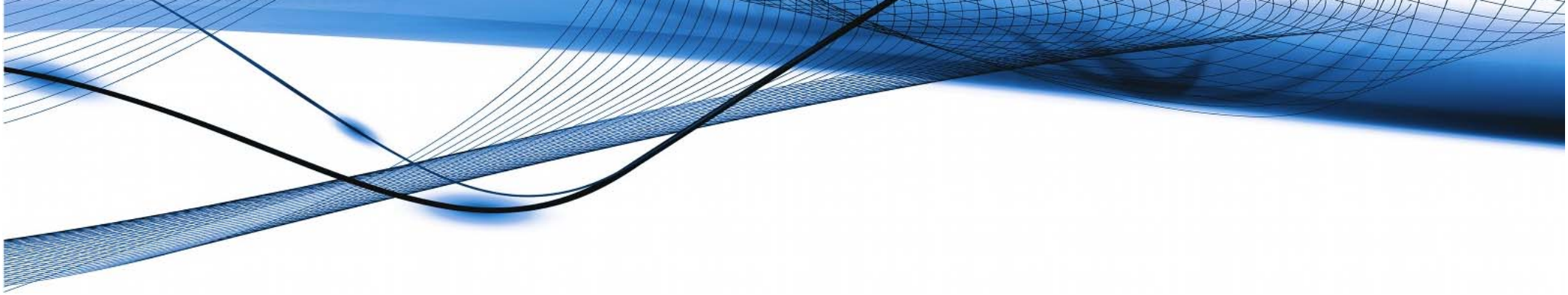


Agile Adoption and the Software Value Chain

Saturday, September 12
AgileChina 2009

敏捷中国大会

ThoughtWorks®

InfoQ
Enterprise Software Development Community

The Agile Manifesto (February 13, 2001)

We are uncovering better ways of developing
software by doing it and helping others do it.

Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

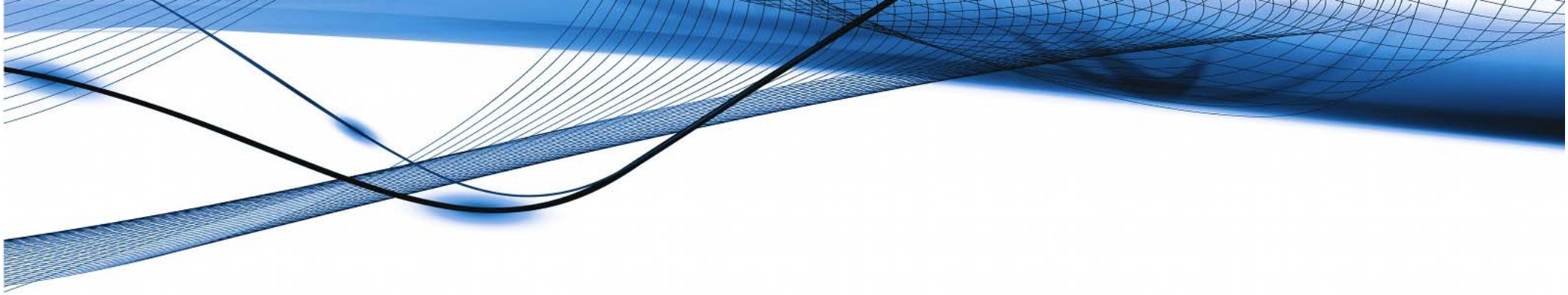
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

Kent Beck / Mike Beedle / Arie van Bennekum / Alistair Cockburn / Ward Cunningham / Martin Fowler
James Grenning / Jim Highsmith / Andrew Hunt / Ron Jeffries / Jon Kern / Brian Marick / Robert C. Martin
Steve Mellor / Ken Schwaber / Jeff Sutherland / Dave Thomas

http://agilemanifesto.org

ThoughtWorks®



Why Agile?

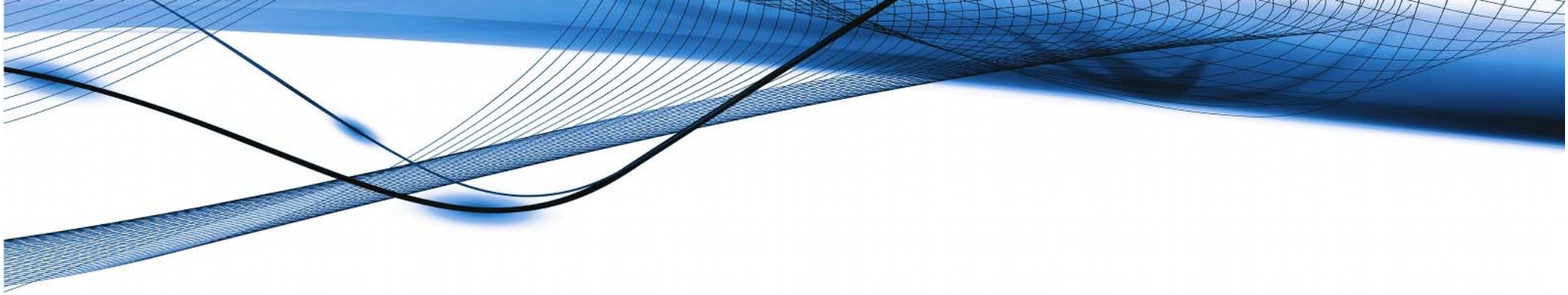
Why do we need better ways of developing software?

Why do we need software process?

Since computers were first invented...

- Business has wanted:
  - Predictable schedule
  - Predictable budget
  - Lots and lots of features
- Business has delivered:
  - Unclear and incomplete specifications
  - Budget cuts
  - Schedule pressure
- Programmers have wanted:
  - Clear and complete specifications
  - Sufficient resources
  - Sufficient time
- Programmers have delivered:
  - Missed deadlines
  - Unforeseen costs
  - Lots and lots of bugs

ThoughtWorks®



Moral:
Nobody wants uncertainty

Successful software development delivers predictable functionality for a predictable budget on a predictable schedule

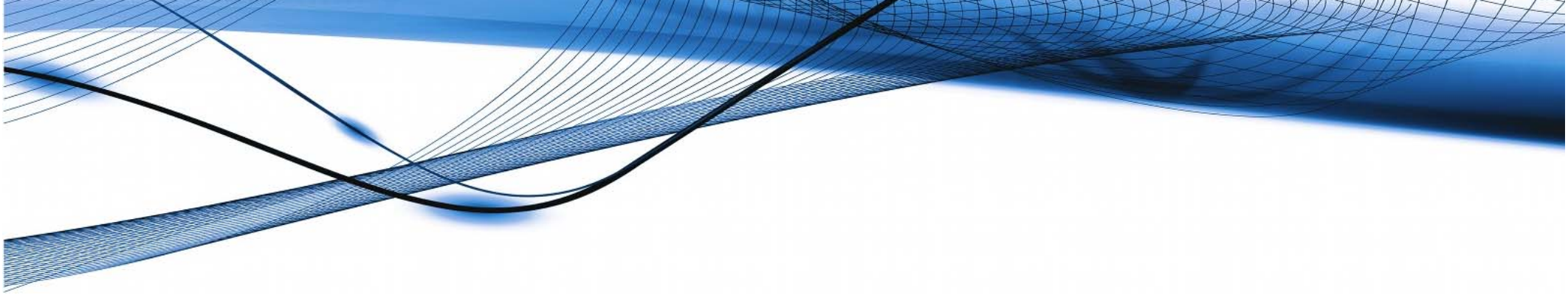
ThoughtWorks©

Solution: The Waterfall approach

- If you can give us a complete specification
- We can tell you how long it will take to build
- And how much it will cost to build

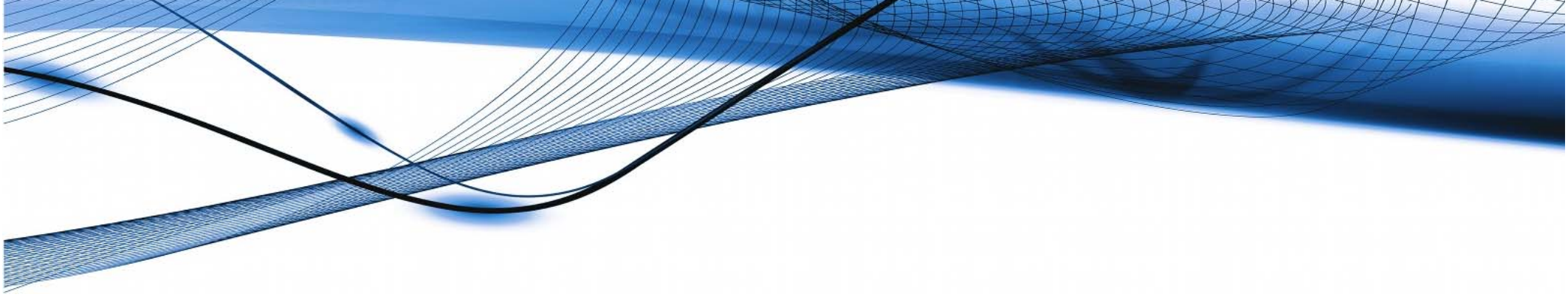
But...

