



**CHALMERS**  
UNIVERSITY OF TECHNOLOGY



UNIVERSITY OF GOTHENBURG

# Lean Software Development

Agile Development Processes

Eric Knauss

# Administrative stuff



- Guest lecture: Currently being clarified.
  - May-19 on Large-Scale Agile by Daniel Borgentun (KnowIT, Volvo Trucks)
  - May-24 on Agility and Architecting by Patrizio Pelliccione
- Any other business?



## SPRINT 2

<http://www.flickr.com/photos/bnsd/15643117918>

Focus on Getting work done

# Course Objectives

Knowledge and understanding	Skills and ability	Judgement and approach
Compare agile and traditional softw. dev,	Forming a team organically	Explain: people/commun. centric dev.
Relate lean and agile development	Collaborate in small software dev. teams	Apply fact: people drive project success
Contrast different agile methodologies	Interact and show progress continuously	Describe: No single methodology fits all
Use the agile manifest and its accompanying principles	Develop SW using small and frequent iterations	Discuss: methodology needs to adopt to culture
Discuss what is different when leading an agile team	Use test-driven dev. and automated tests	Sprint 3
Sprint 2	Refactor a program/design	
	Be member of agile team	
	Incremental planning using user stories	

## Legend

Addressed

Open

Mainly in project

Focus today



# Lean History

- Toyota (1980s):  
"Lean Manufacturing", revolutionize the automotive industry
  - eliminate waste
  - streamline the value chain (even across enterprises)
  - produce on request (just in time), and
  - focus on the people who add value.
- Mary and Tom Poppendieck: transferred principles and practices from manufacturing to the software development



# Software Development as Manufacturing

	Manufacturing Mass Production	Waterfall Model
Hidden Cost	<ul style="list-style-type: none"><li>Transportation</li><li>Managing inventory and storage</li><li>Capital costs of inventory</li></ul>	<ul style="list-style-type: none"><li>Handoffs</li><li>Lots of open items; can lead to overwhelming the workforce</li><li>Cost of training people to build software</li></ul>
Risks	<ul style="list-style-type: none"><li>Building things you don't need because production goes on after needs go away</li><li>Inventory becoming obsolete Huge latency if an error occurs</li></ul>	<ul style="list-style-type: none"><li>Building things you don't need because requirements aren't clear or customers change their minds</li><li>Knowledge degrading quickly If a line is discontinued, all WIP wasted</li><li>Errors in requirements discovered late in the process</li><li>Errors in completed code discovered late in testing</li></ul>

# Lean Mindset

Related to, but  
not the same as  
agile mindset!

- Most errors are of a systemic nature  
→ Development system must be improved.
- You must respect your people in order to improve your system.
- Doing things too early causes waste.
  - Do things just before you need to do them
  - This is called “Just-In-Time,” or JIT.
- Focus on shortening time-to-market by removing delays in the development process
  - Using JIT methods to do this is more important than keeping everyone busy.

Lean-Agile Software Development: Achieving Enterprise Agility, by Alan Shalloway,  
Guy Beaver, and Jim Trott. Chapter 1.

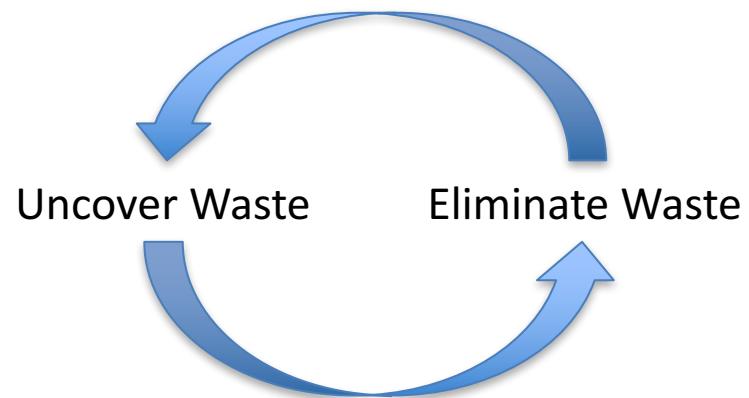
# LSD: 7 Principles, 22 Tools

- Eliminate Waste
  - Seeing Waste, Value Stream Mapping
- Amplify Learning
  - Feedback, Iterations, Synchronization, Set-Based Development
- Decide as Late as Possible
  - Options Thinking, The Last Responsible Moment, Making Decisions
- Deliver as Fast as Possible
  - Pull Systems, Queuing Theory, Cost of Delay
- Empower the Team
  - Self-Determination, Motivation, Leadership, Expertise
- Build Integrity In
  - Perceived Integrity, Conceptual Integrity, Refactoring, Testing
- See the Whole
  - Measurements, Contracts

Cristoph Steindl (IBM), Lean Software Development,  
2004 (Presentation)

