Cómo reducir la fricción en el desarrollo de software

Philippe Kruchten

Mexico, June 26, 2014

Reducing Friction in Software Development

Philippe Kruchten

Mexico, June 26, 2014

Philippe Kruchten, Ph.D., P.Eng., CSDP

Professor of Software Engineering
NSERC Chair in Design Engineering
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, BC Canada
pbk@ece.ubc.ca

Founder and president
Kruchten Engineering Services Ltd
Vancouver, BC Canada
philippe@kruchten.com

 3

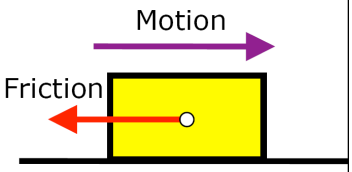
Friction

"There is still much friction in the process of crafting complex software; the goal of creating quality software in a repeatable and sustainable manner remains elusive to many organizations, especially those who are driven to develop in Internet time."

Grady Booch's keynote at ICSE 2000 in Limerick, Ireland

Friction

"Friction: the resistance that one surface or object encounters when moving over another."

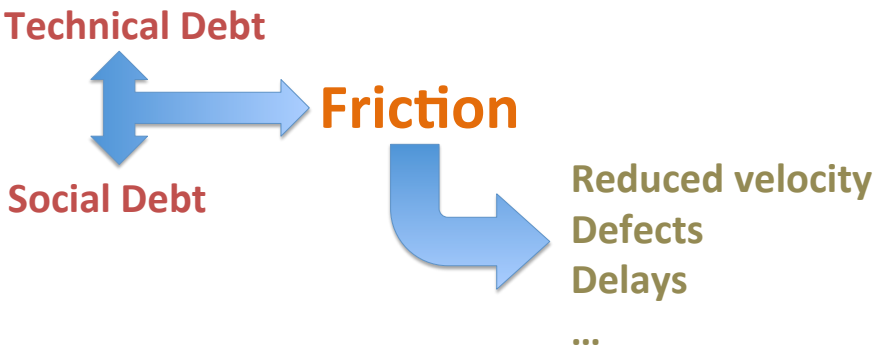


In software development, friction is the *set of phenomena that limits or constraints our progress*, therefore reduces our velocity (or productivity).

Technical debt causes friction.

 6

Friction and Debt



 7

Outline

- What is technical debt?
- The technical debt landscape
- Causes of technical debt
  - Cost vs. value
- Limits of the metaphor
- Tackling Technical debt
- Friction in software development

 8

Technical Debt

- Concept introduced by Ward Cunningham
- Often mentioned, rarely studied
- All experienced software developers “feel” it.
- Drags long-lived projects and products down

 9

Origin of the metaphor

- Ward Cunningham, at OOPSLA 1992

"Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite…
The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object-oriented or otherwise."

Cunningham, OOPSLA 1992

 10

Technical Debt (S. McConnell)

- Implemented features (visible and invisible) = assets = non-debt
- Type 1: unintentional, non-strategic; poor design decisions, poor coding
- Type 2: intentional and strategic: optimize for the present, not for the future.
  - 2.A short-term: paid off quickly (refactorings, etc.)
    - Large chunks: easy to track
    - Many small bits: cannot track
  - 2.B long-term

McConnell 2007

 11

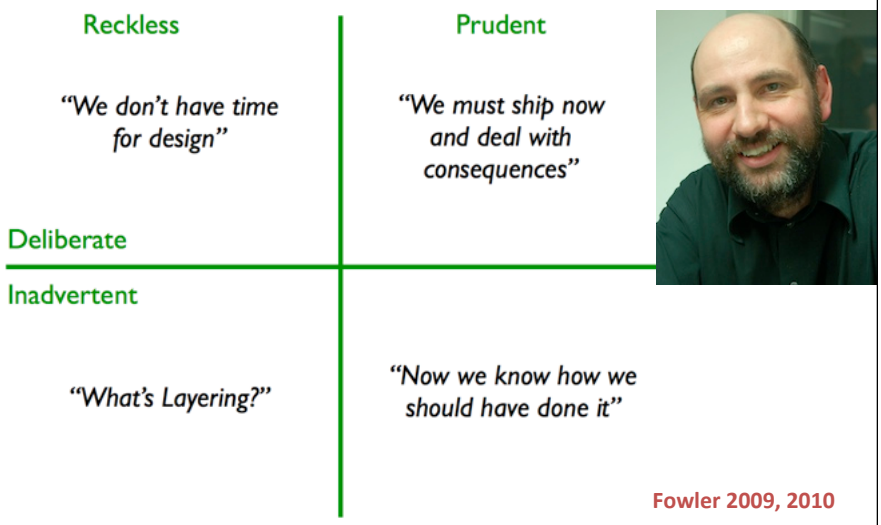
Technical Debt Definition (2013)

- A design or construction approach that is expedient in the short term, but that creates a technical context in which the same work will cost more to do later than it would cost to do now (including increased cost over time).

McConnell 2013

 12

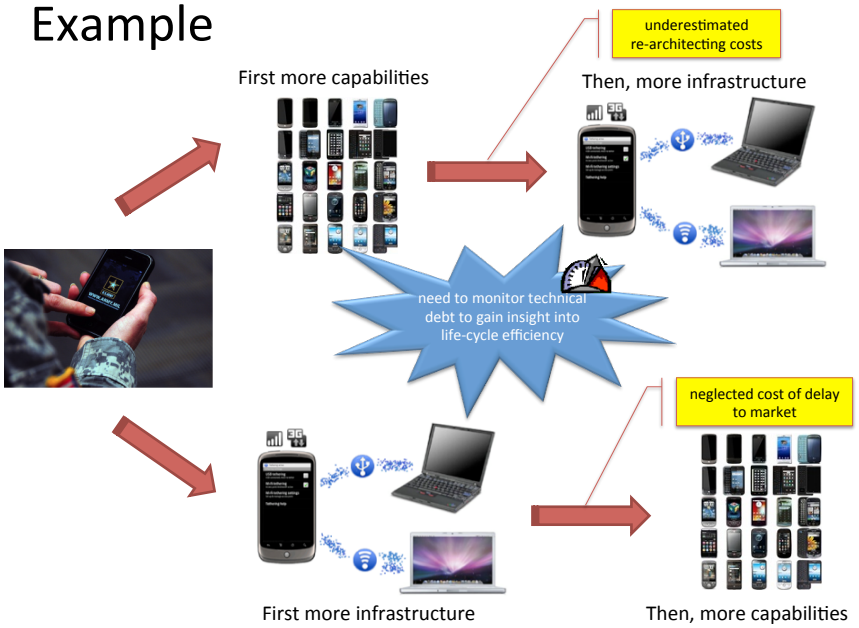
Technical Debt (M. Fowler)



Fowler 2009, 2010

 13

Example



Ozkaya, SEI,2010

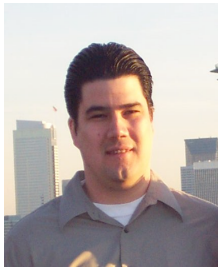
 14

Technical Debt (Chris Sterling)

- Technical Debt: issues found in the code that will affect future development but not
  those dealing with feature completeness.

Or

- Technical Debt is the decay of component and intercomponent behaviour when the application functionality meets a minimum standard of satisfaction for the customer.



