Managing Technical Debt and Software Architecture

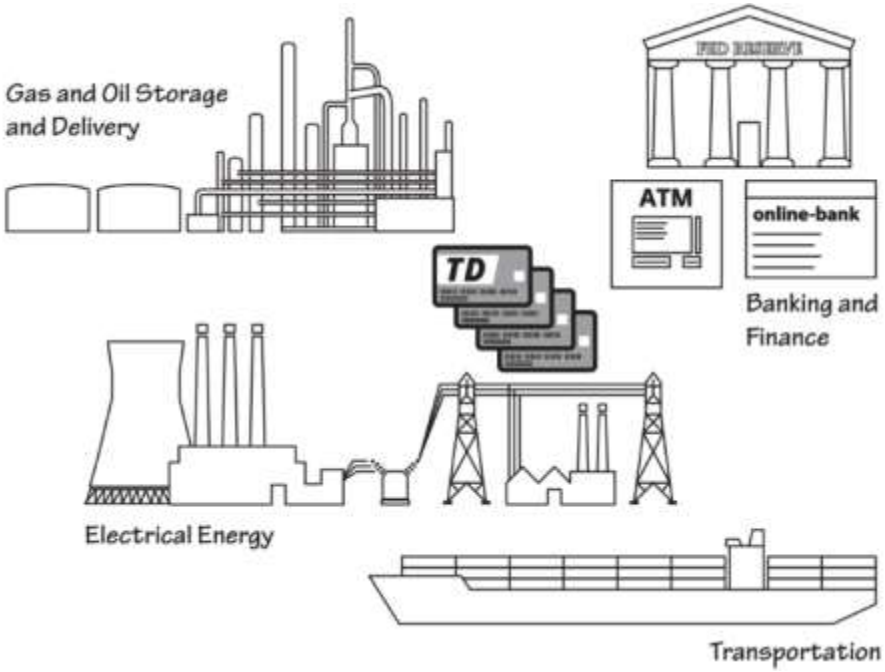
Ipek Ozkaya

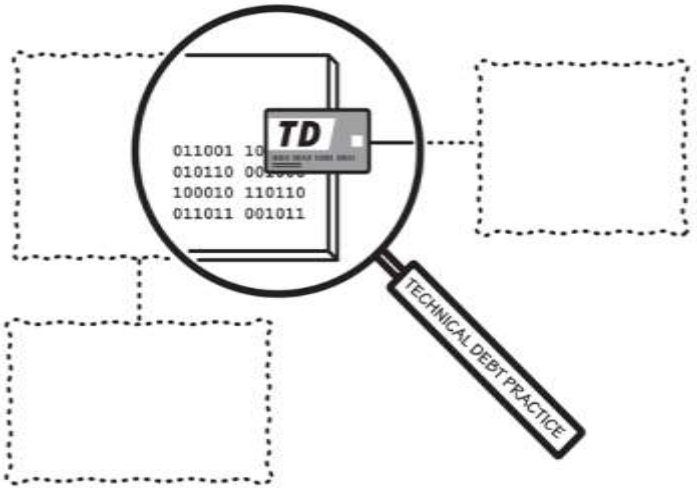
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

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.

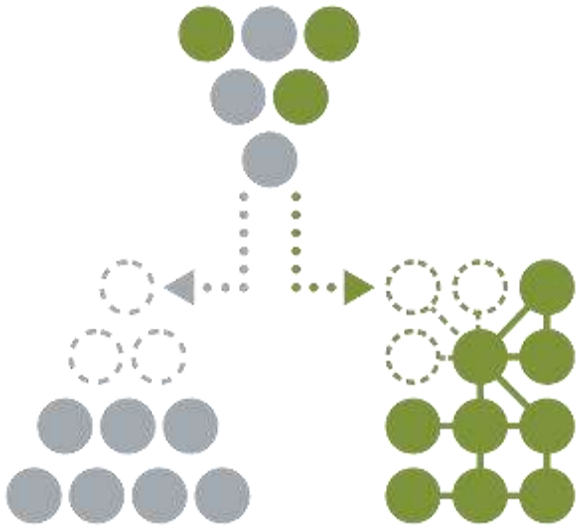
DM21-0939

ALL SYSTEMS HAVE TECHNICAL DEBT!





Technical Debt: A Definition



In software-intensive systems, technical debt consists of design or implementation constructs that are expedient in the short term but set up a technical context that can make future changes more costly or impossible.

Technical debt presents an actual or contingent liability that impacts internal system qualities.

