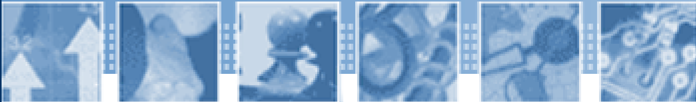
Access to the Experts

CUTTER CONSORTIUM



Technical Debt: Assessment and Reduction

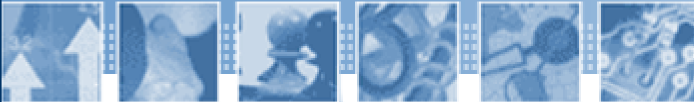
Israel Gat

Agile 2011
Salt Lake City, UT
August 8, 2011

Agenda

- Part I: Technical Debt in the Overall Context of the Software Process
- Part II: What Really is Technical Debt?
- Part III : Case Study – NotMyCompany, Inc.
- Part IV: The Tricky Nature of Technical Debt
- Part V: Unified Governance
- Part VI: Process Control Models
- Part VII: Reducing Technical Debt
- Part VIII: Takeaways

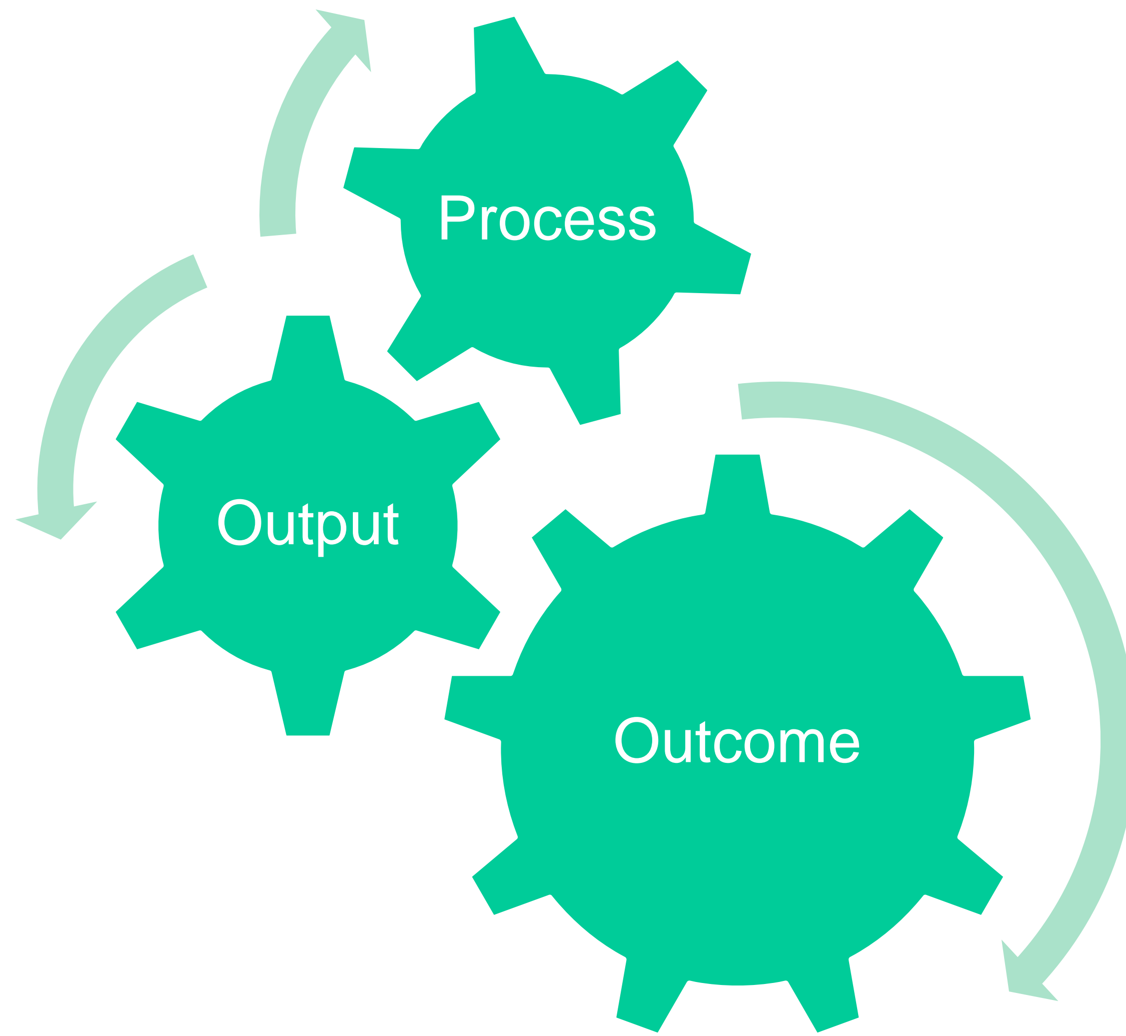
© Cutter Consortium



Part I:
Technical Debt in the Overall Context of the Software Process

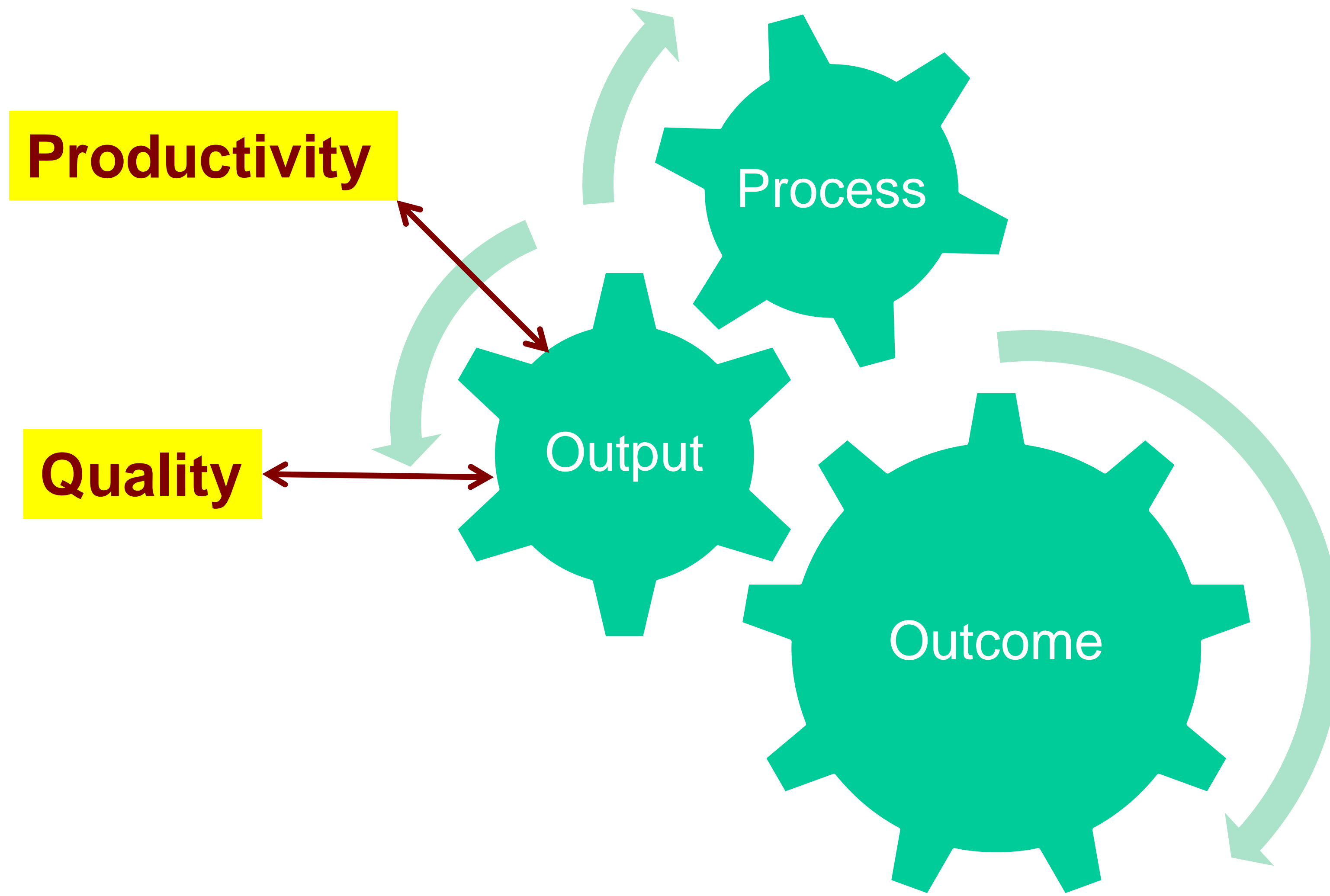
- A Holistic Model of the Software Process
- Two Aspects of Output
- Three Aspects of Technical Debt
- Six Aspects of Software