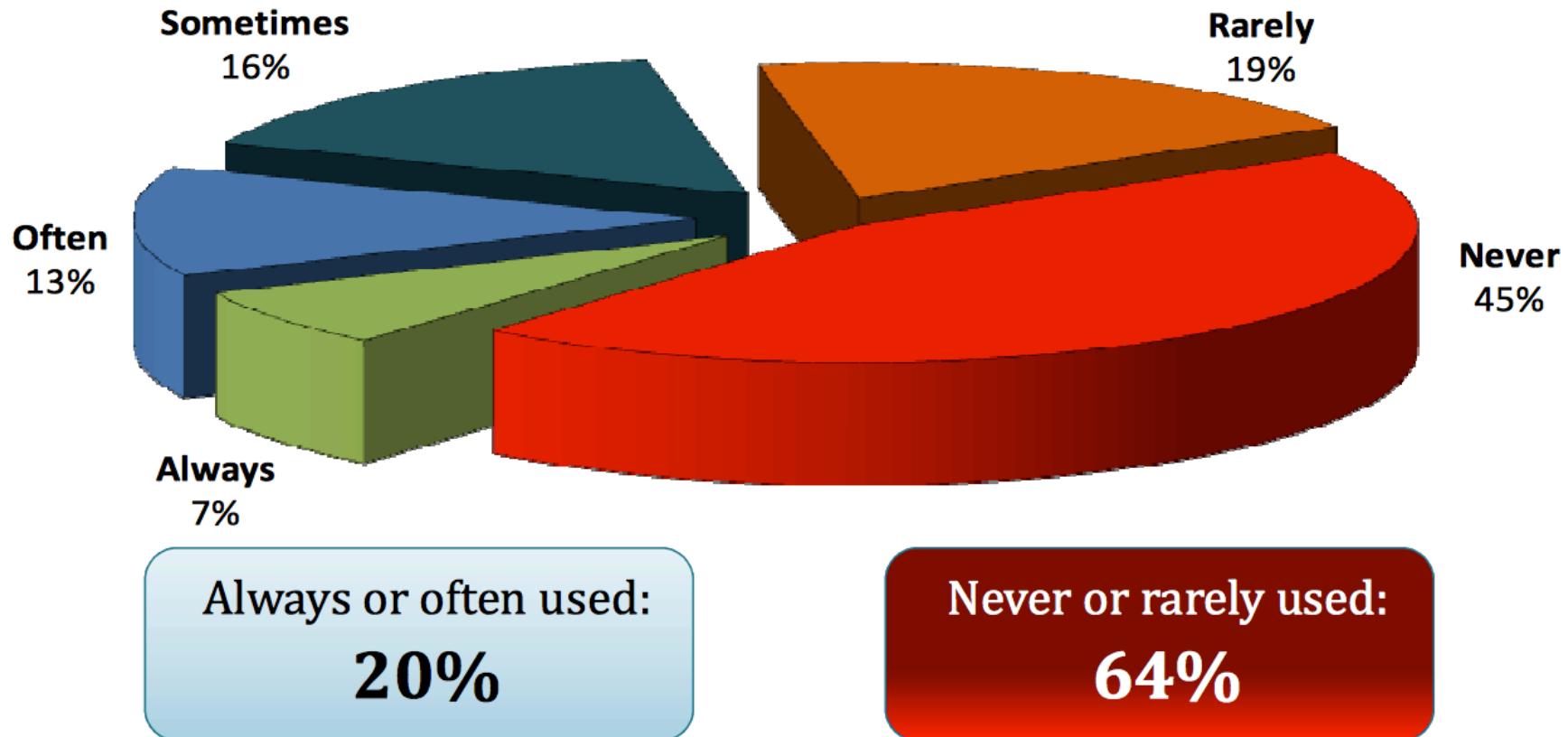
# Principle #1: Eliminate Waste

- ... does not mean to throw away all documentation
  - spend time only on what adds real customer value.
- Eliminating waste is the most fundamental lean principle.
  - First step to implementing lean development:  
**learning to see waste.**
  - Second step:  
**uncover** biggest sources of waste and **eliminate** them.



Cristoph Steindl (IBM), Lean Software Development,  
2004 (Presentation)

# Rates of Feature Usage in Software Projects



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

# #1: Eliminating Waste

## Tool #1: Seeing Waste

1. Inventory
  - Partially done work
2. Extra processing
  - Paperwork
3. Over production
  - Unused features
4. Transportation
  - Task switching
5. Waiting
  - For staffing,
  - For approval,
  - For review,
  - For excessive requirements,
  - For testing,
  - For deployment
6. Motion
  - Handovers, finding answers for questions
7. Defects
  - Waste produced by defect = impact x time

Cristoph Steindl (IBM), Lean Software Development,  
2004 (Presentation)

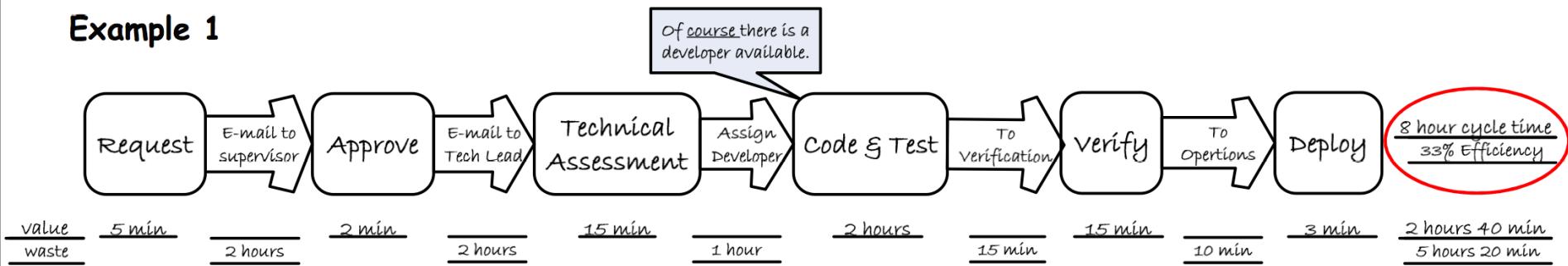
# #1: Eliminating Waste

## Tool #2: Value Stream Mapping

1. Pretend you are a customer request
  - Imagine yourself going through each step of your process
  - Don't ask people; walk around, look at data, find out yourself.
2. Go to the start (where a customer request comes into your organization)
3. Draw a chart (average customer request, from arrival to completion)
4. Work with people involved in each activity
  - Sketch all process steps necessary to handle request,
  - Capture average amount of time it spends in each step.
5. Draw Timeline (at the bottom of the map), showing how much time
  - Request spends in value-adding activities.
  - Request spends waiting or in non-value adding activities.

# Examples: Value-Stream Mapping

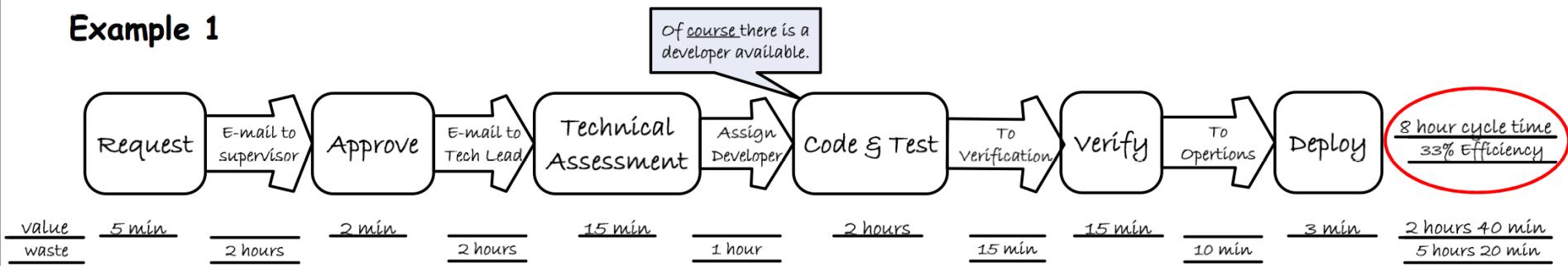
## Example 1



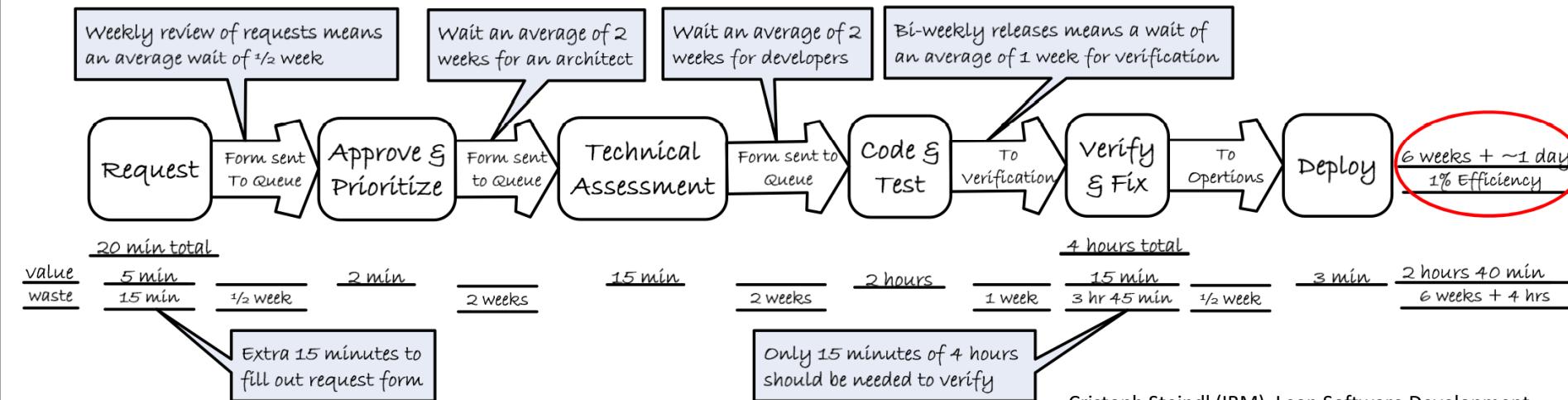
Cristoph Steindl (IBM), Lean Software Development,  
2004 (Presentation)