- What you specify won't necessarily be what you need
- When it is finally built, your needs may have changed
- By the time you find out, it will likely be too late to limit your lost time and money



So, does Waterfall eliminate the uncertainty?

Does it eliminate the right uncertainty?



New moral: Successful software recovers the costs of production

Waterfall process does not provide certainty of success

ThoughtWorks©

The Agile Manifesto (February 13, 2001)

We are uncovering better ways of developing
software by doing it and helping others do it.

Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

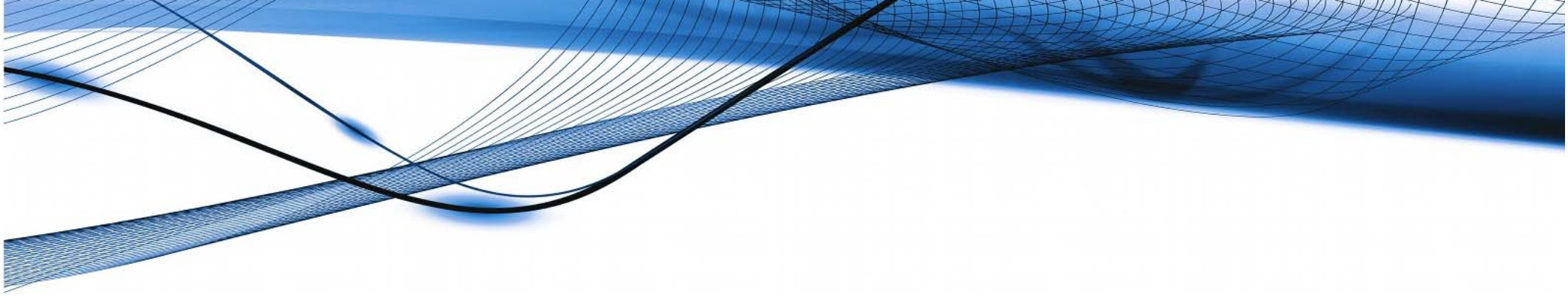
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

Kent Beck / Mike Beedle / Arie van Bennekum / Alistair Cockburn / Ward Cunningham / Martin Fowler
James Grenning / Jim Highsmith / Andrew Hunt / Ron Jeffries / Jon Kern / Brian Marick / Robert C. Martin
Steve Mellor / Ken Schwaber / Jeff Sutherland / Dave Thomas

http://agilemanifesto.org

ThoughtWorks®



The Agile Manifesto: Value over Predictability

ThoughtWorks©

Solution: The Agile approach

- If you actively participate in our process
- We will give you working code soon, and frequently
- According to your most immediate, highest value needs
- And adjust to your changes quickly and cheaply

But...

- We can't predict the distant future
- So we won't be able to tell you for certain what exactly your budget will buy, within your schedule
- Until we get there

Benefits of the Agile approach

- Shorter time from project conception to code in production means investment in development resources sees quicker return to business value
- At any given point in time, investment in development resources is being spent on greatest business value
- At any point in time, resources can be reallocated to higher priority or more valuable functionality
- Any discrepancy between stated and actual business requirements can be detected and remedied within weeks

Agile process overview

- Project scope is decomposed into small, testable, independent units of functionality, called “stories”
- Developers estimate effort to implement stories, and the customer prioritizes stories by business value
- Development activity is divided into short cycles (typically one to four weeks each), called “iterations”
- Each iteration, the customer selects stories for development by value and expected effort
- Each iteration, developers implement and deliver the selected stories to the customer

User Stories

- User Stories are the fundamental unit of Agile software development
- Stories form the basis for metrics and planning
- Stories are the “common currency” between business and developers
- Good stories provide the foundation for successful* software development
- Good stories satisfy “INVEST”

INVEST: Independent

- Business value stands on its own merit
- The Agile contract with business states business may choose any story in any iteration
- …or choose none at all (cancel project at any time)
- Story should be worth doing in isolation
- Should be possible to implement story in isolation
  - Not always possible, but should always be a goal

INVEST: Negotiable

- Clear distinction between “essential” and “incidental”
- Essential aspects are required for business value
- Incidental aspects allow for the discretion of developers (i.e. negotiable)
- Business analysis determines what is essential and what is negotiable, and this is made explicit prior to development

INVEST: Valuable

- Business value is tangible and immediate
- Business analysis should, ideally, be able to trace value back to business value drivers (revenue, cost, risk)
- Ideally, business analysis should be able to quantify value (e.g. business case)

INVEST: Estimable

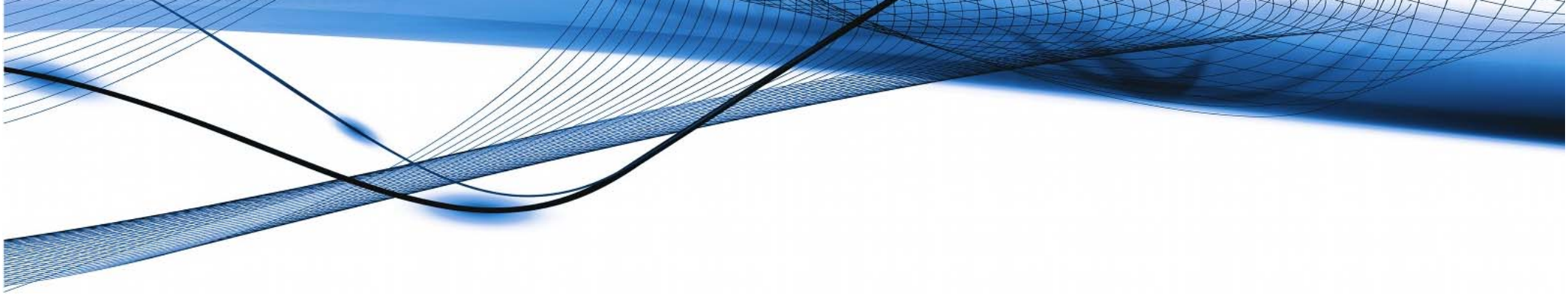
- Implementation effort can reasonably be foreseen
- Importance of “essential” vs. “incidental” negotiation
- Importance of completeness and accuracy of acceptance criteria

INVEST: Small

- Scope is limited to the minimum immediate, independent value
- Decomposing large stories into smaller stories with independent value is a key business analysis skill
- Be clear on the distinction between “small” stories and “large” acceptance criteria
  - A story will have multiple essential acceptance criteria
  - A story will satisfy INVEST

INVEST: Testable

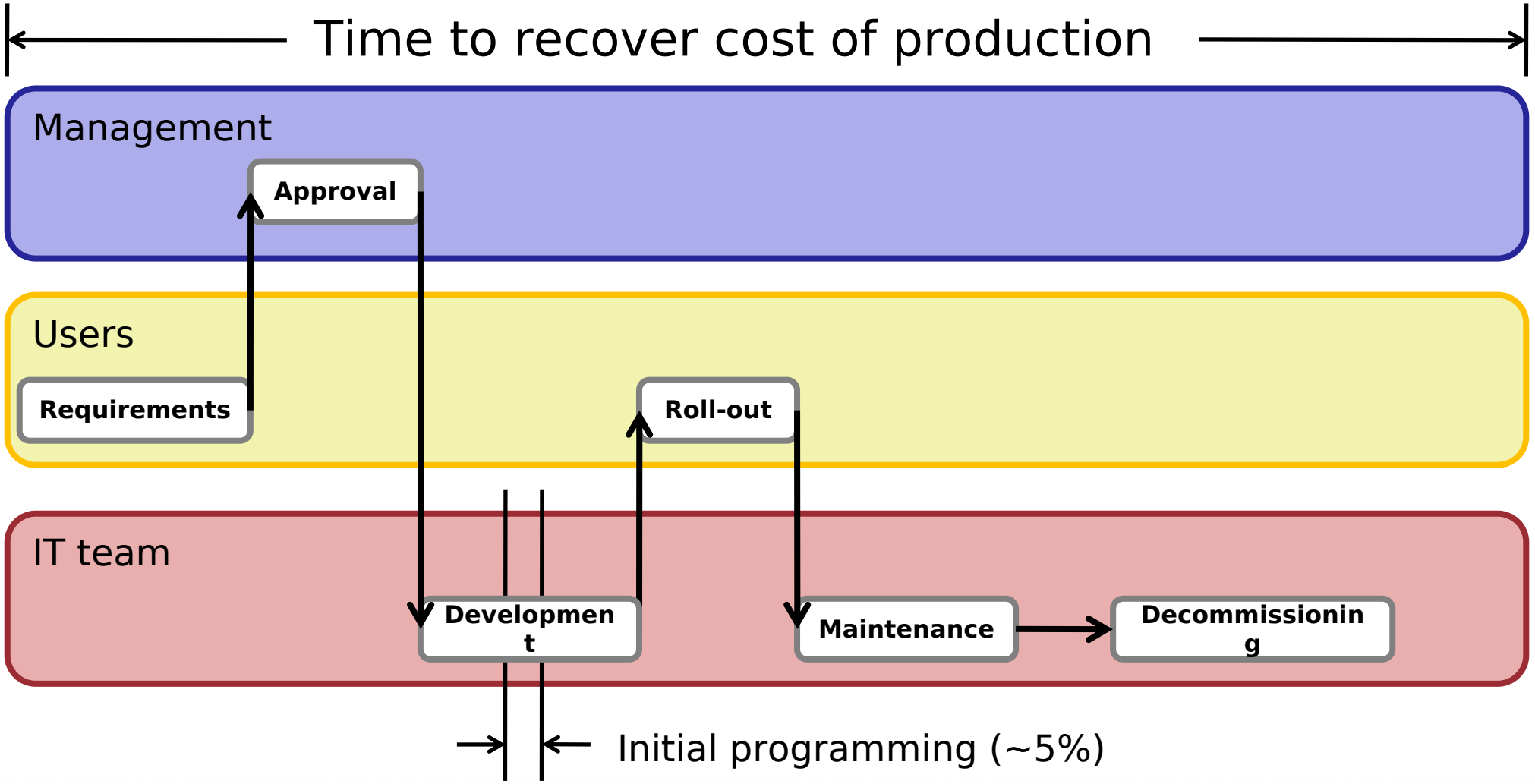
- Successful completion can be demonstrated objectively
- Test plans help define stories; business analysis should coordinate with quality analysis
- Tests are how developers know they’re done
- The quality of story testability contributes directly to the efficiency and reliability of the development process



