Technical Debt

from metaphor to theory and practice

Philippe Kruchten
Helsinki, August 21st 2012

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

 2

Outline

- What is technical debt? Several viewpoints.
- The technical debt landscape
- Structural or architectural debt
- Research on technical debt
- "Managing" technical debt
- Summary, useful pointers

 3

Acknowledgements

- Research on TD partly funded by the Software Engineering Institute | Carnegie Mellon
  - Ipek Ozkaya, Rod Nord, Nanette Brown
  - They have also contributed to building this presentation over the last 2 years.
- UBC master students Erin Lim Kam-Yan and Marco Gonzalez-Rojas …
  - … with some industry partners

 4

Technical Debt

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

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

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

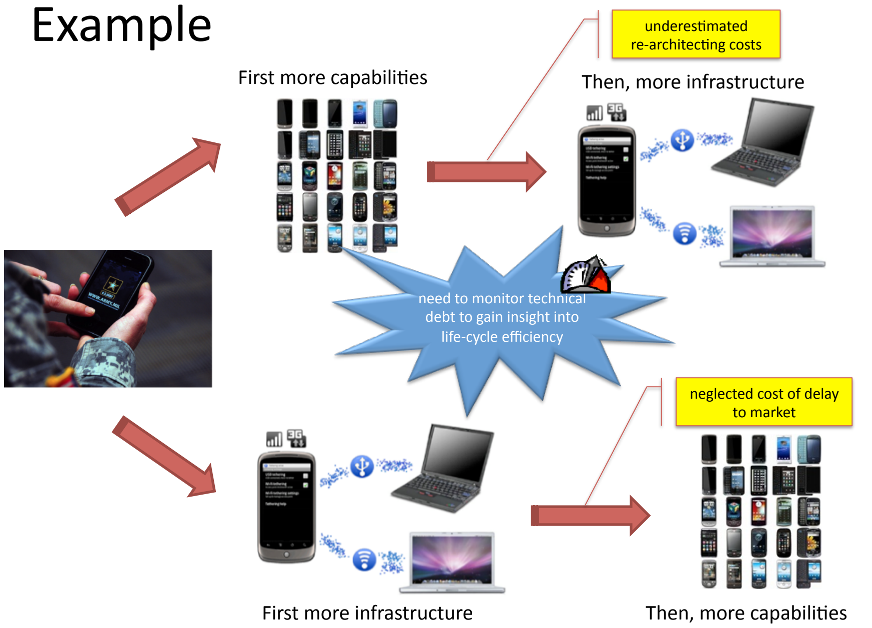
McConnell 2007

Technical Debt (M. Fowler)

	Reckless	Prudent
Deliberate	*"We don't have time for design"*	*"We must ship now and deal with consequences"*
Inadvertent	*"What's Layering?"*	*"Now we know how we should have done it"*

Fowler 2009, 2010

Example



Ozkaya, SEI,2010

 9

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.

 10

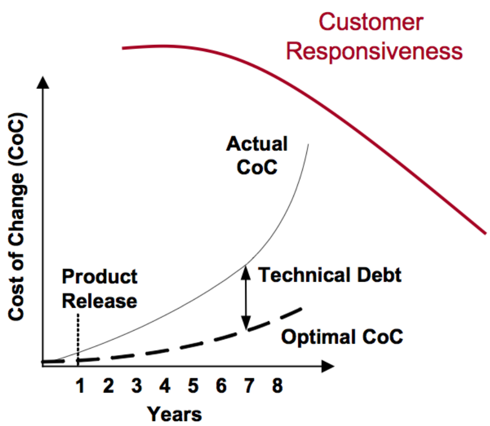
Technical Debt (S. McConnell)