# Examples: Value-Stream Mapping

## Example 1

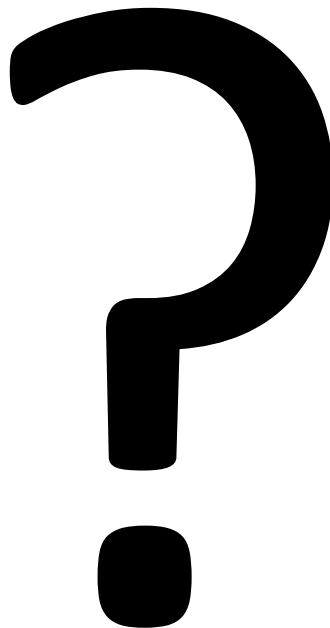


## Example 2



Cristoph Steindl (IBM), Lean Software Development, 2004 (Presentation)

# Group work



With your neighbors and  
about your agile projects:

- Discuss waste and value
- Discuss the value-stream (e.g. how long does it take to get Terese's or Magnus' requests during acceptance tests on their or on my phone?)

# #Principle 2: Amplify Learning

- Imagine a company with SW development challenges
- Usual reaction: Increase discipline / control
  - Specify requirements more completely
  - All agreements with customer are written
  - Changes are controlled more carefully
  - More tracing
  - ...
- As control theory predicts, this generally makes a bad situation worse

Cristoph Steindl (IBM), Lean Software Development,  
2004 (Presentation)

# #2: Amplify Learning

## Tool #3: Feedback

If a problem develops:

1. Make sure that all feedback loops are in place
2. Increase the frequency of feedback loops in problem areas

Instead of...	...focus on
Gather more requirements	

Cristoph Steindl (IBM), Lean Software Development,  
2004 (Presentation)

# #2: Amplify Learning

## Tool #3: Feedback

If a problem develops:

1. Make sure that all feedback loops are in place
2. Increase the frequency of feedback loops in problem areas

Instead of...	...focus on
Gather more requirements	Discuss UI prototypes
Accumulate defects	Test as soon as code is written
More documentation and detailed planning	Check out ideas by writing code
Study carefully which tool to use	Bring top 3 candidates in house and test them

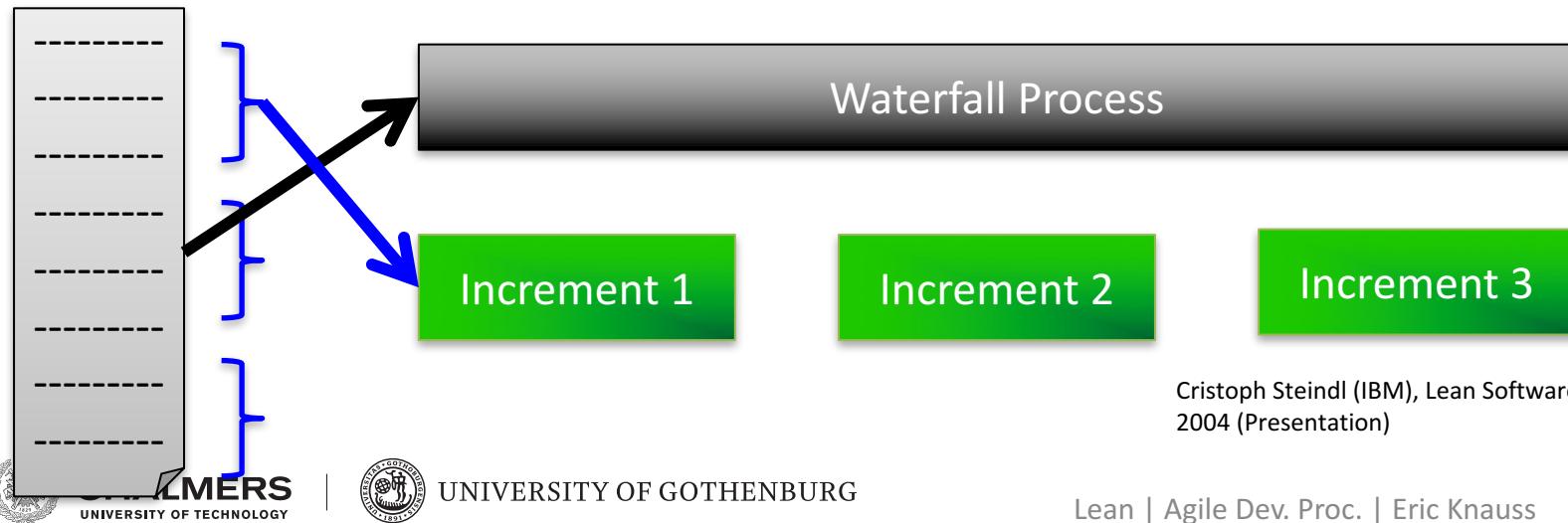
Cristoph Steindl (IBM), Lean Software Development,  
2004 (Presentation)

## #2: Amplify Learning

### Tool #4: Short iterations

#### Short iterations

- Increase control
- Help synchronizing developers and customer
- Force decisions to be made



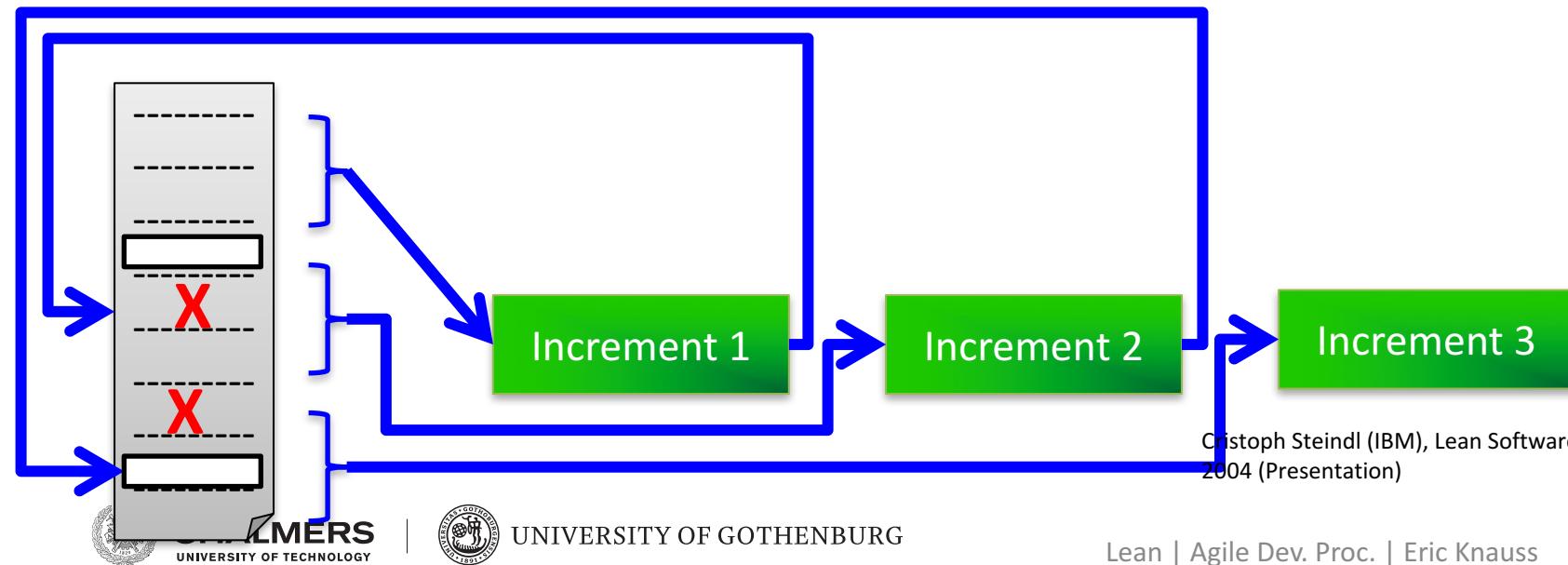
Cristoph Steindl (IBM), Lean Software Development,  
2004 (Presentation)

