Managing Technical Debt in Agile Environments

Robert Nord
In collaboration with Stephany Bellomo, Ipek Ozkaya
Software Solutions Division
August 2018

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Document Markings

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

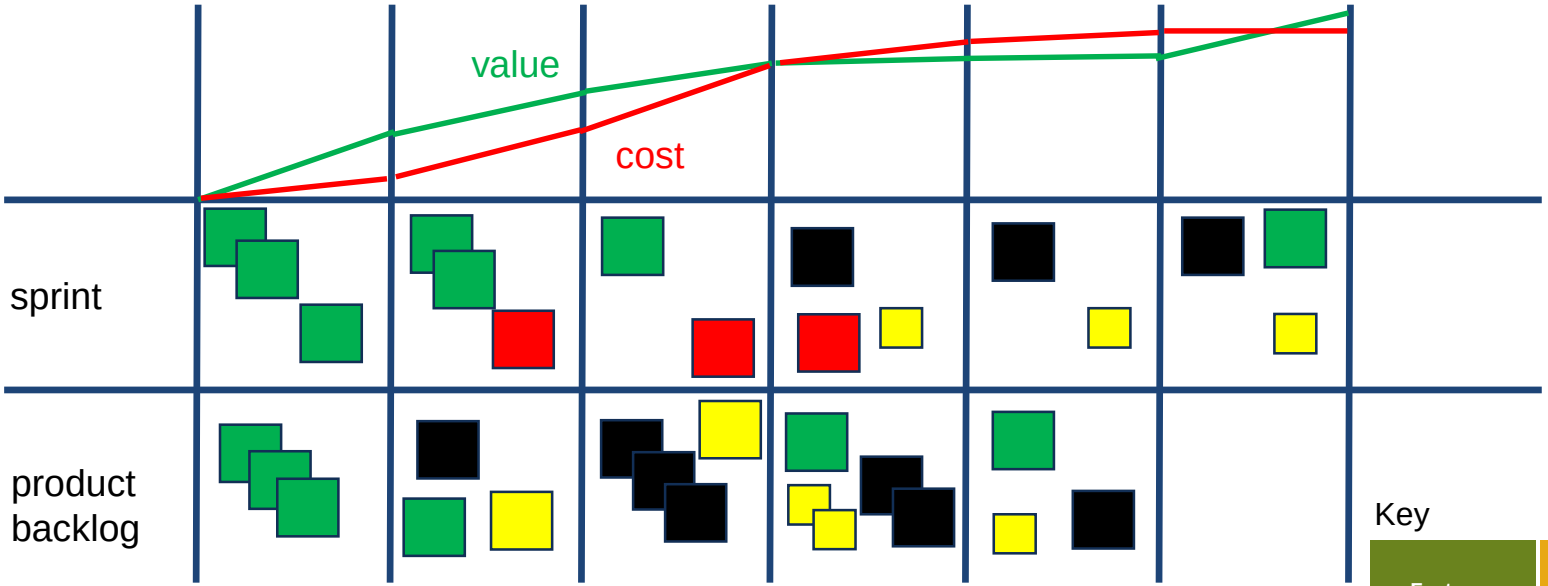
DM18-0922

Abstract

Technical debt can be defined as 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. If managed well, some debt can accelerate design exploration. Left unrecognized and unmanaged, accumulated technical debt results in increased development and sustainment costs.

To meet the challenge of uncovering, communicating, and managing technical debt, the Software Engineering Institute has developed a systematic approach. It includes techniques for making technical debt visible, determining what type of debt the project has, and integrating debt into project planning.

A Technical Debt Story



Managing Technical Debt

Your technical debt toolbox

- Become aware
- Assess the information
- Build a registry
- Decide what to fix
- Take action

Become aware

Toolbox: Become Aware

Ensure all the people involved have a common understanding of what technical is and how it affects any project.

- Provide a clear, simple definition of technical debt in the context of your project.
- Educate the team, people in the project environment, external contractors.
- Create a technical debt category in your issue tracking system.

What is Technical Debt?

Technical debt* is a collection of design or implementation choices that are expedient in the short term, but that can make future changes more costly or impossible.

Technical debt represents current and future liability whose impact is both on the quality of the system as well as overall project resources.

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

Is this an actionable description of technical debt?

A training system, meeting the customers high-level requirements but not meeting the expected functionality of the end users of the system.

- Cause - poor requirements gathering
- Staff shortage, lack of software engineering experience, customer reps lack of understanding and outdated methods of getting from requirements to a design (text documents, no UML or similar model)
- Impact has been a huge re-design, many areas of functionality being altered in the way they work

An Actionable Description

One of two modules was upgraded.

