LEAN SOFTWARE DEVELOPMENT

AN AGILE TOOLKIT

Mary Poppendieck
mary@poppendieck.com
www.poppendieck.com

THE TOYOTA PRODUCTION SYSTEM

Taiichi Ohno

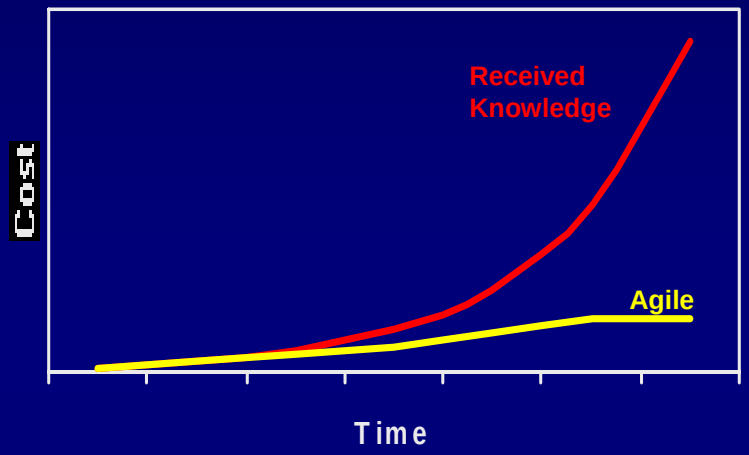
(1912-1990)

- Approach to Production
  - Build only what is needed
  - Stop if something goes wrong
  - Eliminate anything which does not add value
- Philosophy of Work
  - Respect for Workers
  - Full utilization of workers' capabilities
  - Entrust workers with responsibility & authority

Changing the Mental Model

- Received Knowledge:
  - Die Change is Expensive
  - Don't Change Dies
- Taiichi Ohno
  - Economics Requires Many Dies Per Stamping Machine
  - *One Minute Die Change*

Cost of Change



- Received Knowledge:
  - Code Change is Expensive
  - Freeze Design Before Code
- The Agile Imperative
  - Economics Requires Frequent Change In Evolving Domains
  - *Last Responsible Moment*

CONCURRENT ENGINEERING

- 1981 – GM Starts the G-10 Project
  - 1988 – Buick Regal
  - 1989 – Olds Cutlass & Pontiac Grand Prix
  - 2 Years Late
- 1986 – Honda Starts the New Accord Project
  - 1989 – Introduced as 1990 model
  - 1990's – Largest-selling model in North America
- A New Mental Model
  - Instead of
    - Haste Makes Waste
    - Quality Costs More
  - We know
    - Delay Makes Waste
    - Quality Saves

The Machine That Changed The World, Womack, 1990

STAMPING DIES

Toyota

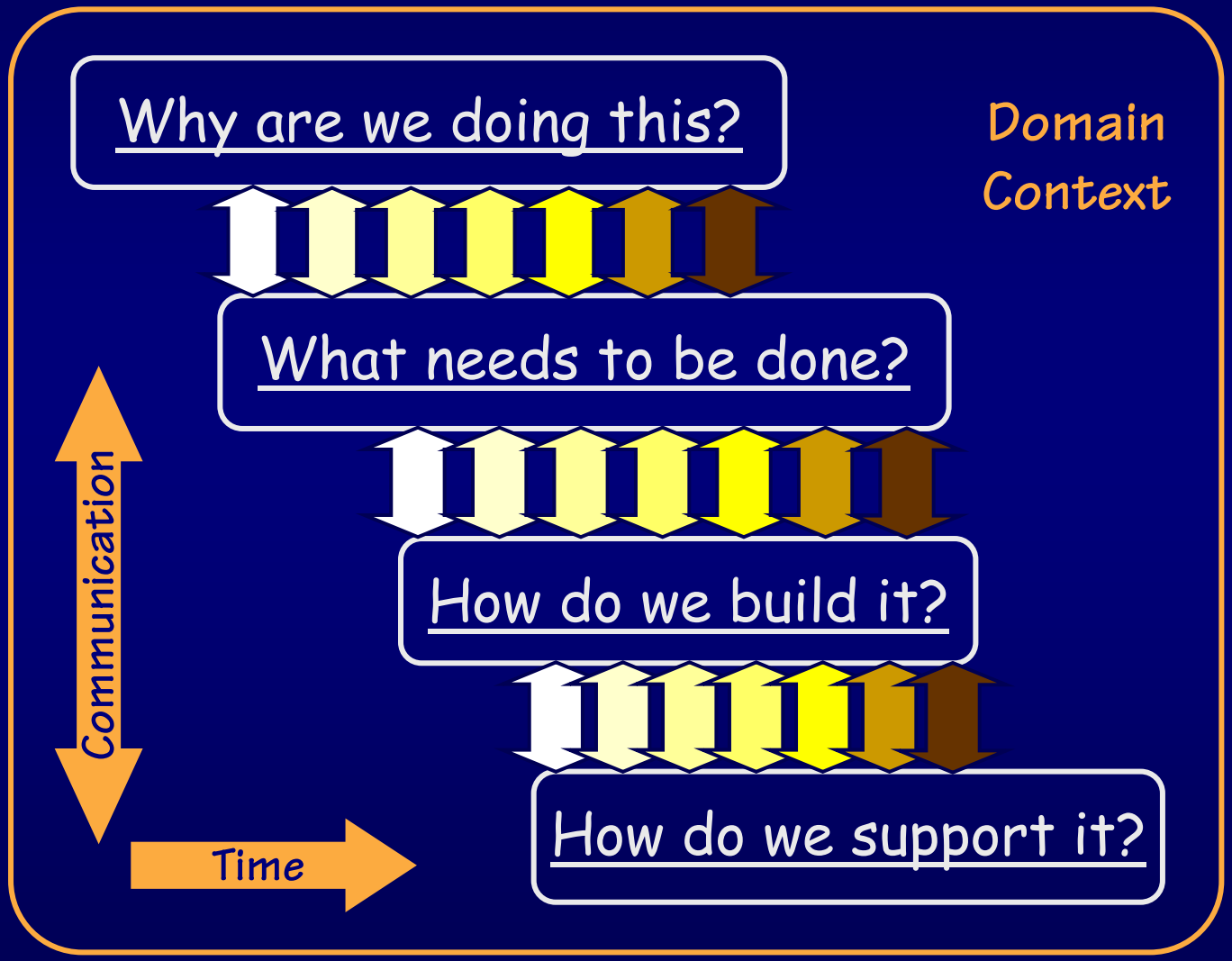
- Mistakes very expensive
- Never-ending changes
- Early Design – Early Cut
- Focus: Reduce Time
- Designer goes to supplier shop, discusses changes, implements immediately, submits for later approval
- Target cost (including changes)
- 10-20% cost for changes
- Half the time, half the cost

Typical US

- Mistakes very expensive
- Never-ending changes
- Wait to Design – Wait to Cut
- Focus: Reduce Waste
- Designer must get multiple signatures for changes, sends to purchasing which sends change document to vendor
- Fixed cost (changes are extra, profit source for supplier)
- 30-50% cost for changes
- Twice the time, twice the cost

Clark & Fujimoto, *Product Development Performance*, 1991

CONCURRENT SOFTWARE DEVELOPMENT



PRINCIPLES OF LEAN THINKING

1. ELIMINATE WASTE
2. INCREASE FEEDBACK
3. DELAY COMMITMENT
4. DELIVER FAST
5. BUILD INTEGRITY IN
6. EMPOWER THE TEAM
7. SEE THE WHOLE

PRINCIPLE 1: ELIMINATE WASTE

- Waste
  - Anything that does not create value for the customer
  - The customer would be equally happy with the software without it
- Prime Directive of Lean Thinking
  - Create ***Value*** for the customer
  - Improve the ***Value Stream***

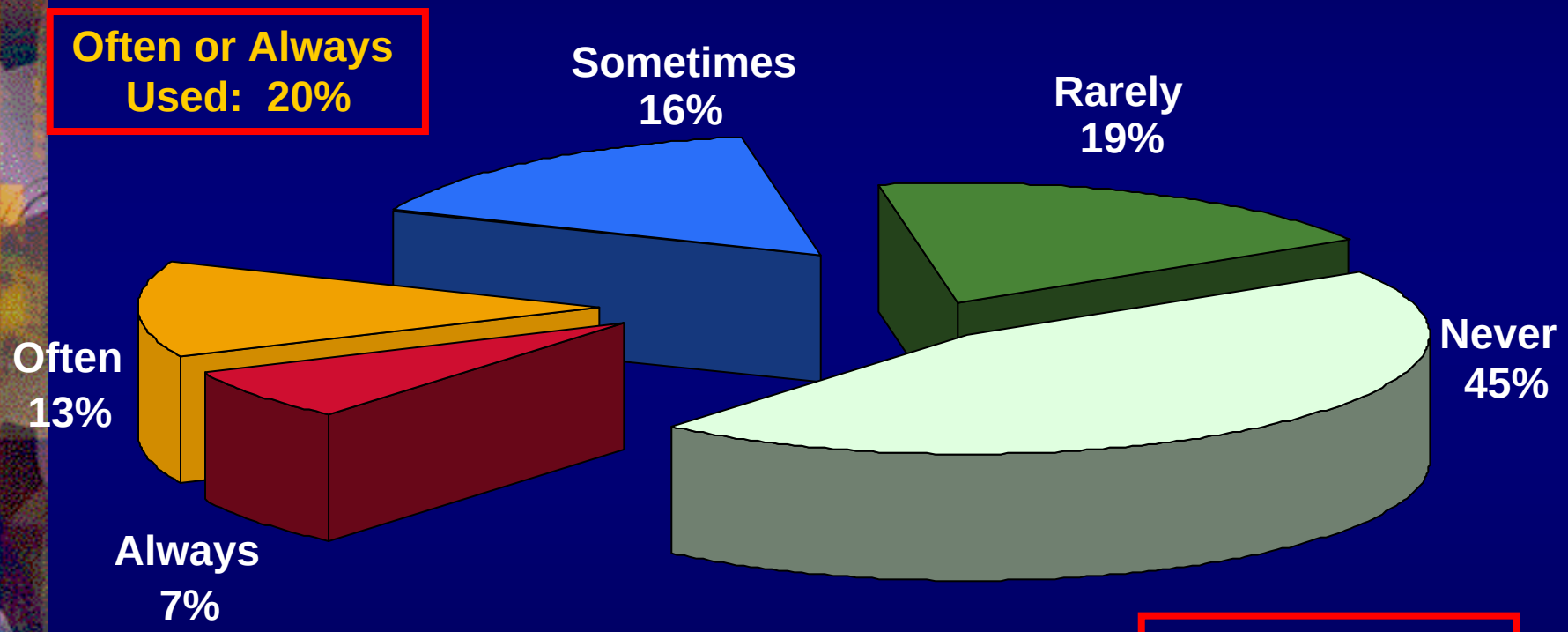
SEEING WASTE

Seven Wastes of Manufacturing*	Seven Wastes of Software Development
Inventory	Partially Done Work
Extra Processing	Paperwork
Overproduction	Extra Features
Transportation	Building the Wrong Thing
Waiting	Waiting for Information
Motion	Task Switching
Defects	Defects

* Shigeo Shingo, an engineer at Toyota and a noted authority on just-in-time techniques.