- TD: 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

McConnell 2011

 11

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 12

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

 13

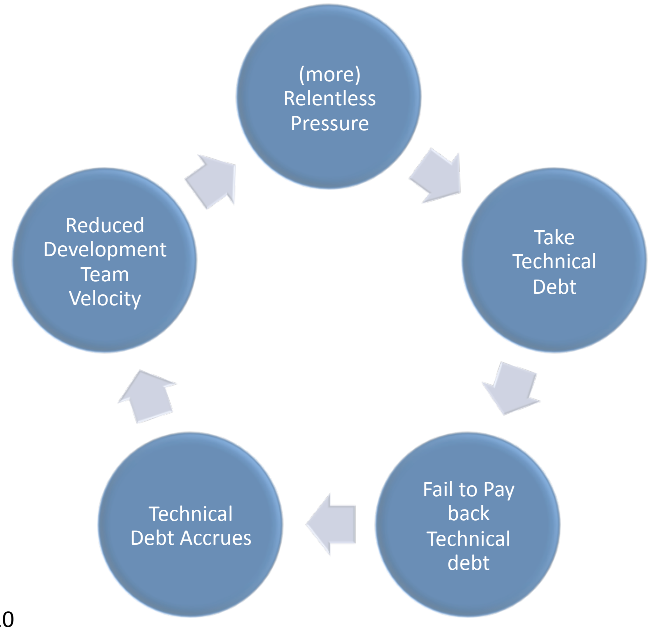
Example: TD is the sum of…

• 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

 14



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

 15

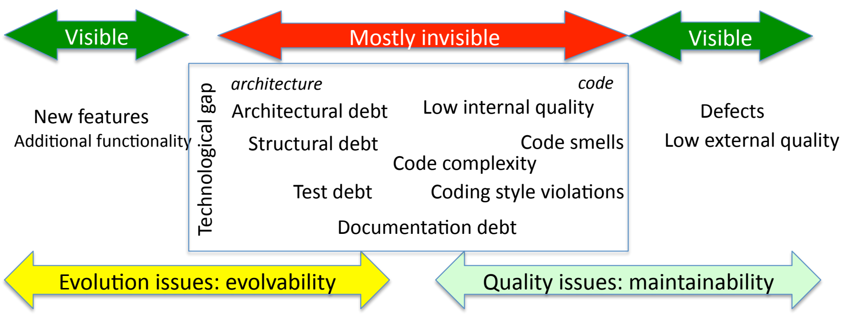
Causes of Technical Debt

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

 16

Technical debt landscape



Value of Software Architecture

A little détour

Value and cost

- Architecture has no (or little) externally visible "customer value"
- Iteration planning (backlog) is driven by "customer value"
- *Ergo:* architectural activities are often not given attention
- BUFD & YAGNI & Refactor!

 22

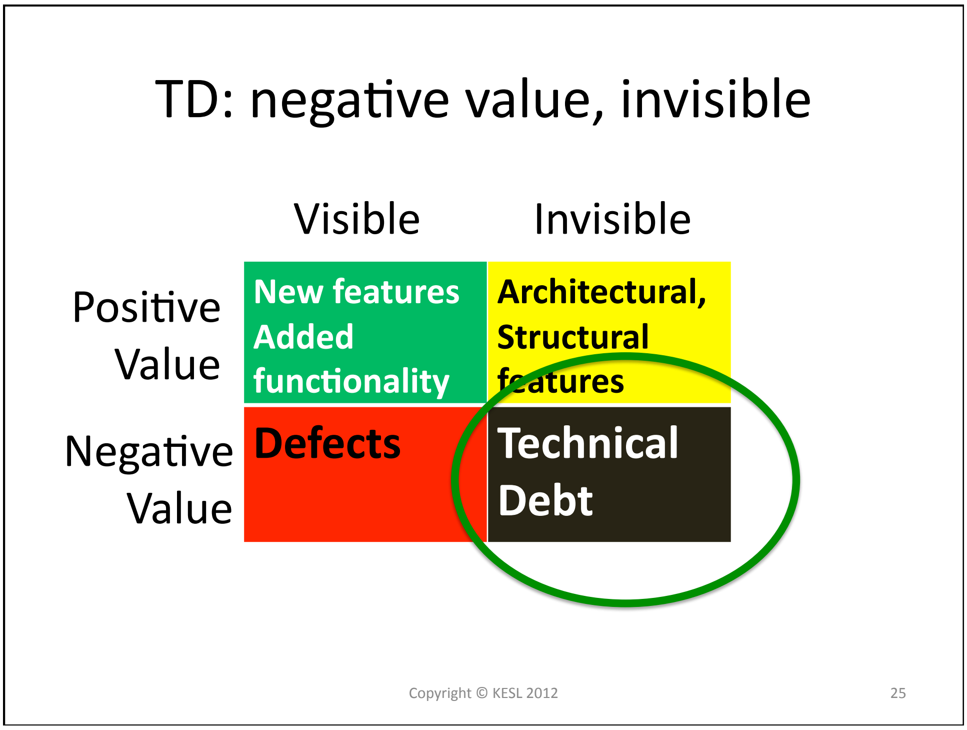
Value and cost

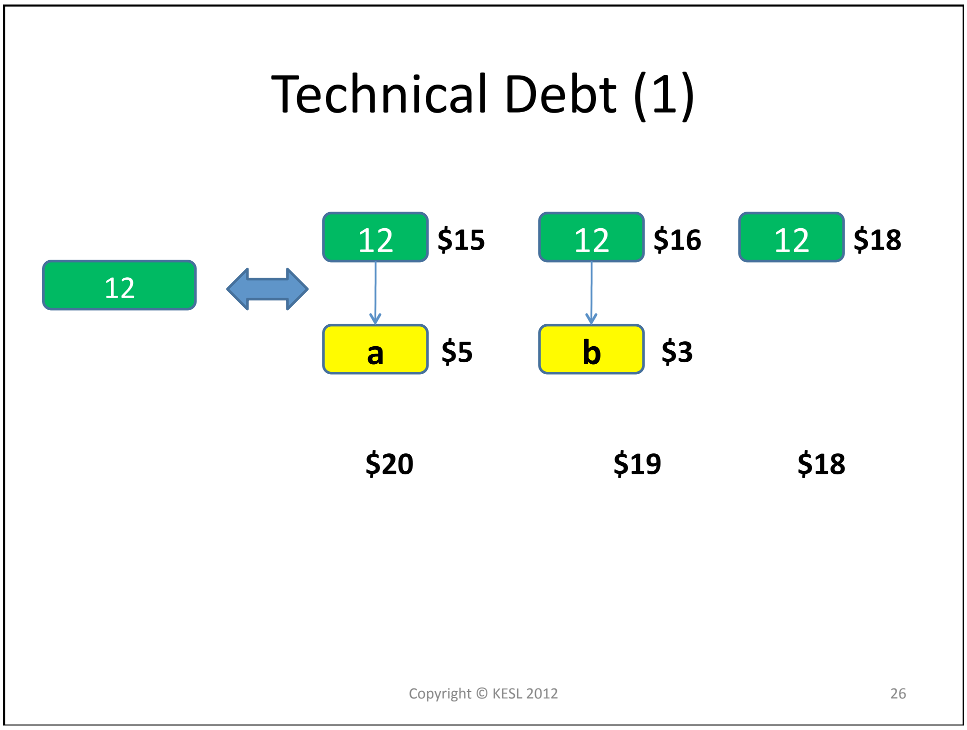
- Cost of development is not identical to value
- Trying to assess value and cost in monetary terms is hard and often leads to vain arguments

