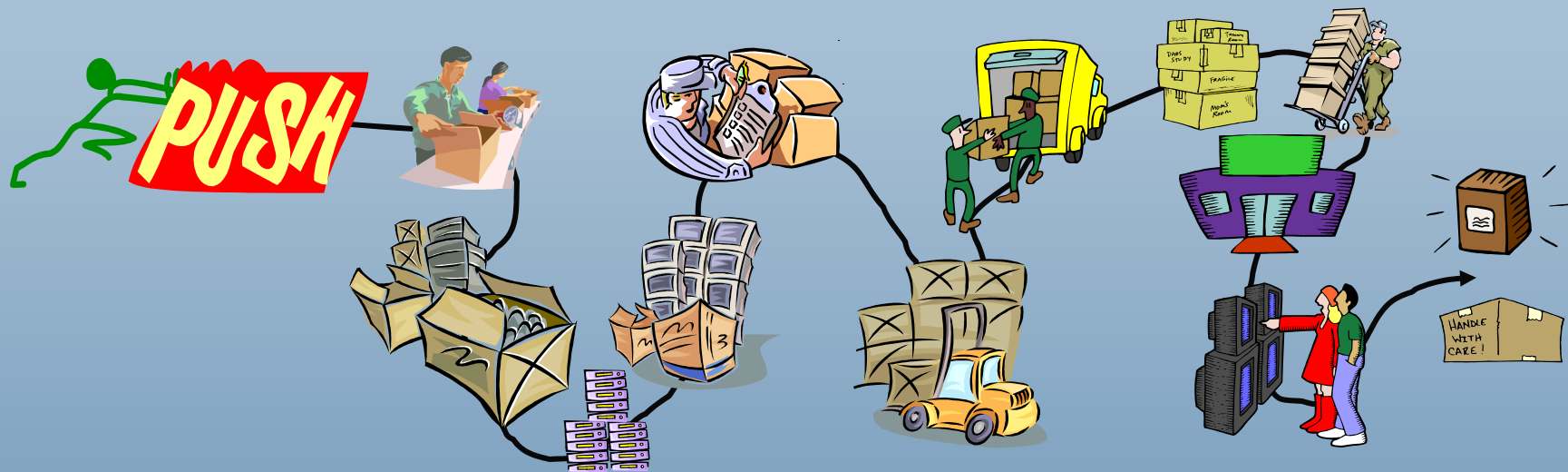
- The speed at which a customer need is reliably and repeatedly translated into deployed software.





# Little's Law

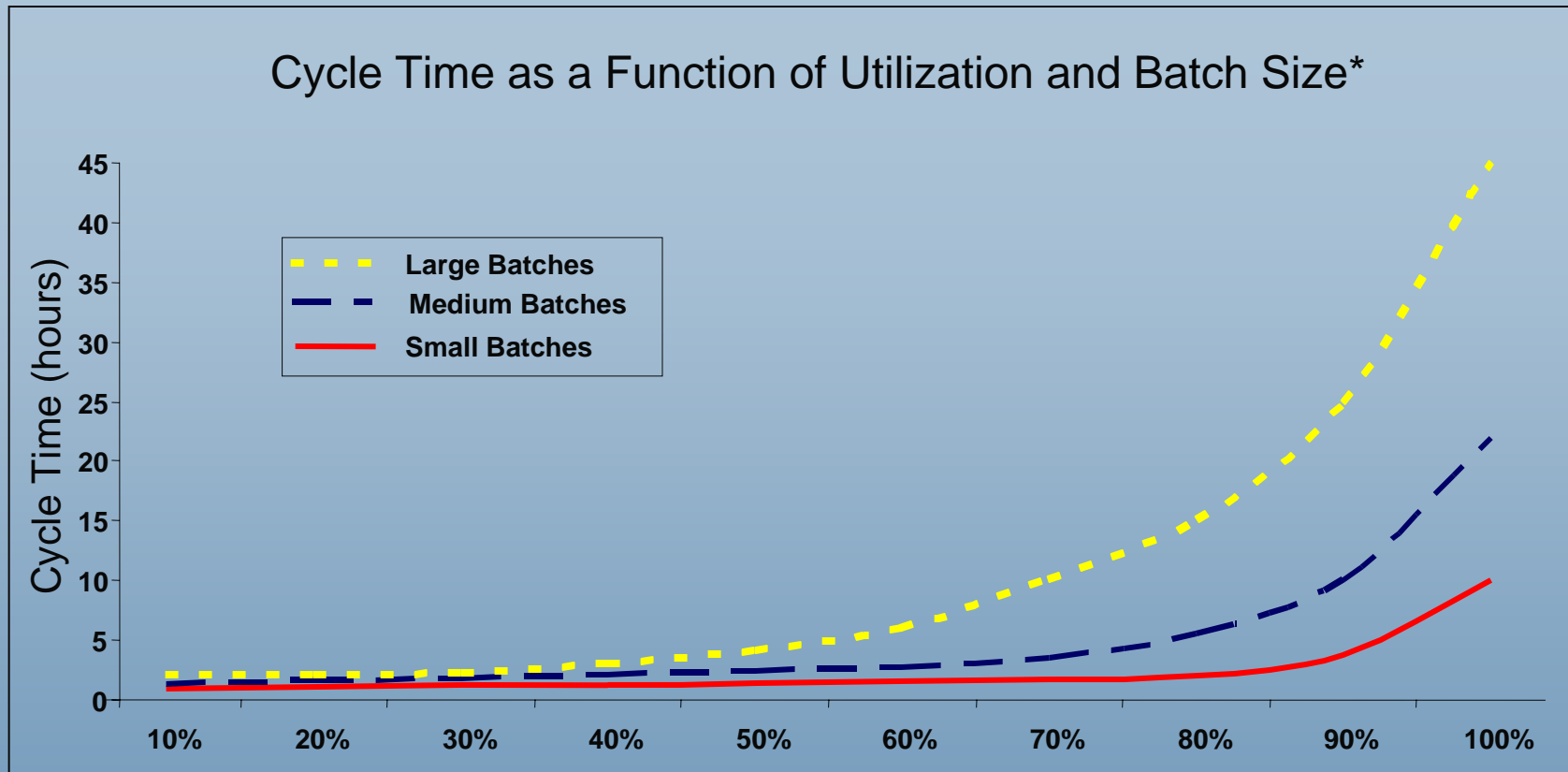
$$\text{Total Cycle Time} = \frac{\text{Number of Things in Process}}{\text{Average Completion Rate}}$$





