Achieving Agility at Scale
Improving Software Economics
Managing Innovation

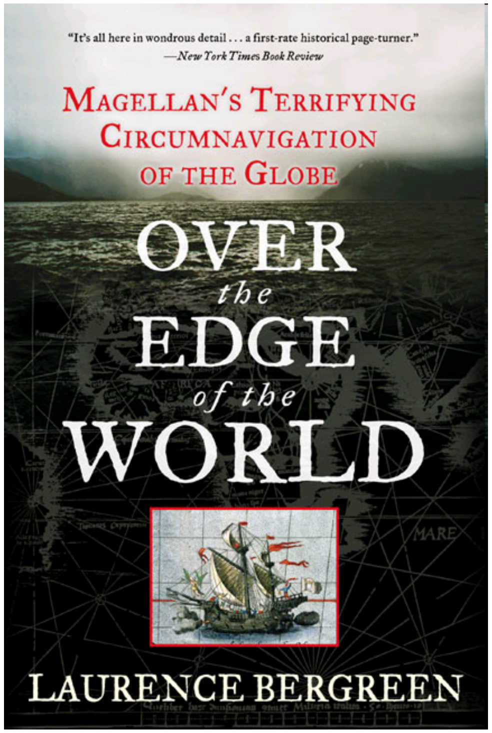
Martin Nally
CTO IBM Rational, IBM Fellow

Executive Summit Exec02

The nature of innovation

- Innovation is not the predictable realization of a new idea
- Innovation is a risky plunge into the unknown, but with a clear objective and a strong sense of general direction, a key idea

What I read on my vacation


"It's all here in wondrous detail . . . a first-rate historical page-turner."
—New York Times Book Review
Magellan's Terrifying Circumnavigation of the Globe
OVER the EDGE of the WORLD
MARE
LAURENCE BERGREEN

- Magellan knew exactly what he wished to achieve and his general plan for getting there
  - Find a passage through South America
  - Claim the Moluccas (Slice Islands) for Spain
- There were many unknowns and risks along the way
- There was no detailed plan – he steered as he went

- How would Magellan have managed a software project?

The project model

Standard model	Modern model
Distinct development phase	*Continuously evolving systems*
Distinct handoff to maintenance	*No distinct boundary between development and maintenance*
Requirements-design-code-test sequence	*Sequence of released capabilities with ever increasing value*
Phase and role specific tools	*Common platform of integrated process / tooling*
Collocated teams	*Distributed, web based collaboration*
Standard engineering governance	*Economic governance tailored to risk / reward profiles*
Engineering practitioner led	*Business value and outcome led*

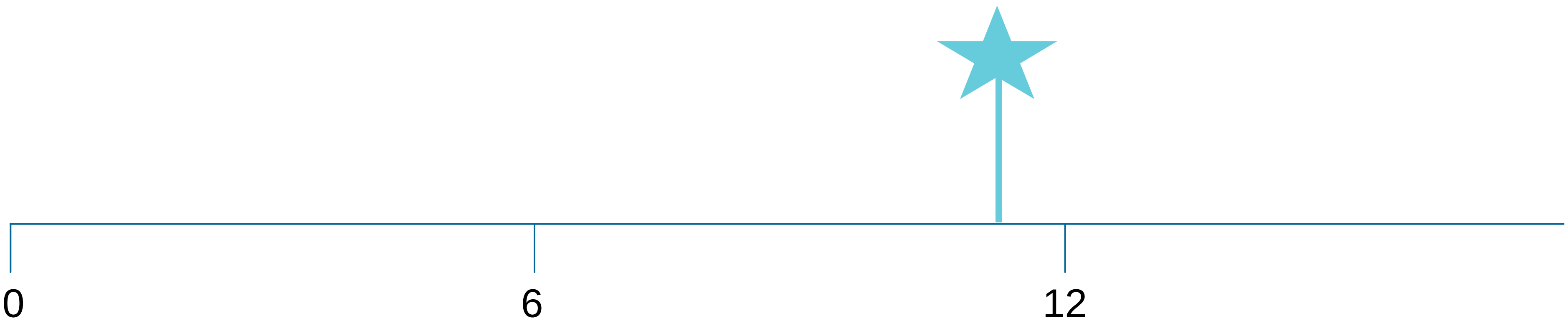
Critical culture shifts

Conventional Governance	*Agile Governance*
Activity-based management Mature processes, PMI/PMBOK Plan in detail, then track variances	**Results-based management** More art than engineering Plan/steer/plan/steer…
Adversarial relationships Paper exchange, speculation	**Honest collaborative communication** Progressions/digressions, facts
Requirements first Assumes certainty in desired product Avoid change	**Architecture (*risk mitigation*) first** Admits uncertainties Manage change
Early false precision "More detail = higher quality"	**Evolving artifacts** Scope (Problem specs) Design (Solution specs) Constraints (Planning specs)
Apply too much or too little process Process is primary, blind adherence	**Right-size the process** Desired results drive process Manage variances

Schedule risk: Imagine you have 12 months to deliver a business critical system

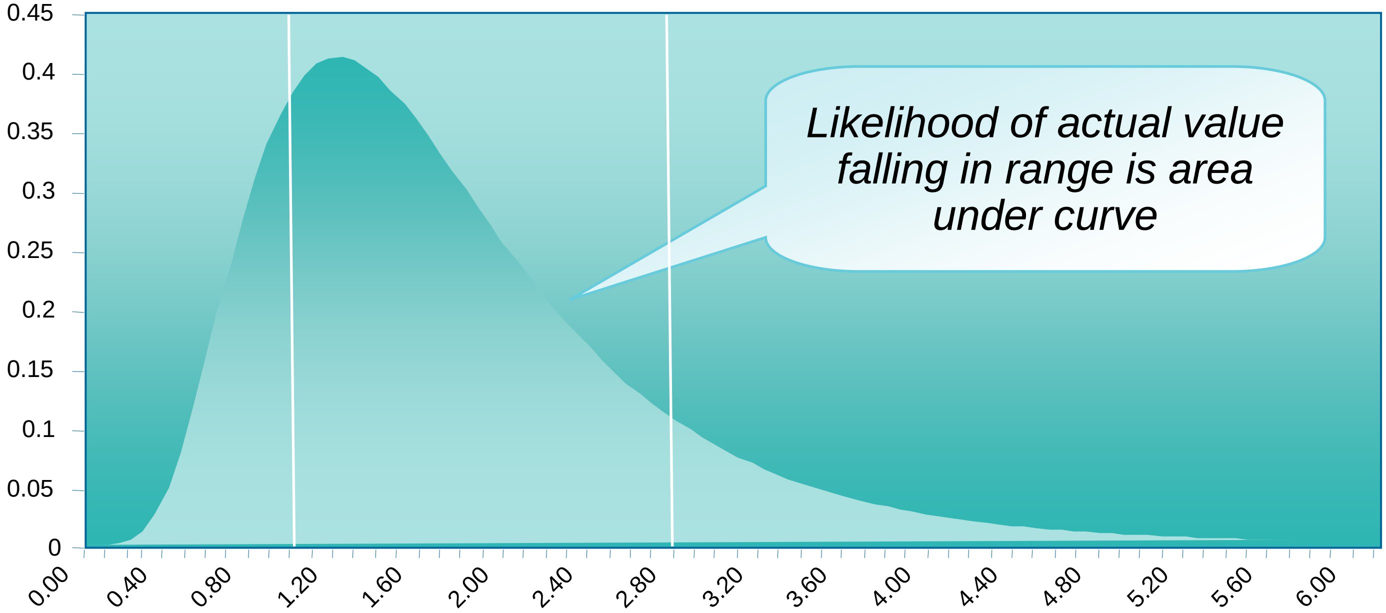
- Your estimators tell you it will be done in 11 months

- What do you do with the information?
  - Rest easy, believing there is no risk?



