STERLING BARTON

Managing Software Debt

Continued Delivery of High Value as Systems Age

Chris Sterling

Technology Consultant / Agile Coach /
Certified Scrum Trainer

Sterling Barton, LLC

Web: www.SterlingBarton.com

Email: chris@sterlingbarton.com

Blog: www.GettingAgile.com

Follow Me on Twitter: @csterwa

Hash Tag for Presentation: #swdebt

Topics Being Covered

- Problems Found with Aging Software
- Software Debt Explained
  - Technical Debt
  - Quality Debt
  - Configuration Management Debt
  - Design Debt
  - Platform Experience Debt
- The Wrap Up
  - A Story of What is Possible

Problems Found with Aging Software

- Software gets difficult to add features to as it ages
- Business expectations do not lessen as software ages
- Software must remain maintainable and changeable to meet needs of business over time

Lack of emphasis on software quality attributes contributes to **decay**

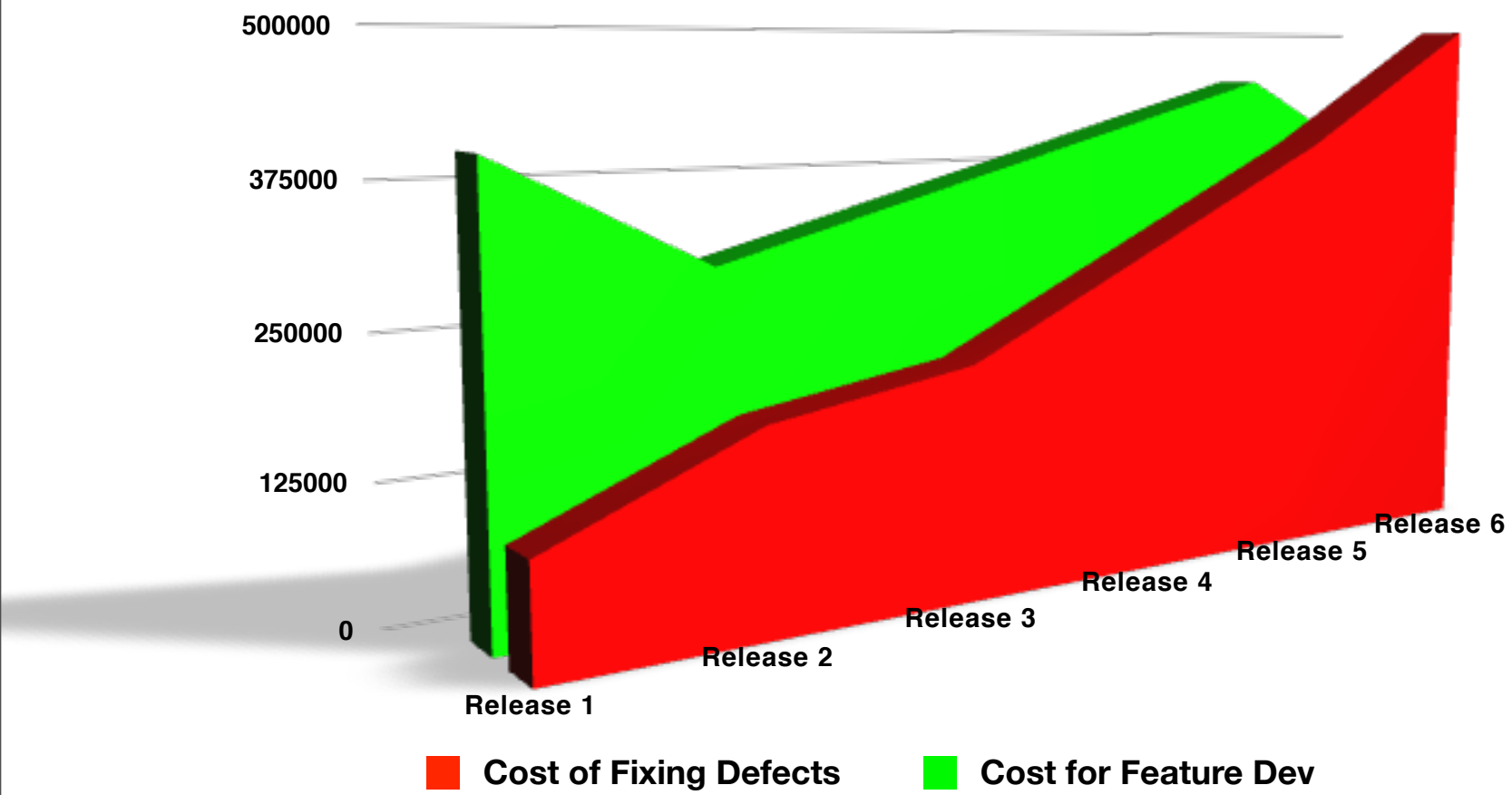
The “Rewrite”, “NextGen” or “Like-to-like Migration”

- “It will be easy since we worked on the original version” - although we understand the domain we will be fighting with new features, technology, tools, and processes
- “We don’t have any other options” - Refactoring and test automation are potential alternatives to like-to-like migrations.

Limited Platform Expertise

Risk and costs increase as expertise becomes more limited for aging software platforms.

Costs for Release Stabilization Increase Over Time



Extreme Specialization

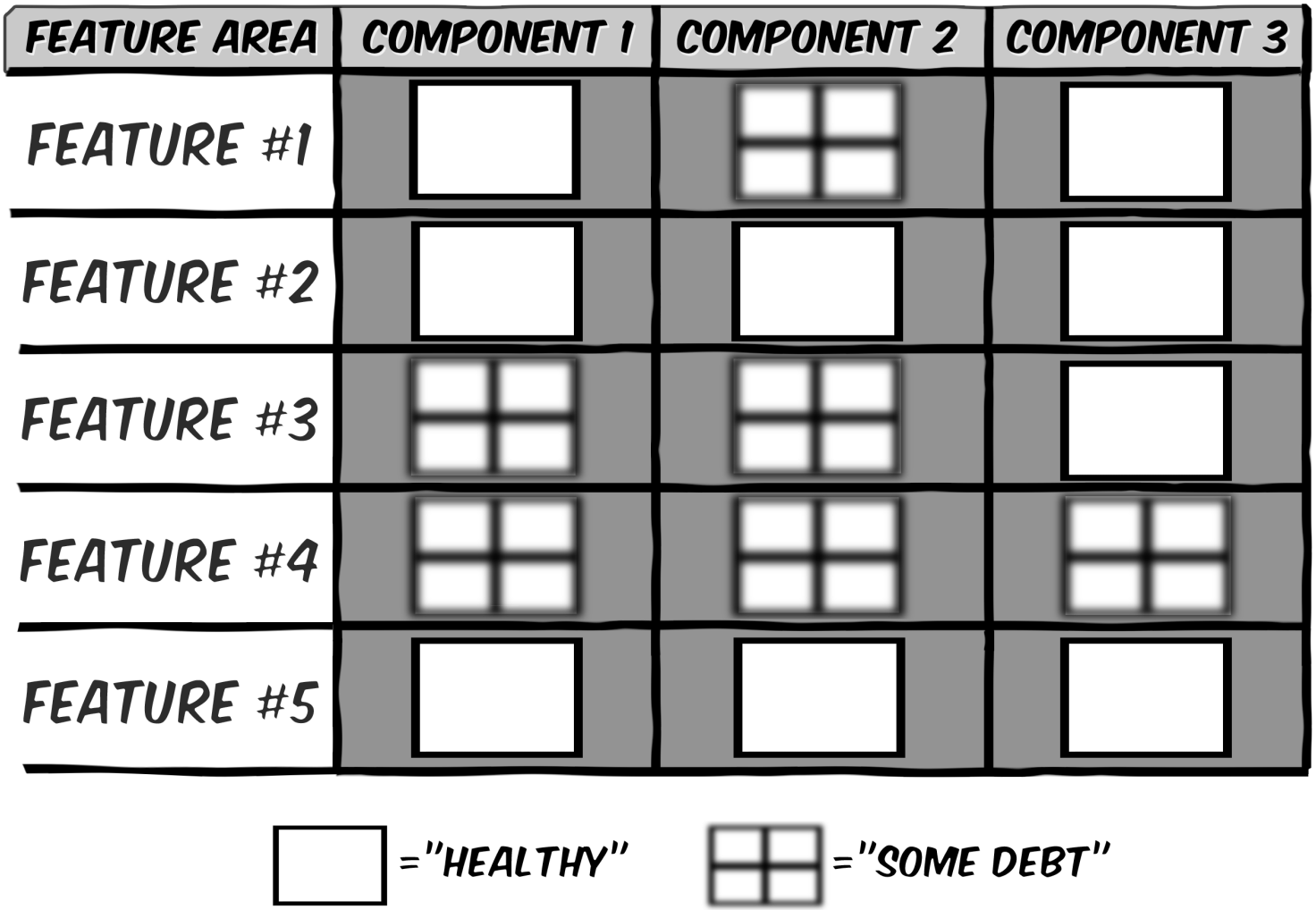
- Knowledge and capability to maintain legacy software decays with time
- Costs to maintain rarely used software platforms are higher
- Leads to waiting for people in specialized roles to finish their tasks in support of development effort

