LEAN
SOFTWARE DEVELOPMENT
AN AGILE TOOLKIT

Mary Poppendieck
mary@poppendieck.com
www.poppendieck.com
THE TOYOTA PRODUCTION SYSTEM

- 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

Taiichi Ohno (1912-1990)
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

- 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

**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

Why are we doing this?

What needs to be done?

How do we build it?

How do we support it?

Domain Context

Communication

Time
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*

<table>
<thead>
<tr>
<th>Inventory</th>
<th>Partially Done Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extra Processing</td>
<td>Paperwork</td>
</tr>
<tr>
<td>Overproduction</td>
<td>Extra Features</td>
</tr>
<tr>
<td>Transportation</td>
<td>Building the Wrong Thing</td>
</tr>
<tr>
<td>Waiting</td>
<td>Waiting for Information</td>
</tr>
<tr>
<td>Motion</td>
<td>Task Switching</td>
</tr>
<tr>
<td>Defects</td>
<td>Defects</td>
</tr>
</tbody>
</table>

## Seven Wastes of Software Development

* 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

- Often or Always Used: 20%
- Always 7%
- Sometimes 16%
- Rarely 19%
- Never 45%

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
    - 1/3\(^{rd}\) of the time
  - **Wait Time:** ~37 Weeks
    - 2/3\(^{rd}\)s 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

What is the Average Cycle Time
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**

- Set Speed 60 mph
- Comparison
- Throttle
- Speed Sensor

**Software Development**

- Customer
- Developer
- Current Business Needs
- Current Design Intent
- Comparison
- Code
- Current System Capability

March, 2003

Copyright © 2003 Poppendieck LLC
The Fundamental Practice

- Waterfall Doesn’t Work!
- Iterative Incremental Development Works!

A simplistic but inferior idea, similar to medicine’s “four humors”.*

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

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

Cost

- Investment
- Payback
- Profit

-breakeven

Self-Funding

Time

Software by Number by Mark Denne and Jane Cleland-Huang
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, notForecasts.

A Minnesota Wedding
- August 10th
  - 50% Chance of Rain
  - 65-95 °F
- 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!

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

### Avoid Repetition
- Don’t Repeat Yourself
- Once & Only Once
- Never copy & paste
- Never!

### 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 **reliably** and **repeatedly** convert customer requests to deployed software

---

**Diagram:**
- Submit Request
- Measure The Average Cycle Time
- Deploy Code
- Shorter Time = More Maturity
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
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
TIME TO COMPLETE (IN STAFF-DAYS)

(iteration)

Acceptance Tests

Tests Written
Tests Passing

Iteration

March, 2003

Copyright©2003 Poppendieck.LLC
**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:** Reduce Cycle Time
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

Cycle Time as a Function of Utilization and Batch Size

- Large Batches
- Medium Batches
- Small Batches

Cycle Time (hours)

Utilization (%)
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

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

Don’t
• Put off refactoring
• Open up visibility just for testing
• Write time/date brittle tests
• Test generated code
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

Backlog items expanded by team

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**

- Why are we doing this?
- What needs to be done?
- How do we build it?
- How do we support it?

**Refactoring**

- Domain Context
- Communication
- Time

**Why are we doing this?**

- Requirements

**What needs to be done?**

- Feedback

**How do we build it?**

- Test-Driven Development

**How do we support it?**

- Maintenance

**Domain**

- Business Needs
- Design Intent

**System**

- Code
- Current Capability

**Test-Driven Development**

March, 2003

Copyright © 2003 Poppendieck, LLC
TEST-DRIVEN DEVELOPMENT

Requirements

Feedback

Refactoring

Maintenance

Customer → Current Business Needs

Developer → Current Design Intent

Code → Current System Capability

Comparison → System Under Test
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

With Refactoring

Without Refactoring

Productivity vs. 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

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

- **Master Developer** (SD)
  - 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, 3:00 is bad. 9:00?
  - A: Uh, already booked. Can you meet at 3:00?
  - B: No, I can’t. How about 2:00?

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

- Design Solution
- Analyze & Critique
- Modify
- Styling

- Marketing
- Body
- Chassis
- Manufacturing

Set Based Design

- Body Structural Capability
- Suspension Alternatives
- Styling Alternatives
- Acceptable

based on dissertation by Durward K. Sobek, II, 1997
Set-Based Development

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

Communicate Constraints, Not Solutions
Gradually Narrow the Tolerances

March, 2003
SOFTWARE EXAMPLES

Medical Device Software

Choosing Technology

Web Site Design
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

Conceptual Integrity

Developers

Operators

Maintainers

Excellent Detailed Information Flow

Perceived Integrity

Master Developer

Analysts & Testers

Subject Matter Experts

Customers & Process or Product Owners

Users
Why are we doing this?
- Mission & Vision
- Success Model
- Capabilities
- 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

Sequential (phased)

Documents

Richness of Information Media

Frequency of Information Transmission

Direction of Communication

Timing of Upstream-Downstream Information Flow

Stage Overlap (simultaneous)

Face-to-Face (high bandwidth)

Fragmented (piece-by-piece)

Bilateral (feedback)

Early Release of Preliminary Information

Most Integrated

Batch Transmission (one-shot)

Unilateral

Late Release Of Complete Information

Timing of Upstream-Downstream Activities

March, 2003

Copyright ©2003 Poppendieck LLC
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

Bring people together

Give them a challenge

Brainstorm solutions

Present recommendations

Decide at a Town Meeting

Implement immediately

Implement immediately
**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
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


BIBLIOGRAPHY – SOFTWARE DEVELOPMENT

Beck, Kent, Test Driven Development: By Example, Addison-Wesley, 2003
Denne, Mark and Jane Cleland-Huang, Software by Numbers: Low-Risk, High-Return Development, Prentice Hall, 2004
Fowler, Martin, Patterns of Enterprise Application Architecture, Addison-Wesley, 2003