The Software Lifecycle Value Chain

ThoughtWorks©

Software lifecycle overview

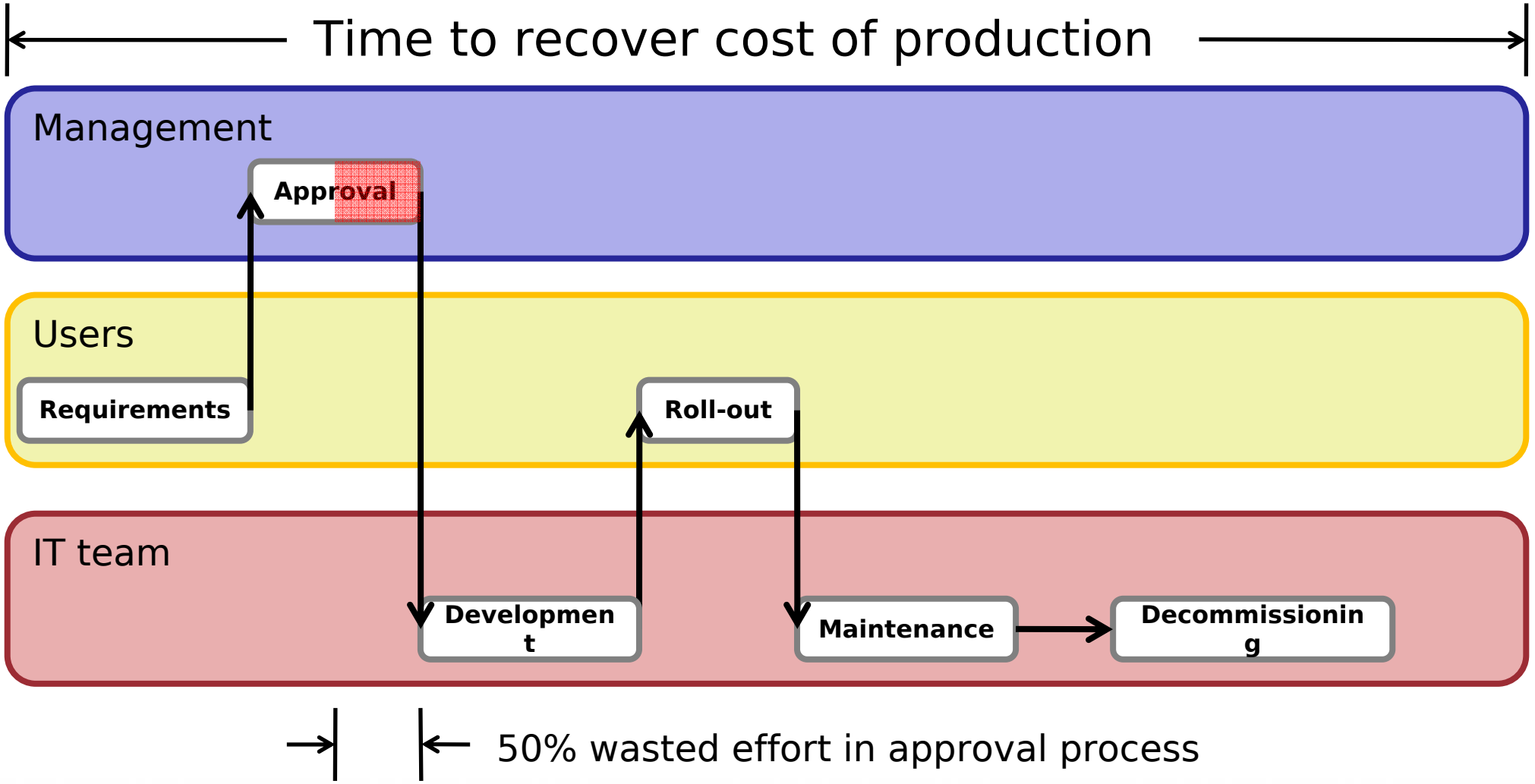


ThoughtWorks®

Local vs. global process inefficiency

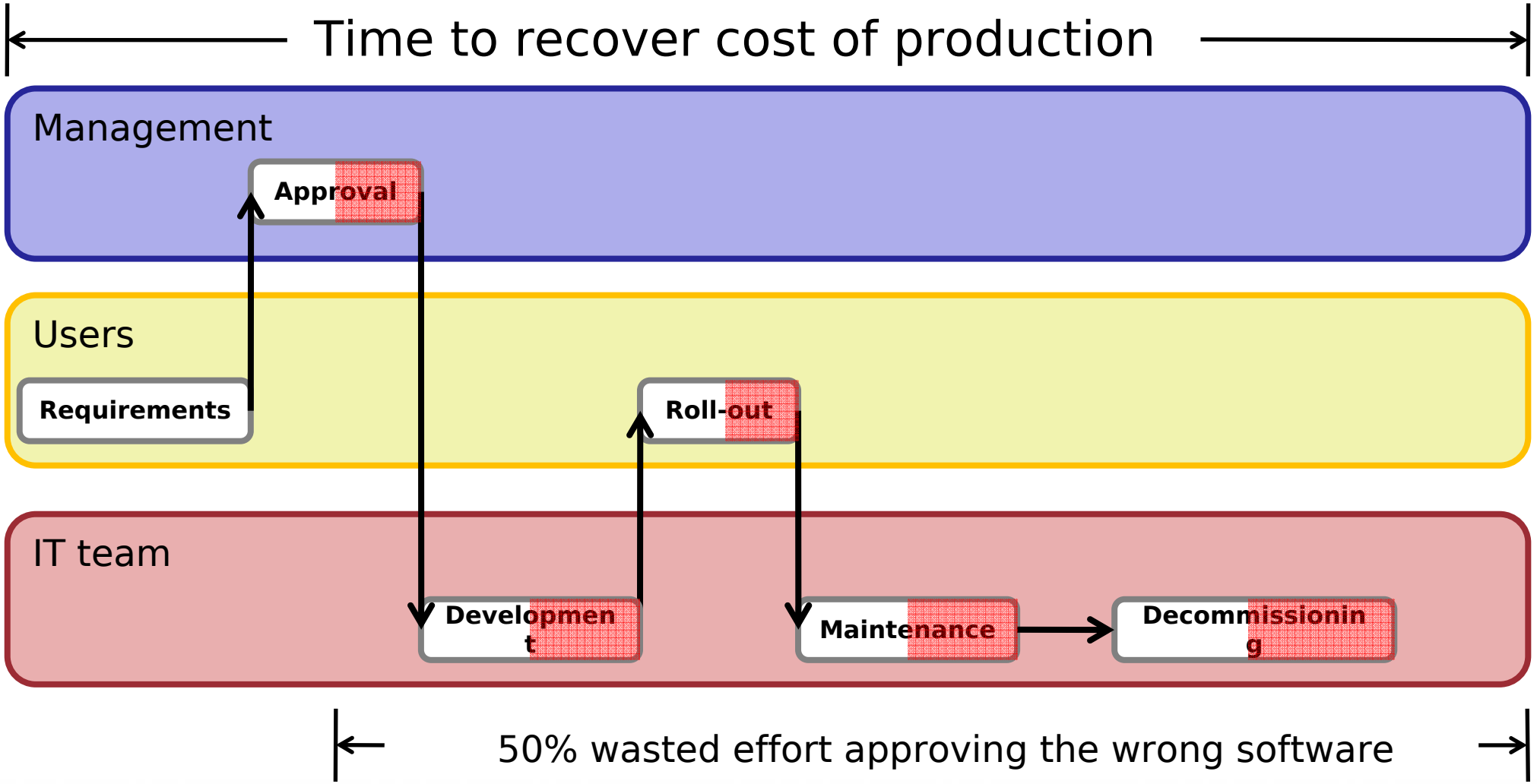
- Local inefficiency
  - Production of output of a single stage is inefficient
  - Output does not create inefficiencies for later stages
  - Example: decisions are delayed because it is hard to coordinate schedules of decision makers
- Global inefficiency
  - Output of a stage creates inefficiency for downstream stages
  - Example: Decision makers decide to make the wrong stuff
- Building the wrong software is the worst kind of waste

