

Software Engineering methodology

Lecture 6, 17.2.2014

Kari Systä

About assignment

- Start work now!
- Sprint meeting scheduling comes to IDLE today or tomorrow
- Deadline on Friday
- Be responsive to other members
 - Assistants can give permission for teams to kick-out passive members.

Initial content of lectures

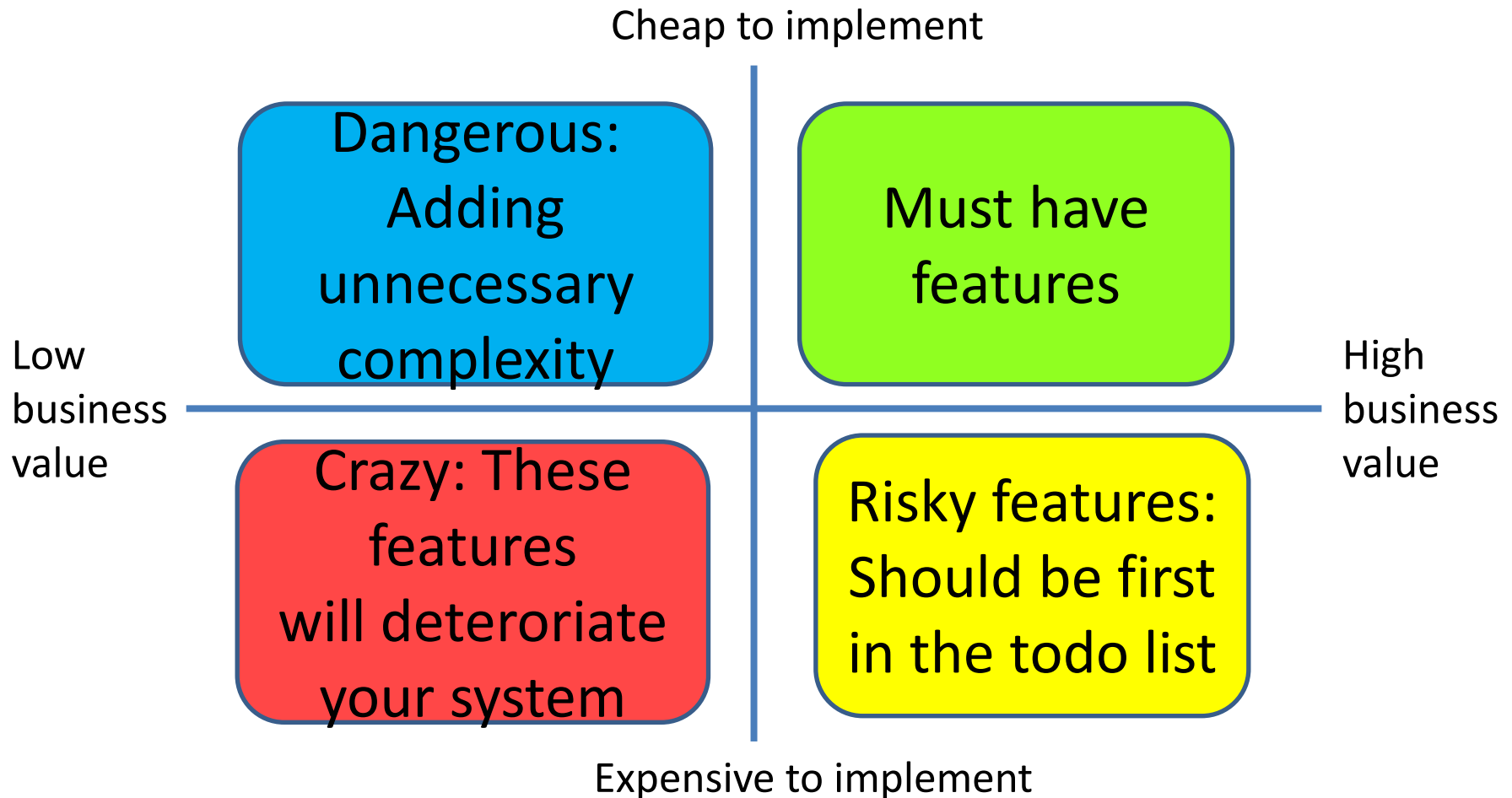
- Introduction
- Life-cycle models, their background
- Project management, product management, project planning – in general management aspects
- Scrum in details
- Requirement elicitation, requirement management, requirements prioritization
- New trends: Lean, Kanban, Customer Development and DevOps details
- Review practices, testing and quality assurance (TIE-21200 will go deeper)
- Version management, configuration management, continuous integration
- Architecture issues, role of architect, architectural quality attributes, product families, (TIE-21300 will go deeper)
- “Quality systems” and process improvement
- Embedded and real-time systems (other courses will go deeper)
- Safety-critical and dependable systems
- Effort estimation
- Software business, software start-ups
- Recap

Today's lecture

- Recap and summary of previous lecture
- Principles of Lean
- Kanban
- DevOps
- Lean Startup
- Continuous Deployment

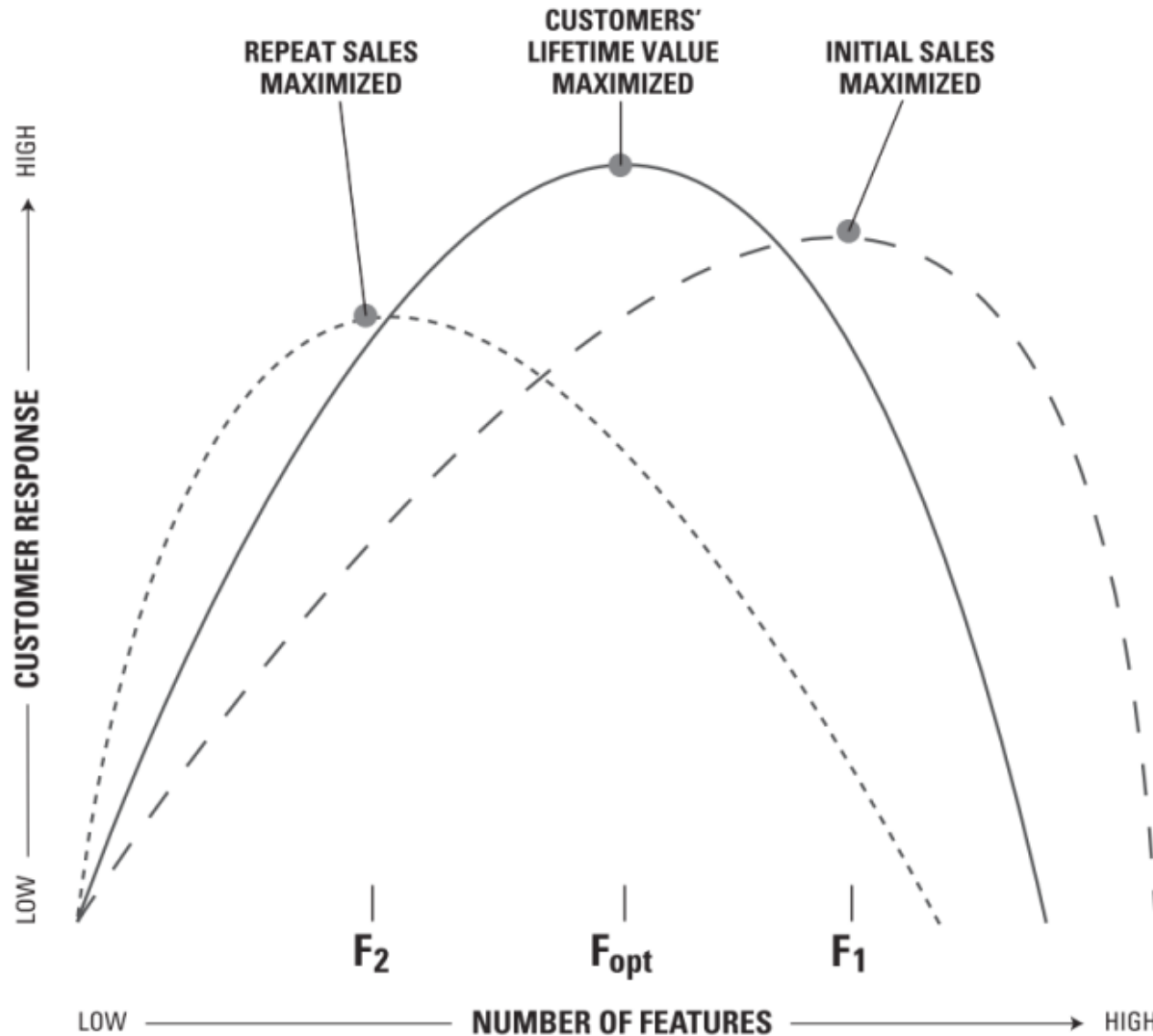
Couple of points about requirements

(wording by prof Pekka Abrahamsson, Bolzano)

