Why Should Government Care About Technical Debt and Software Architecture?

Ipek Ozkaya
(ozkaya@sei.cmu.edu)

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

© 2013 Carnegie Mellon University

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **13 MAR 2014**	2. REPORT TYPE	3. DATES COVERED **00-00-2014 to 00-00-2014**

4. TITLE AND SUBTITLE **Why Should Government Care About Technical Debt and Software Architecture?**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Carnegie Mellon University ,Software Engineering Institute,Pittsburgh,PA,15213**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR’S ACRONYM(S)
	11. SPONSOR/MONITOR’S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
presented at the Agile for Government Summit webinar on 13 Mar 2014.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**19**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

Copyright 2013 Carnegie Mellon University

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

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.

This material has been approved for public release and unlimited distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

DM-0000765

Objective

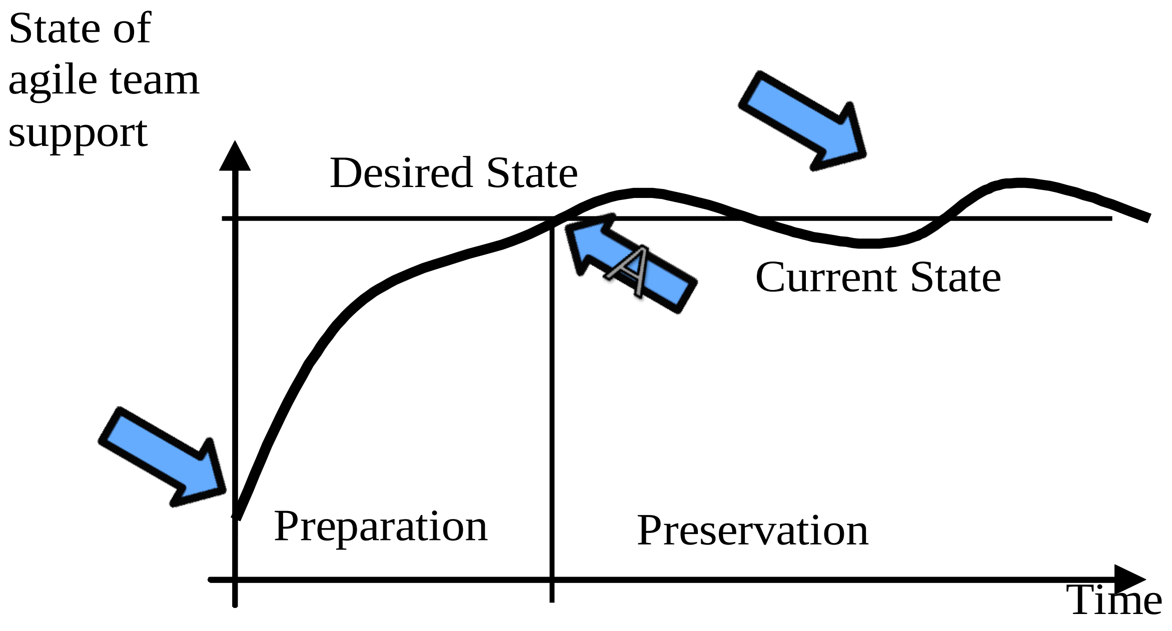
Understand what technical debt is

Provide a different perspective on software development and architecture through managing technical debt

© 2013 Carnegie Mellon University

Support for Delivery Over Time

Projects should not simply produce a product design;
they should plan a desired state that enables teams to quickly deliver releases that stakeholders value (or in terms of lean practices, design a profitable operational value stream for rapidly delivering that product).



F. Bachmann, R. L. Nord, I. Ozkaya, “Architectural Tactics to Support Rapid and Agile Stability.” *CrossTalk 25*, 3 (May/June 2012): 20-25.