Local process inefficiency



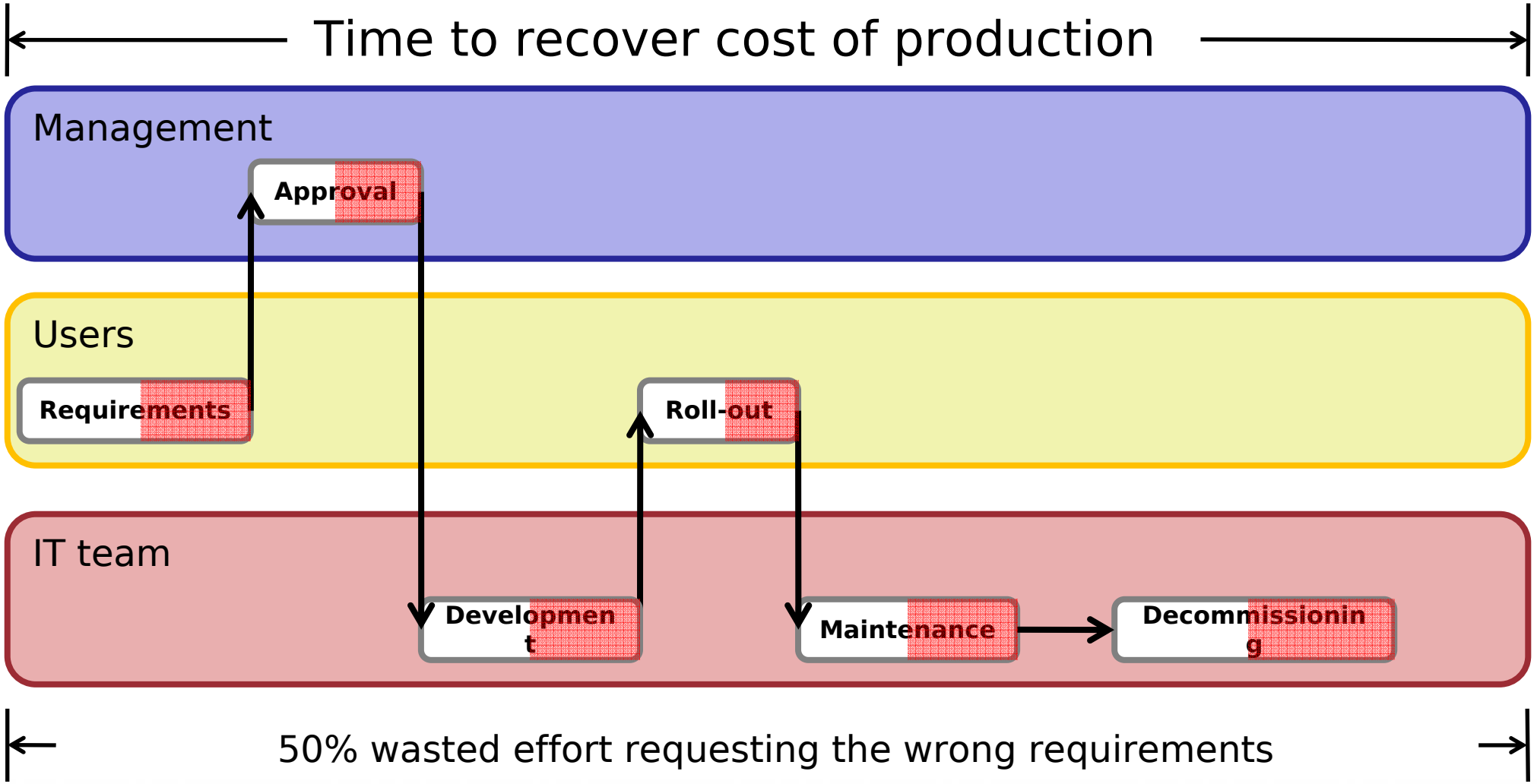
ThoughtWorks®

Global process inefficiency



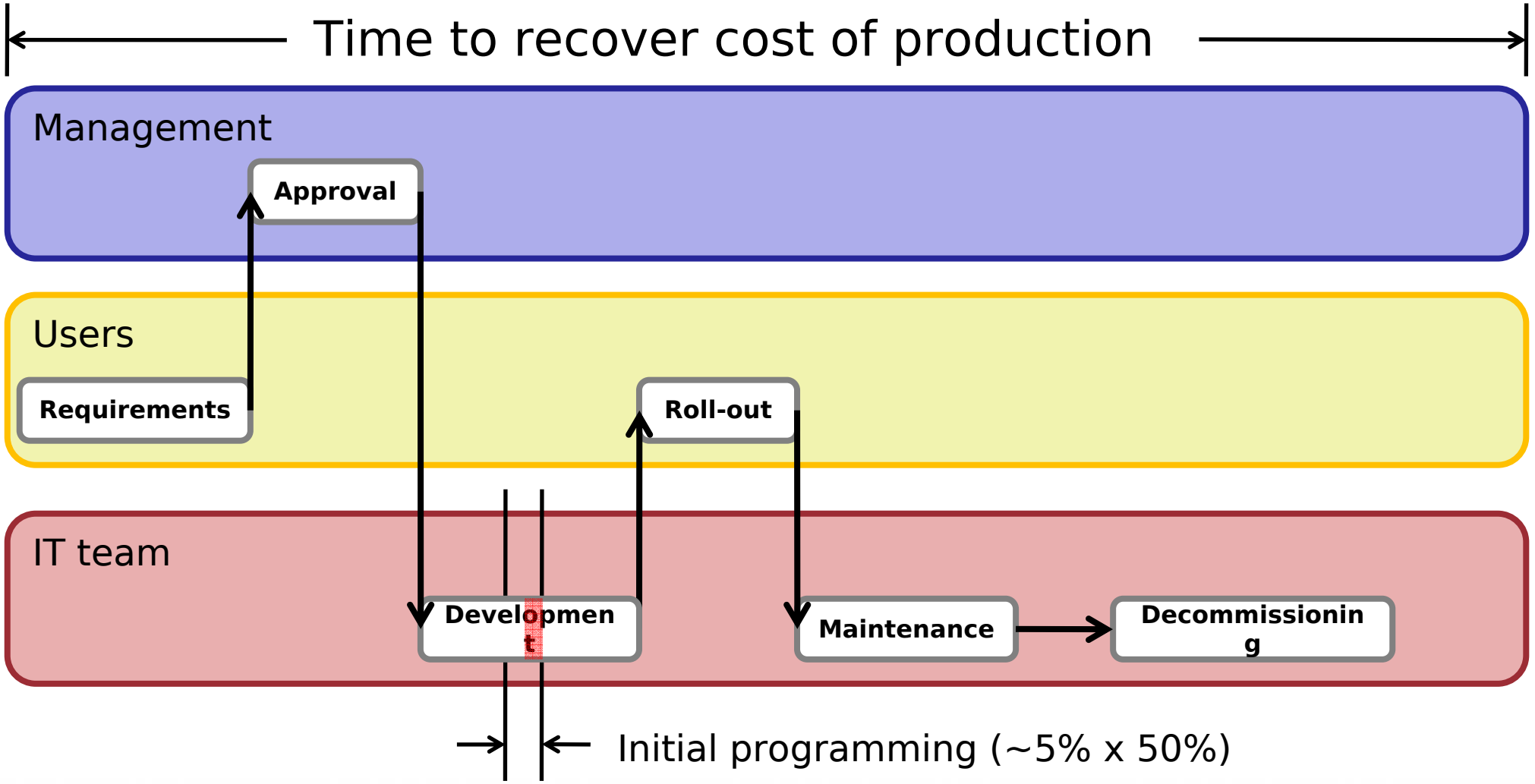
ThoughtWorks®

Global process inefficiency

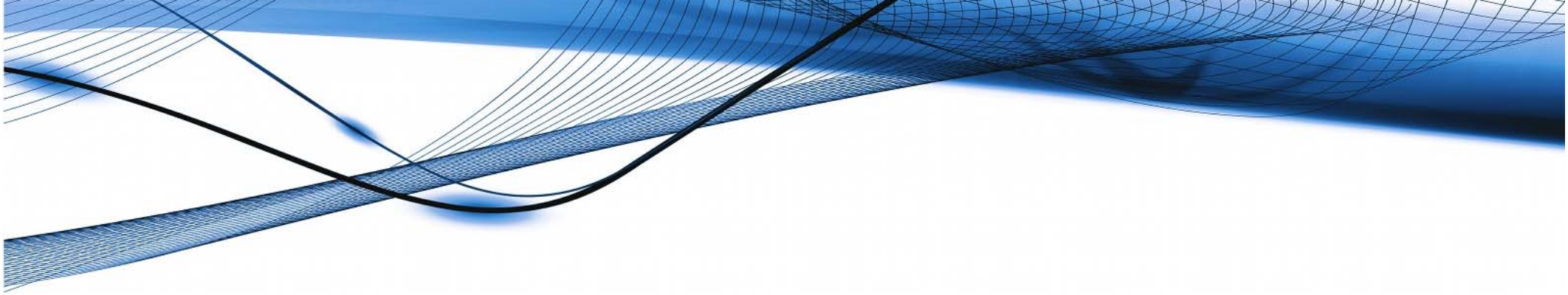


ThoughtWorks®

Local inefficiency in programming



ThoughtWorks®



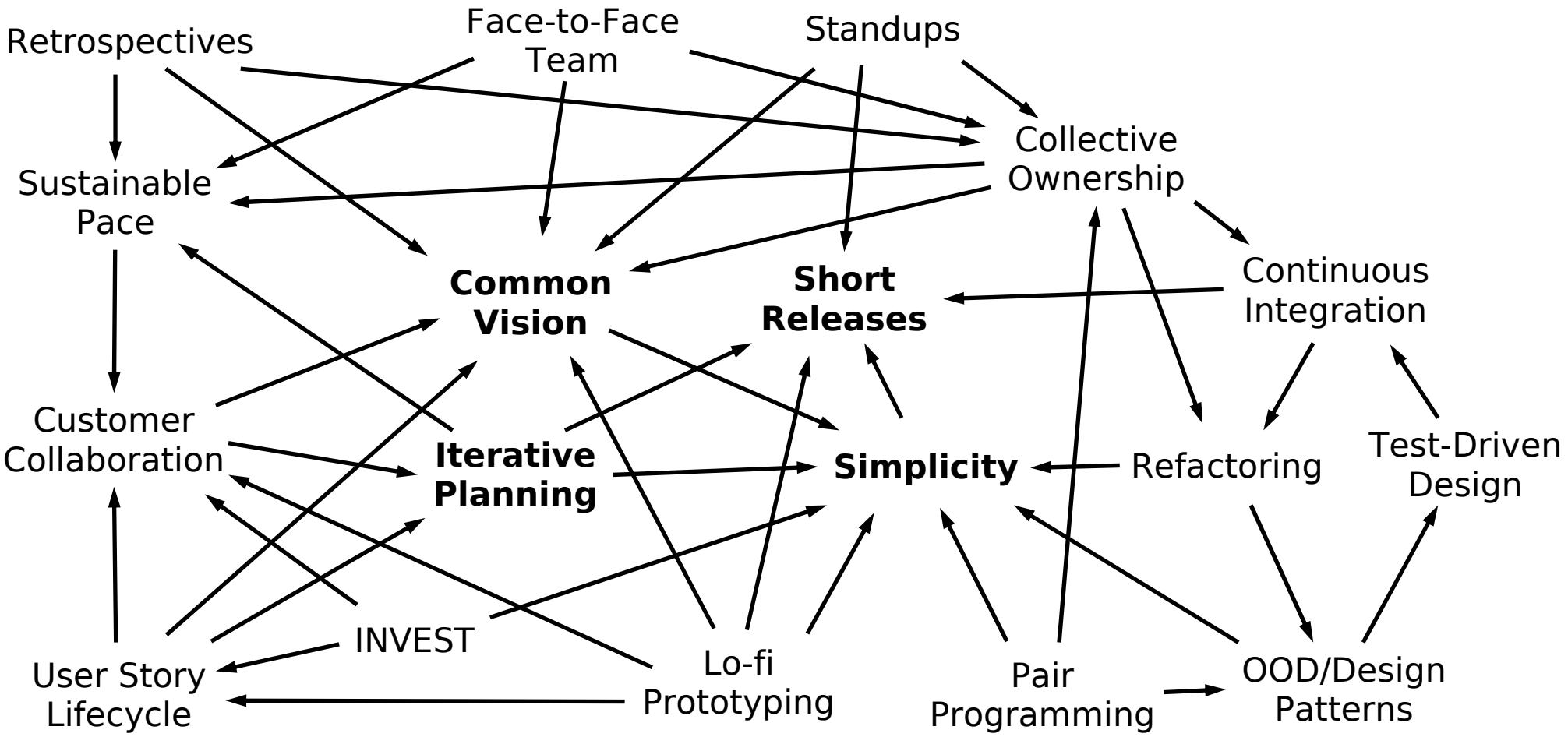
Agile Principles and Practices

ThoughtWorks©