## #2: Amplify Learning

### Tool #4: Short iterations

#### Short iterations

- Increase control
- Help synchronizing developers and customer
- Force decisions to be made



## #2: Amplify Learning

### Tool #5: Synchronisation

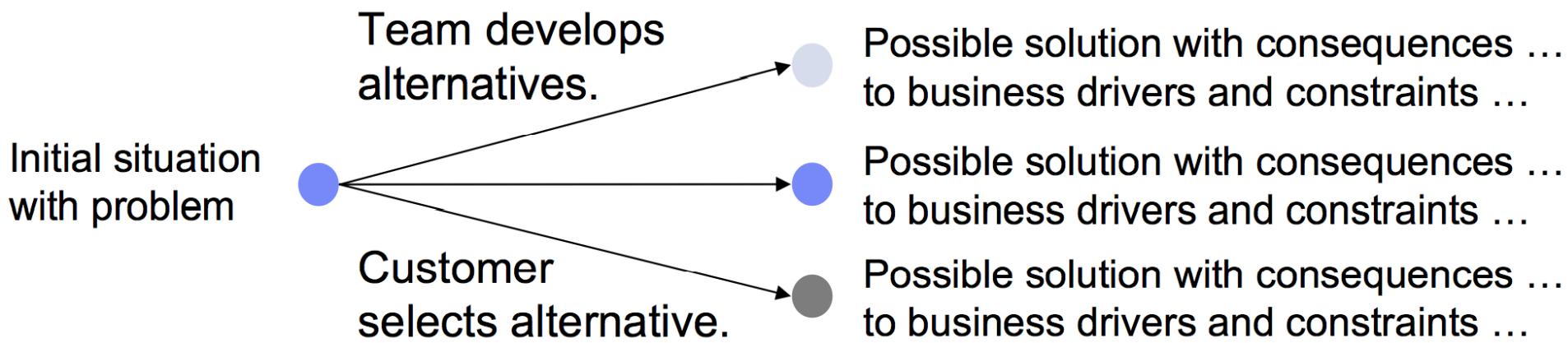
- Sync individuals that work on the same thing
  - Fundamental for every complex development process
- Daily within the team
  - Daily Scrum
  - Daily Build
  - Daily System Test
- Weekly within several teams
  - Push to master at least weekly
  - Weekly meetings (e.g. Scrum-of-Scrum)

Cristoph Steindl (IBM), Lean Software Development,  
2004 (Presentation)

## #2: Amplify Learning

### Tool #6: Set based development

- Communicate about constraints, not choices
  - Use less data to convey more information
  - Defer making choices until they have to be made



Cristoph Steindl (IBM), Lean Software Development,  
2004 (Presentation)

# Principle #3:

## Decide as late as possible

- ...does not mean to procrastinate,
  - but to keep your options open as long as practical,
  - but no longer
- Reason
  - Most common problem: Drill down to details too fast
  - In the presence of the risk to make big mistakes: survey the landscape and delay detailed decisions
- Wait: Is that not equal to sequential development (analyze requirements up-front)?
  - Sequential development implies depth-first
  - Breadth-first is far better to manage risk!

Cristoph Steindl (IBM), Lean Software Development,  
2004 (Presentation)