© 2013 Carnegie Mellon University

Technical Debt*

A design or construction approach that's 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) *S. McConnell*

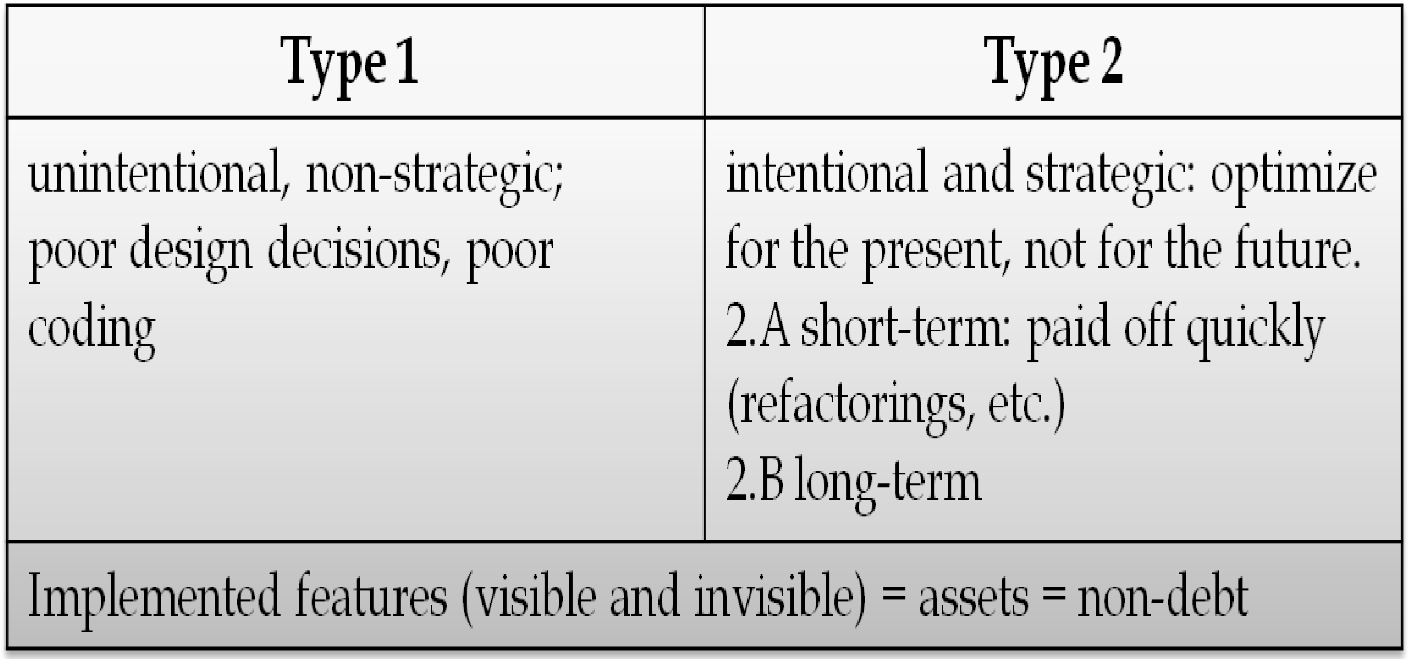
Some examples include:

- Continuing to build on a foundation of poor quality legacy code
- Prototype that turns into production code
- Increasing use of "bad patches", which increases number of related systems that must be changed in parallel