- Use “points” for cost and “utils” for value
- Use simple technique(s) to evaluation cost in points and value in utils.

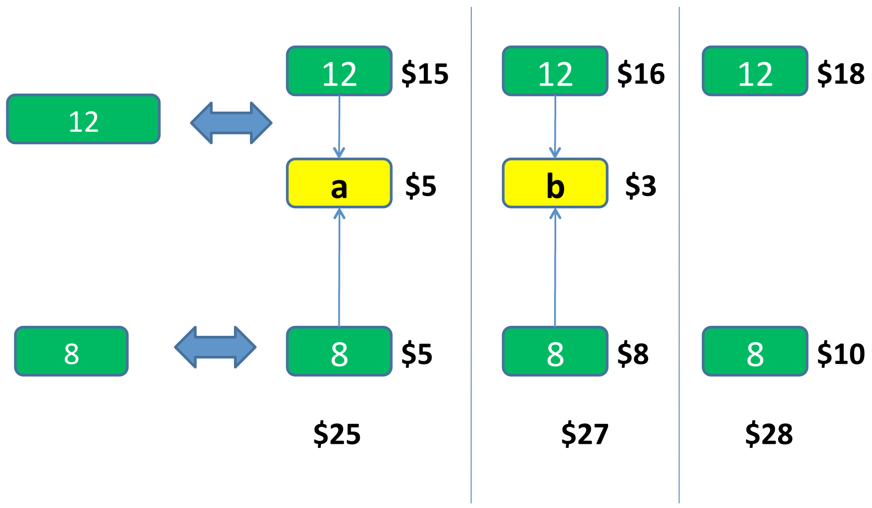
What’s in your backlog?

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


TD: negative value, invisible
Visible
Invisible
Positive Value
Negative Value
New features Added functionality
Architectural, Structural features
Defects
Technical Debt
Copyright © KESL 2012
25

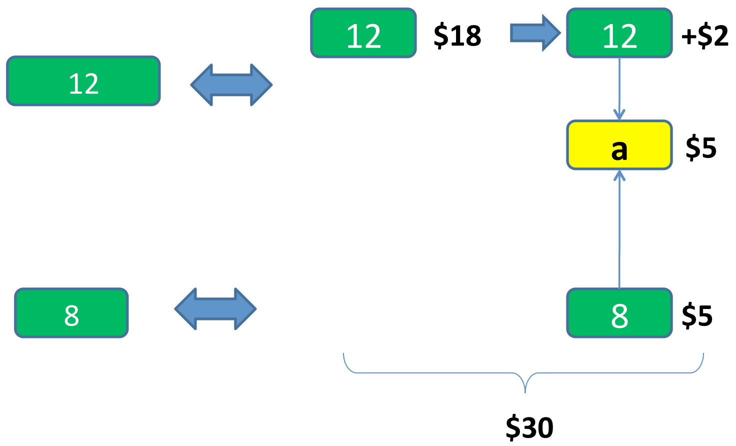

Technical Debt (1)
12
12
$15
a
$5
12
$16
b
$3
12
$18
$20
$19
$18
Copyright © KESL 2012
26

Technical Debt (2)



 27

Technical Debt (3)



 28

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.

