Strategic Management of Architectural Technical Debt

Ipek Ozkaya
Senior Researcher

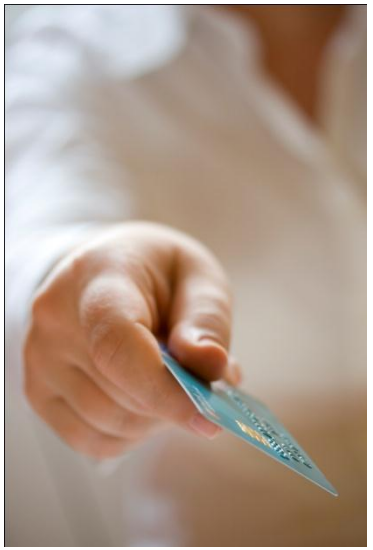
A senior member of the SEI technical staff, Ipek Ozkaya is the co-organizer of the Third International Workshop on Managing Technical Debt, co-creator of the Hard Choices board game, frequent presenter at academic and industry conferences, and author of several articles.

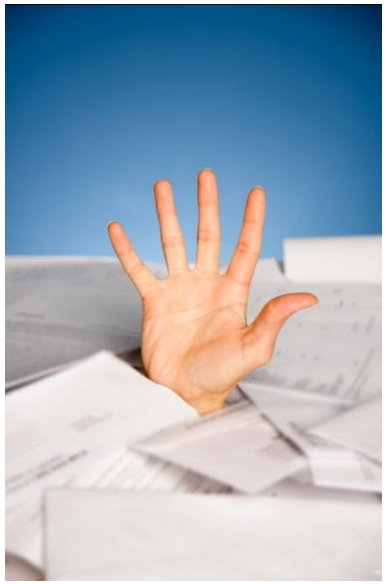
See her full bio at:
www.sei.cmu.edu/go/agile-research-forum/

Technical Debt Metaphor

"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..."





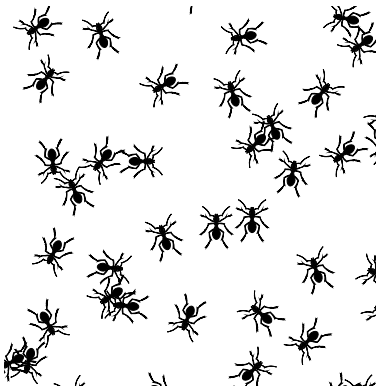
"… The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt."

Cunningham, W. 1992. *The WyCash Portfolio Management System*. OOPSLA '92 Experience Report. http://c2.com/doc/oopsla92.html.

How are Architecture and Technical Debt Related?

Can we really avoid technical debt?

increasing scale of systems



heterogeneous and uncertain workforce

systems expected to be in operation and sustainment for decades

Metaphors and Analogies

A metaphor is a cognitive transfer from one domain to another one. The use of metaphor allows experiences from one domain to illuminate our understanding of another domain.

An analogy is a comparison between two things on the basis of their structure and the purpose of explanation and clarification.

Wikipedia

Technical Debt Analogy

When and how was the debt signed under?

What is the payback term?

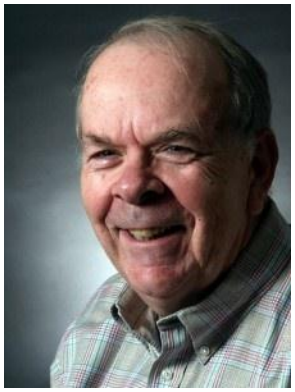
What is the interest rate?

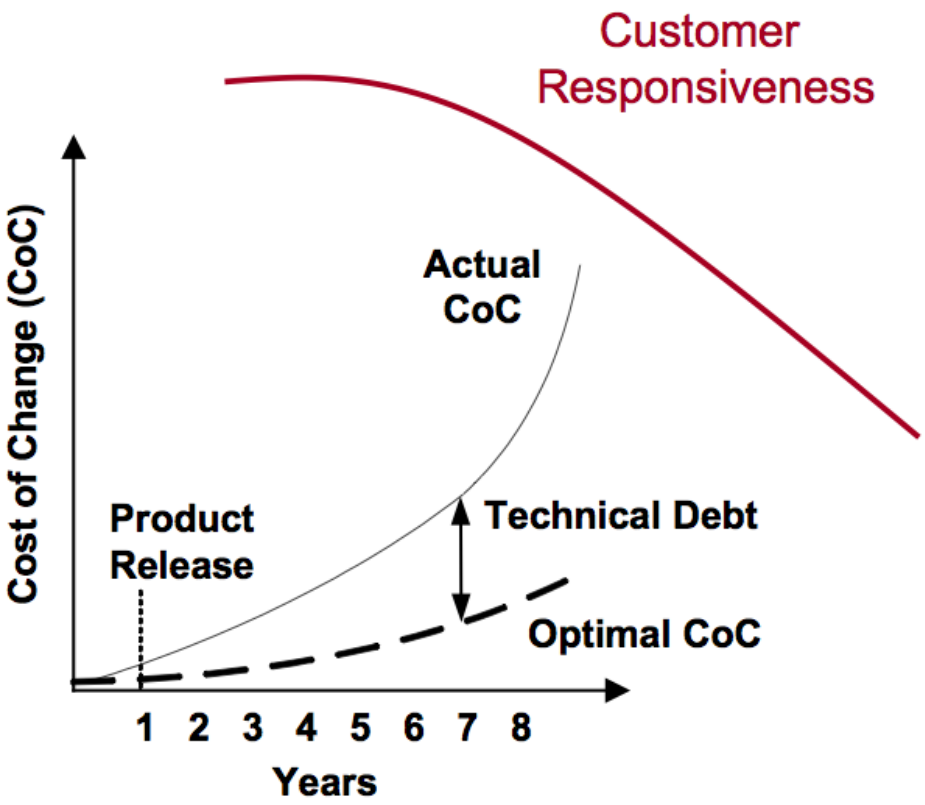
Technical Debt –
Steve McConnell

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

McConnell, S. 2007. *Technical Debt. 10x Software* Development [cited 2010 June 14]; http://blogs.construx.com/blogs/stevemcc/archive/2007/11/01/technical-debt-2.aspx.

Technical Debt – Jim Highsmith





- Only 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

Highsmith, J. 2009. Agile Project Management: Creating Innovative Products , Addison-Wesley.

Technical Debt –
Philippe Kruchten

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

Kruchten, P. 2009. *What colour is your backlog?* Agile Vancouver Conference. http://pkruchten.wordpress.com/Talks.

Polling Question

In which of these areas do you observe technical debt the most?