Maybe you realize that program parameters (cost, schedule, effort, quality, …) are random variables

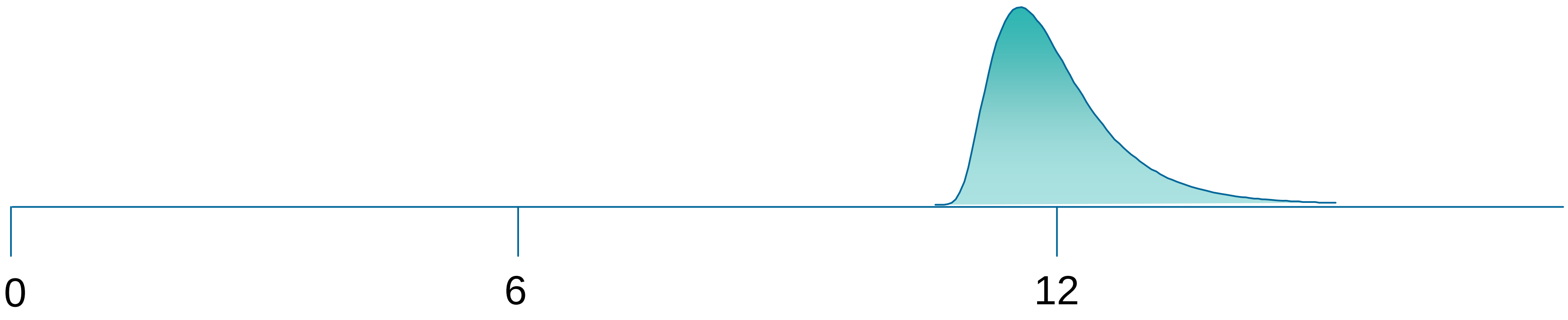
- Area under curve describes probability of measurement falling in range



Likelihood of actual value falling in range is area under curve

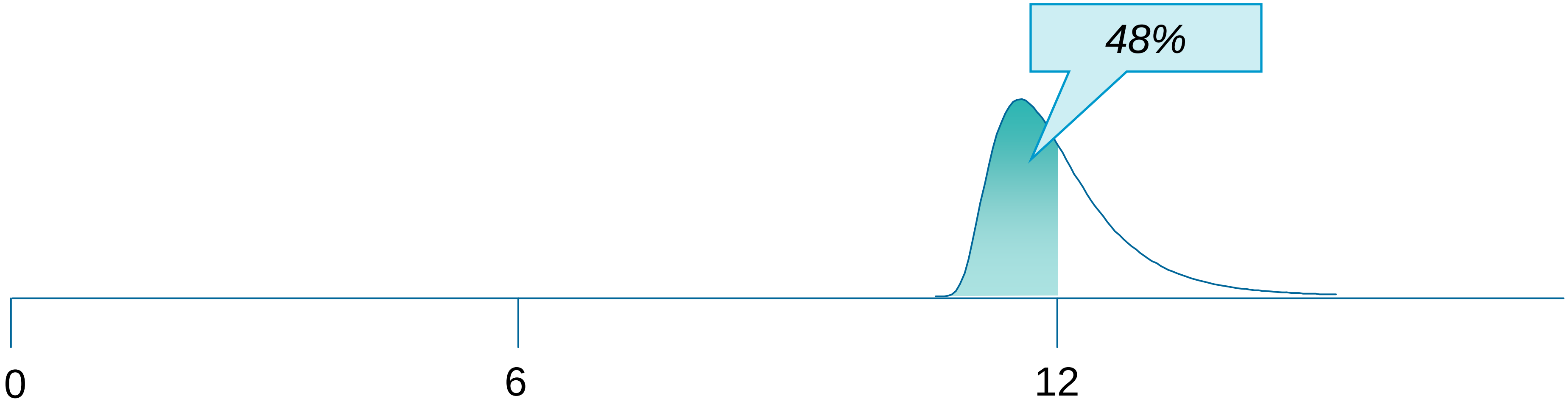
Imagine you have 12 months to deliver a business critical systems

- So you ask for the distribution and discover there is some uncertainty



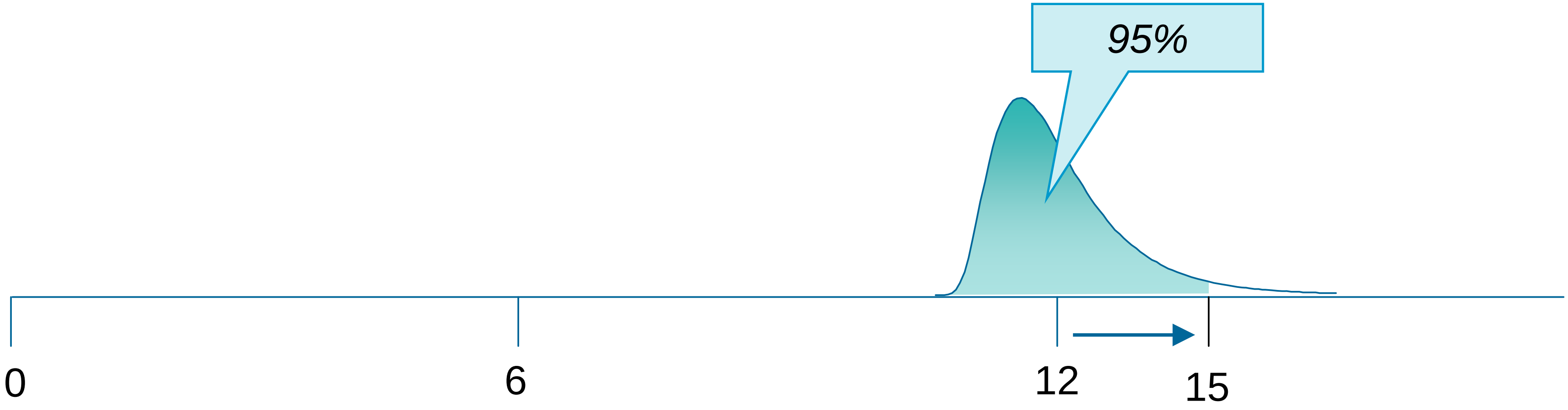
Imagine you have 12 months to deliver a business critical systems

- In fact there is less than 50% chance of making the date



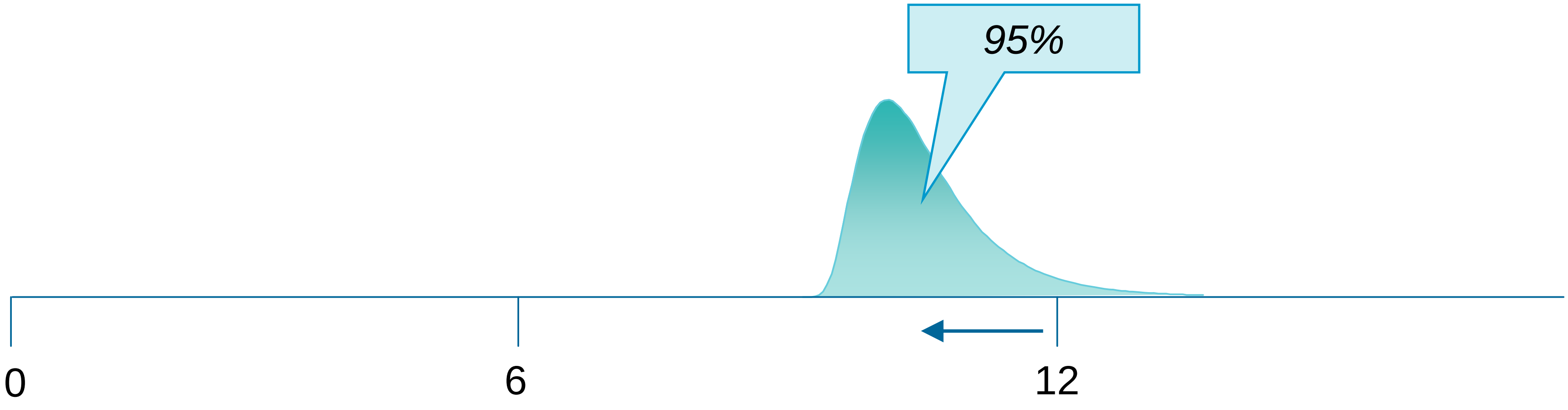
Then what?

- Move out the date to improve likelihood of shipping?



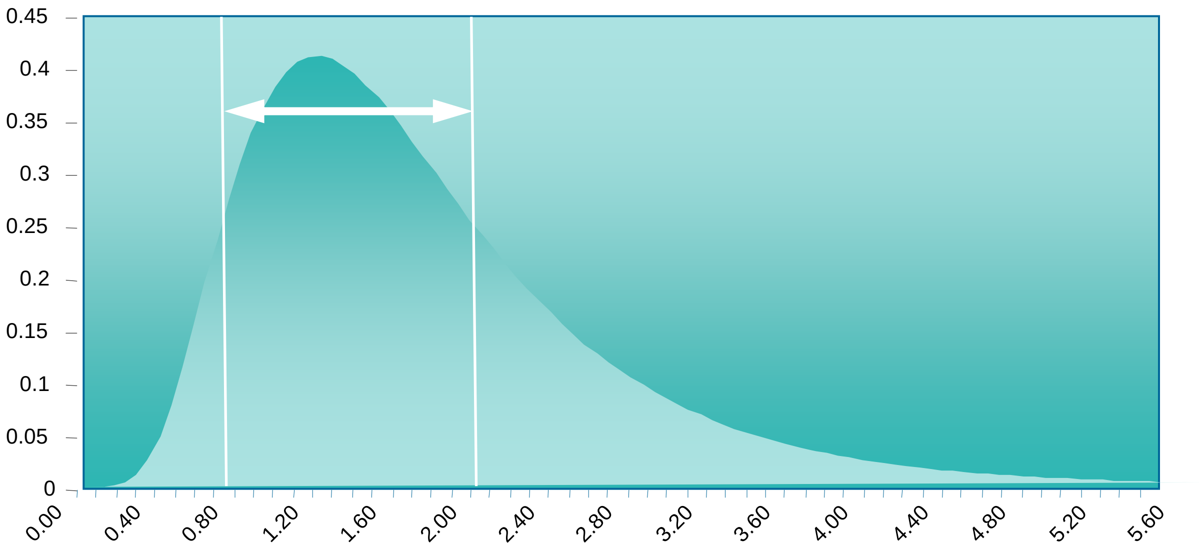
Then what?