- Unfortunately, the second module required months of unplanned work, due to close-coupling between the modules
- Developers disregarded the scoping rules, due to schedule / budget, which led to module coupling
- Impact: 12 KSLOC unplanned work

Technical debt is a software design issue that:

Exists in an **executable system artifact**, such as code, data model, build scripts, automated test suites;

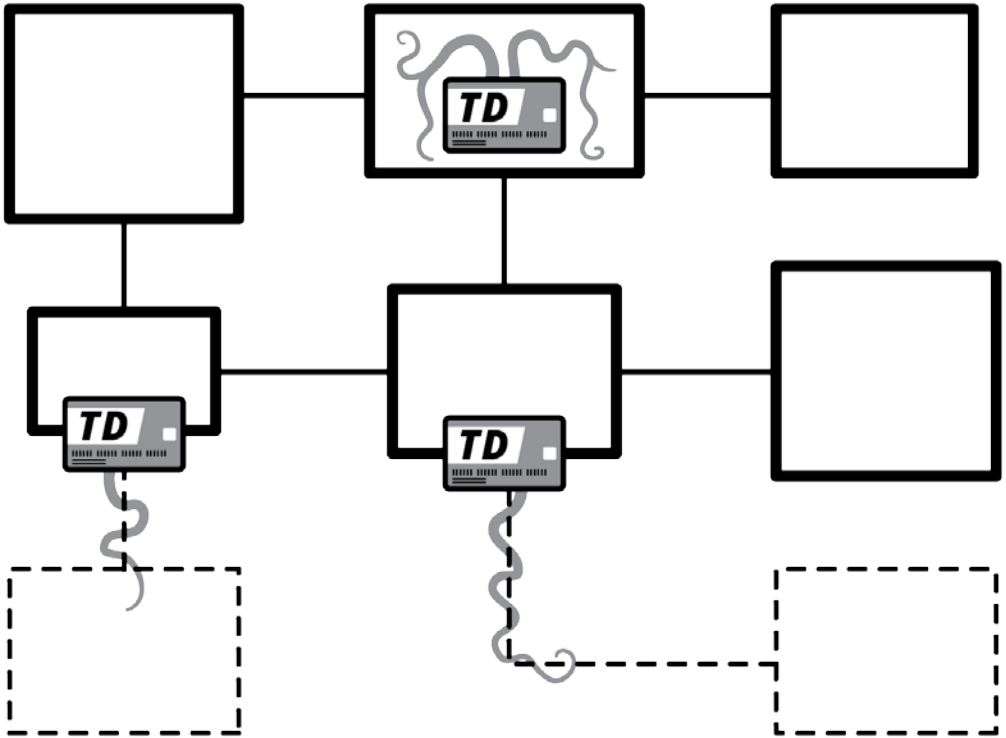
Is traced to **several locations** in the system, implying issues are not isolated but propagate throughout the system artifacts;

Has a **quantifiable and increasing** effect on system attributes (e.g., increasing defects, negative change in code quality).

Common Consequences of Technical Debt

- Teams spend almost all of their time fixing defects, and new capability development is continuously slipping.
- Integration of products built by different teams reveals that incompatibilities cause many failure conditions and lead to significant out-of-cycle rework.
- Progress toward milestones is unsatisfactory because unexpected rework causes cost overruns and project-completion delays.
- Recurring user complaints about features that appear to be fixed.
- Out-dated technology and platforms require lengthy convoluted solutions and added complexity in maintaining or extending the systems.