STERLING BARTON

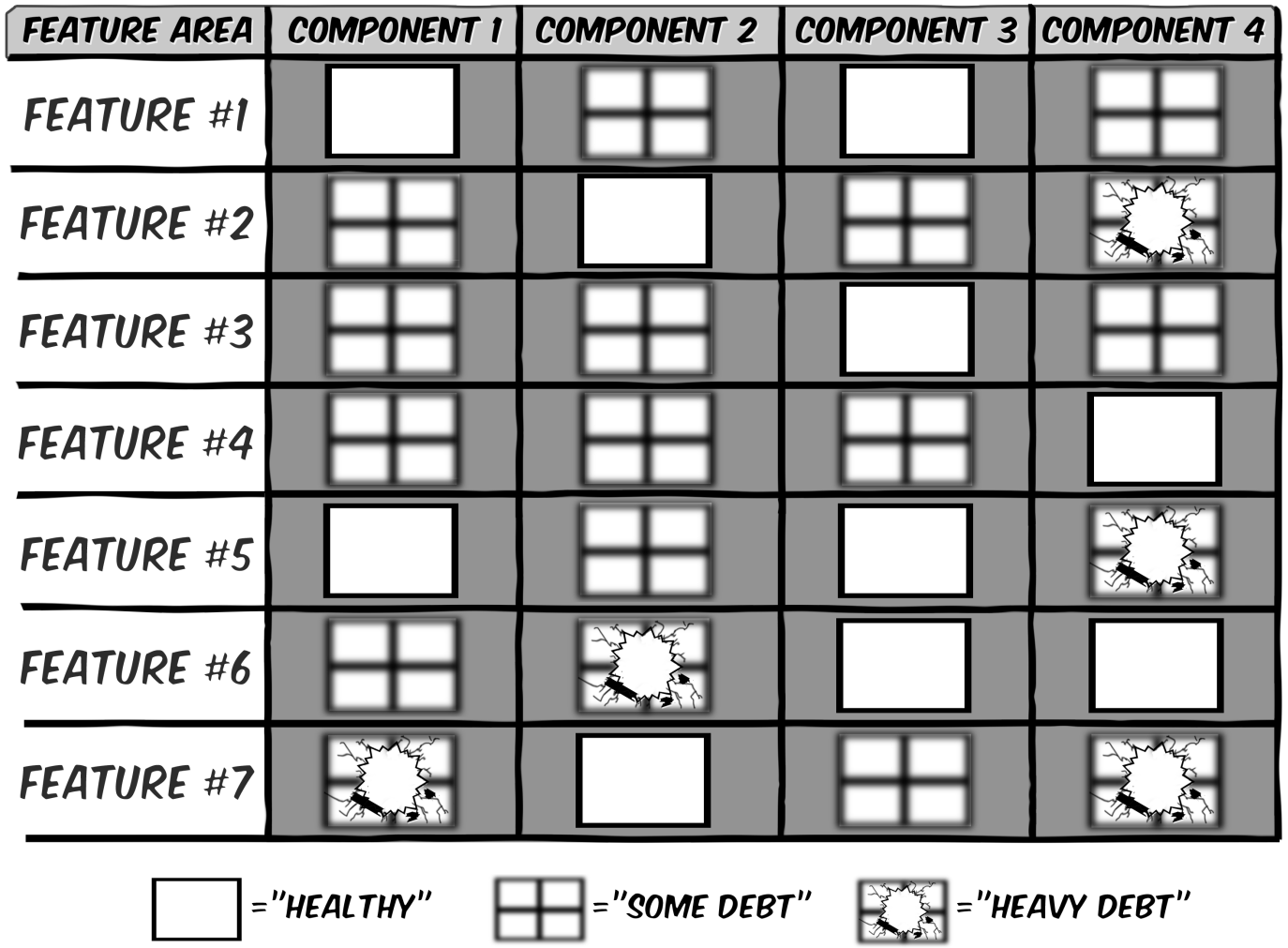
Software Debt

Creeps into software slowly and leaves organizations with liabilities

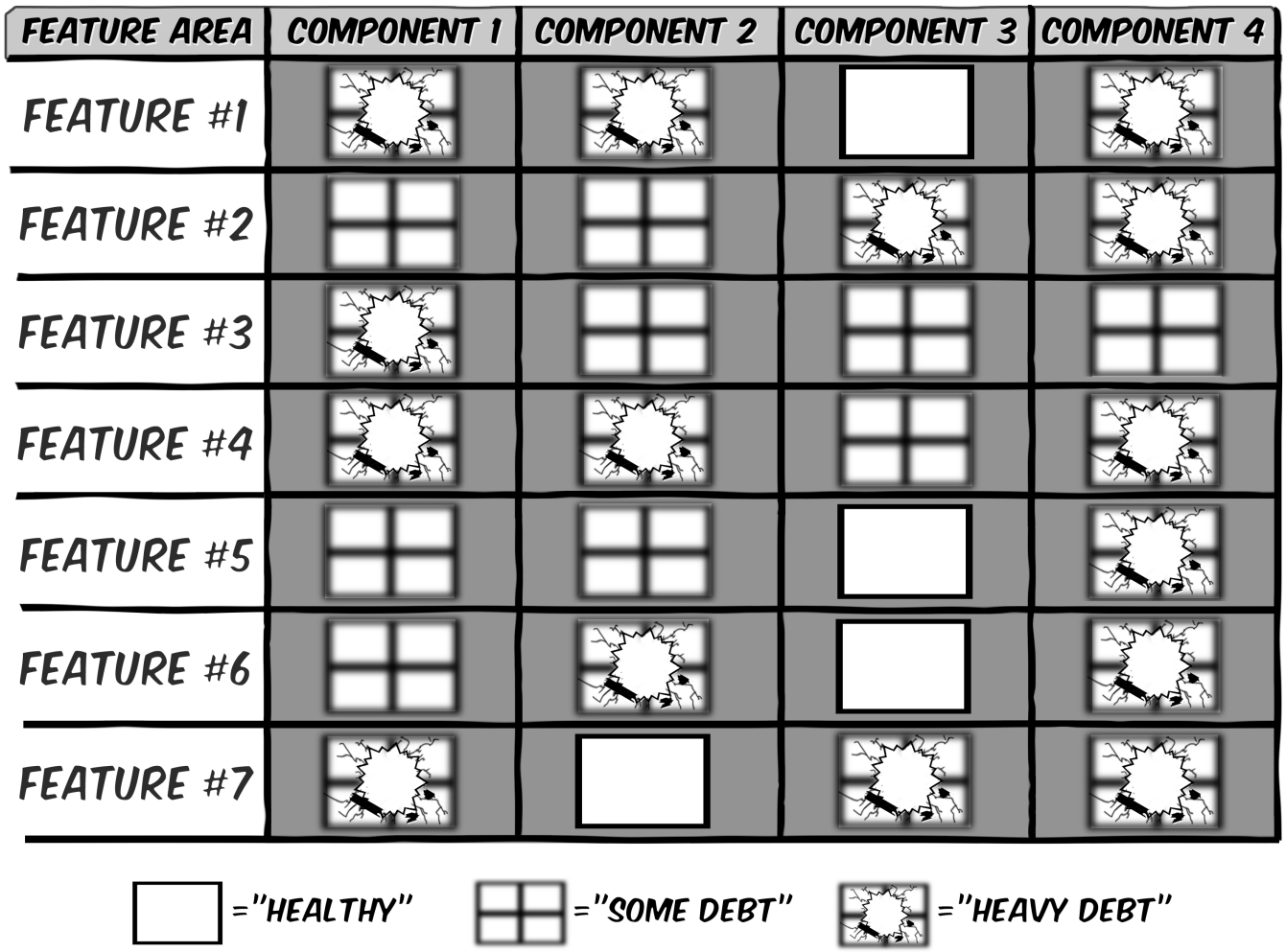
Software Debt Creeps In



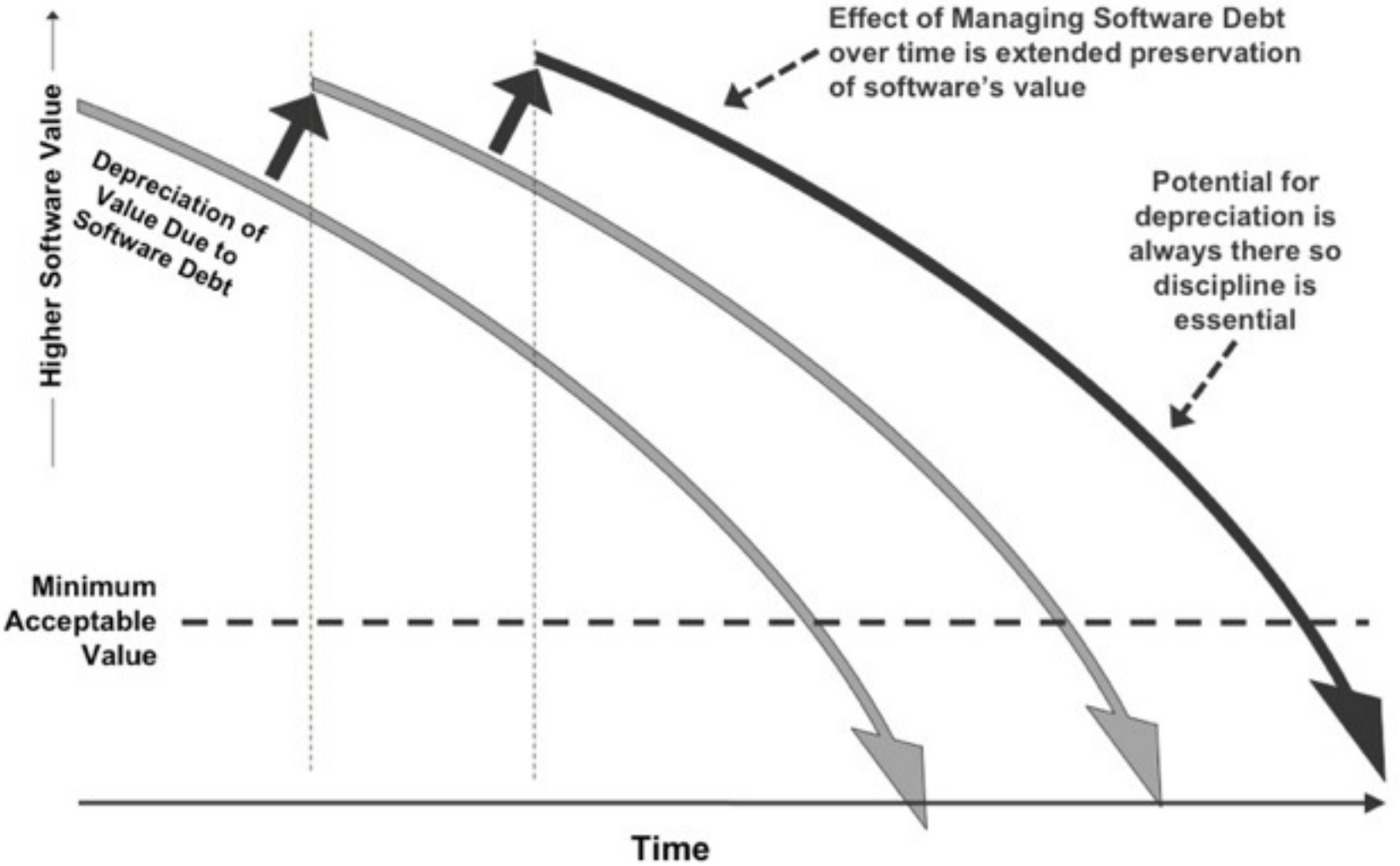
Software Debt Creeps In



Software Debt Creeps In



Managing Software Debt – an Overview



Managing Software Debt

- It is impossible to stop software debt from creeping into our software
- Managing debt in software is based on putting frequent feedback mechanisms in place for code, quality assurance, configuration management, design, and organization of people on project team
- Feedback mechanisms should be frequent, automated, and easy to use to support their continued use or modified to new needs

Types of Software Debt

- Technical Debt
- Quality Debt
- Configuration Management Debt
- Design Debt
- Platform Experience Debt

STERLING BARTON

Technical Debt

Issues in software that will impede future development if left unresolved

* Ward Cunningham on “Technical Debt”