A Holistic Model of the Software Process



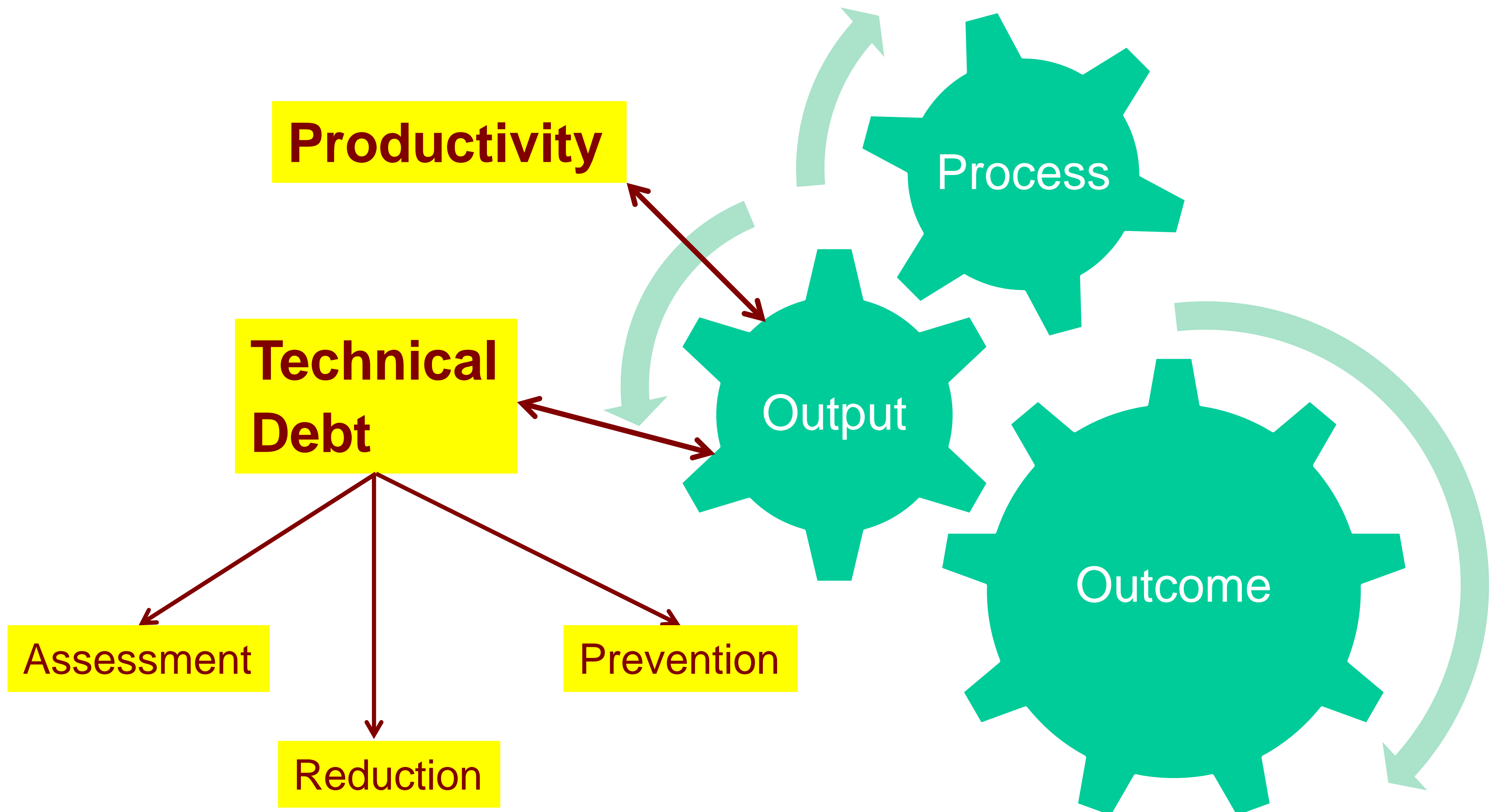
© Cutter Consortium

Two Aspects of Output



© Cutter Consortium

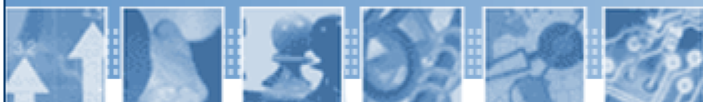
Three Aspects of Technical Debt



© Cutter Consortium

Six Aspects of Software

© Cutter Consortium



Part II: What Really is Technical Debt?

- What’s in a Metaphor?
- Code Analysis
- Time is Money
- Monetizing Technical Debt
- Typical Stakeholder Dialog Around Technical Debt
- Analysis of the Cassandra Code
- Project Dashboard

What's in a Metaphor?

- Ward Cunningham's Metaphor:
  - "A little debt speeds development so long as it is paid back promptly with a rewrite"
- Definition for today:
  - "Quality issues in the code other than function/feature completeness"
    - It is about doing the system right ("Intrinsic Quality")
    - ***Not*** about doing the right system ("Extrinsic Quality")
- Typical technical debt components:
  - Complexity
  - Duplication
  - Rule violations
  - Test coverage
  - Documentation

© Cutter Consortium

Code Analysis

- One technical debt tends to pile over another, which piles over yet another technical debt that piles…
  - To find your current level of debt, you can't simply add the week you borrowed last year to the two weeks you borrowed three months ago
  - Rather, you need to inspect the code

© Cutter Consortium

Time is Money

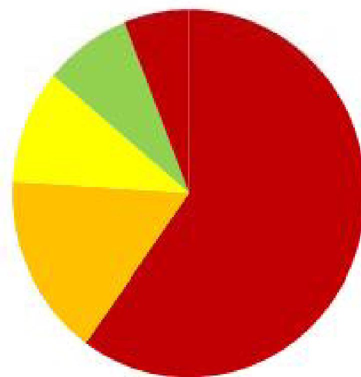
- Think of the amount of money the borrowed time represents – the $$ grand total required to eliminate all issues found in the code

© Cutter Consortium

Example I: Monetized Technical Debt

- Accrued technical debt in the amount of $500K
- On 200K lines of code
- The makeup of the debt is represented in the pie chart below

