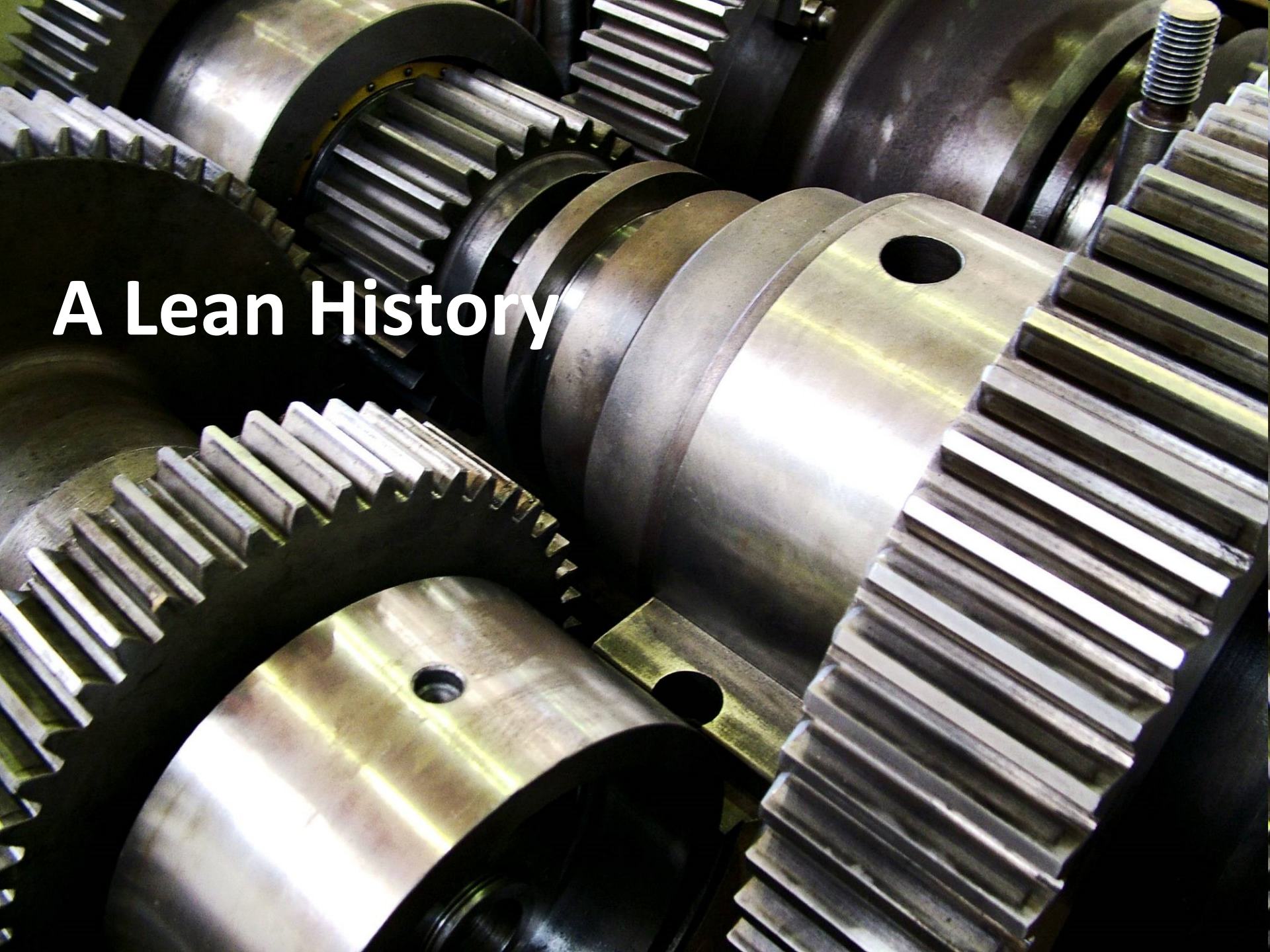
Lean Software Development Principles

John P Vajda, PMP, CSM

Lean Software Development Principles

- A Lean History
- The 7 Principles of Lean
- The 22 Tools of Lean


A Lean History

A Lean History

- **Lean** is a manufacturing & production practice that considers the expenditure of resources for any goal other than the creation of value for the end customer to be wasteful, and thus a target for elimination.
- "value" is defined as any action or process that a customer would be willing to pay for.
- Lean is centered around *preserving value with less work*.
- Lean manufacturing is based on optimizing flow, increasing efficiency, decreasing waste, and using empirical methods to decide what matters, rather than uncritically accepting pre-existing ideas
- Toyota was a leader in implementing lean practices in the 80s

Reference: http://en.wikipedia.org/wiki/Lean_manufacturing

The Seven Principles of Lean Thinking: + 22 Tools

- Eliminate Waste
- Amplify Learning
- Decide as Late as Possible
- Deliver as Fast as Possible
- Empower the Team
- Build Integrity In
- See the Whole

Reference: http://en.wikipedia.org/wiki/Lean_manufacturing

Principle # 1: Eliminate Waste

The Seven Principles: Eliminate Waste

What is waste?