- Code; our code has become very hard to maintain due to clones, cycles, random bug fixes
- Architecture; we have made suboptimal architectural decisions that we need to rearchitect soon
- Process; we have skipped practices such as reviews, necessary testing and documentation that we are now paying for
- All of the above
- None of the above

Only Three Strategies

Do nothing, it gets worse

Replace, high cost/risk

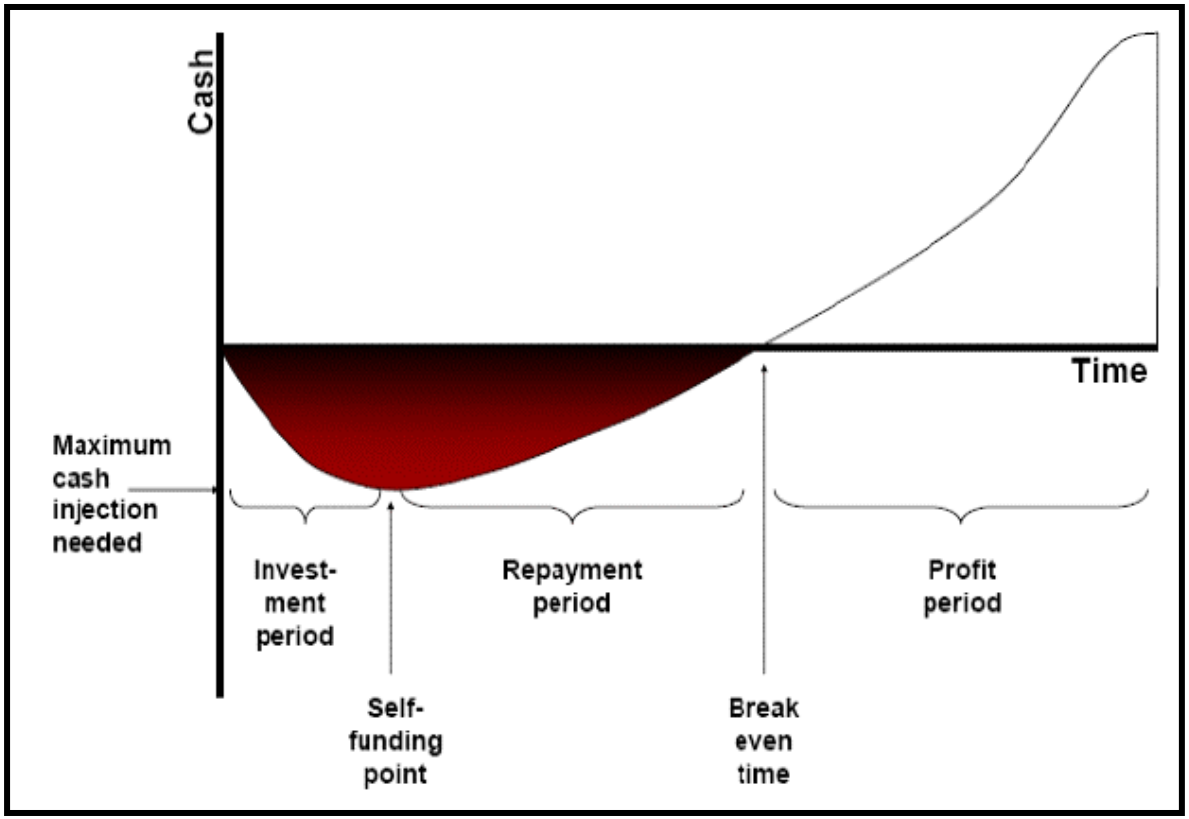
Incremental refactoring, commitment to invest

Why Take on Debt

Shortening time to market

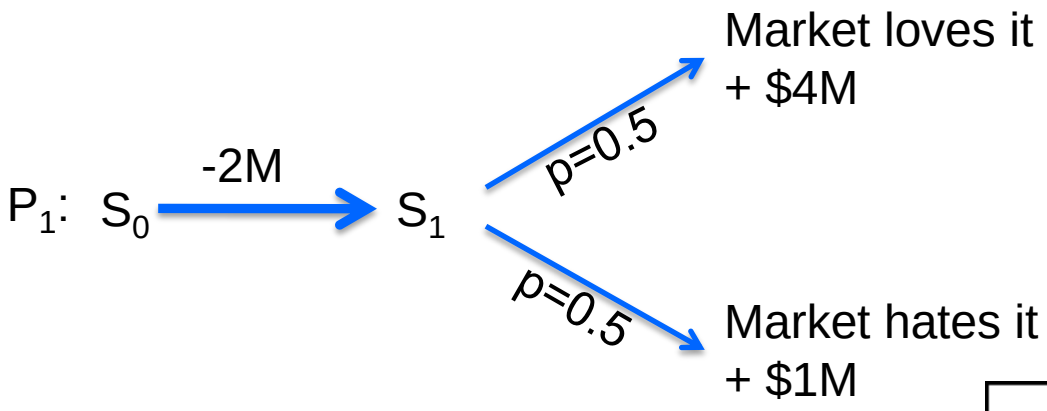
Preservation of startup capital

Delaying development expense



Denne, M., Cleland-Huang, J. 2003. Software by Numbers, Prentice Hall.

Deciding to Take on Debt -1



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

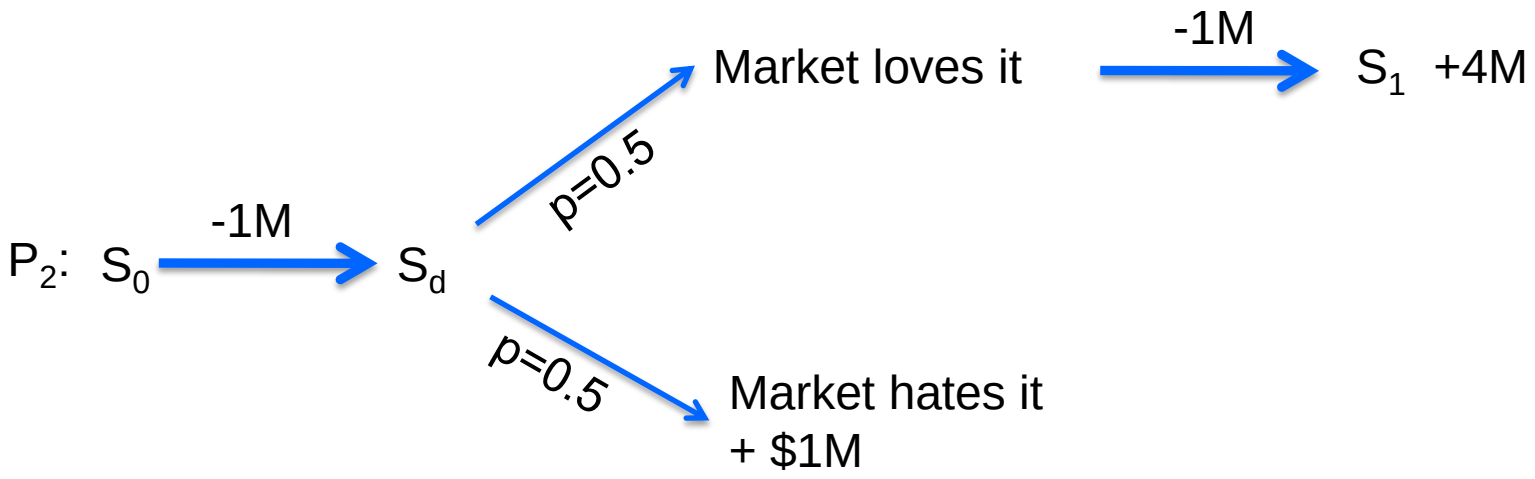
Net Present Value (NPV)