THE BIGGEST SOURCE OF WASTE

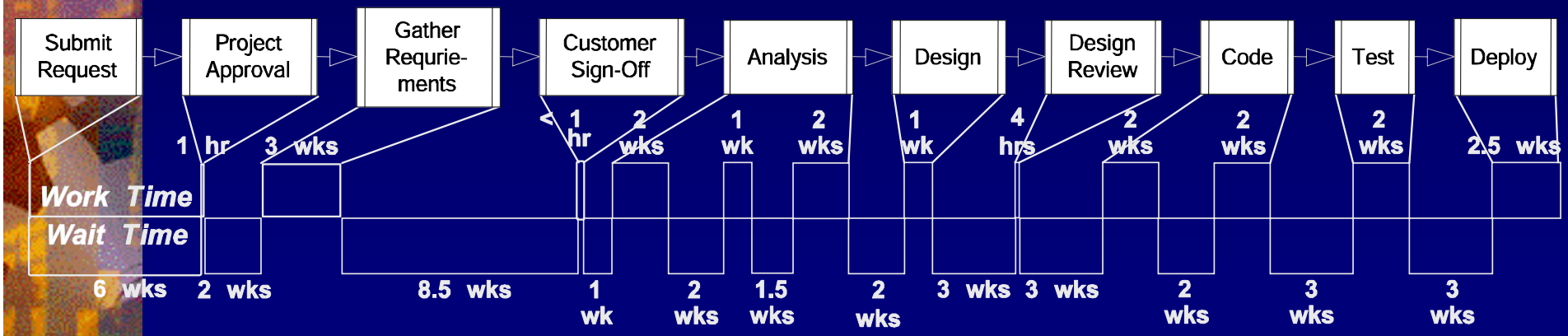
Features and Functions Used in a Typical System



Rarely or Never Used: 64%

Standish Group Study Reported at XP2002 by Jim Johnson, Chairman

TRADITIONAL VALUE STREAM



- Total Time: ~55 weeks
  - Work Time ~17.6 weeks
    - 1/3rd of the time
  - Wait Time ~37 Weeks
    - 2/3rds of the time

- Bottlenecks:
  - Approvals
  - Sign Offs
  - Design Review
  - Testing
  - Deployment

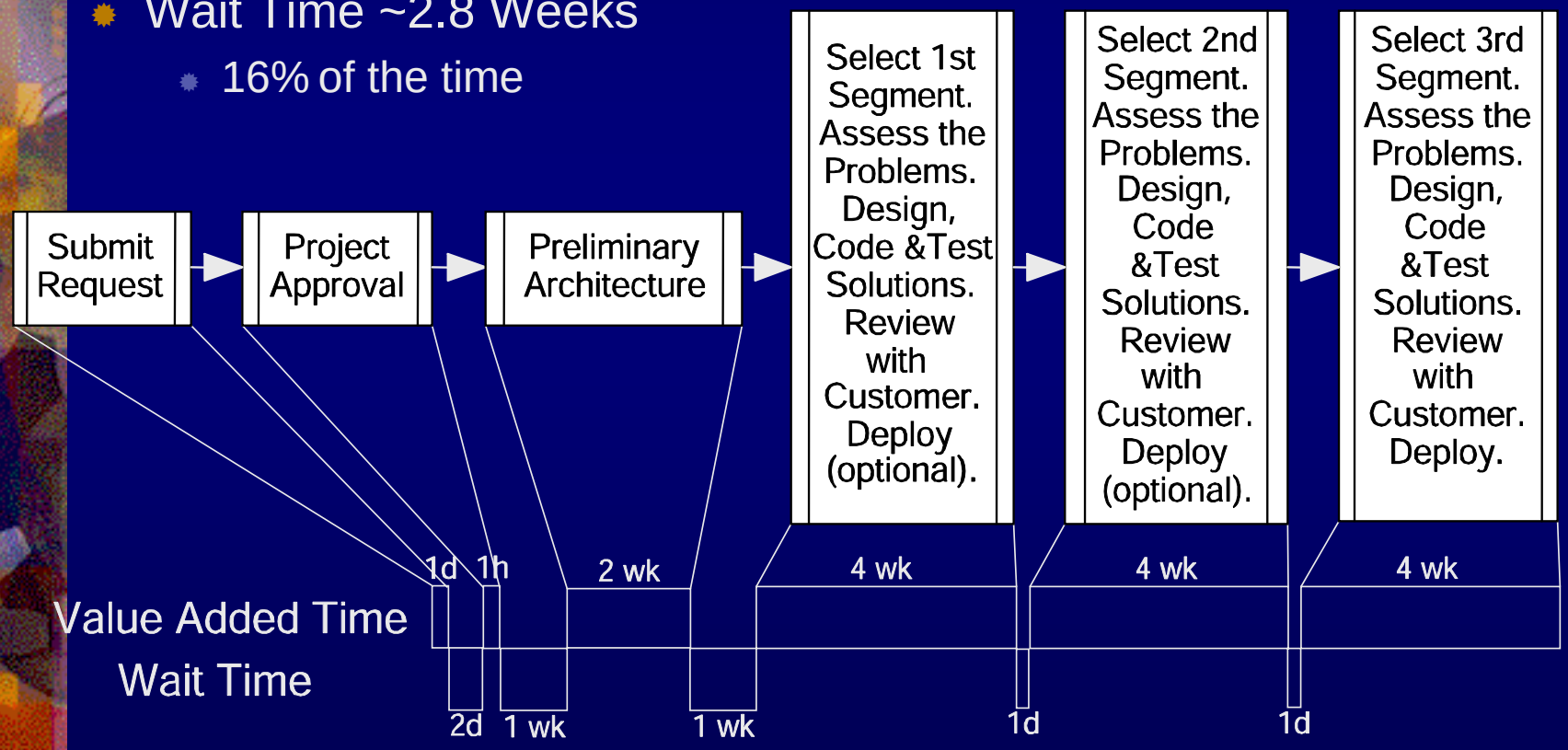
LEAN VALUE STREAM

Total Time: ~17 weeks

- Work Time ~14.2 weeks
  - 84% of the time
- Wait Time ~2.8 Weeks
  - 16% of the time

Levers:

- Concurrent Development
- Effective Gating Process



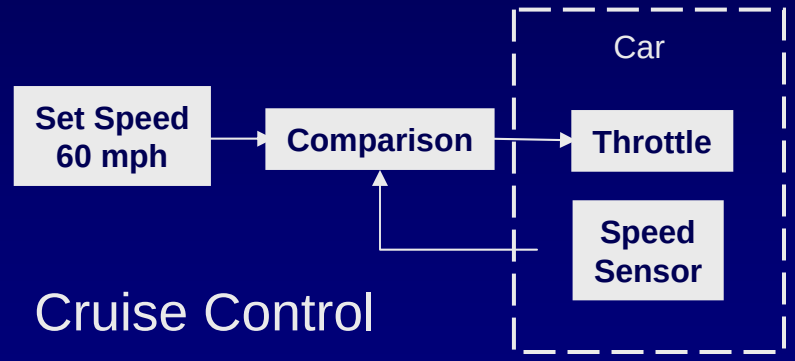
EXERCISE

- Choose a system you know about
  - Estimate % of the features are always or often used
- Choose a development cycle you are familiar with
  - Estimate the average it takes to convert customer requests into deployed software

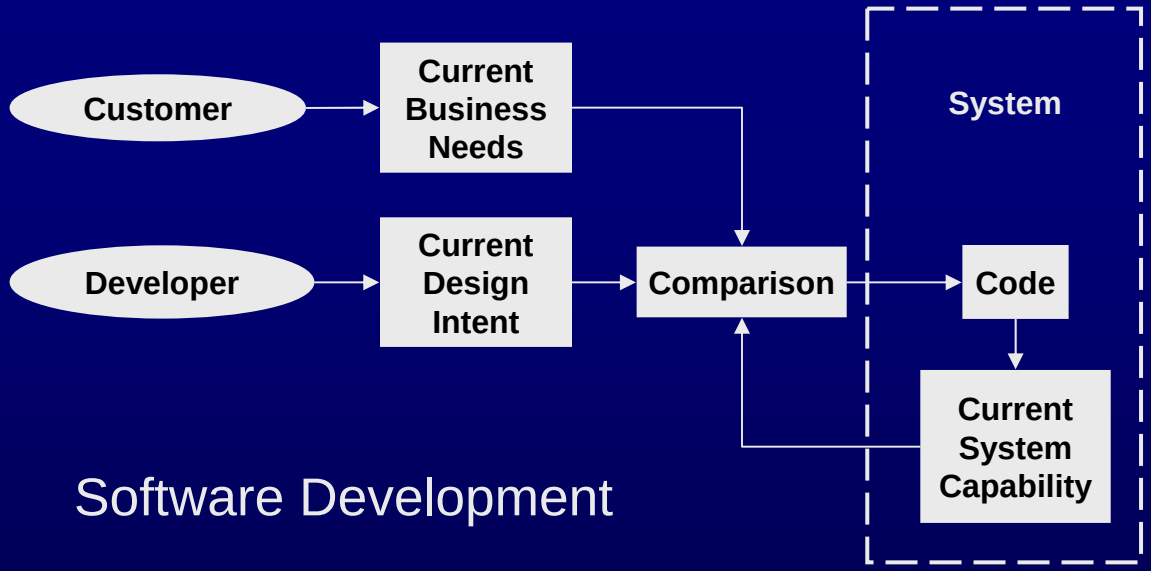
PRINCIPLES OF LEAN THINKING

1. ELIMINATE WASTE
2. INCREASE FEEDBACK
3. DELAY COMMITMENT
4. DELIVER FAST
5. BUILD INTEGRITY IN
6. EMPOWER THE TEAM
7. SEE THE WHOLE

PRINCIPLE 2: INCREASE FEEDBACK



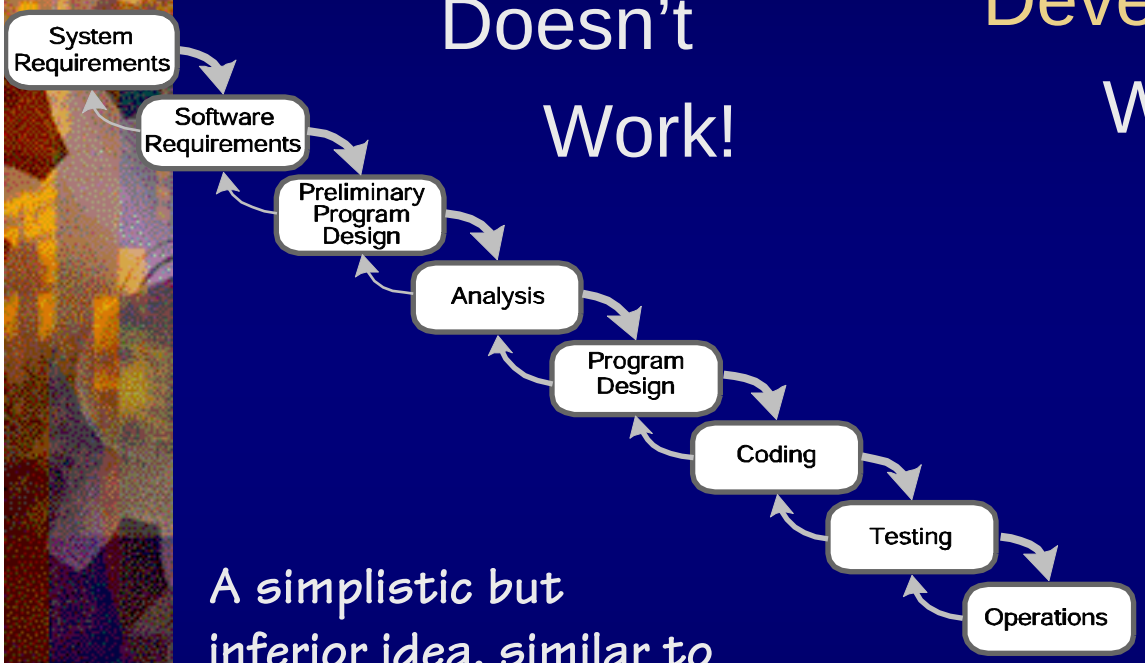
Cruise Control



Software Development

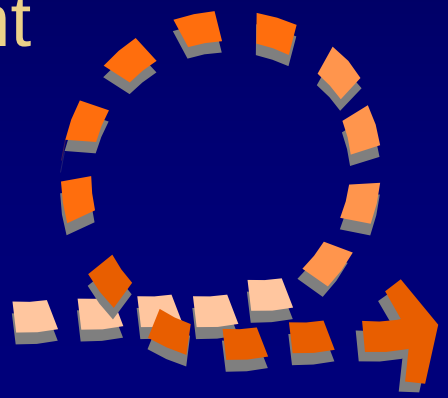
THE FUNDAMENTAL PRACTICE

- Waterfall Doesn't Work!



A simplistic but inferior idea, similar to medicine's "four humors".*

- Iterative Incremental Development Works!