Agile principles and practices

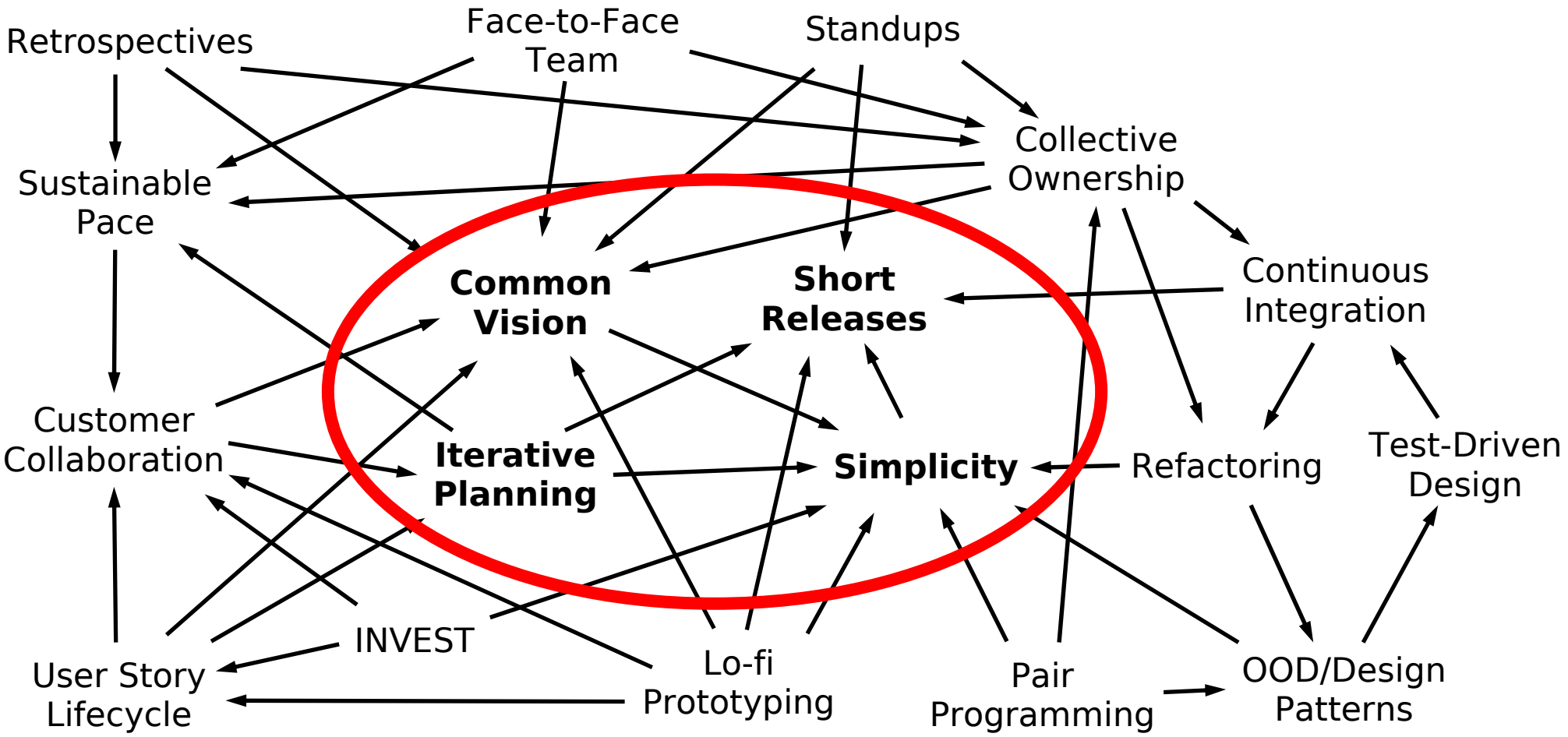
- Agile principles align with “Lean” principles
  - Accelerate feedback
  - Increase quality
  - Reduce waste
- “Efficiently produce high quality, high value software”
- Agile practices advance Agile principles
- Agile practices reinforce each other
- Agile practices interact with organizational practices
  - (for better, or for worse…)