 15

Time is Money (I. Gat)

- Convert this in monetary terms:
  "Think of the amount of money the borrowed time represents – the grand total required to eliminate all issues found in the code"

Gat 2010

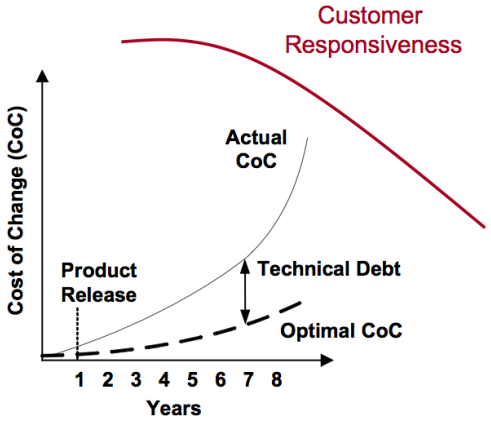
Example: TD is the sum of…

- Code smells 167 person days
- Missing tests 298 person days
- Design 670 person days
- Documentation 67 person days

Totals

Work	1,202 person x days
Cost	$577,000

Tech Debt (Jim Highsmith)

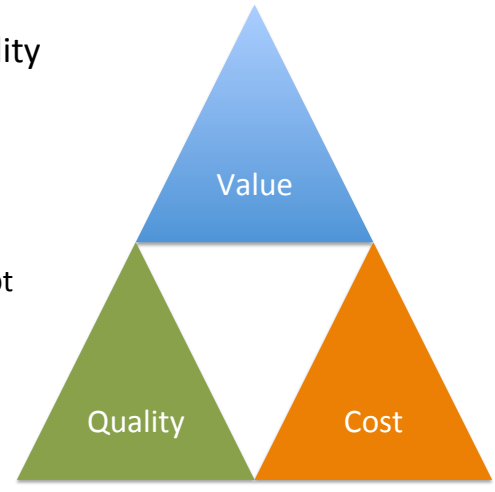


- Once on far right of curve, all choices are hard
- If nothing is done, it just gets worse
- In applications with high technical debt, estimating is nearly impossible
- Only 3 strategies
  1. Do nothing, it gets worse
  2. Replace, high cost/risk
  3. Incremental refactoring, commitment to invest

Source: Highsmith, 2009

Value, Quality, Constraints

- Value = extrinsic quality
  - Metric: Net present value
- Quality = intrinsic quality
  - Metric: Technical debt
- Constraints = cost, schedule, scope
  - Metric: Cost



Highsmith 2010

State of affairs

- Opinions, posturing, proclamations
- Little objective facts

"...there is a plethora of attention-grabbing pronouncements in cyberspace that have not been evaluated before they were published, often reflecting the authors' guesses and experience on the subject of Technical Debt."

Spinola et al. 2013

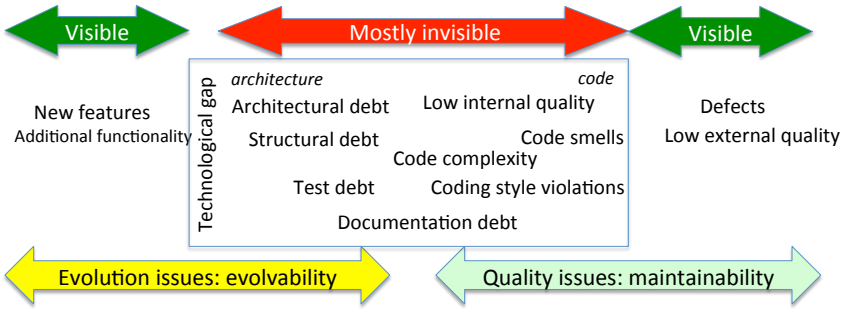
 20

Outline

- What is technical debt?
- The technical debt landscape
- Causes of technical debt
  - Cost vs. value
- Limits of the metaphor
- Tackling Technical debt
- Friction in software development

 21

Technical debt landscape



Kruchten et al 2012

 22

Outline

- What is technical debt?
- The technical debt landscape
- Causes of technical debt
  - Cost vs. value
- Limits of the metaphor
- Tackling Technical debt
- Friction in software development

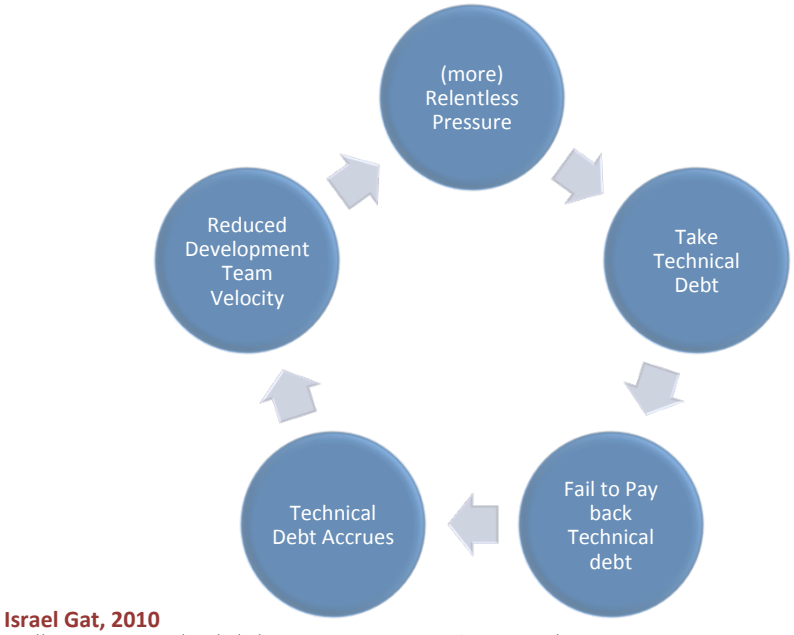
 23

Causes of Technical Debt

TECHNOLOGY	PROCESS
• Technology limitations • Legacy code • COTS • Changes in technology • Project maturity	• Little consideration of code maintenance • Unclear requirements • Cutting back on process (code reviews) • Little or no history of design decisions • Not knowing or adopting best practices
PEOPLE	**PRODUCT**
• Postpone work until needed • Making bad assumptions • Inexperience • Poor leadership/team dynamics • No push-back against customers • "Superstars" – egos get in the way • Little knowledge transfer • Know-how to safely change code • Subcontractors	• Schedule and budget constraints • Poor communication between developers and management • Changing priorities (market information) • Lack of vision, plan, strategy • Unclear goals, objectives and priorities • Trying to make every customer happy • Consequences of decisions not clear

Lim et al. 2012

 24



Israel Gat, 2010

http://theagileexecutive.com/2010/09/20/how-to-break-the-vicious-cycle-of-technical-debt/

 25

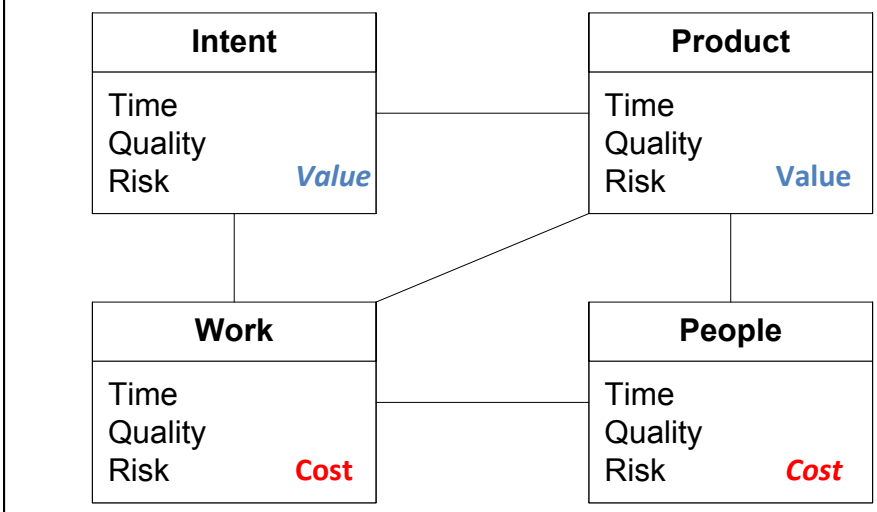
Tensions / Factors to Consider

- Engineers don't like technical debt
  they want to be technically flawless
- Project managers or business people don't mind technical debt
  they want to capture market share
- However, tolerance for TD changes over the system lifetime of the system