- Anything that doesn't add value (as perceived by the customer) to the product
- Unnecessary code or functionality
- Unclear requirements
- Slow internal communications or processes
- Bureaucracy

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 1

The Seven Principles: Eliminate Waste

Tools 1: Learn to see waste

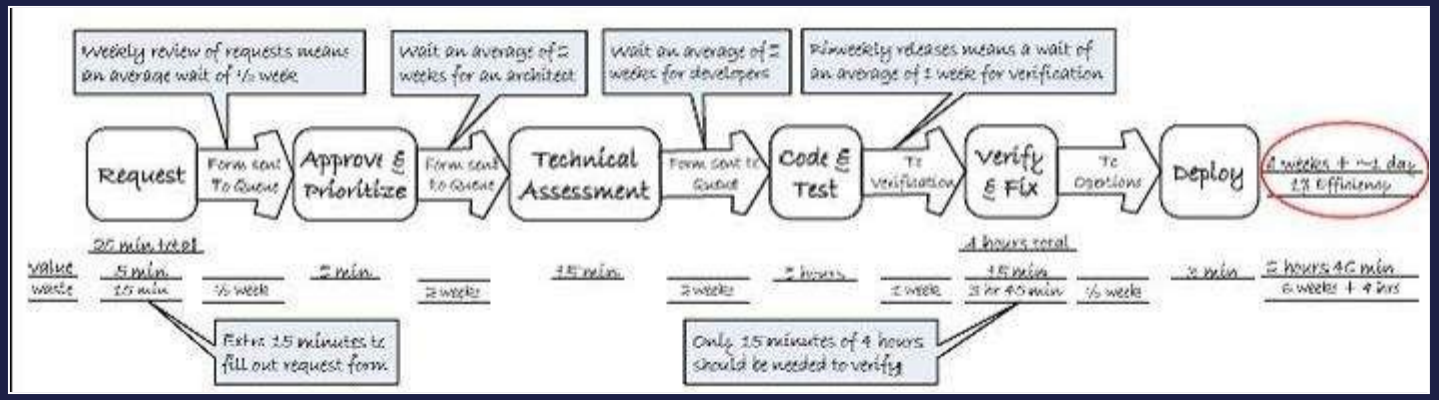
Wastes of Manufacturing	Wastes of Software Development
Inventory	Partially work done
Extra processing	Paperwork or excess documentation
Overproduction	Extra features
Transportation	Building the wrong thing
Waiting	Waiting for the information
Motion	Task switching & Motion
Defects	Defects

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 1

The Seven Principles: Eliminate Waste

Tool 2: Learn to reduce waste

- Reduce management activities such as unnecessary tracking systems. Minimize tracking by create a smooth flowing work system
- Rethink Authorization systems. Make “approvals” unnecessary
- Retrain you brain to see waste. Ask yourself “Why am I really doing this?”
- Map your value stream to indentify inefficiencies



Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 1

The Seven Principles: Eliminate Waste

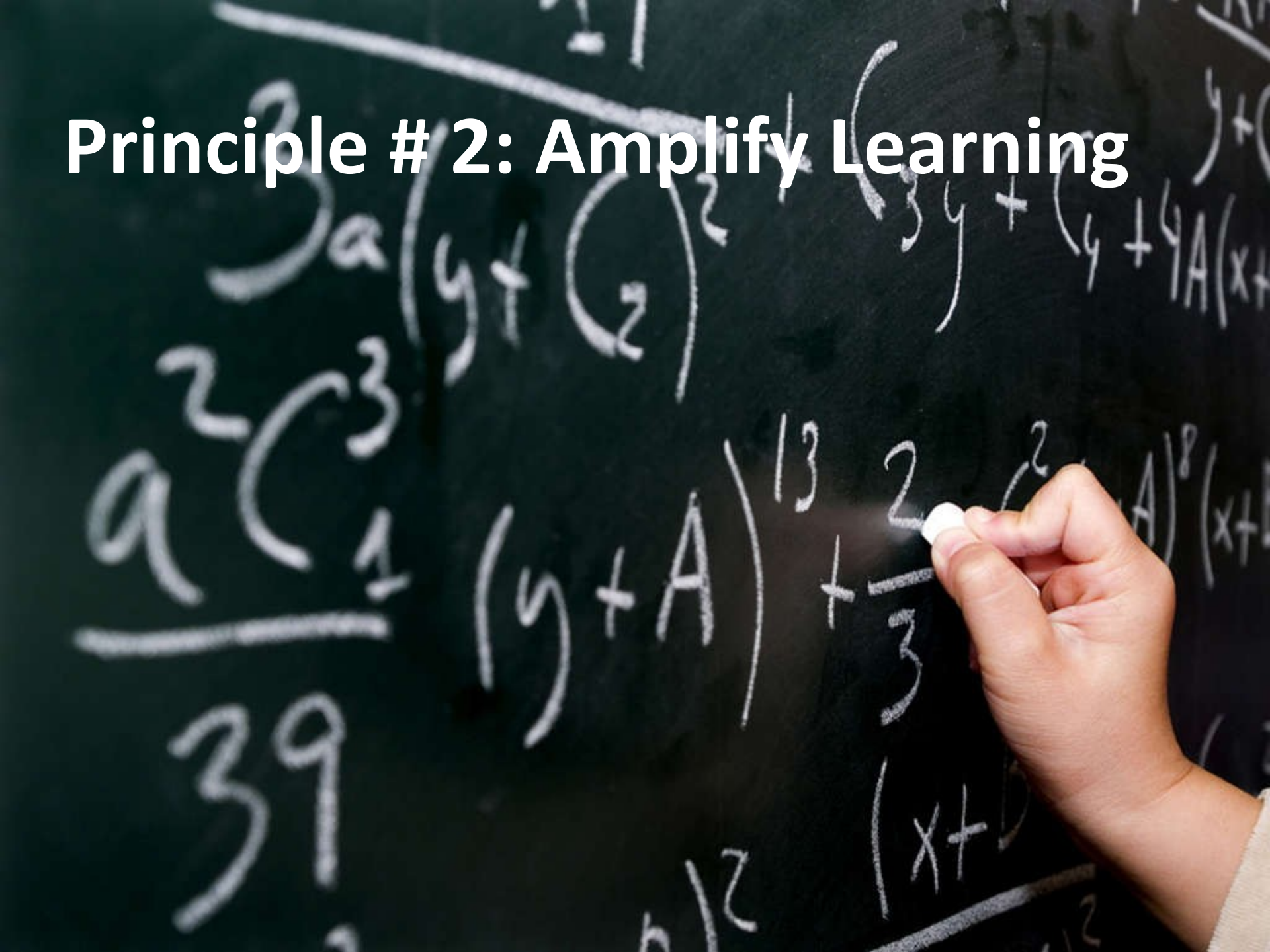
Try This!