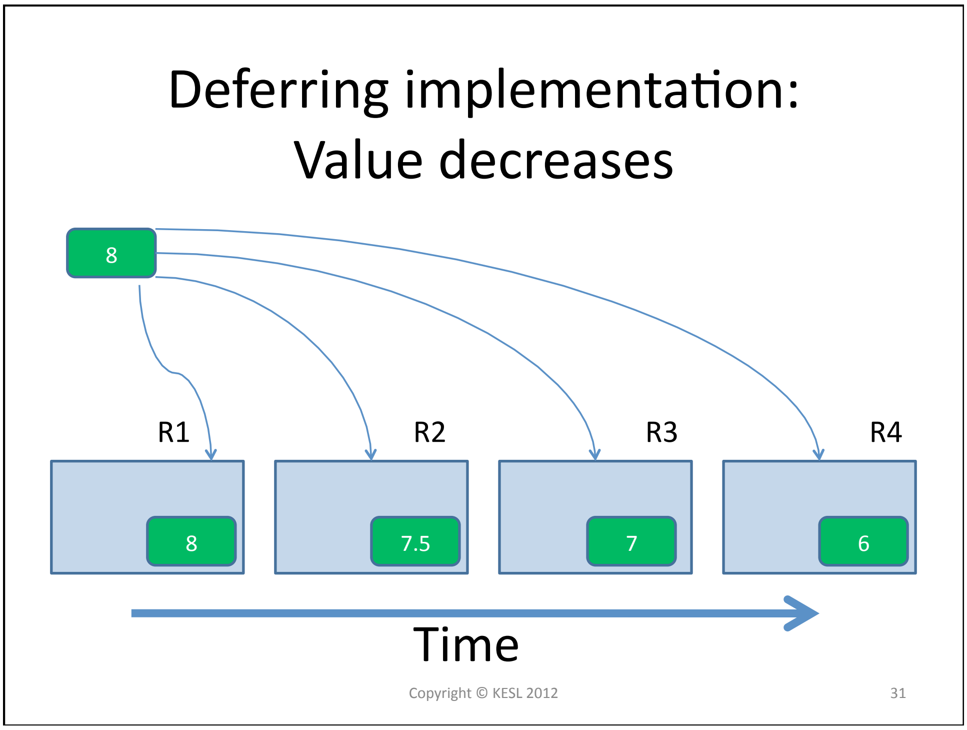
 29

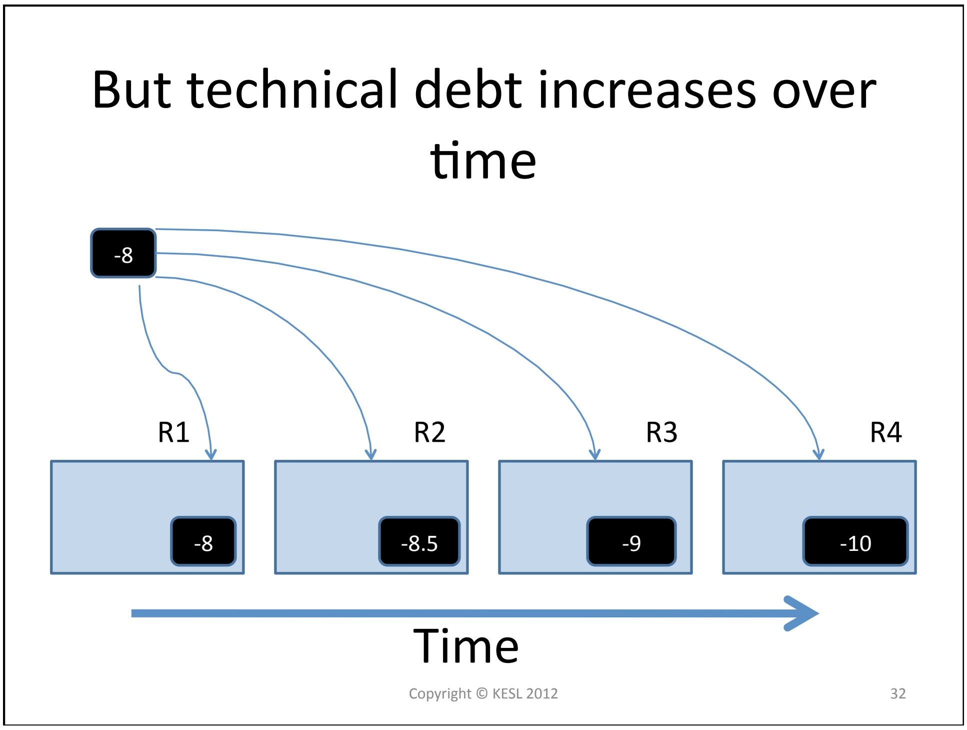
Interests (?)

- 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

 30


Deferring implementation:
Value decreases
8
R1
R2
R3
R4
8
7.5
7
6
Time
Copyright © KESL 2012
31


But technical debt increases over
time
-8
R1
R2
R3
R4
-8
-8.5
-9
-10
Time
Copyright © KESL 2012
32

Advances

Areas of further investigation

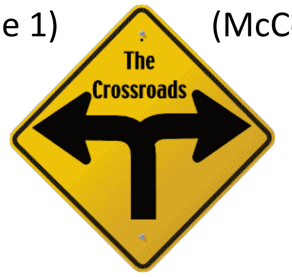
Research on TD

- Characterize objectively and quantitatively the amount of technical debt in a given system
- Taxonomy of technical debt => Better detection
- Causes of technical debt => Improved prevention
- Project management strategies to control and to cope with technical debt

- Tools and methods to deal with code smells, etc.
- Application of Real Options, Dependency Structure Matrix, or other value-based technique

 35

- Code level debt (McConnell type 1)
- Structural debt (McConnell type 2)



 36

Code level

- A.K.A., Code smells
- Much research, though fragmented
- Many tools to do static code analysis
- Example: Code replication (clones)
- Approach: detect + refactoring

 37

Tech Debt = Maintainability?

