Introduction to Lean Software Development

Speed – Quality – Low Cost

Principles of Lean Software Development

1. Eliminate Waste
   ✓ Drive out Complexity
2. Focus on Learning
   ✓ Use Feedback-Driven Development
3. Build Quality In
   ✓ Mistake-Proof Every Step
4. Defer Commitment
   ✓ Build Change-Tolerant Code
5. Deliver Fast
   ✓ Don’t Batch & Queue
6. Respect People
   ✓ Decide as Low as Possible
7. Optimize the Whole
   ✓ Appreciate the System
Principle 1: Eliminate Waste

Put on Customer Glasses

MUDA
anything that
does not add
VALUE

The 80-20 Rule

Features and Functions Used in a Typical System

Often or Always Used: 20%

Sometimes Used: 16%

Rarely Used: 19%

Never Used: 45%

Standish Group Study Reported at XP2002 by Jim Johnson, Chairman

Rarely or Never Used: 64%
**Principle 2: Focus on Learning**

**Software Development**
Finding ways to turn over more and more of what we know to computers so that we have more space left in our minds to discover more interesting things.

Consider the Truck-Driving Problem:
Program a computer to drive a truck across America unattended, on existing roads, in normal traffic.

This is not a one-pass problem.
It will be solved one module at a time over many years.

Nor is it a software problem.
Don’t even think about the problem without the truck.

In software development –
We try to solve too many truck driving problems
- All at once Without access to the truck!

---

**Cycles of Discovery**

- **Iteration Planning**
- **Daily**
- **Iteration Execution**
- **One Iteration Ahead**
- **Deployment**
- **Stories & Tests**
- **Every 2-4 Weeks**
- **Deployment**
- **Ready Software**
- **Feedback**

---

May 07
Copyright © 2007 Poppendieck, LLC
Benefits
1. **Staged Delivery** => better cash flow / higher profits.
   - Minimum useful feature sets, highest payback first.
2. **Feedback** => closer fit to the real customer problem.
   - IKIWISH: “I’ll know it when I see it.”
3. **Velocity** => story-points delivered each iteration.
   - Velocity is a measure of the capacity of the team.

Dangers
1. Irreversible decisions made too early can be expensive.
   - Defer commitment / maintain options for critical decisions.
2. Lack of systems thinking can lead to sub-optimization.
   - Design for operations and build change-tolerant code.
3. Do not use velocity as a performance measurement.
   - Stable teams in a known domain establish a reliable velocity.

---

**Principle 3:**
**Build Quality In**

**1920’s:**
- **Invention:** Looms that detect a broken warp thread when it breaks and stop immediately.

**1950’s:**
- Assembly lines where workers pull a cord to stop-the-line the moment a defect is detected.
Don’t Tolerate Defects

There are Two Kinds of Inspection
1. Inspection to Find Defects – WASTE
2. Inspection to Prevent Defects – Essential

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

A Quality Process Builds Quality In
If you routinely find defects during verification
– your process is defective.

Building Block Disciplines
Simplicity
✓ Minimum Useful Feature Sets
✓ Common Infrastructure
  ✗ Architecture
  ✗ Conventions
  ✗ Tools
✓ Refactoring
  ✗ Continuous Improvement of the Code Base
  ✗ No Repetition

Mistake-Proofing
✓ Configuration Management
✓ One Click Build
✓ Continuous Integration with Automated Testing
  ✗ Unit Tests
  ✗ Acceptance Tests
  ✗ Production Test Harnesses
  ✗ STOP if the tests don’t pass
✓ Frequent Deployment
  ✗ Automated Release Packaging
  ✗ Automated Install

Copyright©2007 Poppendieck, LLC
**Types of Testing**

### Business Facing
- **Acceptance Tests**
  - Business Intent
  - (Design of the Product)
- **Usability Testing**
- **Exploratory Testing**
- **Unit Tests**
  - Developer Intent
  - (Design of the Code)
- **Property Testing**
  - Response,
  - Security,
  - Scaling,
  - Resilience

### Technology Facing
- Automated: Every Day
- Manual: As Early as Practical
- Automated: Every Build
- Tool-Based: As Early as Possible

---

**Principle 4: Defer Commitment**

**A Pilot from Innsbruck**

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

**Step One:** Accept that Change is not the Enemy.

- 60-80% of all software is developed after first release.

**Step Two:** Invest in Change-Tolerant Practices

- A development process that anticipates change will result in software that tolerates change.
  1. Feedback-Driven Development
  2. Options-Based Development
  3. Test-Driven Development

---

**Principle 5: Deliver Fast**

*Competing on the basis of speed:*

- Significant competitive advantage
- Large barrier to entry

*When speed matters*

*You don’t want to be left behind*
The Flow of Value

**Churn**

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

Why do early specification and late testing seem like such a good idea?

Don’t Batch & Queue

**Lists**

How long is your defect list?

How far apart are your releases?

How many weeks (years?) of work do you have in your backlog?

Why do lists & queues seem like a good idea?
**Principle 6: Respect People**

*Three Stonecutters were asked:*

“What are you doing?”

*I’m cutting stones!*

*I’m building a cathedral.*

*I’m earning a living.*

---

**Engaged People**

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

Stone Cutters or Cathedral Builders?

---

The Litmus Test:

*When workers are annoyed by their job – Do they complain, ignore it, or fix it?*
**Principle 7: Optimize the Whole**

Software is rather useless  
– all by itself

Software is embedded  
In hardware  
In a process  
In an activity

**Appreciate the System**

Build a complete product  
✓ Not just Software  
Look at the whole picture  
✓ Across the Value Stream  
✓ For the Entire Lifecycle

Software is going to be around for a LONG time….

*Saving money in development at the expense of production makes no sense.*

**Measure UP**

**Decomposition**

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

**Example**

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

**Aggregation**

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

**Example**

✓ Measure Cost, Schedule, & Scope  
  × Quality & Customer Satisfaction fall between the cracks  
  × Measure Business Case Realization instead!
Three System Measurements

**Average Cycle Time**
- From Product Concept
- To First Release
  or
- From Feature Request
- To Feature Deployment
  or
- From Defect
- To Patch

**The Business Case**
- P&L or
- ROI or
- Goal of the Investment

**Customer Satisfaction**
- A measure of sustainability

---

Thank You!

*More Information: www.poppendieck.com*