Mutually reinforcing Agile practices



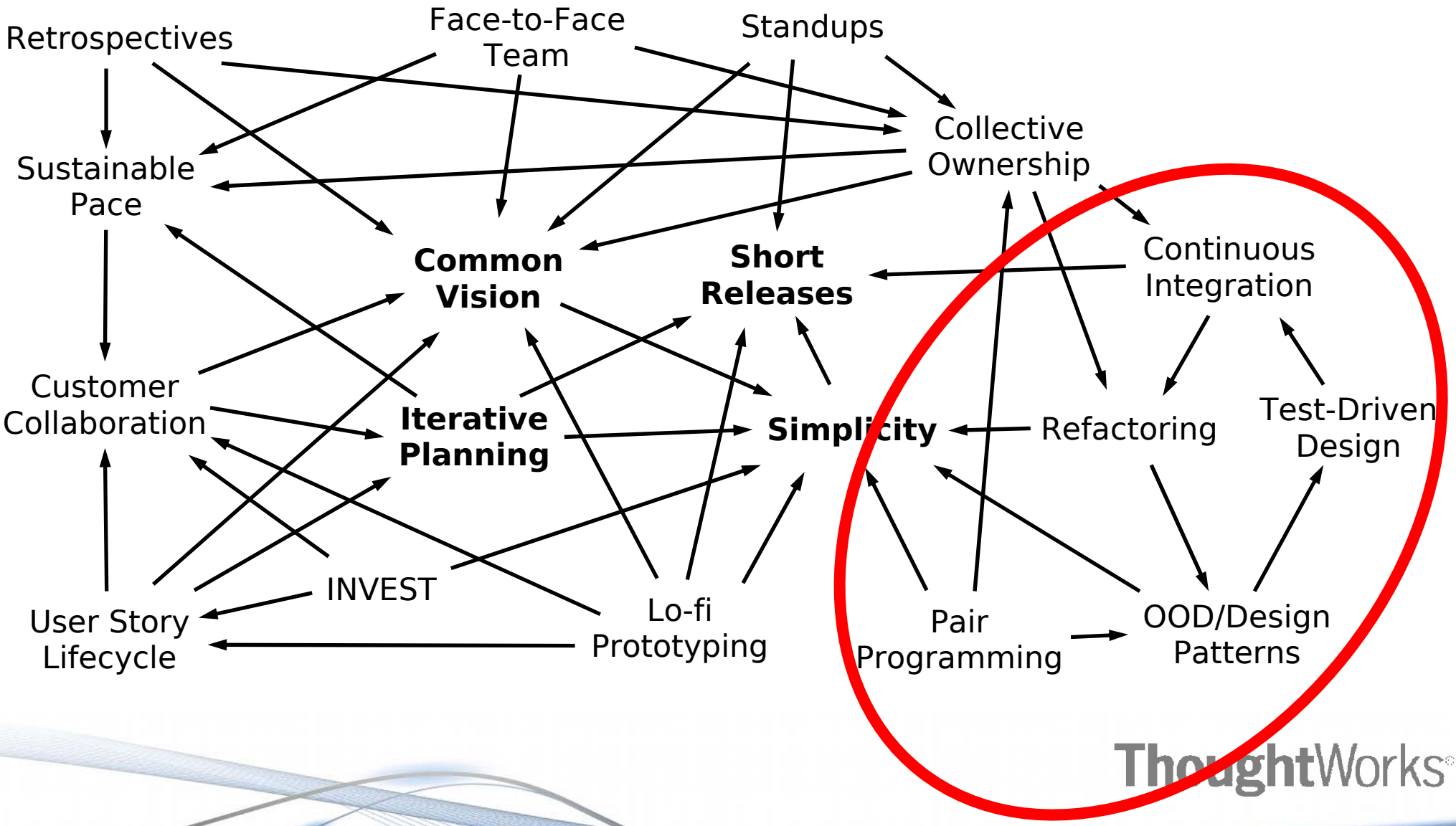
ThoughtWorks®

Core practices



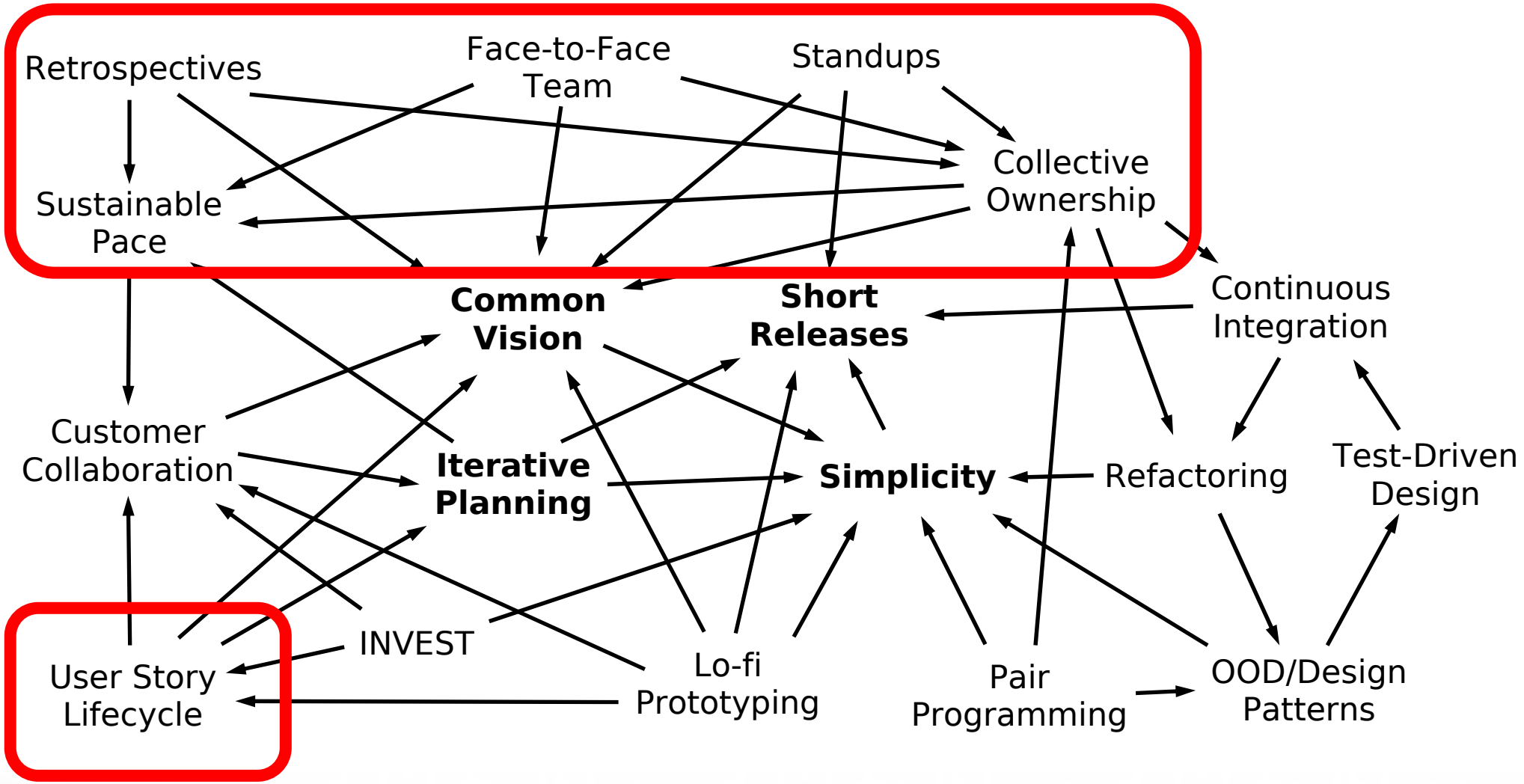
ThoughtWorks®

Development practices



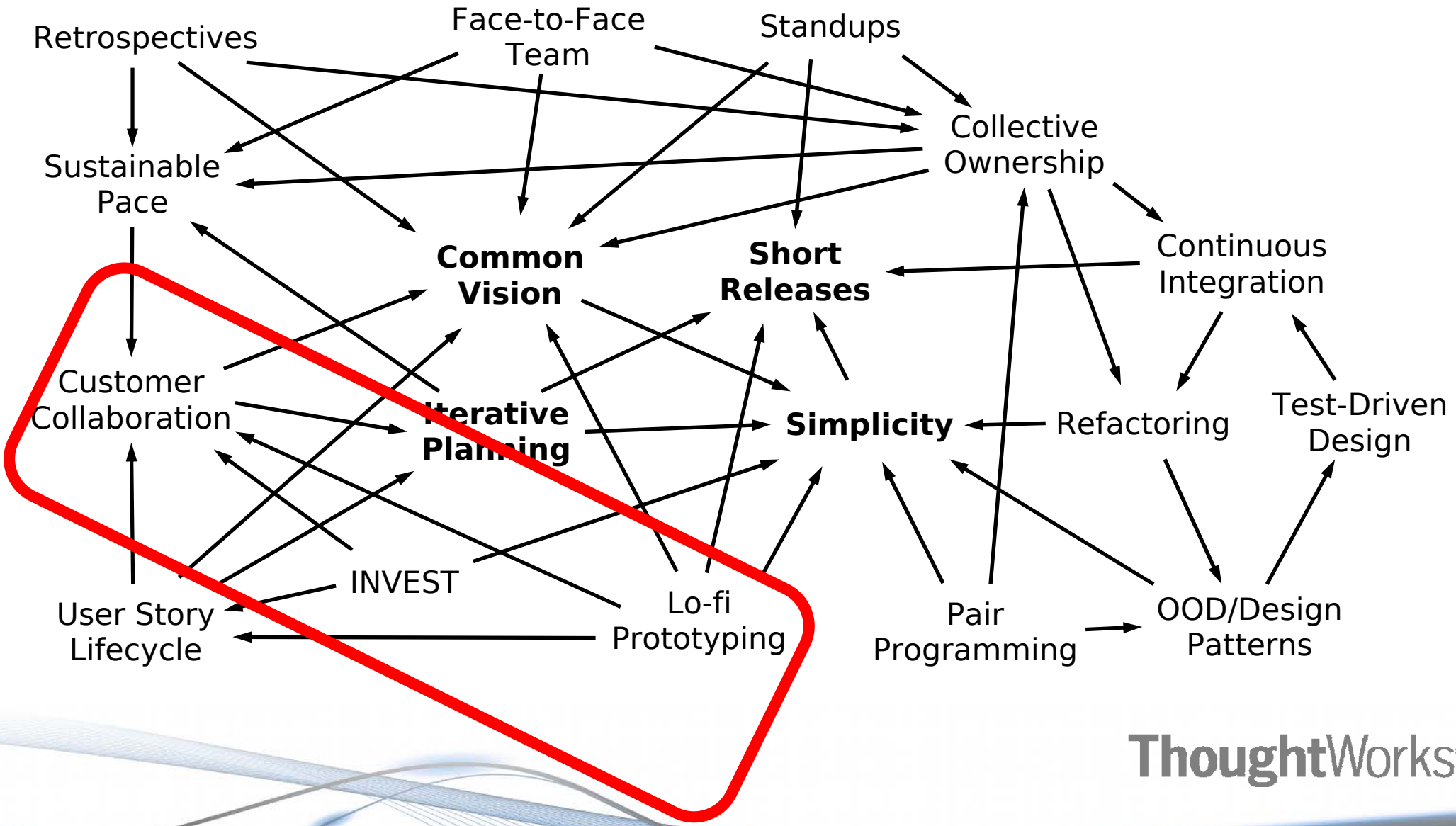
ThoughtWorks®

Project management practices



ThoughtWorks®

Business analysis practices



ThoughtWorks®

Agile Practice: Common Vision

What it is:

All key stakeholders (sponsors, users, developers) in the software development process iteratively converge on a common understanding of project objectives, constraints, and priorities.