- Example:
  - SIG (Software Improvement Group), Amsterdam

 38

Debt at the Architectural Level

- Harder to detect with tools
- Less researched

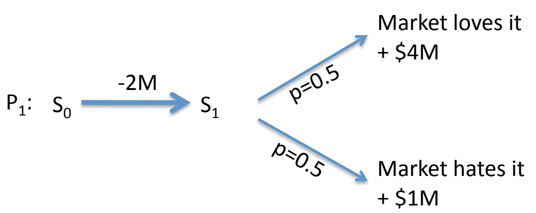
- A few paths to explore:
  - Dependency structure matrices
  - Business Theories:
    - Real Option
    - Net present value

 39

Real Options Theory

- Often mentioned, but rarely put in application in software

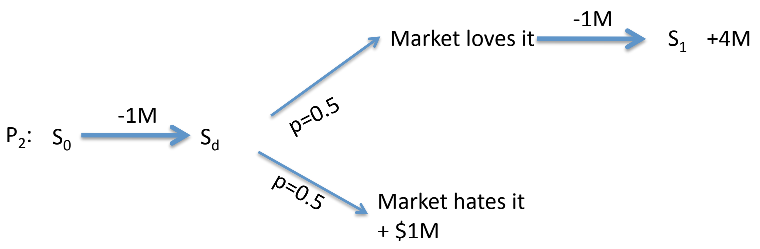
TD and Real Options



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

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

TD and Real Options (2)



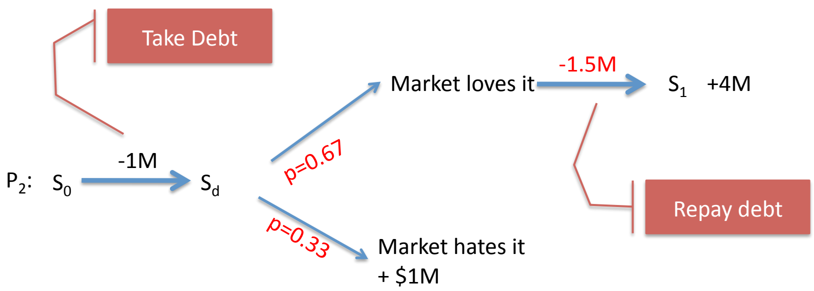
NPV (P_2) = -1M + 0.5x3M + 0.5x1M = 1M

Taking Technical Debt has increased system value.

Source: K. Sullivan, 2010

 42

TD and Real Options (3)

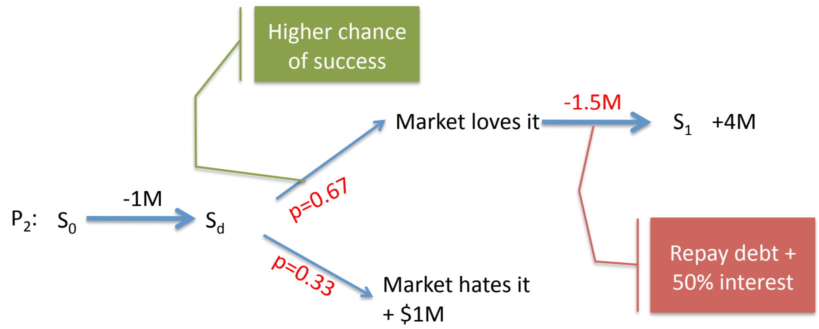


NPV (P_3) = -1M + 0.67 x 2.5M + 0.33 x 1M = 1M

More realistically:
Debt + interest
High chances of success

 43

TD and Real Options (3)