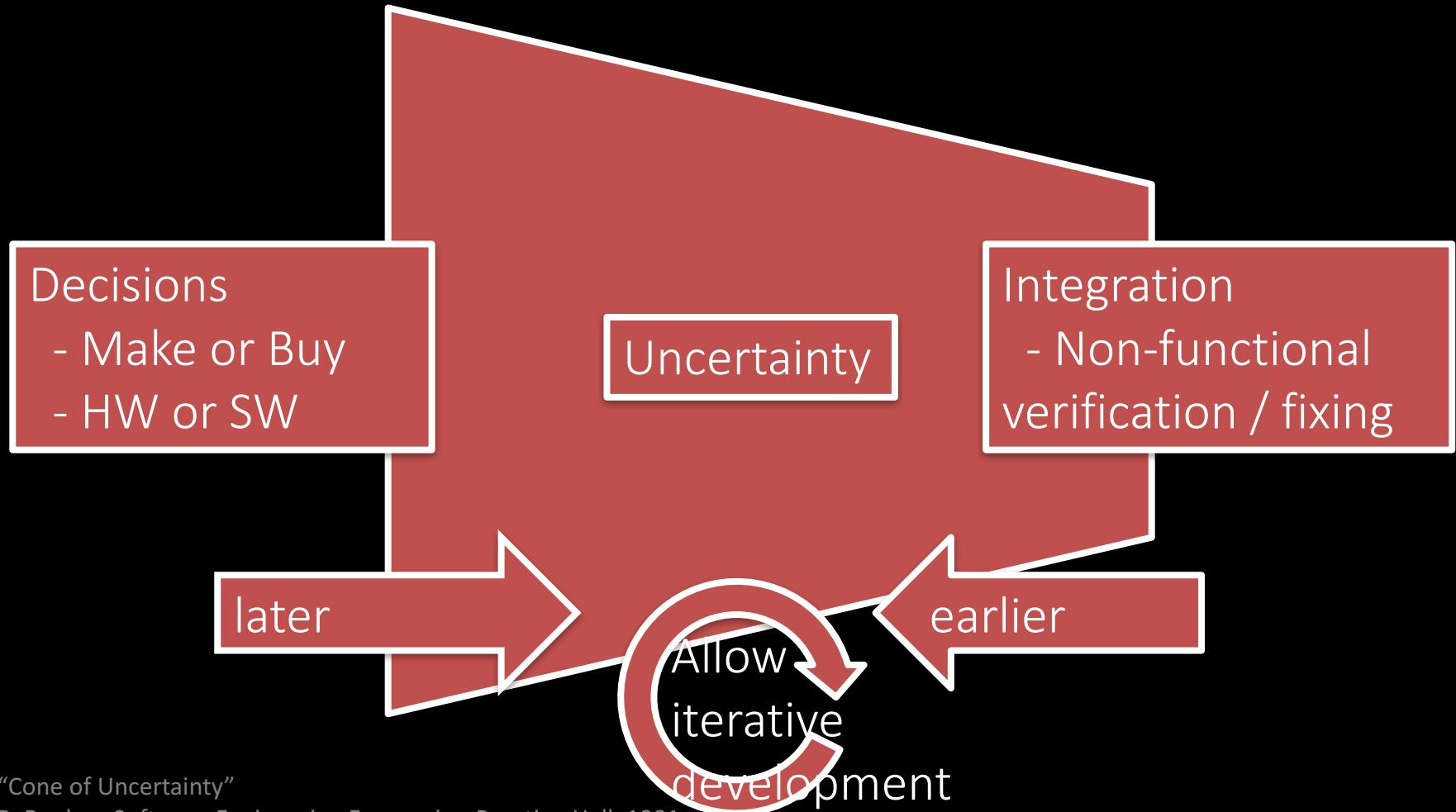
## #3: Decide as late as possible

### Tool #7: Options Thinking

- Common sense:
  - Resist making irrevocable decisions under uncertainty
  - Options allow fact-based decisions  
(based on learning, not speculation)
  - E.g. premature design commitment restricts learning, limits usefulness of the product, and increases the cost of change
- Plans and predictions are not bad, but avoid making irrevocable decisions based on speculation
  - Develop options, communicate them and decide together with the customer.
  - But: options are not free and it takes expertise to know which options to keep open.

Cristoph Steindl (IBM), Lean Software Development,  
2004 (Presentation)

# Example in Automotive Ecosystem



"Cone of Uncertainty"

B. Boehm. Software Engineering Economics. Prentice-Hall, 1981.

## #3: Decide as late as possible

### Tool #8: The Last Responsible Moment

- Last responsible moment = the moment at which failing to make a decision eliminates an important alternative
  - Concurrent development allows you to wait for that moment
- Share partially complete work
  - Avoid increasing length of the feedback loop and forcing irreversible decisions to be made sooner than necessary
  - Good work is a discovery process, done through short, repeated exploratory cycles.
- Develop a sense of
  - how to absorb changes
  - what is critically important in the domain
  - when decisions must be made

## #3: Decide as late as possible

### Tool #9: Making decisions



Intuitive decision making is more mature than rational decision making

- Usually leads to better decisions
- Rational decision making
  - Decompose problem, remove context, apply analytical techniques, and discuss process and results
  - Suffers from tunnel vision, intentionally ignoring the instincts of experienced people.
  - Unlikely to detect high-stakes mistakes.

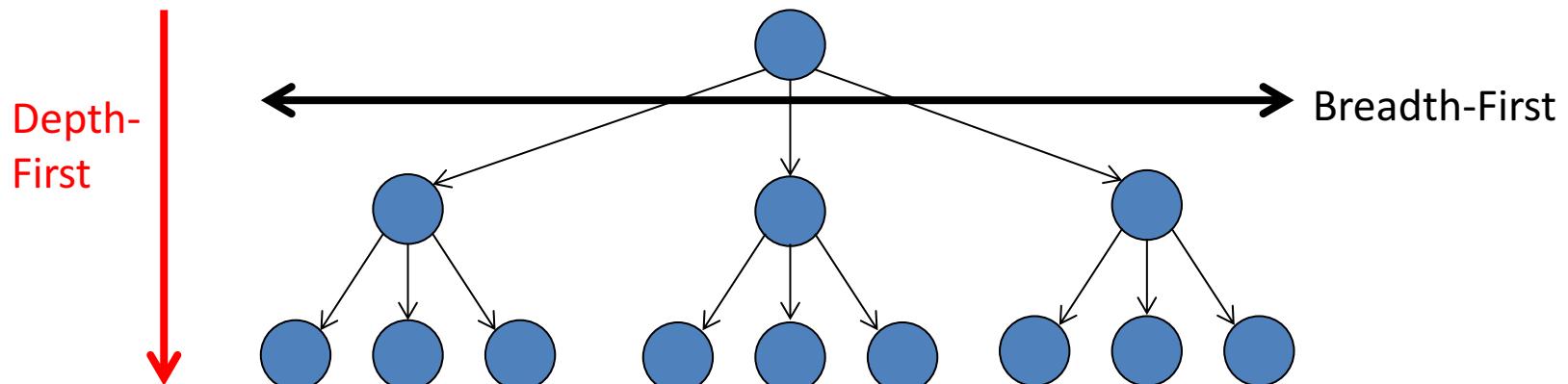
*Developing people with skill to make wise decision beats*

*developing decision-making processes that think for people*

# Principle #3:

## Decide as late as possible

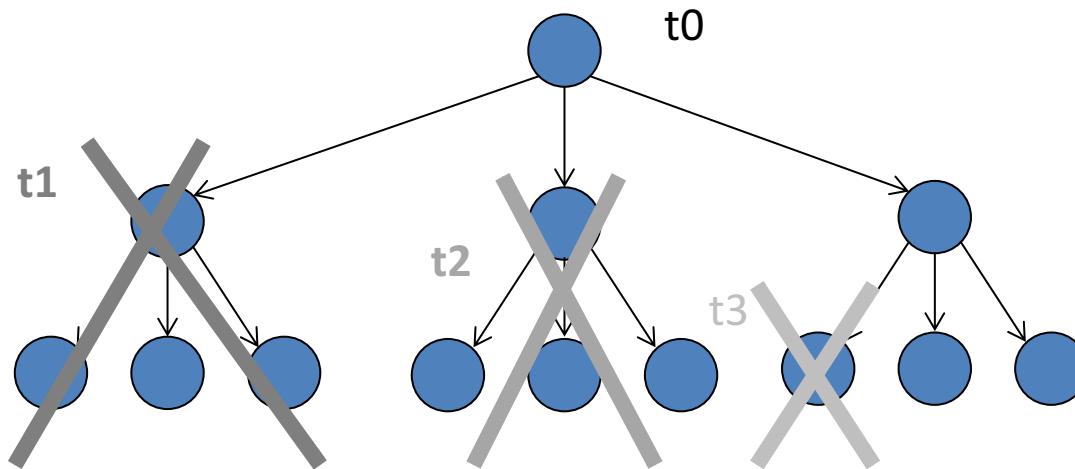
- Options Thinking, Last Responsible Moment, Making Decisions



# Principle #3:

## Decide as late as possible

- Options Thinking, Last Responsible Moment, Making Decisions



# Principle #4:

## Deliver as fast as possible

- ...does not mean to rush and do sloppy work, but to deliver value to customers as soon as they ask for it
- Customers like rapid delivery
  - Often translates to increased business flexibility
  - Deliver faster than their customers can change their minds
  - Fewer resources tied up in work-in progress.
- *Deliver as Fast as Possible* complements *Decide as Late as Possible*
  - The faster you can deliver, the longer you can delay decisions
  - E.g. if you can make a software change in a week, then you do not have to decide exactly what you are going to do until a week before the change is needed.
  - Allows to keep options open until uncertainty is reduced, which leads to more informed, fact-based decisions.