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

© 2013 Carnegie Mellon University

Technical Debt –
Steve McConnell

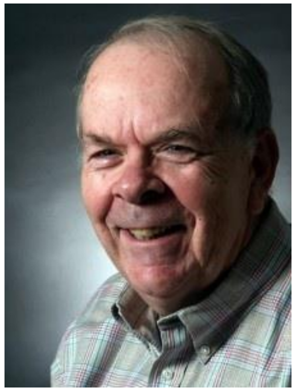


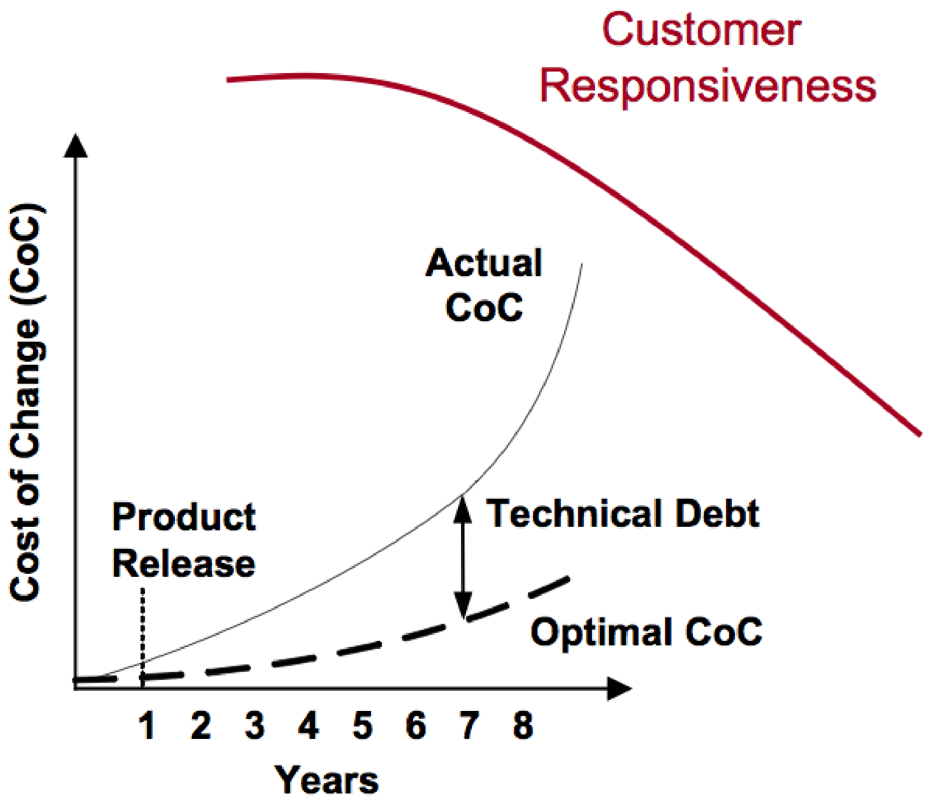
Type 1	Type 2
unintentional, non-strategic; poor design decisions, poor coding	intentional and strategic: optimize for the present, not for the future. 2.A short-term: paid off quickly (refactorings, etc.) 2.B long-term
Implemented features (visible and invisible) = assets = non-debt	

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.

© 2013 Carnegie Mellon University

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.