How it works:

Stakeholders see, assess, and provide feedback following the iterative development cycle.

Multiple feedback and communication opportunities are built into the process.

Helps alleviate:

Wasted time and development resources due to conflicting understanding of problem, solution, or priorities amongst stakeholders.

Common adoption challenges:

Stakeholders are unable to commit or coordinate time for participation.

Stakeholders are unable to agree on project objectives, constraints or priorities.

Agile Practice: Short Releases

What it is:

Working software is released to production frequently during the course of the software development project.

How it works:

Project scope is decomposed into small, independently-releasable groupings.

Development practices focus on minimizing overall time to production-ready code.

Helps alleviate:

Wasted value of unused, but paid-for work-in-progress software.

Uncertainty of the suitability for purpose of work-in-progress software in the production environment.

Common adoption challenges:

Requirements gathering and analysis process is too cumbersome for frequent small releases.

Infrastructure or deployment processes are too cumbersome for frequent small releases.

Agile Practice: Iterative Planning

What it is:

Project planning is adapted iteratively based on new information collected over the course of the project.

How it works:

Project productivity ("velocity") metrics are collected as working software is completed, and used to refine earlier projections of scope and schedule.

Prioritization of scope is iteratively adjusted and replanned based on evolving stakeholder needs and cost-benefit analyses.

Helps alleviate:

Late surprises of over-schedule or over-budget delivery.

Wasted investment in low-priority or obsolete functionality

Common adoption challenges:

Sponsor organization requires longer planning and approval cycles.

Sponsor organization requires cumbersome replanning processes.

Agile Practice: Simplicity

What it is:

Software that is (relatively free) from waste, duplication, and complexity.

How it works:

Software development processes focus on producing only necessary code, just-in-time.

Software development processes focus on iteratively removing complexity from finished code.

Helps alleviate:

High cost of change for software in production.

High risk of change (fragility) for software in production

Common adoption challenges:

Sponsors are unwilling to budget for up-front costs of development practices supporting simplicity.

Software developers lack necessary training or design skills.

Requirements process requires up-front “over-engineering”.

ThoughtWorks®

Agile Practice: Customer Collaboration

What it is:

The project sponsors and end users are "part of the team", continuously involved in guiding the development process over the course of the project.

How it works:

Business analysts and project managers have frequent (daily), informal, face-to-face interaction with business stakeholders.

Communication between development team members and relevant business stakeholders is lightweight, ad hoc, and goal-specific.

Helps alleviate:

Wasted time or development effort due to incorrect or unclear understanding of actual business objectives, constraints or priorities.

Common adoption challenges:

Communication practices in the sponsor organization are too cumbersome.

Sponsor organization culture discourages integration of business stakeholders with development activities.

Business stakeholders have inadequate time to engage with development team.

ThoughtWorks®

Agile Practice: Standups

What it is:

A short, focused daily meeting for all development team members to communicate work status.

How it works:

Participants stand up, to encourage efficiency and focus of communication.

Each team member reports: yesterday's tasks completed, today's planned tasks, and any issues impeding progress.

Helps alleviate:

Misalignment of individual efforts with team objectives.

Wasted time due to stalled tasks.

Hidden risks or issues affecting project delivery.

Common adoption challenges:

Inconsistent focus on communication of relevant items.

Sidetracking.

Participation of team observers
("pigs vs. chickens")

ThoughtWorks®

Agile Practice: Face-to-face Team

What it is:

Software development team members work in a common space, face-to-face.

How it works:

A common space is provided containing all the supporting facilities for software development, and all the team members work together in that space, rather than spread out in separated areas or cubicles.

Helps alleviate:

Lack of "situational awareness" by team members of development status: task progress, new issues, etc.

Delays and miscoordination due to inefficient, high-latency communication.

Unfocused activity of individual team members

Common adoption challenges:

Sponsor organization lacks suitable facilities.

Sponsor organization culture does not support collective workspace practices.

ThoughtWorks®

Agile Practice: Retrospectives

What it is:

At the end of each iteration and release, team members and interested stakeholders review processes and lessons learned.

How it works:

Team members review what went well and what went less well during the just-completed delivery milestone.

Team members identify what practical improvements could be made in process or support to improve performance during the subsequent delivery milestone.

Helps alleviate:

Avoidable waste and delays due to process inefficiencies or lack of appropriate support.

Common adoption challenges:

Participants fail to accurately identify root causes and solutions.

Team individually or collectively fails to follow up on improvements identified during prior retrospectives.

Sponsor organization is unsupportive of improvements identified during retrospectives.

ThoughtWorks®

Agile Practice: Collective Ownership

What it is:

All development artifacts are the collective responsibility of the respective roles on the development team.

How it works:

Programmers share responsibility for all software, not just the software they wrote.

Business analysts and quality analysts share responsibility for all acceptance criteria and test plans, not just the ones they wrote.

Tools and processes support communication and coordination of activities to avoid conflicts..

Helps alleviate:

Development delays due to concentration of required tasks in areas “owned” by one individual (“bottlenecks”).

Delivery risk due to unavailability of an “owner” of a critical area (“bus factor”).

Common adoption challenges:

Team culture doesn’t support collective ownership and responsibility.

Tool and process support is inadequate, leading to inefficiencies and conflicts.

ThoughtWorks®

Agile Practice: Sustainable Pace

What it is:

The allocation of tasks to software development team members over the course of the project is continuously balanced with a healthy, sustainable workload.

How it works:

Iterative project metrics are used to measure the natural throughput of the development team.

Iterative planning is used to balance the workload with the team throughput.

Helps alleviate:

Inefficient team utilization due to
- unfocused task tracking
- task under-assignment
- "rush" production (low-quality output)
- over-assignment (burnout).

Common adoption challenges:

Project sponsors insist on a consistently unhealthy, unsustainable workload.

Undisciplined project management is unable to iteratively balance workload with throughput.

Undisciplined team processes fail to yield reliable project throughput metrics.

ThoughtWorks®

Agile Practice: User Story Lifecycle

What it is:

The functional scope of a software development project is decomposed into granular units, known as “stories”.

Each story passes through a series of processing stages, from story identification to production release.

How it works:

The software development process is organized as a story processing pipeline.

Each role has responsibility for processing a story at specific stages.

Pipeline throughput forms the basis for project metrics and planning.

Helps alleviate:

Underutilization of various roles at various stages of software development.

Inflexible planning or replanning of work in progress.

Large-scale surprises in required time or effort for planned scope.

Common adoption challenges:

Inadequate training or coordination of team members leads to ineffective handoff of pipeline stages.

Stories are not defined at an appropriate size for efficient processing.

ThoughtWorks®

Agile Practice: Low-fidelity Prototyping

What it is:

Rapid, iterative design of user interaction and user interface layout using whiteboard, pen-and-paper or computer drawing application.

How it works:

Business analysts and user interaction designers work collaboratively with end users to brainstorm, consolidate, and refine designs.

Once the low-fidelity prototypes are finalized, they are attached as acceptance criteria for the respective user stories, for the programmers to reference in their implementation.

Helps alleviate:

Wasted effort developing application functionality or interaction that doesn't address immediate end-user requirements or expectations.

Hard-to-use (i.e. expensive to train) user interfaces for developed software.

Common adoption challenges:

Insufficient skills or training for business analysts and user-interaction designers.

Inadequate expectation management of the (large) gap between a completed prototype and completed software.

Agile Practice: INVEST

What it is:

All stories entering the software development pipeline satisfy the INVEST criteria: Independent, Negotiable, Valuable, Estimable, Small, Testable

How it works:

Business analysts work with business stakeholders to decompose project scope into stories that satisfy INVEST, and use the INVEST criteria as the basis for writing story acceptance criteria and test planning.

Helps alleviate:

Inefficient development effort for stories that are too big, too small, too vague, or unimportant.

Confusion and misunderstanding about the intent of a story in development.

Common adoption challenges:

Insufficient skills or training for business analysts.