Principle: All systems have technical debt



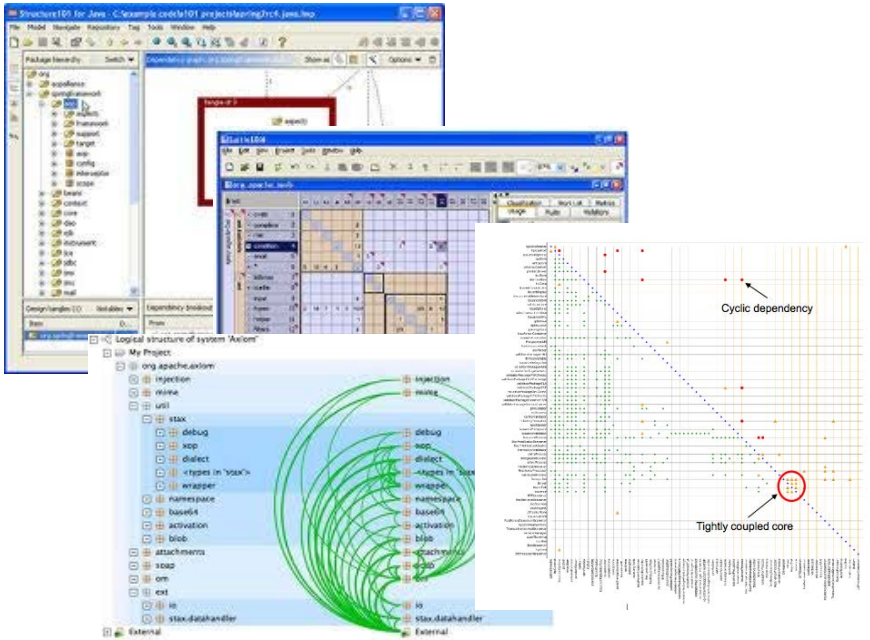
Assess the information

Toolbox: Assess the Information

Understand the state of the project, what debt you are facing, what causes it, and what are the consequences.

- Understand the business context to guide the use of analysis tools
- Create coding, architecture, and production infrastructure standards
- Organize small brainstorming sessions around the question:
  what design decision did we make that it is costing us so much?

Take Advantage of Tool Support



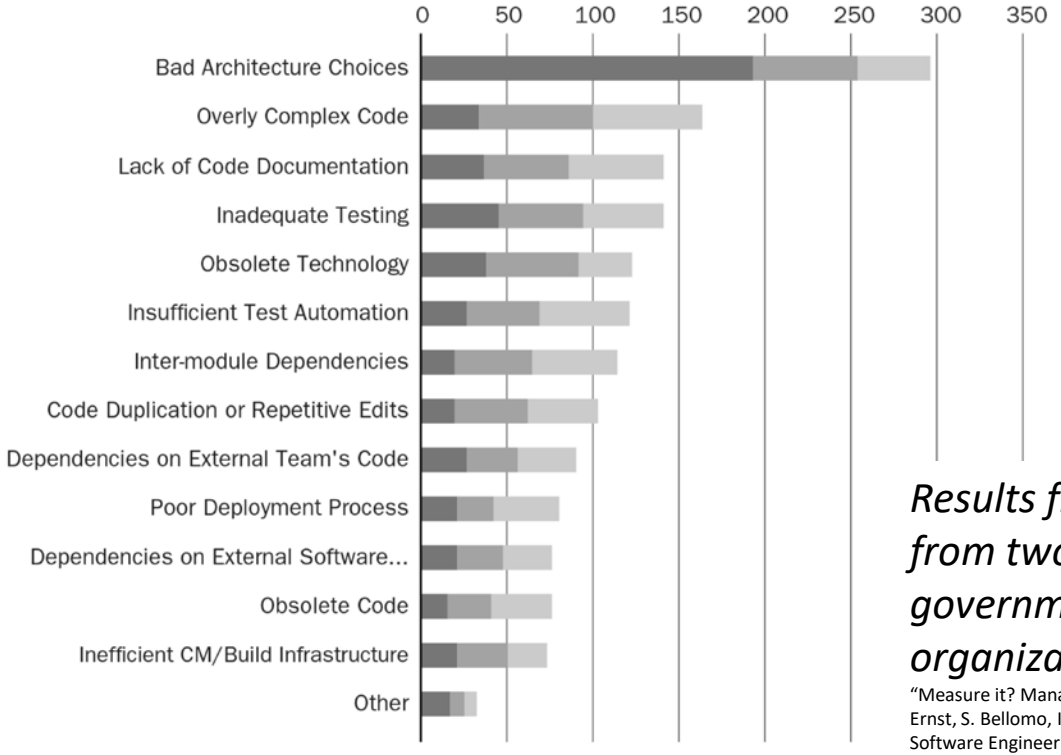
Tools can help assess aspects of software complexity and structural quality.

This is only a starting point!

Information from these tools needs to be coupled with an understanding of:

- Number of defects and their locations
- Areas where systems change a lot
- Areas developers avoid
- Architecture decisions
- Risk liability
- ….