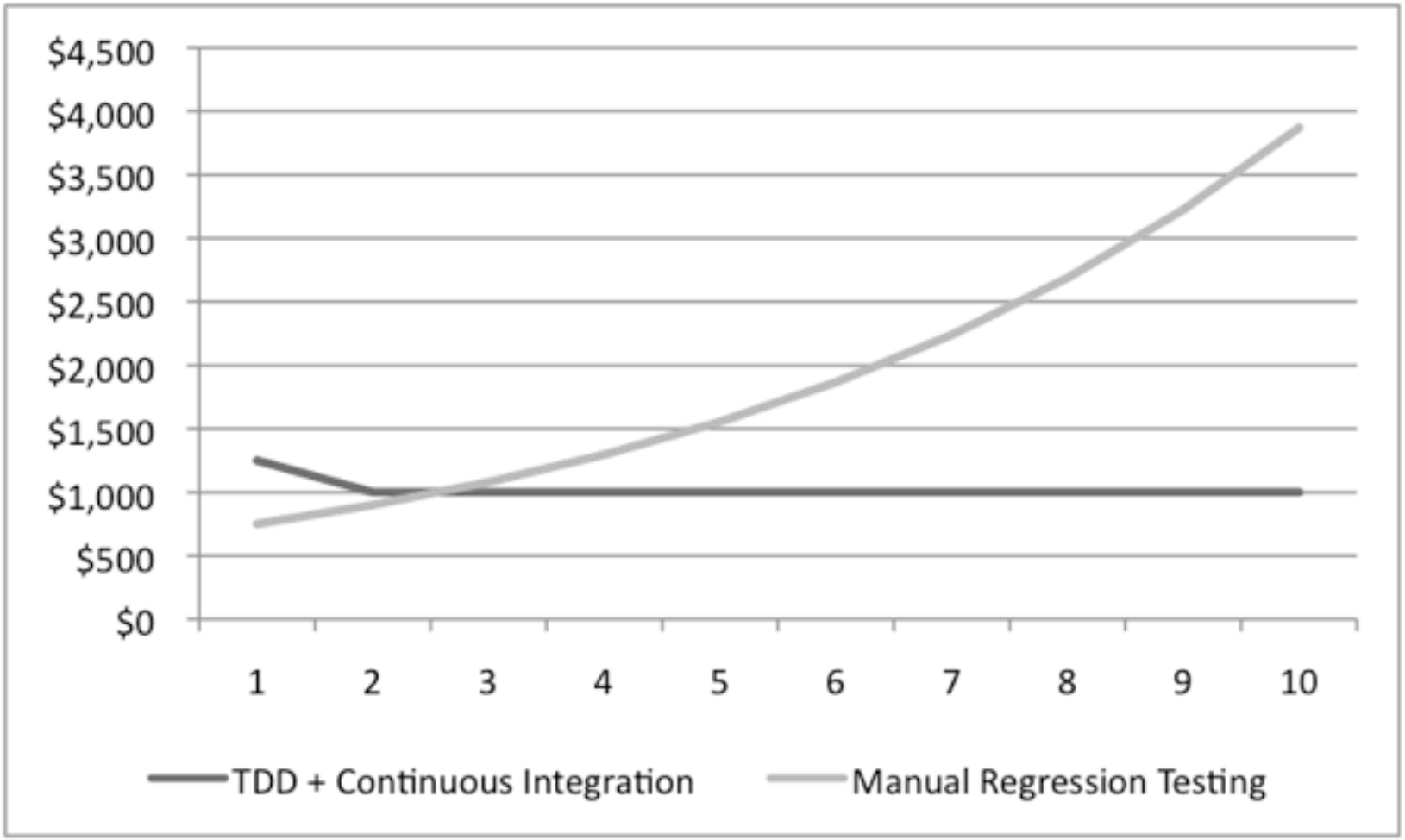
- Technical Debt includes those internal things that you choose not to do now, but which will impede future development if left undone. This includes deferred refactoring.
- Technical Debt doesn’t include deferred functionality, except possibly in edge cases where delivered functionality is “good enough” for the customer, but doesn’t satisfy some standard (e.g., a UI element that isn’t fully compliant with some UI standard).

* Ward Cunningham - “Technical Debt” - http://c2.com/cgi/wiki?TechnicalDebt

My Definition of “Technical Debt”

“Technical debt is the decay of component and inter-component behavior when the application functionality meets a minimum standard of satisfaction for the customer.”

Regression Costs - Manual vs. Automated



Principles of Executable Design

- The way we design can always be improved.
- We'll get it "right" the third time.
- We will not get it "right" the first time.
- Design and construct for change rather than longevity.
- Lower the threshold of pain.

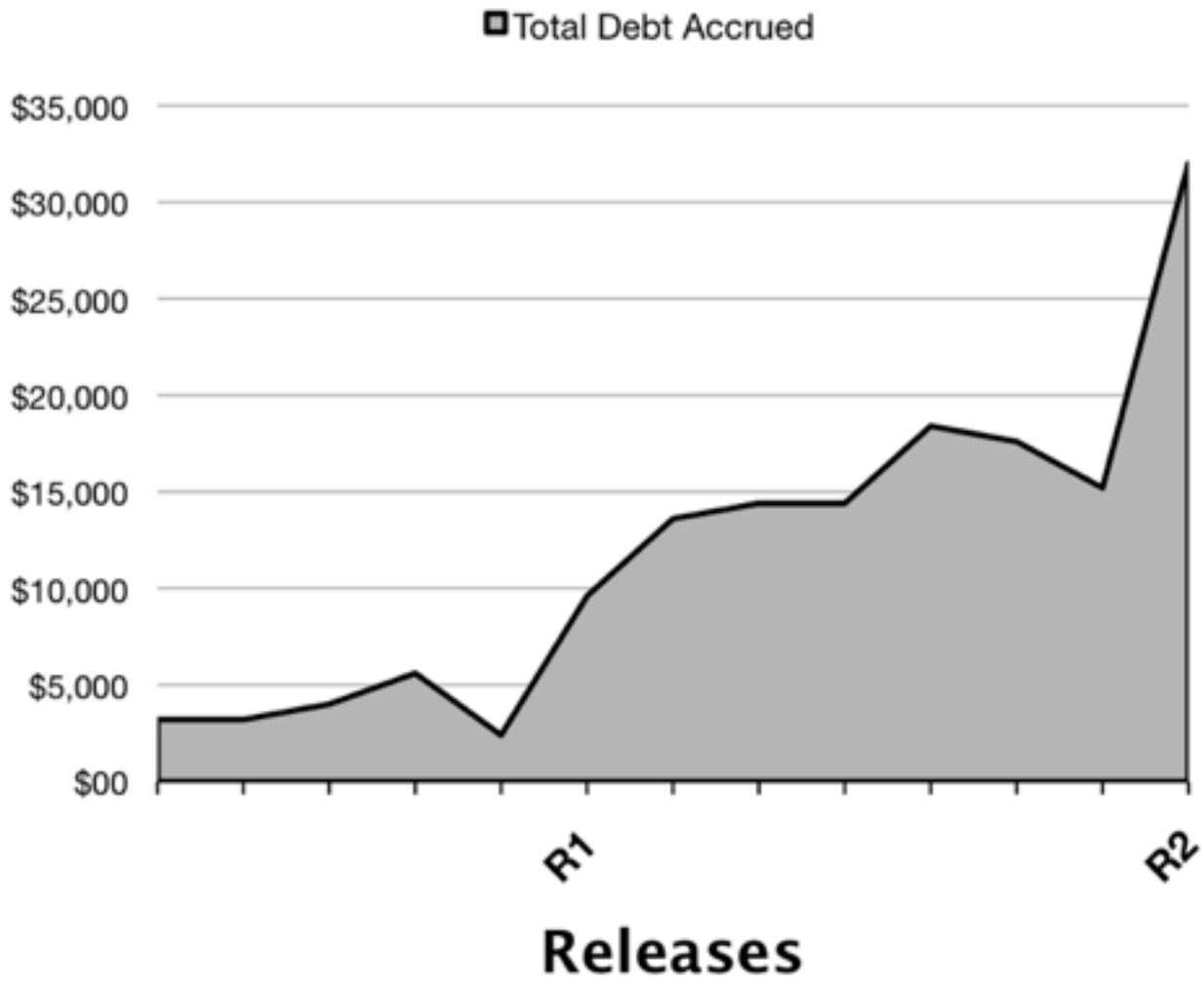
If we are not enhancing the design then
we are just writing a bunch of tests.