- **1.1** Make a list of the 10 – 15 most important activities in your organization. Pretend you are a customer and rate them 1 to 5. (1 being the customer doesn't really care, 5 being they value this highly)
- **1.2** Take the 2 lowest scoring (**the waste**) and plan to cut the time on these activities in half
- **1.3** At a team meeting, discuss the 7 principles of Lean and ask the following questions about **the waste:**

  - Do you agree this is "waste" why or why not?
  - Estimate how much time it consumes each week
  - What can or should be done to reduce that time?

- **2**. Develop a Value Stream of an incoming request a map a timeline of it's progress. Compare added value versus waiting. Take the biggest cause of delay and plan to cut that in half**.**

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 1



Principle # 2: Amplify Learning

The Seven Principles: Amplify Learning

The Nature of Software Development

- Realization of purpose of use rather than conforming to requirements
- Not intended to produce repeatable results, but to create solutions to unique customer problems
- Design is done best using discovery solutions: short, repeated cycles of investigation, experimentation and checking results
- We like to Try it, Test it, and Fix it!
- Knowledge generation (or feedback) loops are critical

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 2

The Seven Principles: Amplify Learning

Tool 3: Feedback

- Introduce and increase feedback loops into the development process
- Run tests as soon as code is written, don't let defects accumulate
- Less Documentation, and more coding with real time feedback
- Forgo requirements gathering sessions for prototype reviews of UI
- Don't over study which tools are the best, take the top 3 and evaluate them
- Encourage and accept immediate customer response to your work

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 2

The Seven Principles: Amplify Learning

Tool 4: Iterations

- Are Short useful cycles of software development: Design > Programmed > Tested > Integrated > Delivered
- Allow the practice of “Queuing Theory”, small batches moving rapidly through an efficient system
- Allow Feedback to increase, thusly increasing control
- Allow an option-based (open options) approach
- Allow the ability to “decide as late as possible”
- Require team planning, short complete cycles and team commitment
- Allow scope to be “negotiable”, by prioritizing highest value items first
- Short feedback loops create convergence, and a move toward uniformity

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 2

The Seven Principles: Amplify Learning

Tool 5: Synchronization

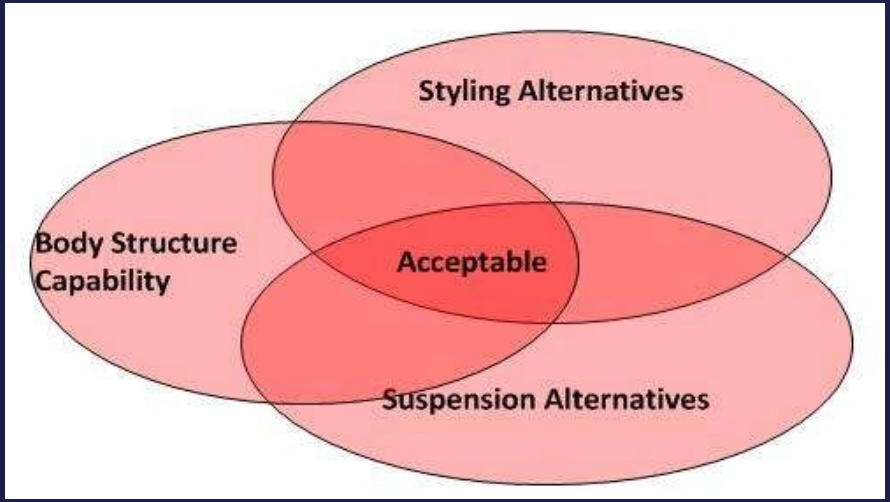
- Requires a configuration management system
- Requires a daily build and smoke test
- Requires automated testing
- Allows you to build spanning applications, “driving a nail through the entire system”
- Allows you to develop separate components simultaneously using an architecture matrix strategy
- Requires a high level of communication

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 2

The Seven Principles: Amplify Learning

Tool 6: Set Based Development

- Is about communicating constraints not choices or solutions
- You develop multiple options, communicate constraints, and let solutions emerge
- Will let iterations leave latitude for implementing the rest of the system as you progress



Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 2

The Seven Principles: Amplify Learning

Try This!