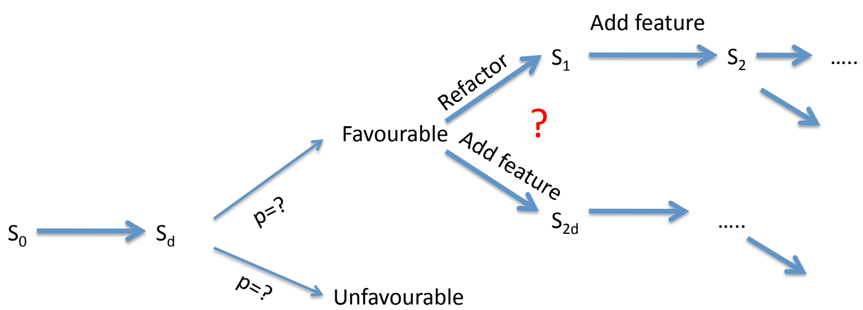


NPV (P_3) = -1M + 0.67 x 2.5M + 0.33 x 1M = 1M

More realistically:
Debt + interest
High chances of success

 44

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

 45

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

 46

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

 47

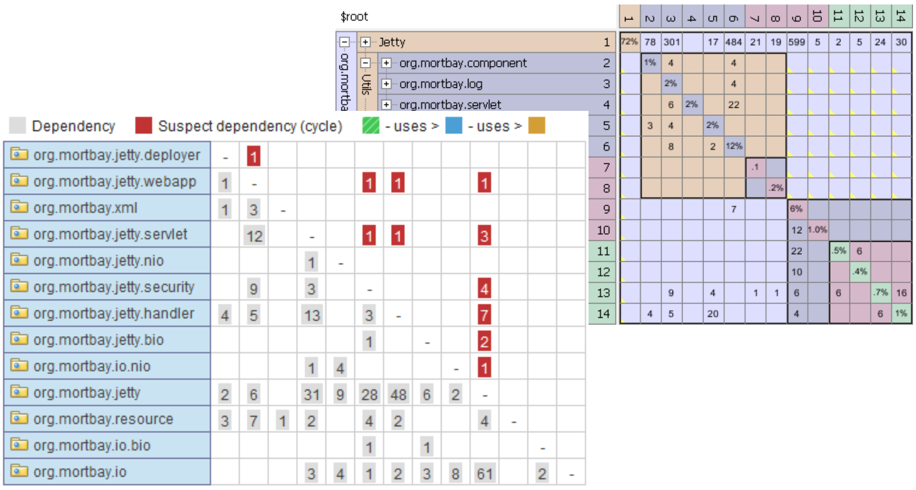
Dependency Structure Matrix

	A	B	C
A		Strength of B's dependency on A	
B	Strength of A's dependency on B		Strength of C's dependency on B
C			

Dependencies for MS-Lite

		Alarm Notificat 1	L&R Engine 2	Alarm Engine 3	Alarm Manage 4	Logon 5	Client Updates 6	Rule Processo 7	Adapter Mana 8	User Session 9	Data Access 10	Cache 11	FSS Adapter 12	Data Persisten 13	Publish-Subsc 14
US 01	1						X			X					X
US 02	2						X			X					X
US 03	3			X				X		X					
US 04	4	X		X	X			X		X					X
US 05	5				X						X				
US 06	6		X					X		X					X
US 07	7		X	X										X	
US 08	8					X					X				
US 09	9								X				X		
ACT 14	23								X						
ACT 15	24								X				X		X
ACT 16	25								X				X		X
ACT 17	26						X			X		X			X
ACT 18	27	X					X								
ACT 19	28	X													
ACT 20	29					X					X				
ACT 21	30									X					

Dependency Structure Matrix



 50

Propagation cost

- "Density" of the DSM
  - Proposed by McCormack et al. in 2006
  - Several limitations as a tool to measure T.D.
- Improved PC:
  - Boolean to continuous value (=dependency "strength")
  - Changes not uniformly spread throughout the code
  - Less sensitive to size of code

McCormack et al. 2006

 51

Exploring other variations

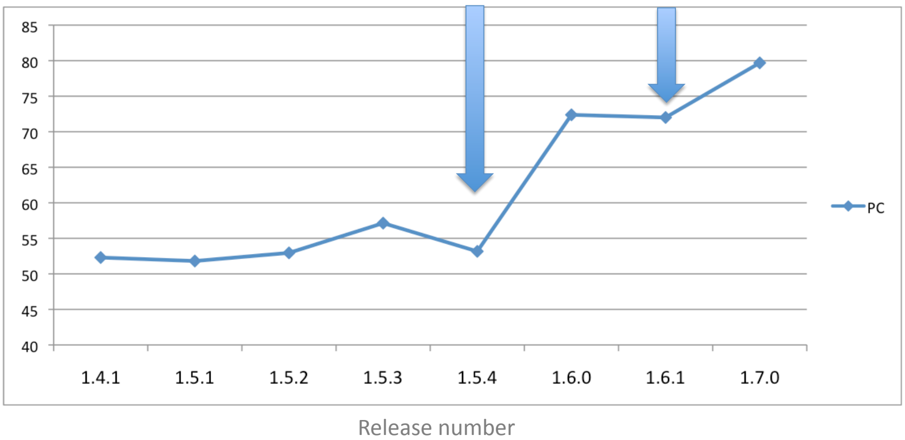
- Size of components
  - Add some weighting factor related to the size of the component A and B, where A depends on B
- Nothing very useful so far; need more experimentation and validation on large real systems

Nord et al. 2012

 52

Example of PC: Evolution of Ant

Technical debt reduction



 53

DSM

- Value of DSM not fully explored yet
  - Concept of propagation cost
  - Concept of density
  - Need to integrate values and costs
- Tools to produce or manipulate DSM
  - SonarJ
  - Lattix
- "What if" scenarios

 54

"Tackling" technical debt

Tackling Technical Debt

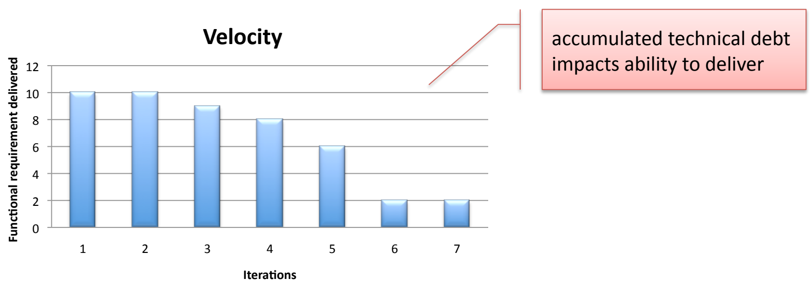
Attitude, approaches found:

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!