$$NPV = \sum_{t=1}^{T} \frac{\text{Cash Flow}_t}{(1+i)^t} - \text{Initial Cash Investment}$$

t = Cash Flow Period
i = Interest Rate Assumption

Denne, M., Cleland-Huang, J. 2003. Software by Numbers, Prentice Hall.

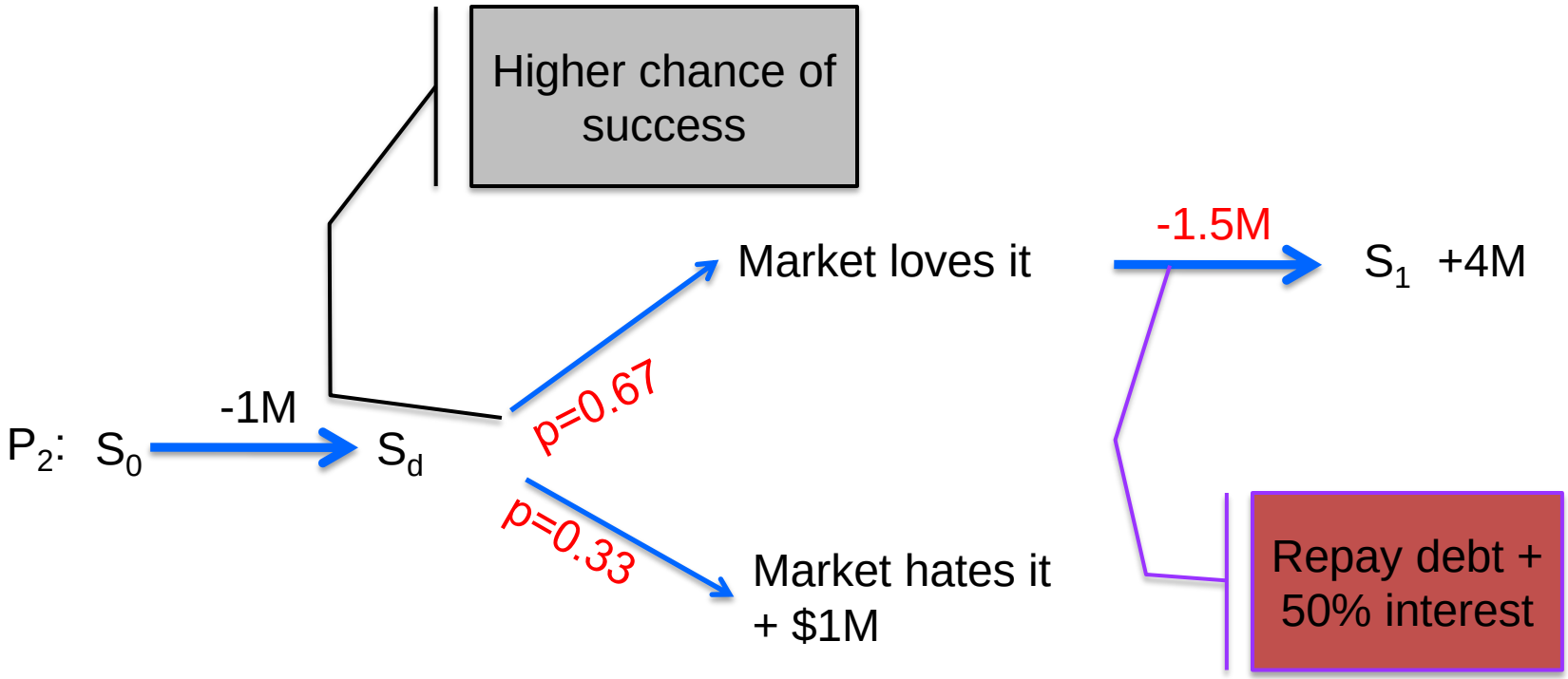
Deciding to Take on Debt -2



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

Taking *technical debt* has increased system value.

Deciding to Take on Debt -3



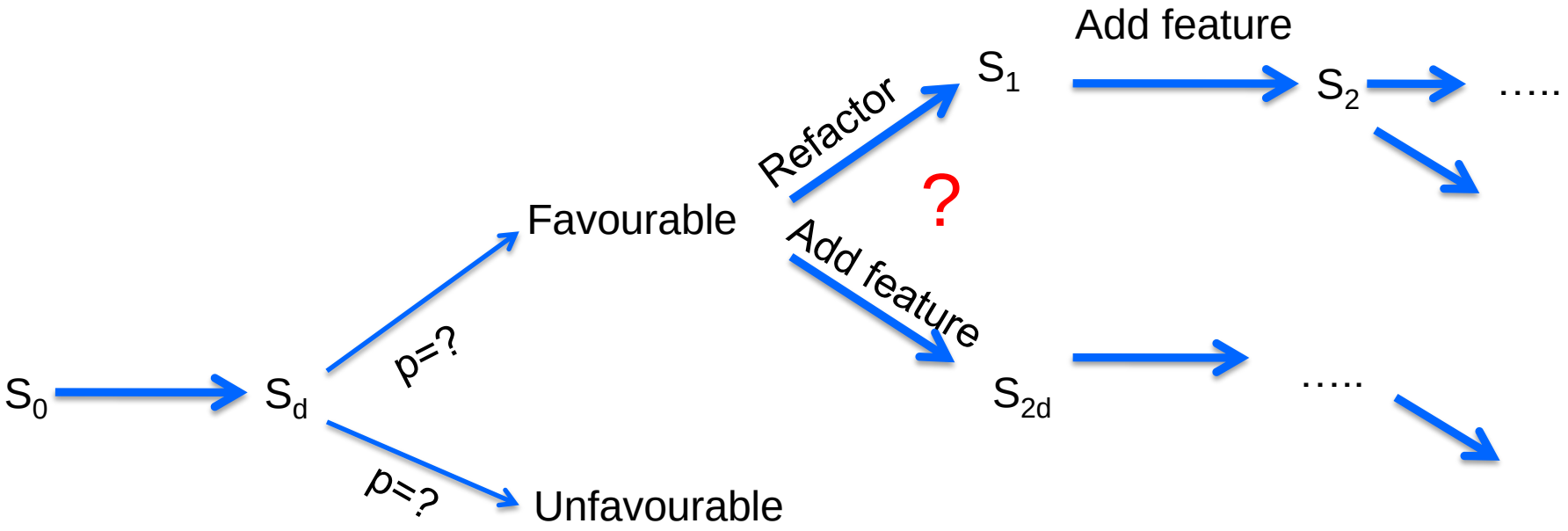
$$NPV\ (P_3) = -1M + 0.67 \times 2.5M + 0.33 \times 1M = 1.005\ M$$