© 2013 Carnegie Mellon University

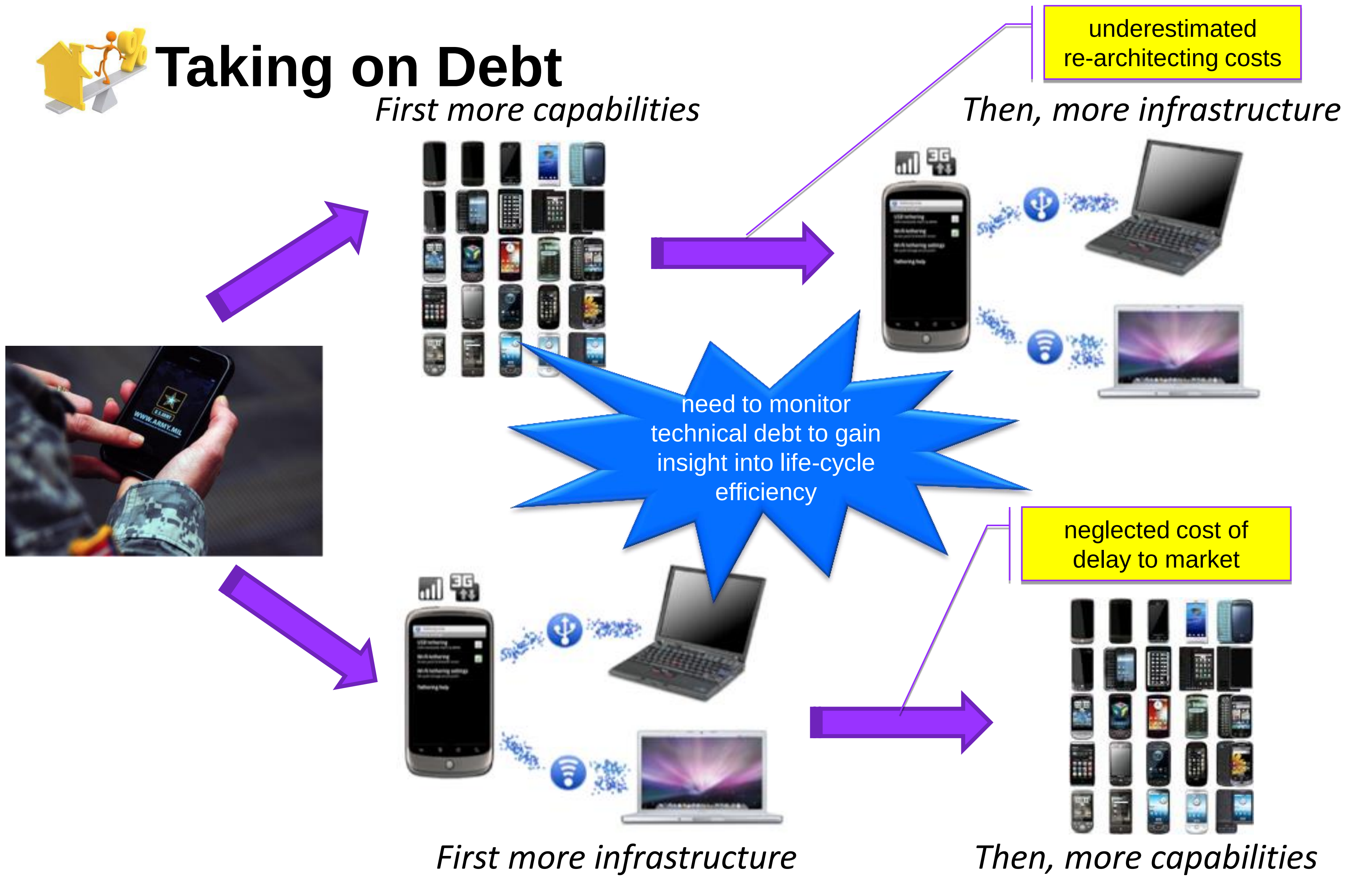
Technical Debt Analogy

When and how was the debt signed under?

What is the interest rate?

What is the payback strategy?