- Or move in the estimate by sacrificing quality or content?



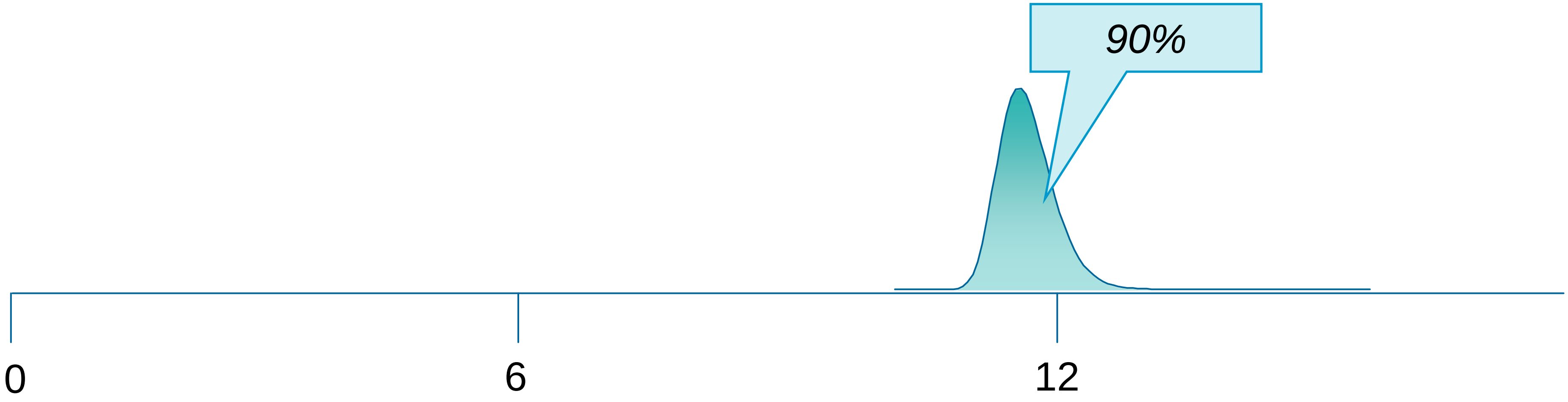
Managing variances in scope, solution, plans: The real key to improving software economics

- Sources of uncertainty and variance
  - Lack of knowledge
  - Lack of confidence
  - Lack of agreement
- Reduction of variance reflects
  - Increased predictability of outcome
  - Increased knowledge about
    - Client needs
    - Technology capability
    - Team capability
  - Good decisions



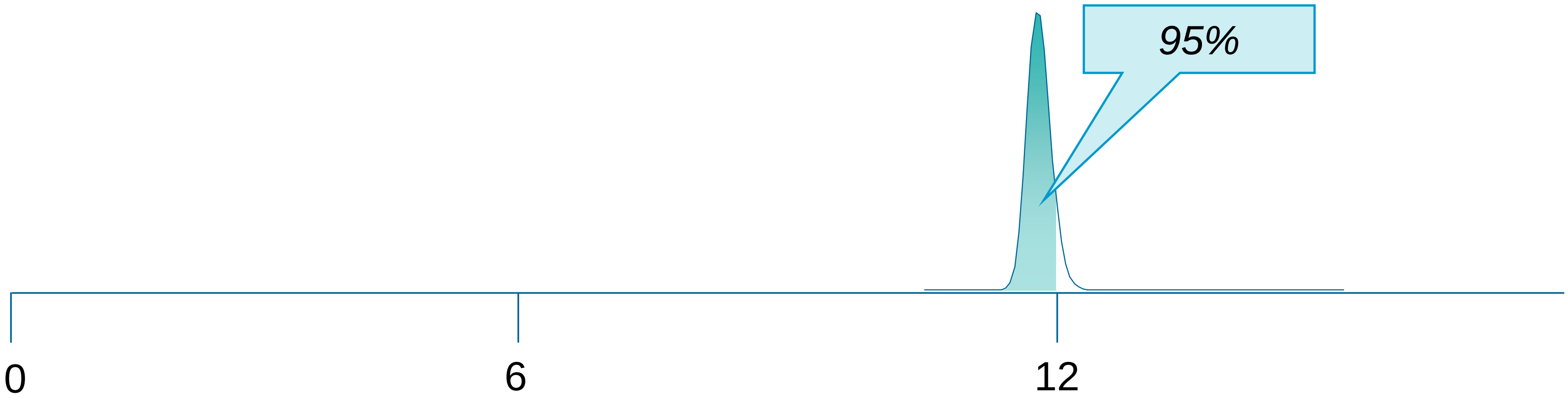
Then what?

- Determine the source of the variance
- Over the project lifecycle, reduce the variance to improve likelihood of shipping



Then what?

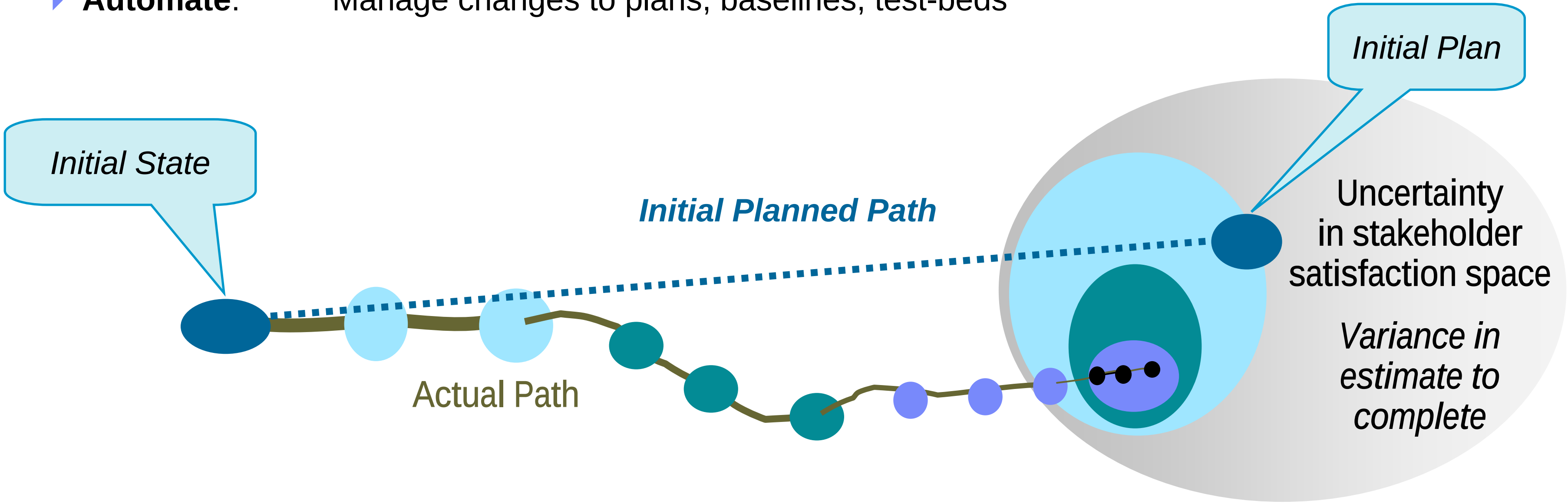
- Over the lifecycle, reduce the variance further to improve likelihood of shipping



Measure and steer

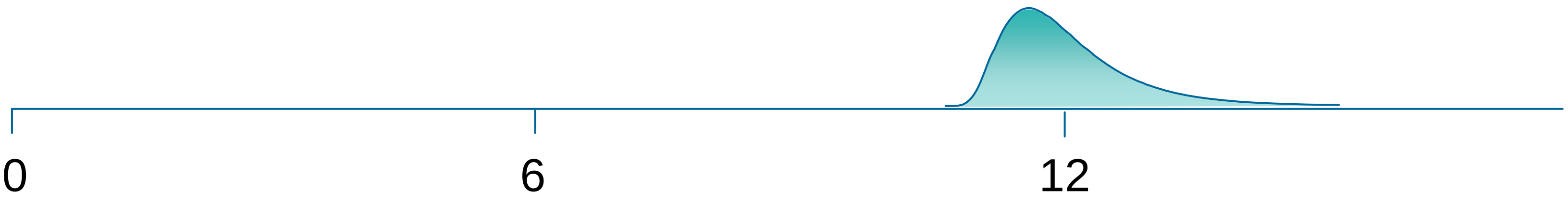
- At onset of program
  - **Report:** Establish estimates/variances of effort, cost, establish initial plan
  - **Collaborate**: Set initial scope and expectations with stakeholders
  - **Automate**: Establish a collaborative development environment