# *The Utilization Paradox*

Cycle Time as a Function of Utilization and Batch Size\*



## *Queuing Theory 101*

\*This assumes batch size is proportional to variability.



# *Reducing Software Development Cycle Time*

1. Even out the Arrival of Work
2. Minimize the Number of Things In Process
3. Minimize the Size of Things In Process
4. Establish a Regular Cadence
5. Limit Work to Capacity
6. Use Pull Scheduling



Queuing Theory



## *Case Study: Even Out the Arrival of Work*

- Large Financial Organization
  - Most IT work came January – June
  - Business was not cyclical
  - Management Measurements created uneven demand
  - Solution:
    - Change the sub-optimizing measurements



## *Case Study: Minimize the Number of Things in Process*

- Health Maintenance Organization
  - Doctors had 60 day waiting lists
  - Reduced to 2 days
    - Limit new patients to reduce demand
    - Work off waiting list backlog over 6 months
  - Result:
    - No difference in doctor workload or types of problems seen
    - Patients seen in 2 days instead of 2 months



# *Case Study: Minimize the Size of Things in Process*

- Case Study: Randy Mott
  - CIO Wall Mart 1994 – 2000
  - CIO Dell 2000 – Present



- Measurements:

- All projects 9 months or less (Preferably 6 months)
- 100% on time delivery
- Every project tied to measurable bottom line improvement
- All software integrated into the standardized infrastructure
- All software globally useful

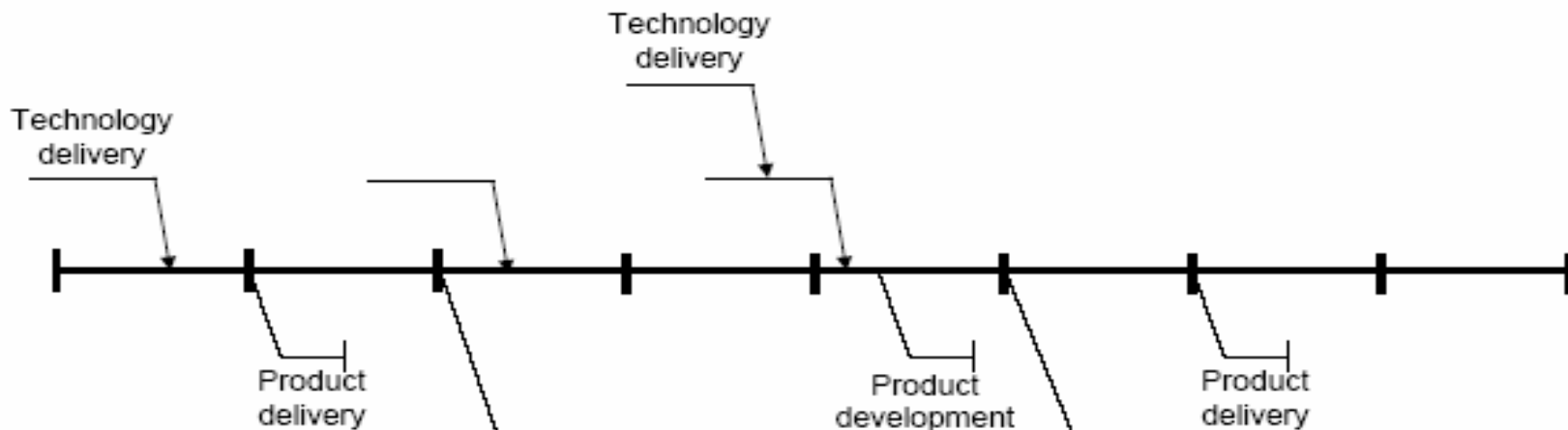
- Management:

- Global Roadmap (6-8 Quarters)
- Local Plans
- < 500 projects / year



# Case Study: Establish a Regular Cadence

## Uncouple Technology Delivery from Product Delivery



Bret Dodd  
R&D Section Manager  
Hewlett-Packard Company  
Lean Design & Development 2005

Note the consistent  
"heartbeat"

Any product class  
can take a drop and  
go to market



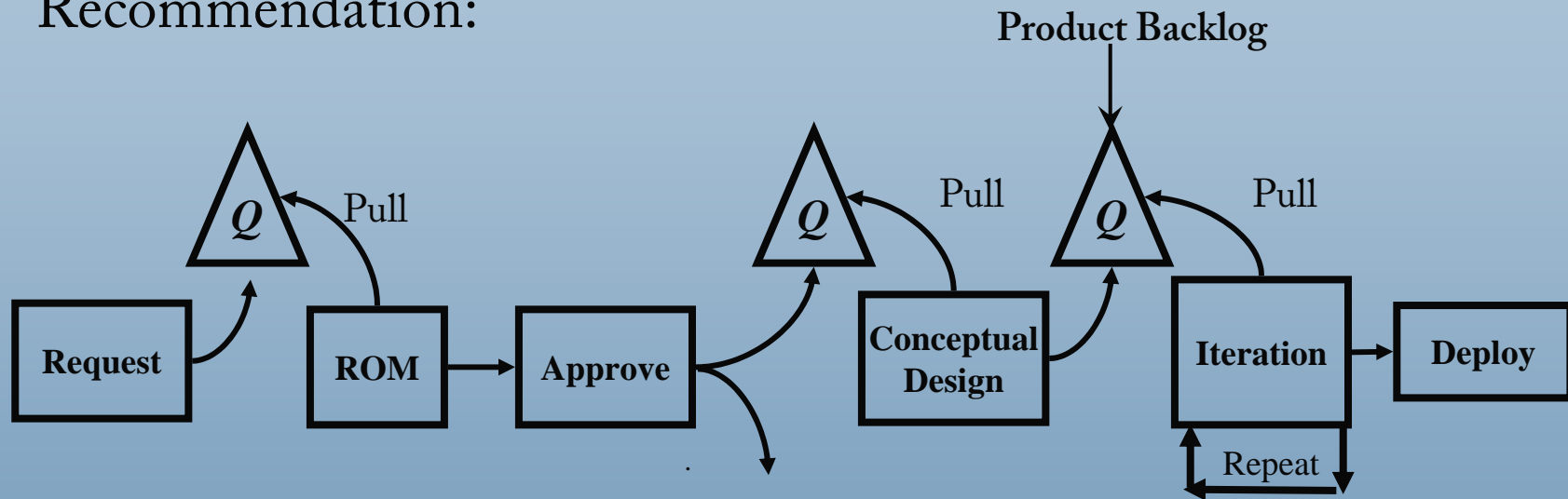
# *Case Study: Limit Work to Capacity*

- Two teams did a value stream map
  - Similar feature requests, each needing 2 hours of coding
  - Organization 1: Cycle time 9 hours
    - Of course resources were available as soon as 'go' decision was made
  - Organization 2: Cycle time 32 days
    - Got stuck 2x for 2 weeks waiting for resources
- IT Support Organization
  - SLA Commitment:
    - Critical problems – 2 hour cycle time
    - Important problems – 2 day cycle time
    - Routine problems – next release (every 2 weeks)
  - Staffing based on SLA, not Utilization



# Case Study: Use Pull Scheduling

- Large Financial Institution
  - Full day every 2 weeks ‘discussing’ priorities
- Recommendation:

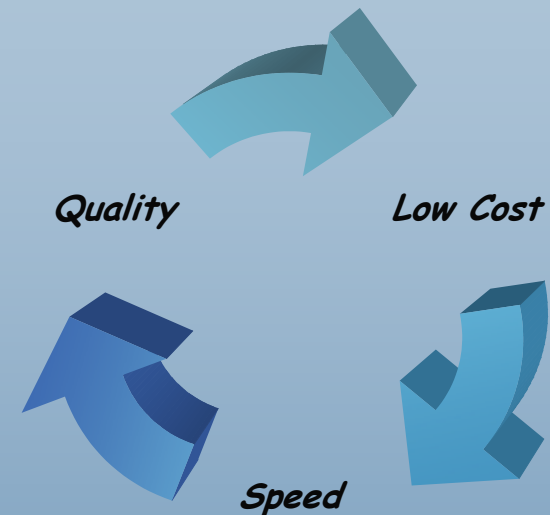


- Manage the Queues
  - Pull from Queues to limit work to capacity
  - Keep Queues small to decrease cycle time