Taking on Debt

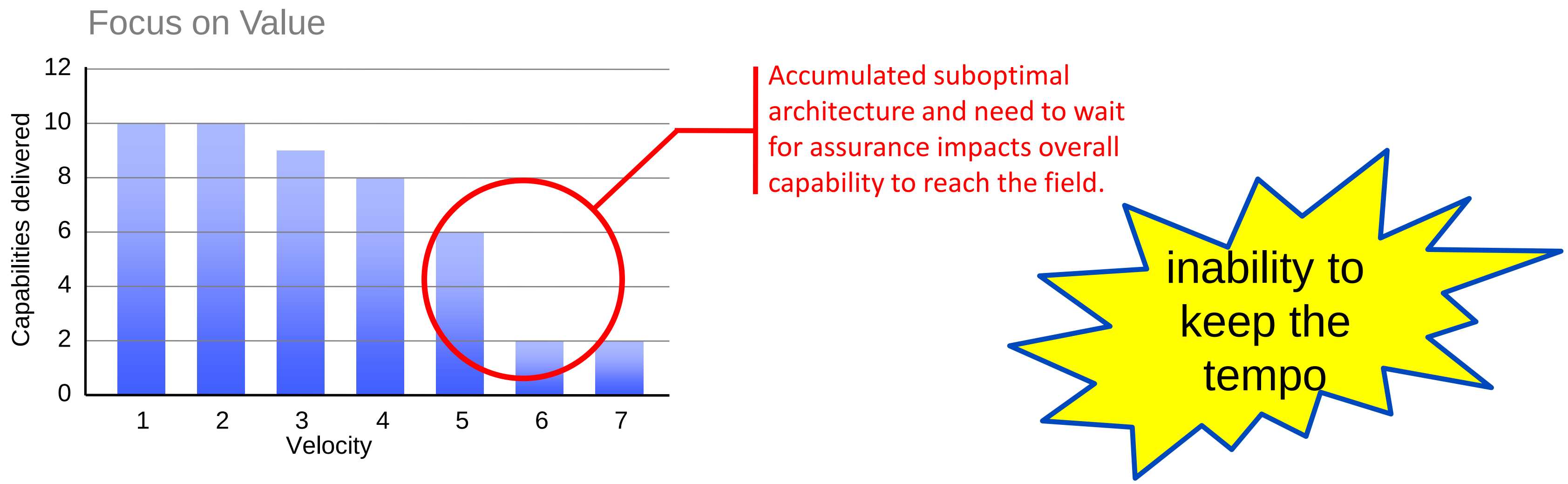

First more capabilities
underestimated re-architecting costs
Then, more infrastructure
need to monitor technical debt to gain insight into life-cycle efficiency
neglected cost of delay to market
First more infrastructure
Then, more capabilities

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

Understanding the interest rate – 1

Standard iteration management in agile development

➔ functional, high-priority stories allocated first.



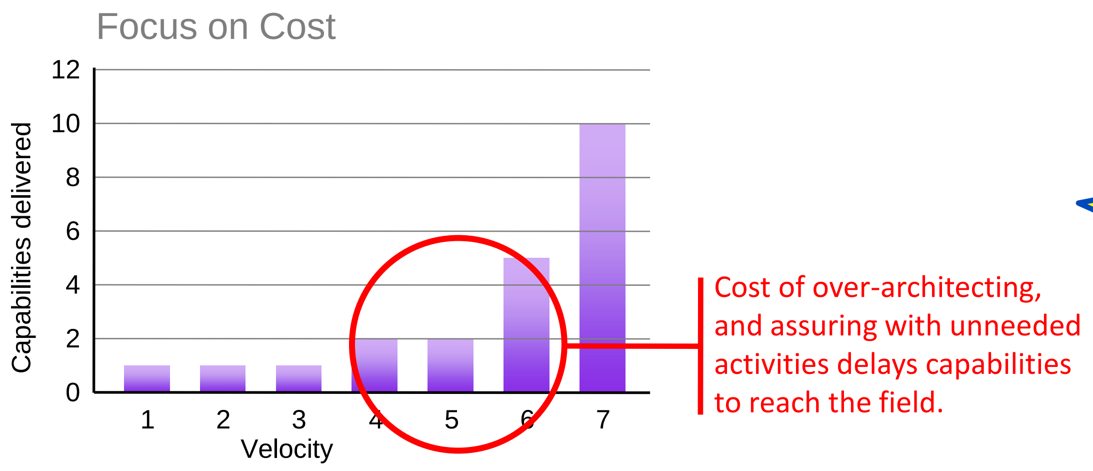
Tracking and monitoring mechanism is solely based on customer features delivered.

© 2013 Carnegie Mellon University

Understanding the interest rate – 2

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.