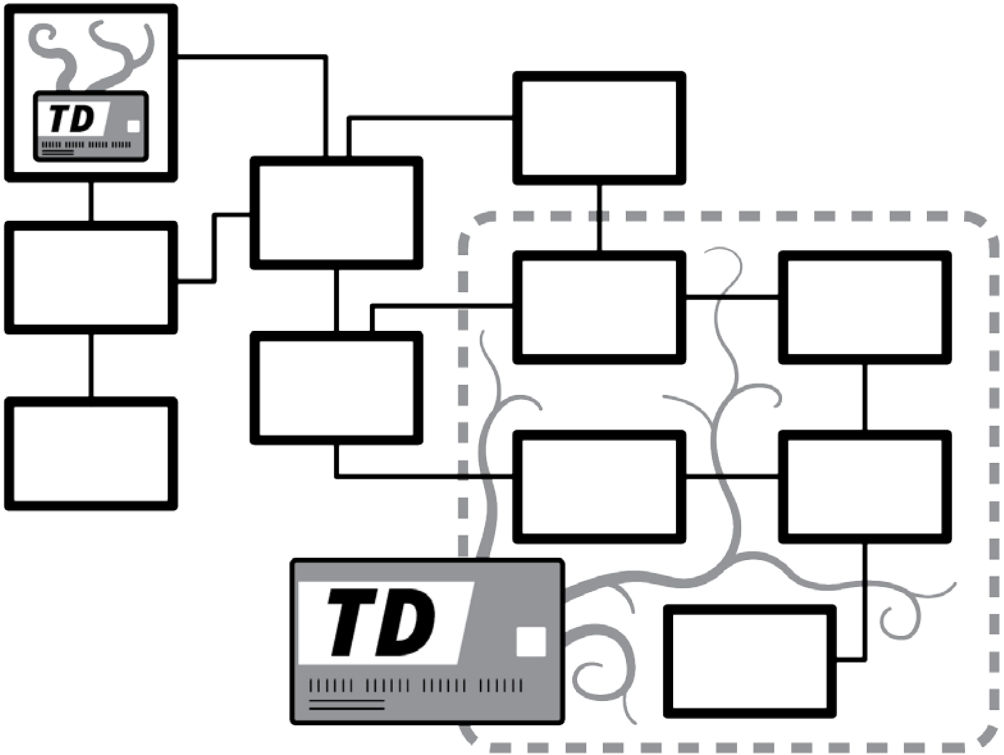
Software Architecture and Design Trade-offs Matter



Results from over 1800 developers from two large industry and one government software development organization.

"Measure it? Manage it? Ignore it? Software Practitioners and Technical Debt" N. Ernst, S. Bellomo, I. Ozkaya, R. Nord, I. Gorton, Int. Symp on Foundations of Software Engineering 2015.

Principle: Architecture debt has the highest cost of ownership



Build a registry

Toolbox: Build a Registry

Build some form of inventory of technical debt.

- Refine the technical debt category into a technical debt description
- Prioritize technical debt as part of your backlog
- Record decisions to intentionally incur debt

Incorporate Tracking into Existing Practices

Deployment & Build	Out-of-sync build dependencies
	Version conflict
	Dead code in build scripts
Code Structure	Event handling
	API/Interfaces
	Unreliable output or behavior
	Type conformance issue
	UI design
	Throttling
	Dead code
	Large file processing or rendering
	Memory limitation
	Poor error handling
	Performance appending nodes
	Encapsulation
	Caching issues
Data Model	Data integrity
	Data persistence
	Duplicate data
Regression Tests	Test execution
	Overly complex tests

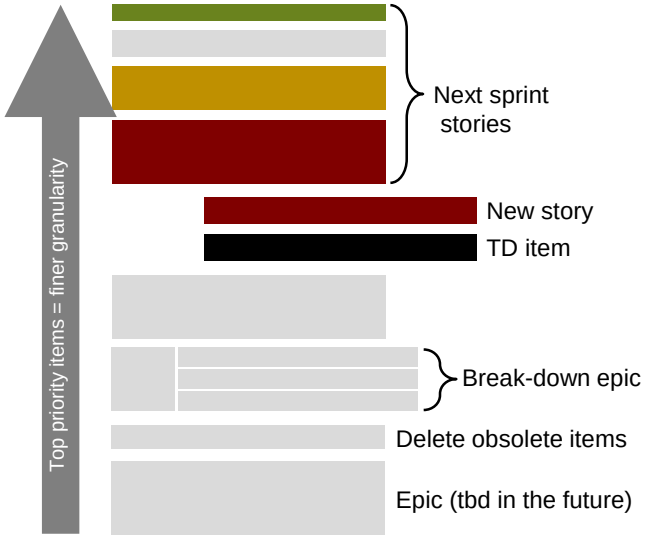
Record technical debt similar to user stories, defects, vulnerabilities, and the like.