# *Principles of Lean Software Development*

1. Eliminate Waste
2. Amplify Learning
3. Build Integrity In
4. Empower the Team
5. Delay Commitment
6. Deliver Fast
7. Optimize the Whole



l e a n



# Principle 7: 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.

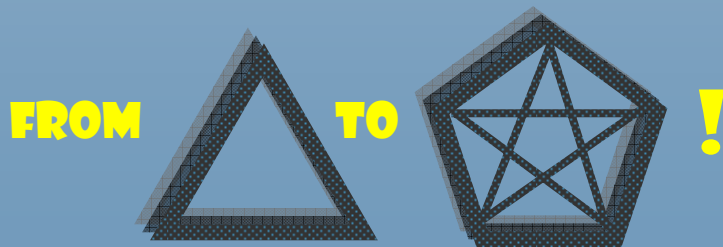
# Measurements

## ■ 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!

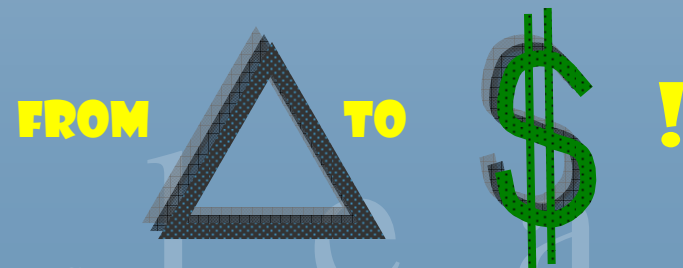


## ■ Aggregation

- 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

- Project success at 
  - On time
  - Globally applicable
  - Measurable bottom line result