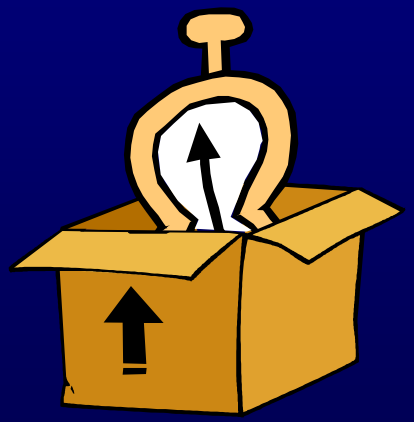


Recommended by software engineering thoughtleaders, associated with numerous successful large projects & recommended by standards boards.*

* Craig Larman, "A History of Iterative and Incremental Development", IEEE Computer, June 2003

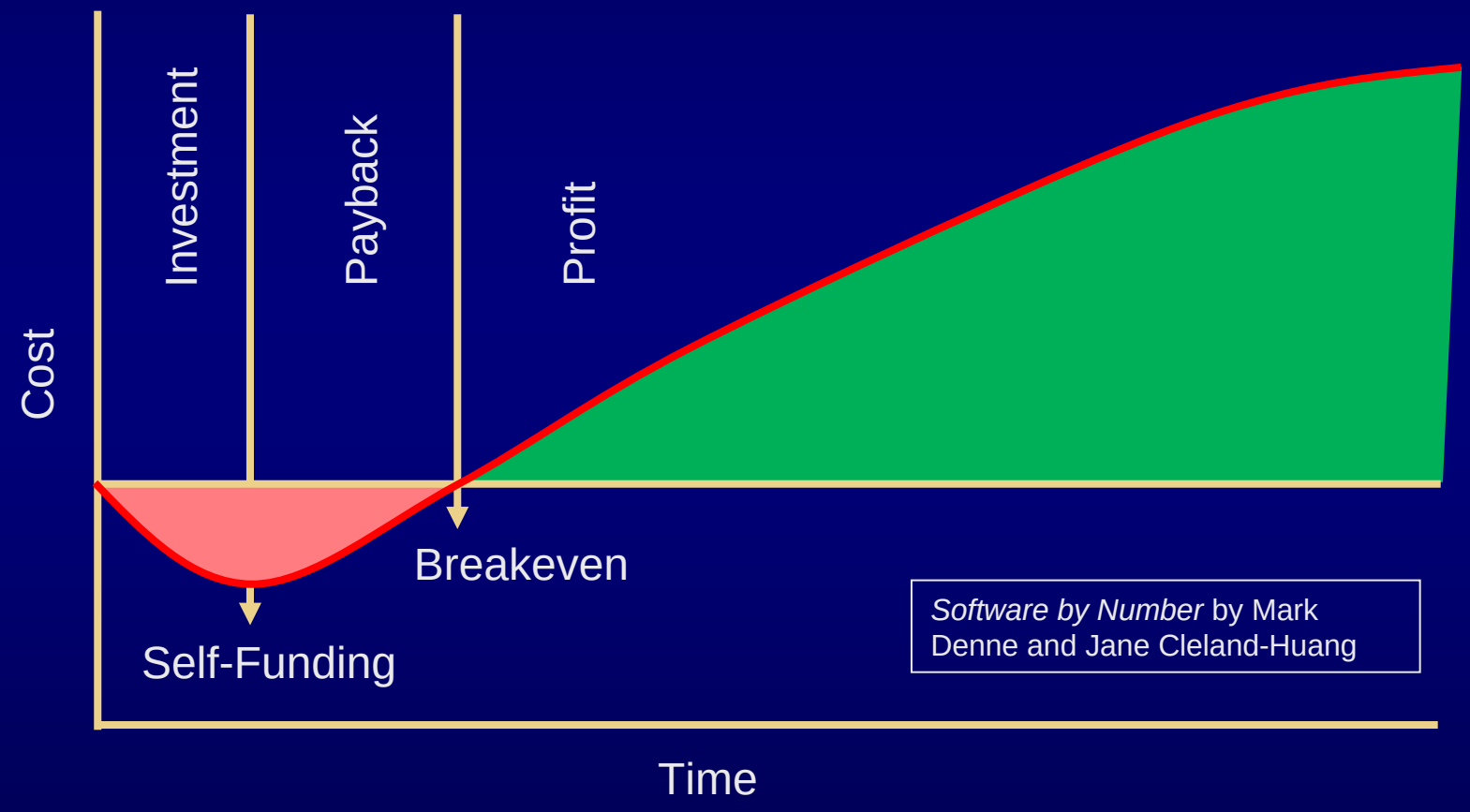
SIMPLE RULES OF ITERATION

- Business Sets Priority
  - Minimum Marketable Features (MMF)
- Development Team Determines Effort
  - Team chooses and commits to iteration goal
- Use a Short Time Box
  - Drop features to meet the deadline
- Deliver on Commitment
  - Develop Confidence
- Create Business Value
  - Potentially Deployable Code



MINIMUM MARKETABLE FEATURES (MMF)

Deploy Early & Often – Move Profit Forward



Software by Number by Mark Denne and Jane Cleland-Huang

Rx FOR TROUBLED PROJECTS

- Increase Feedback !
  - Customer Feedback to Team
  - Team Feedback to Management
  - Product Feedback to Team
  - Upstream-Downstream Feedback
- Don’t Decrease Feedback
  - Adding Yet More Process Rarely Helps

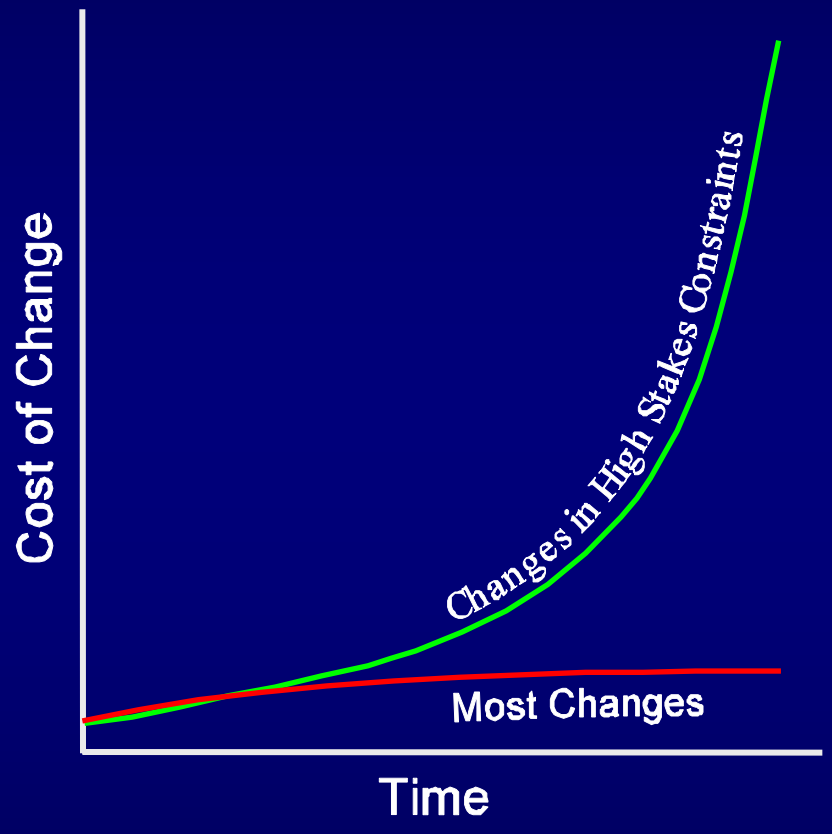
PRINCIPLE 3: DELAY COMMITMENT

- The technology changes rapidly
- The business situation evolves
- Software will change!
  - Software products
    - Improve with age
    - Architecture is expected to change over time
  - Custom software
    - Becomes brittle with age
    - Architecture is not expected to change
    - But 60-70% of software development occurs after initial release to production

COST ESCALATION

Two Kinds of Change

- High Stakes Constraints
  - Examples:
    - Language
    - Layering
    - Usability
    - Security
    - Scalability
  - Rule:
    - Only a Few
    - At a High Level
- Most Changes
  - Keep the Cost Low!



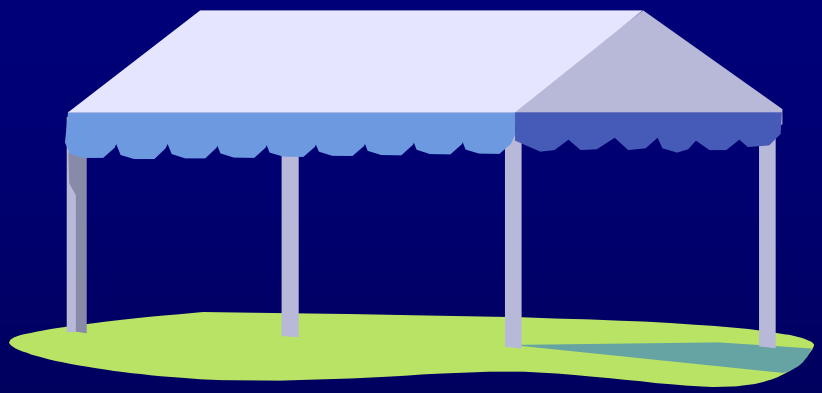
PREDICTABLE OUTCOMES

To Get Predictable Outcomes, Don't Predict!

Make Decisions based of Facts, not Forecasts.

A Minnesota Wedding

- August 10th
  - 50% Chance of Rain
  - 65-95 ºF
- Invitations must be sent in June



DELAY COMMITMENT

- Share partially complete design information.
- Develop a sense of how to absorb changes.
- Avoid extra features.
- Develop a quick response capability.
- Develop a sense of when to make decisions.

Software Delaying Tactics

Encapsulate Variation

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

Avoid Repetition

- Don't Repeat Yourself
- Once & Only Once
- Never copy & paste
- Never!

Separate Concerns

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

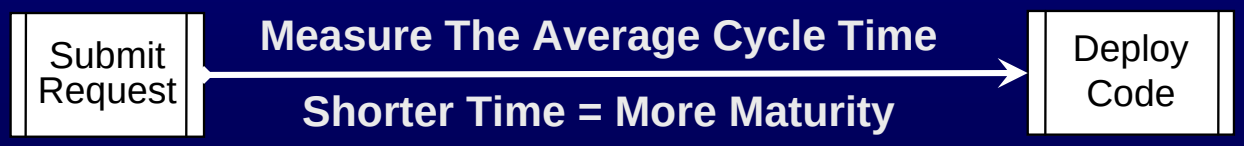
Defer Implementation

- You Aren't Goanna Need It
- It costs a bundle to maintain and a bundle to change

PRINCIPLE 4: DELIVER FAST

- The most disciplined organizations are those that respond to customer requests
  - Rapidly
  - Reliably
  - Repeatedly

- Software Development Maturity
  - The speed at which you <u>reliably</u> and <u>repeatedly</u> convert customer requests to deployed software



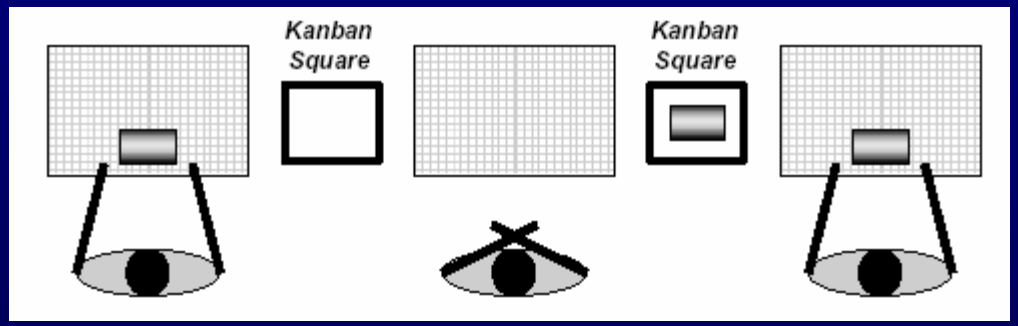
PRINCIPLES OF SPEED

- Pull from customer demand
  - Pull with an order
  - Don't push with a schedule