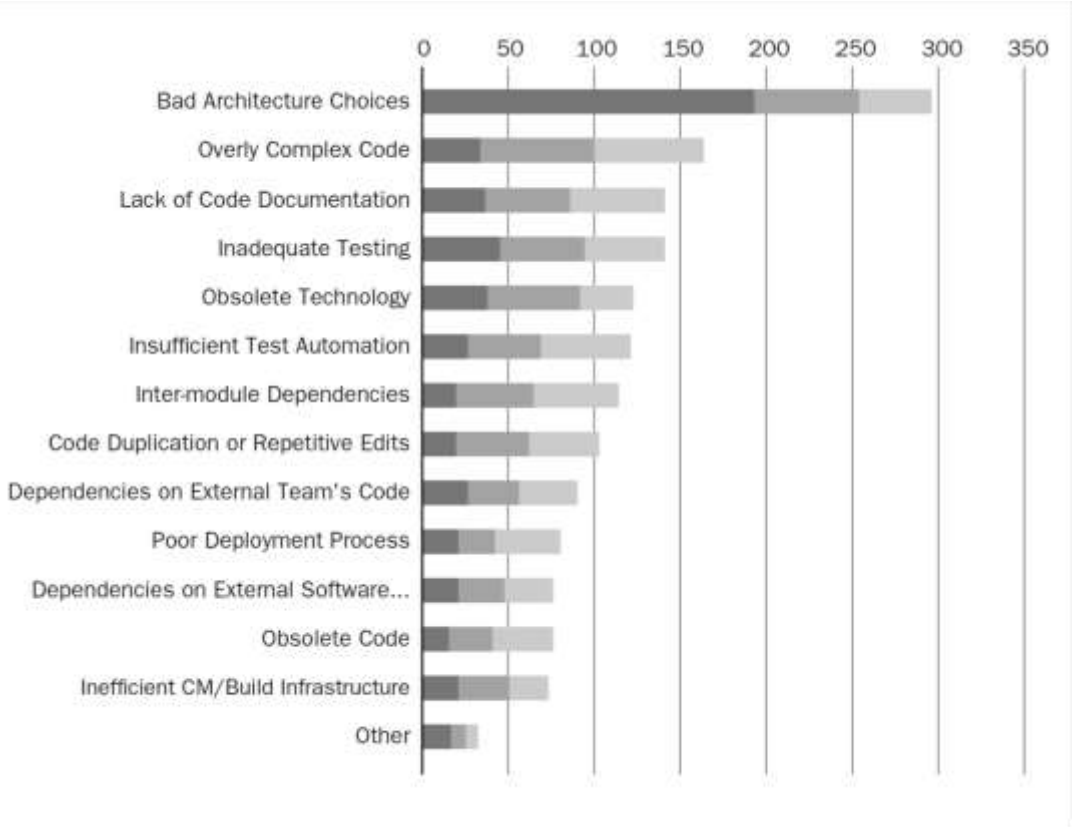
A Typical Technical Debt Example

A decade ago processors were not as powerful. To optimize for performance we would not insert code for exception handling when we knew we would not divide by zero or hit an out of bounds memory condition. These areas now are hard to track and have become security nightmares.

Technical debt is a **software design issue** that:

- Exists in an executable system artifact, such as code, build scripts, data model, 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 maintainability and code quality indicators).

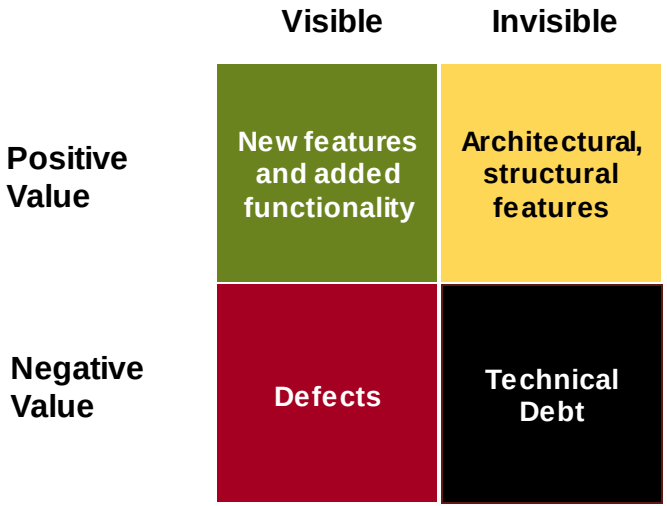
Software Architecture and Design Trade-offs Matter