## #4: Deliver as fast as possible

### Tool #10: Pull-System

- Fundamental: Every team member knows what to do to make the most effective contribution to business
  - Option 1: tell them what to do (“command & control”)
  - Option 2: set things up so they can figure it out for themselves (“self-organization”).
  - In a fast-moving environment, only the second option works
- Information radiators / feedback devices:
  - One of the features of a pull system is visual control, or management by sight.
  - Everyone must be able to see what is going on, what needs to be done, what problems exist, what progress is being made.

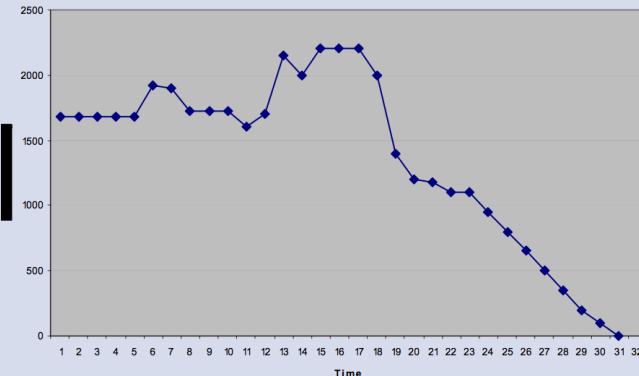


## #4: Deliver as fast as possible

### Tool #10: Pull-System

- Fundamental: Every team member knows what to do to make
  - Operations
  - Operations ("simplifying)
  - Inventory
- Information
  - One of the features of a pull system is visual control, or management by sight.
  - Everyone must be able to see what is going on, what needs to be done, what problems exist, what progress is being made.

Likely Backlog Trend



but it's not the backlog that matters. It's the backlog trend. "completing backlog items can find the right backlog only through visual backlog de



<http://www.developertesting.com/archives/2004/04/xtremefeedbackforSoftwareDevelopment.html>

## #4: Deliver as fast as possible

### Tool #11: Queuing Theory

- Cycle time
  - The fundamental measurement of a queue
  - When you are in a queue, you always want cycle time to be as short as possible.
- Steady rate of arrival
- Steady rate of service:
- Slack



## #4: Deliver as fast as possible

### Tool #11: Queuing Theory

- Cycle time
- Steady rate of arrival
  - When arrival of demand is spread out to match the capacity of the system, queues, and therefore cycle times, will be shortened
  - One way to control the rate of work arrival is to release small packages of work.
- Steady rate of service
- Slack

## #4: Deliver as fast as possible

### Tool #11: Queuing Theory

- Cycle time
- Steady rate of arrival
- Steady rate of service
  - The easiest way to remove variability in the processing time is to increase the number of servers that process work in a single queue.
  - Small work packages will allow parallel processing of the small jobs by multiple teams so that if one is stalled by a problem, the rest of the project can proceed without delay.
- Slack

## #4: Deliver as fast as possible

### Tool #11: Queuing Theory

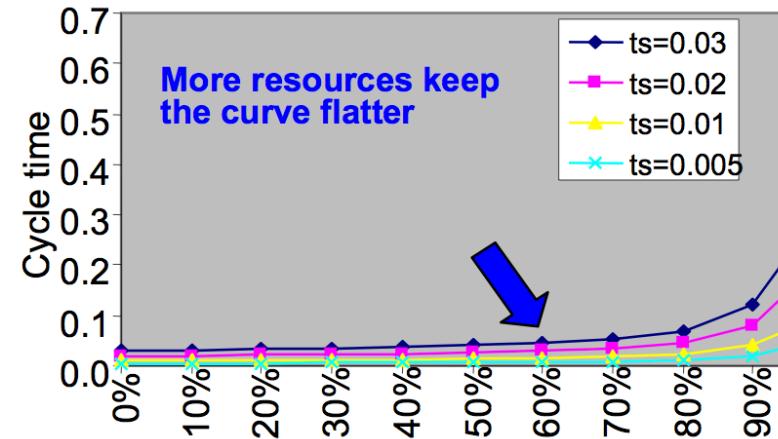
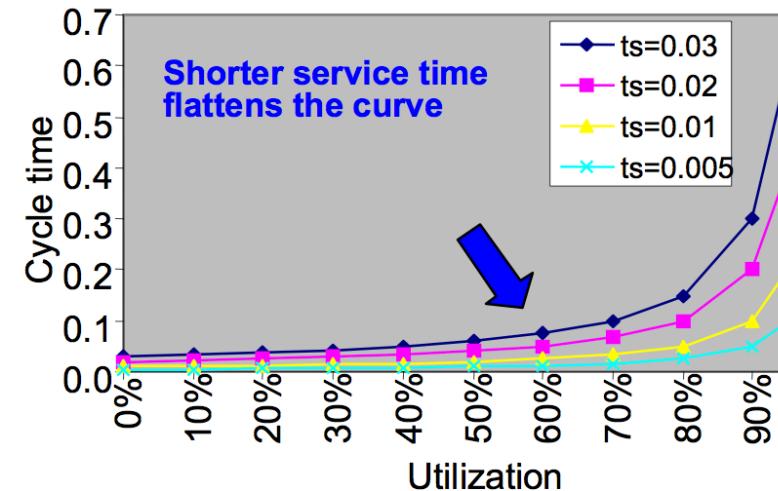
- Cycle time
- Steady rate of arrival
- Steady rate of service
- Slack
  - Short cycle times are not possible if resources are overloaded.
  - Full utilization provides no value to the overall value stream; in fact, it usually does more harm than good.



## #4: Deliver as fast as possible

### Tool #11: Queuing Theory

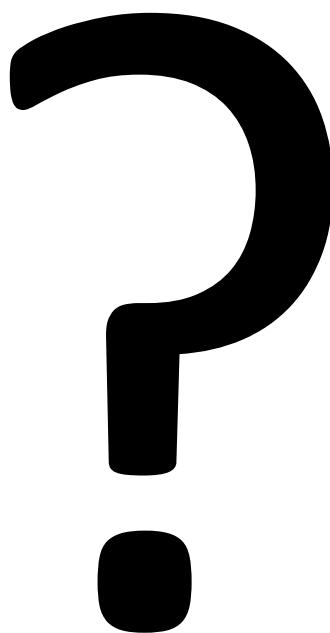
- Cycle time
- Steady rate of arrival
- Steady rate of service
- Slack



# How Queues work

- To increase throughput: look for the current bottleneck that is slowing things down and fix it
  - Do not increase the utilization of non-bottleneck areas. Don't keep piling up work that can't be used immediately.
- As variability (in arrival time or processing time) increases, cycle time will increase.
- As batch size increases, variability in arrival and processing time increases, and therefore cycle time will increase.
- As utilization increases, cycle time will increase nonlinearly.
- Continuous flow requires a reduction in variability.
- Variability may be reduced by an even arrival of demand, small batches, an even rate of processing, and parallel processing.
- Decreasing variability early in the process has larger impact than decreasing variability late in the process.

# Group work



With your neighbors and  
about your agile projects:

- How can you apply queuing theory in your projects?