More realistically:

Debt + interest

High chances of success

Deciding to Take on Debt -4



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

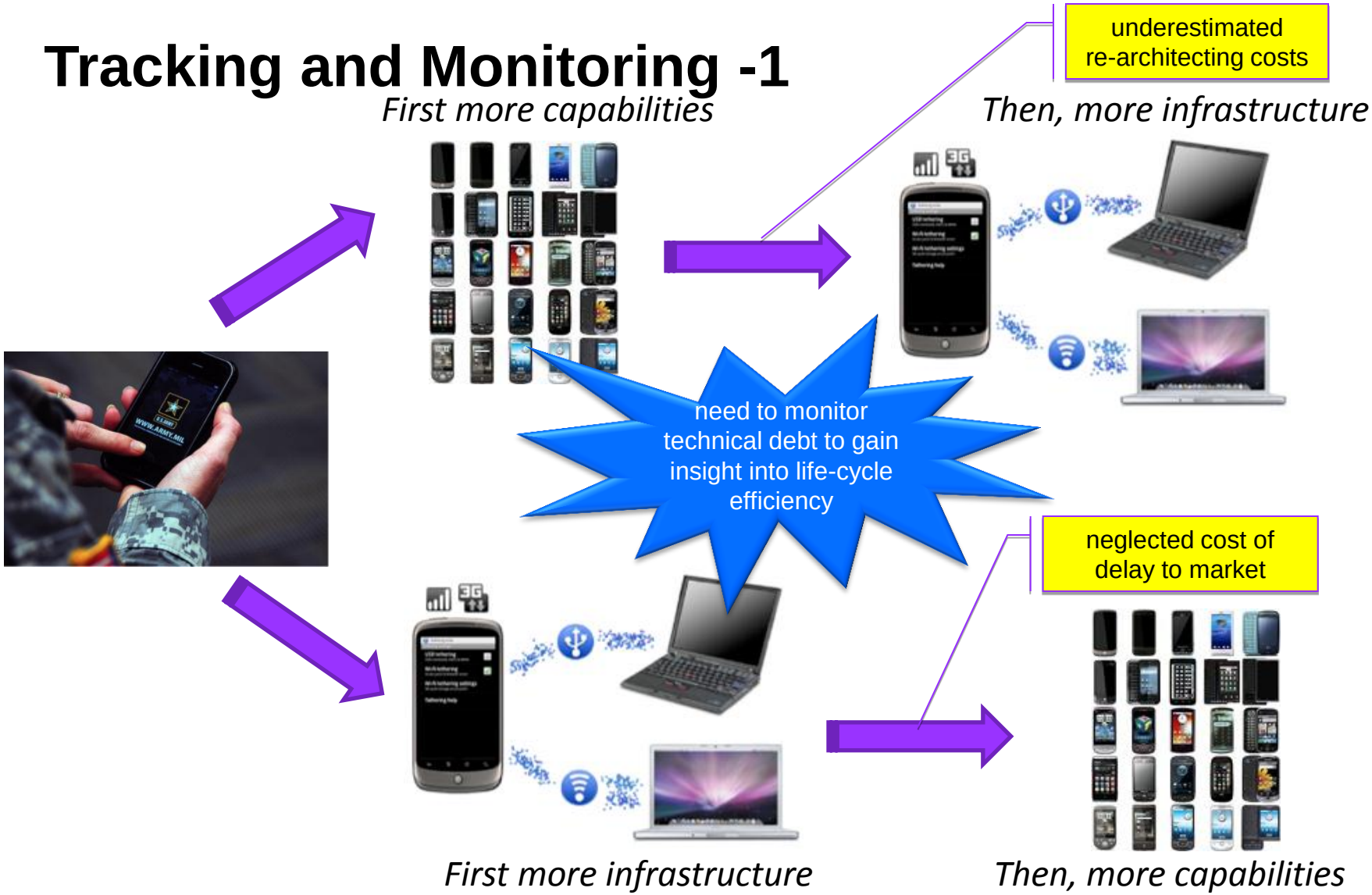
Tracking and Analyzing Debt

Eliciting debt and quantifying the impact is not a repeatable engineering practice yet.

- Factors to consider include defects, velocity, cost of rework
- Mapping such indicators onto cost of development
- Comparing with the value of paying back debt versus not