STERLING BARTON

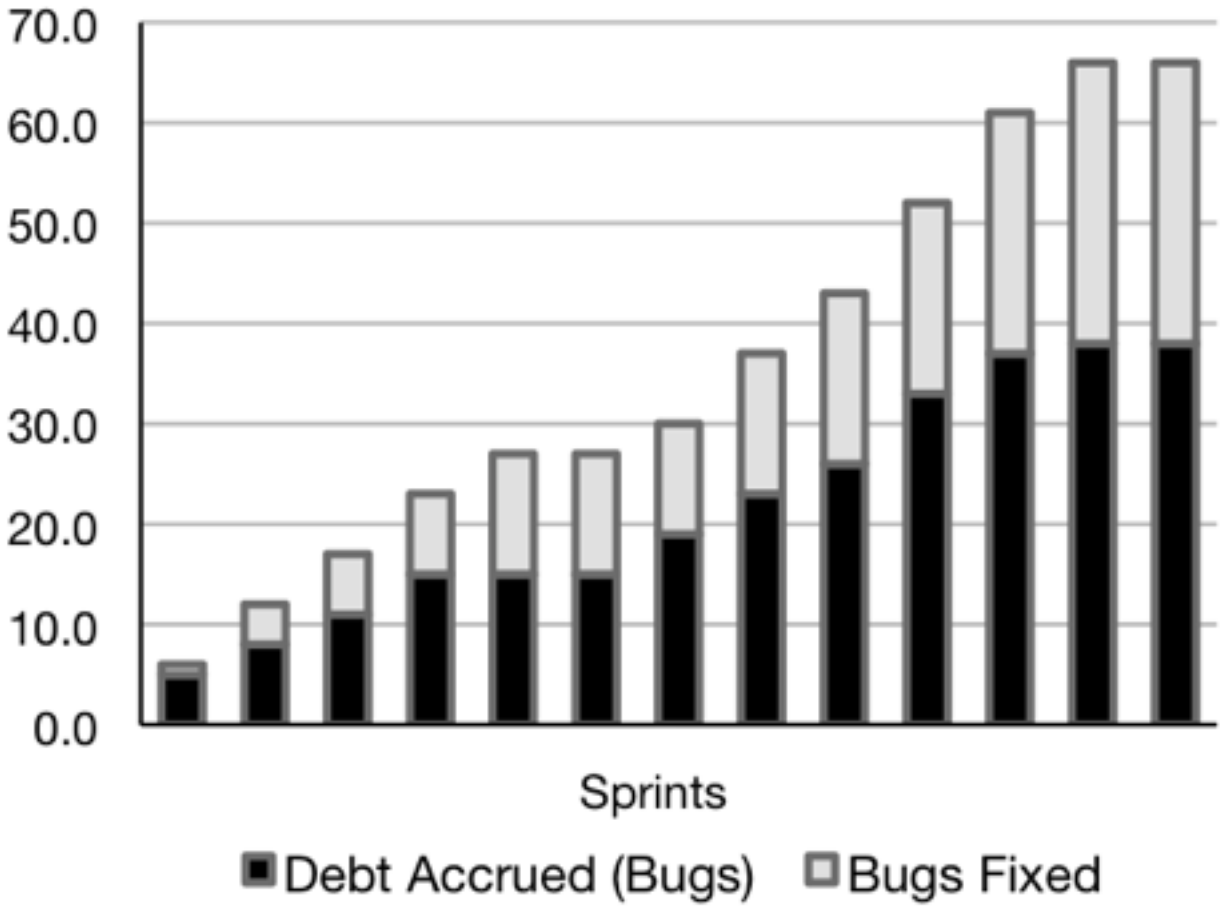
Quality Debt

A lack of quality will lessen the value per feature added over time

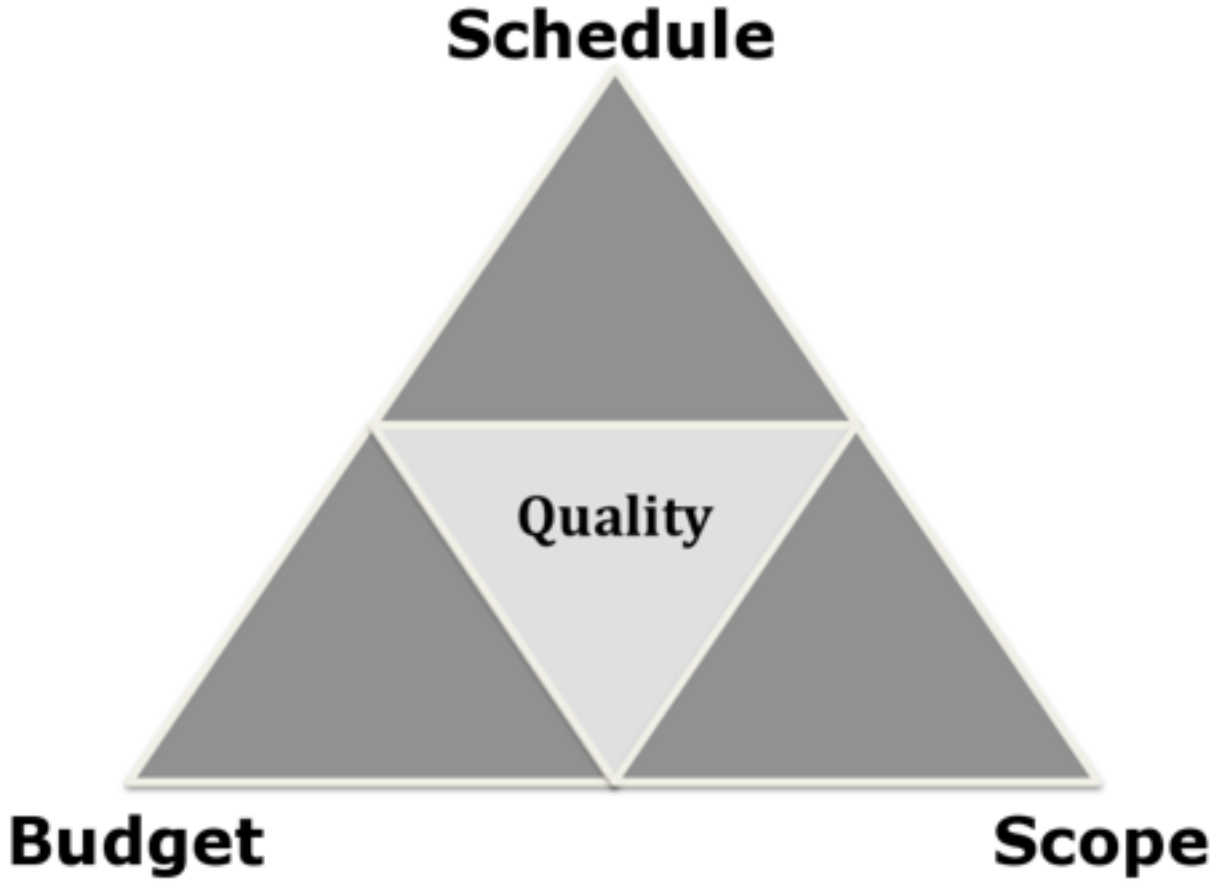
Accrual of Quality Debt with Releases



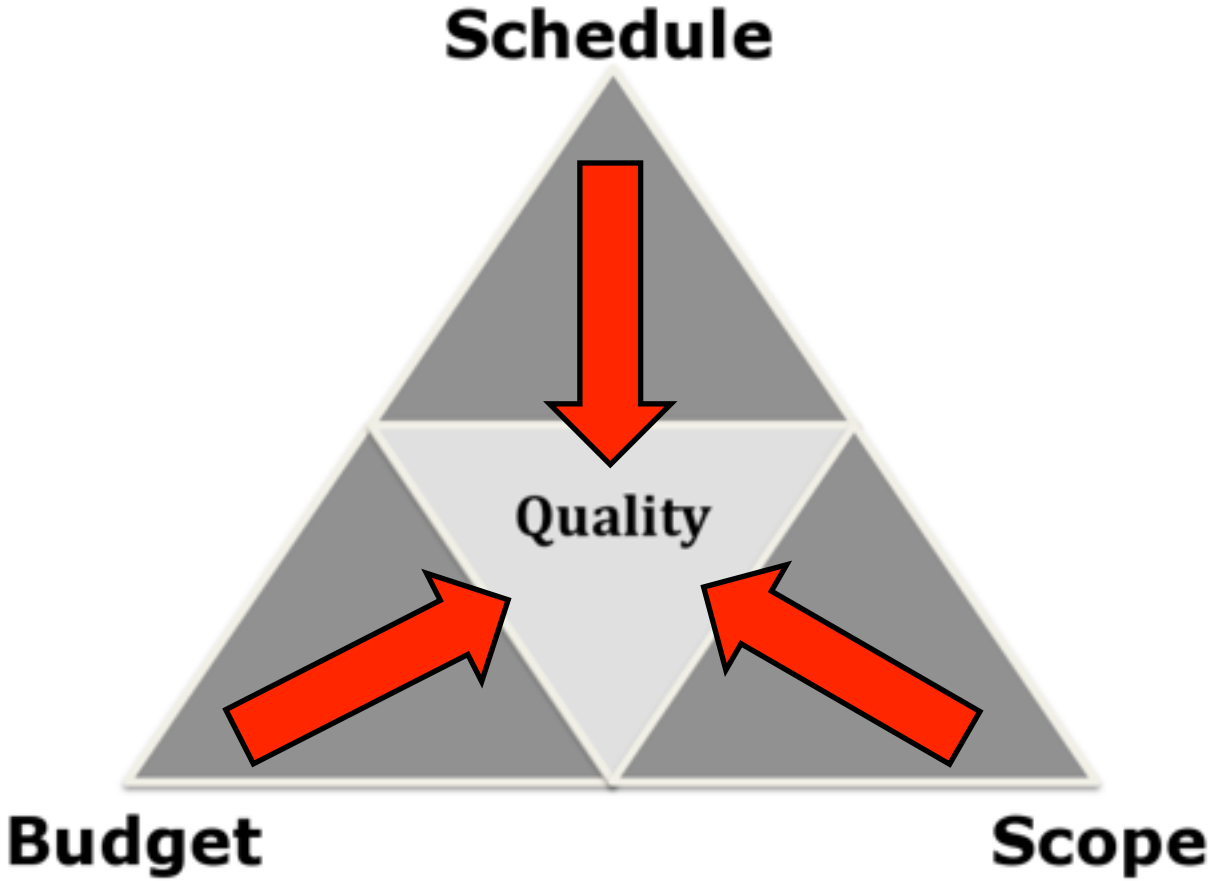
Break/Fix Only Prolongs the Agony



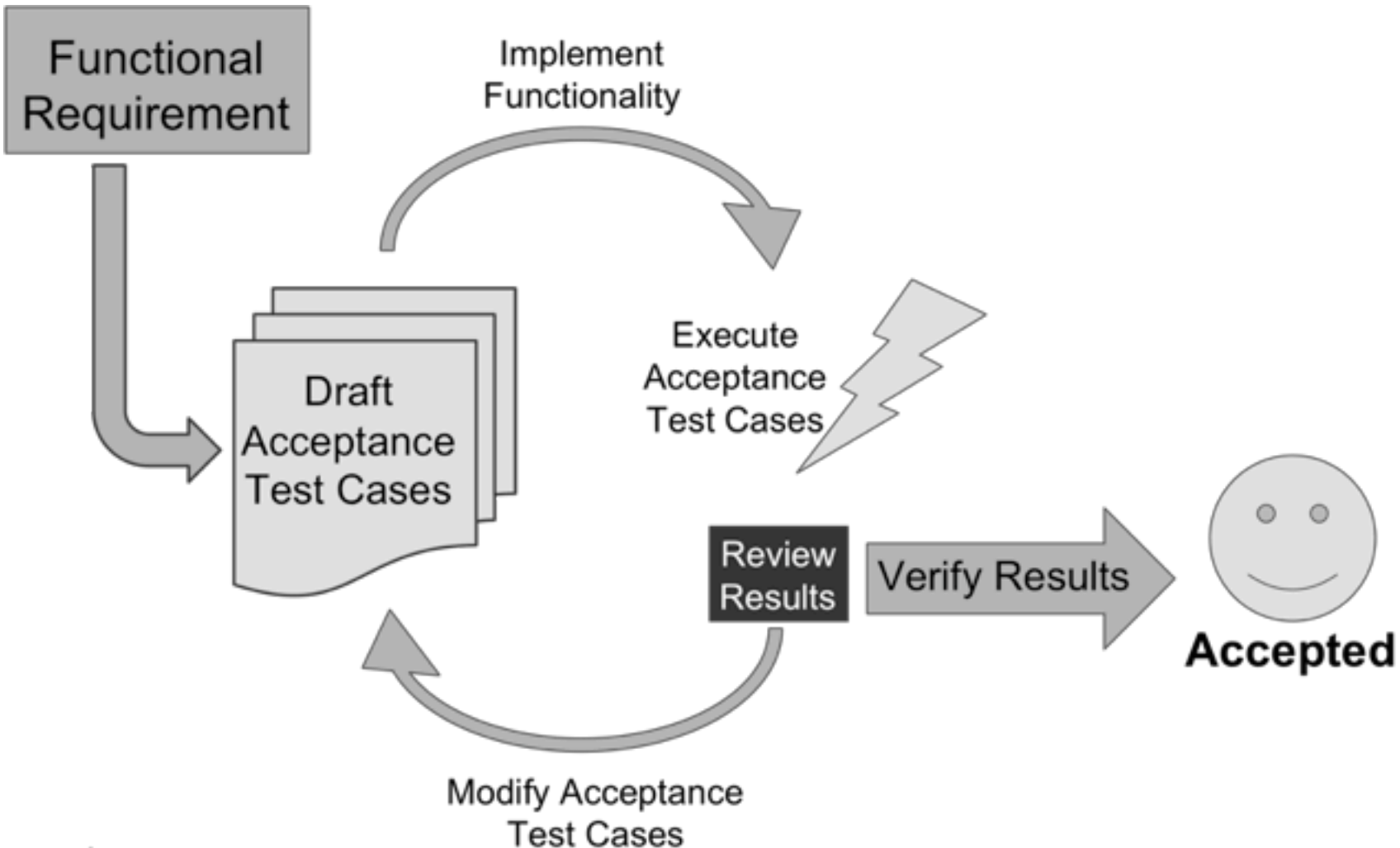
Effect of Project Constraints on Quality