Lim et al. 2012

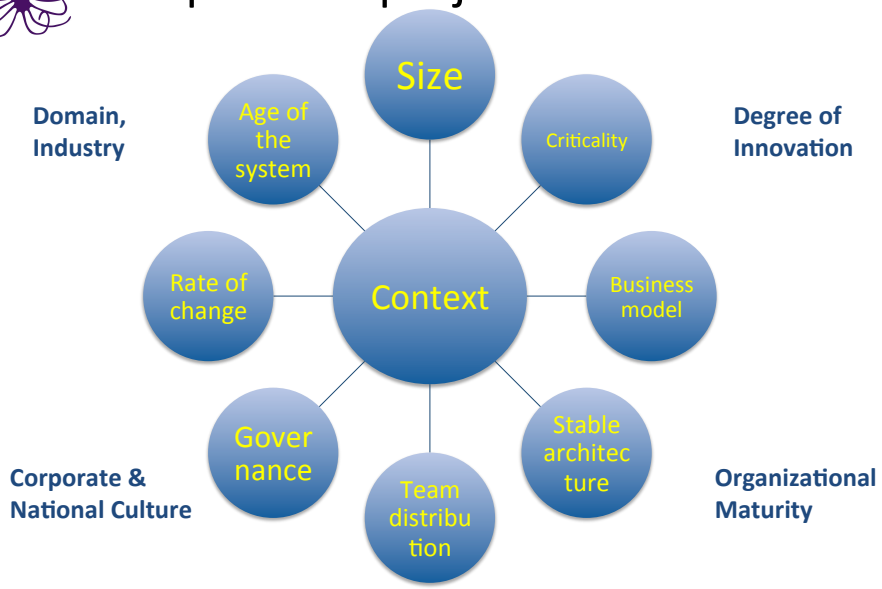
 26

Frog: "All projects are the same"



 27

Octopus: "All projects are different!"



 28

A **project** is all the **work** that **people** have to accomplish over **time** to realize in a **product** some specific **intent**, at some level of **quality**, delivering **value** to the business at a given **cost**, while resolving many **uncertainties and risk.**

All aspects of software projects are affected by **context**: size, criticality, team distribution, pre-existence of an architecture, governance, business model, which will guide with practices will actually perform best, within a certain domain and culture.

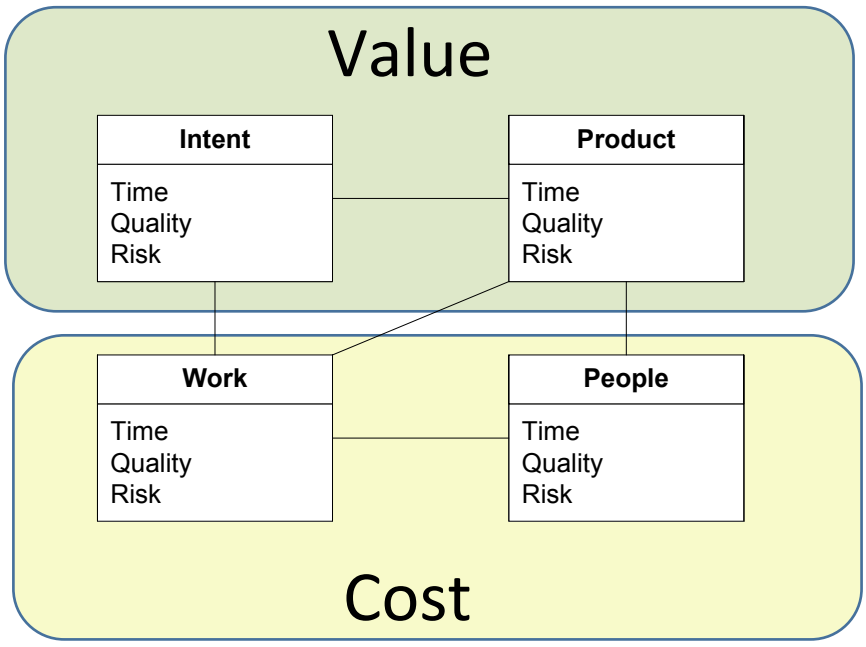
 29

Value and Cost

- **Value**: to the business (the users, the customers, the public, etc.)
- **Cost**: to design, develop, manufacture, deploy, maintain

- Simple system, stable architecture, many small features:
  - Roughly, value aligns to cost
- Large, complex, novel systems ?
  - Not quite so

 30



 31

What's in your backlog?

	Visible	Invisible
Positive Value	**New features Added functionality**	**Architectural, Structural features**
Negative Value	**Defects**	**Technical Debt**

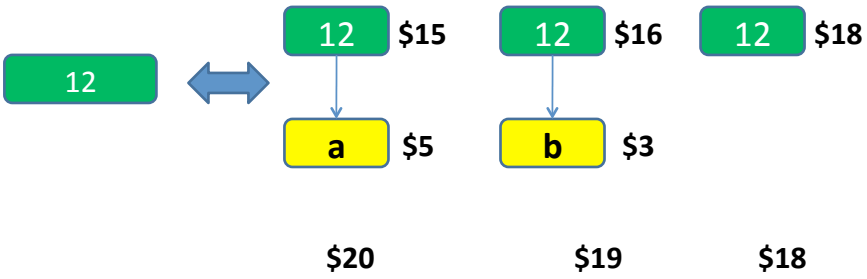
 32

TD: negative value, invisible

	Visible	Invisible
Positive Value	**New features Added functionality**	**Architectural, Structural features**
Negative Value	**Defects**	**Technical Debt**

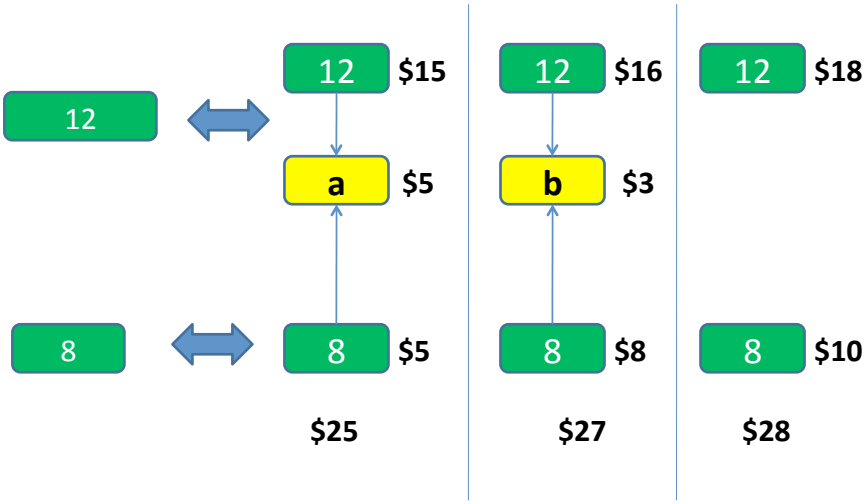
 33

Technical Debt (1)