 57

Ignorance is bliss

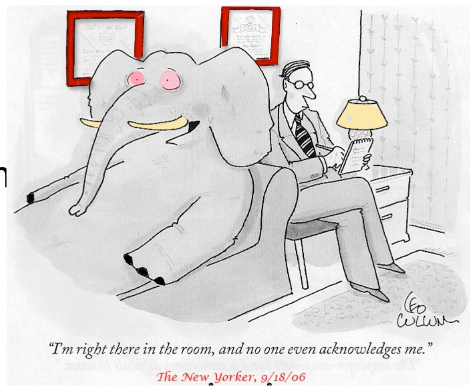
You're just slower, and slower, but you do not know it, or do not know why



 58

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



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

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

 61

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”??

 62

Five star ranking

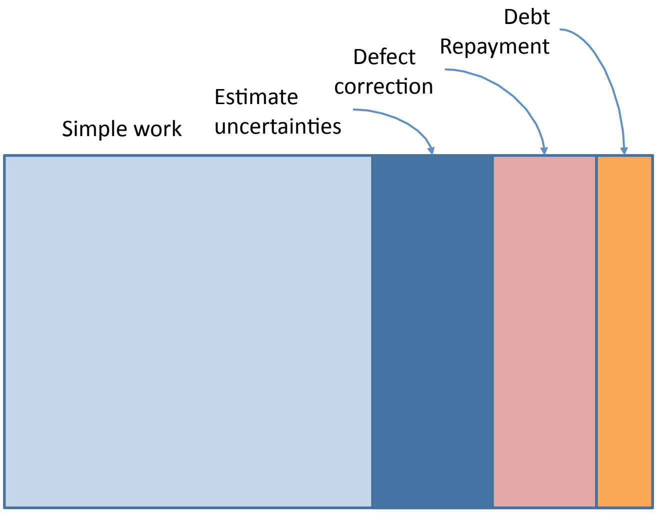
- Define some *maintainability* index
- Benchmark relative to other software in the same category
- Re-assess regularly (e.g., weekly)
- Look at trends, correlate changes with recent changes in code base

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

Constant debt reduction

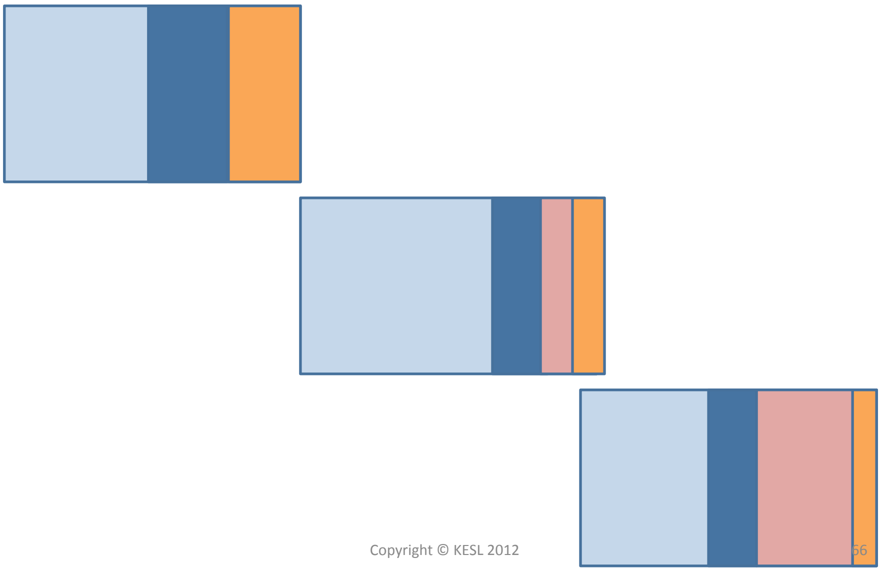
- Make technical debt a visible item on the backlog
- Make it visible outside of the software dev. organization
- Incorporate debt reduction as a regular activity
- Use buffer in longer term planning for yet unidentified technical debt
- Lie (?)

Buffer for debt repayment



 65

A later release



 66

We are agile, so we're immune!

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

 67

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.*

 68

TD: a few suggestions

- Inform

- Identify debt; name it
- Classify debt: code quality, or structural
- Assign value and cost (immediate and future)
- Make it visible (put in backlog)
- Prioritize with other backlog elements

Remember

- Technical debt is not a defect
- Technical debt is not necessarily a bad thing

	Visible	Invisible
Positive Value	**Visible Feature**	**Hidden, architectural feature**
Negative Value	**Visible defect**	**Technical Debt**

Also…

- A suitable system architecture is not likely to spontaneously emerge out of weekly refactorings
- *How much architecture do you need or have?*