Analysis tools, mostly looking at code metrics, provide quality insights

Tracking and Monitoring -1

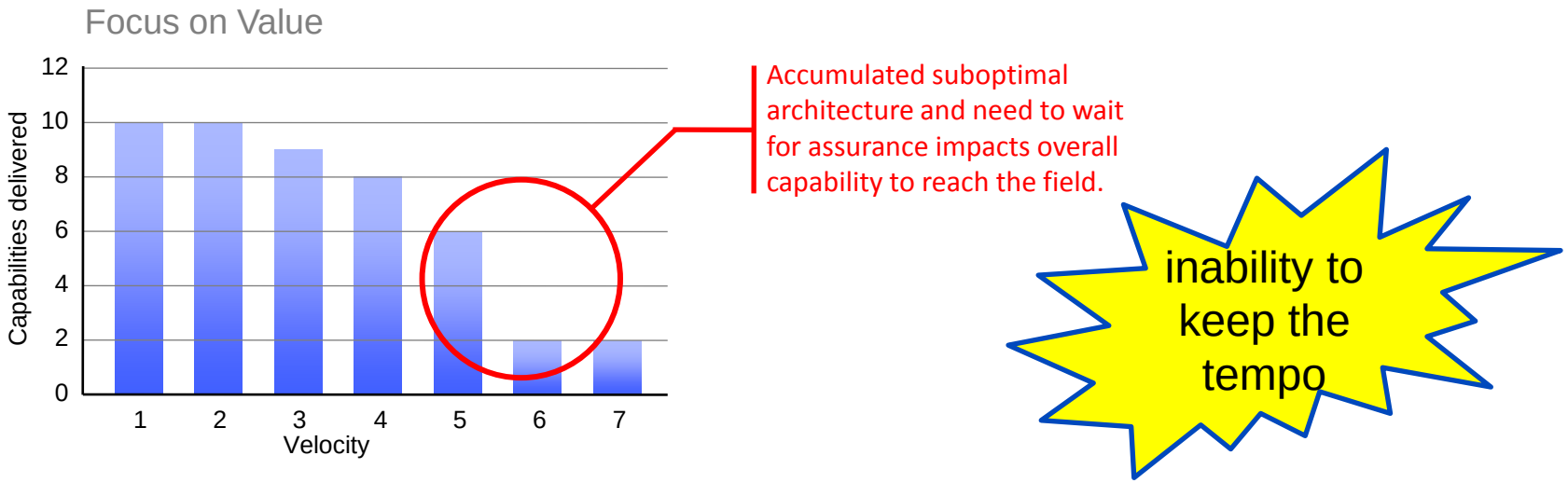


Brown, N., Nord, R., Ozkaya, I. 2010. Enabling Agility through Architecture, *Crosstalk*, Nov/Dec 2010.

Tracking and Monitoring -2

Standard iteration management in agile development

➔ functional, high-priority stories allocated first.

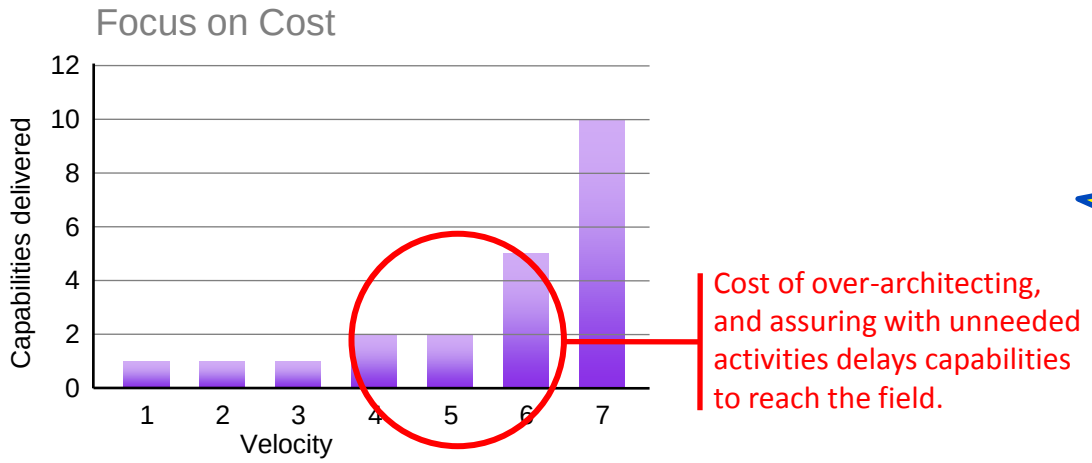


Tracking and monitoring mechanism is solely based on customer features delivered.

Tracking and Monitoring -3

Standard iteration management in architecture-centric development processes

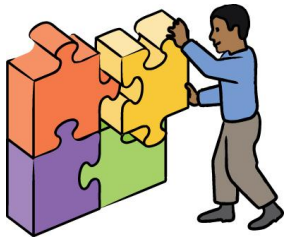
➔ up-front requirements and design tasks allocated first.



No explicit and early tracking and monitoring mechanisms that is development artifact specific.

Measurable Insights into Delivery

Path 1: *value focused; functionality first.*

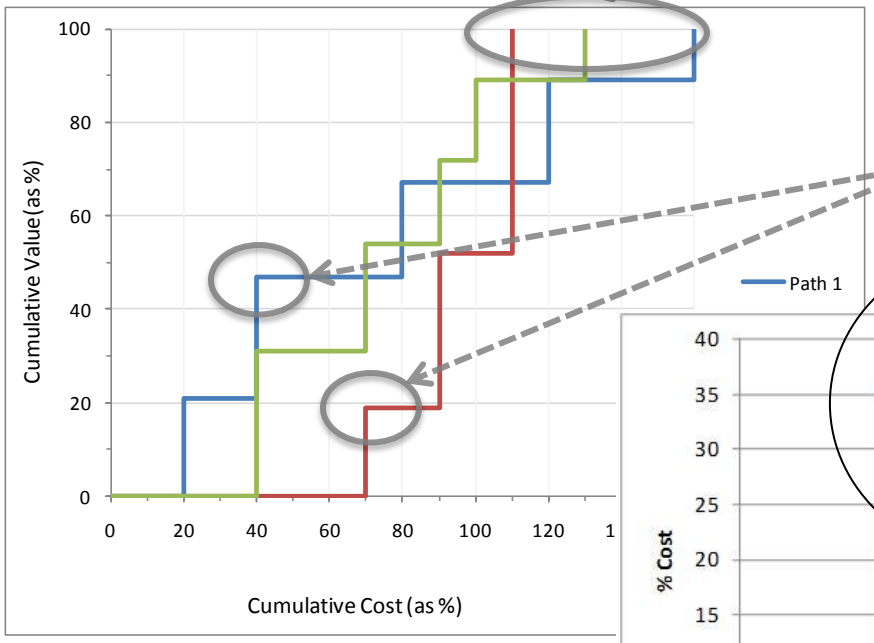


Path 2: *cost focused; architecture push*.

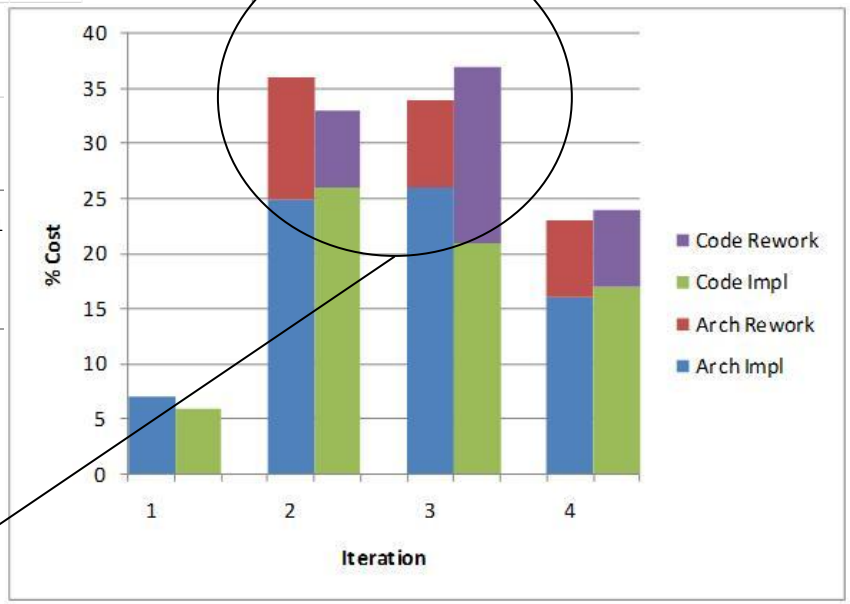
Economic Models

Added cost as a result of implementing architecture in retrospect

Capabilities delivered at different times



Value of Capabilities Delivered over Total Effort



aggregating the amount of rework at each iteration based on architecture changes

Technical Debt and Tools

There has been an increasing focus on tools for the purpose of structural analysis.