1. Take your most difficult problem and devise a way to increase feedback

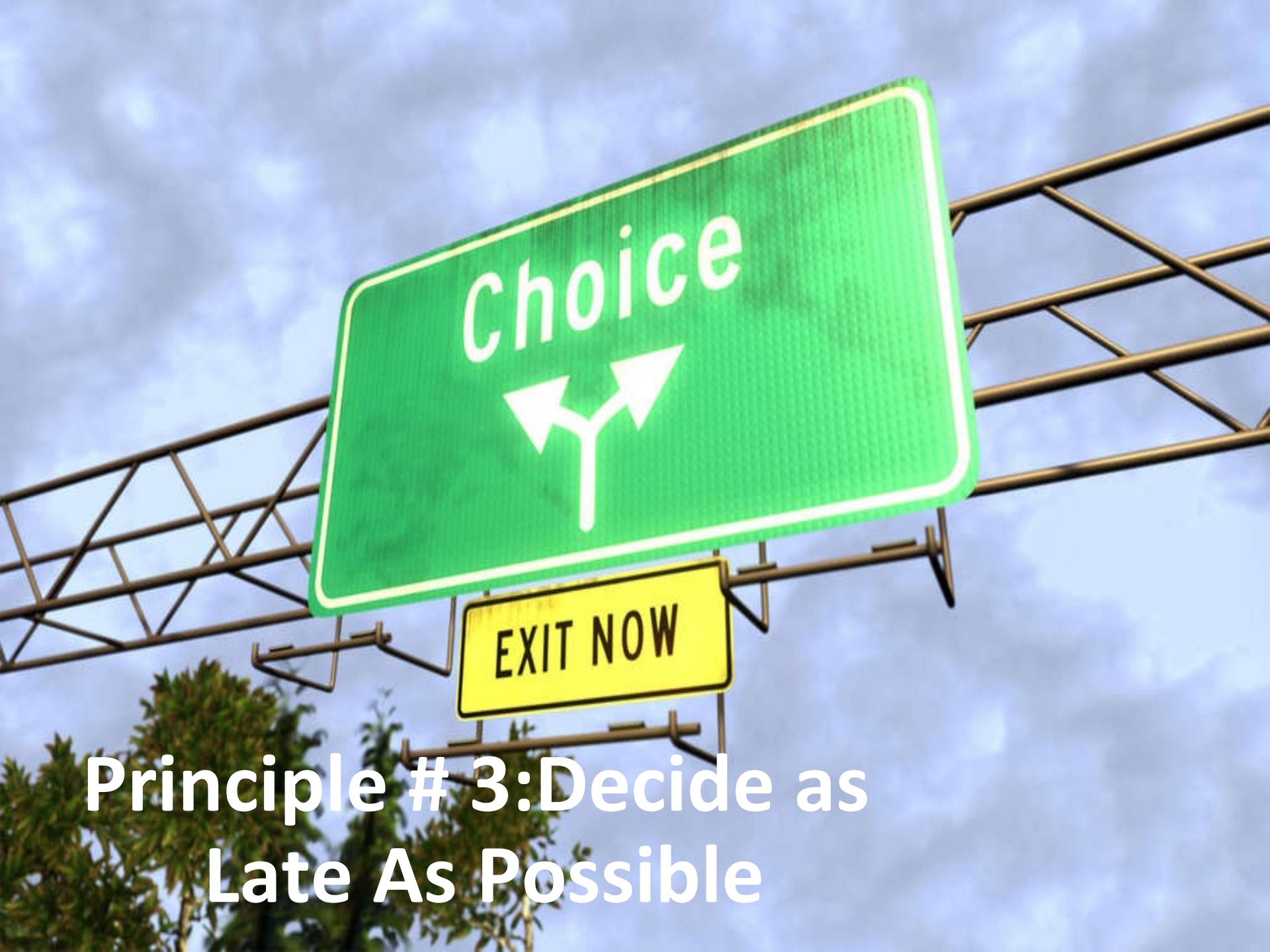
Development to Management - Ask these questions:

- Was the team properly staffed?
- Where request for resources met?
- What is getting in the way?
- What can be changed to make things easier?

Development to Customers- Ask these questions:

- How well does the work delivered solve your problem?
- How can it be improved?
- Does the work delivered give you confidence in the direction of the overall project?

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 1



Principle # 3:Decide as Late As Possible

The Seven Principles: Decide as Late as Possible

Sequential vs. Concurrent

Sequential	Concurrent
Depth First (Drilling into the details too fast)	Breadth First (seeing things in full view over time)
Order of Creation (On day 1, We created requirements)	Highest valued features first
Rigid, change is costly	Adaptable, change is manageable
Cost escalation is high when issues are found late	Cost escalation is low, as issues are found incrementally
High stakes decisions are made based on assumptions and with uncertainly	High stakes decisions can be deferred until the last responsible moment

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 3

The Seven Principles: Decide as Late as Possible

Tool 7: Options Thinking

- A right, but not an obligation to do something in the future
- Customer needs aren't always clear or understood
- You can't predict the future, so maintain flexibility, until uncertainty is removed
- Options are like trade offs, they aren't free and have a cost

Example: Building the GMA architecture (and having a back up strategy to manage the data) with the ability to use the data created by the GMDB, without actually knowing if the GMDB project would be successful.

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 3

The Seven Principles: Decide as Late as Possible

Tool 8: The Last Responsible Moment

- The moment at which failing to make a decision eliminates an important alternative
- This isn't Procrastination!

How do you do it?

- Share partially complete design information
- Organize for collaboration
- Develop a change oriented mindset
- Practice Object Orient Design or Component Based Development
- Know what is critically important in the domain
- Develop a sense of "when" decisions need to be made

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 3

The Seven Principles: Decide as Late as Possible

Tool 9: Making Decisions

Depth First	Breadth First
Making early commitments	Delaying commitments
Needs an agreed to point to “zero” in on	Requires someone savvy to understand how the details will emerge and when it’s time to commit
Rational decision making	Intuitive decision making
Predictions and assumptions drives decisions	Real time information and feedback drives decisions

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 3

The Seven Principles: Decide as Late as Possible

Try This!