- At each iteration, improve estimates and report
  - **Report**: Values and variances of progress achieved, quality achieved, resources expended
  - **Collaborate**: With stakeholders to refine scope and plans
  - **Automate**: Manage changes to plans, baselines, test-beds

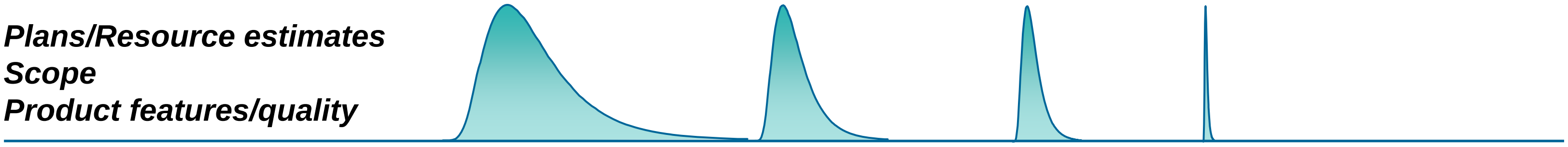

Initial Plan
Initial State
Initial Planned Path
Uncertainty in stakeholder satisfaction space
Variance in estimate to complete
Actual Path

Agile Governance = Managing Uncertainty = Managing Variance

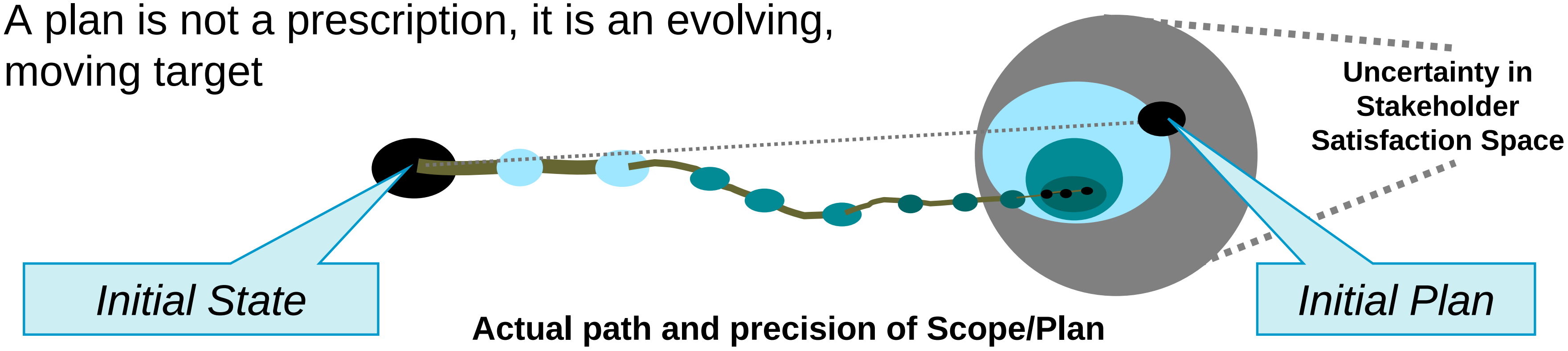
- A completion date is not a point in time, it is a probability distribution



- Scope is not a requirements document, it is a continuous negotiation


Plans/Resource estimates
Scope
Product features/quality

- A plan is not a prescription, it is an evolving, moving target

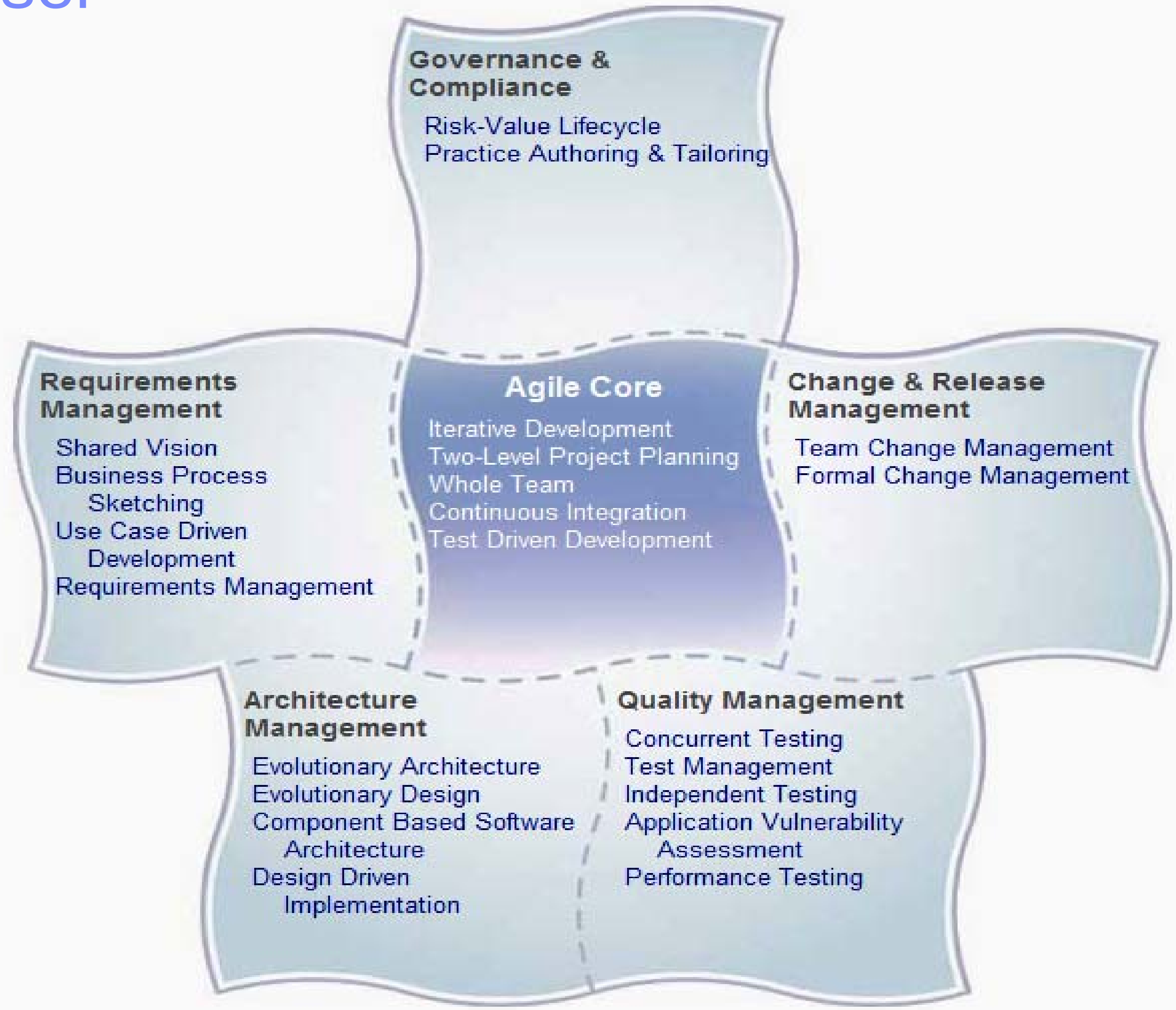

Uncertainty in Stakeholder Satisfaction Space

Initial State

Initial Plan

Actual path and precision of Scope/Plan

Practices included as part of Rational Method Composer



Governance & Compliance

- Risk-Value Lifecycle
- Practice Authoring & Tailoring

Requirements Management

- Shared Vision
- Business Process Sketching
- Use Case Driven Development
- Requirements Management

Agile Core

- Iterative Development
- Two-Level Project Planning
- Whole Team
- Continuous Integration
- Test Driven Development

Change & Release Management

- Team Change Management
- Formal Change Management

Architecture Management

- Evolutionary Architecture
- Evolutionary Design
- Component Based Software Architecture
- Design Driven Implementation