Start with a simple *issue type* labeled *technical debt*. This practice pretty quickly helps recognize specific aspects of your technical debt.

Scout for project management and technical review practices that can be revised to include discussing and recording technical debt, augmenting technical debt issues with its consequences if not resolved.

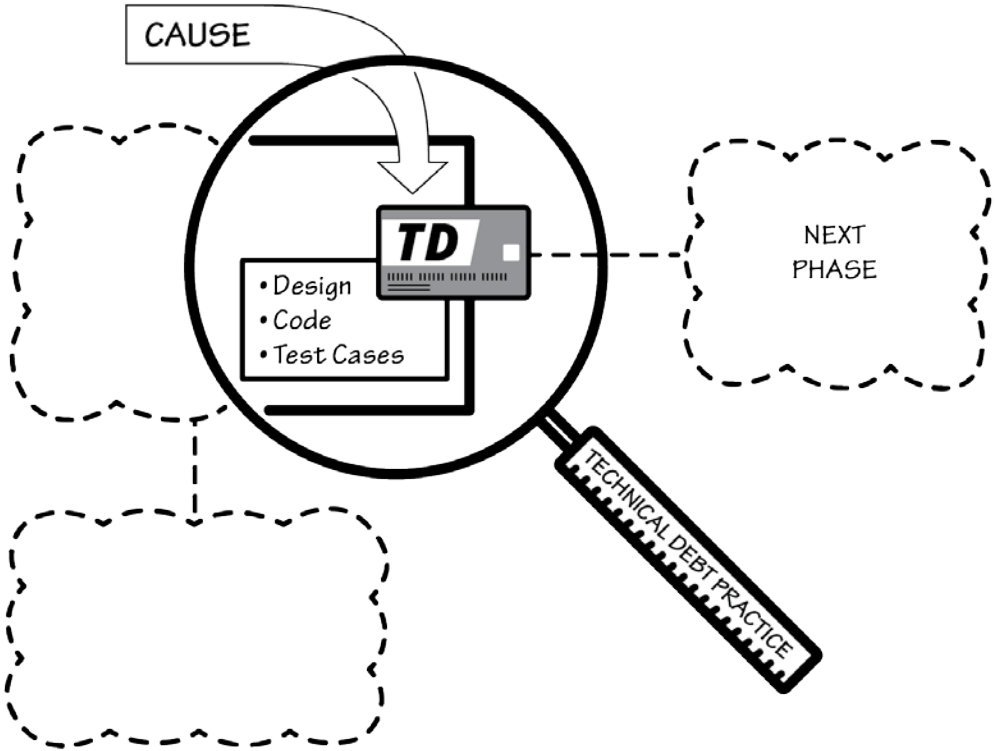
Stephany Bellomo, Robert L. Nord, Ipek Ozkaya, Mary Popeck: Got technical debt?: surfacing elusive technical debt in issue trackers. MSR 2016: 327-338

Describe Technical Debt Items



Name	Connect #Gateway-1631: Remove empty Java packages
Summary	The re-architecture of the source code to support multiple adaptor specifications has introduced a new Java packaging scheme. Numerous empty Java package folders across multiple projects.
Consequences	No impact to functionality; however, may lead to confusion for users implementing enhancements or modifications to the source code.
Remediation approach	New and existing classes have been moved into these new package folders; however, the previous package folders have been left in place with no class files.
Reporter / assignee	Gateway developers

Principle: Technical debt must trace to the system



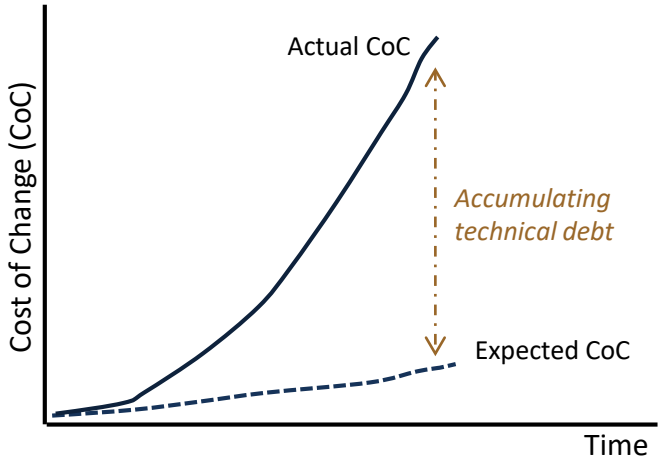
Decide what to fix

Toolbox: Decide What to Fix

Look over the registry as you plan the release for items you will actually tackle and reduce your technical debt.