- Make work self-directing
  - Visual Workplace
- Rely on local signaling and commitment
  - Kanban
  - Scrum Meetings

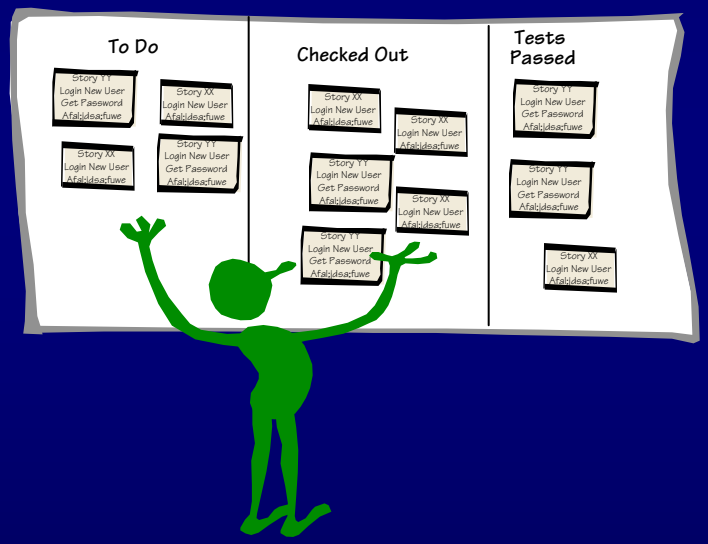
- Use Small Batches
  - Limit the amount of work in the pipeline

MANUFACTURING: KANBAN

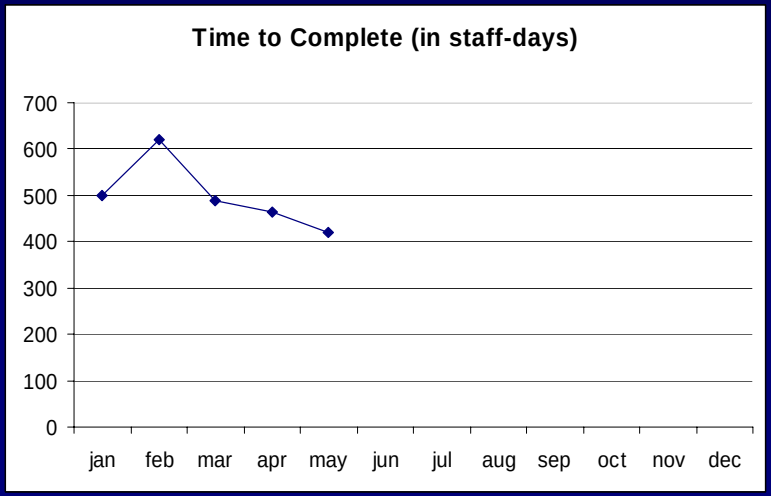


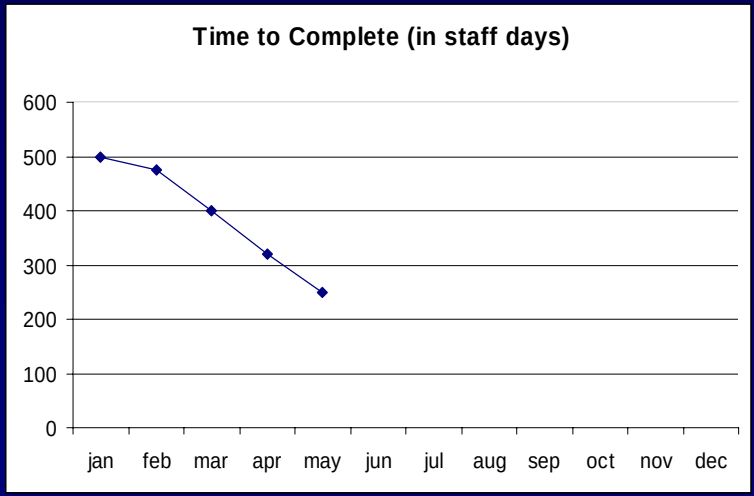
SOFTWARE KANBAN

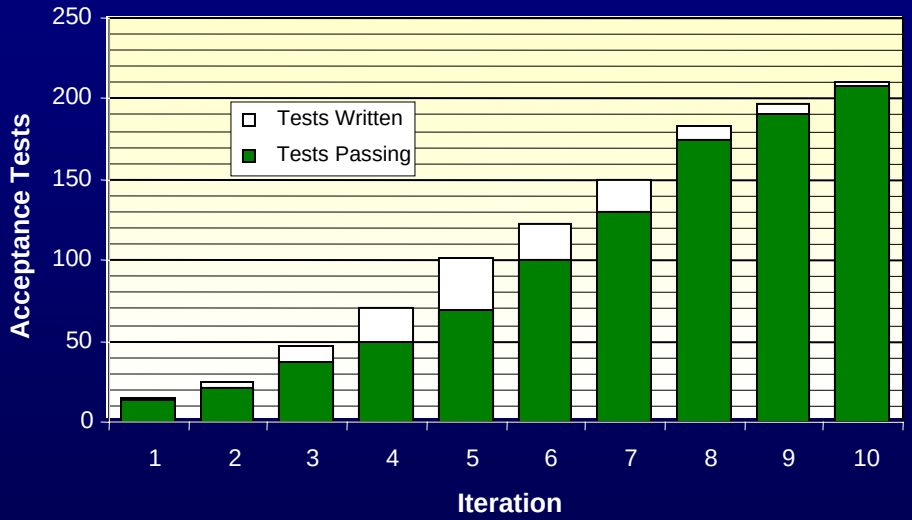
- Story Cards or Iteration Feature List
  - How do developers know what to do?
- Information Radiators
  - White Boards
  - Charts on the Wall
- Daily Meetings
  - Status
  - Commitment
  - Need



MAKE CONVERGENCE VISIBLE







QUEUES

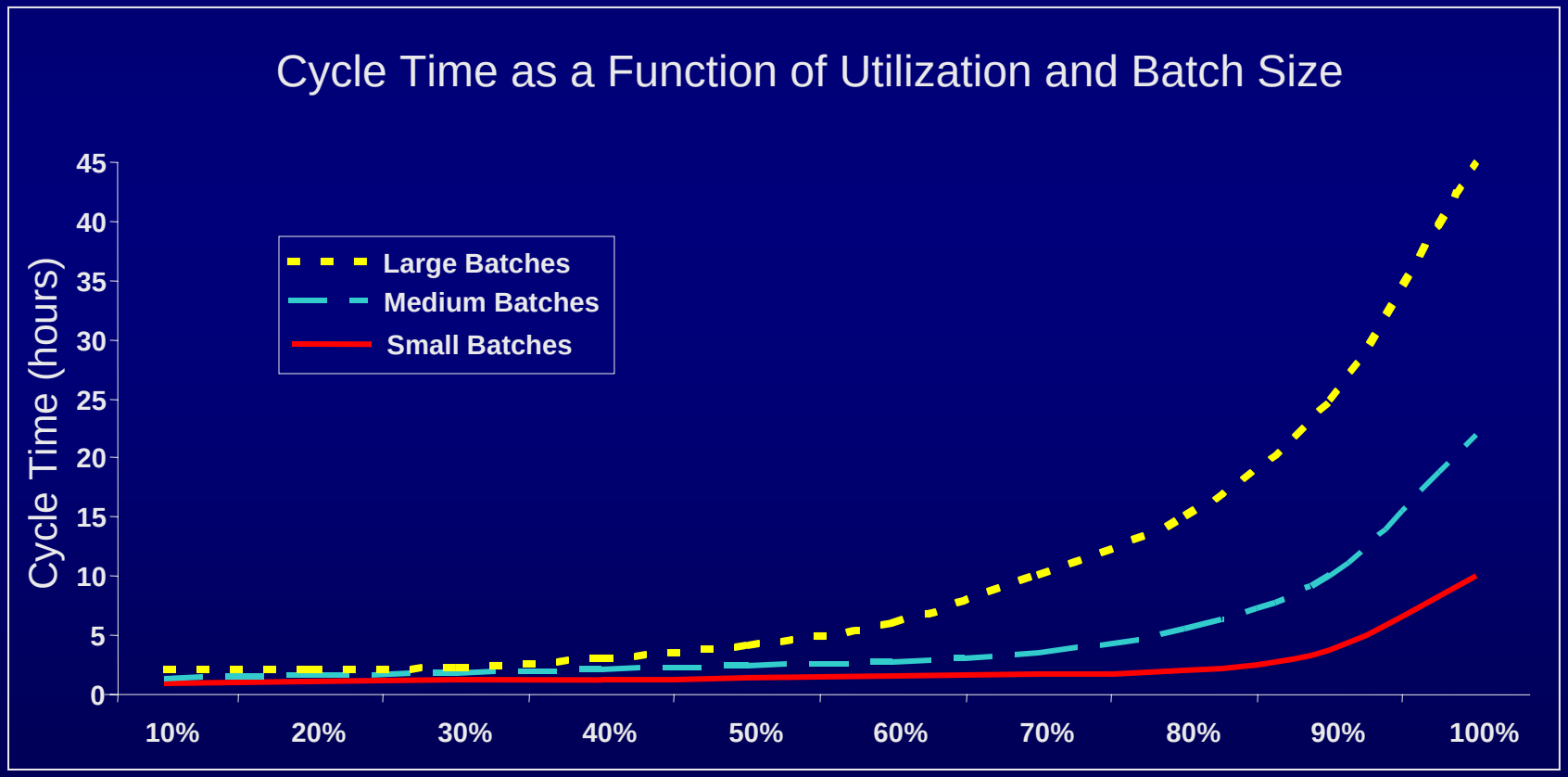
- Cycle Time
  - Average End-to-End Process Time
    - From Entering The Terminal
    - To Arriving at the Gate
- Time Spent in a Queue is Wasted Time
- The Goal: <u>Reduce Cycle Time</u>

Reducing Cycle Time

1. Steady Rate of Arrival
   Develop In Short Iterations
2. Steady Rate of Service
   Test Features Immediately
3. Small Work Packages
   Integrate Features Individually
4. Reduce Utilization
   You Don't Load Servers to 90%
5. Eliminate Bottlenecks
   Everyone Pitches In Wherever They Are Needed

QUEUEING THEORY LESSONS

1. Small Batches Move Faster
2. Slack Resources Decrease Cycle Time



XP'S 12 PRACTICES

1. The Planning Aim
2. Small Releases
3. Metaphor
4. Simple Design
5. Testing
6. Refactoring
7. Pair Programming
8. Collective Ownership
9. Continuous Integration
10. Sustainable Pace
11. On-Site Customer
12. Coding Standards

CASE STUDY: XP

- Discussion
  - How do XP practices
    - Increase Feedback
    - Delay Commitment
    - Deliver Fast
- Examples
  - Gearworks
  - Your experience

From Gearworks developers

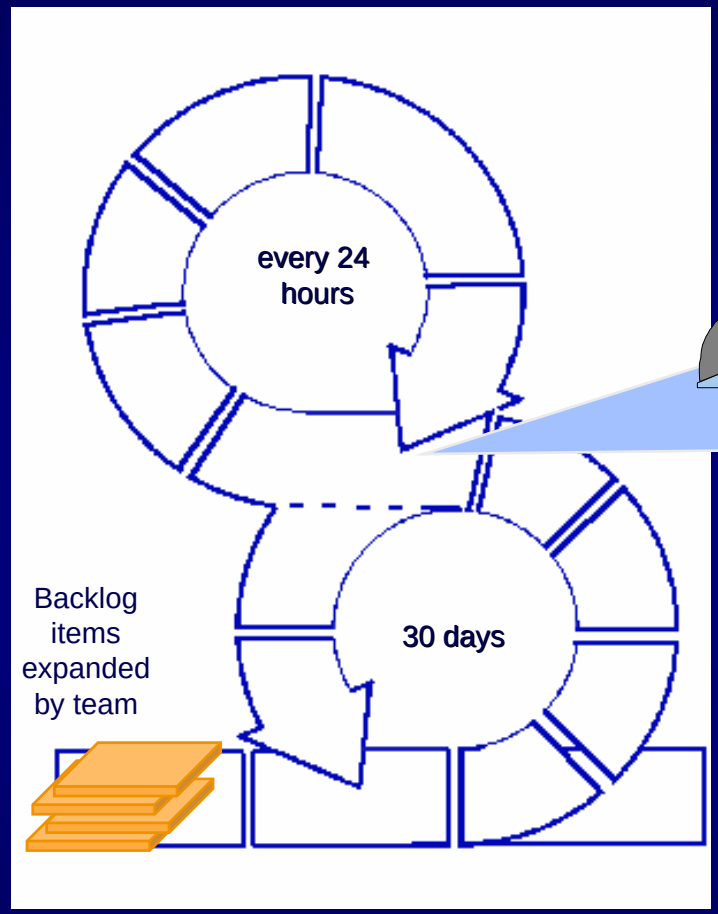
Don't

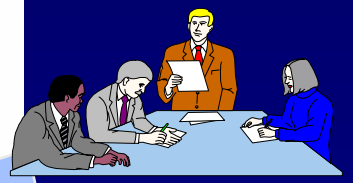
- Put off refactoring
- Open up visibility just for testing
- Write time/date brittle tests
- Test generated code

Do

- Write tests before code
- Eliminate duplication
- Refactor mercilessly
- Leave code better than you found it
- Only write tests for contracts
- Write tests for bugs (before fixing them)
- Don't be afraid to throw away code
- Use local databases

SCRUM





Scrum: 15 minute daily meeting.
Teams member respond to basics:
1) What did you do since last Scrum Meeting?
2) Do you have any obstacles?
3) What will you do before next meeting?