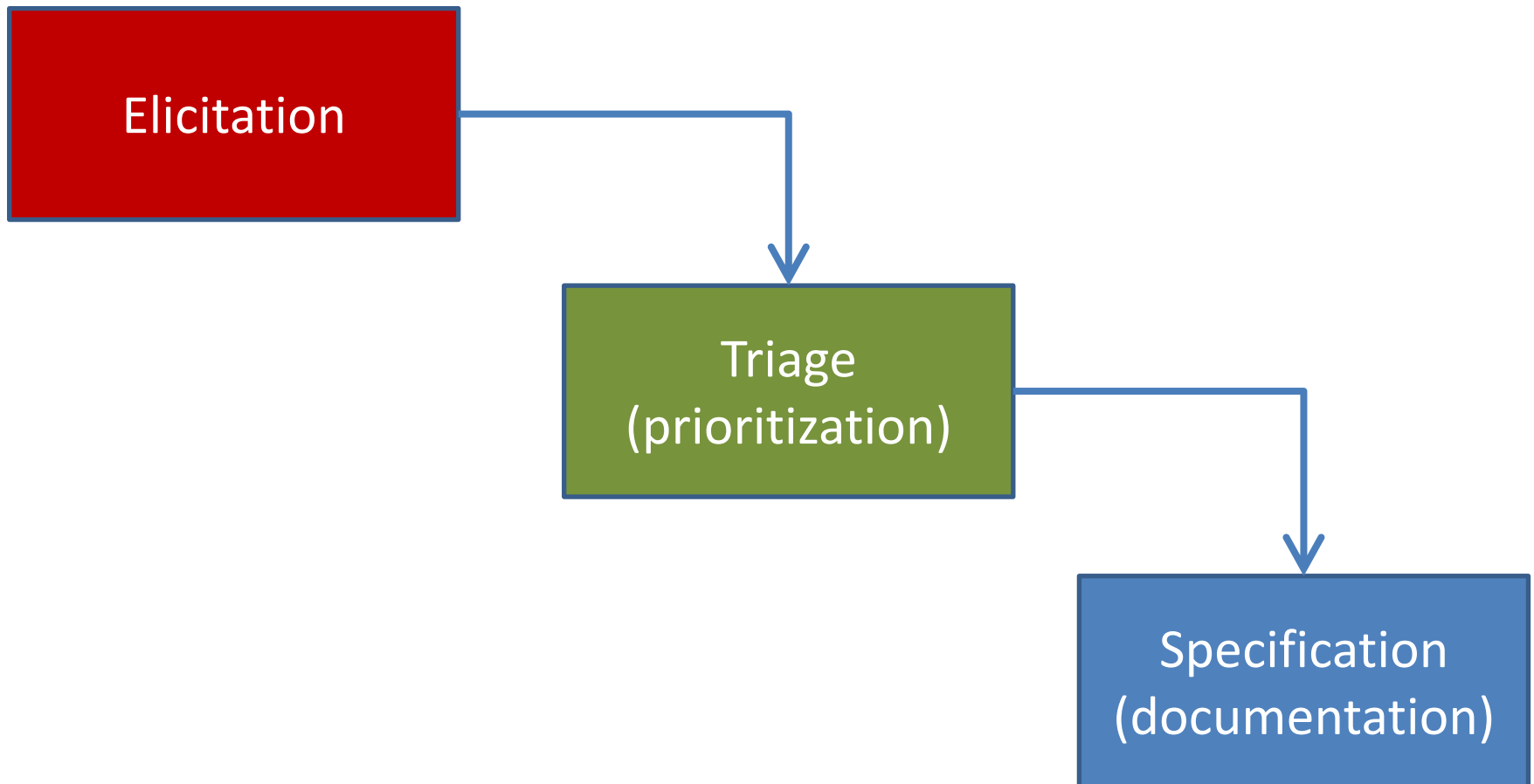


About value of features

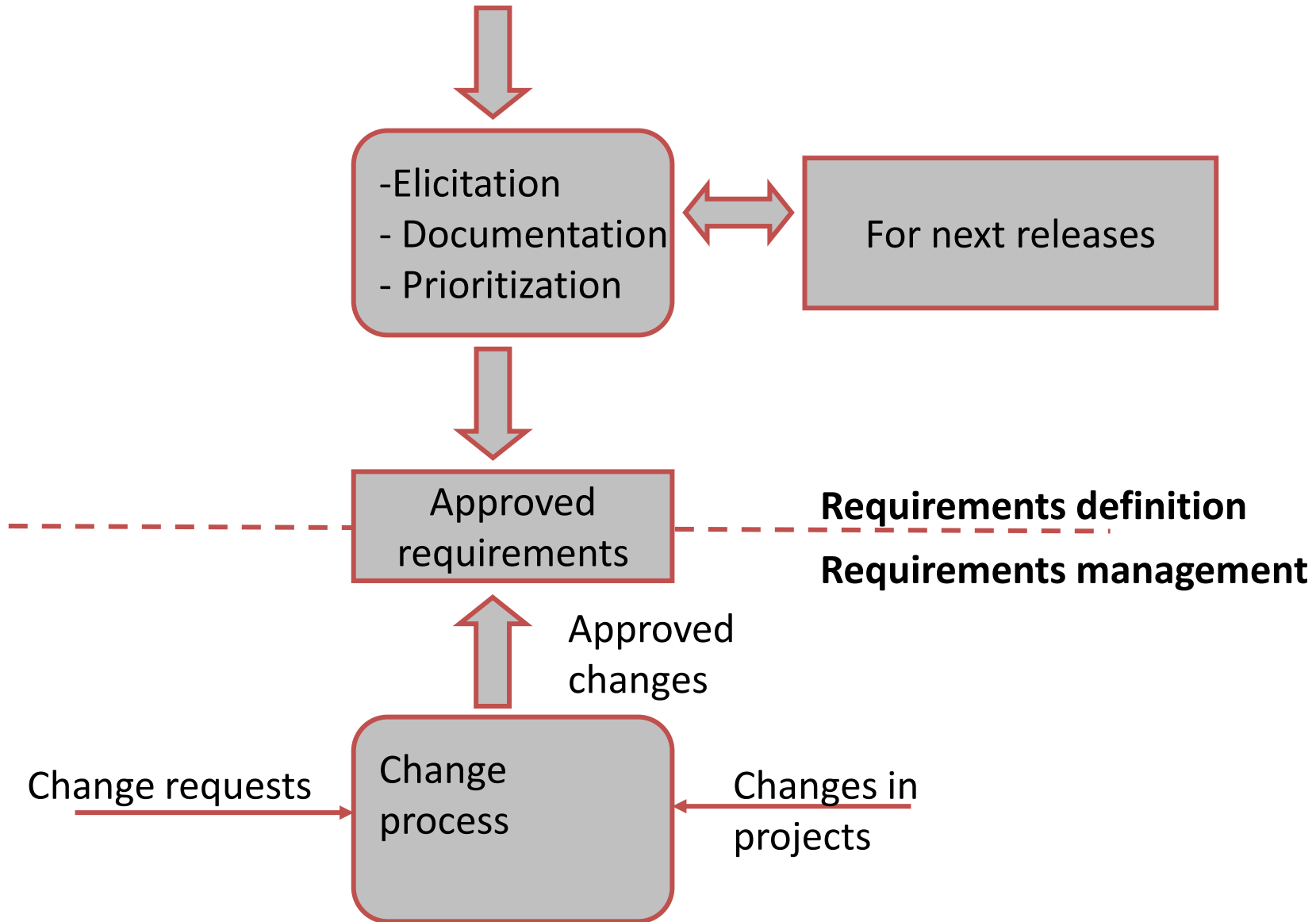
(Rust R.T., Thompson D.V., and Hamilton R.W. Quelle: Defeating Feature Fatigue in Make Sure All Your Products are Profitable, 2nd edition, s. 38-48, Harward Business Review, February 2006.)