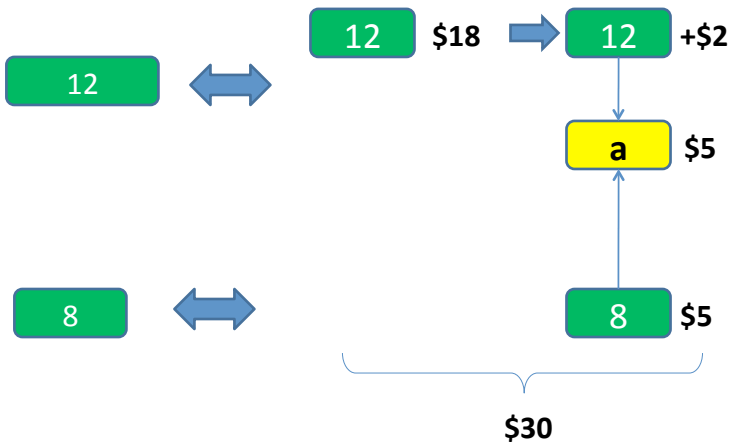
 34

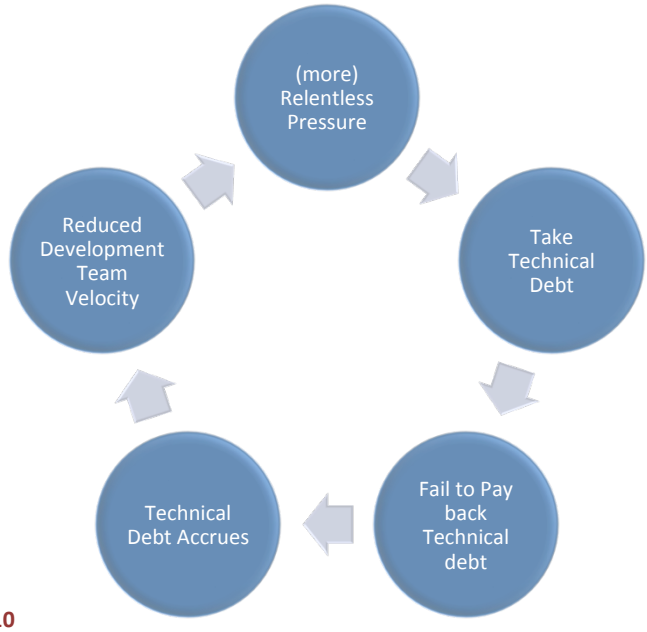
Technical Debt (2)



 35

Technical Debt (3)





Israel Gat, 2010
http://theagileexecutive.com/2010/09/20/how-to-break-the-vicious-cycle-of-technical-debt/

Technical Debt

- Defect = Visible feature with negative value
- Technical debt = Invisible feature with negative value
  - Cost …. of fixing
  - Value …. of repaying technical debt, interests loss of productivity, etc.

 38

Interests

- In presence of technical debt,
  cost of adding new features is higher;
  velocity is lower.
- When repaying (fixing), additional cost for retrofitting already implemented features
- Technical debt not repaid => lead to increased cost, forever
- Cost of fixing (repaying) increases over time

M. Fowler, 2009

 39

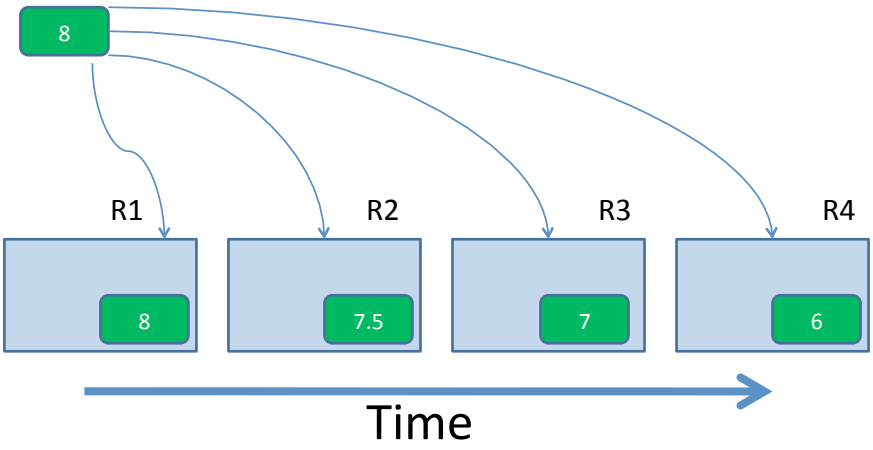
TD litmus test

- If you are not incurring any interest, then it probably is not a debt

McConnell 2013

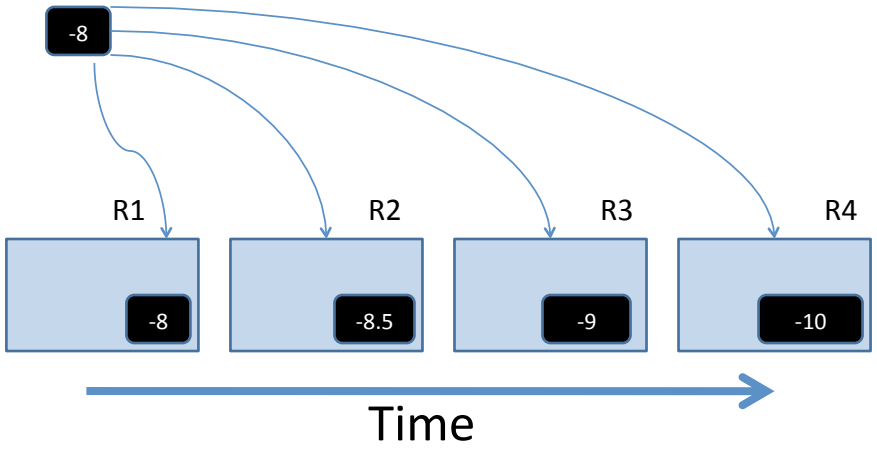
 40

Deferring implementation: Value decreases



 41

But technical debt increases over time



 42

Outline

- What is technical debt?
- The technical debt landscape
- Causes of technical debt
  - Cost vs. value
- Limits of the metaphor
- Tackling Technical debt
- Friction in software development

 43

Tech Debt (mis)-conceptions

- Technical debt reifies an abstract concept
- Technical debt does not equate to bad quality
- Technical debt can be induces by a shift in context
- Defects are not technical debt
- Lack of progress is not technical debt
- New features yet to be implemented is not technical debt

 44

It's only a Metaphor!

- Metaphors give meaning to form, help ground our conceptual systems.
- Cognitive transfer: source domain to target domain
  - the <target> is the <source>

Lakoff and Johnson (1980) *Metaphors we live by*

- *Do not push any metaphor too far….*

 45

Where the metaphor breaks

- Technical debt does not always have to be repaid
- What does it mean to be “debt free”?
  - TD has a large part of subjectivity
- Negative connotation
- May increase the value of a project for a time
- Tech Debt as Investment?

Where the metaphor breaks

- Initial investment at T0 in an environment E0. Now in T2, E has changed to E2, a mismatch has occurred, which creates a debt.
  - The debt is created by the change of environment. The right decision in the right environment at some time may lead to technical debt.
- Prudent, inadvertent

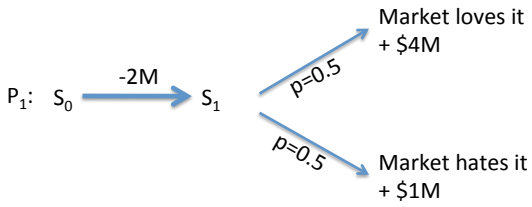
Where the metaphor breaks…

- Technical debt depends on the future
- Technical debt cannot be measured
- You can walk away from technical debt
- Technical debt should not be completely eliminated
- Technical debt cannot be handled in isolation
- Technical debt can be a wise investment

Real Options Theory

- Often mentioned, but rarely put in application in software

TD and Real Options

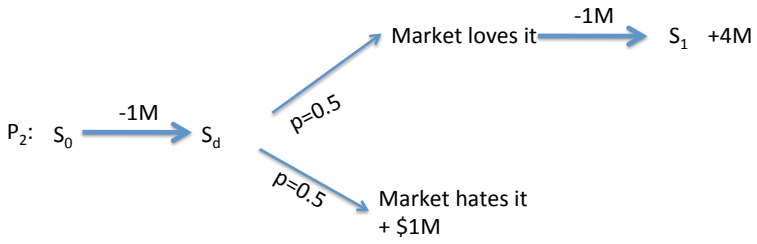


NPV (P_1) = -2M + 0.5x4M + 0.5x1M = 0.5M

Source: K. Sullivan, 2010
at TD Workshop SEI 6/2-3

 50

TD and Real Options (2)