# Measure UP

## Span of Control

Hold people accountable for what they can *control*

Measure at the individual level

Fosters competition

## 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!”

## Span of Influence

Hold people accountable for what they can *influence*

Measure at the team level

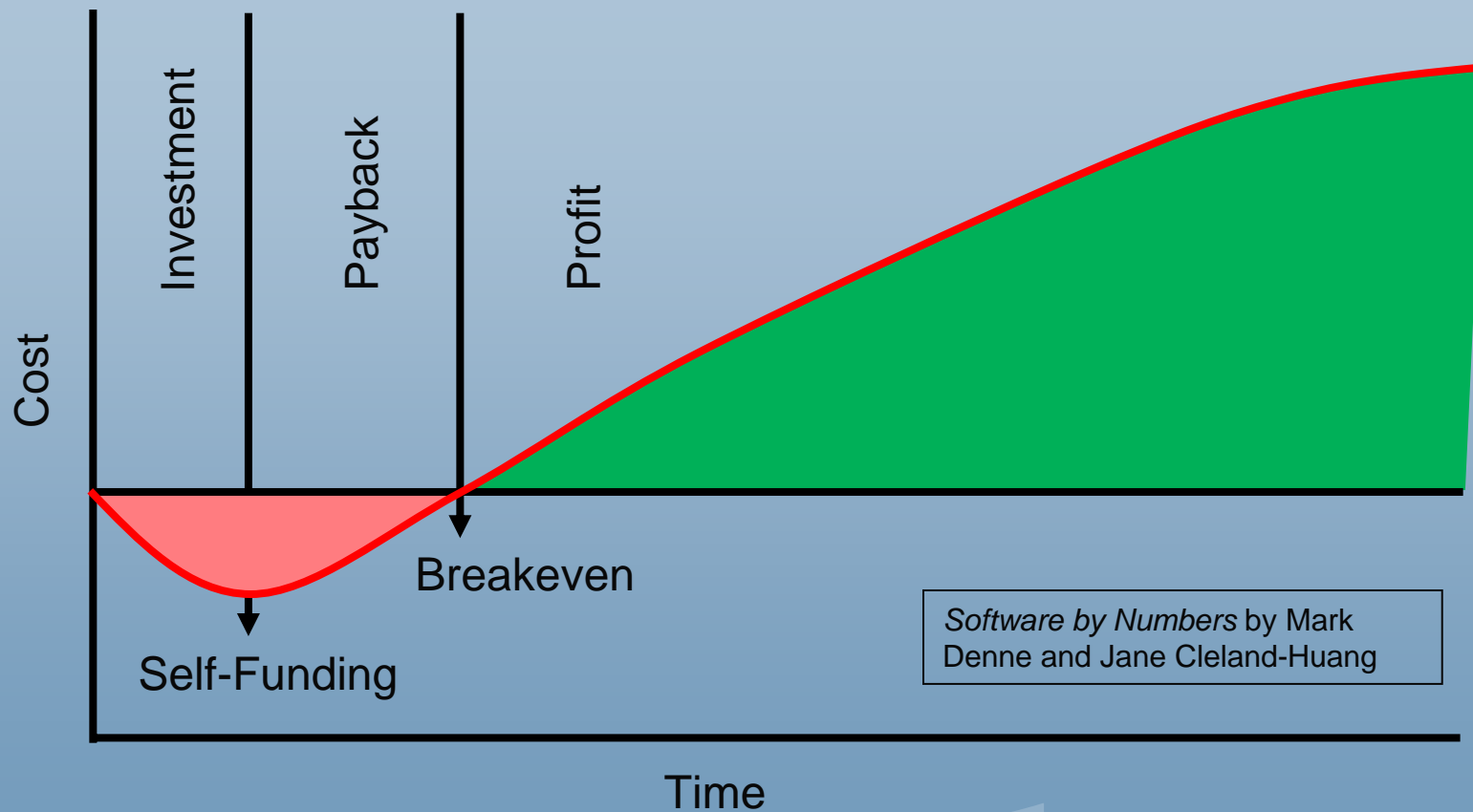
Fosters collaboration

## Product Manager

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

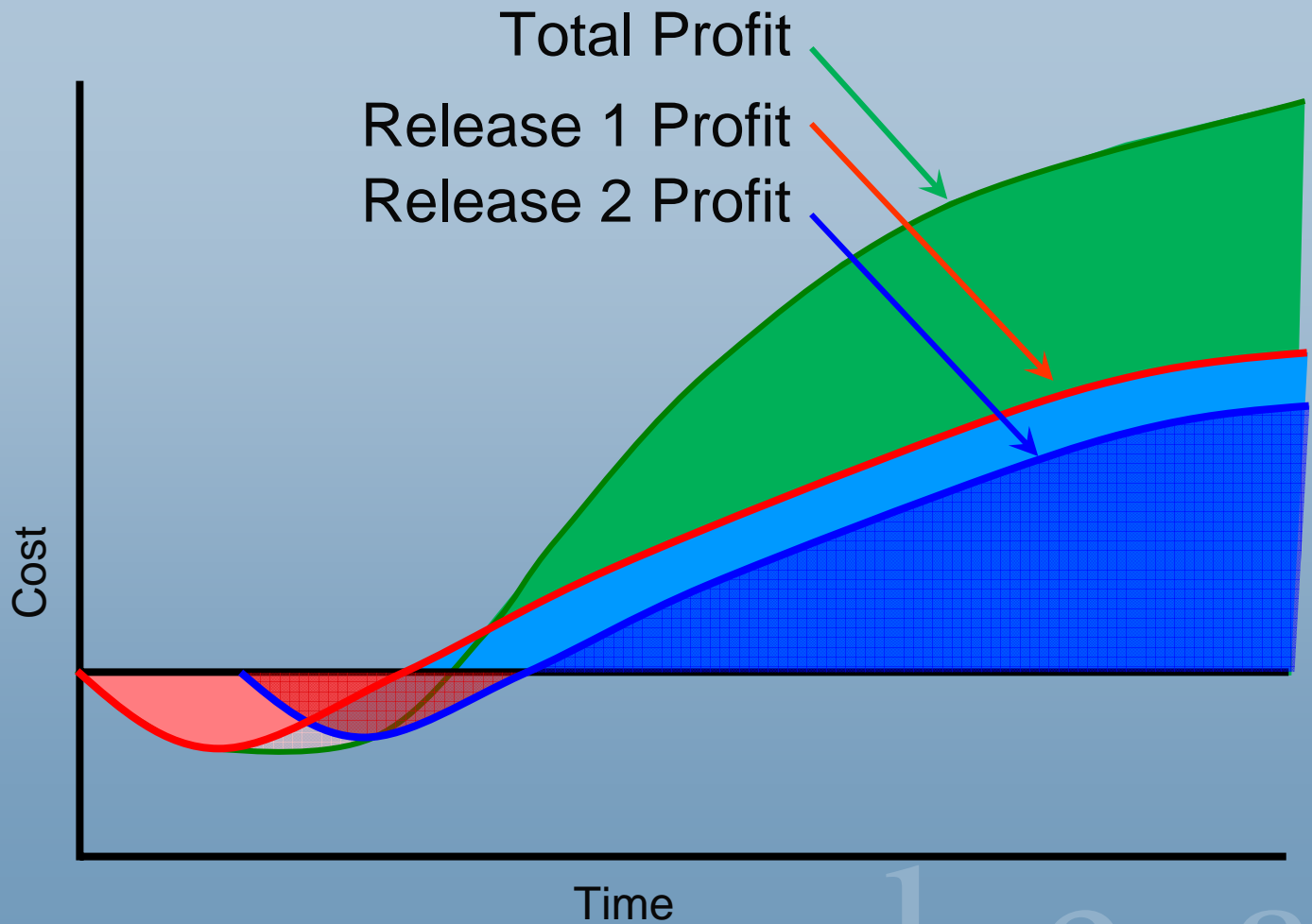


# *A Financial Model*

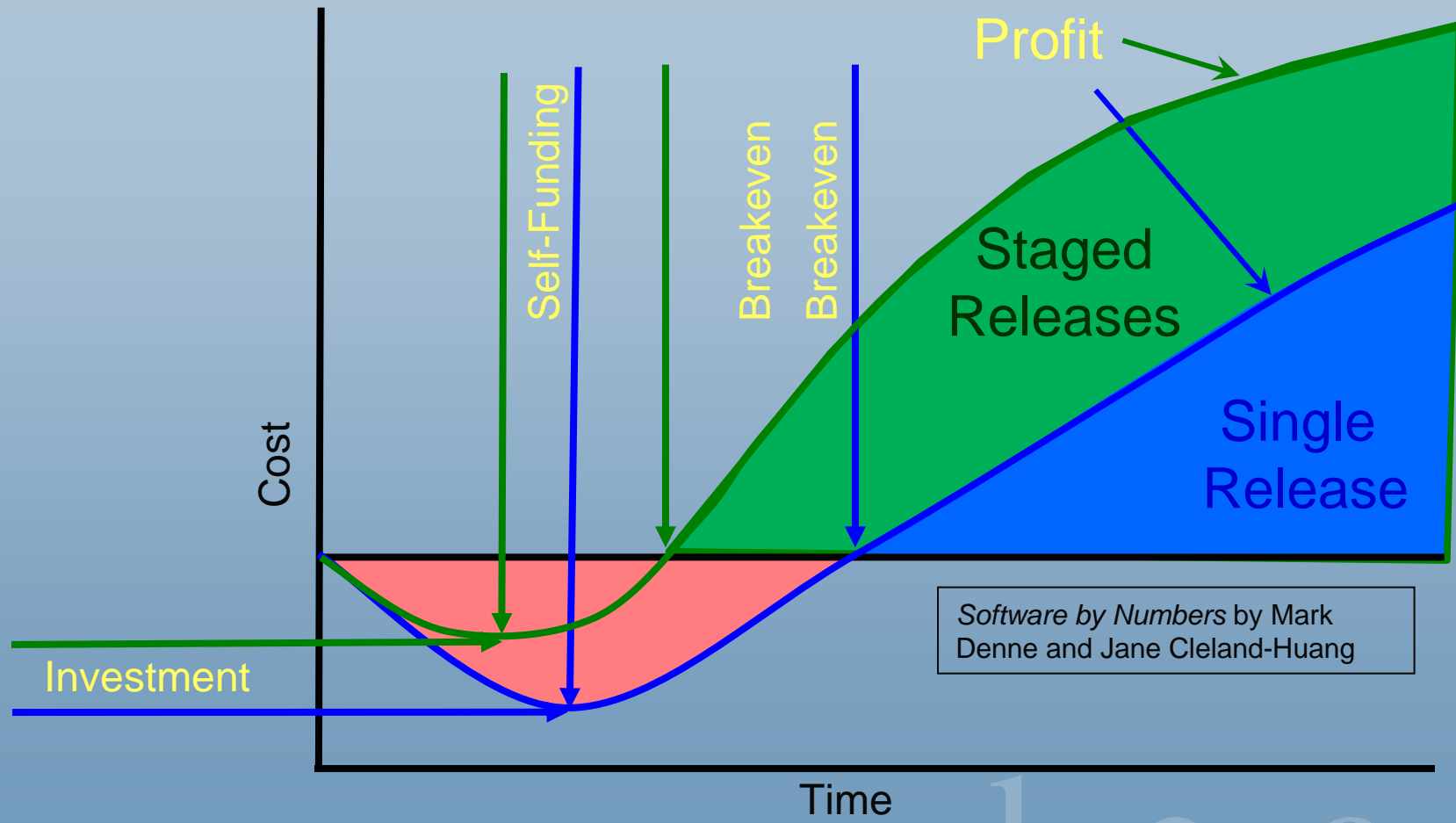




# Staged Releases



# Increased Profit





# *Financial Models*

- Financial models inform investment decisions
  - Profit and Loss (P&L)
  - Return on Investment (ROI)
- They can also inform tradeoff decisions
  - Possibly this is the best use of economic models
  - Every team should be informed enough to make trade-off decisions themselves.
  - Teams make better decisions when they can See the Whole.



# *The Business Case*

- Case Study 1: ROI
  - Consulting firm & used car holding company
    - Re-write of used car management software
    - Reducing car holding time was worth many £'s
    - Very happy customer → More work for consulting firm
- Case Study 2: P&L
  - 3M Product Development
    - Every product development team has an accountant
    - Full team develops P&L as part of its daily work
    - P&L drives approval decisions at management reviews