Effect of Project Constraints on Quality



Acceptance Test-Driven Development



STERLING BARTON

A Fit Case Study

Cost reduction using Fit for test automation and data conversion

Manual Regression Testing

- Testing was taking 75 person hours during 2 full test runs consisting of:
  - Comprehensive manual regression testing
  - Data conversion and validation
- Cost for testing was $17,000 each iteration

Introducing Fit into Testing Process

- After 8 iterations team had introduced healthy amount of Fit fixtures and automated tests
- Reduced 70+ hour test runtime down to 6 hours which now included:
  - Fit automated regression testing
  - Data conversion and validation automated with Fit fixtures
- Reduced cost of testing each iteration from $17,000 to $7,000

STERLING BARTON

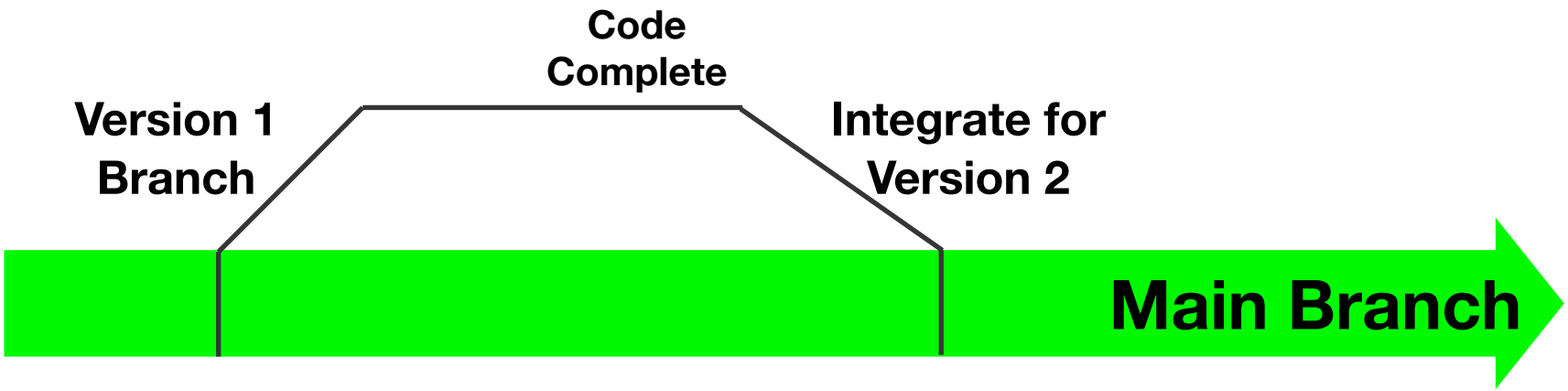
Configuration Management Debt

Unpredictable and error-prone release management

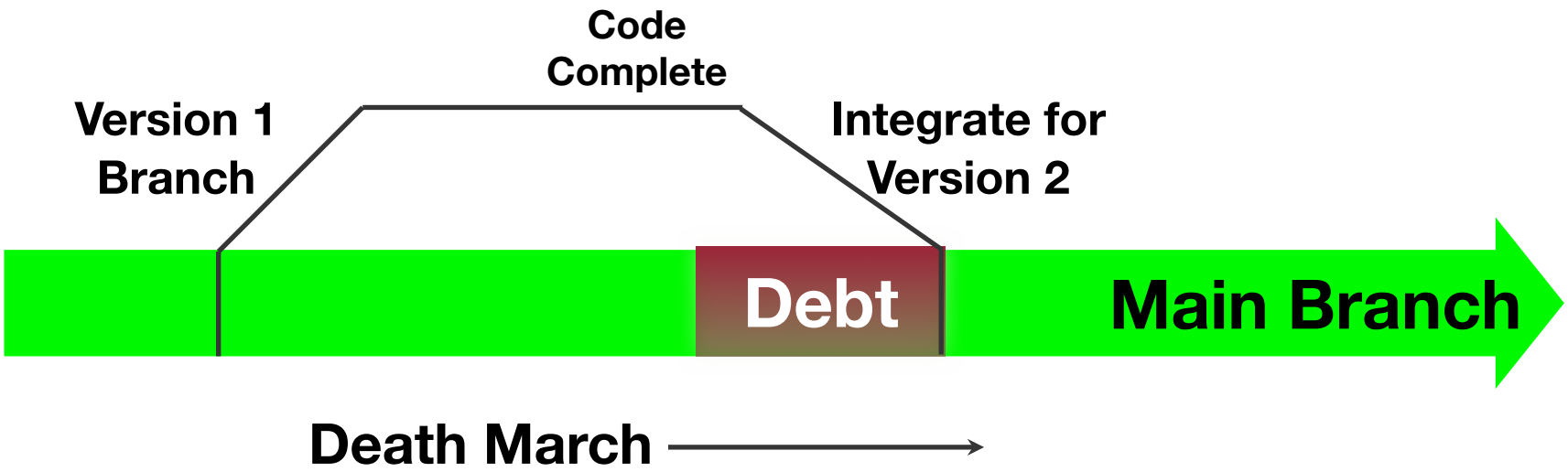
Traditional Source Control Management

Traditional Source Control Management

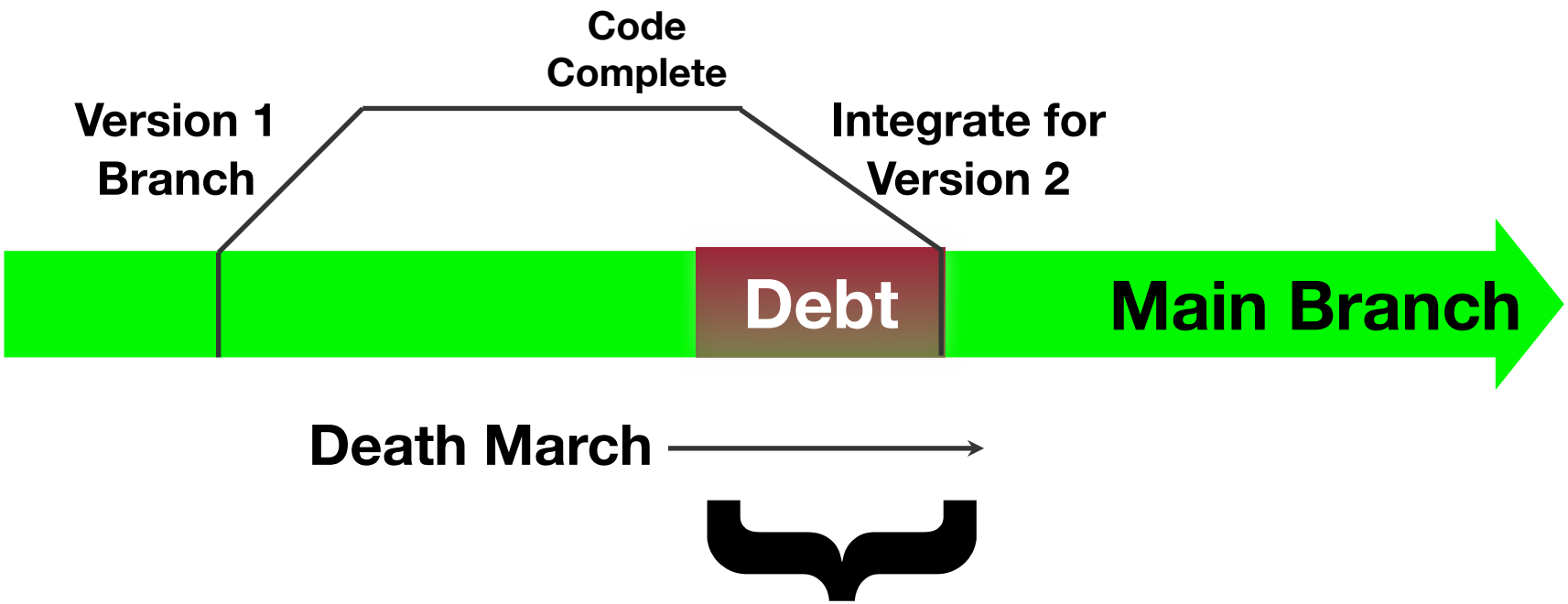
Traditional Source Control Management



Traditional Source Control Management



Traditional Source Control Management



Debt accrues quickly within stabilization periods

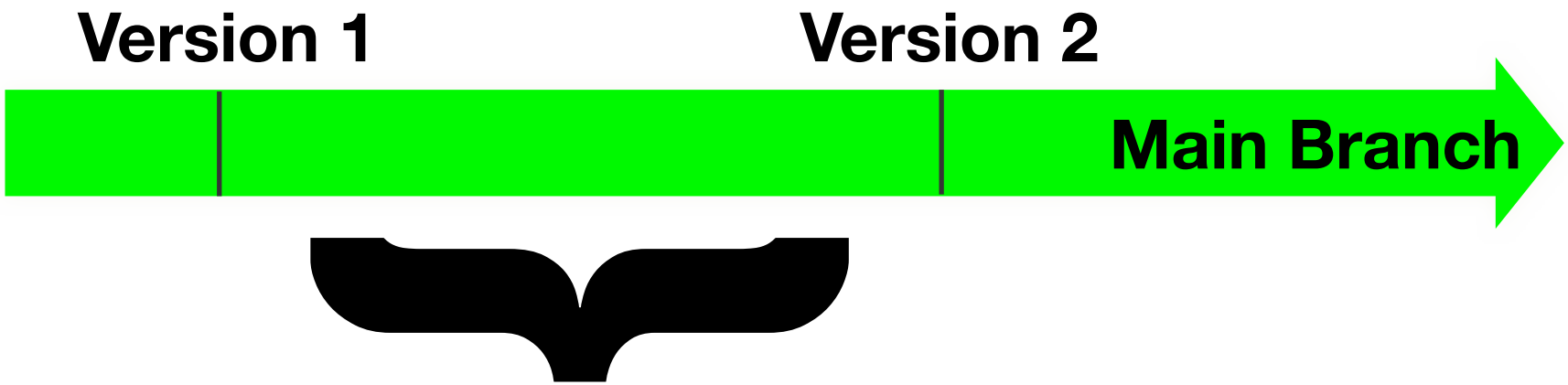
Flexible Source Control Management

Flexible Source Control Management

Flexible Source Control Management

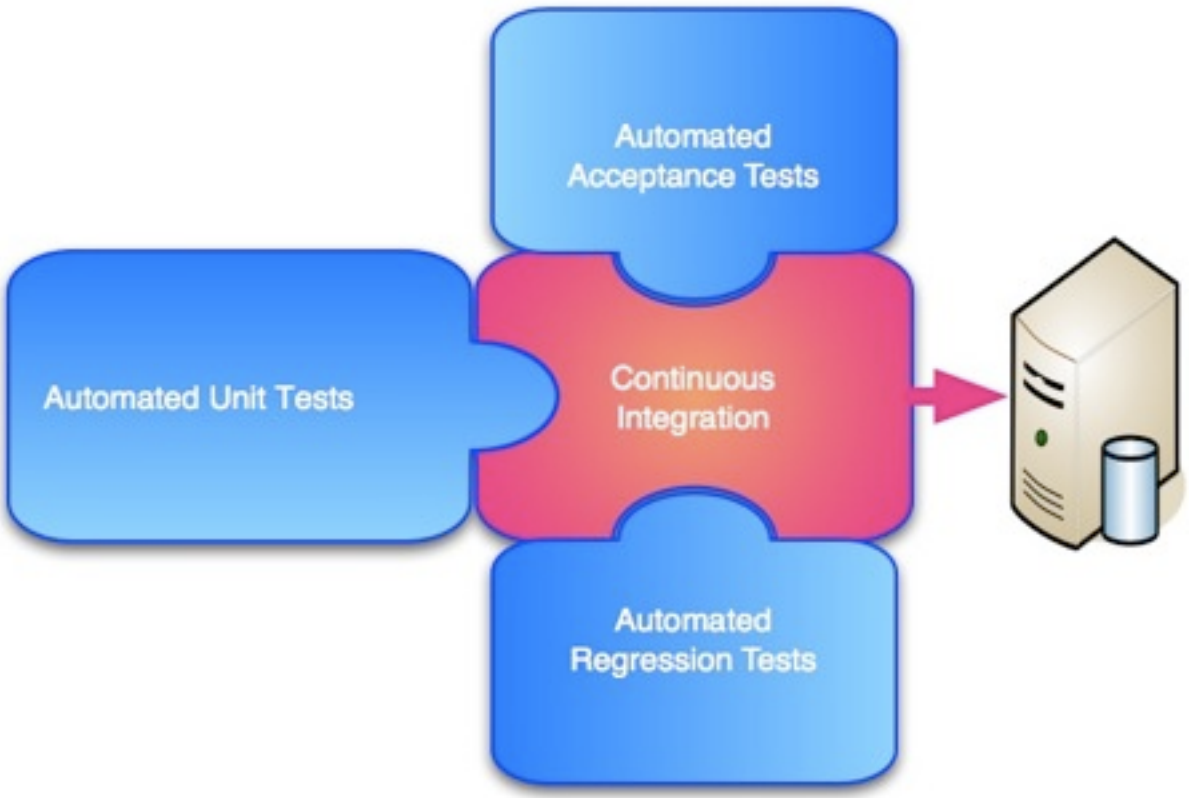
Flexible Source Control Management

Flexible Source Control Management

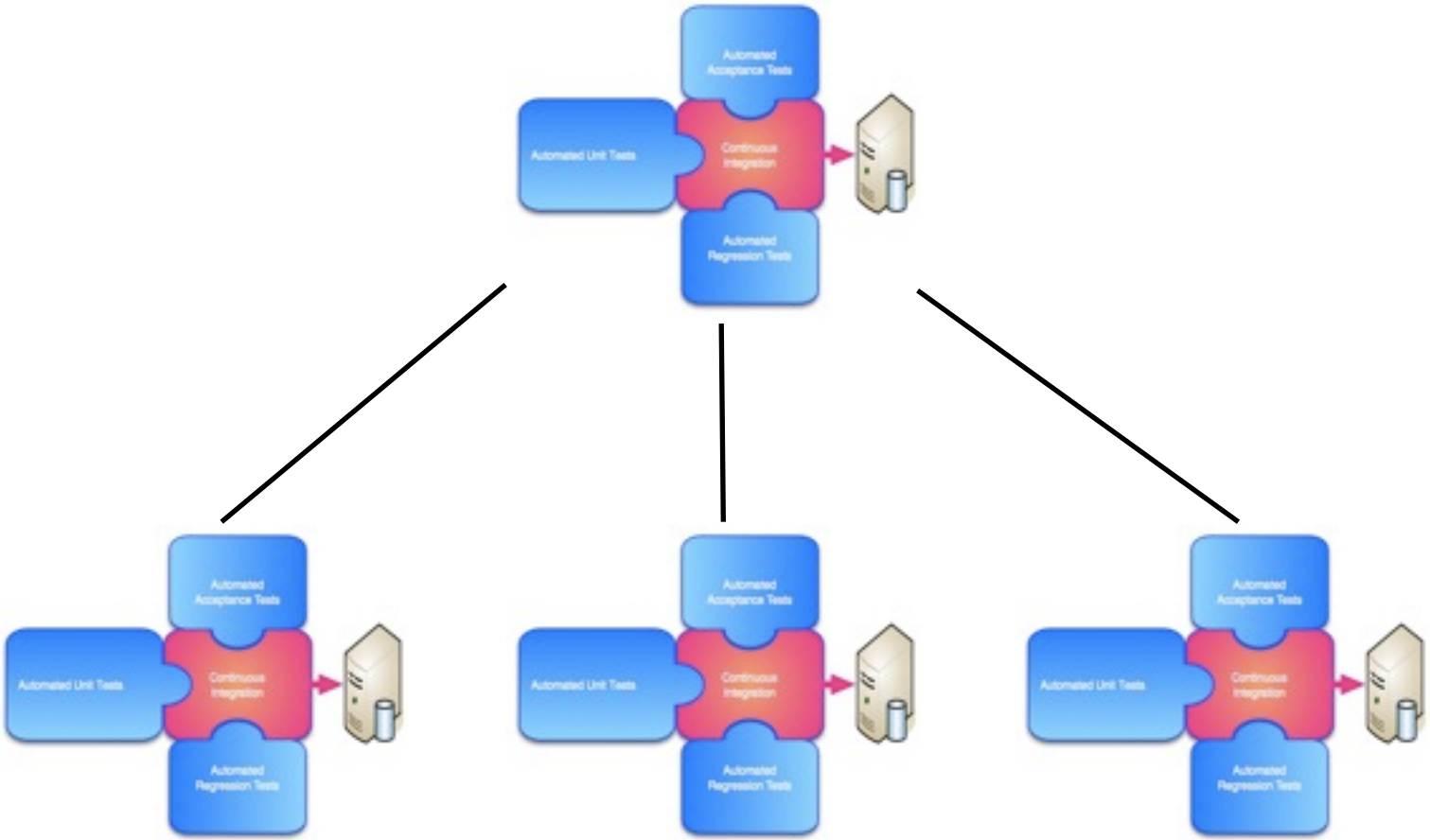


Not Easy! Must have proper infrastructure to do this.