© 2013 Carnegie Mellon University

Only Three Strategies

Do nothing, it gets worse

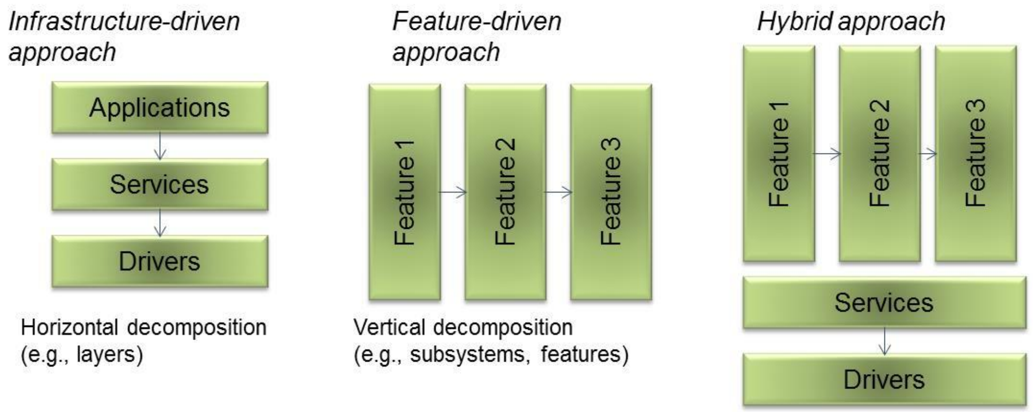
Replace, high cost/risk

Incremental refactoring, commitment to invest

© 2013 Carnegie Mellon University

Tactics to consider

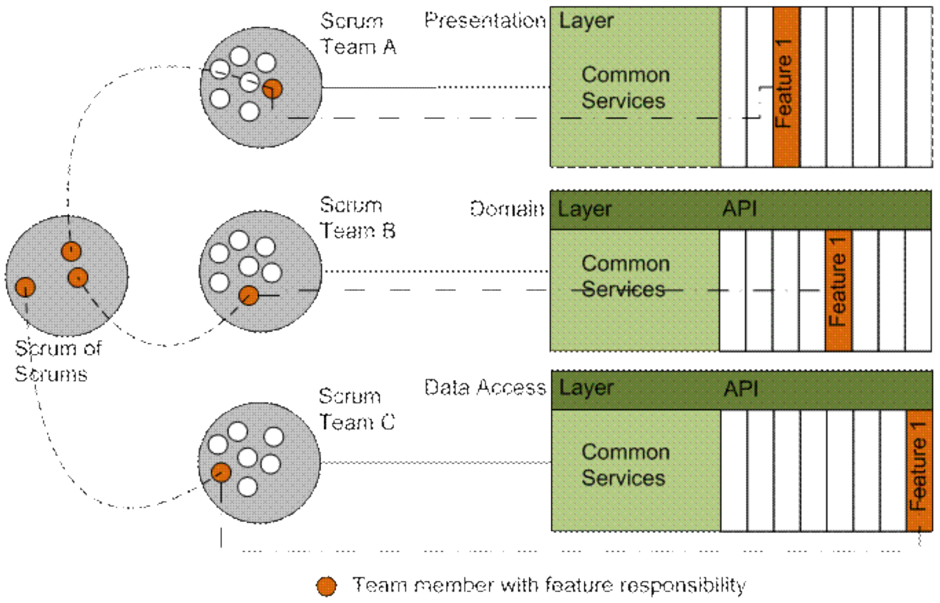
Align feature and system decomposition.



Infrastructure-driven approach
Applications
Services
Drivers
Horizontal decomposition (e.g., layers)
Feature-driven approach
Feature 1
Feature 2
Feature 3
Vertical decomposition (e.g., subsystems, features)
Hybrid approach
Feature 1
Feature 2
Feature 3
Services
Drivers

Create an architectural runway.

Use matrix teams and architecture.