## Cost of Delay: On Time Commercialization

ON TIME	Assume	Year -2	Year -1	Year 0	Year 1	Year 2	Year 3	Year 4	Year 5
Revenue									
Average Selling Price	- 10% / yr			1,000	900	810	729	656	590
Total Market Units				10,000	20,000	40,000	60,000	40,000	20,000
Market Share				30%	40%	50%	50%	50%	50%
Units Sold				3000	8000	20000	30000	20000	10000
Total Revenue				3,000,000	7,200,000	16,200,000	21,870,000	13,122,000	5,904,900
Expense									
Unit Mfg & Distribution Cost	- 5% / yr			200	190	181	171	163	155
Unit Warranty & Support Cost	- 10% / yr			200	180	162	146	131	118
Total Unit Cost				400	370	343	317	294	273
Manufacturing/Support Cost				1,200,000	2,960,000	6,850,000	9,518,250	5,882,425	2,728,542
Gross Margin \$				1,800,000	4,240,000	9,350,000	12,351,750	7,239,575	3,176,358
Gross Margin %				60%	59%	58%	56%	55%	54%
Engineering		2,000,000	2,000,000	1,500,000	750,000	750,000	500,000	500,000	500,000
Marketing	15% sales			450,000	1,080,000	2,430,000	3,280,500	1,968,300	885,735
G&A	5% sales			150,000	360,000	810,000	1,093,500	656,100	295,245
Total Expense		2,000,000	2,000,000	3,300,000	5,150,000	10,840,000	14,392,250	9,006,825	4,409,522
Profit (Loss)				-\$300,000	2,050,000	5,360,000	7,477,750	4,115,175	1,495,378
% of Revenue				-10%	28%	33%	34%	31%	25%
Cumulative Revenue				3,000,000	10,200,000	26,400,000	48,270,000	61,392,000	67,296,900
Cumulative Expense		2,000,000	4,000,000	7,300,000	12,450,000	23,290,000	37,682,250	46,689,075	51,098,597
Cumulative Profit		-2,000,000	-4,000,000	-4,300,000	-2,250,000	3,110,000	10,587,750	14,702,925	16,198,303
Cumulative Profit % of Revenue				-143%	-22%	12%	22%	24%	24%

## Cost of Delay: Six Month Delay In Commercialization

LATE	ASSUME	Year -2	Year -1	Year 0	Year 1	Year 2	Year 3	Year 4	Year 5
Revenue									
Average Selling Price	- 10% / yr			1,000	900	810	729	656	590
Total Market Units				10,000	20,000	40,000	60,000	40,000	20,000
Market Share				10%	30%	40%	40%	40%	40%
Units Sold				1000	6000	16000	24000	16000	8000
Total Revenue				1,000,000	5,400,000	12,960,000	17,496,000	10,497,600	4,723,920
Expense									
Unit Mfg & Distribution Cost	- 5% / yr			200	190	181	171	163	155
Unit Warranty & Support Cost	- 10% / yr			200	180	162	146	131	118
Total Unit Cost				400	370	343	317	294	273
Manufacturing/Support Cost				400,000	2,220,000	5,480,000	7,614,600	4,705,940	2,182,834
Gross Margin \$				600,000	3,180,000	7,480,000	9,881,400	5,791,660	2,541,087
Gross Margin %				60%	59%	58%	56%	55%	54%
Engineering		2,000,000	2,000,000	1,500,000	750,000	750,000	500,000	500,000	500,000
Marketing	15% sales			150,000	810,000	1,944,000	2,624,400	1,574,640	708,588
G&A	5% sales			50,000	270,000	648,000	874,800	524,880	236,196
Total Expense		2,000,000	2,000,000	2,100,000	4,050,000	8,822,000	11,613,800	7,305,460	3,627,618
Profit (Loss)				-1,100,000	1,350,000	4,138,000	5,882,200	3,192,140	1,096,303
% of Revenue				-110%	25%	32%	34%	30%	23%
Cumulative Revenue				1,000,000	6,400,000	19,360,000	36,856,000	47,353,600	52,077,520
Cumulative Expense		2,000,000	4,000,000	6,100,000	10,150,000	18,972,000	30,585,800	37,891,260	41,518,878
Cumulative Profit		-2,000,000	-4,000,000	-5,100,000	-3,750,000	388,000	6,270,200	9,462,340	10,558,643
Cumulative Profit % of Revenue				-510%	-59%	2%	17%	20%	20%

# ROI Example: Call Center Upgrade

# calls per day	10,000
av. minutes per call	0.53
total time (hours) per day	88
peak call rate	8.2
staff utilization rate	75%
required staffing level	14
average hourly pay	\$7.50
regular hours in month	176
total regular monthly pay	\$18,480
% overtime	19%
\$ overtime	\$1,980
total base pay	\$20,460
supervision	\$2,455
benefits	\$7,161
turnover	2
training	\$840
total salary and benefits	\$30,916
system admin	\$10,000
content maintenance	\$5,000
hardware	\$6,500
facilities	\$2,960
total monthly cost	\$55,376
revenue	\$60,000
profit	\$4,624
margin	8%
customer satisfaction	92%
first call resolution	58%
abandoned calls	3.60%
downtime hours	3.5

**Assumptions:** A 5% increase in customer satisfaction would help to secure \$10,000 in additional revenue, which would generate an additional 2000 calls per month. A 10% lower time per call would allow the center to operate with two less people