Trends show

- increasing sophistication,
- support for some structural analysis in addition to code analysis,
- first steps towards analyzing financial impact by relating structure analysis to cost and effort for rework.

Tools for Structural Analysis

Architecture-Related Capabilities*

- Architecture Visualization Techniques
- Architecture Quality Analysis Metrics
- Architecture Compliance Checking
- Architecture “Sandbox”

*Not all tools have all capabilities

Tools for Technical Debt Analysis –
Architecture Visualization Techniques*

- Dependency Structure Matrix
- Conceptual Architecture
- Architectural Layers
- Dependency Graph

Telea, A.; Voinea, L.; Sassenburg, H.; , "Visual Tools for Software Architecture Understanding: A Stakeholder Perspective," Software, IEEE , vol.27, no.6, pp.46-53, Nov.-Dec. 2010.

* Not all tools use all architecture visualization techniques.

Tools for Technical Debt Analysis –

Architecture Quality Analysis Metrics* Capabilities -5

- Component Dependencies
- Cyclicity
- Architectural Rules Compliance
- Architectural Debt

* The following slides show representative architecture quality analysis metrics from the following sources:

• Lattix. Lattix Releases Lattix 5.0. http://www.lattix.com/node/38

• SonarSource, Sonar. Metric definitions. http://docs.codehaus.org/display/SONAR/Metric+definitions

• Hello2morrow, SonarJ. Sonar Integration. http://www.hello2morrow.com/products/sonarj/sonar

• CAST, Application Intelligence Platform. http://www.castsoftware.com/Product/Application-Intelligence-Platform.aspx

Not all tools have all Architecture Quality Analysis Metrics

Deciding to Pay Down Debt

Eliciting business indicators that accumulate the interest on debt:

- Increased amount of defects
- Slowing rate of velocity
- Change of business and technology context
- A future business opportunity
- Time to market

Strategies and Techniques

Part of standard operating procedure

- Adding technical debt to the backlog
- Amortize 10%
- Specialized iteration: hardening cycle, hackathon

Architectural tactics*

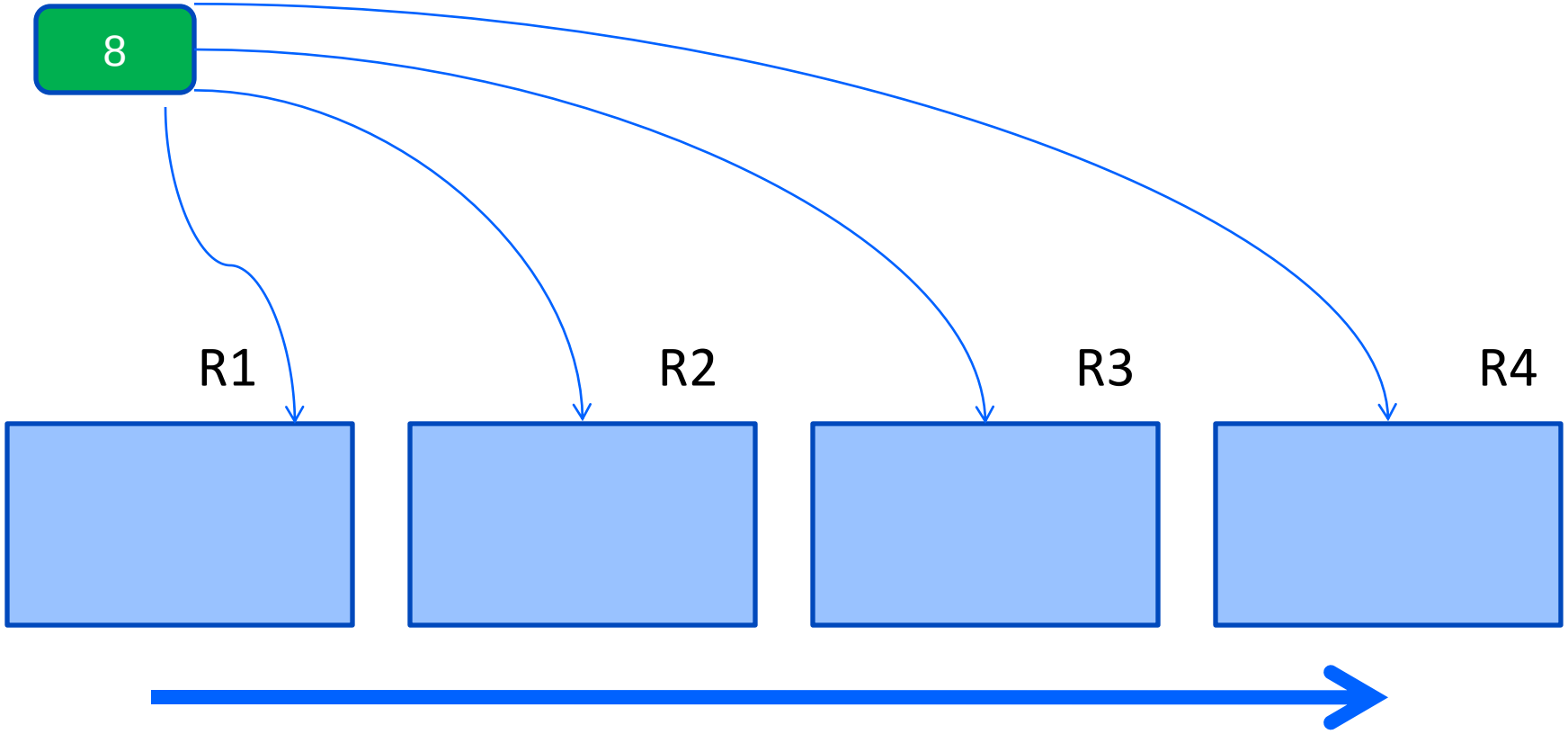
- Align feature and system decomposition
- Architectural runway
- Matrixed teams

* Bachmann, F.; Nord, R.; Ozkaya, I..; , “Architectural Tactics to Support Rapid and Agile Stability" *Crosstalk* , May/June 2012.