Sprint Backlog:
Feature(s) assigned to sprint

Product Backlog:
Prioritized product features desired by the customer

New functionality is demonstrated at end of sprint

CASE STUDY: SCRUM

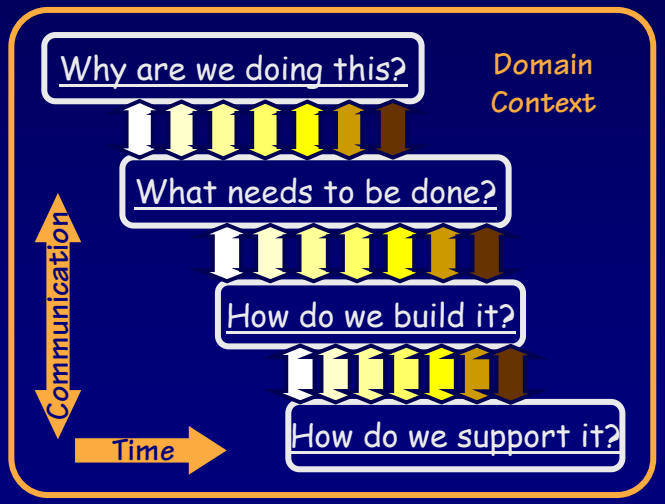
- How does Scrum
  - Increase Feedback
  - Delay Commitment
  - Deliver Fast
- Examples
  - Minnesota Secretary of State UCC System
  - Your examples

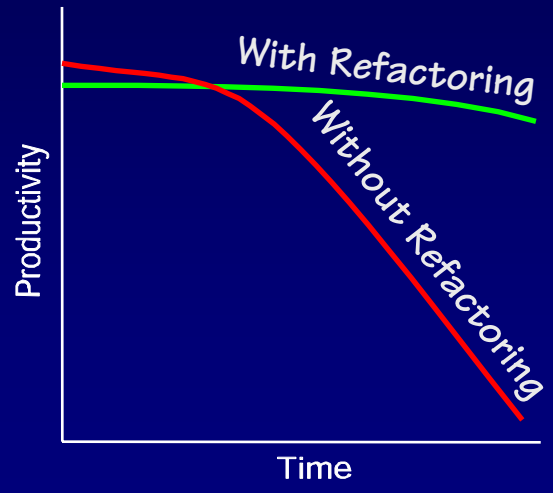
BREAK

PRINCIPLES OF LEAN THINKING

1. ELIMINATE WASTE
2. INCREASE FEEDBACK
3. DELAY COMMITMENT
4. DELIVER FAST
5. BUILD INTEGRITY IN
6. EMPOWER THE TEAM
7. SEE THE WHOLE

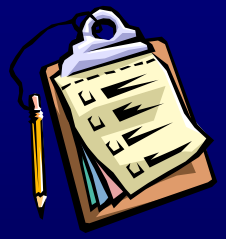
PRINCIPLE 6: BUILD INTEGRITY IN

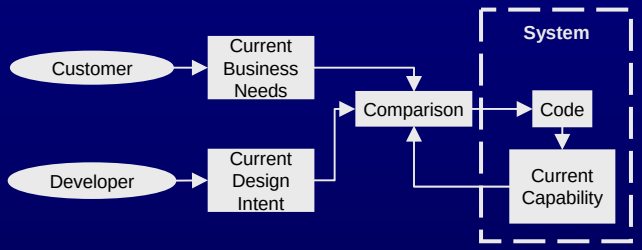


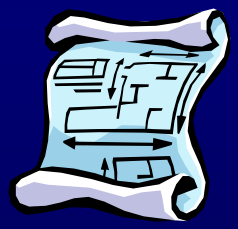


Integrated Product Teams

Refactoring





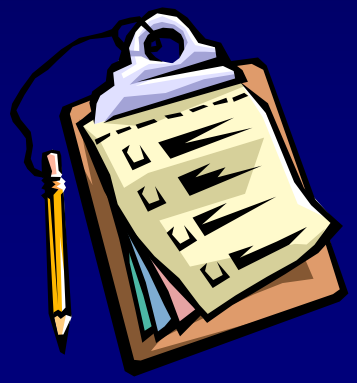


Requirements **Feedback** **Refactoring** **Maintenance**

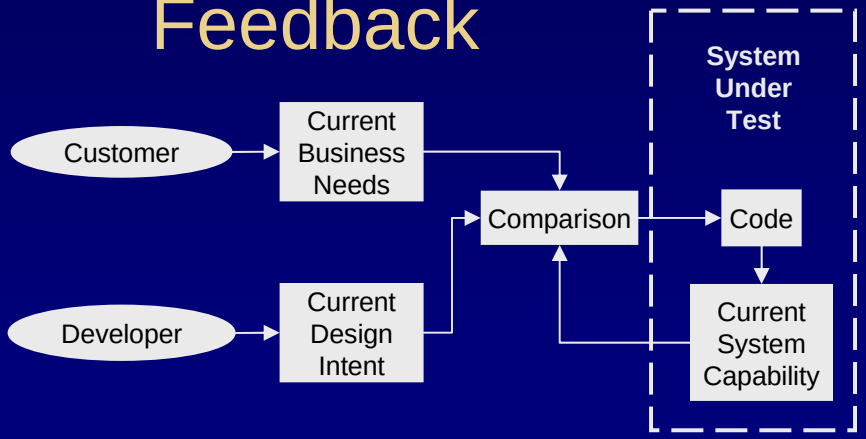
Test-Driven Development

TEST-DRIVEN DEVELOPMENT

Requirements

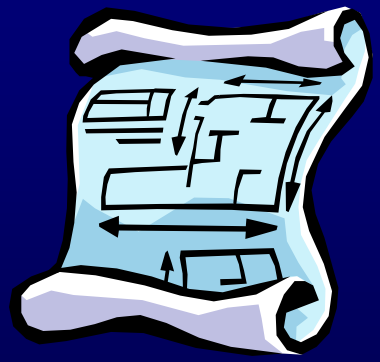


Feedback



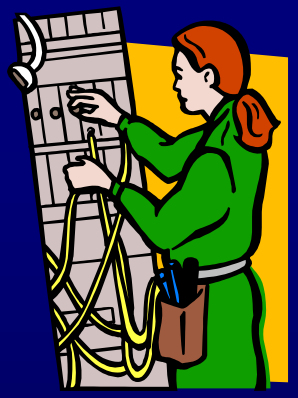
Refactoring

Maintenance



AUTOMATED TESTS: THE KEY DISCIPLINE OF AGILE

- Don't attempt iterative development without automated tests
- Developers will to write tests anyway
  - Why not write the test first?
  - Why not capture the tests and automate them?
  - Why not make tests a part of the code base?
- Legacy code is code without a test harness



Agile Testing

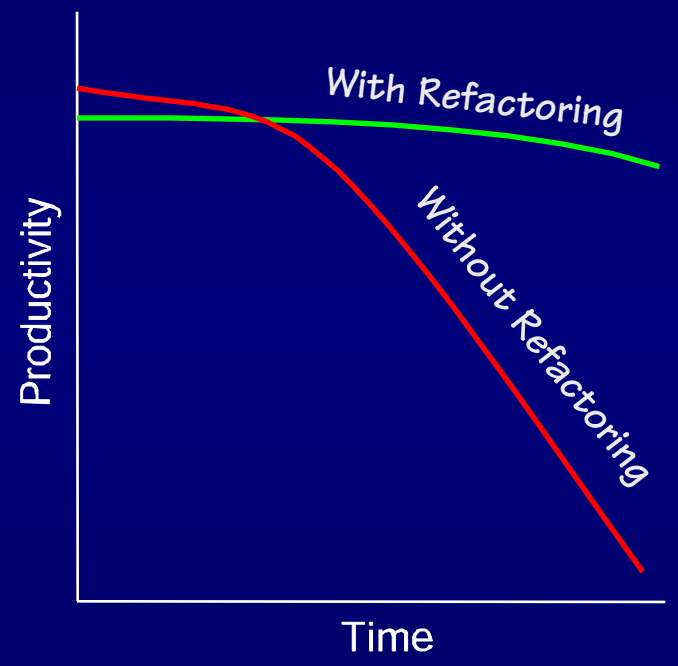
- Types of tests
  - Developer Tests
    - Do the underlying mechanisms work?
  - Customer Tests
    - Is the business purpose achieved?
  - -ability Tests
    - Load/Stress
    - Security
    - Usability
      - Never automated!
    - Etc.

Testing Discussion

- What is your company's testing practice?
  - Is testing integrated with development?
  - Is testing driven by requirements documents?
    - Could test documents replace requirements documents?
  - How much testing is automated?

REFACTORING

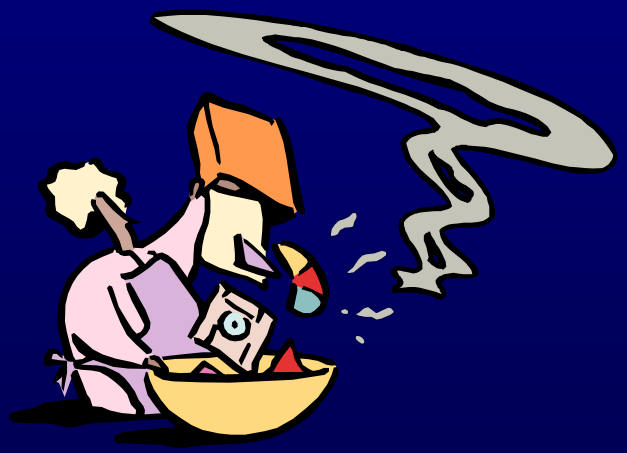
1. Simplicity
   - The goal of most patterns
2. Clarity
   - Common language
   - Encapsulation
   - Self-documenting code
3. Suitable for Use
   - Usability
   - Performance
4. No Repetition
   - NO REPITITION!
5. No Extra Features
   - No Code Before its Time
   - No Code After its Time



Isn't Refactoring Rework?

Absolutely not!