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

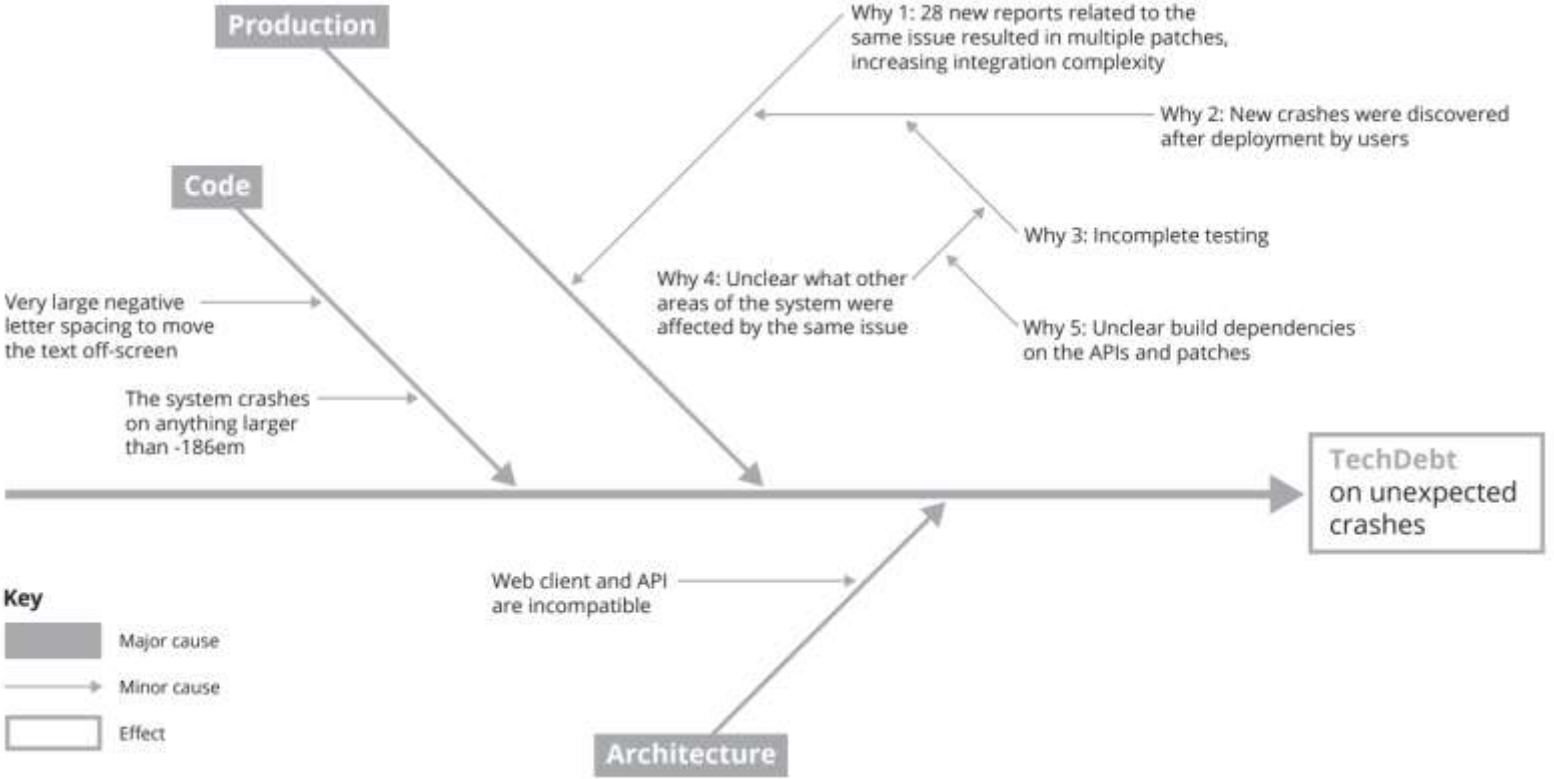
"Measure it? Manage it? Ignore it? Software Practitioners and Technical Debt" N. Ernst, S. Bellomo, I. Ozkaya, R. Nord, I. Gorton, Int. Symp on Foundations of Software Engineering 2015.

Essential Elements of Software Development



Kruchten, P. Nord, R.L., Ozkaya, I. 2019. Managing Technical Debt Reducing Friction in Software Development, Pearson Addison-Wesley.

An Example Technical Debt Causal Chain



Recognizing Technical Debt

Technical debt can be recognized directly or indirectly by

- Recording decisions to intentionally incur debt
- Conducting design and architecture reviews
- Analyzing development and management artifacts for symptoms
- Talking to development teams