## #4: Deliver as fast as possible

### Tool #12: Cost of delay

- Create a simple economic model (get good estimates from Marketing)
  - Show how differences in revenue and market share affect profits
  - If delay means
    - “loss of early high pricing” or
    - “long-term loss of market share”,cost of delay can be very high
- Keep the model simple
  - Make sure everyone understands and buys into the economic model
- Economic models may help to
  - Justify cost of reducing cycle time, eliminating bottlenecks, and purchasing tools that increase speed
  - Drive development decisions
  - Empower the team

# Principle #5:

## Empower the team

- ...does not mean to abandon leadership, but to let the people who add value use their full potential.
- Lean thinking = believing that frontline workers should determine and continually improve the way they do their jobs.
- Discipline is important, but experimentation and feedback are more effective than trying to get things right the first time.
- Empowerment is a critical motivating factor:
  - Move decisions to the lowest possible level in an organization while developing the capacity of those people to make decisions wisely.
- In a lean organization, people who add value are the center of organizational energy.
  - Frontline workers have process design authority and decision-making responsibility; they are the focus of resources, information and training.

Cristoph Steindl (IBM), Lean Software Development,  
2004 (Presentation)

# Principle #5:

## Empower the team

- ...does not mean to abandon leadership, but to let the people who add value use their full potential.
- Lean thinking = believing that frontline workers should determine and continually improve the way they do their jobs.
- Discipline is important, but experimentation and feedback are more effective than trying to get things right the first time.
- Empowerment is a critical motivating factor:
  - Move decisions to the lowest possible level in an organization while developing the capacity of those people to make decisions wisely.
- In a lean organization, people who add value are the center of organizational energy.
  - Frontline workers have process design authority and decision-making responsibility; they are the focus of resources, information and training.

Cristoph Steindl (IBM), Lean Software Development,  
2004 (Presentation)

# Principle #5:

## Empower the team

- **Tool #13:** Self-Determination



*"Here comes Edward Bear now,  
down the stairs behind Christopher  
Robin. Bump! Bump! Bump! on the  
back of his head. It is, as far as he  
knows, the only way of coming  
down stairs. He is sure that there  
must be a better way, if only he  
could stop bumping for a moment  
to think of it."*

**Winnie -The Pooh**

**A.A. Milne, 1926**

Cristoph Steindl (IBM), Lean Software Development,  
2004 (Presentation)

# Principle #5:

## Empower the team

- **Tool #13:** Self-Determination
- **Tool #14:** Motivation

Vision statements from William McKnight (leader of 3M from 1930s through the 1950s):

- “Hire good people, and leave them alone.”
  - “If you put fences around people, you get sheep. Give people the room they need.”
  - “Encourage, don't itpick. Let people run with an idea.”
  - “Give it a try – and quick!”
- 
- **Tool #15:** Leadership
    - Not management!

Cristoph Steindl (IBM), Lean Software Development,  
2004 (Presentation)

# Principle #5:

## Empower the team

- **Tool #13:** Self-Determination
- **Tool #14:** Motivation
- **Tool #15:** Leadership
  - Not management!

Managers	Leaders
<p>Cope with Complexity</p> <ul style="list-style-type: none"><li>• Plan and Budget</li><li>• Organize and Staff</li><li>• Track and Control</li></ul>	<p>Cope with Change</p> <ul style="list-style-type: none"><li>• Set Direction</li><li>• Align People</li><li>• Enable Motivation</li></ul>

Cristoph Steindl (IBM), Lean Software Development,  
2004 (Presentation)

# Principle #5:

## Empower the team

- **Tool #13:** Self-Determination
- **Tool #14:** Motivation
- **Tool #15:** Leadership
- **Tool #16:** Expertise

# Principle #6:

## Build Integrity in

- ...does not mean big, upfront design, but don't try to tack on integrity after the fact, build it in.
- Kim Clark (Harvard Business School): some companies consistently develop superior products.
  - Key differentiator: the products had integrity.
    - External (perceived) integrity: the totality of the product achieves a balance of function, usability, reliability, and economy that delights customers.
    - Internal (conceptual) integrity: the system's central concepts work together as a smooth, cohesive whole.
- The measure of perceived integrity is roughly equivalent to market share
- Conceptual integrity is a prerequisite for perceived integrity

The way to build a system with high perceived and conceptual integrity is to have excellent information flows both from customer to development team and between the upstream and downstream processes of the development team.

Cristoph Steindl (IBM), Lean Software Development,  
2004 (Presentation)

## Principle #6: Build integrity in

### Tool #17: Perceived integrity

*“Companies that consistently achieve perceived integrity have a way of constantly keeping **customer values in front of the technical people** making detailed design decisions. Chief engineers have added to their engineering and leadership skills the ability to understand the target customer base and create a vision.”*

*“Customers will know a good design when they see it, but they can't envision it beforehand.*

*To make matters worse, as their circumstances change, so will customers' perception of system integrity. ”*

Cristoph Steindl (IBM), Lean Software Development,  
2004 (Presentation)

# Principle #6: Build integrity in

## Tool #18: Conceptual integrity

- Use integrated problem solving to assure excellent technical information flow
  - Understanding the problem and solving the problem at the same time, not sequentially.
  - Preliminary information is released early; information flow is not delayed until complete information is available.
  - Information is transmitted frequently in small batches, not all at once in a large batch.
  - Information flows in two directions, not just one.
  - The preferred media for transmitting information is face-to-face communication as opposed to documents.

Cristoph Steindl (IBM), Lean Software Development,  
2004 (Presentation)

## Principle #6: Build integrity in

Tool #19: Refactoring

Tool #20: Testing

- If there is not enough time, move resources from writing requirements to writing customer tests

# Principle #7: See the Whole

This area	Must attend this work
Business	<ul style="list-style-type: none"><li>• Continuously prioritize and decompose incremental needs across the organization</li><li>• Manage a portfolio of business needs</li><li>• Do release planning</li></ul>
Management	<ul style="list-style-type: none"><li>• Organize cross-functional teams that can deliver incremental, end-to-end features</li><li>• Manage the value stream</li><li>• Bring visibility to impediments</li></ul>
Delivery team	<ul style="list-style-type: none"><li>• Work together, every day, and deliver fully tested and integrated code</li><li>• Learn how to deliver business needs incrementally</li><li>• Become proficient at acceptance test-driven development and refactoring</li></ul>

Lean-Agile Software Development: Achieving Enterprise Agility, by Alan Shalloway,  
Guy Beaver, and Jim Trott. Chapter 1.

# Principle #7: See the whole

## Tool **#21**: Measurement

- Avoid local sub-optimization
  - Try to give the whole picture
  - Do not measure individual performance, but give information
  - Make clear how measurements relate to business goals

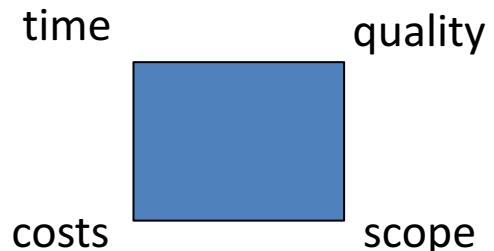
# Principle #7: See the whole

## Tool #21: Measurement



# Principle #7: See the whole

## Tool #22: Contracts



- Remember: up to 45% of the features delivered might never be used (according to a study of the Standish Group).
- Barry Boehm and Philip Papaccio (1988): the best way to develop low-cost, high-quality software is to write less code.
- Rigid control of scope tends to expand, not reduce, the scope. Save money overall by collaborating with the customer by using some form of optional scope contract.