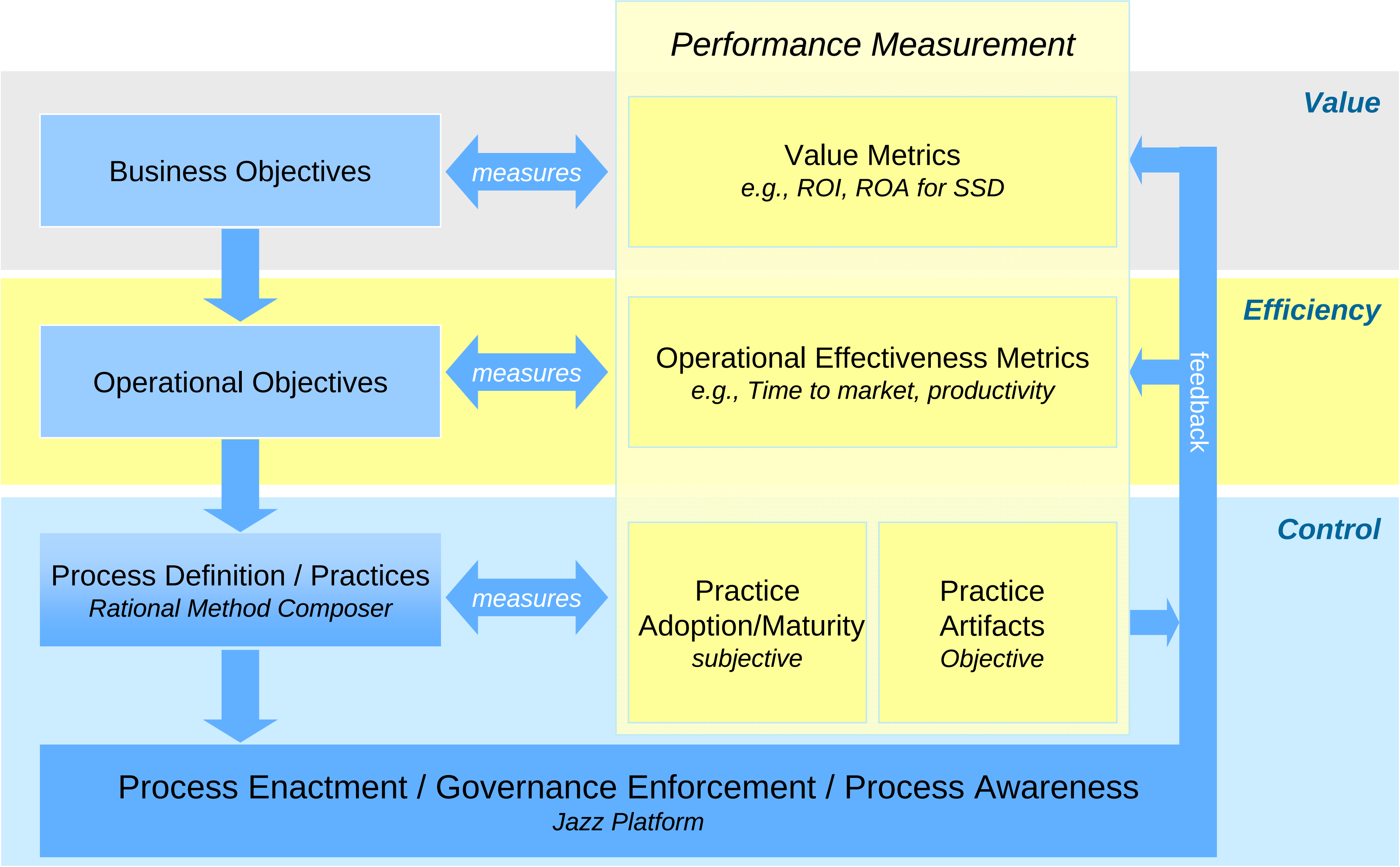
Quality Management

- Concurrent Testing
- Test Management
- Independent Testing
- Application Vulnerability Assessment
- Performance Testing

Agenda

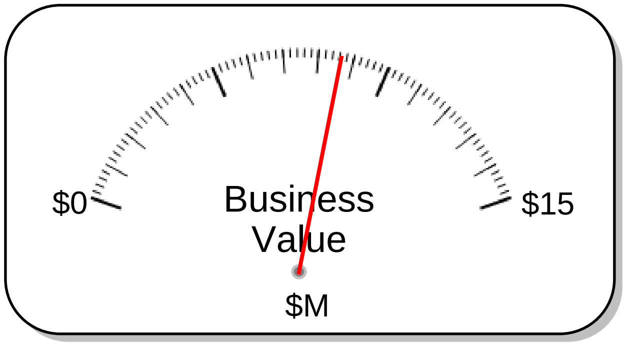
- Transitioning to Agile Software Delivery
  - Economic governance
  - Steering and managing uncertainty
- **Metrics and Measures**
  - **Instrumentation for gaining control, improving efficiency and optimizing value**
- Improving Software Economics
  - A framework for improvement priorities

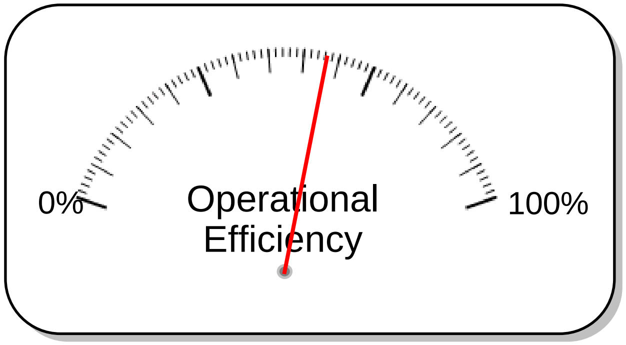
Software and systems need a control framework

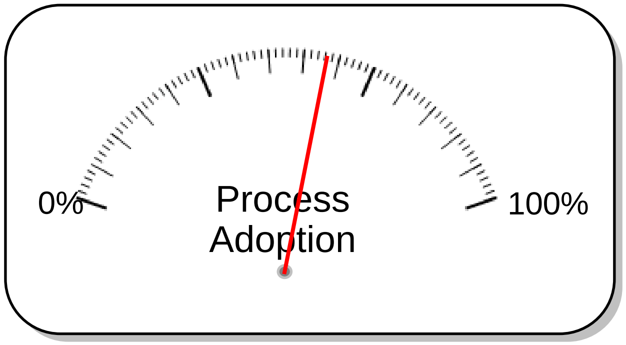


Meters for software and systems development and delivery improvement

- Value
  - Return on Investment (ROI)
  - Return on Assets (ROA)
  - Product revenue profile
- Efficiency
  - Time to market, productivity
  - Program portfolio investment profile
  - Defect phase containment, scrap and rework rates
  - Application service levels
  - Defect densities, requirements churn, design churn
  - Skills improvement, training cost reduction
- Control
  - Practice adoption, project checkpoints
  - Artifact time between gates
  - Collaboration, skills mix