- Estimate the cost to pay and the cost not to pay
- Budget for incremental debt reduction with most code-level refactoring
- Plan for more significant system-wide refactoring that spreads across iterations with major structural debt reduction

Understand the Cost of Accepting Technical Debt



For each instance of technical debt

- Understand range of consequences
- Measure what you can
- Qualitatively assess what you can't
- Reconcile data with assessments

Make informed trade-off decisions about remediation.

Develop a Payback Strategy Balancing Value for Cost

Do nothing

- There is benefit in carrying the debt and deferring payment
- The cost of reducing the debt far exceeds the benefit
- Business decides to abandon the system to optimize value

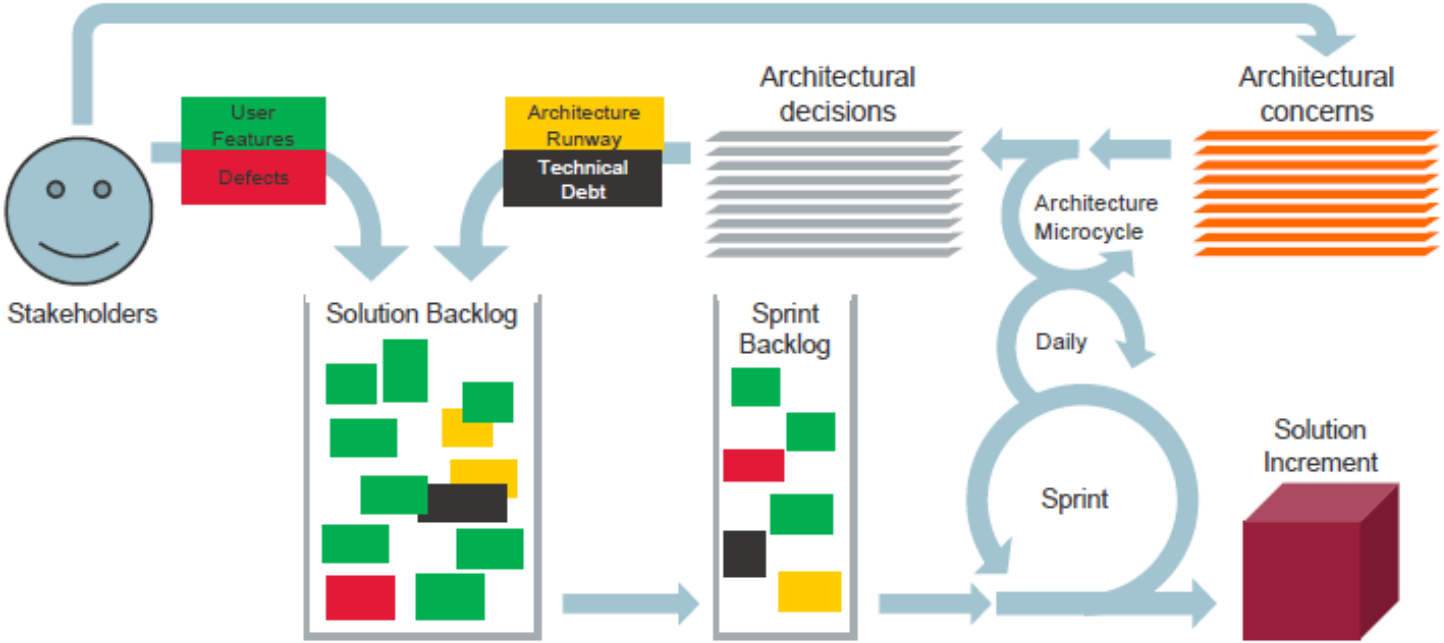
Repay

- Break the build; stop the release train
- Focused refactoring release
- Replace when the cost of reducing the debt far exceeds the benefit