Breakdown of Technical Debt



- Test coverage
- Duplication
- Rule violations
- Complexity

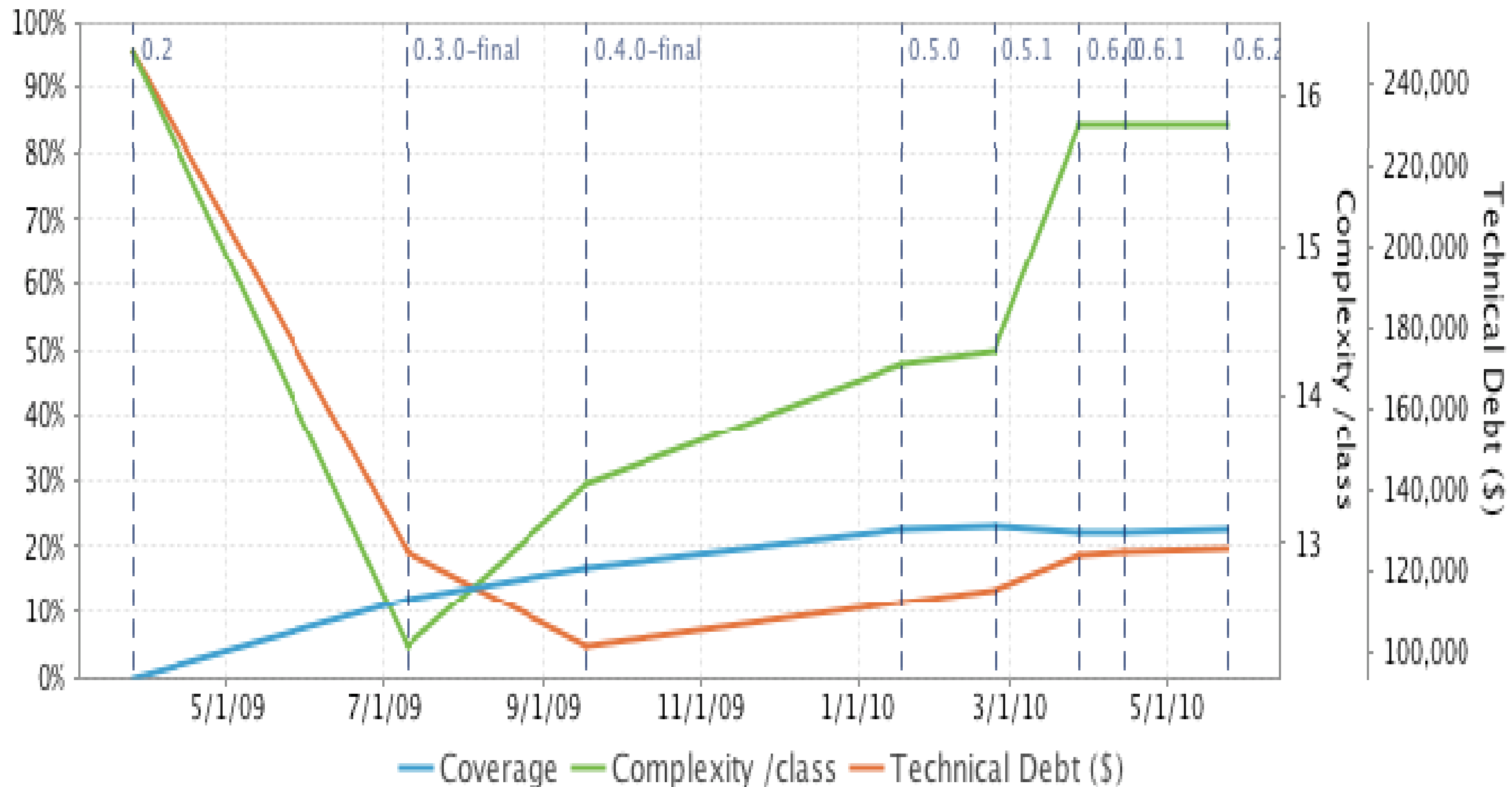
© Cutter Consortium

Typical Stakeholders Dialog Around Technical Debt

- "Technical debt of $500K over 200K lines of code"
- "60% of the debt is due to lack of unit test coverage"
- "'Pay back' 70% of unit test coverage debt prior to shipping the software"
- "Other kinds of debt will be paid back during the first year after release"
- "Rule violation will be the #1 priority during the period after release"
- "Once we reach technical debt level of $100K we will shift back resources from technical debt reduction to feature development"

© Cutter Consortium

Example II: Analysis of the Cassandra Code



Since the 0.4.0 release both Complexity (per class) and Technical Debt have increased.

© Cutter Consortium

Example III: Project Dashboard

Version 6.x - Mon, 26 Jul 2010 13:58 - profile Nemo rules

Lines of code
162,306
325,036 lines
87,756 statements
1,060 files

Classes
1,447
103 packages
14,271 methods
+1,262 accessors

Comments
26.6%
58,891 lines
59.1% docu. API
5,418 undocu. API
1,164 commented LOCs

Duplications
7.1%
22,998 lines
566 blocks
174 files

Complexity
3.1 / method
30.9 / class
42.2 / file
Total: 44,773

10000
5000
0
1 2 4 6 8 10 12
Methods Classes

Events All

2010-07-26	Version	6.x	
2009-06-07	Version	6.0.x	
2009-02-15	Alert	Orange	

Key : org.apache:tomcat
Language : java
Alerts feed

Rules compliance
83.7%
Usa. Eff. Mai. Por. Rel.

Violations
10,072

Blocker	0
Critical	0
Major	8,794
Minor	65
Info	1,213

Alerts : Duplicated lines (%) > 5.

SIG Maintain. Model

(A)nalysability	-
(C)hangeability	0
(S)tability	--
(T)estability	--

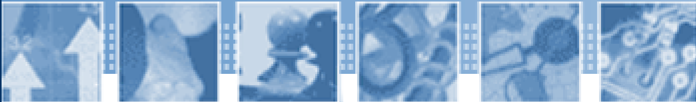
T A S C

Tags
356
0 mandatory
356 optional
FIXME TODO @todo @deprecate

Technical Debt
11.0%
$ 341,563
683 man days
Duplication Violations Complexity Comments
No information available on coverage
No information available on design

Source: Chris Sterling

© Cutter Consortium



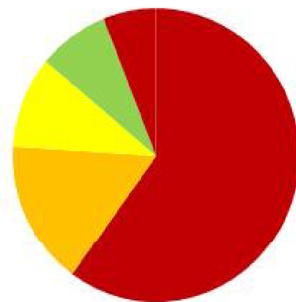
Part III: Case Study – NotMyCompany, Inc.

- NotMyCompany Highlights
- Modernizing Legacy Code
- Error Proneness

NotMyCompany Highlights

- Hosted eCommerce platform for small retailers:
  - One stop shopping
  - White-glove service
  - Three nines availability
  - Business as a service (warehousing, distribution)
- Challenges:
  - Legacy code – 200KLOC - $500K technical debt

Breakdown of Technical Debt



- Test coverage
- Duplication
- Rule violations
- Complexity

© Cutter Consortium

NotMyCompany Highlights (Cont'd)

- Expansion – Acquisition of SocialAreUS
- How Often Should the Line be Stopped?
- Agile Versus ITIL

© Cutter Consortium

Exercise #1 – Modernizing Legacy Code

© Cutter Consortium

Exercise – Modernizing Legacy Code

- Read the NotMyCompany case study through the section entitled ***Exercise #1*** in the handout

- Discuss the following questions in your table/group:
  1. Does the strategy summarized in the slide "Typical Stakeholders Dialog" make sense as a debt reduction strategy?
  2. Which best practices would you recommend for implementing this strategy?
  3. What would be a compelling argument for adopting a 'Reduce Complexity First' strategy?

- Report back

- Time allocation – 40 minutes:
  - 30 minutes for reading the case study and group discussion
  - 10 minutes for group reports