- Refactoring is the outcome of learning
- Refactoring is the cornerstone of improvement
- Refactoring builds in the capacity to change
- Refactoring doesn't cost, it pays



Techniques for Emergence

- Use automated test harnesses
  - Legacy software is software without a test harness
- Refractor ruthlessly
  - Refactoring is NOT rework
- Use devisable architectures
  - Based on a deep understanding of the domain
- Provide Technical Leadership
  - And Communities of Expertise
- Use Set-Based Design
  - Keep Options Open

LEADERSHIP

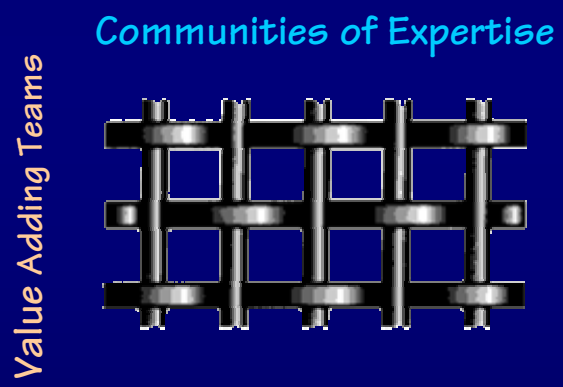
- Champion
  - Creates the vision
  - Recruits the team
  - Finds Support
  - ‘Responsible’ for the design

- TOYOTA Chief Engineer
  - Understands the Target Customer
  - Writes the Product Concept
  - Brings Customer Vision to Technical Workers
  - Makes Key Technical Tradeoff Decisions

- SD Master Developer
  - Also Known As:
    - Architect
    - Systems Engineer
    - Chief Programmer

Communities of Expertise

- Matrix
  - 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

POINT-BASED VS. SET-BASED

Point Based Design

Set up a meeting using the point-based model.

A: My best time is 10:00. Can you make it?

B: No, I can't. How about 2:00?

A: Uh, already booked. Can you meet at 3:00?

B: No, 3:00 is bad. 9:00?

?

Set Based Design

Now set up the meeting by communicating about sets.

A: I can meet 10:00 - 1:00 or 3:00 - 5:00. Can you make any of these times?

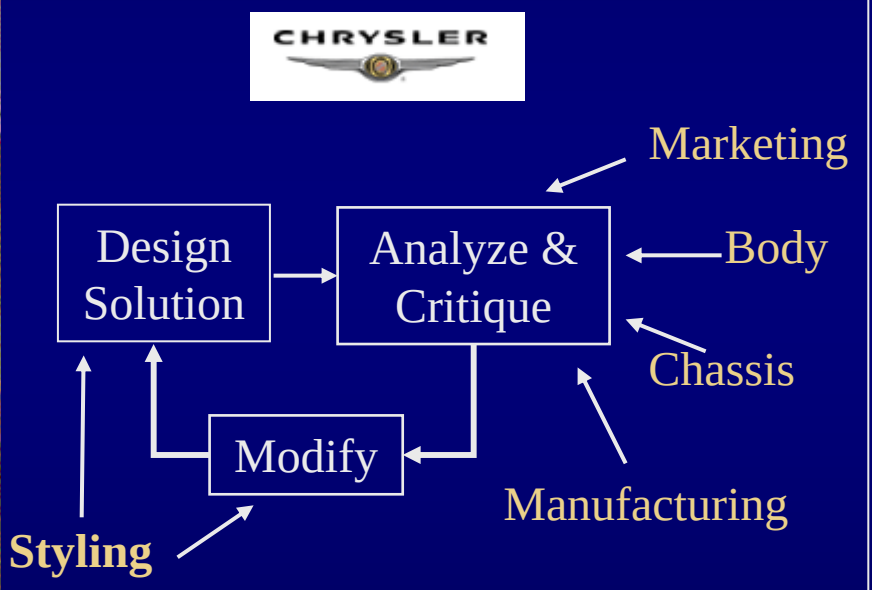
B: Let's meet 12:00 - 1:00.

You already understand this!

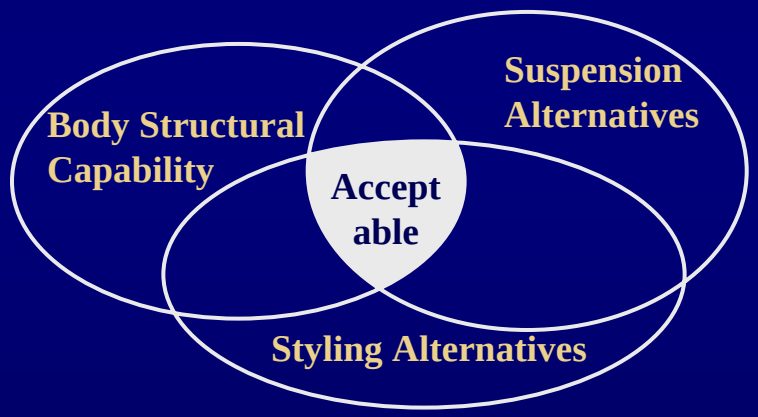
based on dissertation by Durward K. Sobek, II, 1997

SET-BASED DESIGN IS COUNTERINTUITIVE

Point Based Design

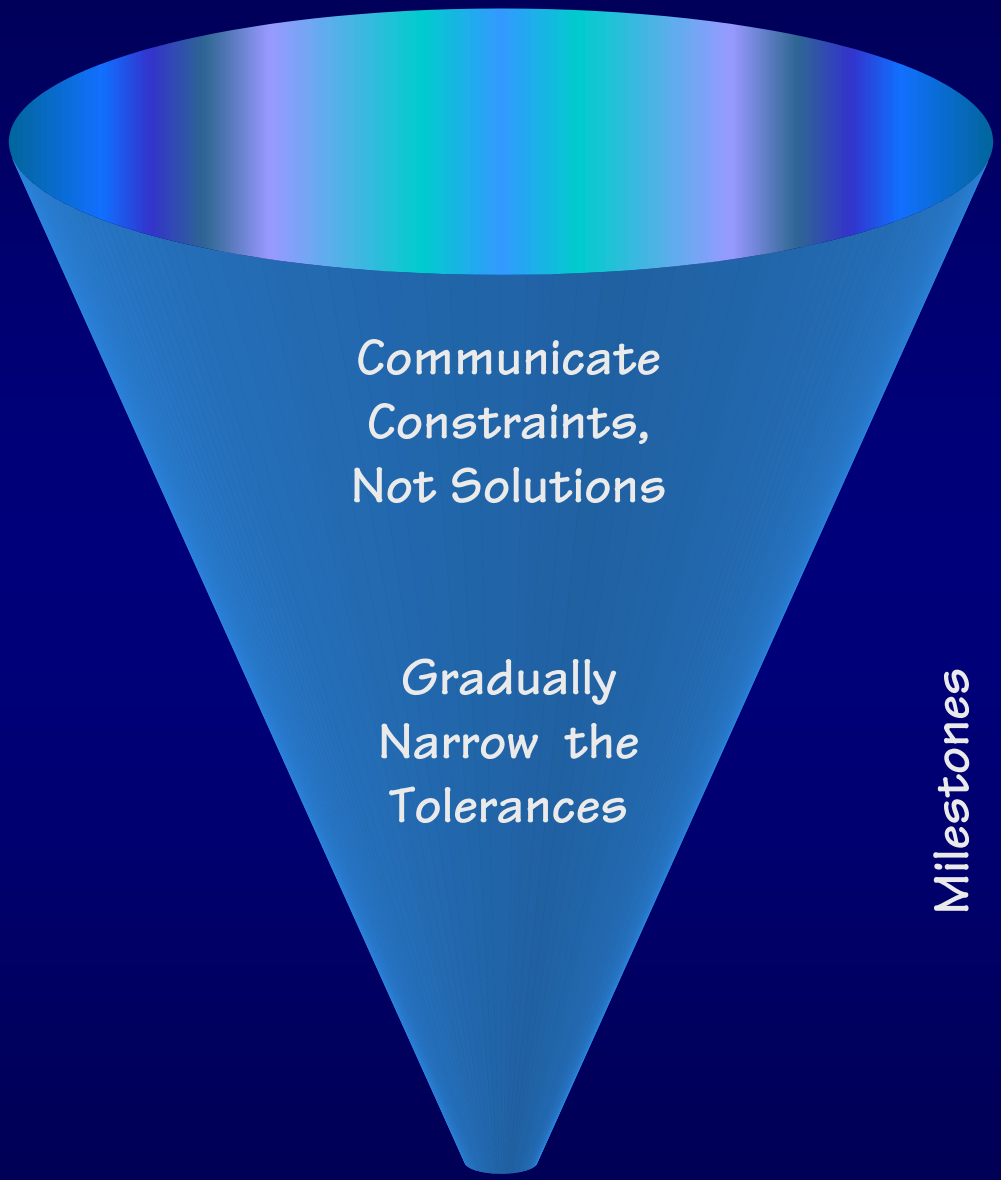


Set Based Design



based on dissertation by Durward K. Sobek, II, 1997

SET-BASED DEVELOPMENT

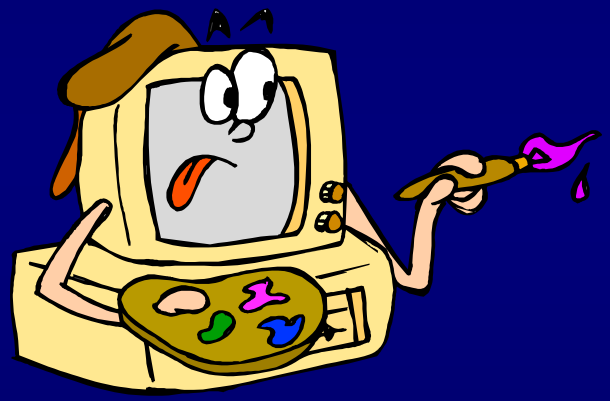


Milestones

- → Vehicle concept
- → Vehicle sketches
- → Clay models
- → Design structure plans
- → First prototype
- → Second prototype
- → Production trials
- → Release to production

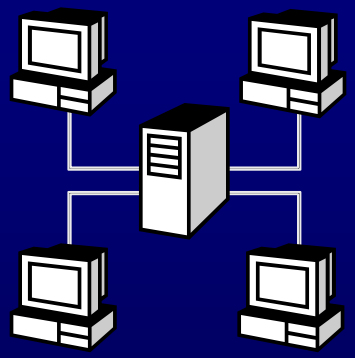
SOFTWARE EXAMPLES

Medical Device Software



Web Site Design

Choosing Technology



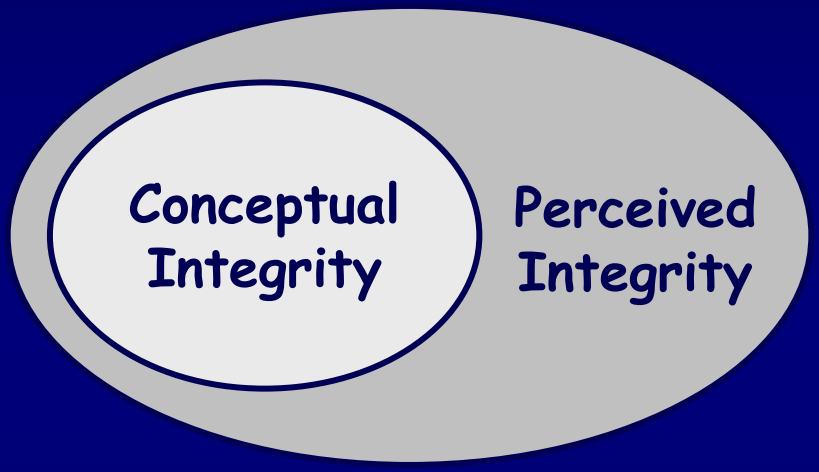
Discussion

- Should TDD be done from developer tests or customer tests?
- Should legacy code be refractored or discarded?
- Is there a place for specialists?
- What is the role of an architect?