Commit to invest

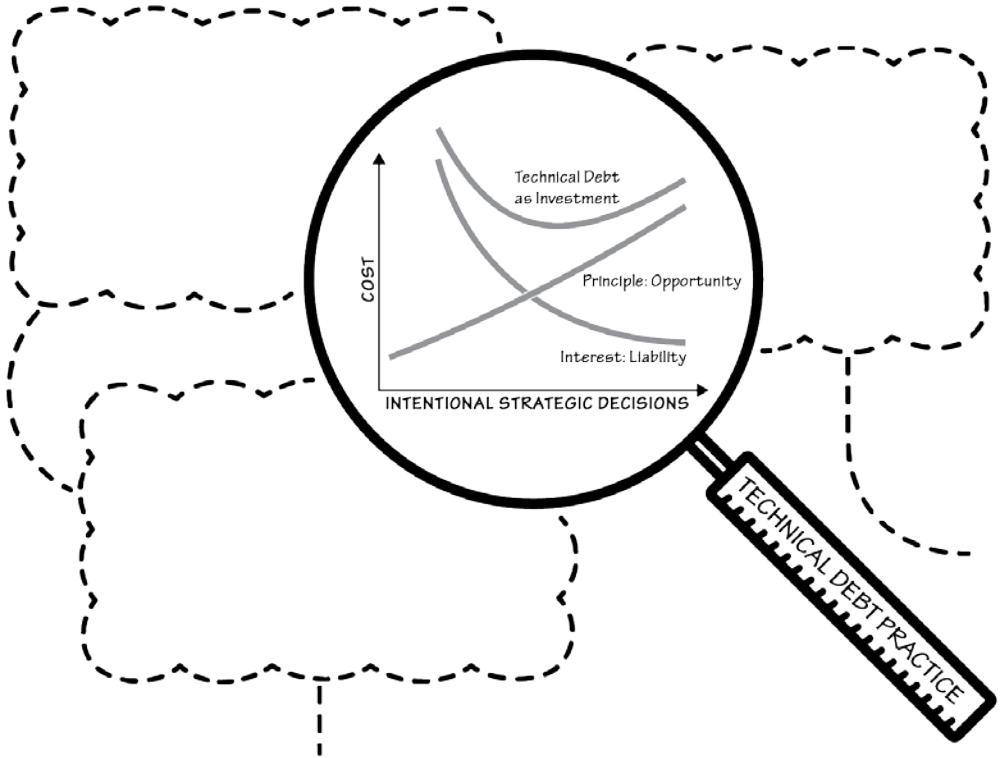
- Incremental refactoring over multiple releases
- Balance new feature development with refactoring effort to continue to generate value
- Differentiate strategic technical debt from debt that emerges from low code quality

Include Technical Debt Management in Release Planning



Poort, E. Selling the Business Case for Architectural Debt Reduction, *Ninth International Workshop on Managing Technical Debt* – XP 2017

Principle: Technical debt is not synonymous with bad quality



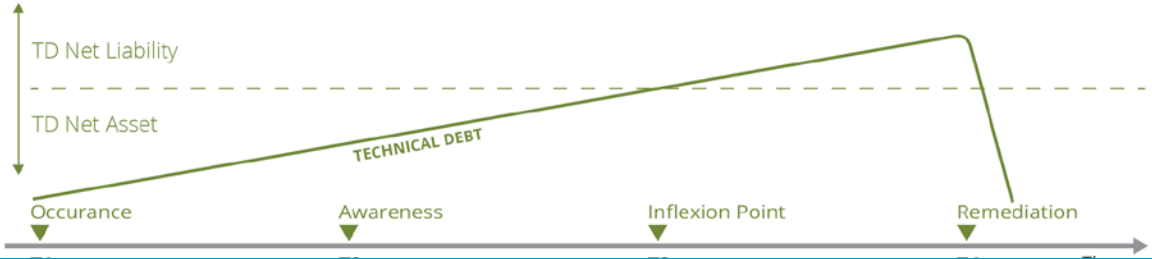
Take action

Toolbox: Take Action

Include technical debt identification and management in all software development and business governance practices.

- Aim to reduce debt at each development cycle
- Factor technical debt into business decisions about the opportunity cost of delaying features and reducing risk liability
- Gather key measures of effort or cost to assist in future decision making

Recognize the Technical Debt Timeline



"[Contractor] developed our software tool and delivered the code to the government for maintenance. The code was poorly designed and documented therefore there was a very long learning curve to make quality changes. We continue to band aide over 1 million lines of code under the maintenance contract. As time goes by, the tool becomes more bloated and harder to repair."