Requirements management

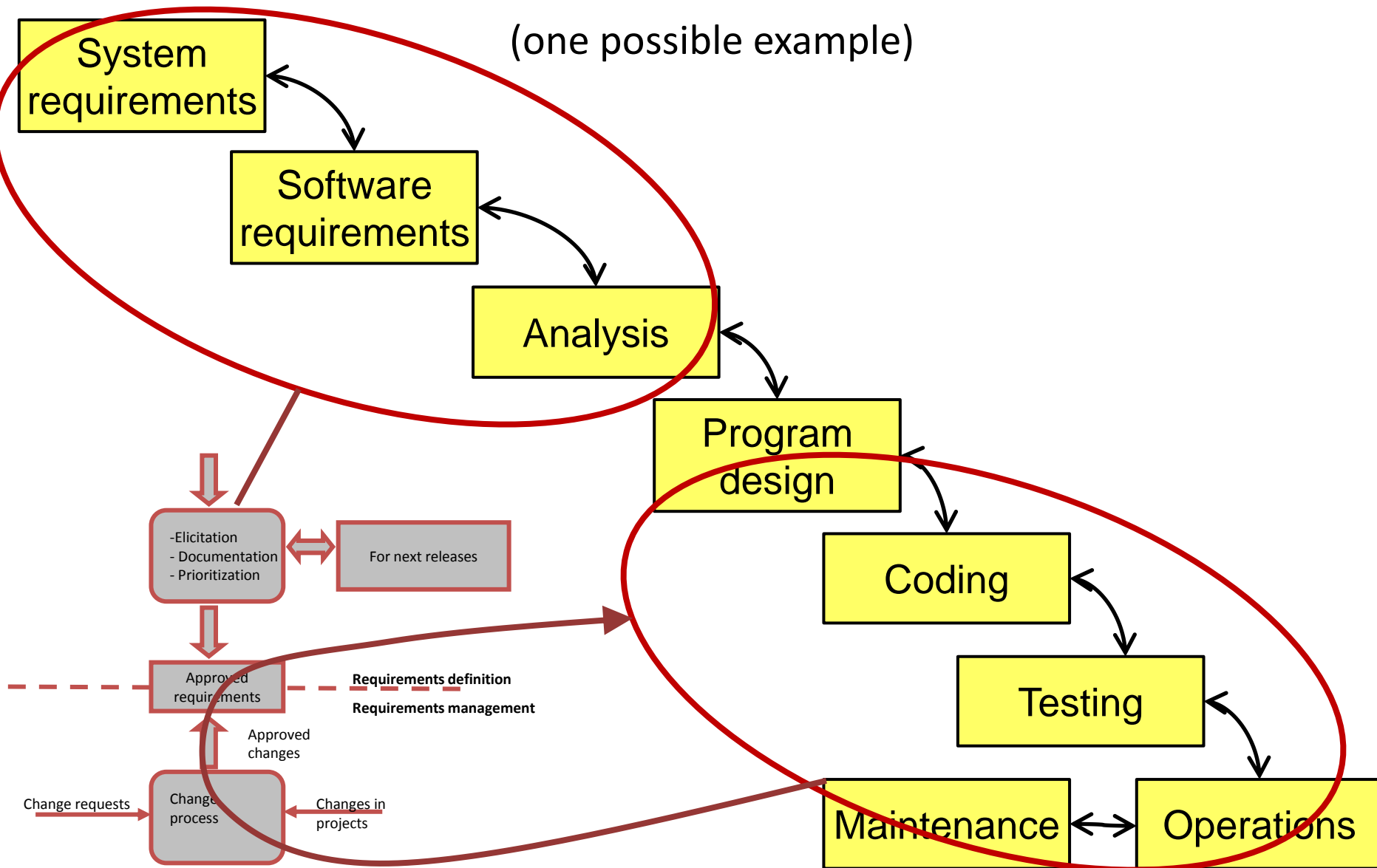


Requirements definition vs management



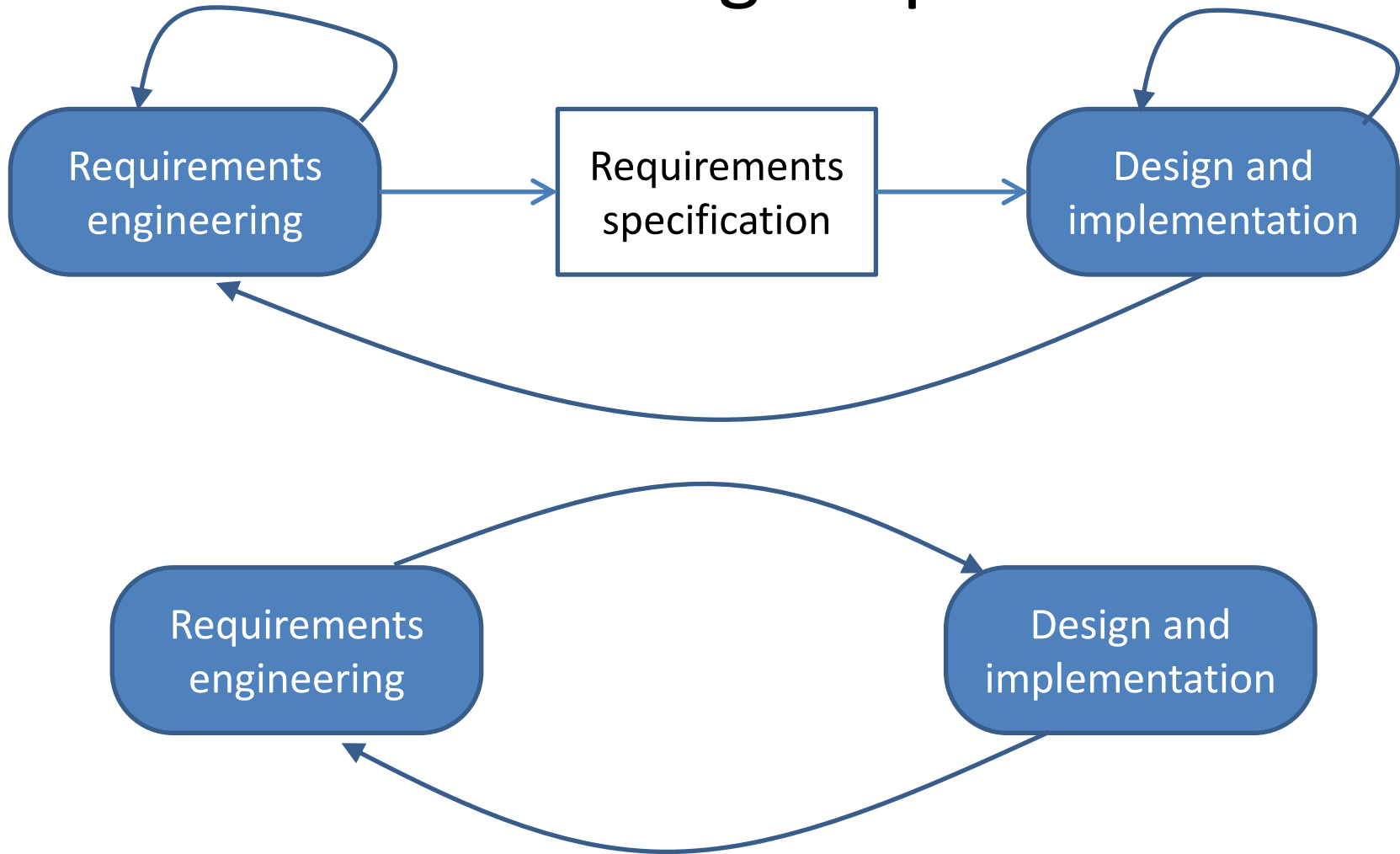
Documentation is a crucial part of waterfall

(one possible example)

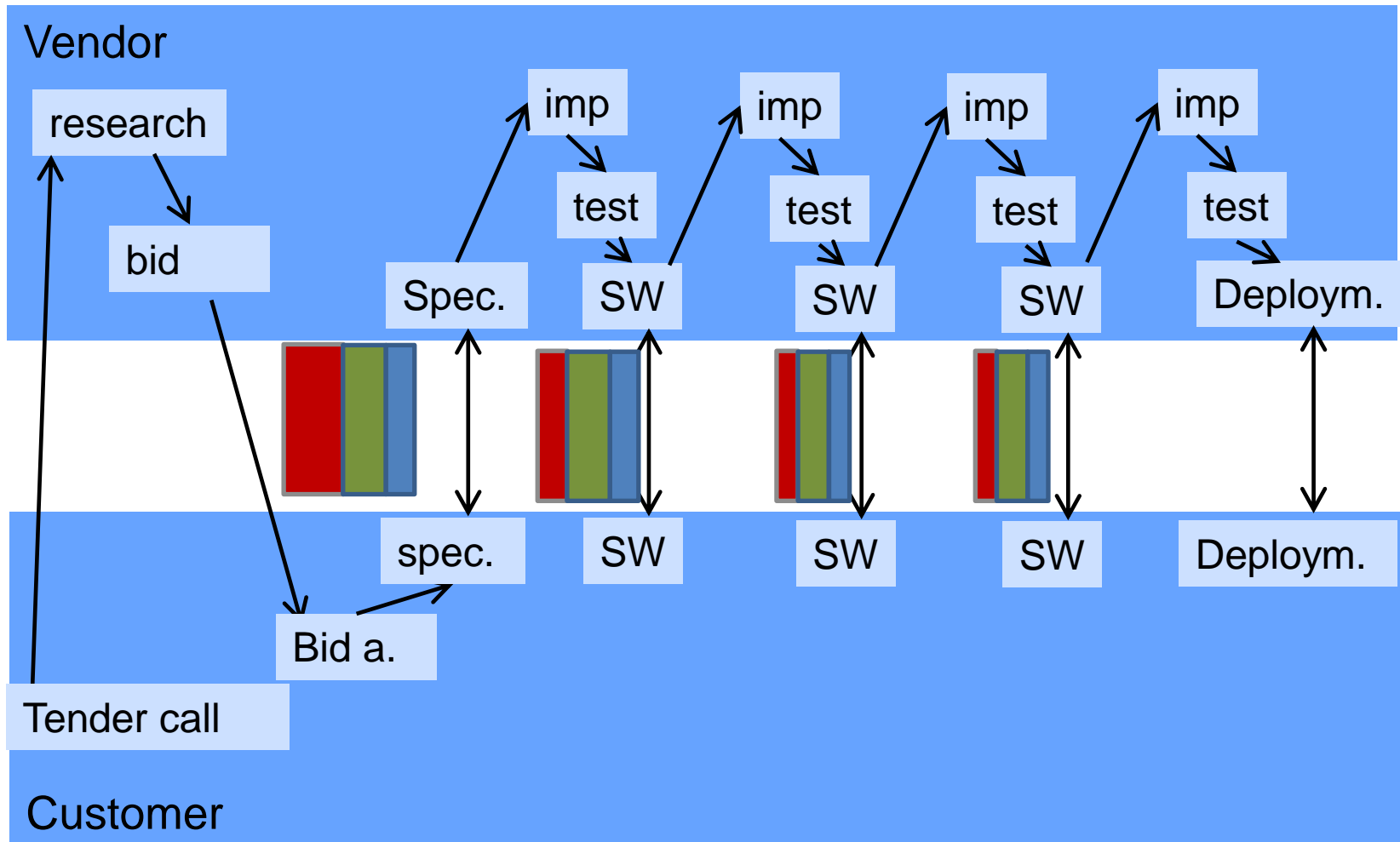


But as Sommerville describes it

Plan-driven vs. agile specification



Iterative, agile



LEAN SW DEVELOPMENT

http://en.wikipedia.org/wiki/Lean_software_development

- Lean software development (LSD) is a translation of lean manufacturing and lean IT principles and practices to the software development domain. Adapted from the Toyota Production System, a pro-lean subculture is emerging from within the Agile community.
- The term lean software development originated in a book by the same name, written by Mary Poppendieck and Tom Poppendieck. The book presents the traditional lean principles in a modified form, as well as a set of 22 tools and compares the tools to agile practices.