SOFTWARE INTEGRITY

- Perceived (External) Integrity

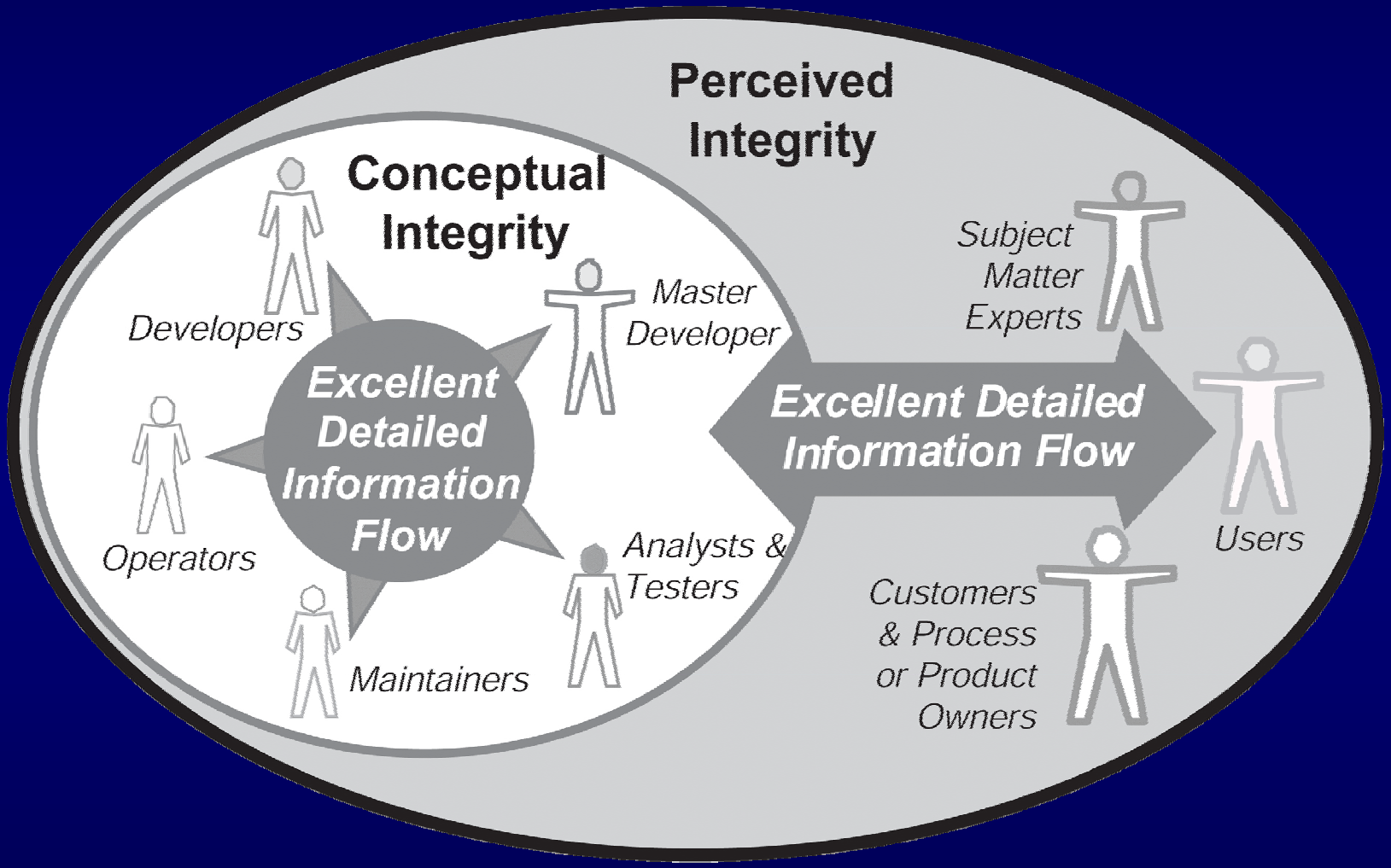
  The totality of the system achieves a balance of function, usability, reliability and economy that delights customers.



- Conceptual (Internal) Integrity

  The system's central concepts work together as a smooth, cohesive whole.

INTEGRITY COMES FROM EXCELLENT, DETAILED INFORMATION FLOW



Agile Customer Toolkit

Why are we doing this?

Mission & Vision
Capabilities
Success Model
Priorities

What needs to be done?

Role Model, UC Model, UI Model
MMF's, User Stories -> Customer Tests

How do we build it?

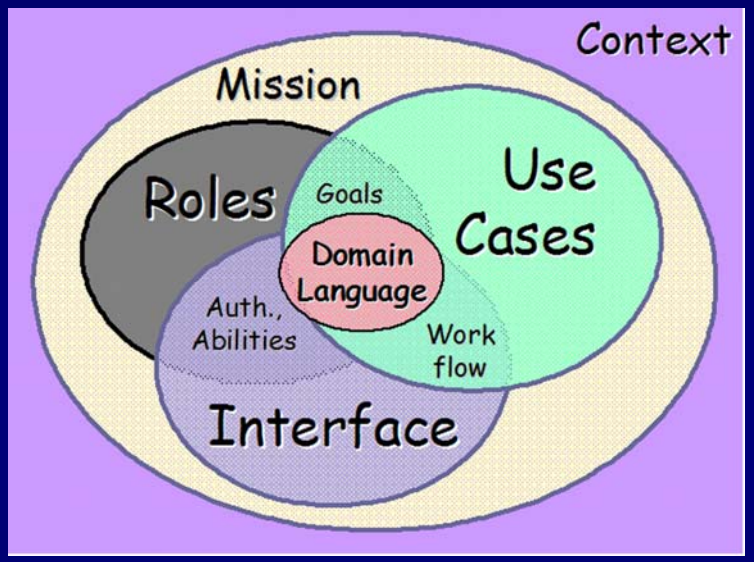
Programmer Tests -> Working Software

How do we support it?

One Domain Language

DOMAIN DRIVEN DESIGN

- Find the right words
  - Domain Language
- Decide what to do
  - Roles
    - *Characters*
  - Use Cases
    - *Plot, Dialog*
  - Interfaces
    - *Action*
- Understand Constraints
  - -abilities



CONCEPTUAL INTEGRITY

Integrated Product Team

Least Integrated		Most Integrated
Sequential (phased)	Timing of Upstream-Downstream Activities	Stage Overlap (simultaneous)
Documents e-mail	Richness of Information Media	Face-to-Face (high bandwidth)
Batch Transmission (one-shot)	Frequency of Information Transmission	Fragmented (piece-by-piece)
Unilateral	Direction of Communication	Bilateral (feedback)
Late Release Of Complete Information	Timing of Upstream-Downstream Information Flow	Early Release of Preliminary Information

DISCUSSION: INTEGRATED PRODUCT TEAMS

- You are asked to recommend members for an IPT for your organization.
  - What functions would you have on it?
  - What level of people in the organization?
  - Who would lead it?
  - How often would it meet?
  - Sketch a typical meeting agenda.

PRINCIPLES OF LEAN THINKING

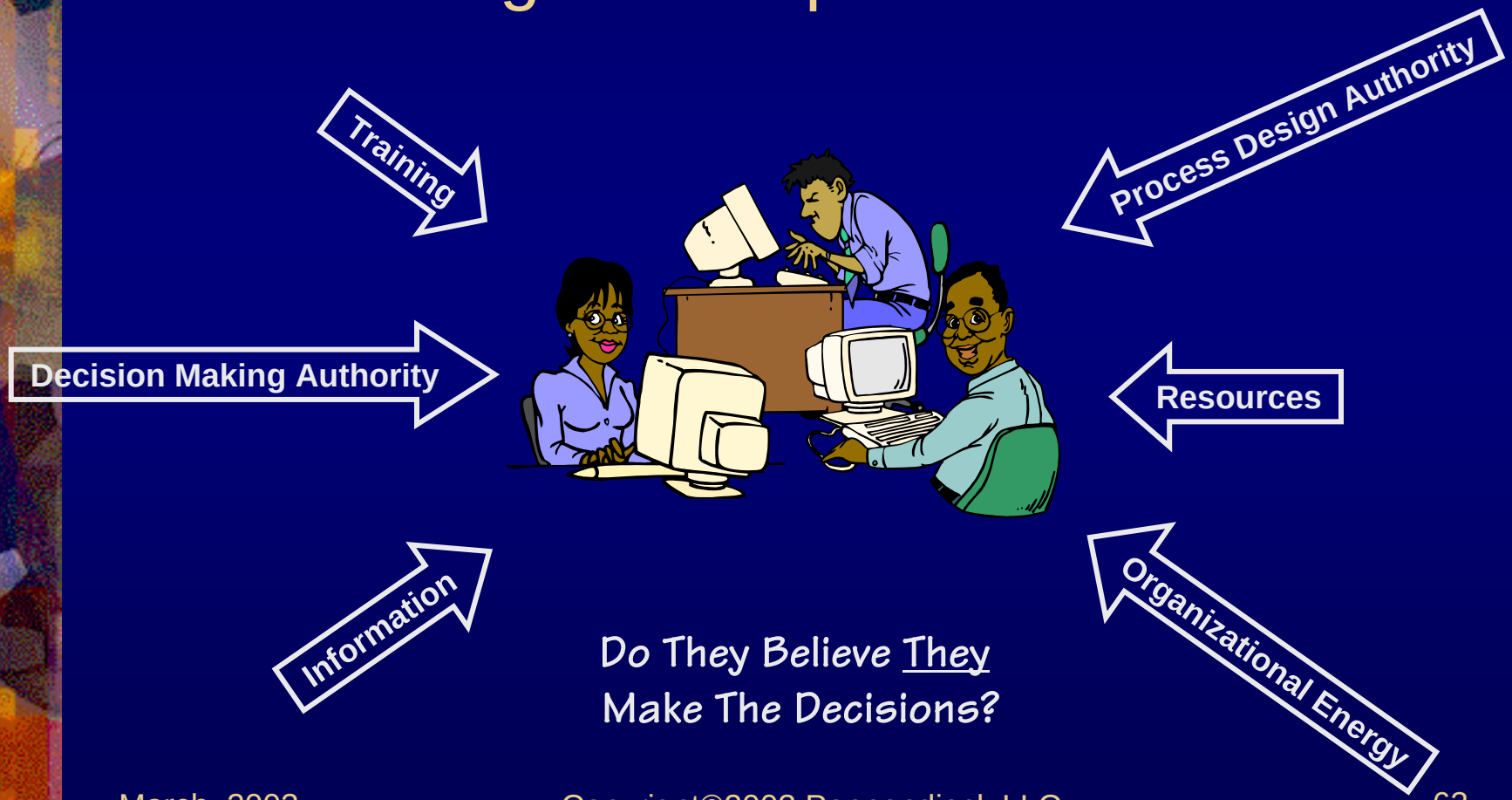
1. ELIMINATE WASTE
2. INCREASE FEEDBACK
3. DELAY COMMITMENT
4. DELIVER FAST
5. BUILD INTEGRITY IN
6. EMPOWER THE TEAM
7. SEE THE WHOLE

PRINCIPLE 6: EMPOWER THE TEAM

- 1982 – GM Closed the Fremont, CA Plant
  - Lowest Productivity
  - Highest Absenteeism
- 1983 – Reopened as NUMMI (Toyota & GM)
  - Same work force
  - White-collar jobs switch from directing to support
  - Small work teams trained to design, measure, standardize and optimize their own work
- 1985
  - Productivity & quality doubled, exceeded all other GM plants
  - Drug and alcohol abuse disappeared
  - Absenteeism virtually stopped
  - Time to expand the plant

VALUE THOSE WHO ADD VALUE

- Who decides what they do next?
- Who designs their processes?