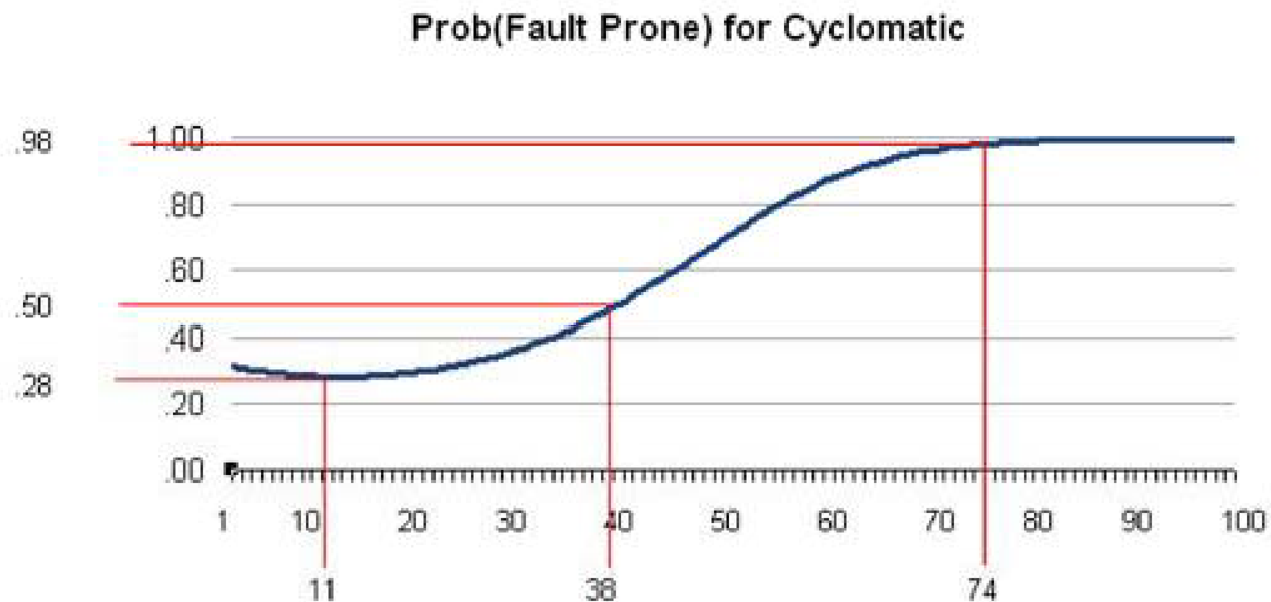
© Cutter Consortium

Continue Reading Only After Reporting Back on the Exercise

© Cutter Consortium

Answer to Question #3 in Exercise #1

- Cyclomatic complexity in excess of ~30 per file for a significant number of Java files

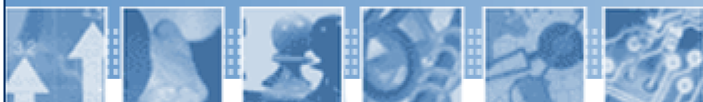


(Source: http://www.enerjy.com/blog/?p=198)

© Cutter Consortium

Access to the Experts

CUTTER CONSORTIUM



Part IV: The Tricky Nature of Technical Debt

- The Explicit Form of Technical Debt
- The Implicit Form of Technical Debt
- The Strategic Impact of Technical Debt
- No Good Strategy Following Prolonged Neglect

The Explicit Form of Technical Debt

- Resource allocation decisions:
  - “Functional testing is good enough for us… no need to waste precious resources to do unit testing…”

    [Confession of a VP of development with numerous Cyclomatic complexity readings in the hundreds…]

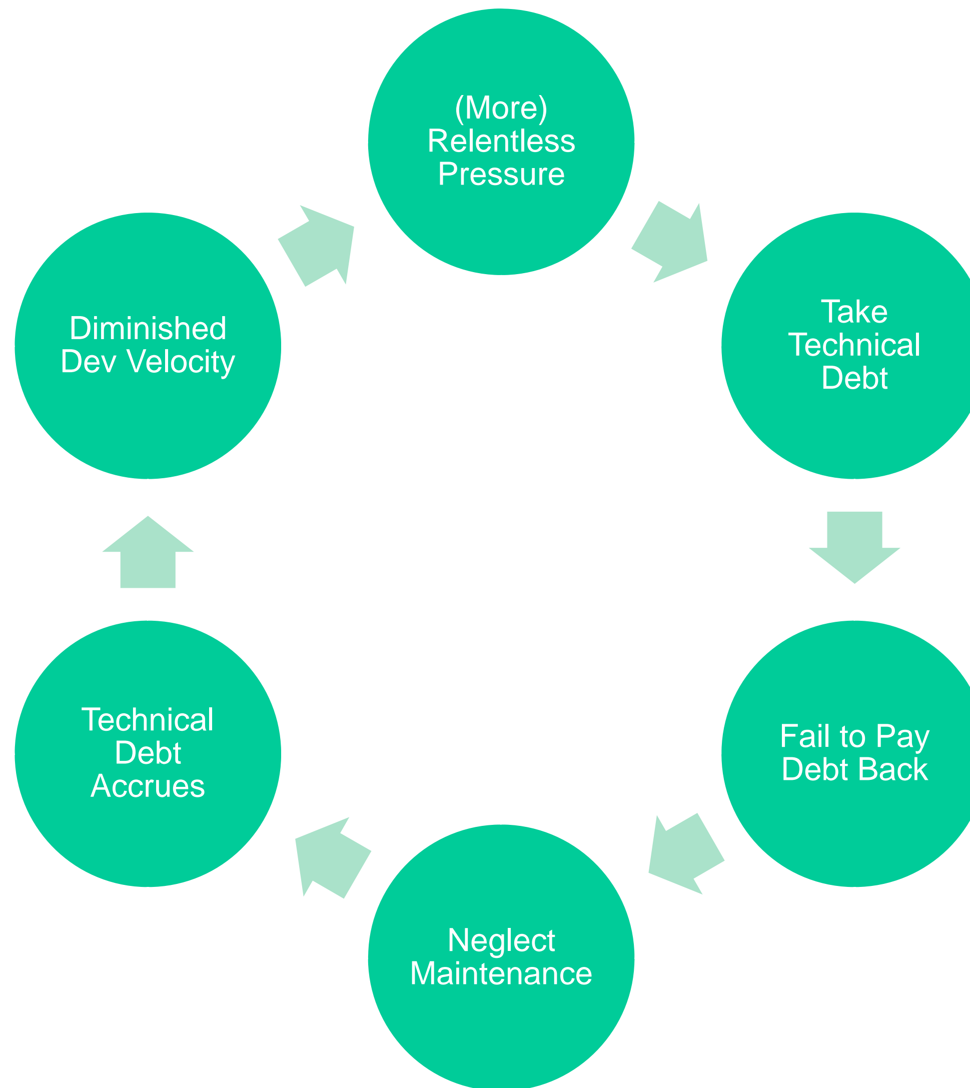
© Cutter Consortium

The Implicit Form of Technical Debt

- Implicit forms – in the nature of things:
  - Relentless function/feature pressure leads to taking technical debt and neglecting measures to keep software decay in check