Toyota Way

(<http://www.toyotaway.com/>)

What is lean?

- Providing value to your customer by reducing wasteful practices
- Philosophy comes from the world famous Toyota Production System
- Preserving value with less work
- Recognize profit potentials through elimination of wasteful production practices

Toyota Way 14 Principles

(<http://icos.groups.si.umich.edu//Liker04.pdf>)

Section III: Add Value to the Organization by Developing Your People

- Principle 9. Grow leaders who thoroughly understand the work, live the philosophy, and teach it to others.
- Principle 10. Develop exceptional people and teams who follow your company's philosophy.
- Principle 11. Respect your extended network of partners and suppliers by challenging them and helping them improve.

Section IV: Continuously Solving Root Problems Drives Organizational Learning

- Principle 12. Go and see for yourself to thoroughly understand the situation (genchi genbutsu).
- Principle 13. Make decisions slowly by consensus, thoroughly considering all options; implement decisions rapidly (nemawashi).
- Principle 14. Become a learning organization through relentless reflection (hansei) and continuous improvement (kaizen).

Toyota Way 14 Principles

(<http://icos.groups.si.umich.edu//Liker04.pdf>)

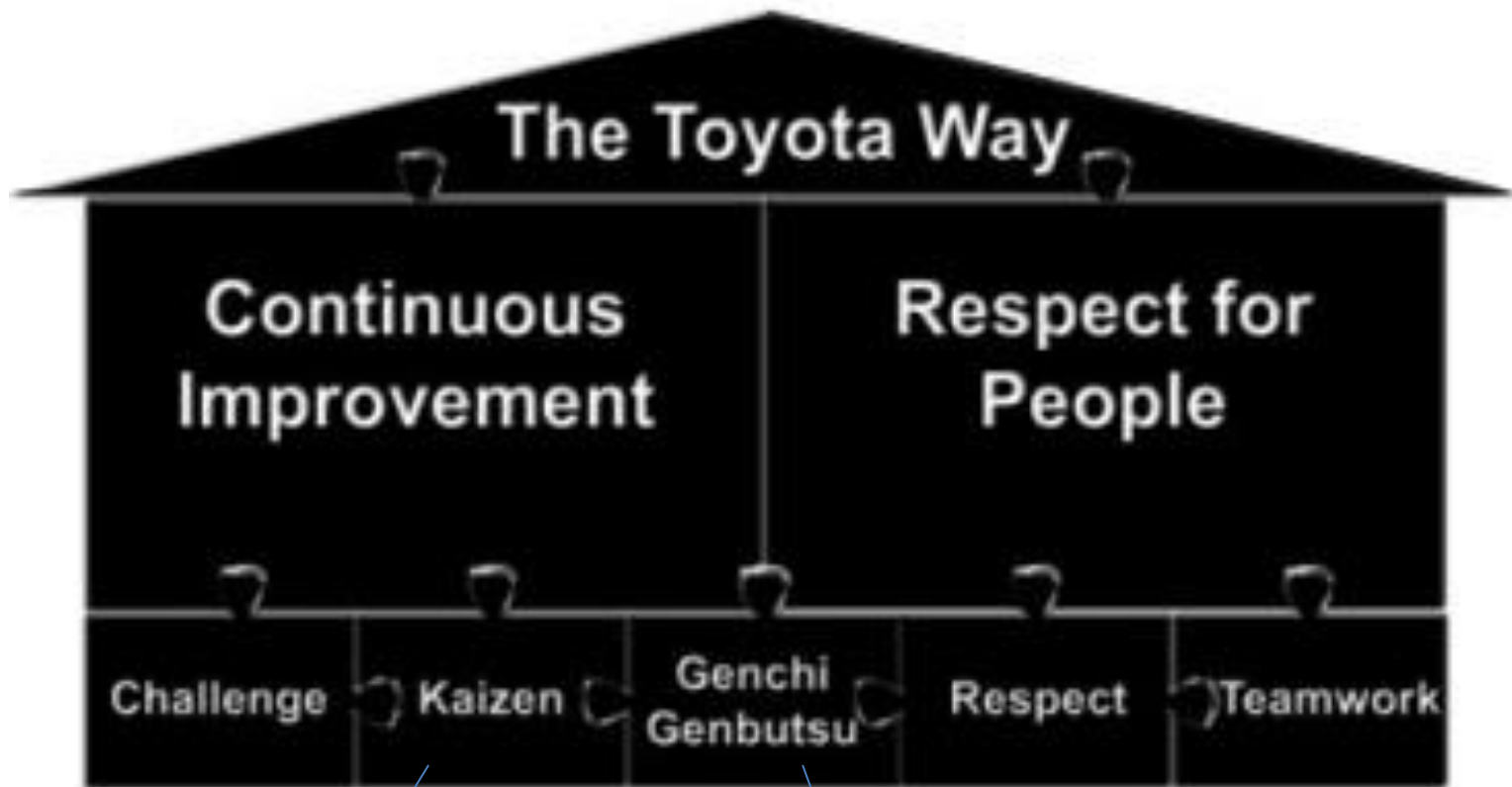
Section I: Long-Term Philosophy

- Principle 1. Base your management decisions on a long-term philosophy, even at the expense of short-term financial goals.

Section II: The Right Process Will Produce the Right Results

- Principle 2. Create a continuous process flow to bring problems to the surface.
- Principle 3. Use “pull” systems to avoid overproduction.
- Principle 4. Level out the workload (heijunka). (Work like the tortoise, not the hare.)
- Principle 5. Build a culture of stopping to fix problems, to get quality right the first time.
- Principle 6. Standardized tasks and processes are the foundation for continuous improvement and employee empowerment.
- Principle 7. Use visual control so no problems are hidden.
- Principle 8. Use only reliable, thoroughly tested technology that serves your people and processes.

<http://www.toyota.eu>



Continuous
improvement

thoroughly understand
the situation

Principles of Lean SW Development

Wikipedia

- Eliminate waste
- Amplify learning
- Decide as late as possible
- Deliver as fast as possible
- Empower the team
- Build integrity in
- See the whole

www.poppendieck.com

- Eliminate Waste
- Keep getting better
- Learn first
- Deliver fast
- Energize workers
- Build quality in
- Focus on customers

Eliminate Waste

Wikipedia

Lean philosophy regards everything not adding value to the customer as waste (muda). Such waste may include:

- unnecessary code and functionality
- delay in the software development process
- unclear requirements
- insufficient testing (leading to avoidable process repetition)
- bureaucracy
- slow internal communication

Poppendick:

The three biggest wastes in product development are:

- **Building the Wrong Thing**
"There is nothing so useless as doing efficiently that which should not be done at all." – Peter Drucker
- **Building the Thing Wrong**
If it seems like there is not enough time to build it right, then there certainly is not enough time NOT to build it right.
- **A Batch and Queue Mentality**
Work in progress hides defects, gets obsolete, causes task switching, and delays realization of value.

Amplify learning; Keep getting better

The best approach for improving a software development environment is to amplify learning.

The accumulation of defects should be prevented by running tests as soon as the code is written.

Instead of adding more documentation or detailed planning, different ideas could be tried by writing code and building.

The process of user requirements gathering could be simplified by presenting screens to the end-users and getting their input.

The learning process is sped up by usage of short iteration cycles – each one coupled with refactoring and integration testing.

There is no such thing as a best practice.

- **Change as Fast as the World Changes** Yesterday's wisdom becomes today's obstacle and tomorrow's folly.
- **Pay Attention to the Small Stuff** Reliable performance comes when noise is not tolerated, when small failures are deeply investigated and corrected.
- **Use the Scientific Method** Establish a hypothesis, conduct many rapid experiments, create concise documentation, and implement the best alternative. Then choose another problem and do it again.

Decide as late as possible; Learn first

As SW development is always associated with uncertainty, better results should be achieved with an options-based approach, delaying decisions until they can be made based on facts and not on assumptions and predictions.

An agile approach can move the options earlier for customers, thus delaying certain crucial decisions until customers have realized their needs better.

This also allows later adaptation to changes and the prevention of costly earlier technology-bounded decisions.

This does not mean that no planning should be involved – on the contrary, planning activities should be concentrated on the different options and adapting to the current situation.

Planning is useful. Learning is essential.

- **The Predictability Paradox**
Predictable organizations do not guess about the future and call it a plan; they develop the capacity to learn quickly and rapidly respond to the future as it unfolds.
- **Integrating Events**
Knowledge-based development seeks out knowledge gaps, develops multiple options for solutions, and frequently synchronizes all teams developing the system.
- **The Last Responsible Moment**
Don't make expensive-to-change decisions before their time – and don't make them after their time!

Deliver as fast as possible

In the era of rapid technology evolution, it is not the biggest that survives, but the fastest.