Economic Models –

Release Planning – 1

In which release?

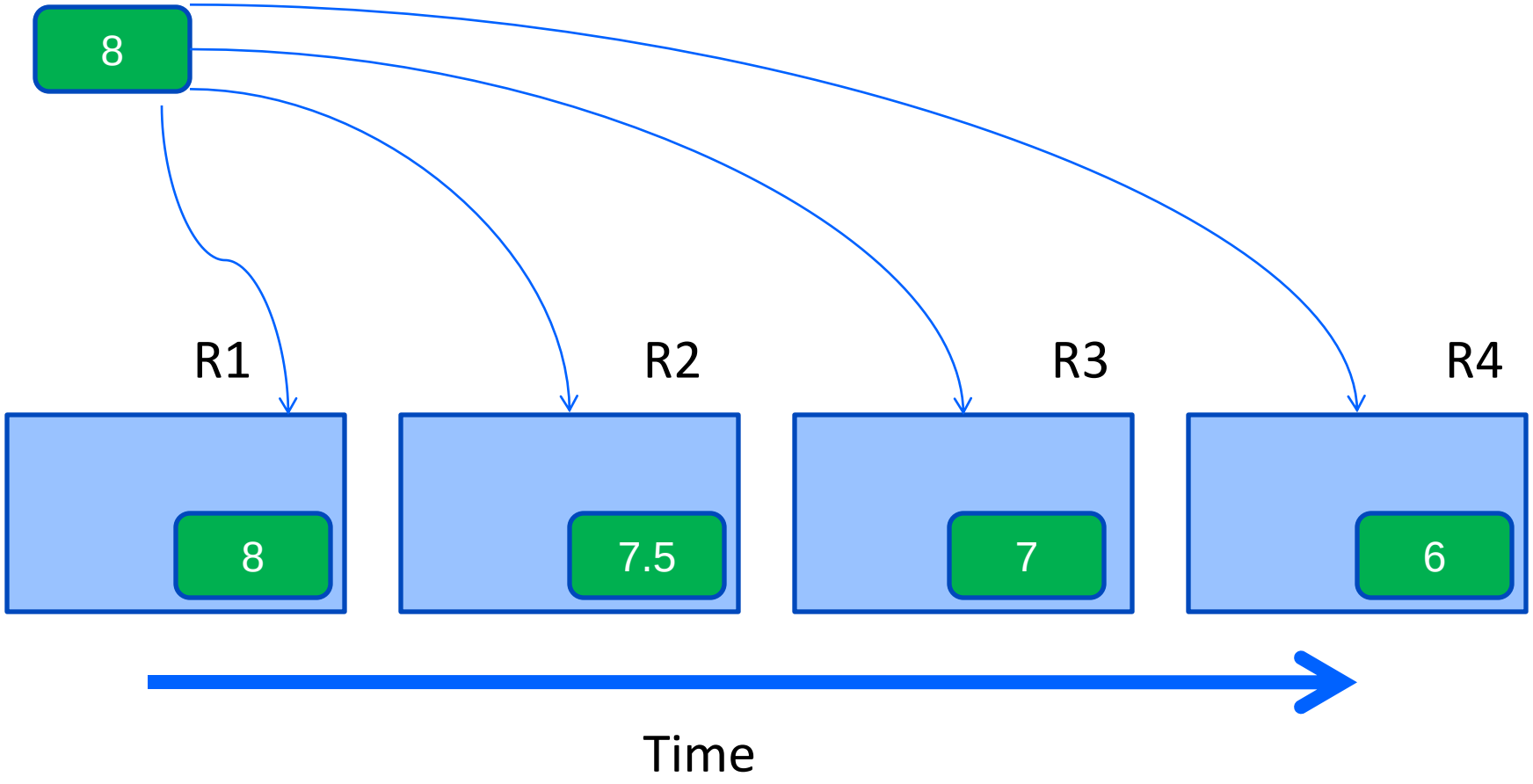


Kruchten, P. 2009.
What colour is your backlog?
Agile Vancouver Conference.
http://pkruchten.wordpress.com/Talks.