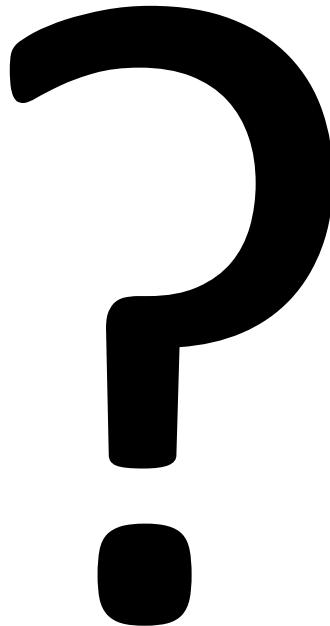
Cristoph Steindl (IBM), Lean Software Development,  
2004 (Presentation)

# LSD: 7 Principles, 22 Tools

- Eliminate Waste
  - Seeing Waste, Value Stream Mapping
- Amplify Learning
  - Feedback, Iterations, Synchronization, Set-Based Development
- Decide as Late as Possible
  - Options Thinking, The Last Responsible Moment, Making Decisions
- Deliver as Fast as Possible
  - Pull Systems, Queuing Theory, Cost of Delay
- Empower the Team
  - Self-Determination, Motivation, Leadership, Expertise
- Build Integrity In
  - Perceived Integrity, Conceptual Integrity, Refactoring, Testing
- See the Whole
  - Measurements, Contracts

Cristoph Steindl (IBM), Lean Software Development,  
2004 (Presentation)

# Group work



With your neighbors:

- Give an example of a software development scenario that is suitable for lean software development and one example that is more suitable for agile software development (e.g. XP or Scrum).
- Discuss similarities and differences between agile software development and lean software development.

# Agile vs. Lean



- Agile tends to emphasize communication at the local level
  - Lean advocates a holistic view of the delivery chain
- 
- XP: Decide now, change later
  - Lean: Defer commitment
- 
- Learning goal today:
    - Relate lean and agile
    - Contrast different agile methodologies
      - Agile has to do a lot with culture and spirit

Lean-Agile Software Development: Achieving Enterprise Agility, by Alan Shalloway,  
Guy Beaver, and Jim Trott. Chapter 1.

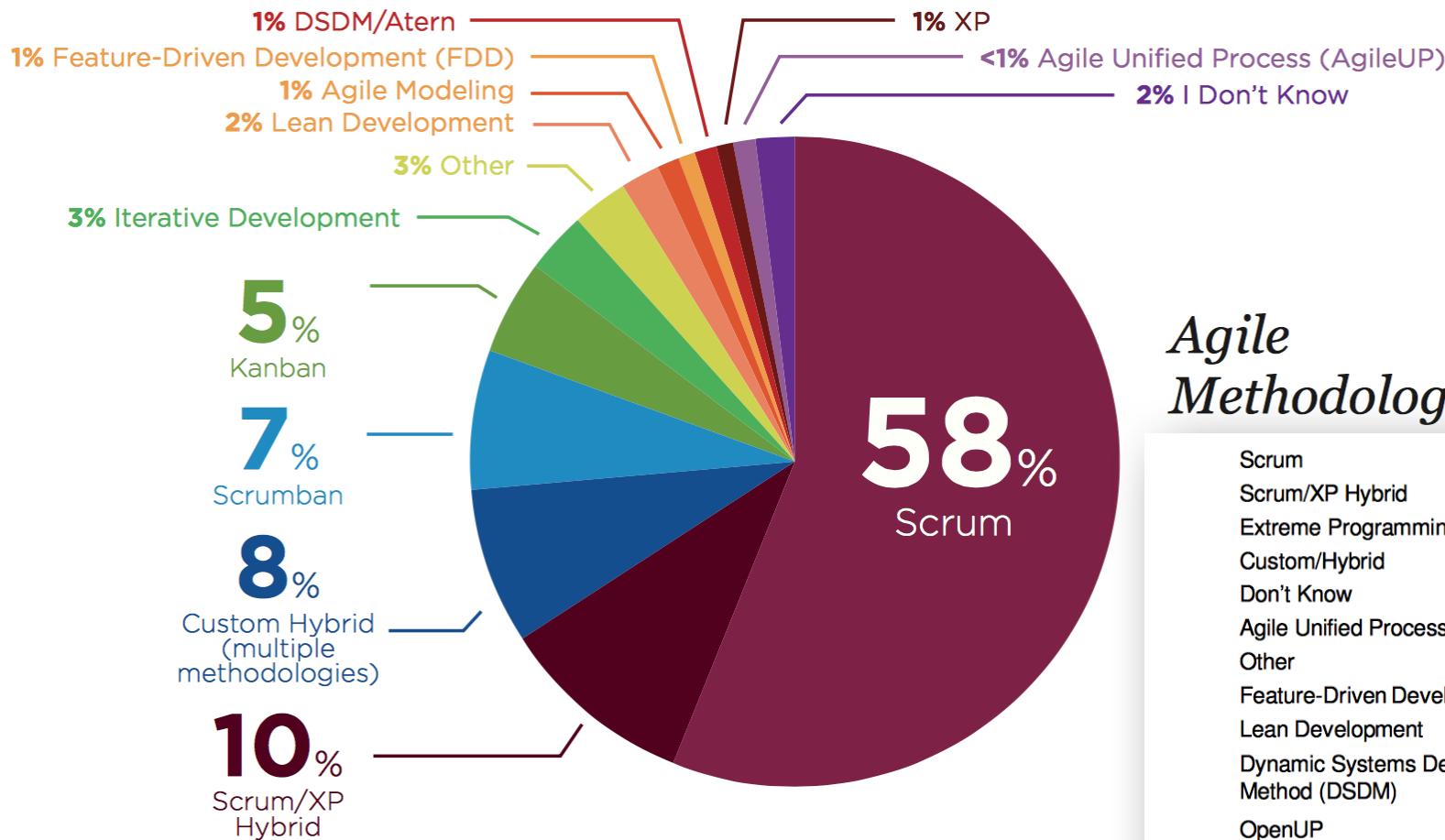
# Further reading

- Yasuhiro Monden (1998), *Toyota Production System, An Integrated Approach to Just-In-Time*, Third edition, Norcross, GA: Engineering & Management Press
- Mary Poppendieck, Tom Poppendieck (2003), "Lean Software Development: An Agile Toolkit", Addison-Wesley Professional

# AGILE METHODS AND PRACTICES

2015 – VersionOne

<http://info.versionone.com/state-of-agile-report-thank-you.html>



## *Agile Methodologies Used*

Scrum	49.1%
Scrum/XP Hybrid	22.3%
Extreme Programming (XP)	8.0%
Custom/Hybrid	5.3%
Don't Know	3.7%
Agile Unified Process (AgileUP)	2.2%
Other	2.2%
Feature-Driven Development (FDD)	2.1%
Lean Development	1.9%
Dynamic Systems Development Method (DSDM)	1.4%
OpenUP	0.6%
Agile Modeling	0.6%
Crystal	0.5%