The sooner the product is delivered without major defects, the sooner feedback is received, and incorporated into the next iteration.

The shorter the iterations, the better the learning and communication within the team.

The just-in-time production ideology could be applied to software development, recognizing its specific requirements and environment.

This is achieved by presenting the needed result and letting the team organize itself and divide the tasks for accomplishing the needed result for a specific iteration.

At the beginning, the customer provides the needed input.

Create a steady, even flow of work, pulled from a deep understanding of value.

Speed, Quality & Low Cost are Fully Compatible

Companies that compete on the basis of speed have a big cost advantage, deliver superior quality, and are more attuned to their customers' needs.

Focus on Flow Efficiency, not Resource Efficiency

Resource efficiency interferes with the smooth flow of value; it often delivers half the value for twice the effort.

Manage Workflow rather than Task-based Schedules

The best way to establish reliable, predictable deliveries is to establish reliable, repeatable workflows.

Empower the team; Energize workers

Traditionally the managers tell the workers how to do their own job. In a "Work-Out technique"" managers are taught how to listen to the developers, so they can explain better what actions might be taken, as well as provide suggestions for improvements.

The lean approach: "find good people and let them do their own job," encouraging progress, catching errors, and removing impediments, but not micro-managing.

Another mistaken belief has been the consideration of people as resources.

People might be resources from the point of view of a statistical data sheet, but in software development

The developers should be given access to the customer; the team leader should provide support and help in difficult situations, as well as ensure that skepticism does not ruin the teams spirit.

The time and energy of bright, creative people are the scarce resources in today's economy.

- **Purpose**
A meaningful purpose inspires and energizes workers.
- **Challenge**
Provide challenge, feedback, and an environment that enables everyone to become excellent.
- **Responsibility**
The most productive groups are semi-autonomous teams – with an internal leader – that accept end-to-end responsibility for meaningful accomplishments.

Build integrity in; build quality in

The customer needs to have an overall experience of the System – this is the so-called perceived integrity: how it is being advertised, delivered, deployed, accessed, how intuitive its use is, price and how well it solves problems.

Conceptual integrity means that the system's separate components work well together as a whole with balance between flexibility, maintainability, efficiency, and responsiveness.

- Should be understood as whole
- Information is received in pieces

One of the healthy ways towards integral architecture is refactoring.

Automated tests are also considered part of the production process, and therefore if they do not add value they should be considered waste.

Find and fix defects the moment they occur.

Mistake-Proof the Process

Think of tests as specifications. Use them to establish confidence in the correctness of the system **at any time** during development, **at every level** of the system.

Integrate Early and Often

Every development process ever invented had as its primary purpose to find and fix defects as early in the development process as possible.

Don't Tolerate Defects

If you expect to find defects during final verification, your development process is defective.

See the whole; focus on customers

Software systems are not simply the sum of their parts, but also the product of their interactions.

Defects in software tend to accumulate during the development process – by decomposing the big tasks into smaller tasks, and by standardizing different stages of development, the root causes of defects should be found and eliminated.

The larger the system, the more organizations that are involved in its development and the more parts are developed by different teams, the greater the importance of having well defined relationships between different vendors.

"If you organize around the consumer, the rest of it will follow." – Eric Schmidt

Ask the Right Questions

Innovation begins with a fresh perspective, a keen insight, a penetrating question.

Solve the Right Problems

Do not focus on the products you are building, focus on the problems customers are encountering.

Design a Great Experience

It is not enough for customers to be satisfied, they should *love* your products.

KANBAN

Kanban

(wikipedia based on David Anderson. Kanban – Successful Evolutionary change for your Technology Business. Blue Hole Press, April 2010)

Visualise

- Visualising the flow of work and making it visible is core to understanding how work proceeds. Without understanding the workflow, making the right changes is harder.
- A common way to visualise the workflow is to use a card wall with cards and columns. The columns on the card wall representing the different states or steps in the workflow.

Limit WIP

- Limiting work-in-process implies that a pull system is implemented on parts or all of the workflow.
- The critical elements are that work-in-process at each state in the workflow is limited and that new work is “pulled” when there is available capacity within the local WIP limit.

Manage flow

- The flow of work through each state in the workflow should be monitored, measured and reported. By actively managing the flow the continuous, incremental and evolutionary changes to the system can be evaluated to have positive or negative effects on the system.

Kanban

Make policies explicit

- Until the mechanism of a process is made explicit it is often hard or impossible to hold a discussion about improving it.

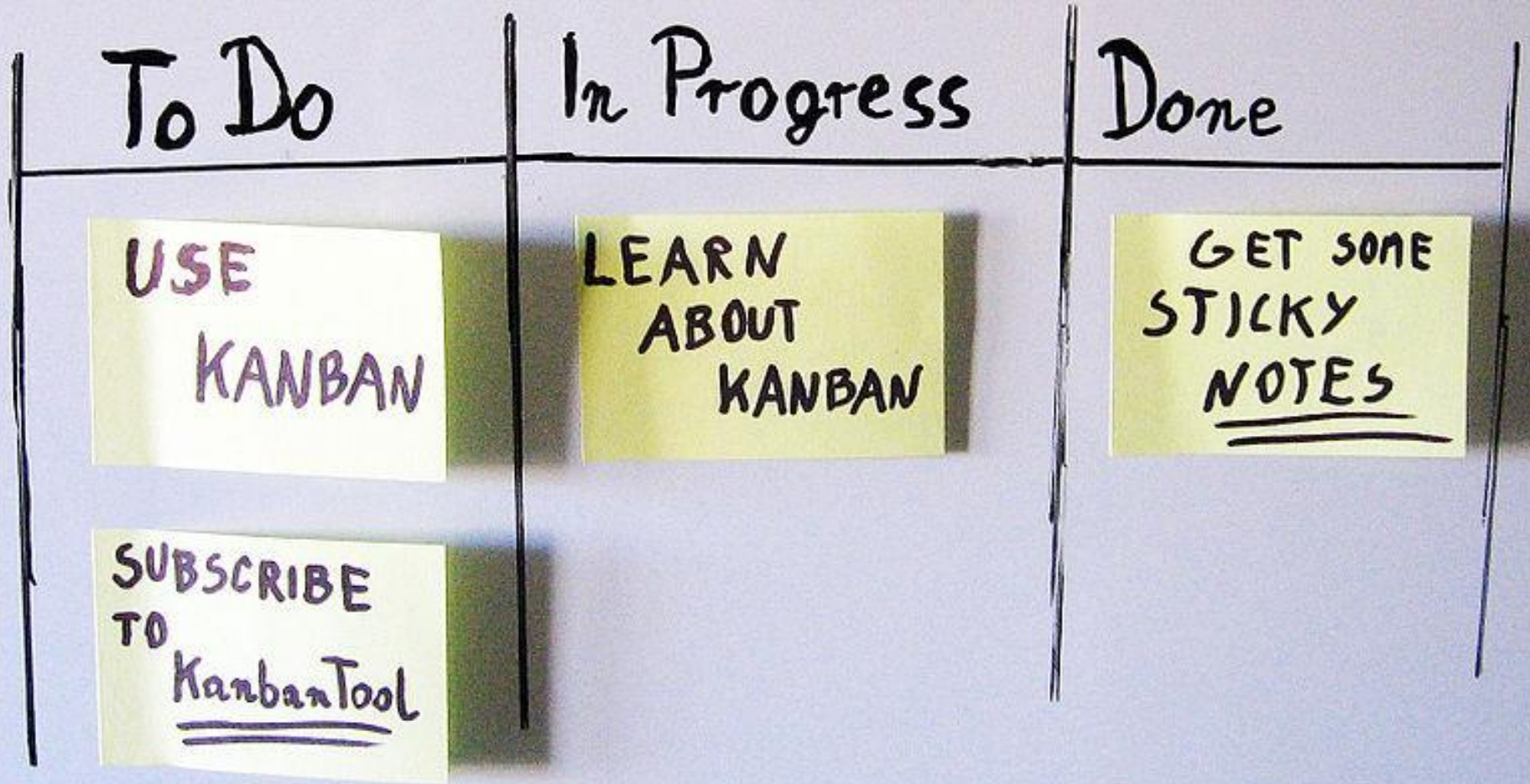
Implement feedback loops

- Collaboration to review flow of work and demand versus capability measures, metrics and indicators coupled with anecdotal narrative explaining notable events is vital to enabling evolutionary change.

Improve collaboratively, evolve experimentally (using models and the scientific method)

- The Kanban method encourages small continuous, incremental and evolutionary changes that stick.
- When teams have a shared understanding of theories about work, workflow, process, and risk, they are more likely to be able to build a shared comprehension of a problem and suggest improvement actions which can be agreed by consensus.
- The Kanban method suggests that a scientific approach is used to implement continuous, incremental and evolutionary changes. The method does not prescribe a specific scientific method to use.