- **1**. Make a list of decisions that need to be made and group them into 2 categories, tough to make and easy to make. Discuss what you would need to make the tough decisions easier.

- **2**. Evaluate your personality, are you more inclined to use Depth or Breadth first problem solving? Find someone who is the opposite of you and decide how to approach a decision together.

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 1

Principle # 4: Deliver as Fast as Possible

The Seven Principles: Deliver Fast as Possible

Why Deliver Fast?

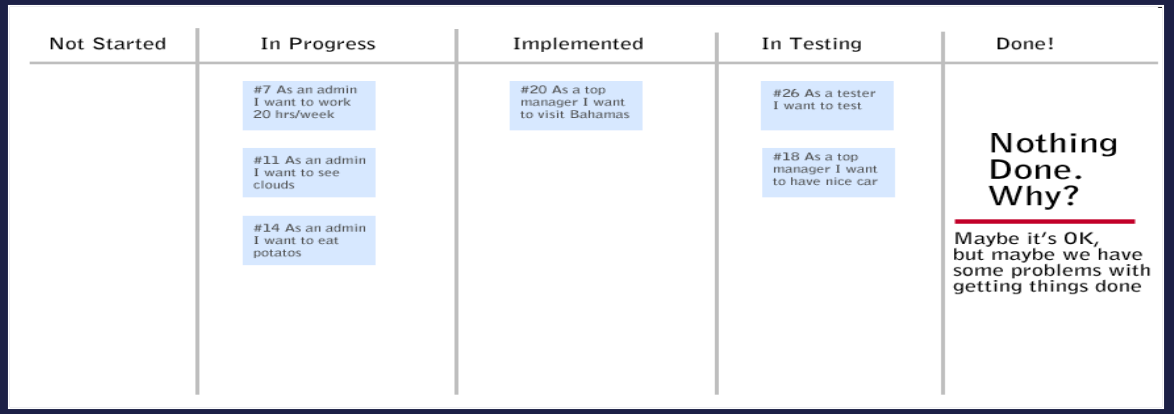
- Customers like rapid delivery. Look at how UPS and FedEx provide ordered products to customers in 1 to 3 days
- Rapid delivery means less time for customers to change their minds
- In-process, or partially done work can have undiscovered defects
- The faster you deliver the longer you can delay decisions. Being able to make a change in a week, lets you wait to make a decision until that week.

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 4

The Seven Principles: Deliver Fast as Possible

Tool 10: Pull Systems

- Allows people to figure out for themselves what needs to be done. Work becomes self directing
- Complex linear schedules can be negated as soon as one “glitch” or variation is introduced
- A Just in Time approach allows for decision about work to be made real time, not in advance



Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 4

The Seven Principles: Deliver Fast as Possible

Tool 11: Queuing Theory

- The key is to reduce cycle time, or the time it takes to get from one end of the process to the other

Steady Rate of Arrival

- Practice a steady rate of arrival and control what comes into your queue.
- Releasing small packages of work, allows you to spread it evenly thorough the team
- Setting priorities and selecting work is critical
- Releasing work frequently is even more critical

Steady Rate of Service

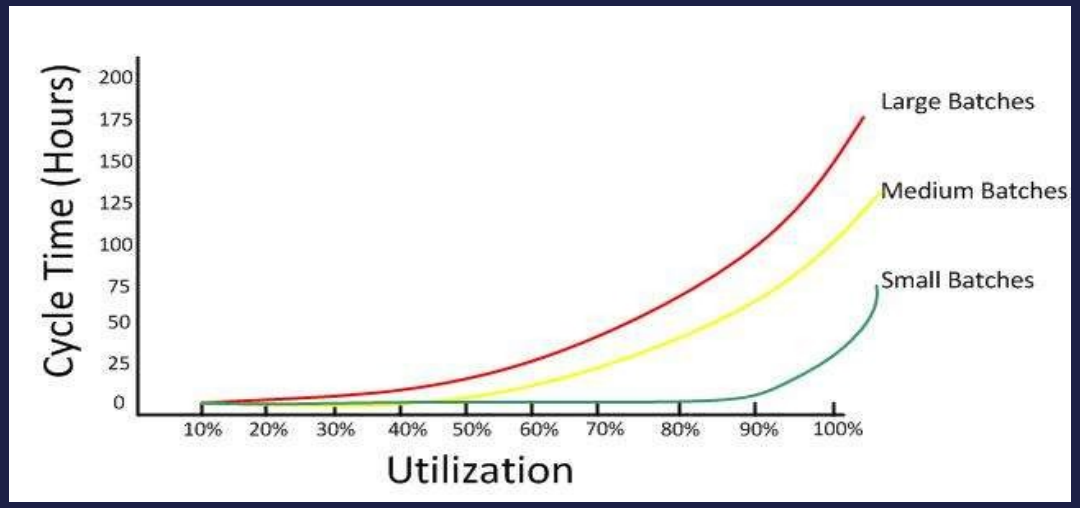
- Remove the variability from processing time
- Smaller work packages have less that can go wrong
- Parallel the processing of work to avoid bottlenecks
- Deliver consistently!

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 4

The Seven Principles: Deliver Fast as Possible

Managing Slack

- 100% utilization isn't always the most efficient, as we need to room to change.
- Practice the 80/20 rule for work load
- Large batches of work take longer to process through the "queue" and requires more people to be completed.