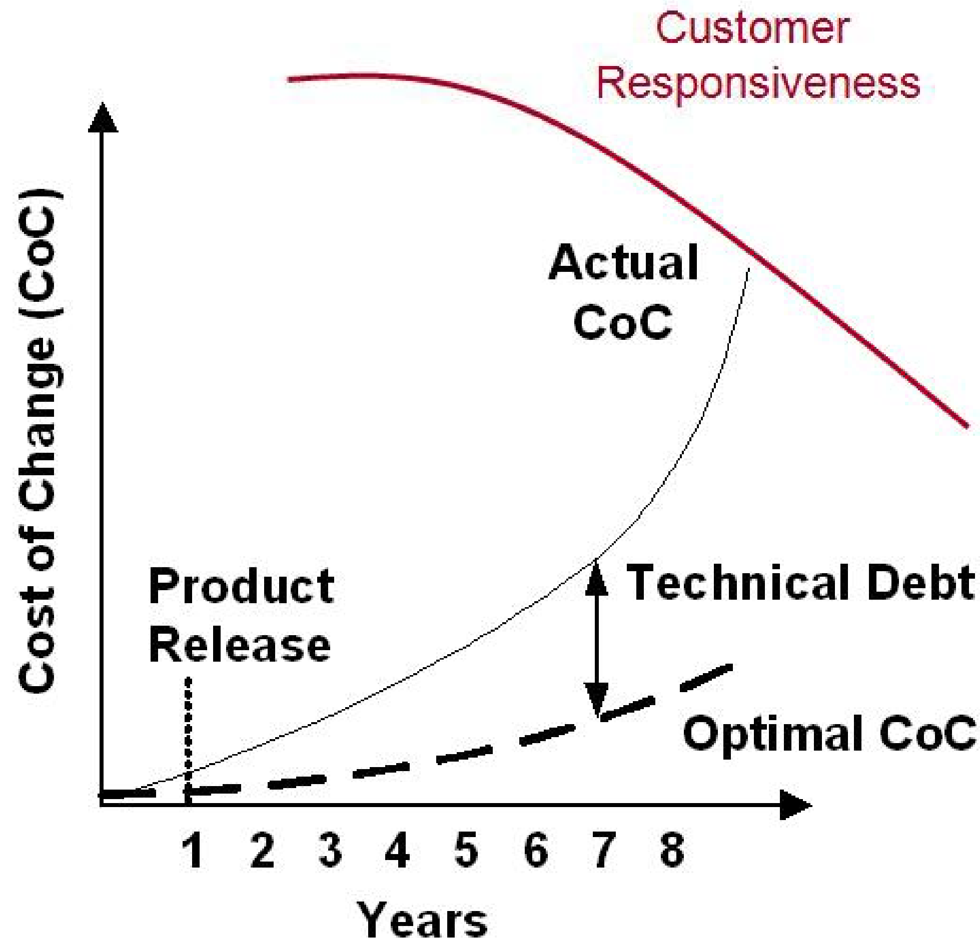
© Cutter Consortium

The Vicious Cycle of Technical Debt



© Cutter Consortium

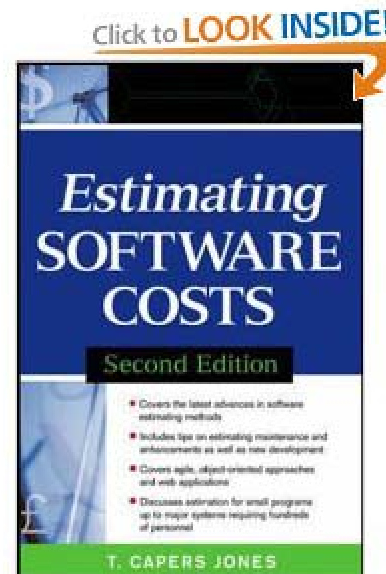
The Strategic Effect of Technical Debt



- Once on far right of curve, all choices are hard
- If nothing is done, it just gets worse
- In applications with high technical debt, estimating is nearly impossible
- Only 3 strategies
  - Do nothing, it gets worse
  - Replace, high cost/risk
  - Incremental refactoring, commitment to invest

No Good Strategy Following Prolonged Neglect

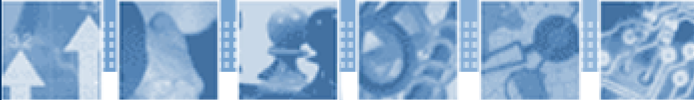
- "Indeed, the economic value of lagging applications is questionable after about three to five years. The degradation of initial structure and the increasing difficulty of making updates without 'bad fixes' tends towards negative returns on investment (ROI) within a few years."



© Cutter Consortium

Access to the Experts

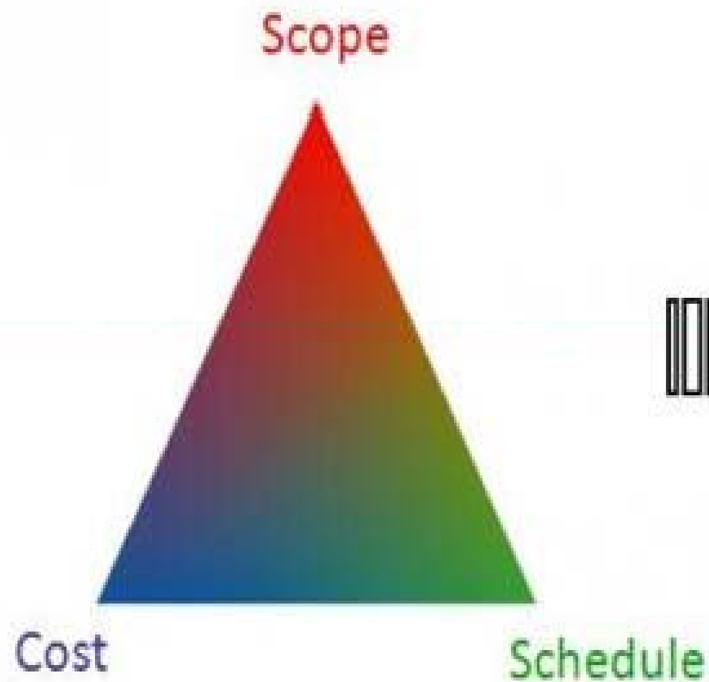
CUTTER CONSORTIUM

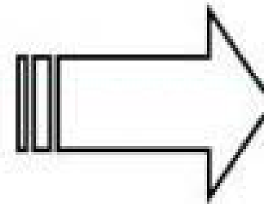


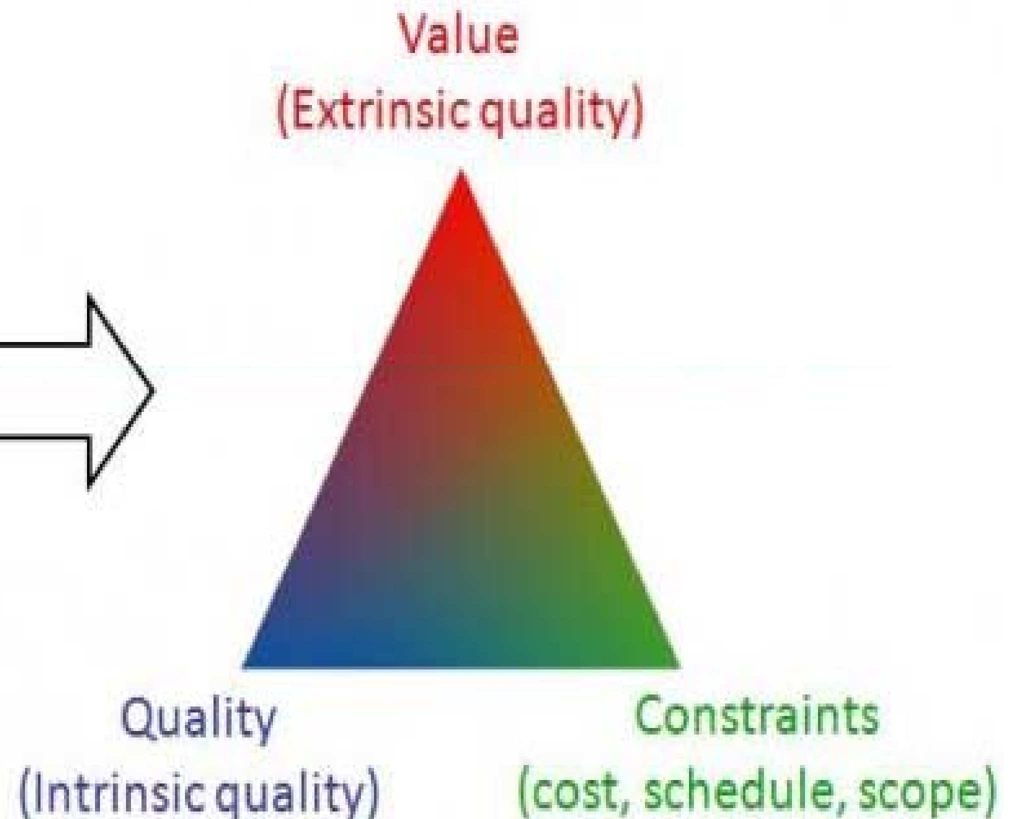
Part V: Unified Governance

- How We View Success
- Three Core Metrics
- Productivity, Affordability, Risk
- What is the Real ROI?