Why Should Government Care About Technical Debt and Software Architecture?

Practical Approaches from the Ground

Warren Ellmore

Everware-CBDI
NORTH AMERICA

Technical Debt is good (as long as it's managed)

- Technical Debt is essentially the result of trade-offs - deferred decisions, deferred priorities, deferred capabilities, deferred skills.
- Technical Debt arises when current sprint work is unblocked by a decision on what can be implemented now and what can be deferred.

Example: You know you need security but decide to defer that for a later sprint so that functional capability can continue to be developed/implemented.

Example: You need to interface with a legacy system but that API isn't ready yet so you decide to hack a quick stub just for now.

Government Context: Unfortunately, Technical Debt is often misunderstood by business owners and frequently ignored in favor of business functionality. Managing and resolving Technical Debt is often more difficult because of contract terms, size and complexity, and a general lack of skills and experience in risk management as it relates to Agile development.

© 2013 Everware-CBDI

Everware-CBDI NORTH AMERICA

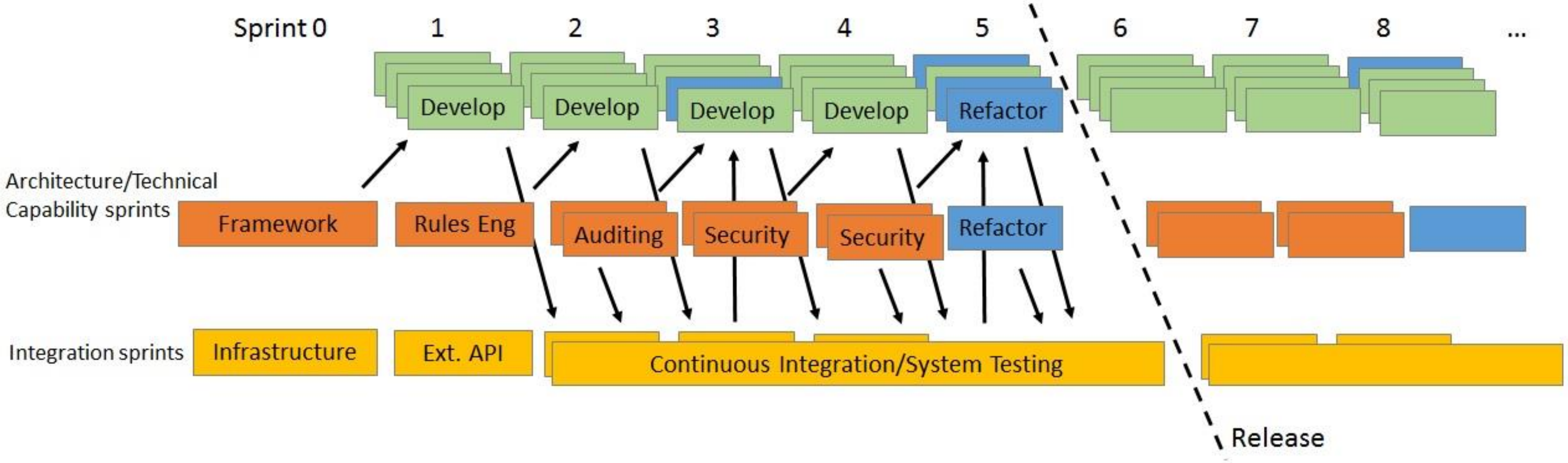
Technical Debt can be Managed

- Requires more than just logging a "ToDo" or adding to the backlog
- All parties must be involved and have insight – PMO responsibility
- Establish and refine an understanding of:
    - Scope of impact and the accumulated risk curve
    - "Value" and priority within the release strategy
    - Dependencies of scheduled functionality on resolution (partial or full)
    - Ongoing decisions can ease or exacerbate a particular debt
- Choose Technologies, Architectures and Frameworks that meet the business/mission requirements and minimize Technical Debt impact
    - Available skills, known technologies vs. new "shiny objects"
    - Leverage componentization – separation of concerns, service architecture
    - Reuse existing capability – services, components, models, patterns, specifications …