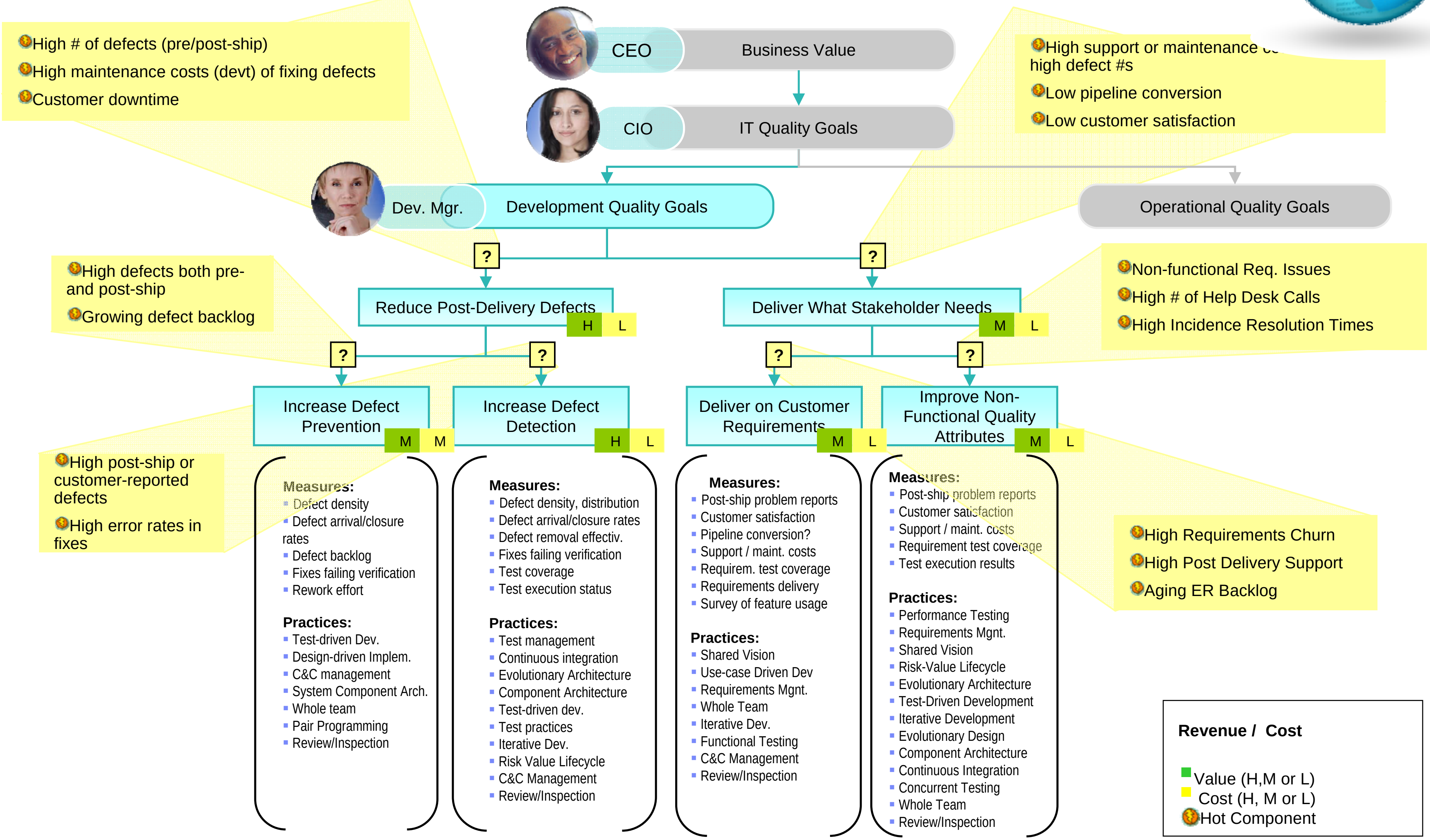


Tailor to organizational and project context

- Agree on business value measures: Cost, profit, return on assets, market share, etc.
- Determine project mix type
  - Choose appropriate operational measures
  - Choose practices to achieve measures for project mix
  - Establish measures and feedback channels for closed loop control

	Variance Examples		
	Low	***Medium***	***High***
Value (Business Measures)	Cost of operations	Market share growth Time to market for new features	Profitability of one-of-a-kind system
Efficiency (Operational Measure)	Cost per change request Individual productivity	Cost per change request Team Productivity	Architectural stability Organizational productivity
Controls	Self check for practices	Beta releases Defect densities, removal rates	Stakeholder demonstrations
Practices	Requirements management Change management Iterative development	Agile planning Test driven development	Shared vision Risk based lifecycle Evolutionary Architecture

Select practices and measures based on business and operational objectives



CEO — Business Value

CIO — IT Quality Goals

Dev. Mgr. — Development Quality Goals

Operational Quality Goals

- High # of defects (pre/post-ship)
- High maintenance costs (devt) of fixing defects
- Customer downtime

- High support or maintenance co high defect #s
- Low pipeline conversion
- Low customer satisfaction

?

Reduce Post-Delivery Defects — H L

- High defects both pre- and post-ship
- Growing defect backlog

?

Deliver What Stakeholder Needs — M L

- Non-functional Req. Issues
- High # of Help Desk Calls
- High Incidence Resolution Times

?

Increase Defect Prevention — M M

- High post-ship or customer-reported defects
- High error rates in fixes

Measures:

- Defect density
- Defect arrival/closure rates
- Defect backlog
- Fixes failing verification
- Rework effort

Practices:

- Test-driven Dev.
- Design-driven Implem.
- C&C management
- System Component Arch.
- Whole team
- Pair Programming
- Review/Inspection

?

Increase Defect Detection — H L

Measures:

- Defect density, distribution
- Defect arrival/closure rates
- Defect removal effectiv.
- Fixes failing verification
- Test coverage
- Test execution status

Practices:

- Test management
- Continuous integration
- Evolutionary Architecture
- Component Architecture
- Test-driven dev.
- Test practices
- Iterative Dev.
- Risk Value Lifecycle
- C&C Management
- Review/Inspection

?

Deliver on Customer Requirements — M L

Measures:

- Post-ship problem reports
- Customer satisfaction
- Pipeline conversion?
- Support / maint. costs
- Requirem. test coverage
- Requirements delivery
- Survey of feature usage

Practices:

- Shared Vision
- Use-case Driven Dev
- Requirements Mgnt.
- Whole Team
- Iterative Dev.
- Functional Testing
- C&C Management
- Review/Inspection

?

Improve Non-Functional Quality Attributes — M L

Measures:

- Post-ship problem reports
- Customer satisfaction
- Support / maint. costs
- Requirement test coverage
- Test execution results

Practices:

- Performance Testing
- Requirements Mgnt.
- Shared Vision
- Risk-Value Lifecycle
- Evolutionary Architecture
- Test-Driven Development
- Iterative Development
- Evolutionary Design
- Component Architecture
- Continuous Integration
- Concurrent Testing
- Whole Team
- Review/Inspection

- High Requirements Churn
- High Post Delivery Support
- Aging ER Backlog

Revenue / Cost

🟩 Value (H,M or L)
🟨 Cost (H, M or L)
● Hot Component

Agenda

- Transitioning to Agile Software Delivery
  - Economic governance
  - Steering and managing uncertainty

- Metrics and Measures
  - Instrumentation for gaining control, improving efficiency and optimizing value

- **Improving Software Economics**
  - **A framework for improvement priorities**

Four patterns of success in achieving Agility at Scale

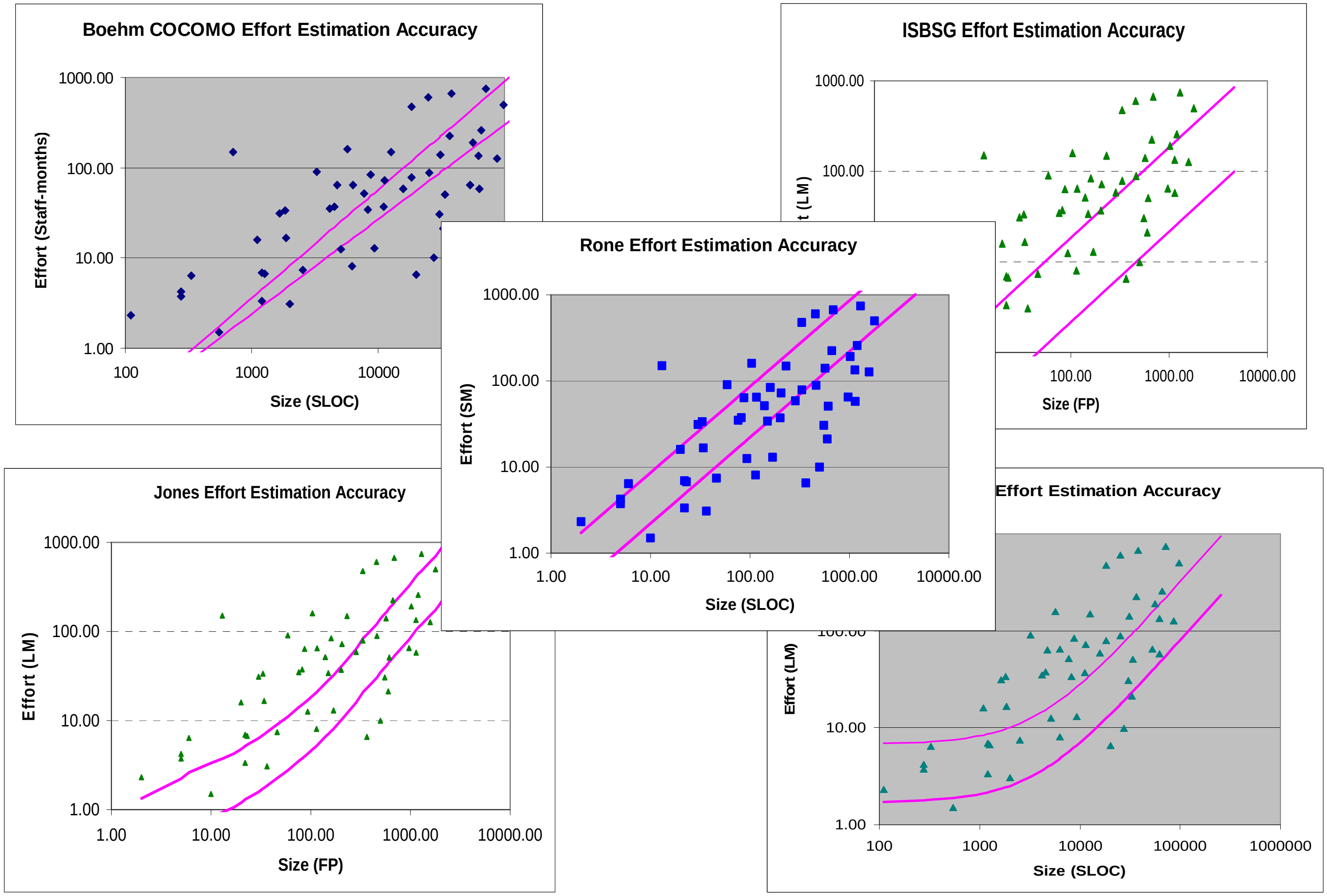
1. **Scope management ➔ *Asset based development***
   Solutions evolve from requirements AND requirements evolve from available assets
   As opposed to getting all the requirements right up front