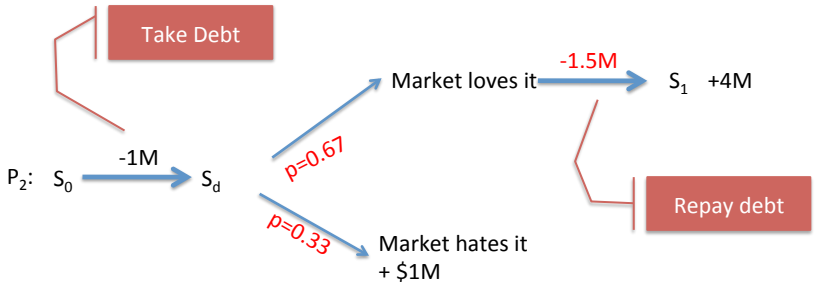
NPV (P_2) = -1M + 0.5x3M + 0.5x1M = 1M

Taking Technical Debt has increased system value.

Source: K. Sullivan, 2010

 51

TD and Real Options (3)

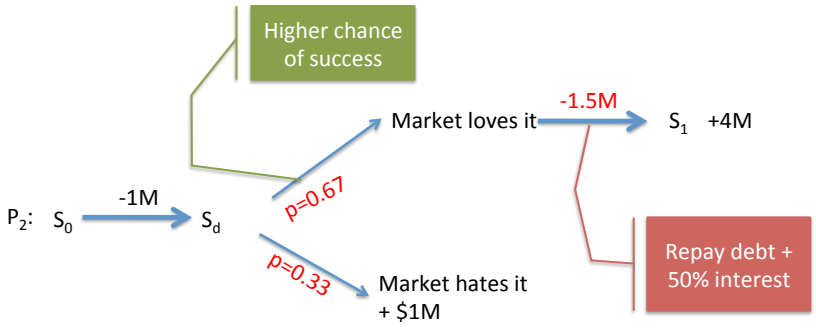


NPV (P_3) = -1M + 0.67 x 2.5M + 0.33 x 1M = 1M

More realistically:
Debt + interest
High chances of success

 52

TD and Real Options (3)

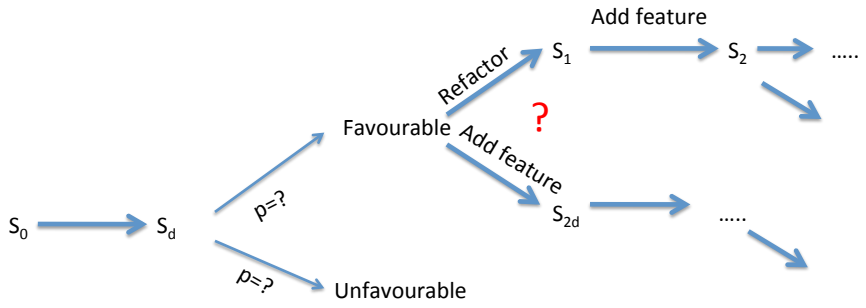


NPV (P_3) = -1M + 0.67 x 2.5M + 0.33 x 1M = 1M

More realistically:
Debt + interest
High chances of success

 53

TD and Real Options (4)



Not debt really, but options with different values…
Do we want to invest in architecture, in test, etc…

Source: K. Sullivan, 2010

 54

Options Theory

- Often mentioned, but rarely put in application in software
- Not even scratched the surface
- Pay-off not obvious, though…
  - Too much guesswork involved to trust results,
  - Lot of work involved

 55

Potential vs. actual debt

- Potential debt
  - Type 1:OK to do with tools (see Gat & co. approach)
  - Type 2: structural, architectural, or technological gap: Much harder
- Actual debt
  - When you know the way forward

K.Schmid 2013

 56

Outline

- What is technical debt?
- The technical debt landscape
- Causes of technical debt
  - Cost vs. value
- Limits of the metaphor
- Tackling Technical debt
- Friction in software development

 57

How do people “tackle” technical debt

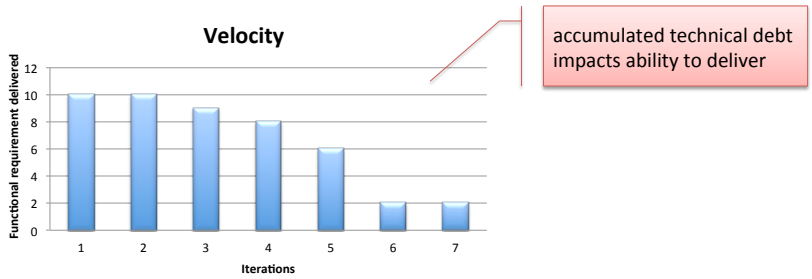
Tackling Technical Debt

Attitudes and approaches found:

1. Ignorance is bliss
2. The elephant in the room
3. Big scary $$$$ numbers
4. Five star ranking
5. Constant reduction
6. We’re agile, so we are immune!

Ignorance is bliss

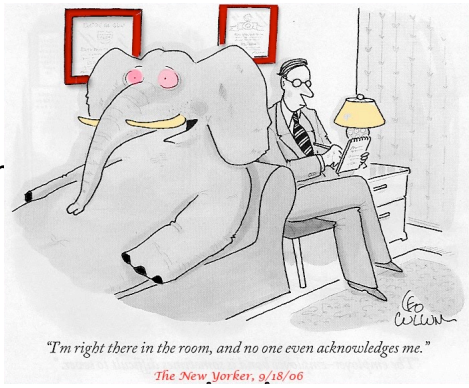
You're just slower, and slower, but you do not know it, or do not know why



 60

The elephant in the room

- Many in the org. know about technical tech.
- Indifference: it's someone else's problem
- Organization broken down in small silos
- No real whole product mentality
- Short-term focus



 61

Big scary $$$$ numbers

- Code smells 167 person days
- Missing test 298 person days
- Design 670 person days
- Documentation 67 person days

Totals

Work 1,202 person x days

Cost $577,000

 62

Static analysis + Consulting

- Cutter Consortium: Gat, et al.
  - Use of Sonar, etc.
  - Focused on code analysis
  - TD = total value of fixing the code base
- CAST software
- ThoughtWorks

Debt analysis engagements
Debt reduction engagements

 63

Issues

- Fits the metaphor, indeed.
- Looks very objective… but…
- Subjective in:
  - What is counted
  - What tool to use
  - Cost to fix

Not all fixes have the same resulting value.
Sunk cost are irrelevant, look into the future only.
What does it mean to be “Debt free”??

 64

Five star ranking

- Define some *maintainability* index
- Benchmark relative to other software in the same category
- Re-assess regularly (e.g., weekly)
- Look at trends, correlate changes with recent changes in code base

- SIG (Software Improvement Group), Amsterdam
- Powerful tool behind

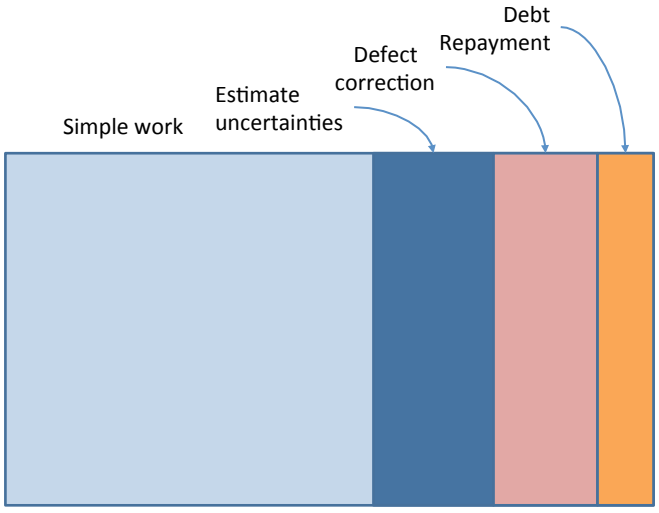
 65

Constant debt reduction

- Make technical debt a visible item on the backlog
- Make it visible outside of the software dev. organization
- Incorporate debt reduction as a regular activity
- Use buffer in longer term planning for yet unidentified technical debt
- Lie (?)