The larger the batch of work, the slower it will be completed, and the more utilization it will take.

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 4

The Seven Principles: Deliver Fast as Possible

Tool 12: Cost of Delay

- Rapid Development will save you time and money, management may think differently ☺
- Determine what delayed delivery will cost you by using a profit and loss statement
- Develop an Economic Application Model (breakdown of costs)
- Use your P&L and Economic Models to drive trade off decisions

P&L				
	Assumptions	Year 0	Year 1	Year 2
Revenue				
Avg. Selling price	Decreases 10% a yr	$1,000	$900	$800
Total market units		10,000	20,000	40,000
Market Share		20%	30%	40%
Units Sold		3,000	8,000	20,000
Total Revenue		$3,000,000	$7,200,000	$16,200,000
Expense				
Mfg & Distribution Cost	Decrease 5% a yr	$200	$190	$181
Support Cost	Decrease 10% a yr	$200	$180	$162
Total Unit Cost		$400	$370	$343
Mfg Support Costs		$1,200,000	$2,960,000	$6,850,000
Gross Margin		$1,800,000	4,240,000	9,350,000
Gross Margin %		60%	59%	58%

This is just a section of a P&L

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 4

The Seven Principles: Deliver Fast as Possible

Try This!

- **1.** Create a P&L and economic model for an application you lead the development for. More information on this process can be found here M & T Poppendieck, Lean Software Development. 2003 : Chapter 4

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 4

Principle # 5: Empowering the Team

The Seven Principles: Empower the Team

What is a "Mature" Organization?

- Lean Assumption #1 – A mature organization looks at the whole system; it doesn't only focus on optimizing disparate or separate parts.

- Lean Assumption #2 – A mature organization focuses on learning effectively and empowers the people who do the work to make decisions.

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 4

The Seven Principles: Empower the Team

Tool 13: Self-Determination

- Transfer practices from one environment to another is often an mistake
- Let the team design their own working procedures
- Remember Management's role is coach, train, and assist the teams
- It is key to understand the fundamentals principals that make up practices, and **transform** those principles into new practices
- Managers need to improve as much as individual workers. A feedback loop is critical going both ways between managers and workers to drive improvement

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 5

The Seven Principles: Empower the Team

Tool 14: Motivation

- Create a sense of purpose at work. People care about a purpose greater then themselves
- It must be clear
- It must be achievable
- The team must have access to customers
- Let the team make it's own commitments
- Management's role is to provide support, resources, guidance, and protection

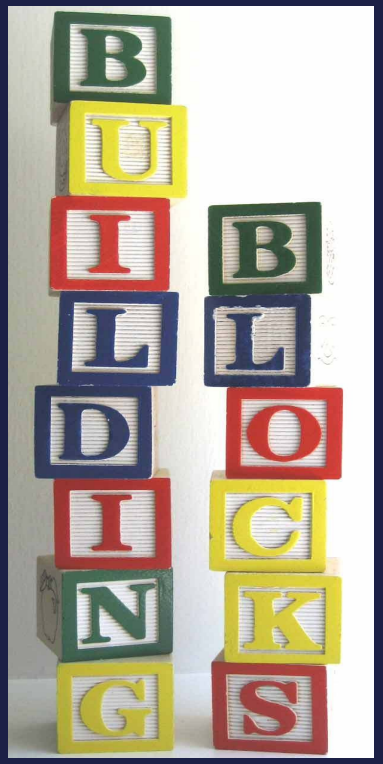
Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 5

The Seven Principles: Empower the Team

Tool 14: Motivation

The Building Blocks:

- Belonging
- Safety
- Competence, discipline and good practices
- Progress by meaningful measurements



Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 5

The Seven Principles: Empower the Team

Tool 15: Leadership

Managers	Leaders
Cope with Complexity	Cope With Change
Plan & Budget	Set Direction
Organize & Staff	Align People
Track & Control	Enable Motivation

- The Team needs Master Developers & Respected Leaders
- A Project Manager needs to become a Project Leader

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 5

The Seven Principles: Empower the Team

Tool 16: Expertise

- Develop and foster Communities of Expertise
- Promote Mentorship and Pair Programming activities
- Encourage training and continued self improvement
- Develop software standards and practice them
- Offer the “Google” approach to personal project work. = 80/20

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 5

The Seven Principles: Empower the Team

Try This!

At the end of each iteration ask the team:

- What is slowing you down, or getting in your way of doing a good job?
- What would help things move faster, better, or cheaper?
- To make a list of good and bad practices. Decide to implement a good practice and eliminate a bad practice. – Make it happen!

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 5

Principle # 6: Build Integrity In

The Seven Principles: Build Integrity Within

- **Perceived Integrity**: is affected by the customer's whole experience of a system
- **Conceptual Integrity:** means that system's central concepts work together as a smooth cohesive whole

Perceived Integrity	Conceptual Integrity
How intuitive is the system?	Does it have an effective balance between flexibility, maintainability, efficiency, and responsiveness?
How does it keep up with changes in the domain?	Can the system evolve and mature?
How well does it solve problems?	Does it have a consistent set of design principles?
How much market share does it have?	Is usability consistent?
How much mind share does it have?	a perquisite for perceived integrity

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 6

The Seven Principles: Build Integrity Within

Tool 17: Perceived Integrity

- Smaller systems should be developed by a single team that has immediate access to the people who will judge the system's integrity
- Short iterations should be used and feedback should be acquired from a wide range of people that can recognize the integrity
- Customer tests provide excellent customer-development communication
- Complex systems should be represented using models and languages the customer understands and programmers can use without major refinement.
- Large systems should have a master developer who has technical credentials and deep customer understanding