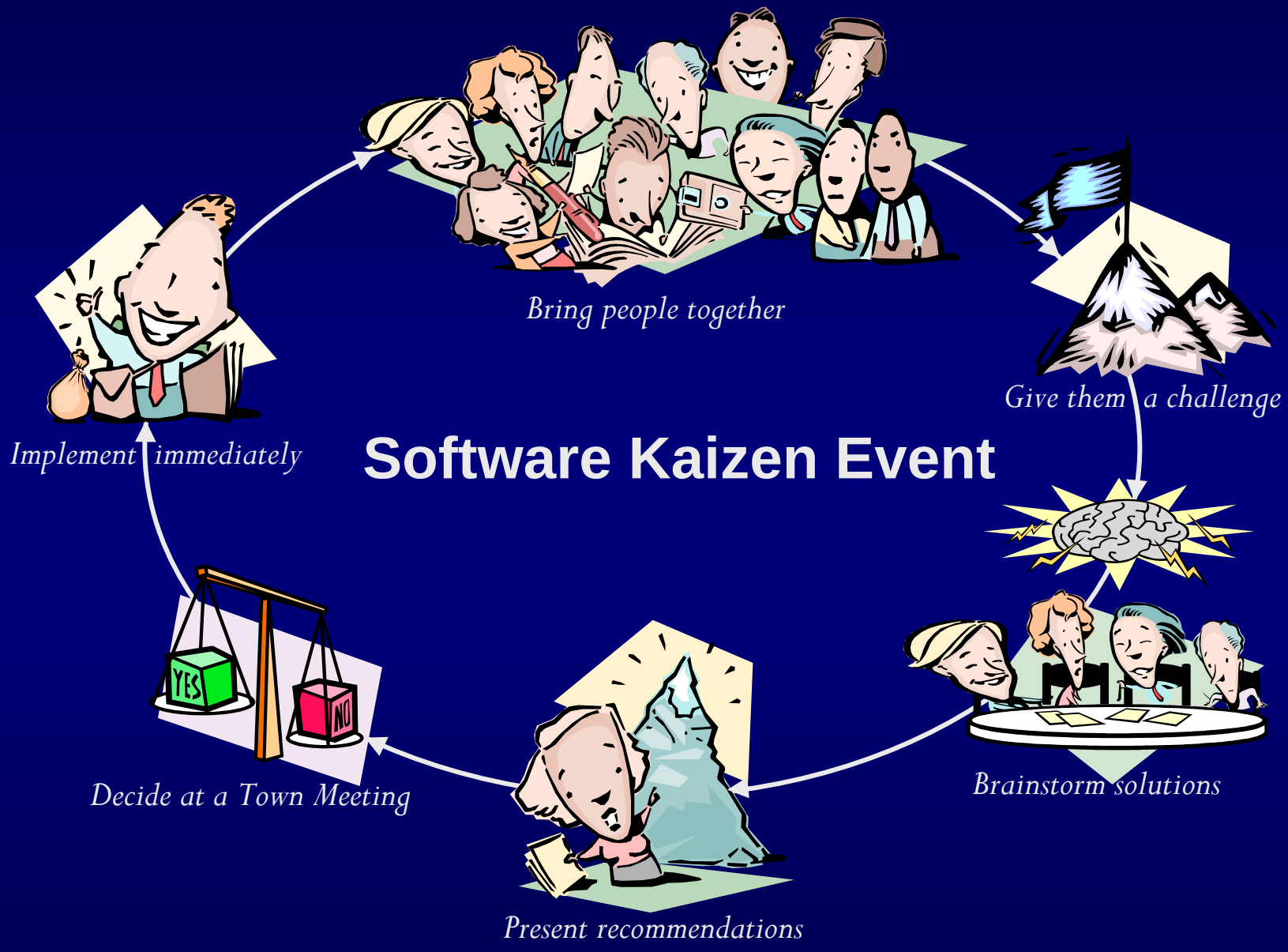


Do They Believe They
Make The Decisions?

Team Commitment

1. Small Team
2. Clear Mission
3. Short Timeframe
4. Staffed with the necessary skills
   - Technology Expertise
   - Domain Experience
5. Enough information to determine feasibility
6. Assured of getting needed resources
7. Freedom to make decisions
8. Basic environment for good programming
   - Coding Standards
   - Version Control Tool
   - Automated Build Process
   - Automated Testing

Software Kaizen Event



PRINCIPLE 7: SEE THE WHOLE

MEASURE **DOWN**

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

Span of Control

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

MEASURE **UP**

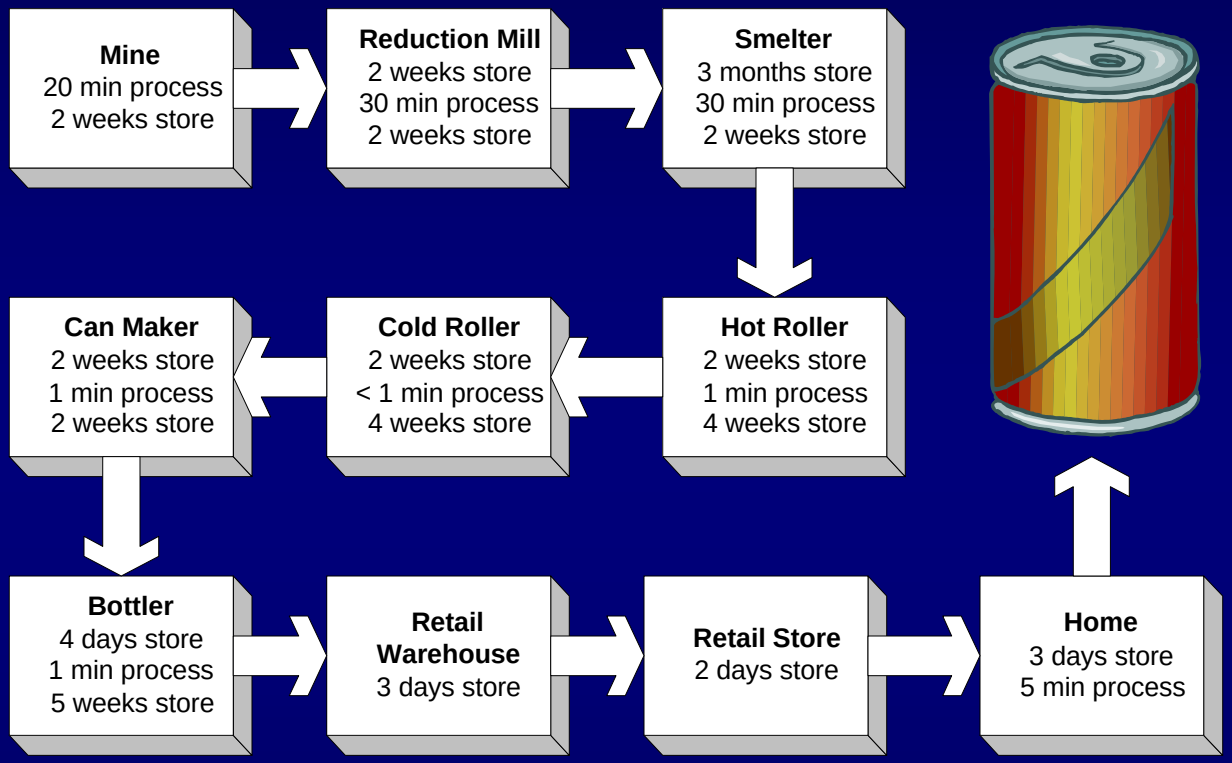
Aggregation

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

Span of Influence

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

BEYOND COMPANY BOUNDARIES



- 319 days
- 3 hours (0.04%) processing time
- Everyone Looking Out For Their Own Interests

From Lean Thinking, by James Womack & Daniel Jones, 1996

OPTIMIZE THE ECONOMIC CHAIN

- In every single case, the Keiretsu (K-ret-soo), that is, the integration into one management system of enterprises that are linked economically, has given a cost advantage of at least 25% and more often 30%.*

- Keiretsu : a group of affiliated companies in a tight-knit alliance that work toward each other's mutual success.
  - GM: 1920's – 1960's
    - Ownership
  - Sears: 1930's – 1970's
    - Partial ownership, contracts
  - Marks & Spencer: 1930's – ?
    - Contracts
  - Toyota: 1950's – present
    - Contracts, economic incentives

** Management Challenge for the 21st Century,* Peter Drucker

How to Get Started

1. Assemble a Keiretsu
2. Map the existing value stream
3. Map the future value stream
   - Use Lean Principles
   - Indicate where key changes are needed
4. Use Kaizen events to create change
5. Repeat from (1.)

EXERCISE

- At what level can you assemble a Keiretsu?
- What organizations would be in the Keiretsu?
- Draw a current map for your Keiretsu.
- Draw a future map.
- List the Kaizen Events for achieving the future map.

PRINCIPLES OF LEAN THINKING

1. ELIMINATE WASTE
2. AMPLIFY LEARNING
3. DECIDE AS LATE AS POSSIBLE
4. DELIVER AS FAST AS POSSIBLE
5. EMPOWER THE TEAM
6. BUILD INTEGRITY IN
7. SEE THE WHOLE

BIBLIOGRAPHY – LEAN THINKING

Austin, Robert D. *Measuring and Managing Performance in Organizations.* Dorset House, 1996.

Christensen, Clayton M. *The Innovator's Dilemma*. Harvard Business School Press, 2000.

Clark, Kim B., & Takahiro Fujimoto. *Product Development Performance: Strategy, Organization, and Management in the World Auto Industry*. Harvard Business School Press, 1991.

Collins, Jim. *Good to Great: Why Some Companies Make the Leap…and Others Don't.* HarperBusiness, 2001.

Drucker, Peter F, *Management Challenges for the 21st Century*, Harper Business2001

Dyer, Jeffrey H. *Collaborative Advantage: Winning Through Extended Enterprise Supplier Networks.* Oxford University Press; 2000.

Freedman, David H. *Corps Business*. HarperBusiness, 2000.

Goldratt, Eliyahu M. *The Goal: A Process of Ongoing Improvement*, 2nd rev. ed. North River Press, 1992.

Klein, Gary. *Sources of Power: How People Make Decisions.* MIT Press, 1999.

O'Reilly, Charles A., III, & Jeffrey Pfeffer. *Hidden Value: How Great Companies Achieve Extraordinary Results with Ordinary People*. Harvard Business School Press, 2000.

Ohno, Taiichi. *The Toyota Production System: Beyond Large-Scale Production.* Productivity Press, 1988.

Reinertsen, Donald G. *Managing the Design Factory: A Product Developer's Toolkit.* Free Press, 1997.

Smith, Preston G., & Donald G. Reinertsen. *Developing Products in Half the Time: New Rules, New Tools*, 2nd ed. John Wiley & Sons, 1998.

Ward, Allen, Jeffrey K. Liker, John J. Cristaino, & Durward K. Sobek, II. "The Second Toyota Paradox: How Delaying Decisions Can Make Better Cars Faster." *Sloan Management Review 36*(3): Spring 1995, 43–61.

Womack, James P., & Daniel T. Jones. *Lean Thinking, Banish Waste and Create Wealth in your Corporation.* Simon and Schuster, 1996. Second Edition published in 2003.

Womack, James P., Daniel T. Jones, & Daniel Roos. *The Machine That Changed the World: The Story of Lean Production.* HarperPerennial, 1991.

BIBLIOGRAPHY – SOFTWARE DEVELOPMENT

Beck, Kent, *Test Driven Development: By Example*, Addison-Wesley, 2003

Beck, Kent, & Martin Fowler. *Planning Extreme Programming.* Addison-Wesley, 2001.

Beck, Kent. *Extreme Programming Explained: Embrace Change.* Addison-Wesley, 1999.

Cockburn, Alistair. *Agile Software Development.* Addison-Wesley, 2002.

Cockburn, Alistair. *Writing Effective Use Cases.* Addison-Wesley, 2000.

Constantine, Larry, & Lucy Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design.* Addison-Wesley, 1999.

Cusumano, Michael A., & Richard W. Selby. *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People.* Simon & Schuster, 1998.

Denne, Mark and Jane Cleland-Huang, *Software by Numbers: Low-Risk, High-Return Development*, Prentice Hall, 2004

Evans, Eric. *Domain Driven Design.* Addison-Wesley, 2003.

Fowler, Martin, *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2003

Highsmith, Jim. *Agile Software Development Ecosystems.* Addison-Wesley, 2002.

Highsmith, James A. *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems.* Dorset House, 2000.

Hohmann, Luke. *Beyond Software Architecture: Creating and Sustaining Winning Solutions*. Addison Wesley, 2003.

Hunt, Andrew, & David Thomas. *The Pragmatic Programmer: From Journeyman to Master.* Addison-Wesley, 2000.

Jeffries, Ron, Ann Anderson, & Chet Hendrickson. *Extreme Programming Installed*. Addison-Wesley, 2001.

Johnson, Jeff. *GUI Bloopers: Don'ts and Do's for Software Developers and Web Designers.* Morgan Kaufmann Publishers, 2000.

Larman, Craig. *Agile & Iterative Development: A Manager's Guide*, Addison-Wesley, 2003

Poppendieck, Mary & Tom Poppendieck. *Lean Software Development: An Agile Toolkit.* Addison-Wesley, 2003.

Schwaber, Ken, & Mike Beedle. *Agile Software Development with Scrum.* Prentice Hall, 2001.

Thimbleby, Harold. "Delaying Commitment." *IEEE Software 5*(3): May 1988.