How We View Success: An Agile Approach to Governance

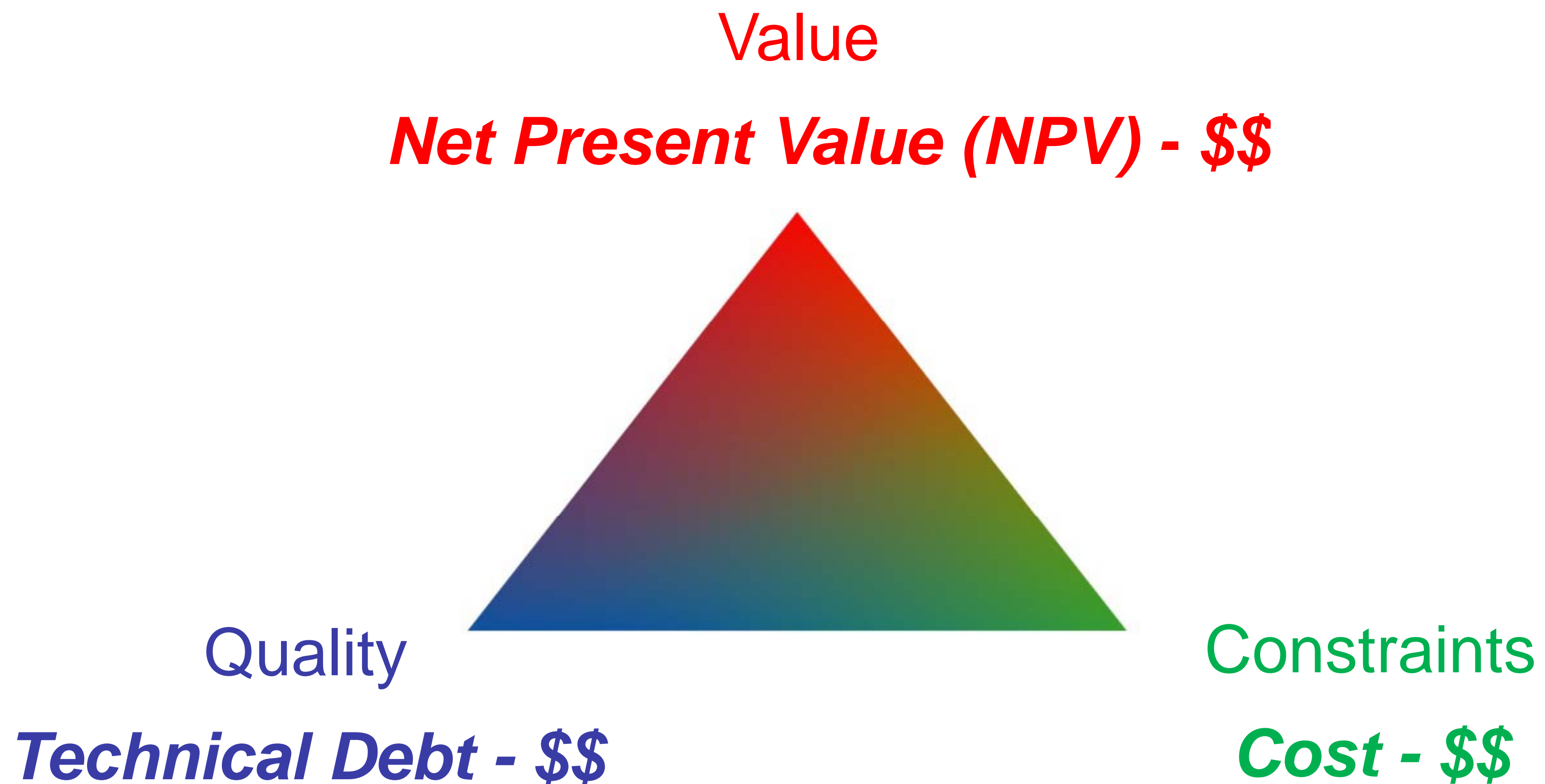






© Jim Highsmith
© Cutter Consortium

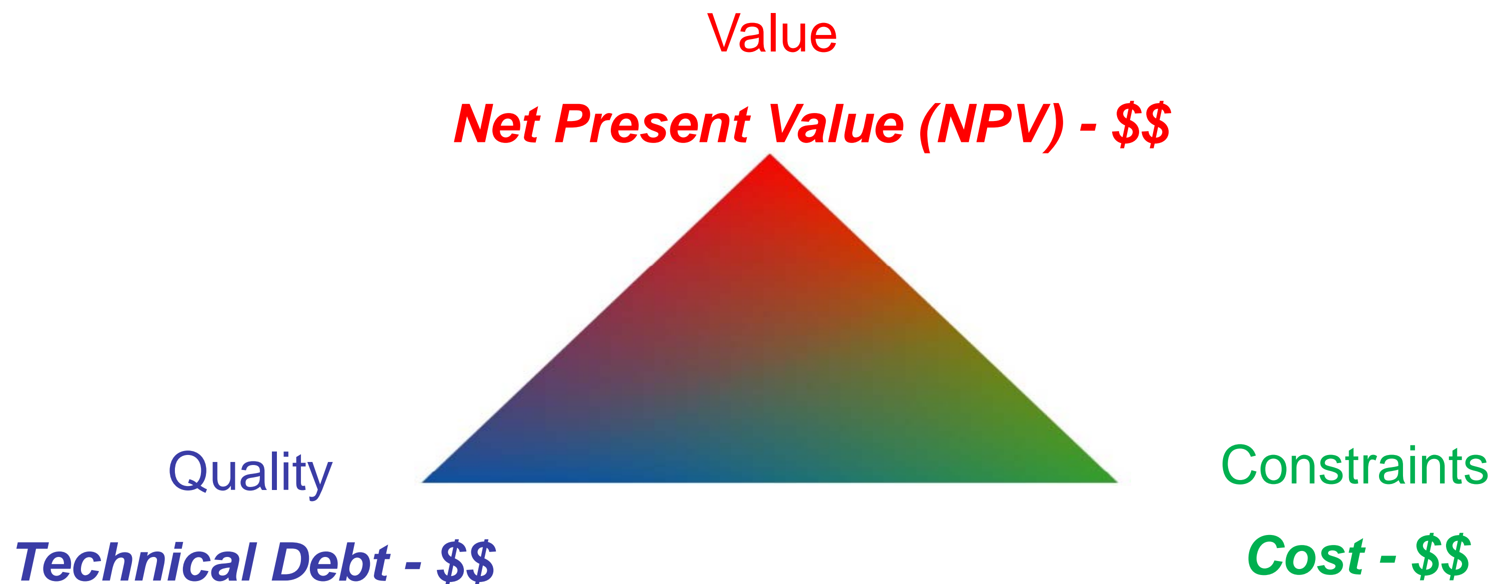
Three Core Metrics



© Israel Gat
© Cutter Consortium

Productivity, Affordability, Risk

- Long-term productivity: Cost > Technical Debt
- Long-term affordability: Value >> Cost + Technical Debt
- Unifying equation: Value >> Cost > Technical Debt
- Risk: Imbalance(s) between the three core metrics

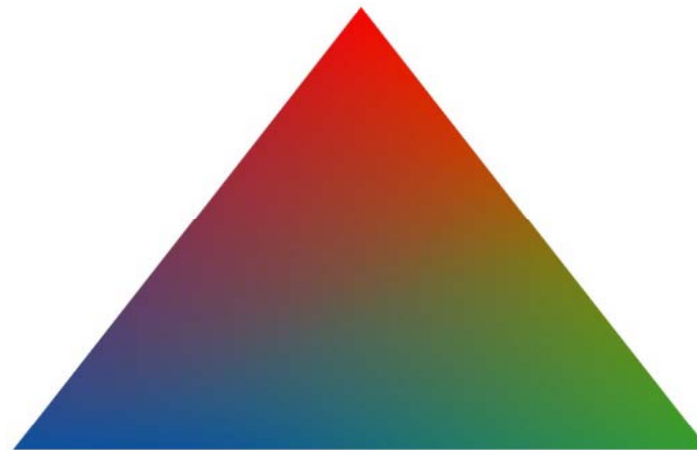


© Israel Gat
© Cutter Consortium

What is the Real ROI?