Kanban board

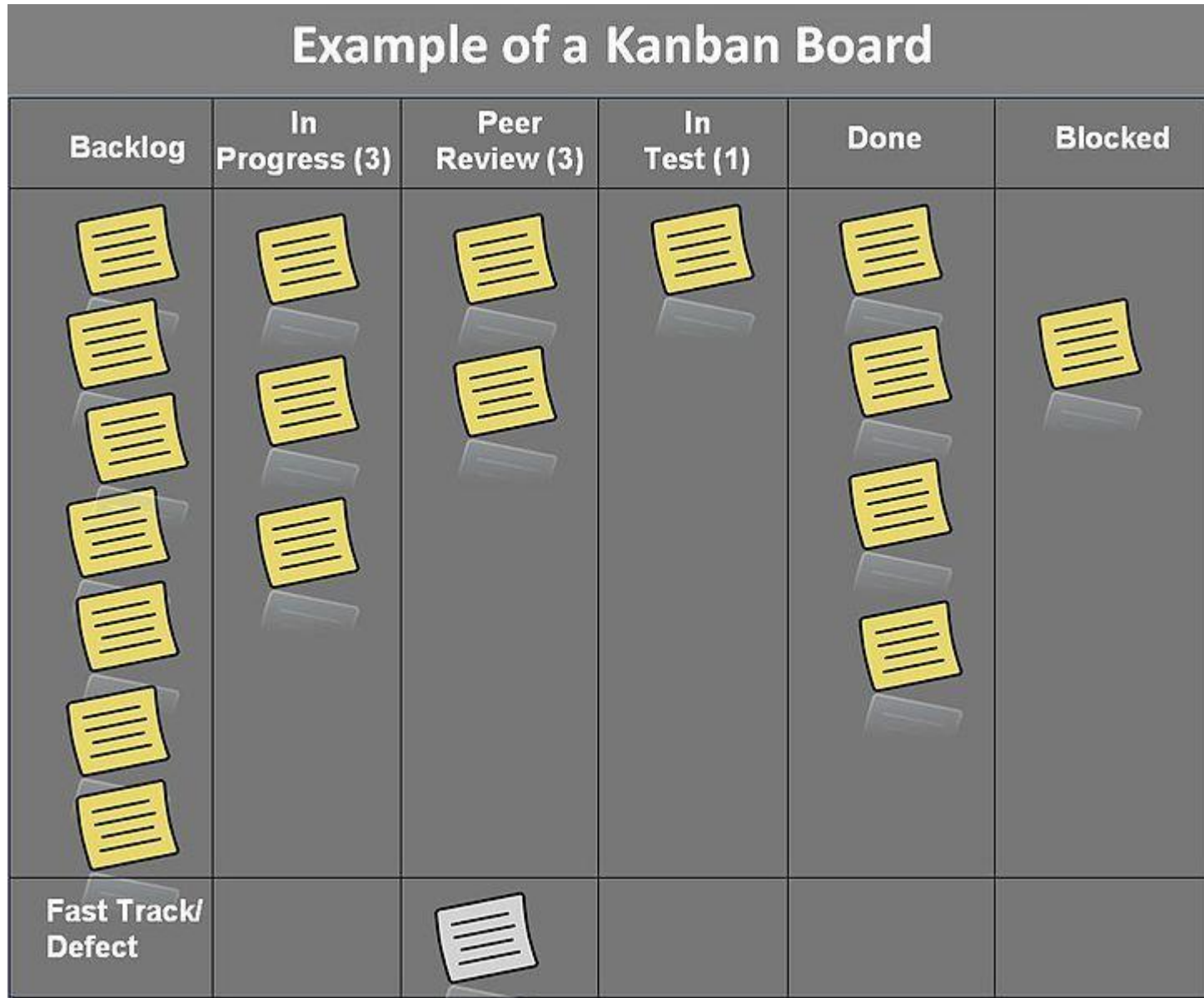


Example of tool support

<http://kanbantool.com/>

The screenshot displays the Kanban Tool interface in a web browser. The browser's address bar shows the URL `board.kanbantool.com`. The page title is "MTVN / Project board". The navigation bar includes links for "Board", "Analytics", "Archive", "Settings", and "Exit". A search bar on the right allows searching for titles, card types, or tags. The main workspace is a Kanban board with columns: "To do", "Accepted", "Buffer", "Doing", "Done", "Review", and "Done". Each column contains task cards, some of which are blue and others grey. The "Buffer" column shows a count of 10 / 3. The "Doing" column has a "Working" sub-header. The "Done" column has a "Done" sub-header. The "Review" column has a "Review" sub-header. The "Done" column has a "Done" sub-header. The board is organized into two rows of cards. The first row contains cards for tasks like "As a user I can remove an item from a basket", "Registered User who has paid can access document", "As a user I can cancel my account", "As a user I can log out", "As a user I can edit my personal data", "As a user I can settle the time", "Add 'About us' page", and "As a user I can ask a team member what to do". The second row contains cards for tasks like "As a user I can check the number of items in my basket", "Review code before updating the website", "As a user I can ask a team member to join my account", "As a user I can sign out", "As a user I can find news cards", "As a user I can ask a team member to join", "As a user I may sign up now for free", "As a user I can edit my password and check my current version", and "As a user I can ask a team member what to do".

http://commons.wikimedia.org/wiki/File:Kanban_board_example.jpg



Motivation behind Kanban

(<http://www.netobjectives.com/files/resources/articles/Demystifying-Kanban.pdf> based on
(<http://www.netobjectives.com/blogs/real-differences-between-kanban-and-scrum>)

- **Controlling the rate of transition.** First-generation methods often require traumatic change to the people and structures in the organization.
- **Allocating specialized skill sets, domain knowledge, or knowledge of legacy code effectively across the organization.** Dedicated and relatively static teams, while desirable, are usually not practical in this regard.
- **Enabling participation of management and leadership.** Scrum often marginalizes management.
- **Providing teams with guiding principles, especially the principles of lean and the theory of constraints.**
- **Providing a better way to learn how to improve.** End-of-iteration retrospectives are too narrow and too late to be valuable over the long haul.
- **Enabling teams to work on the right-sized chunks.** With time-boxing, work gets squeezed into a predefined period and unrelated bits of work get grouped into one sprint.

<http://www.netobjectives.com/blogs/real-differences-between-kanban-and-scrum>
(one opinion!)

	Kanban	Scrum
Explicit policies	Yes	Don't believe it possible
Manage Work in Progress (WIP)	Yes	Not mentioned, don't know how to do it without explicit policies
Visibility of process	Input, work, output	Input and output only. Work is black-box
Management	Inclusive	Keep them at bay
Value stream	Includes product management across products	At team level for one product
Change management	Controllable	Must change to Scrum model – even if disruptive

It is also a cultural issue

(source Cutter IT Journal)