2. **Process management ➔ *Rightsize the process***
   Process and instrumentation rigor evolves from light to heavy
   As opposed to the entire project's lifecycle process should be light or heavy depending on the character of the project

3. **Progress management ➔ *Honest assessments***
   Healthy projects display a sequence of progressions and digressions
   As opposed to progressing to 100% earned value with monotonically increasing progress against a static plan

4. **Quality management ➔ *Incremental demonstrable results***
   Testing needs to be a 1st class, full lifecycle activity
   As opposed to a subordinate, later lifecycle activity

Software cost models



From George Stark, Paul Oman, “A comparison of parametric Software Estimation Models using real project data”, in press

Improving software economics

- Empirical software cost estimation models for:
  - Enterprise modernization, software maintenance
  - New developments, new releases, early prototypes
  - Packaged applications, systems engineering

$$\text{Time or Cost To Build} = (\text{Complexity})^{(\text{Process})} * (\text{Team}) * (\text{Tools})$$

Complexity

- Volume of human generated stuff
  - KSLOC, FPs, UCs
- Quality/performance
- Scope

Process

- Methods
- Maturity
- Agility
- Precedence

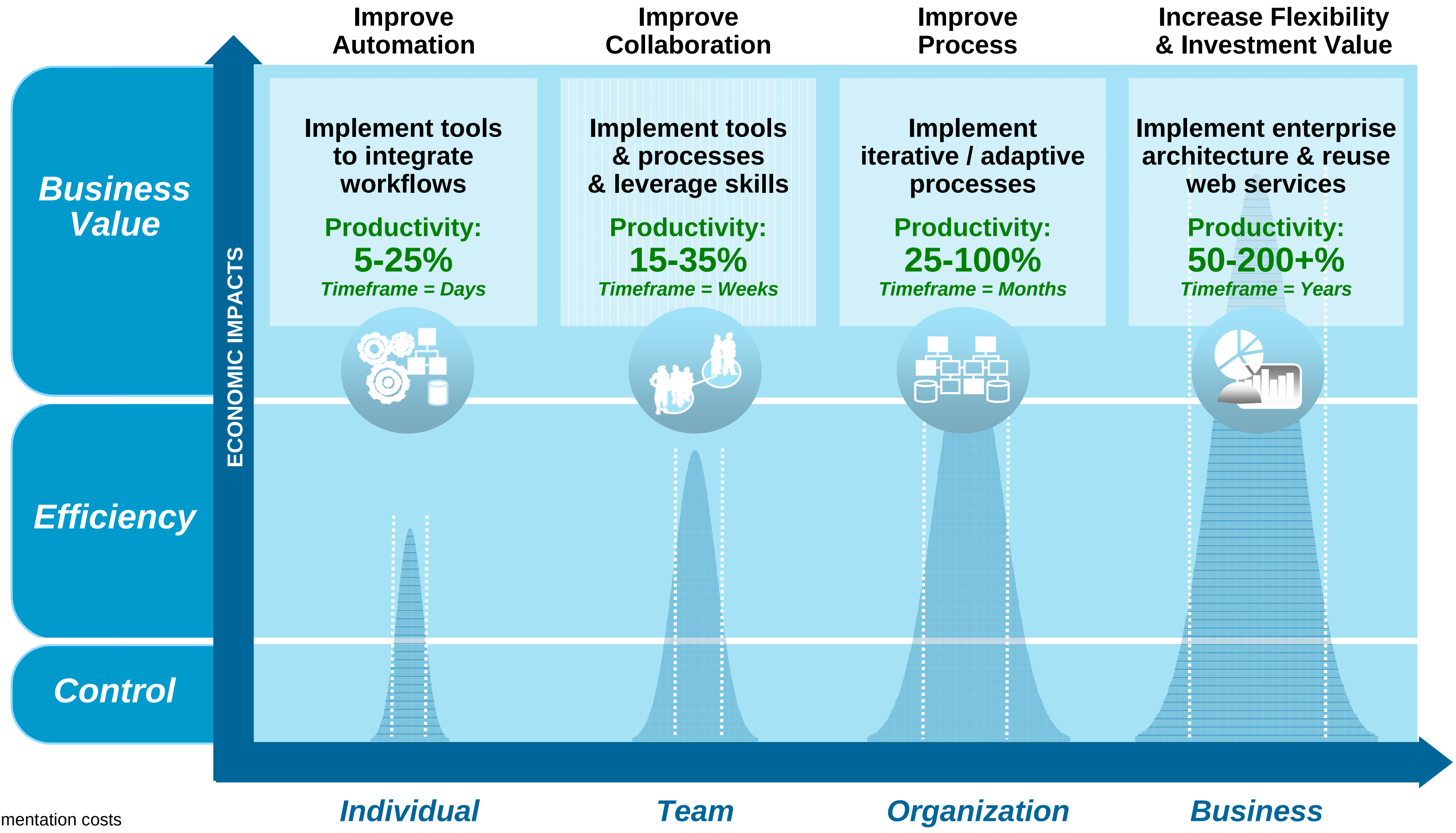
Team

- Skills/Experience
- Collaboration
- Motivation

Tools

- Automation
- Process enactment

Achieve continuous improvement by measuring cost against business outcomes



Improve Automation

Implement tools to integrate workflows

Productivity: 5-25%

Timeframe = Days

Improve Collaboration

Implement tools & processes & leverage skills

Productivity: 15-35%

Timeframe = Weeks

Improve Process

Implement iterative / adaptive processes

Productivity: 25-100%

Timeframe = Months

Increase Flexibility & Investment Value

Implement enterprise architecture & reuse web services

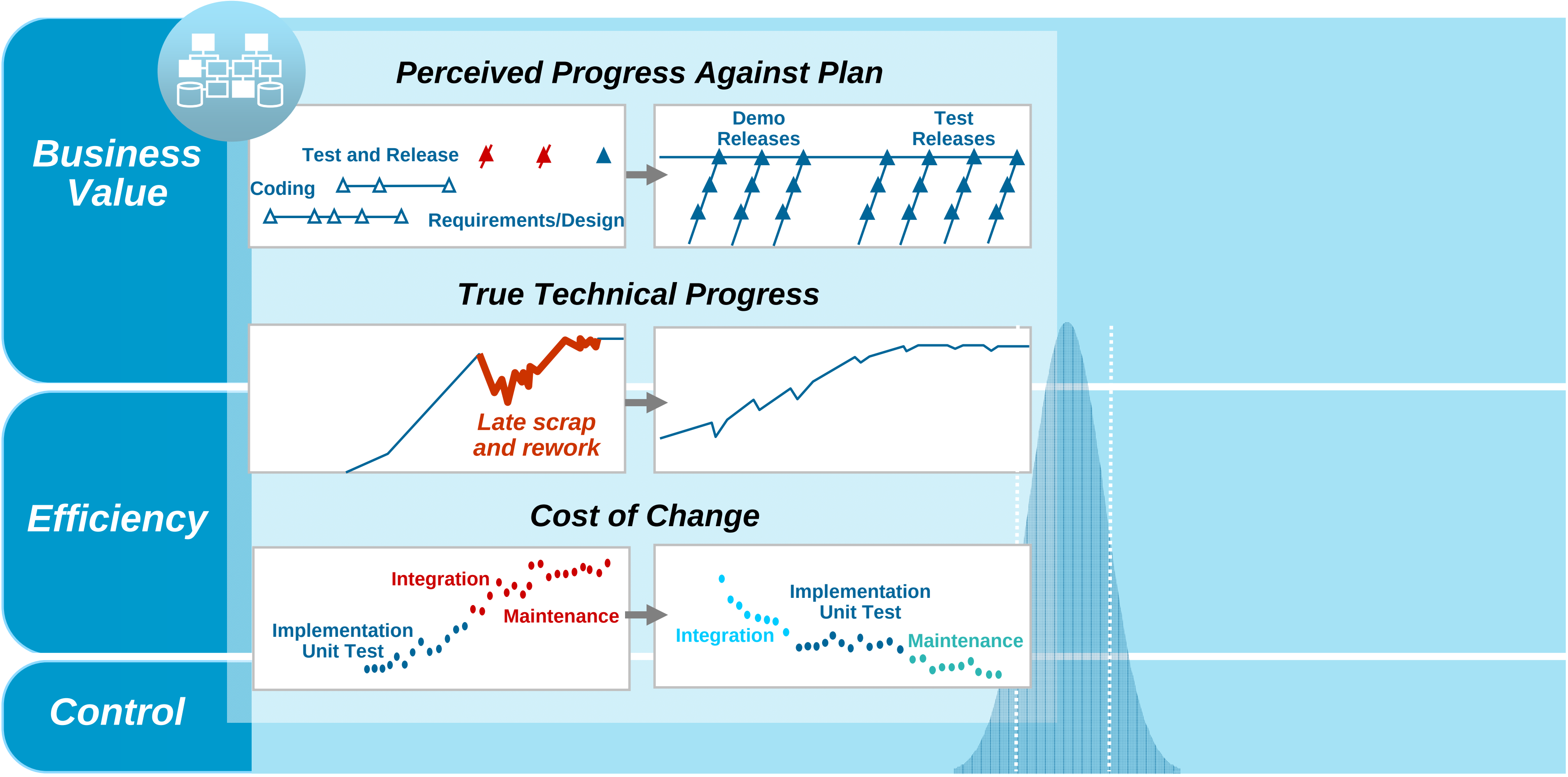
Productivity: 50-200+%

Timeframe = Years

Implementation costs are per person per year

Improve process to increase productivity by 25%-100%

Implement iterative / adaptive processes



Business Value

Efficiency

Control