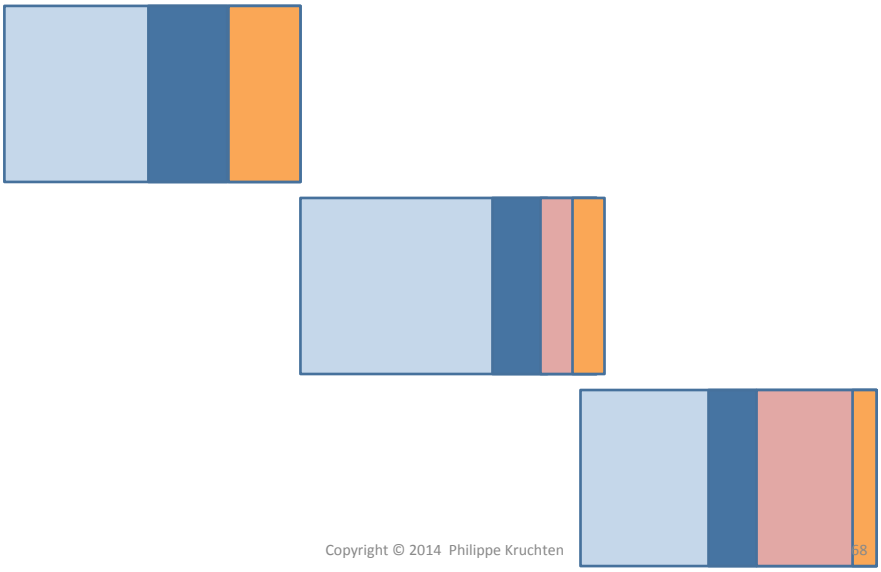
 66

Buffer for debt repayment



 67

A later release



 68

We are agile, so we're immune!

In some cases we are agile and therefore we run faster into technical debt

 69

Agile mottos

- "Defer decision to the last responsible moment"
- "YAGNI" = You Ain't Gonna Need It
  - But when you do, it is technical debt
  - Technical debt often is the accumulation of too many YAGNI decisions
- "We'll refactor this later"
- "Deliver value, early"
- *Again the tension between the yellow stuff and the green stuff*
- *You're still agile because you aren't slowed down by TD yet.*

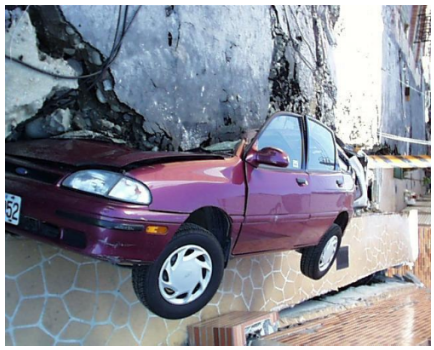
 70

Story of a failure

- Large re-engineering of a complex distributed world-wide system; 2 millions LOC in C, C++, Cobol and VB
- Multiple sites, dozens of data repositories, hundreds of users, 24 hours operation, mission-critical ($billions)
- xP+Scrum, 1-week iterations, 30 then up to 50 developers
- Rapid progress, early success, features are demo-able
- Direct access to "customer", etc.
- *A poster project for scalable agile development*

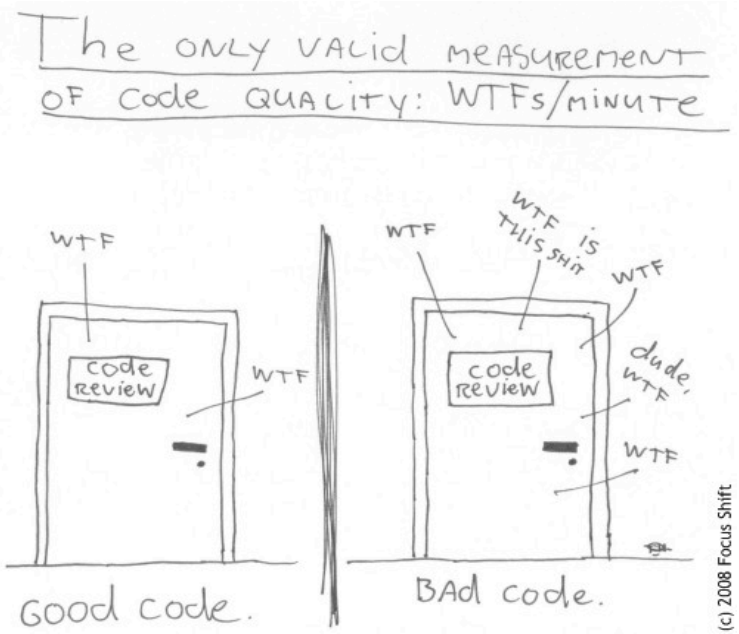
 71

Hitting the wall



- After 4 ½ months, difficulties to keep with the 1-week iterations
- Refactoring takes longer than one iteration
- Scrap and rework ratio increases dramatically
- No externally visible progress anymore
- Iterations stretched to 3 weeks
- Staff turn-over increases
- Project comes to a halt
- Lots of code, no clear architecture, no obvious way forward

 72

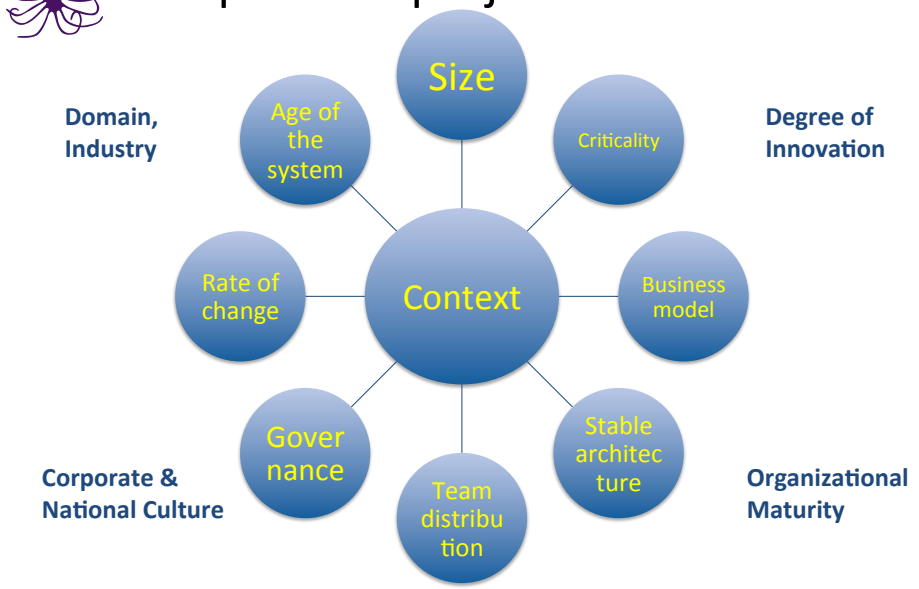


Managing TD…

- Identify sources of TD
- Locate TD
  - Not easy for McConnell type 2
- Quantify TD
  - Principal, Interest
- Define actions
  - Priorities
  - Tooling
- Assessment

 74

Octopus: "All projects are different!"