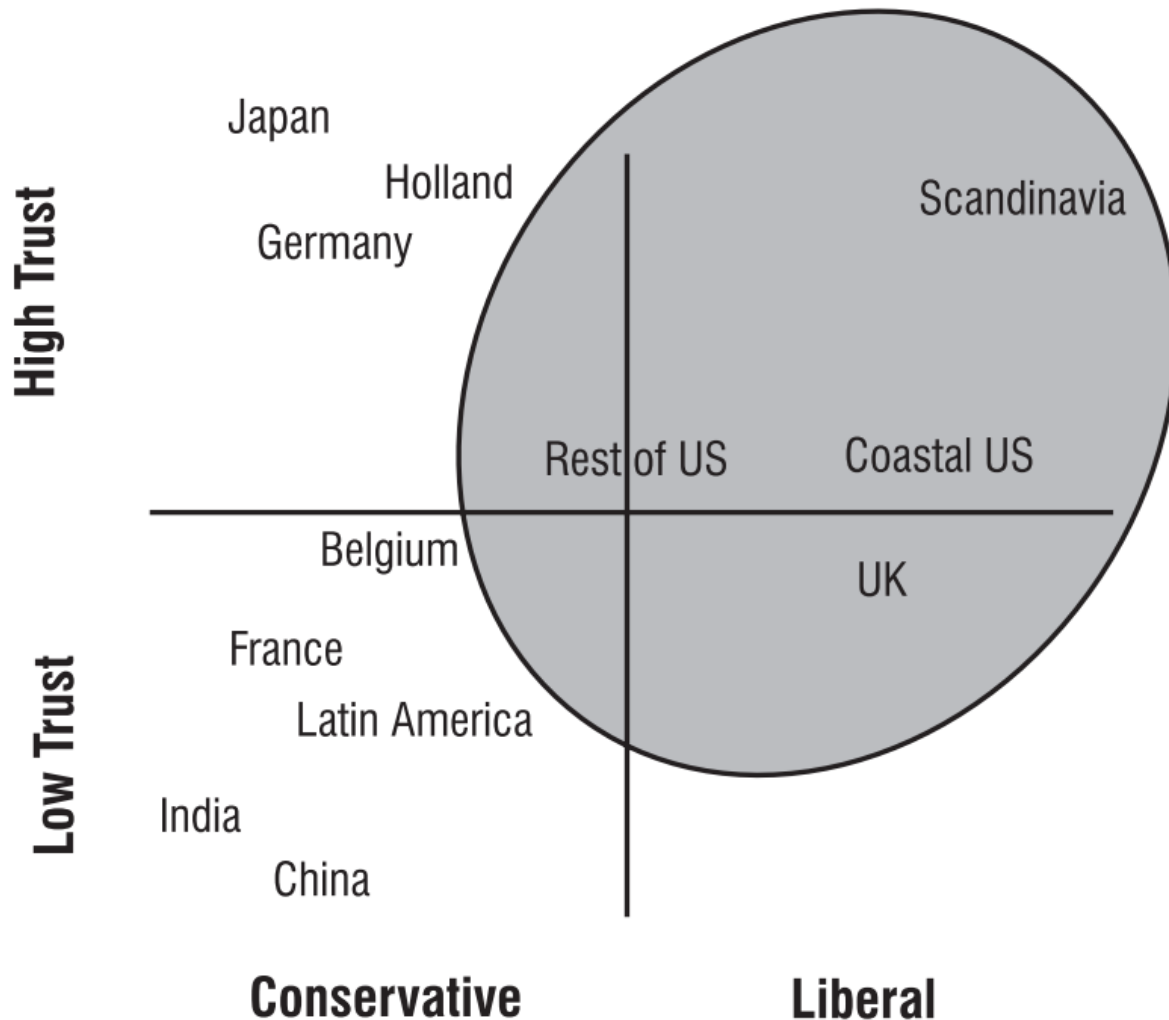
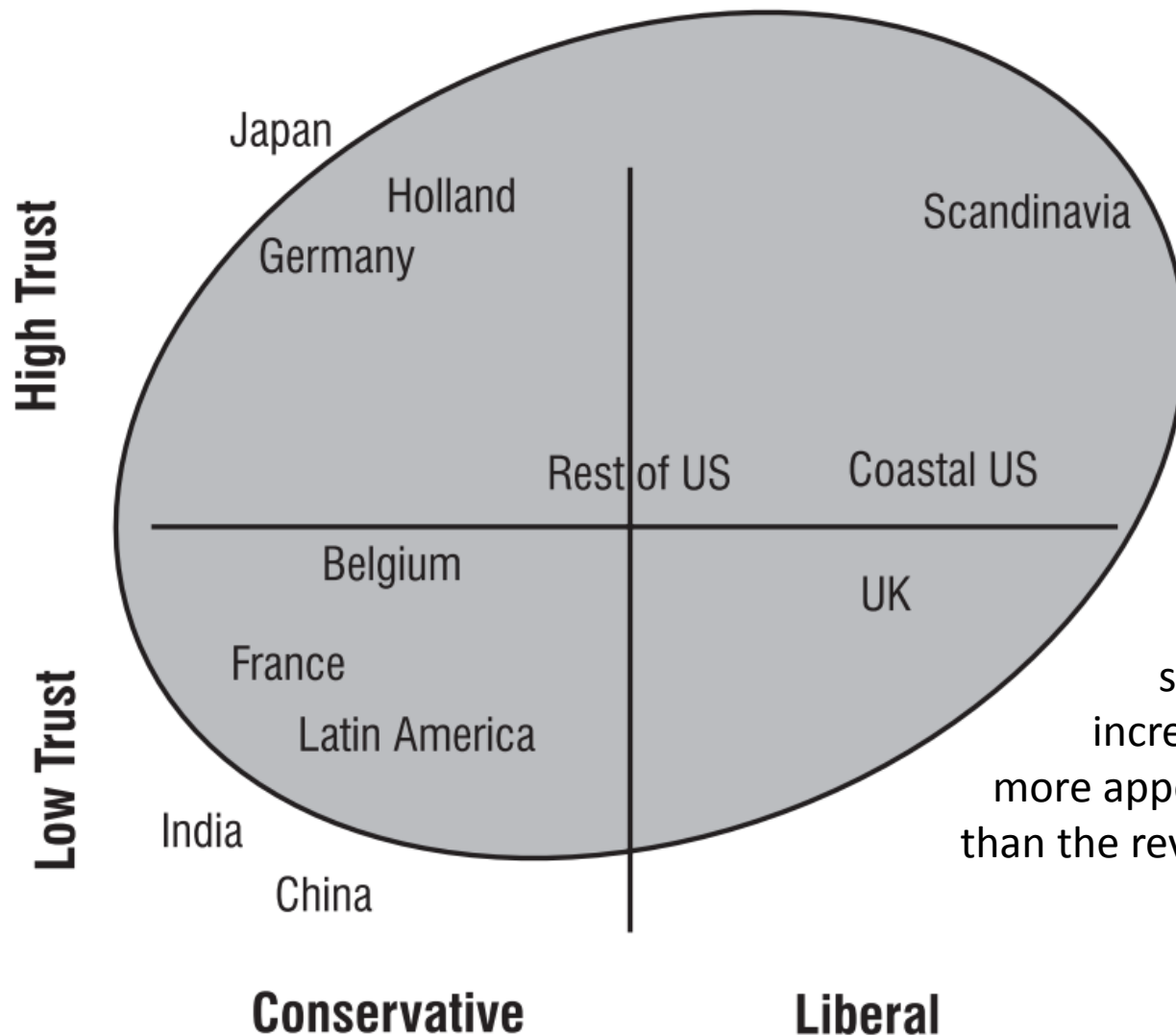


Figure 1 — Early agile adoption.

It is also a cultural issue

(source Cutter IT Journal)



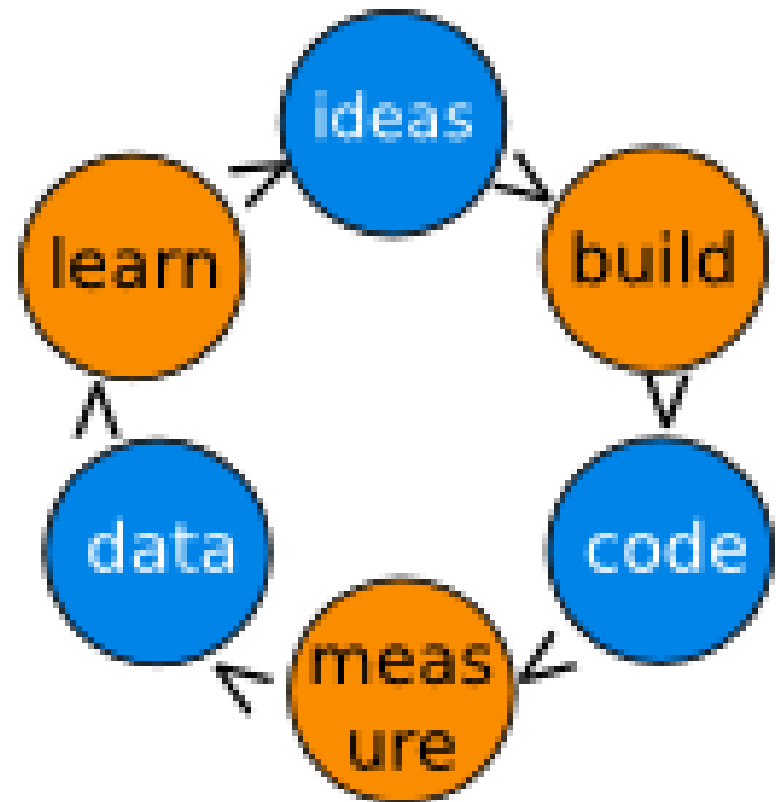
“Note that the adoption is wider and that the more conservative cultures have adopted it fairly rapidly. We attribute this difference specifically to the evolutionary, incremental nature of Kanban. It is more appealing in conservative cultures than the revolutionary approach required by some agile methods.”

Figure 2 — Kanban adoption since 2007.

LEAN STARTUP

Lean startup

- Based on
 - work by Eric Ries, e.g. presentation "The Lean Startup Doing More With Less"
 - Presentations of Pekka Abrahamsson



Understanding Failure

- Not because the technology doesn't work
- No customers or a sustainable business model
- With better management, idea failure doesn't have to lead to company failure

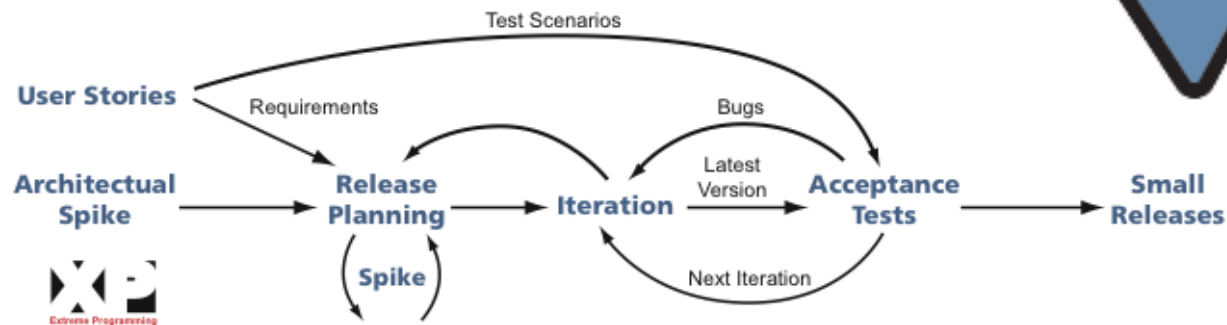
Agile Product Development

Unit of Progress: A line of Working Code

“Product Owner” or in-house customer

Problem: known

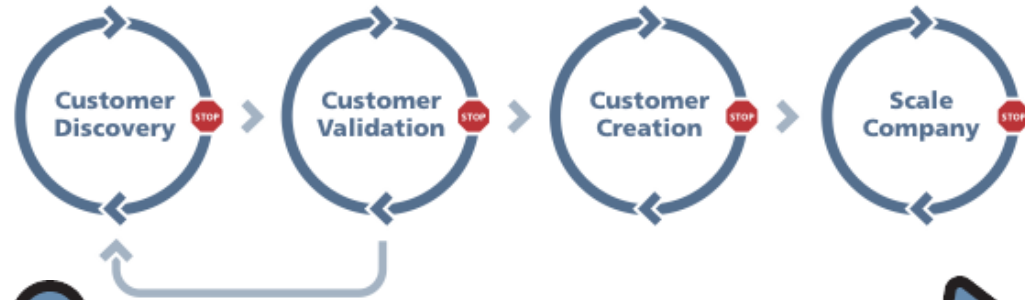
Solution: unknown



Product Development at Lean Startup

Unit of Progress: Validated Learning About Customers (\$\$\$)