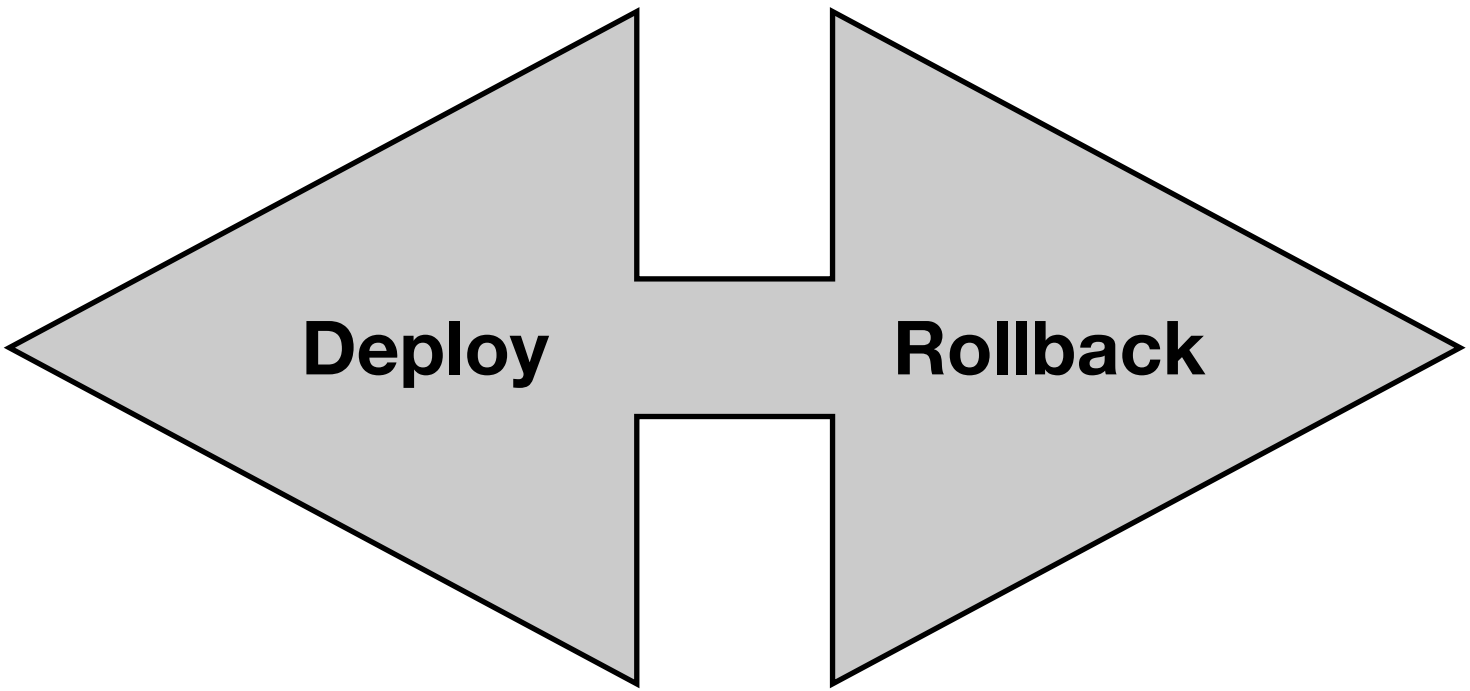
Continuous Integration



Scaling to Multiple Integrations



The Power of 2 Scripts: Deploy and Rollback

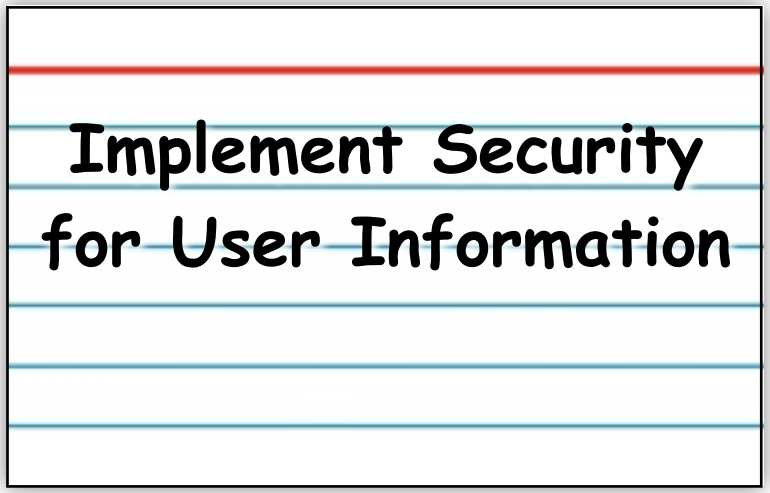


STERLING BARTON

Design Debt

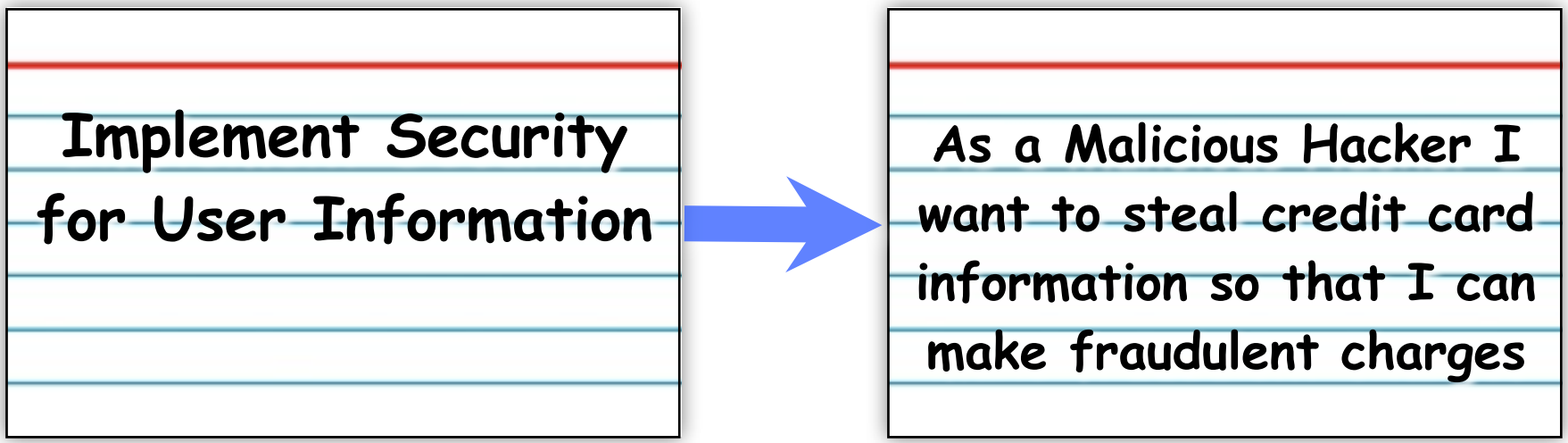
Design decays when not attended to so design software continually

* Abuse User Stories



* From “User Stories Applied” presented by Mike Cohn Agile 2006

* Abuse User Stories



* From “User Stories Applied” presented by Mike Cohn Agile 2006

STERLING BARTON

Platform Experience Debt

Silos of knowledge and increased specialization will increase cost of maintenance over time

How to Combat Platform Experience Debt

- Ignore it (I do not suggest this!)

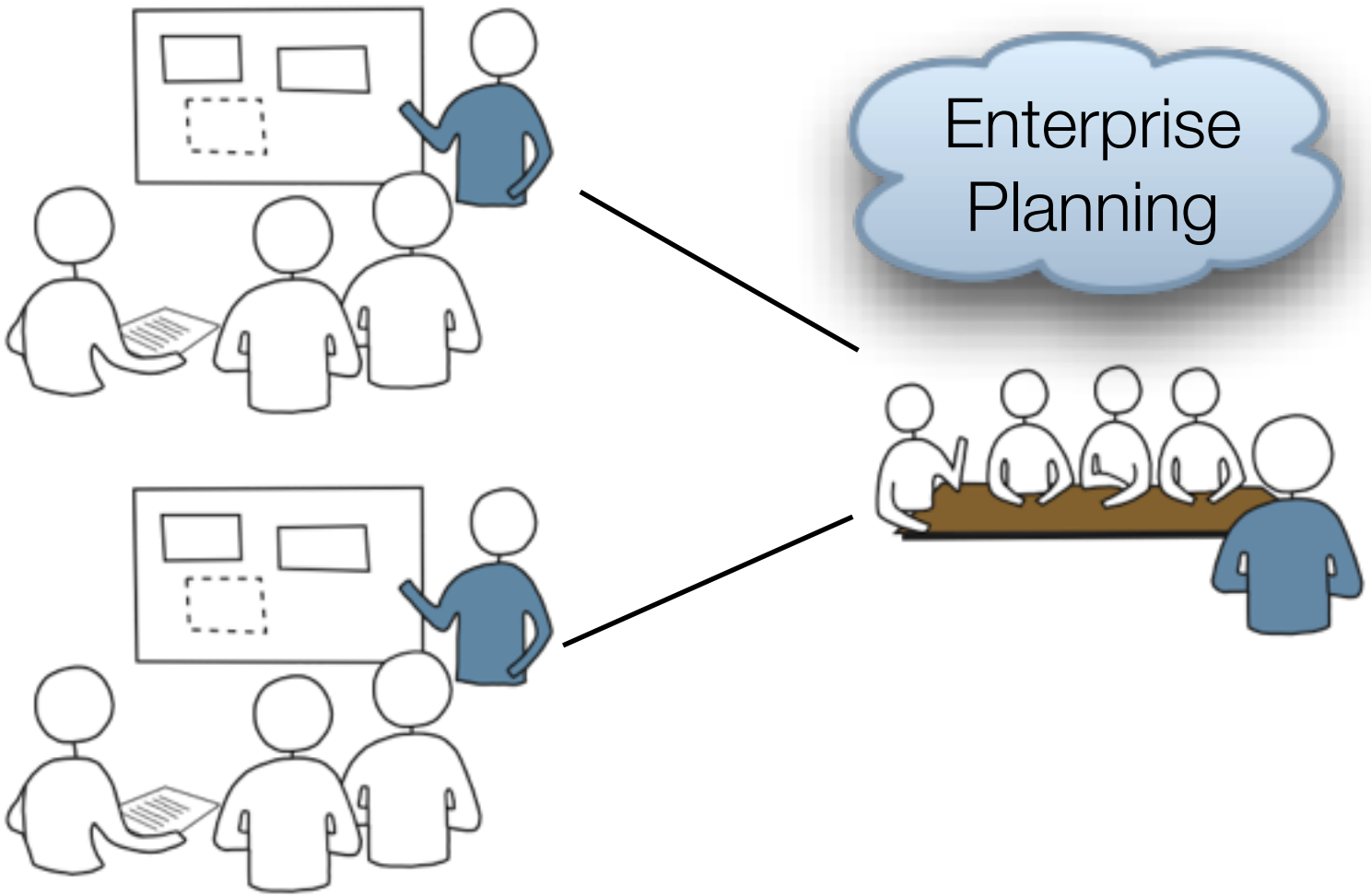
~OR~

- Surround existing functionality with automated functional tests
- Wrap platform interfaces with adapters
- Transfer knowledge of platform to more people
- Rewrite on more current platform
- Move thin slices of functionality to newer platform
- Start platform upgrade discussions and rearrange teams into known effective team

Team Configuration Patterns

- Virtual Architect Pattern
- Integration Team Pattern
- Component Shepherd Pattern
- Team Architect Pattern