Perceived Progress Against Plan

True Technical Progress

Cost of Change

Improving process and increasing flexibility

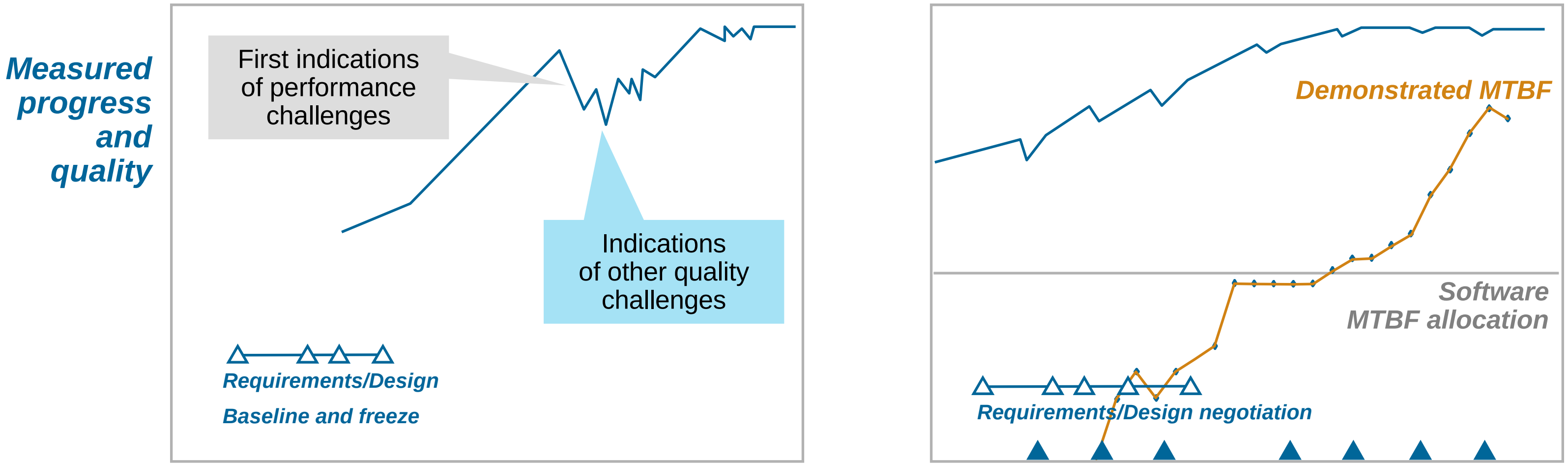
Reducing the significant uncertainties in quality and performance

Improve Process and Increase Flexibility

WATERFALL DEVELOPMENT → **ITERATIVE DEVELOPMENT AND AGILE DELIVERY**

Late quality and performance insight constrains flexibility to make tradeoffs

Continuous quality and performance insight allows flexibility in trading off cost, quality, and features



- Speculative quality requirements
- Unpredictable cost/schedule performance
- Late shoehorning of suboptimal changes that impact quality

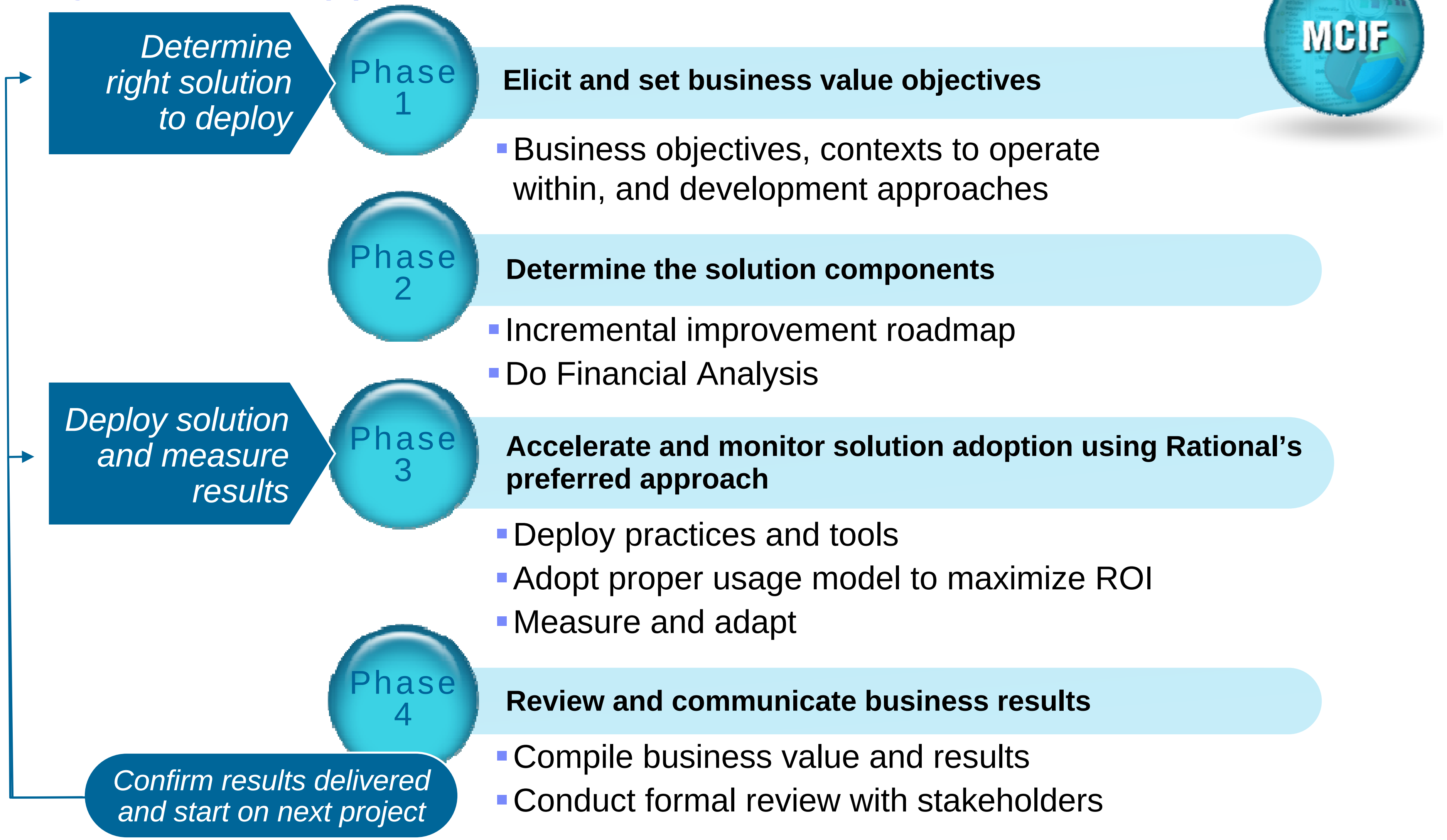
- Release qualities that matter
- Quality progressions/digressions
- Early requirement verification and/or negotiation

Slide 29

A1 4) finally, on slide 4, i changed the heading that was iterative development to both iterative development and agile delivery. not sure if that matters but it matches my paper and danny may prefer agile delivery rather than iterative development.

Author, 4/29/2009

Measured Capability Improvement Framework (MCIF): A systematic approach to software excellence



Determine right solution to deploy

Phase 1

Elicit and set business value objectives

- Business objectives, contexts to operate within, and development approaches

Phase 2

Determine the solution components

- Incremental improvement roadmap
- Do Financial Analysis

Deploy solution and measure results

Phase 3

Accelerate and monitor solution adoption using Rational's preferred approach

- Deploy practices and tools
- Adopt proper usage model to maximize ROI
- Measure and adapt

Phase 4

Review and communicate business results

- Compile business value and results
- Conduct formal review with stakeholders

Confirm results delivered and start on next project

Some final thoughts

Software delivery is a discipline of software economics balancing risks and opportunities

Process enactment and measurement are imperatives to achieving agility at scale

Software delivery requires a platform that is architected for automation, collaboration and reporting

Thank
You