- Some novel projects need an
  - Architecture owner

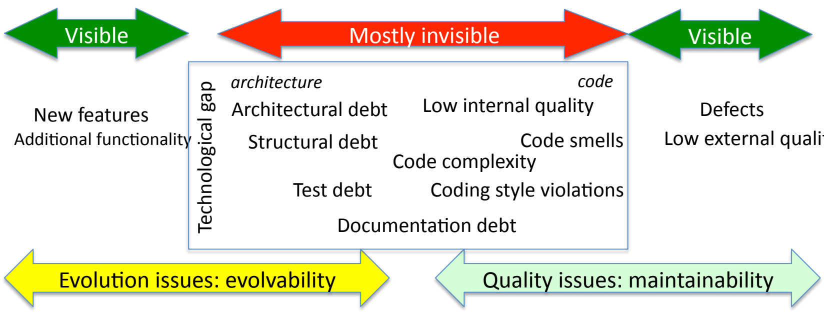
    together with
  - Product owner, and ScrumMaster

 71

Conclusion

- Technical debt is 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.

 73



 74

Upcoming events

- Special issue of IEEE Software on Technical debt November 2012
- Possibly a 4th workshop on Technical Debt at ICSE 2013, in San Francisco
  - Or some other venue…
    - Saturn 2013
    - CompArch 2013

 75

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

 76

References

- Kruchten, P. (2011). Contextualizing Agile Software Development. *Journal of Software Maintenance and Evolution: Research and Practice*. doi: 10.1002/smr.572
- 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.

 77

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

 78

Other pointers

techdebt report http://techdebt.org

ontechnicaldebt http://www.ontechnicaldebt.com/

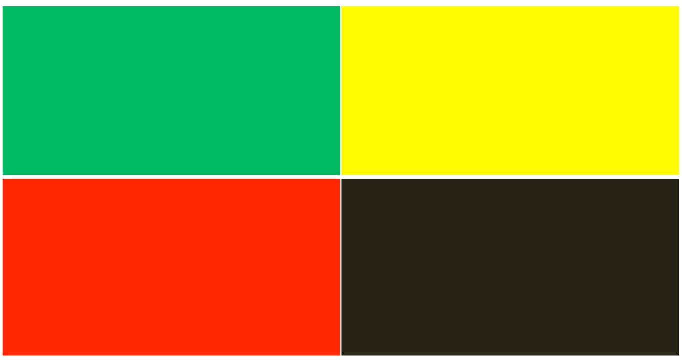
@OnTechnicalDebt

 79

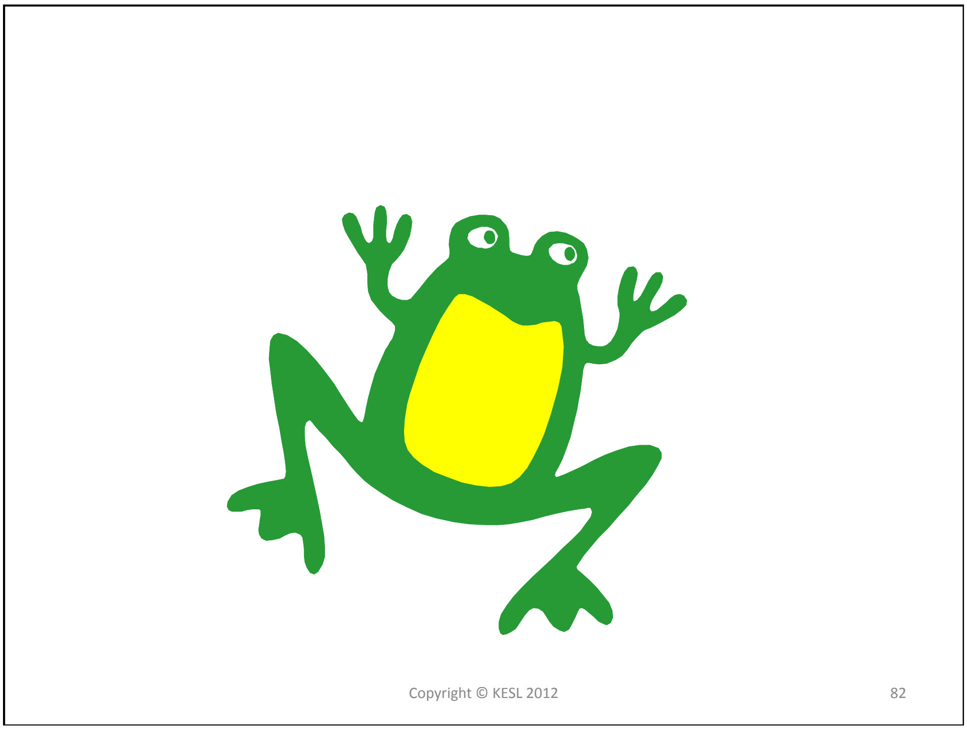
Acknowledgements

- Research on TD partly funded by the Software Engineering Institute | Carnegie Mellon
  - Ipek Ozkaya, Rod Nord, Nanette Brown
  - They have also contributed to building this presentation over the last 2 years.
- UBC master students Erin Lim Kam-Yan and Marco Gonzalez-Rojas …
  - … with some industry partners

 80



 81


Copyright © KESL 2012
82