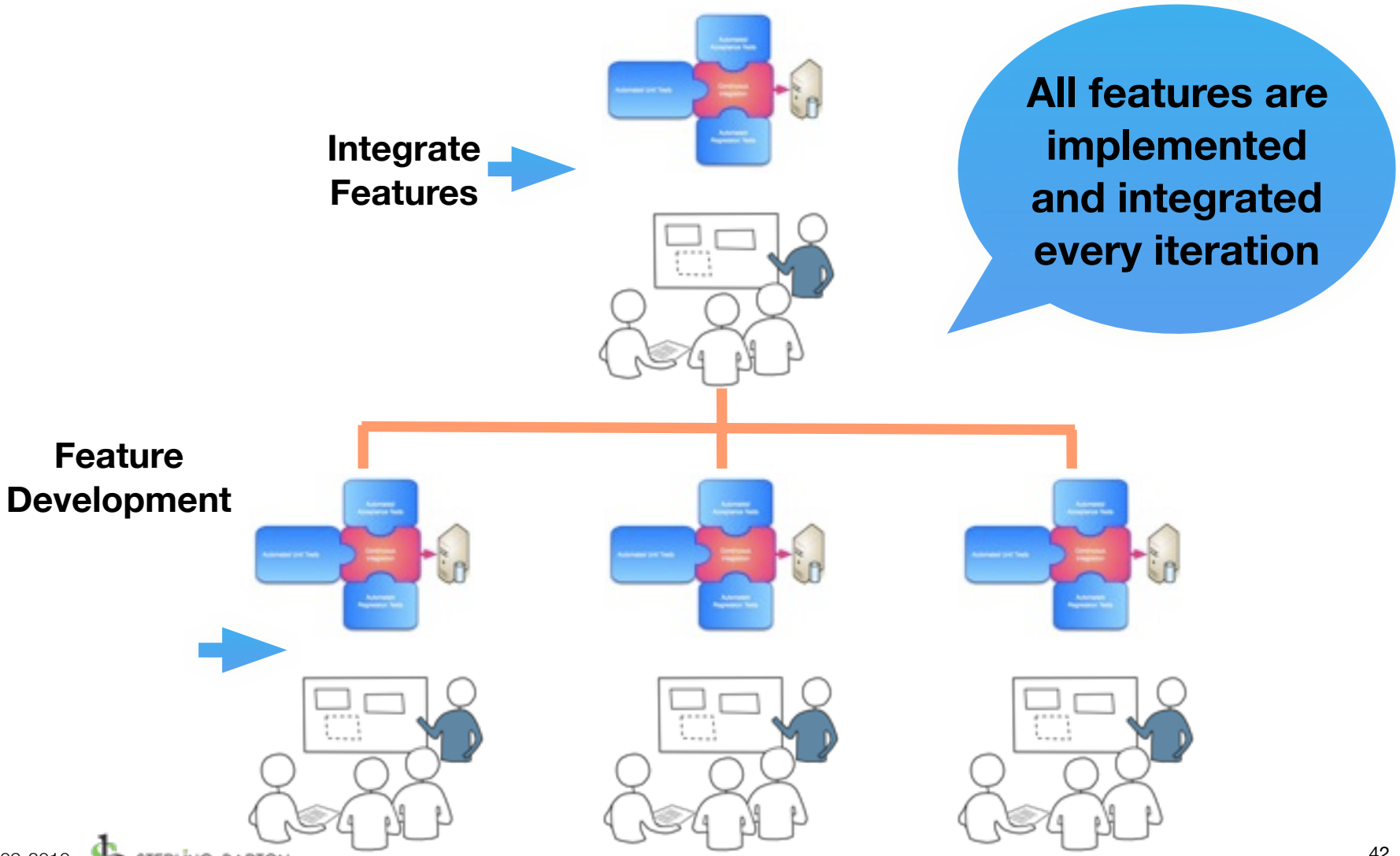
Virtual Team Pattern



Virtual Team Pattern

- Pros
  - Share architecture ideas and needs across teams
  - Based on verbal communication
- Cons
  - Usually singles out special Team Member role
  - Could lead to top-down architecture decisions
  - IT may gain extensive influence and begin to run Product Backlog prioritization for architecture needs

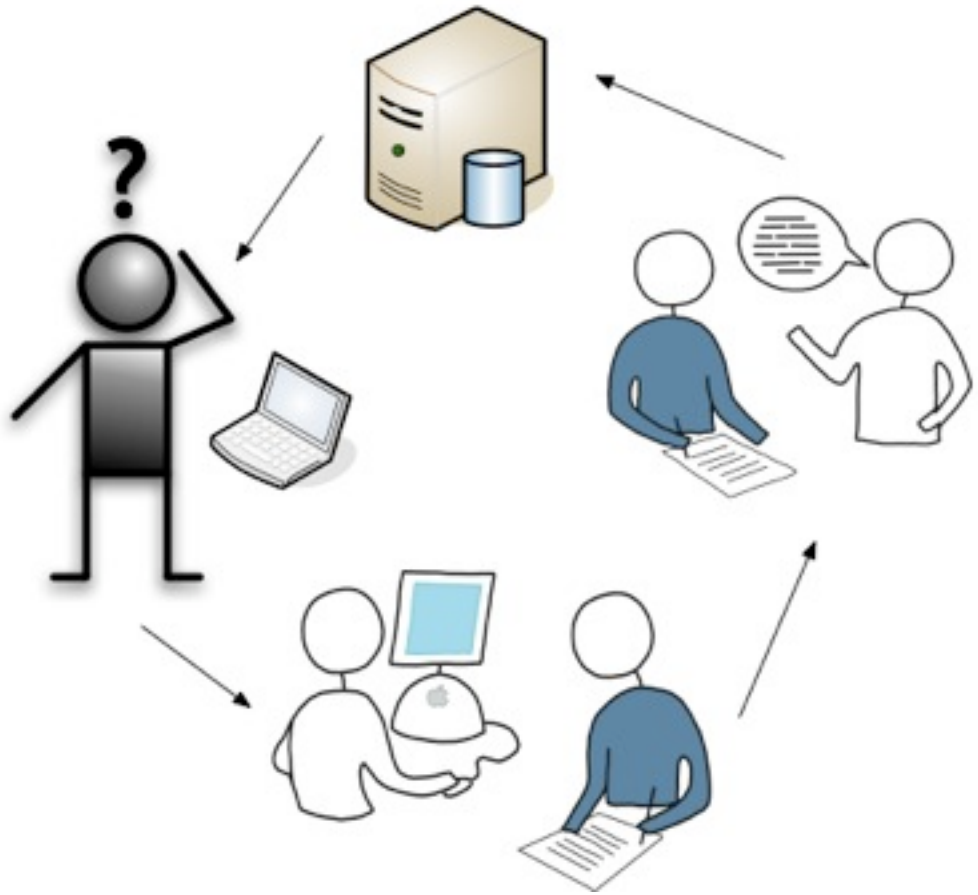
Integration Team Pattern



Integration Team Pattern

- Pros
  - Reduces complexity on Feature Teams
  - Forces delivery from Integration Team instead of interface and deployment designs
- Cons
  - Perpetuates specialized roles
  - Don’t always work on highest value Product Backlog items

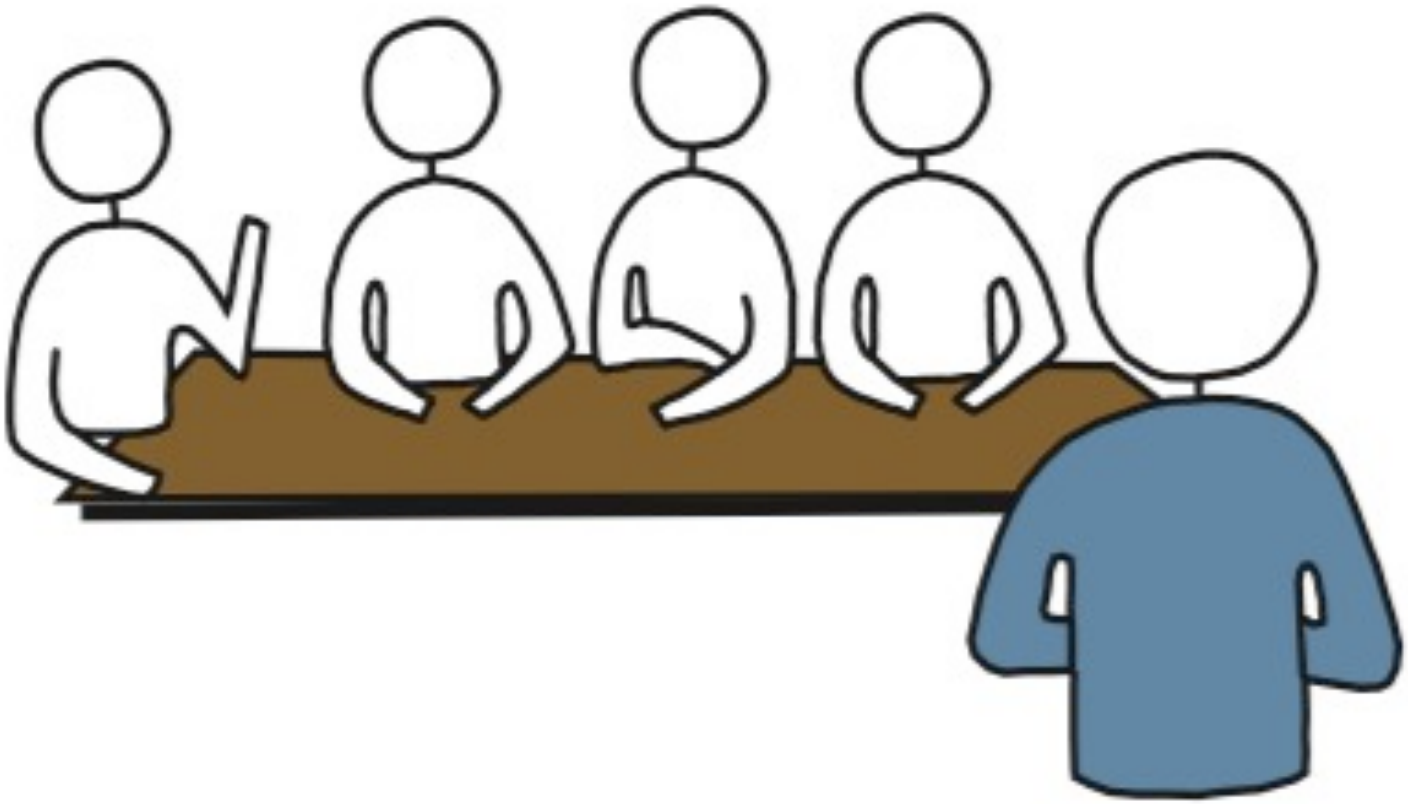
Component Shepherd Pattern



Component Shepherd Pattern

- Pros
  - Share more knowledge within organization to minimize platform experience debt
  - Work on highest value Product Backlog items
- Cons
  - Not always optimal as using individual knowledge
  - Difficult to learn multiple systems across Teams

Feature Team Pattern



Feature Team Pattern

- Pros
  - Team owns architecture decisions
  - Decisions are made close to implementation concerns
- Cons
  - May not have appropriate experience on Team
  - Team could get “stuck” on architecture decisions

What is possible?

High quality can be attained and enables accelerated feature delivery.

A Story: Field Support Application

- 2000+ users access application each day
- Application supports multiple perspectives and workflows from Field Support Operations to Customer Service
- Team of 5 people delivering features on existing Cold Fusion platform implementation
- Migrating to Spring/Hibernate in slices while delivering valuable features
- 36 2-week Sprints, 33 production releases, and only 1 defect found in production
- So, what was the defect you say? Let me tell you…

STERLING BARTON

Lets wrap this up...

What should I take away from this?

Principles for Managing Software Debt

- Maintain one list of work
- Emphasize quality
- Evolve tools and infrastructure continually
- Improve system design always
- Share knowledge across the organization
- And most importantly, get the right people to work on your software!

STERLING BARTON

Thank you

Questions and Answers

Chris Sterling – Sterling Barton, LLC

- Technology Consultant, Agile Consultant and Certified Scrum Trainer
- Developer of AgileEVM (www.AgileEVM.com), a project portfolio decision support tool
- Consults on software technology, Agile technical practices, Scrum, and effective management techniques
- Innovation Games® Trained Facilitator
- Open Source Developer and Consultant
- Software technology, architecture, release management, monitoring, and design consulting for Agile Teams

Email: chris@sterlingbarton.com
www.AgileEVM.com
Web: http://www.sterlingbarton.com
Blog: http://www.gettingagile.com
Follow me on Twitter: @csterwa