© 2013 Everware-CBDI

Technical Debt can be addressed in Sprints

- Plan for periodic refactoring sprints
- Run parallel Architecture/Technical Capability sprints
- Run parallel Integration sprints targeting releases
- Start running functional/performance testing asap and scale with codebase

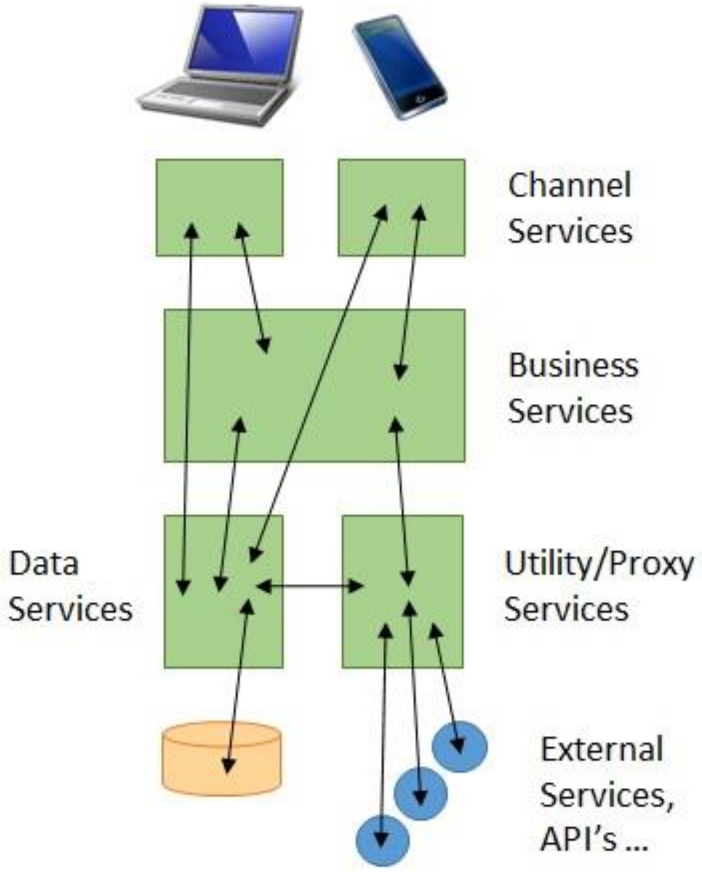


© 2013 Everware-CBDI

Everware-CBDI NORTH AMERICA

Architecture can Reduce the Accumulation and Impact of Technical Debt

- Aim for a “Fuller-stack” Service Architecture
  - Provides isolation reducing change impact scope
  - Provides abstraction for new/untested technology
  - Provides for asset/capability reuse and extension
- Leverage Architectural Models
  - Impact analysis, traceability, knowledge management
  - More easily identify separation points for dividing the work across multiple teams/contracts/providers
  - Evolve to Model-Driven Architecture/Development
    - Business Process Orchestration
    - Code generation, injectable architectural framework



© 2013 Everware-CBDI

Key Takeaways

- Technical debt is unavoidable and can be good – if managed
  - Make architecture features and technical debt visible.
- Plan for its resolution – increase for new/unknown
  - Different kinds of technical debt call for different approaches, e.g. new technology versus low code-quality
- Bridge the gap between the business and technical sides.
  - Associate technical debt with risk.
- Reduce technical debt impact with architecture – sufficient upfront, add capability in parallel sprints.
- Discover unseen technical debt as early as possible by starting continuous integration and system testing following Sprint 1
  - Integrate technical debt into planning and standard operating procedures (e.g., planning, reviews, retrospectives).

© 2013 Everware-CBDI