Management practices, technical contexts, and business contexts all affect the timeline

- Who is responsible at each point
- Amount of time that passes between points
- Available options

Manage Technical Debt within Acquisition Lifecycle

Include language on how technical debt will be managed in contracts

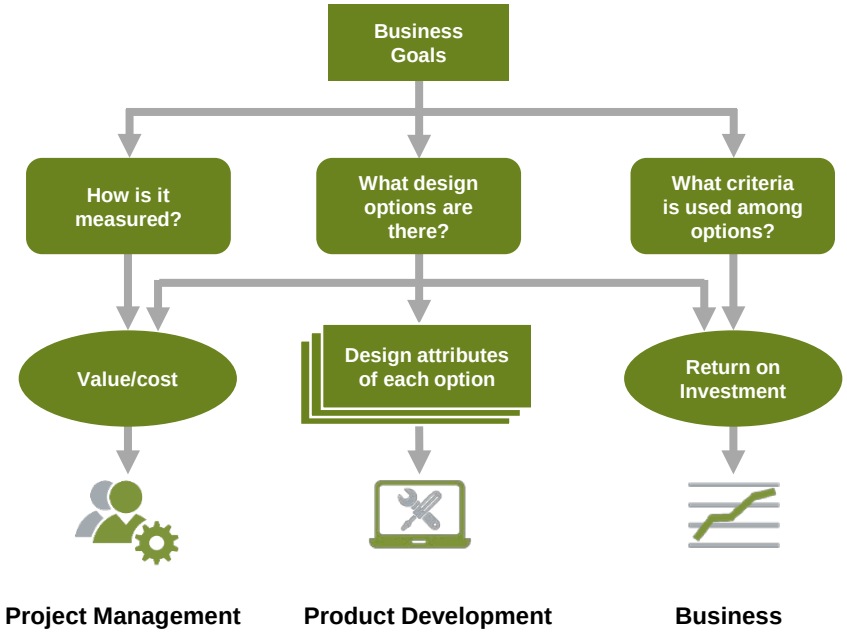
- Percentage of resources to be withheld until high priority technical debt is resolved
- Data to be shared throughout the development life cycle
- Ongoing analysis to be conducted and its results shared
- Incentives to share technical debt the contractor takes on

Include technical debt discussions as part of assessments;
request use of appropriate software quality tools and architecture reviews.

Continuously assess where you are on the technical debt timeline

- Request evidence from contractors
- Helpful data includes commit histories, defect logs, testing results, architecture conformance measures, and software quality analyses

Technical Debt Management as a Catalyzer



Getting started

- Get smarter about technical debt
- Assess technical debt on a project
- Establish metrics for tracking
- Plan for resolving selected technical debt
- Establish or revise practices that fit with your software development life cycle

Contact Information

Robert Nord
Architecture Practices Initiative
Email: rn@sei.cmu.edu

U.S. Mail
Software Engineering Institute
Customer Relations
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
USA

Web
www.sei.cmu.edu
www.sei.cmu.edu/architecture
www.sei.cmu.edu/research-capabilities/all-work/display.cfm?customel_datapageid_4050=6520
www.techdebtconf.org

Customer Relations
Email: info@sei.cmu.edu
SEI Phone: +1 412-268-5800
SEI Fax: +1 412-268-6257

Further Reading

Avgeriou, P., Kruchten, P., Nord, R.L., Ozkaya, I., and Seaman, C.B. "Reducing Friction in Software Development." IEEE Software 33, 1 (Jan./Feb. 2016): 66-73.

Bachmann, F., Nord, R.L., Ozkaya, I. 2012. Architectural Tactics to Support Rapid and Agile Stability. CrossTalk: The Journal of Defense Software Engineering, Special Issue on Rapid and Agile Stability, May/June 2012.

Bellomo, S., Kruchten, P., Nord, R.L., Ozkaya, I. How to Agilely Architect an Agile Architecture. Cutter IT Journal, 27(2), February 2014.

Brown, N., Kruchten, P., Lim, E., Nord, R., Ozkaya, I. June 2017. Hard Choices Game Explained. Downloadable from http://www.sei.cmu.edu/architecture/tools/hardchoices/

Ernst, N.A., Bellomo, S., Ozkaya, I., Nord, R.L., and Gorton, I. "Measure It? Manage It? Ignore It? Software Practitioners and Technical Debt." Conference on the Foundations of Software Engineering, Aug. 30-Sep. 4, 2015, ACM.

Kruchten, P. Nord, R.L., Ozkaya, I. 2012. Technical Debt: From Metaphor to Theory and Practice, IEEE Software, 29(6), Nov/Dec 2012.

Kruchten, P., Nord, R.L., Ozkaya, I., Falessi, D. Technical Debt: Towards a Crisper Definition. Report on the 4th International Workshop on Managing Technical Debt, ACM Sigsoft Software Engineering Notes, Sept 2013.

Ozkaya, I., Gagliardi. M., and Nord, R.L. 2013. Architecting for Large Scale Agile Software Development: A Risk-Driven Approach, Crosstalk 26, 3 (May/June 2013): 17-22.