Economic Models –

Release Planning – 2

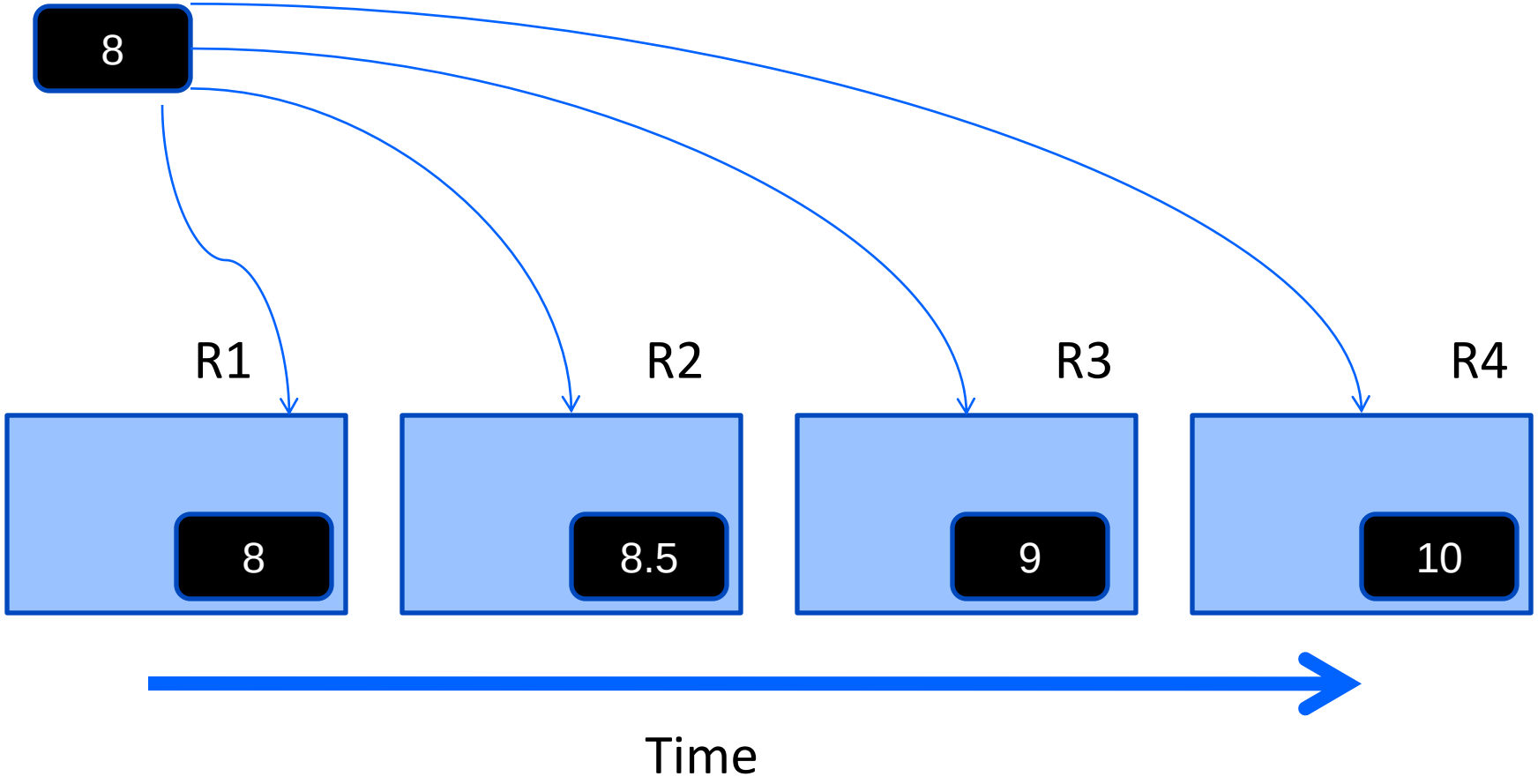
Value decreases over time



Economic Models –

Release Planning – 3

Technical debt increases?



Future Directions in Technical Debt Analysis –
Open Areas of Investigation -1

Technical debt succinctly communicates the issues observed in large-scale, long- term projects:

- There is an optimization problem where optimizing for the short-term puts the long-term into economic and technical jeopardy when debt is unmanaged.
- Design shortcuts can give the perception of success until their consequences slow down projects.
- Software development decisions, especially architectural ones, need to be continuously analyzed and actively managed as they incur cost, value, and debt.

Future Directions in Technical Debt Analysis –
Open Areas of Investigation -2

How to locate most effective refactoring opportunities?

How to identify and manage strategic architectural debt?

How does debt manifest itself in non-code artifacts?

How does debt relate to development processes?

How can debt be visualized with effective tool support?

How to identify dominant sources of debt?

How and when to pay back debt?

How to measure debt?

Brown, N., Cai, Y., Guo, Y., Kazman, R. , Kim, M., Kruchten, P., Lim, E. , MacCormack, A., Nord, R., Ozkaya, I., Sangwan, R., Seaman, C. , Sullivan, K., Zazworka, N. , 2010. Managing Technical Debt in Software-Reliant Systems, *FSE/SDP Workshop on the Future of Software Engineering Research*, Santa Fe November 2010.

Things We Can Do Now

Practices that can be incorporated into managing projects:

- Make technical debt visible.
- Differentiate strategic structural technical debt from technical debt that emerges from low code quality.
- Bridge the gap between the business and technical sides.
- Integrate technical debt into planning.
- Associate technical debt with risk.

Concluding Thoughts

Agile manifesto

the only true measure of progress on a software development project is the *delivery of working software*

Architecture proposition

the only true measure of progress on a software development project is the *delivery of working software* ***that meets its business and quality goals***

Integrated approach for large scale systems

the only true measure of progress on a software development project is the *delivery of working software* ***that meets its business and quality goals today*** ***and in the future through balancing anticipation and adaptation***

NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this presentation is not intended in any way to infringe on the rights of the trademark holder.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

Contact Information

RTSS Program

Linda Northrop
RTSS Program Director
Software Engineering Institute
Pittsburgh, PA 15213
lmn@sei.cmu.edu

Architecture Practice

Robert Nord, Ipek Ozkaya
Software Engineering Institute
rn, ozkaya @sei.cmu.edu

Philippe Kruchten
University of British Columbia
pbk@ece.ubc.ca

Further Reading -1

Cunningham, W. 1992. *The WyCash Portfolio Management System*. OOPSLA '92 Experience Report. http://c2.com/doc/oopsla92.html.

McConnell, S. 2007. *Technical Debt. 10x Software* Development [cited 2010 June 14];

http://blogs.construx.com/blogs/stevemcc/archive/2007/11/01/technical-debt-2.aspx.

Fowler, M. 2009. *Technical debt quadrant,* Blog post at: http://martinfowler.com/bliki/TechnicalDebtQuadrant.html.

Sterling, C. 2010. Managing Software Debt: Building for Inevitable Change, Addison-Wesley Professional.

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

Highsmith, J. 2009. Agile Project Management: Creating Innovative Products , Addison-Wesley.

Kruchten, P. 2009. *What colour is your backlog?* Agile Vancouver Conference. http://pkruchten.wordpress.com/Talks.

Bachmann, F.; Nord, R.; Ozkaya, I..; , “Architectural Tactics to Support Rapid and Agile Stability" *Crosstalk* , May/June 2012.

Brown, N., Nord, R., Ozkaya, I. 2010. Enabling Agility through Architecture, *Crosstalk*, Nov/Dec 2010.

Further Reading -2

Brown, N., Nord, R., Ozkaya, I. , Pais, M. 2011. Analysis and Management of Architectural Dependencies in Iterative Release Planning, Working IEEE/IFIP Conference on Software Architecture, June 2011.

Brown, N., Kruchten, P., Lim, E., Nord, R., Ozkaya, I. 2010. Hard Choices Game Explained.

Hard Choices Strategy Game to Communicate Value of Architecture Thinking game downloadable from http://www.sei.cmu.edu/architecture/tools/hardchoices/

Poppendieck, M., Poppendieck, T. 2009. Leading Lean Software Development, Addison-Wesley Professional.

Denne, M., Cleland-Huang, J. 2003. Software by Numbers, Prentice Hall.

Brown, N., Cai, Y., Guo, Y., Kazman, R. , Kim, M., Kruchten, P., Lim, E. , MacCormack, A., Nord, R., Ozkaya, I., Sangwan, R., Seaman, C. , Sullivan, K., Zazworka, N. , 2010. Managing Technical Debt in Software-Reliant Systems, *FSE/SDP Workshop on the Future of Software Engineering Research*, Santa Fe November 2010.

Telea, A.; Voinea, L.; Sassenburg, H.; , "Visual Tools for Software Architecture Understanding: A Stakeholder Perspective," Software, IEEE , vol.27, no.6, pp.46-53, Nov.-Dec. 2010.