Customer Development

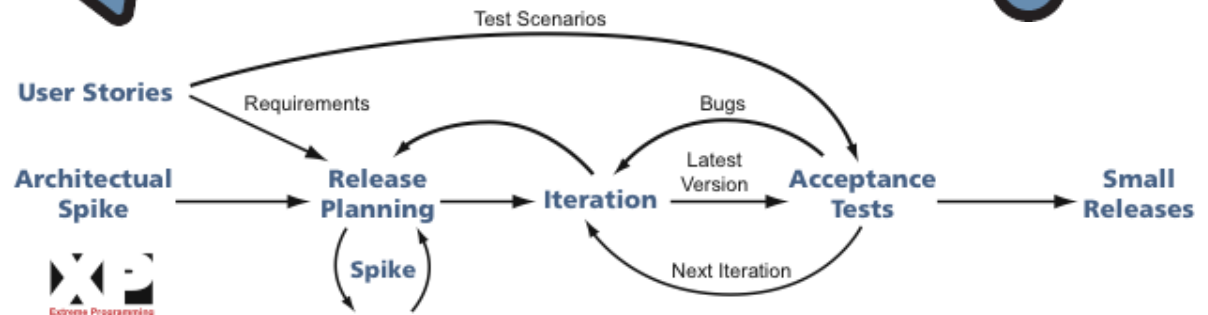


Problem: unknown

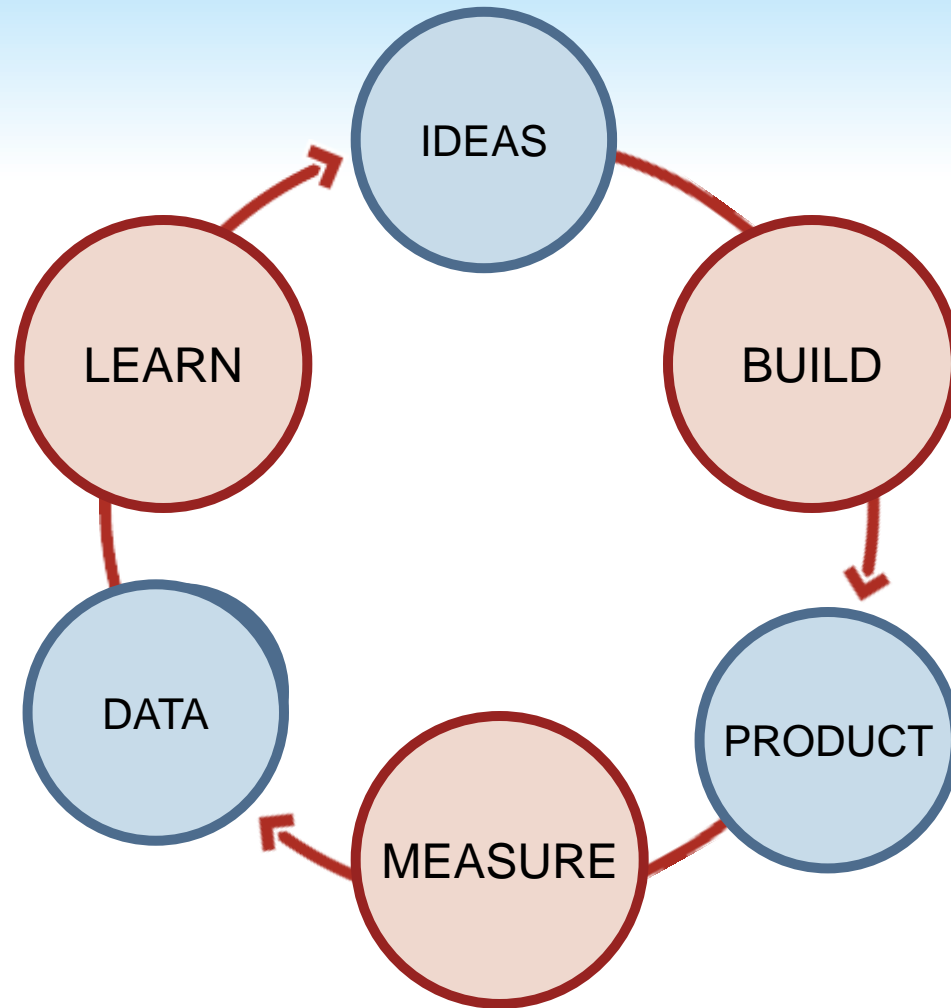
Solution: unknown

Hypotheses,
Experiments,
Insights

Data,
Feedback,
Insights

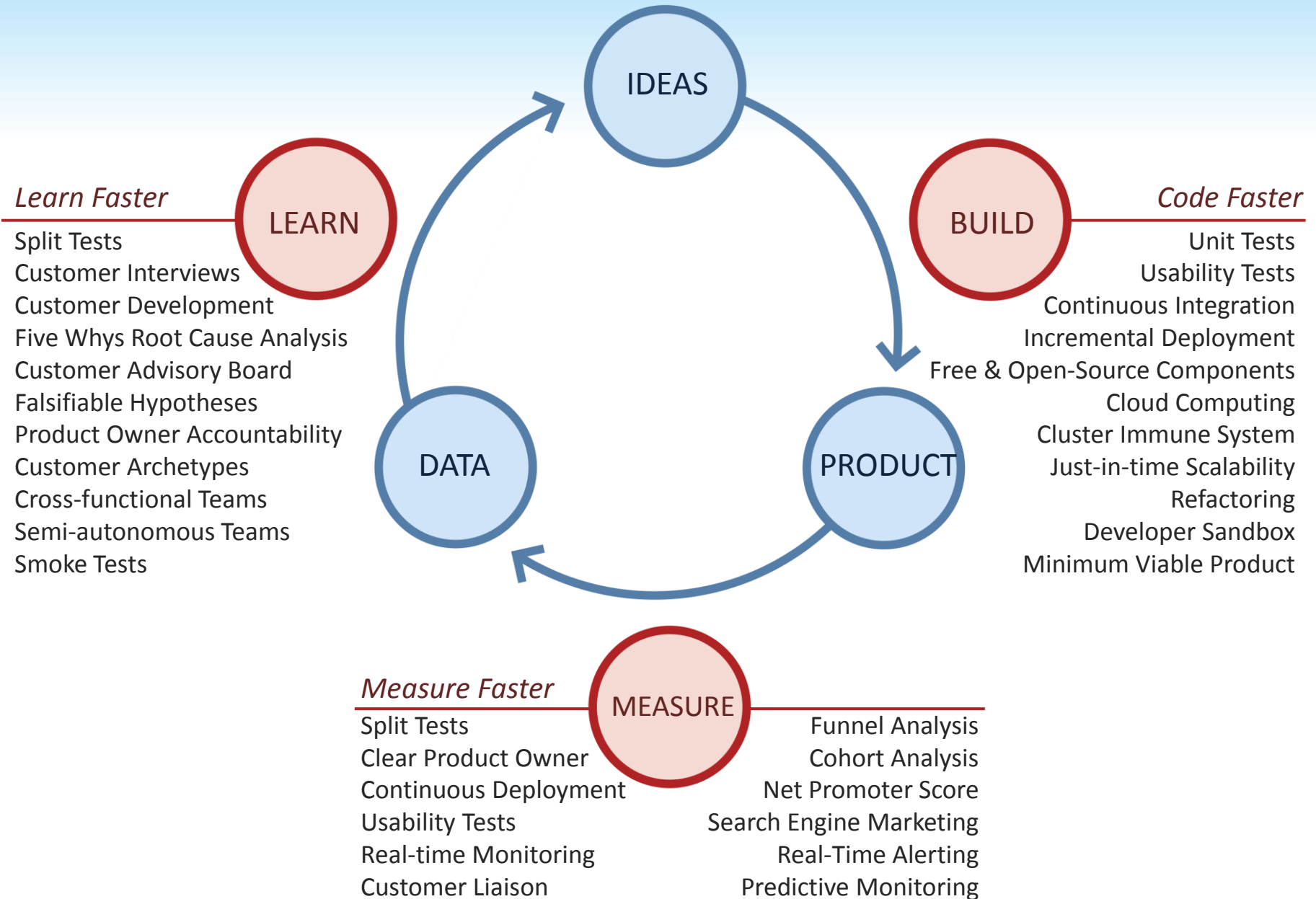


The Startup OODA Loop



Minimize *TOTAL* time through the loop

There's much more...



Key elements and terms of Lean Startup

- A **minimum viable product** (MVP) is the version of a new product which allows a team to collect the maximum amount of validated learning about customers with the least effort.
- **Continuous deployment** is a process whereby all code that is written for an application is immediately deployed into production.
- A **split or A/B test** is an experiment in which "different versions of a product are offered to customers at the same time.
- **Actionable metrics** can lead to informed business decisions and subsequent action. These are in contrast to 'vanity metrics' - measurements that give "the rosiest picture possible" but do not accurately reflect the key drivers of a business.
- A **pivot** is a structured course correction designed to test a new fundamental hypothesis about the product, strategy, and engine of growth.



DONNELLY

"I'm not leaving you. I'm pivoting to another man."