	Baseline	5% increase in Customer Satisfaction	10% lower time per call	With New Operating Systeem	50% less training required	One time savings for 50% reduction training difficulty.
Revenue	\$ 60,000	\$ 70,000	\$ 60,000	\$ 60,000	\$ 60,000	
Expenses:						
Call Center Staffing	\$ 30,916	\$ 36,737	\$ 26,615	\$ 30,916	\$ 30,496	
Support Staffing	\$ 15,000	\$ 15,000	\$ 15,000	\$ 10,000	\$ 15,000	
Hardware & Facilities	\$ 9,460	\$ 9,460	\$ 9,460	\$ 8,000	\$ 9,460	
Total	\$ 55,376	\$ 61,197	\$ 51,495	\$ 48,916	\$ 54,956	
Profit	\$ 4,624	\$ 8,803	\$ 8,505	\$ 11,084	\$ 5,044	
Profit Margin	7.7%	12.6%	14.2%	18.5%	8.4%	
Monthly Benefit		\$ 4,179	\$ 3,881	\$ 6,460	\$ 420	

**Analysis:** The customer stands to gain \$6,400 in profit per month in hardware and system administration costs as soon as the system is installed. In addition, if new features lower call time by 10% and increase customer satisfaction by 5%, the new system will add about \$4,000 in profit per month for each of these benefits. Thus the projected customer benefit is about \$14,500 per month, so each week of delay will cost the customer over \$3000. Features which allow faster training will generate a one-time savings of about \$3000, and less than \$500 per month thereafter. Thus it is clear that the team shouldn't spend more than a week adding faster training features.



# Two Process Capability Measurements

## 1. *Average Cycle Time*

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

*One Measurement  
Assesses Speed,  
Quality, & Cost*

## 2. *Process Cycle Efficiency*

Value Added Time

Total Cycle Time

- A measure of the complexity of the product or process
- A Process Diagnostic



# *Two Measurements of Results*

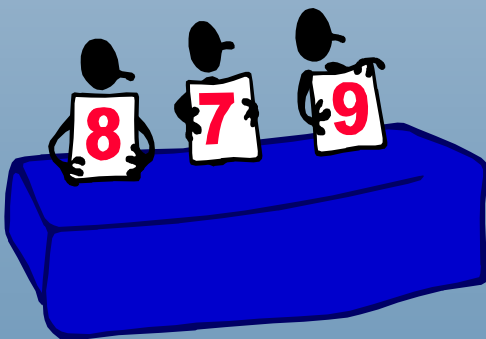
## 1. *The Business Case*

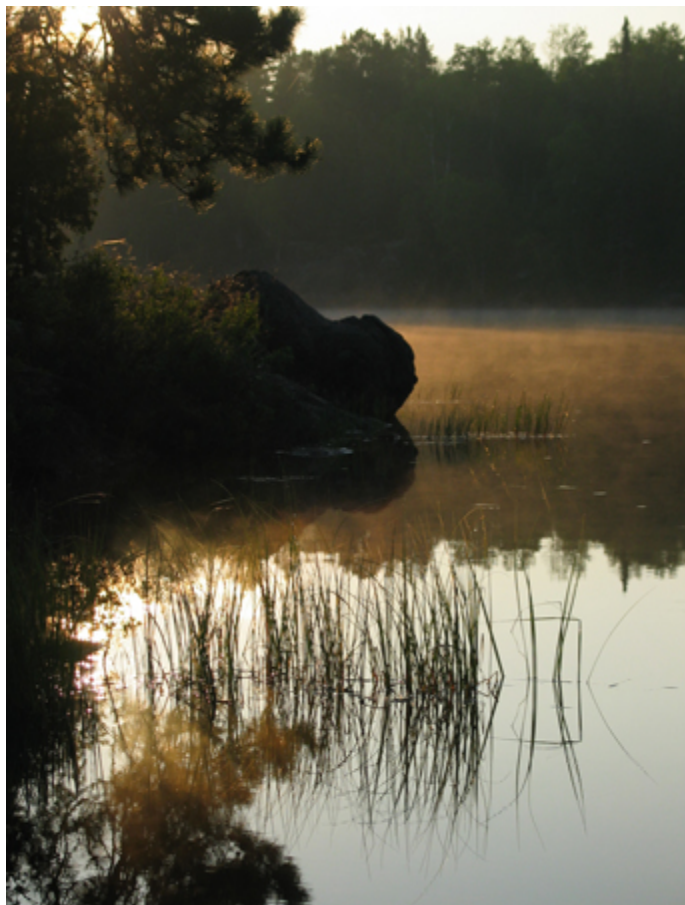
- P&L or
- ROI



## 2. *Customer Satisfaction*

- All customers
  - From downstream processes
  - To ultimate users





l e a n  
software development

Thank You!

*More Information: [www.poppendieck.com](http://www.poppendieck.com)*

mary@poppendieck.com

Mary Poppendieck

[www.poppendieck.com](http://www.poppendieck.com)