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 6

The Seven Principles: Build Integrity Within

Tool 18: Conceptual Integrity

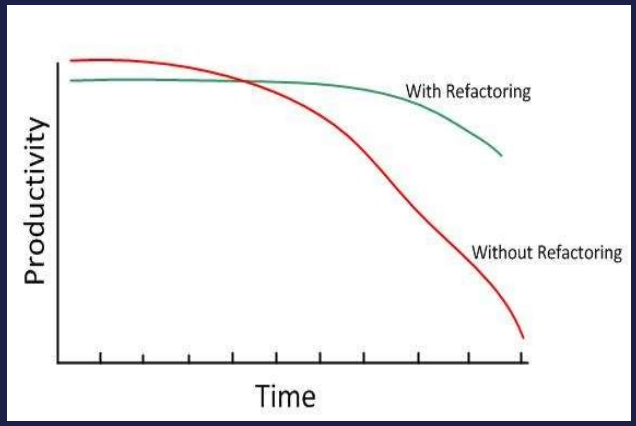
- The effectiveness of communication of all decisions made is critical
- Remove complexity upfront in the design (using existing technology vs. new)
- **Use integrated problem solving:**
  - ✓ Understand the problem and solve the problem at the same time
  - ✓ Preliminary information is released early, not delay until complete information Is available
  - ✓ Information is transferred in small batches not large
  - ✓ Information flows in two directions not one
  - ✓ The preferred method of communication is face to face not just using documentation

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 6

The Seven Principles: Build Integrity Within

Tool 19: Refactoring

- Complex systems have effects that aren't understood at design time
- Architecture must remain healthy as the system matures and evolves
- Maintain Conceptual integrity
  - ✓ **Simplicity:** Simple designs are the best
  - ✓ **Clarity:** Keep code understandable
  - ✓ **Suitability for use:** design for an intended purpose
  - ✓ **No Repetition**: never repeat code
  - ✓ **No Extra Features:** don't build what is not needed



Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 6

The Seven Principles: Build Integrity Within

Tool 20: Testing

- Communicates how things “should “work
- Provides feedback on if the system “actually” works the way it was designed
- Provides scaffolding, allowing developers to make changes through out the development process

Developer Tests	Customer Tests
Unit Tests	Functional Testing
System Tests	Acceptance Testing
Integration Tests	

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 6

The Seven Principles: Build Integrity Within

Communication

- Developer tests communicate how the system should work
- Customer tests communicate, by example, how the system needs to work
- Customer tests can replace requirements (Test Driven Development)
- Designs aren't complete until developer tests are executed (Design Driven Development)

Feedback

- Immediate feedback is a nature part of the development cycle
- Write customer tests during the iteration before the demo
- Automated testing is key for daily builds

Scaffolding

- A supporting framework that lets you do things that would otherwise be dangerous
- Lean techniques could be dangerous if you aren't careful (late code changes, late decisions, etc)
- Automating testing is example of a way to create a safety net
- A Comprehensive Test Suite of customer and developer tests let you assess the health of your product
- To reduce overall maintenance costs, maintain a set of tests through the lifecycle of the system

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 6

The Seven Principles: Build Integrity Within

Try This!

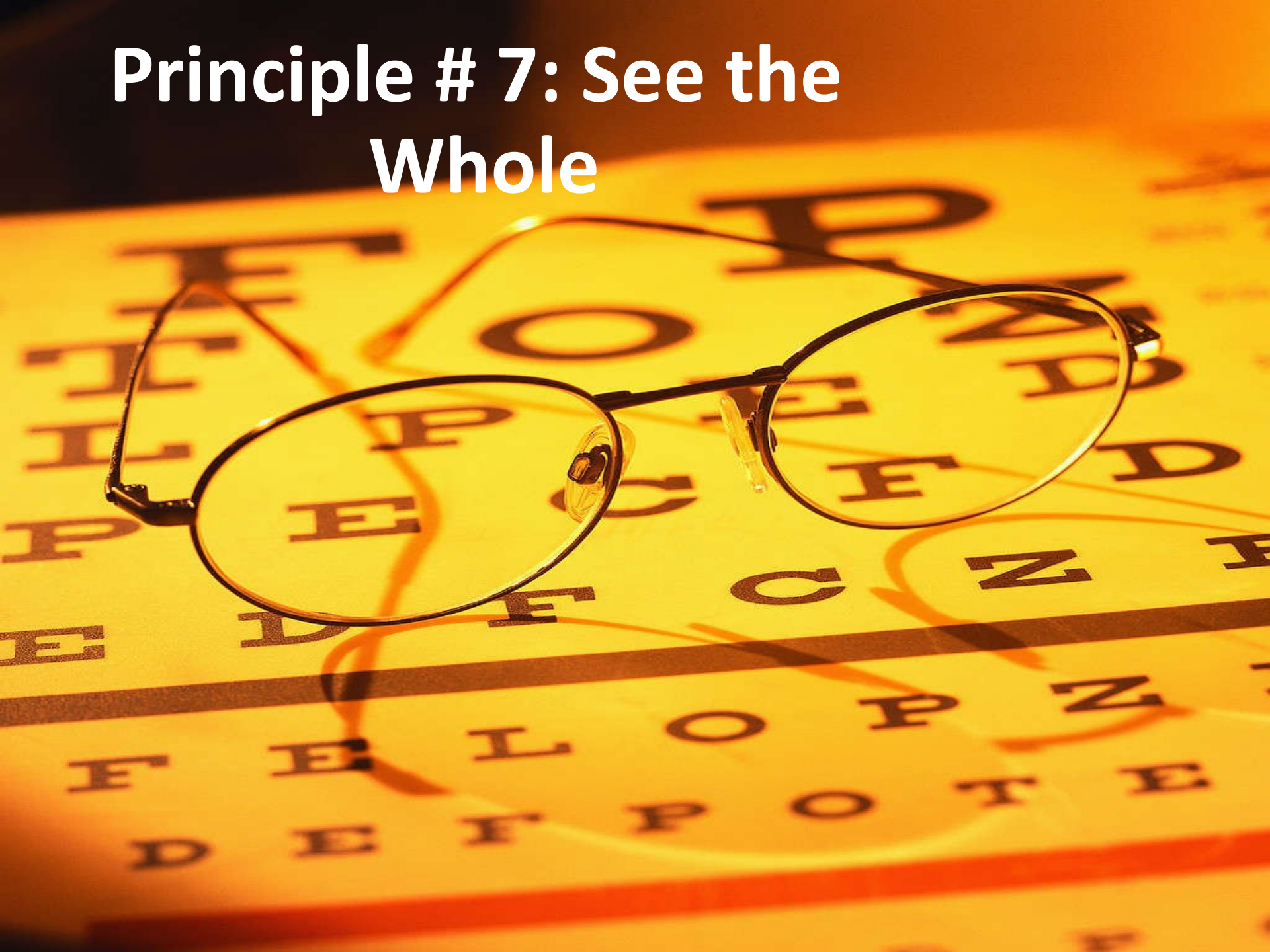
- On 5 large sheets of paper label them each with one of the following titles:

3. Simplicity
4. Clarity
5. Suitability for Use
6. No Repetition
7. No Extra Features

- Take one application and ask each team member to list anything within the current system that that doesn't meet these Lean Standards. Categorize them accordingly. Choose 2 or 3 of the highest priority items and figure out a way to make them meet the standards.

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 6

Principle # 7: See the Whole



The Seven Principles: See the Whole

Systems Thinking & System Dynamics

- A “system” is not just a sum of it’s parts, it’s a product of those interactions
- When systems start to break down, rigid more sequential rules are usually put in place
- A sequential/rigid process may cure one symptom, but not the root cause. It will become increasingly difficult to keep up with changing needs.

A Common Pattern

- **Limits of Growth:** even as one process produces a desired result, it creates a secondary constraint that eventually slows down the effect and limits growth.
- **The Theory of Constraints:** you can remove a constraint to growth in one place, but it will shift to another place. It’s an on going process.
- **Shifting the Burden:** addressing the symptoms, instead of the root cause.
- **Sub Optimization:** the more complex a system, the more you temptation it is to divide into parts

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 7

The Seven Principles: See the Whole

Ask the 5 whys!

- You have increasing defects on a project you ask:

Why #1:

A: New modules were added, that are causing new issues.

Why #2: Why did the new modules generate defects in other modules?

A: They were not tested

Why #3: Why where they not tested?

A: Developers where pressured to deliver before testing could occur

Why #4: Why was there so much pressure?

A: A Manager thought a hard deadline would work to motive the developers

Why #5: Why did they think this approach was necessary?

A: A manager was worried about late delivery and schedule overruns

- Now you can address the root cause: A manager doesn't grasp the burn down of the work and thinks hard deadlines motivate, instead of seeing the system converging and trusting in the progress of the team

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 7

The Seven Principles: See the Whole

Tool 21: Measurements

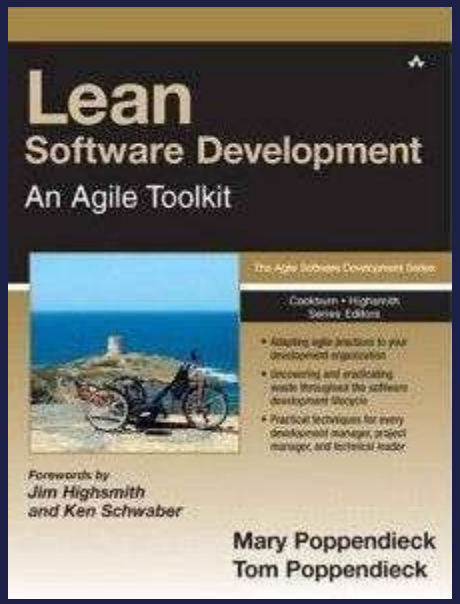
- When you are trying to measure performance, particularly of knowledge workers, you are asking for trouble
- People will often optimize the measurements that their performance is measured against (sub-optimization)
- Don't foster sub-optimized behavior! If you can't measure everything, don't try.
- A defect in code, isn't always the fault of one person. Pointing the blame towards a developer doesn't address the root cause of the system.
- Measurements should encourage optimizing the whole, and the team to collaborate to find better ways to do things.
- Modern culture bases success on individual performance metrics, this drives sub-optimizing behavior.

Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 7

The Seven Principles: See the Whole

Tool 22: Contracts

- Won't be covered in this training: for more information please read this book: Lean Software Development. ☺



Reference: M & T Poppendieck, Lean Software Development. 2003 : Chapter 7

Appendix

- This presentation was possible due to the amazing work done by Mary & Tom Poppendieck, in their book Lean Software Development. 2003
- All slide intro images were borrowed from Google image search and are not proprietary to this presentation
- Thank you!