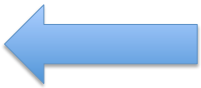
 75

Debt at the Architectural level

- Design Structure Matrix (DSM)
  - a.k.a, Dependency Structure Matrix
- Domain Mapping Matrix (DMM)
- Tools to create and manipulate DSMs and DMMs

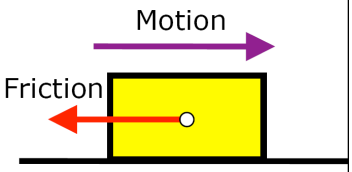
Outline

- What is technical debt?
- The technical debt landscape
- Causes of technical debt
  - Cost vs. value
- Limits of the metaphor
- Tackling Technical debt
- Friction in software development



Friction

"Friction: the resistance that one surface or object encounters when moving over another."

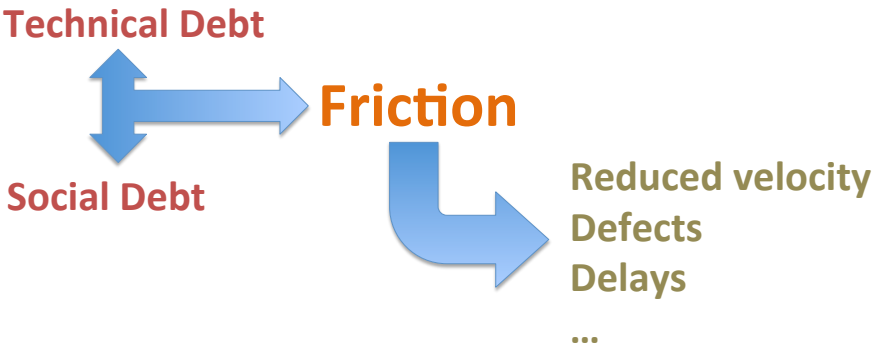


In software development, friction is the *set of phenomena that limits or constraints our progress*, therefore reduces our velocity (or productivity).

Technical debt causes friction.

 78

Friction and Debt



 79

Social debt

- Social debt is a state of a development project which is the result of the accumulation over time of decisions about the way the development team (or community) communicates, collaborates and coordinates.

Tamburri et al. 2013

 80

Social debt

- In other words, decisions about :
  - the organizational structure,
  - the process,
  - the governance,
  - the social interactions,
- or some elements inherited through the people:
  - their knowledge, personality, working style, etc.

Tamburri et al. 2013

 81

Parallel Technical & Social Debt

	Visible	Invisible
Positive Value	**New features Added functionality**	**Architectural, Structural features**
Negative Value	**Defects**	**Technical Debt**

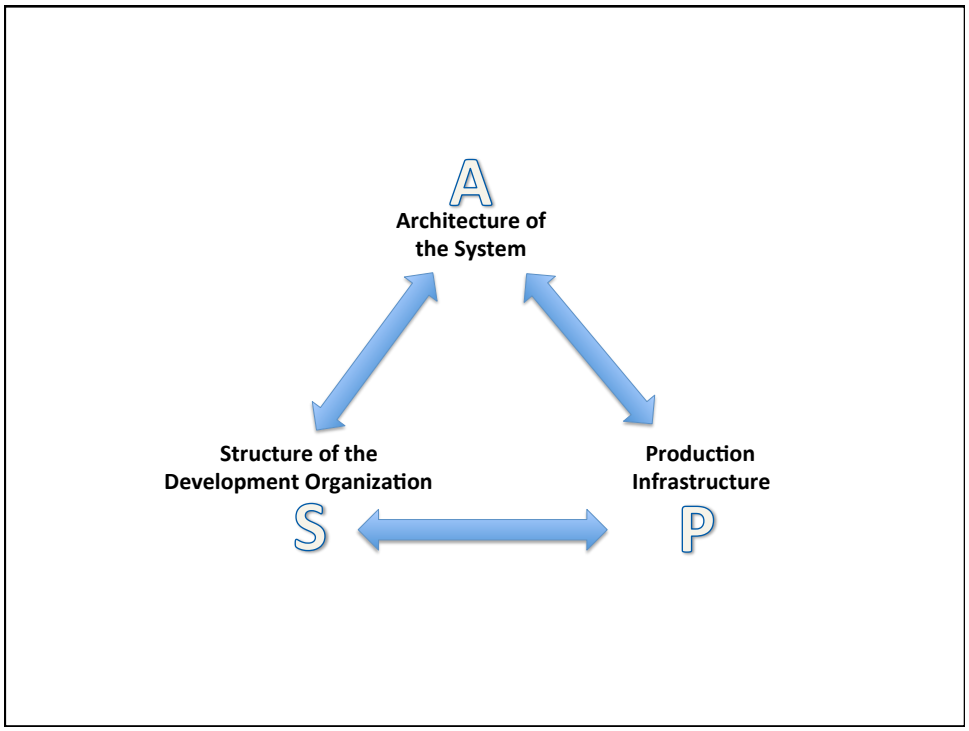
 82

Social debt

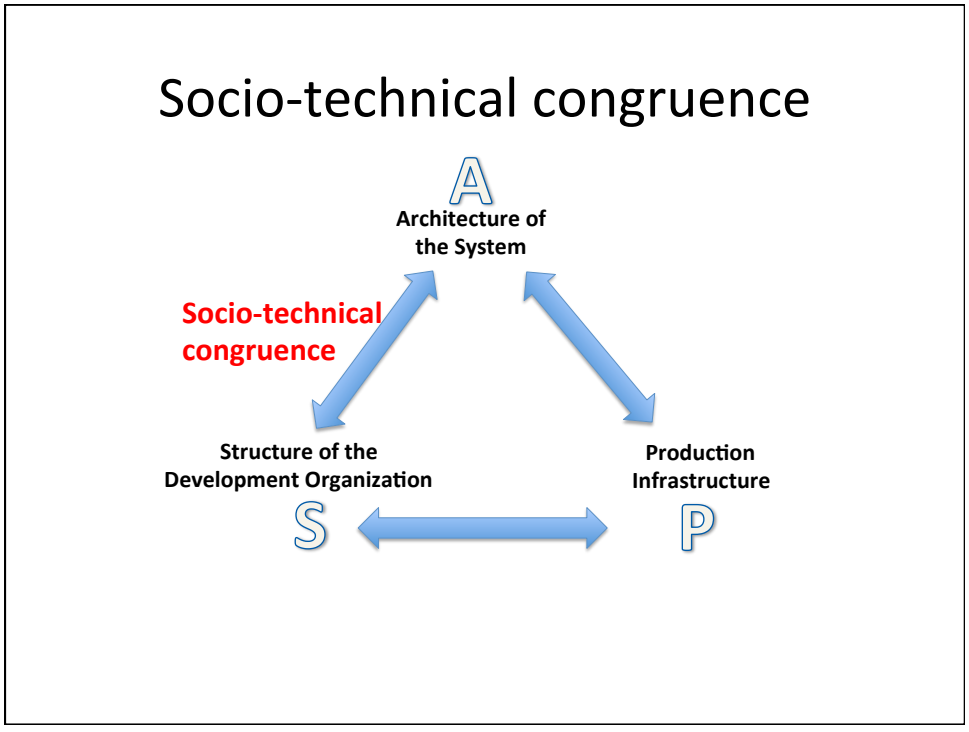
	Visible	Invisible
Positive Value	**Community Features**	**Community Structure**
Negative Value	**Community Defects**	**Social Debt**