Business stakeholders are unable to communicate INVESTable requirements with business analysts.

Agile Practice: Pair Programming

What it is:

Programmers work together in pairs on programming tasks, using a shared development workstation.

How it works:

A pair of programmers signs up for a development task.

At any given time, one programmer has control of the mouse and keyboard, and the other observes and provides feedback.

Control alternates periodically.

Helps alleviate:

Wasted time and development effort due to simple mistakes.

Concentrated “ownership” of code.

Unclear or avoidably complex software design or implementation.

Common adoption challenges:

Team culture does not support pair programming.

Inadequate or inappropriate physical facilities for pair programming.

Inappropriate pair balancing or assignment.

ThoughtWorks®

Agile Practice: Refactoring

What it is:

Programmers improve the internal implementation or organization of software without changing its functional behavior.

How it works:

During the course of development, as new features are added to software, earlier design or implementation decisions may prove inappropriate.

When this occurs, programmers make iterative modifications, verified by tests, toward a more appropriate design or implementation, reusing as much of the original code as practical.

Helps alleviate:

Wasted programmer effort due to working with an accumulated code base that is excessively complicated or fragile.

Slow time to production due to “fix one bug, add one bug” phenomenon.

Common adoption challenges:

Programmers lack appropriate skills or training.

Software code base lacks comprehensive test coverage to allow for safe refactoring.

Project sponsor is unwilling to allocate resources for refactoring.

ThoughtWorks®

Agile Practice: OOD / Design Patterns

What it is:

Programmers organize the design and implementation of software according to standard design principles.

How it works:

Programmers use Object-Oriented Design (OOD) to segregate and specify the responsibilities (concerns) of the code in alignment with the underlying business functionality.

Programmers use design patterns as well-understood, “best practice” solutions to common problems they encounter.

Helps alleviate:

Fragile, complicated, hard-to-modify software.

Hard-to-understand software.

Wasted effort in “reinventing the wheel”.

Common adoption challenges:

Programmers lack appropriate skills or training.

An unsustainable pace leads to rushed implementations with unclean design.

Agile Practice: Test-Driven Design

What it is:

Developers define the functionality of the system in the form of automated tests.

The design of the system is driven by the natural flow and structure of these automated tests.

How it works:

Developers first translate and formalize the story acceptance criteria into automated tests, then implement software to make the tests pass. opportunities are built into the process.

Helps alleviate:

Hard-to-test, hard-to-modify software.

Software with excessive defects.

Common adoption challenges:

Programmers lack appropriate skills or training.

Project sponsors are unwilling to invest in comprehensive automated tests.

ThoughtWorks®

Agile Practice: Continuous Integration

What it is:

Automatic tools continuously verify the software in development is working and functionally correct.

How it works:

As programmers commit modifications to the software, a software agent detects the modification, and automatically builds and integrates the modifications with the complete system, and runs a suite of tests verifying system integrity.

Helps alleviate:

Wasted time and development resources due to tracking down the modification responsible for broken functionality.

Delays releasing to production due to long post-development testing periods.

Common adoption challenges:

Inadequate tool or environment support.

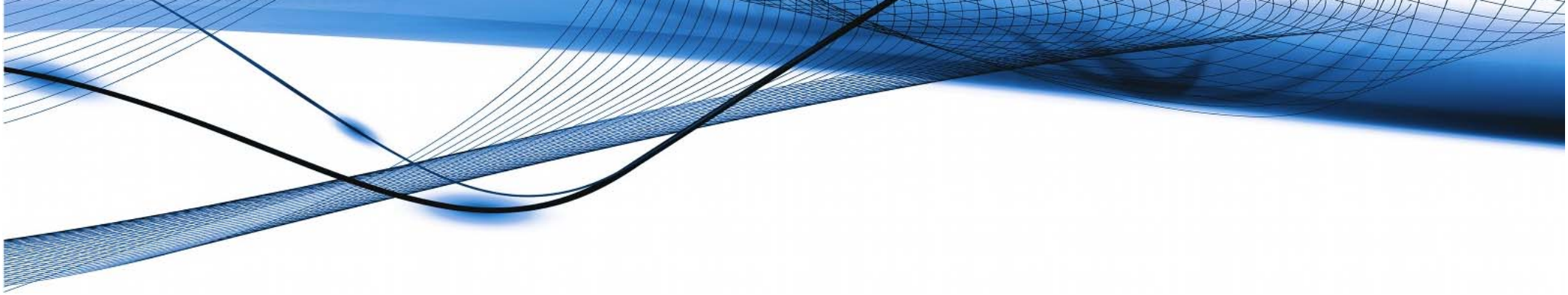
Poor programmer discipline for test writing and maintenance.

ThoughtWorks®

Exercise: Agile Adoption Planning

- As a group, select three Agile practices for StarHub pilot implementation (5 min.)
  - Prioritize on cost-benefit basis
- For each of the three selected Agile practices, develop an implementation plan (10-15 minutes), including:
  - Resources required
  - Time required
  - Adoption challenges
  - Required participants
- Each group selects one person to summarize plan (2-5 minutes each)

ThoughtWorks®



Introduction to Agile Enablement and Organizational Transformation

ThoughtWorks©

Process improvement overview

- Assess the “as-is” process
- Identify small, specific process improvement opportunities (“stories”)
- Prioritize process improvement stories according to business value and implementation effort
- Make a plan
- Implement the selected improvements, adjusting the plan iteratively
- Review and reassess

Agile process assessment

ThoughtWorks® has developed an assessment model that evaluates the capability of a software development process along a number of dimensions of interest. This framework provides a comprehensive overview of all aspects of Agile practice.

These dimensions include:

- Testing
- Configuration Management
- Shared Responsibility
- Collaboration
- Responsiveness
- Requirements
- Simplicity
- Governance
- Lean Alignment

ThoughtWorks®

Testing

Tests provide the safety net that allows Agile projects to proceed at a rapid pace.

Commitment to testing is reflected in the systematic maintenance of comprehensive automated test suites, and vigilant remediation of regressions that cause tests to fail.

Configuration Management

Large, rapid changes of a common code base by multiple developers are a normal characteristic of Agile software development.

A Configuration Management system of tools and practices should support large, efficient modifications without “breaking the build”.

Shared Responsibility

Team flexibility and cooperation support an efficient and resilient process. Pigeon-holed knowledge or skills are potential bottlenecks or single points of process failure

Pair programming and “just-in-time” work assignments are signs of the extent to which the team, not individuals, owns the work.

Collaboration

Communication and collaboration among project stakeholders allows rapid and accurate delivery of business value.

A collaborative development process is supported by co-location, tools, and other practices. At the highest levels, there is continuous involvement of the users and business sponsors.

Responsiveness

The term “Agile” originally referred to the ability of the customer to quickly modify their requirements in response to changing business circumstances.

Responsiveness measures the speed and quality with which requirements changes are accommodated.

Requirements

Requirements are an explicit definition of business value.

The software development process delivers the greatest business value when requirements are developed with the participation of actual users and prioritized by value on a just-in-time basis.

Simplicity

As the design and implementation of the software becomes more simple, incremental changes to the functionality become more efficient and less risky.

This produces greater flexibility in adapting to changing business requirements, which means less (potentially wasted) investment needs to be made in attempting to anticipate such requirements in advance.

Governance

The more adaptive the project planning and management practices, the more closely the software development process will be able to track changing business priorities and accommodate unforeseen challenges or opportunities.

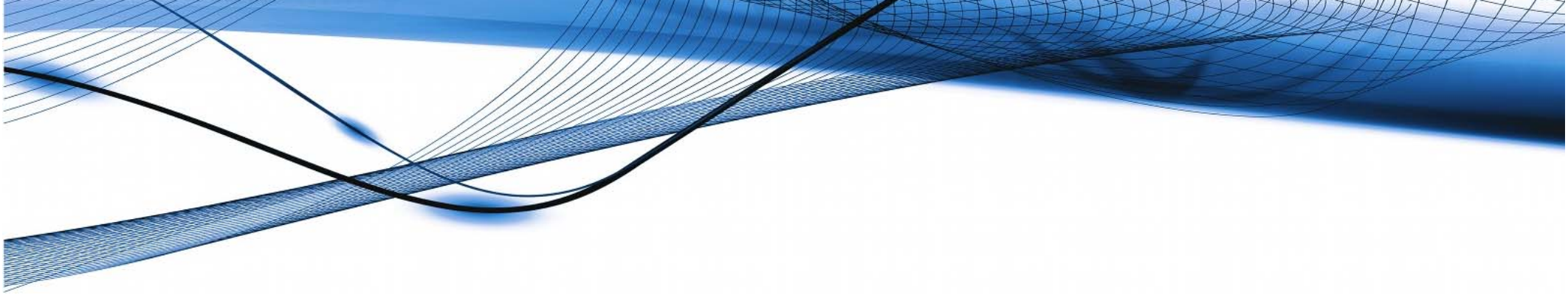
The ultimate objective is the efficient real-time integration of the software development process with business planning and management.

Lean Alignment

Agile software development practices complement lean management objectives.

Waste in the software development process may cause, and may be caused by, waste elsewhere in the end-to-end business value stream.

Systematically identifying, tracking, and eliminating such waste improves overall organizational productivity.



Thank you!

www.thoughtworks.com

ThoughtWorks©