Is your rate of return on investment **900%** or is it actually **233%?!**

Expected Final Value of Investment - $10M



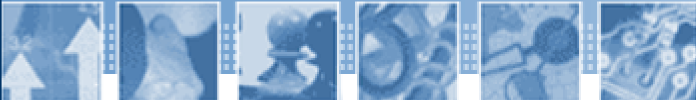
Technical Debt - $2M

Cost - $1M

© Cutter Consortium

Access to the Experts

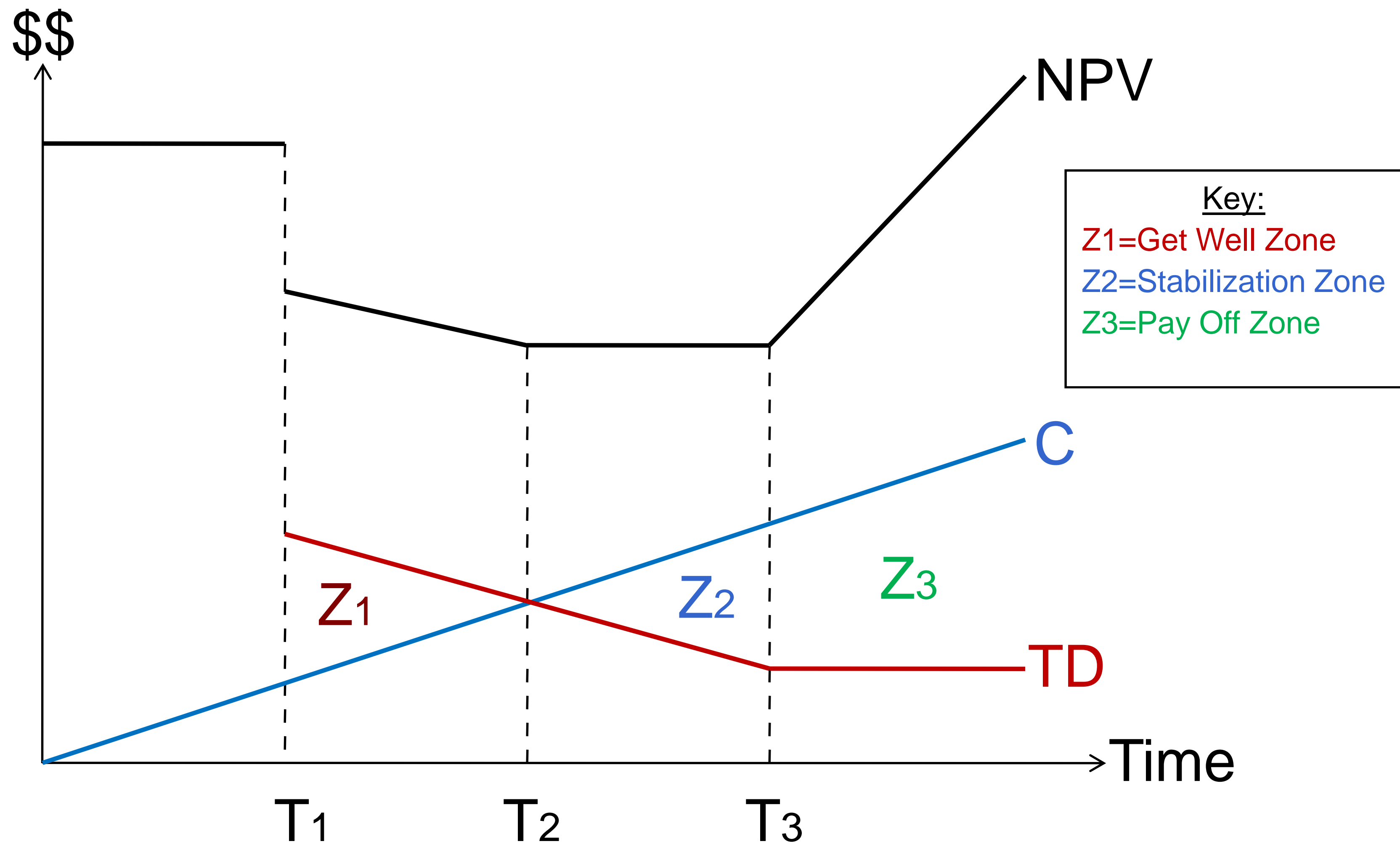
CUTTER CONSORTIUM



Part VI: Process Control Models

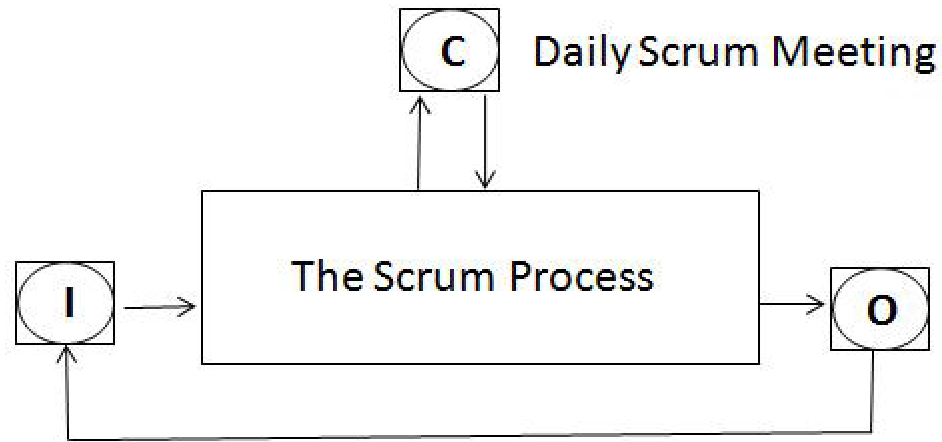
- A Typical Technical Debt Pattern
- Process Control View of Scrum
- Integration of Technical Debt in the Agile Process
- Using Statistical Process Control Methods

A Typical Technical Debt Pattern



© Israel Gat
© Cutter Consortium

Process Control View of Scrum



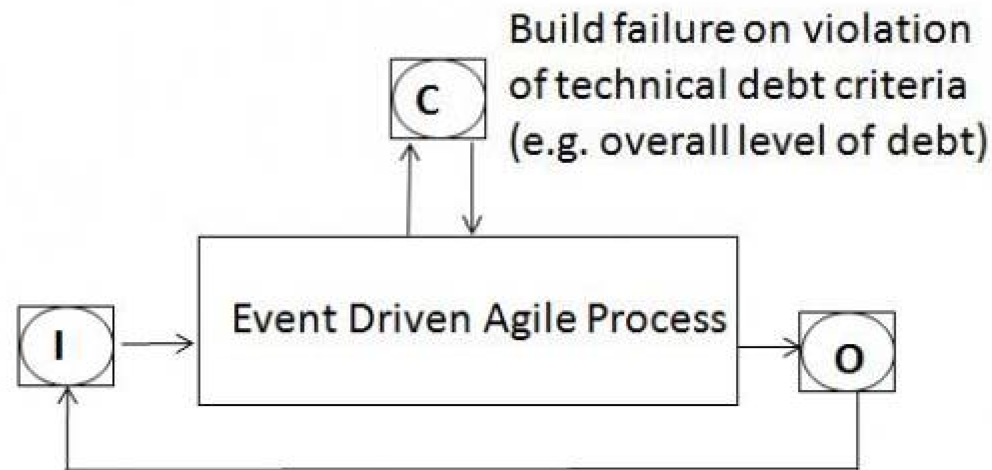
Legend:

I=Input=(Requirements)
C=Control Unit
O= Output=(Code increment)

Source: Agile Software Development with Scrum

© Cutter Consortium

Integration of Technical Debt in the Agile Process



Legend:

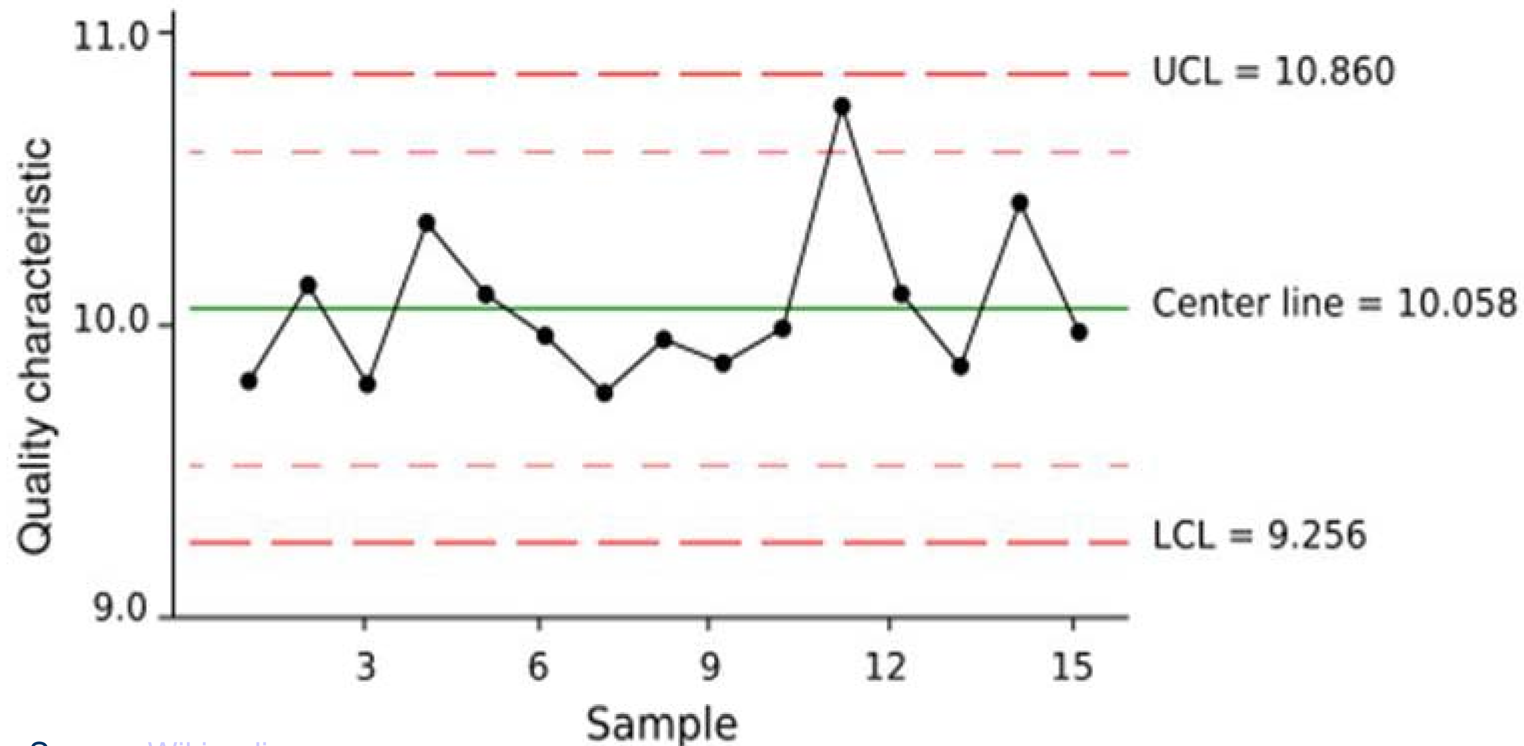
I=Input=(Requirements)
C=Control Unit= ('Stop the line' & convene a team meeting)
O=Output=(Code Increment in the build)

© Copyright 2010 Israel Gat

© Cutter Consortium

Using Statistical Process Control Methods

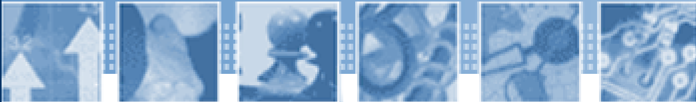
- Use Statistical Process Control methods on Technical Debt samples
  - In the example below, Cyclomatic Complexity per Java Class can be used as the Quality Characteristic



Source: Wikipedia

© Cutter Consortium

Access to the Experts CUTTER CONSORTIUM



Part VII: Reducing Technical Debt

- A Framework for Thinking about and Acting on Technical Debt Issues
- Portfolio Governance

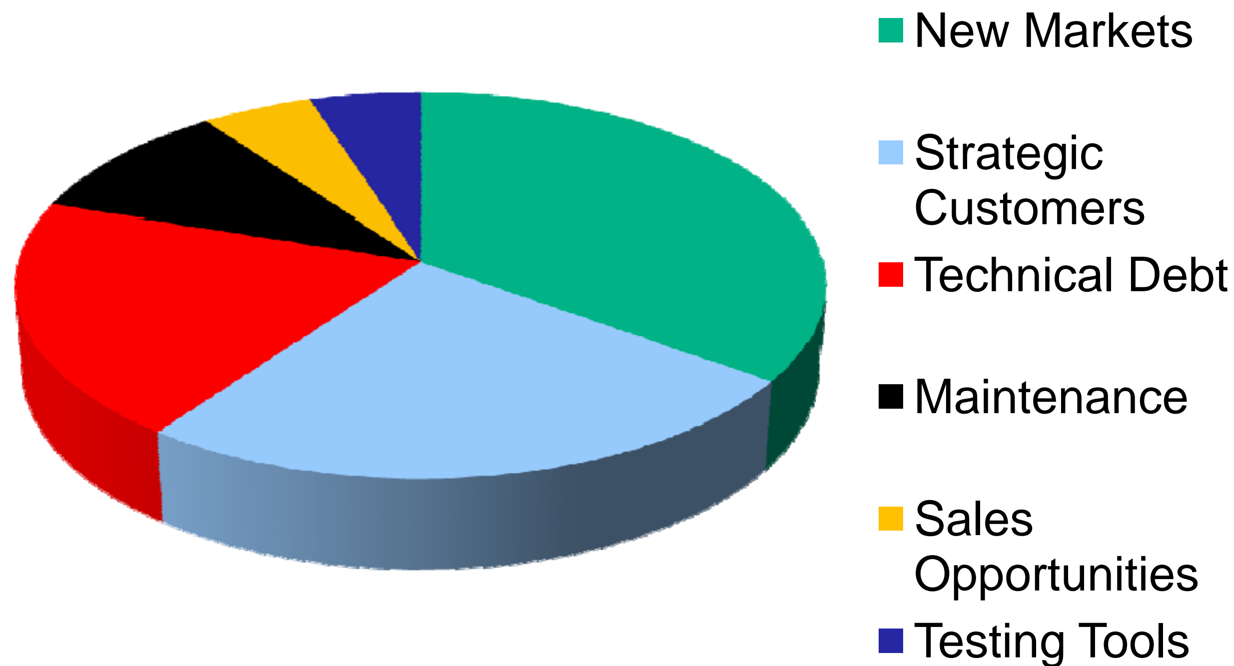
A Framework for the Technical Debt Initiative

- To become actionable, follow the technical debt assessment with a technical debt reduction initiative:
  - SWAT team
  - Evangelism
  - Agile methods
  - Technical debt items as an integral part of the product backlog of every team:
    - If you are starting the technical debt initiative amidst converting to Agile, introduce technical debt as part of the conversion to Agile
  - Governance of the Technical Debt Initiative as a strategic investment theme

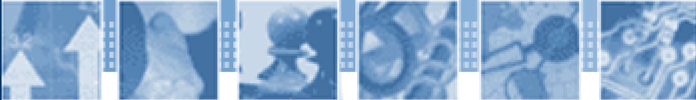
© Cutter Consortium

Portfolio Governance

- Intentionality through Technical Debt as a Strategic Investment Theme



© Cutter Consortium



Part VIII: Takeaway

- Nine Simple Takeaway
- Connecting the dots

Nine Simple Takeaways

- Technical debt shifts the emphasis in software development from ***proficiency*** of the software process to the ***output*** of the process
- It enables moving on and up from ***Random Checks*** to ***Continuous Inspection*** of the code
- It changes the playing fields from ***qualitative*** assessment to ***quantitative*** measurement of the quality of software
- It is an effective ***antidote*** to the relentless function/feature pressure
- It is applicable to ***any amount of code***
- It can be applied at ***any point in time*** in the software life-cycle
- It can be used with ***any software method***, not “just” Agile

© Cutter Consortium

Nine Simple Takeaways (Cont'd)

- It enables effective governance of the software ***process***
- It enables effective governance of the product ***portfolio***

© Cutter Consortium