Tamburri et al. 2013

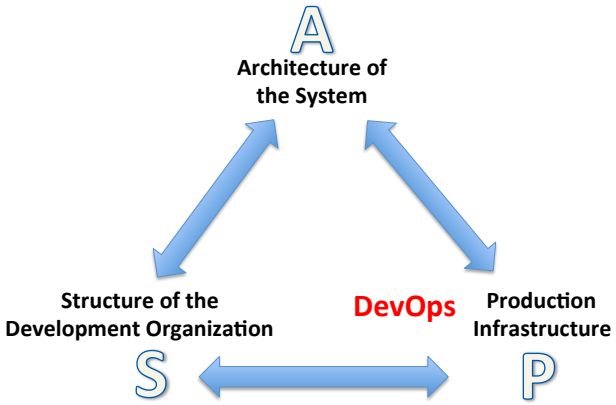
 83



Socio-technical congruence



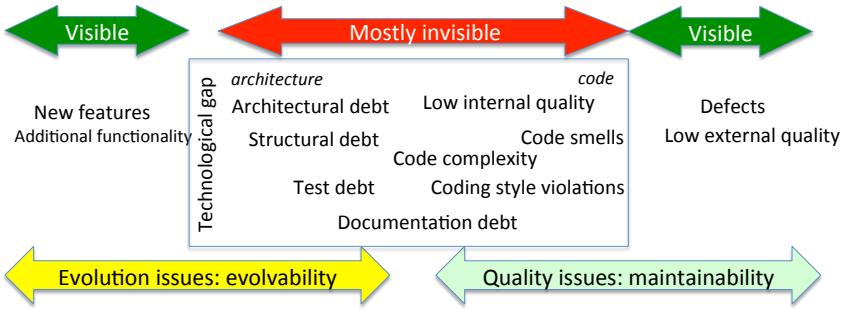
DevOps: Development+Operations



Conclusion

- Technical debt is still more a *rhetorical* category than a *technical* or ontological category.
- The concept resonates well with the development community, and sometimes also with management.
- It bridges the gap between business decision makers and technical implementers.
- It's only a metaphor; do not push it too far.
- It's not all bad.

Technical debt landscape



Kruchten et al 2012

 88

References

- Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., et al. (2010). *Managing Technical Debt in Software-Intensive Systems*. Paper presented at the Future of software engineering research (FoSER) workshop, part of Foundations of Software Engineering (FSE 2010) conference.
- Brown, N., Nord, R., Ozkaya, I., Kruchten, P., & Lim, E. (2011). Hard Choice: A game for balancing strategy for agility. Paper presented at the 24th IEEE CS Conference on Software Engineering Education and Training (CSEE&T 2011), Honolulu, HI, USA.
- Cunningham, W. (1992). *The WyCash Portfolio Management System.* Paper presented at the OOPSLA'92 conference, ACM. Retrieved from http://c2.com/doc/oopsla92.html
- Curtis, B., Sappidi, J., & Szynkarski, A. (2012). Estimating the Principal of an Application's Technical Debt. IEEE Software, 29(6).
- Denne, M., & Cleland-Huang, J. (2004). *Software by Numbers: Low-Risk, High-Return Development,* Prentice Hall.
- Denne, M., & Cleland-Huang, J. (2004). The Incremental Funding Method: Data-Driven Software Development, *IEEE Software*, 21(3), 39-47.
- Fowler, M. (2009), *Technical debt quadrant,* Blog post at: http://www.martinfowler.com/bliki/TechnicalDebtQuadrant.html
- Gat, I. (ed.). (2010). *How to settle your technical debt--a manager's guide*. Arlington Mass: Cutter Consortium.
- Kruchten, Ph. (2010) Contextualizing Agile Software Development," Paper presented at the EuroSPI 2010 conference in Grenoble, Sept.1-3, 2010

 89

References

- Kruchten, P., Nord, R., & Ozkaya, I. (2012). Technical debt: from metaphor to theory and practice. *IEEE Software*, 29(6).
- Kruchten, P., Nord, R., Ozkaya, I., & Visser, J. (2012). Technical Debt in Software Development: from Metaphor to Theory--Report on the Third International Workshop on Managing Technical Debt, held at ICSE 2012 *ACM SIGSOFT Software Engineering Notes*, 37(5).
- Li, Z., Madhavji, N., Murtaza, S., Gittens, M., Miranskyy, A., Godwin, D., & Cialini, E. (2011). Characteristics of multiple-component defects and architectural hotspots: a large system case study. *Empirical Software Engineering*, 16(5), 667-702. doi: 10.1007/s10664-011-9155-y
- Lim, E. (2012). *Technical Debt: What Software Practitioners Have to Say.* (Master's thesis), University of British Columbia, Vancouver, Canada.
- Lim, E., Taksande, N., & Seaman, C. B. (2012). A Balancing Act: What Software Practitioners Have to Say about Technical Debt. *IEEE Software*, 29(6).
- MacCormack, A., Rusnak, J., & Baldwin, C. Y. (2006). Exploring the structure of complex software designs: An empirical study of open source and proprietary code. *Management Science, 52*(7), 1015-1030.
- Nord, R., Ozkaya, I., Kruchten, P., & Gonzalez, M. (2012). In search of a metric for managing architectural technical debt. Paper presented at the *Working IEEE/IFIP Conference on Software Architecture (WICSA 2012*), Helsinki, Finland.
- McConnell, S. (2007) *Notes on Technical Debt*, Blog post at: http://blogs.construx.com/blogs/stevemcc/archive/2007/11/01/technical-debt-2.aspx
- Special issue of *Cutter IT Journal* on Technical Debt, edited by I. Gat (October 2010) Cutter IT Journal, 23 (10).
- Sterling, C. (2010) *Managing Software Debt*, Addison-Wesley.

References (cont.)

- R. O. Spinola, N. Zazworka, A. Vetrò, C. B. Seaman, and F. Shull, "Investigating Technical Debt Folklore: Shedding Some Light on Technical Debt Opinion," in Proceedings of the 4th Workshop on Managing Technical Debt, at ICSE 2013, P. Kruchten, I. Ozkaya, and R. Nord, Eds., IEEE, 2013.
- K. Schmid, "On the Limits of the Technical Debt Metaphor," in Proceedings of the 4th Workshop on Managing Technical Debt, at ICSE 2013, P. Kruchten, I. Ozkaya, and R. Nord, Eds., IEEE, 2013, pp. 63-66.
- K. Schmid, "A Formal Approach to Technical Debt Decision Making," in Proceedings of the Conference on Quality of Software Architecture QoSA'2013, Vancouver, 2013, ACM.

Other sources (Talks/slides)

- Gat, I., Heintz, J. (Aug. 19, 2010) *Webinar: Reining in Technical Debt*, Cutter Consortium.
- McConnell, S. (October 2011) Managing technical debt. (Webinar)
- Kniberg, H. (2008) *Technical debt-How not to ignore it*, at Agile 2008 conference.
- Kruchten, P. (2009) *What colour is your backlog?* Agile Vancouver Conference. http://philippe.kruchten.com/talks
- Sterling, C. (2009) http://www.slideshare.net/csterwa/managing-software-debt-pnsqc-2009
- Short, G. (2009) http://www.slideshare.net/garyshort/technical-debt-2985889
- West, D. (January 2011), *Balancing agility and technical debt*, Forrester & Cast Software

 92

Other pointers

techdebt report http://techdebt.org

ontechnicaldebt http://www.ontechnicaldebt.com/

@OnTechnicalDebt

 93

Acknowledgements

- My research partly funded by the
  Software Engineering Institute | Carnegie Mellon
  - Ipek Ozkaya, Rod Nord
  - They have also contributed to building this tutorial over the last 2 years.
- UBC master student Erin Lim Kam-Yan…
  - And some industry partners in Canada

94

Slides?

http://philippe.kruchten.com/talks/

 95

 96

SG CONFERENCE

 97