Detecting Technical Debt

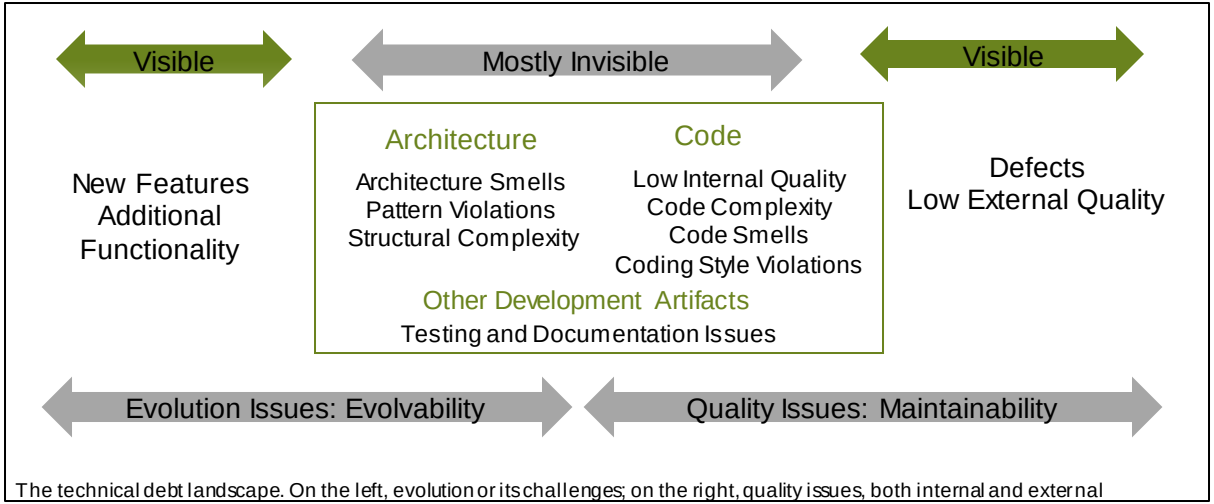
Detect technical debt from code, where code-level conformance and structural analysis indicate maintainability and concerns related to the structure of the system and the codebase.

Detect technical debt from symptoms that signal architecture issues.

Detect technical debt from architecture during design reviews and analysis of decisions.

Detect technical debt from development and deployment infrastructure, which are not typically part of the delivered system but may impact its delivery, security, and quality.

The Technical Debt Landscape



The technical debt landscape. On the left, evolution or its challenges; on the right, quality issues, both internal and external

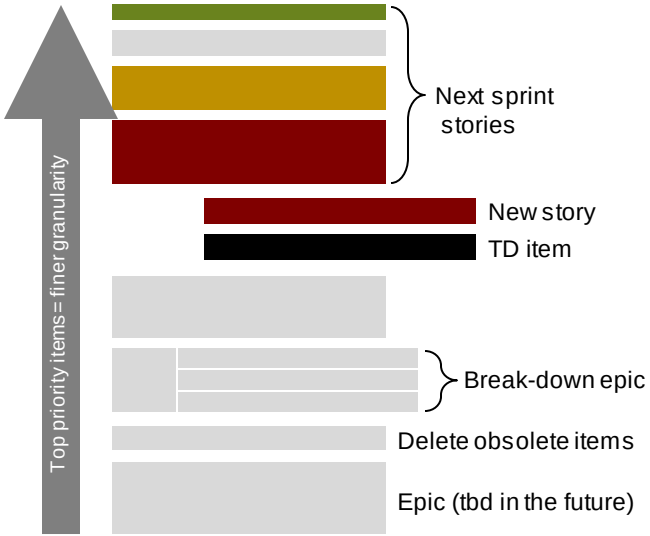
Kruchten, P. Nord, R.L., Ozkaya, I. 2012. Technical Debt: From Metaphor to Theory and Practice. IEEE Software, 29(6), Nov/Dec 2012.

Making Technical Debt Visible

Making technical debt visible implies communicating and tracking technical debt

- Timely
- Concretely identifying what and where
- Including experienced and potential consequences
- Involving all relevant stakeholders

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

Incorporate Tracking into Existing Practices

Incentivize developers and acquisition organizations to disclose technical debt when they recognize it through simple practices.

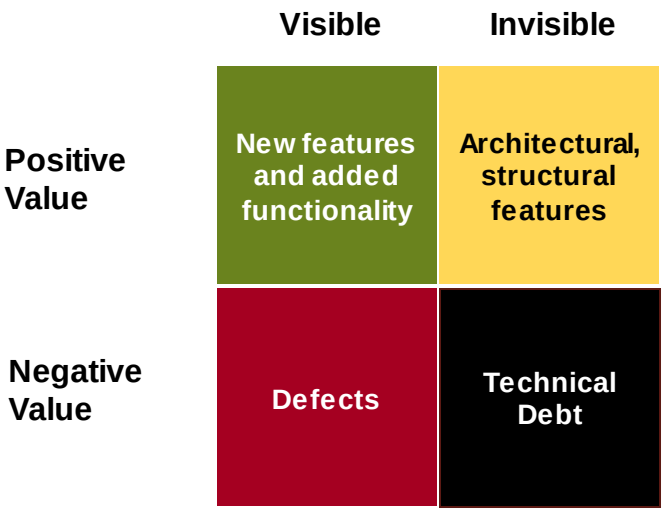
Start with a simple *issue type* labelled *technical debt*. This practice pretty quickly helps recognize specific aspects of your technical debt.

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

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

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

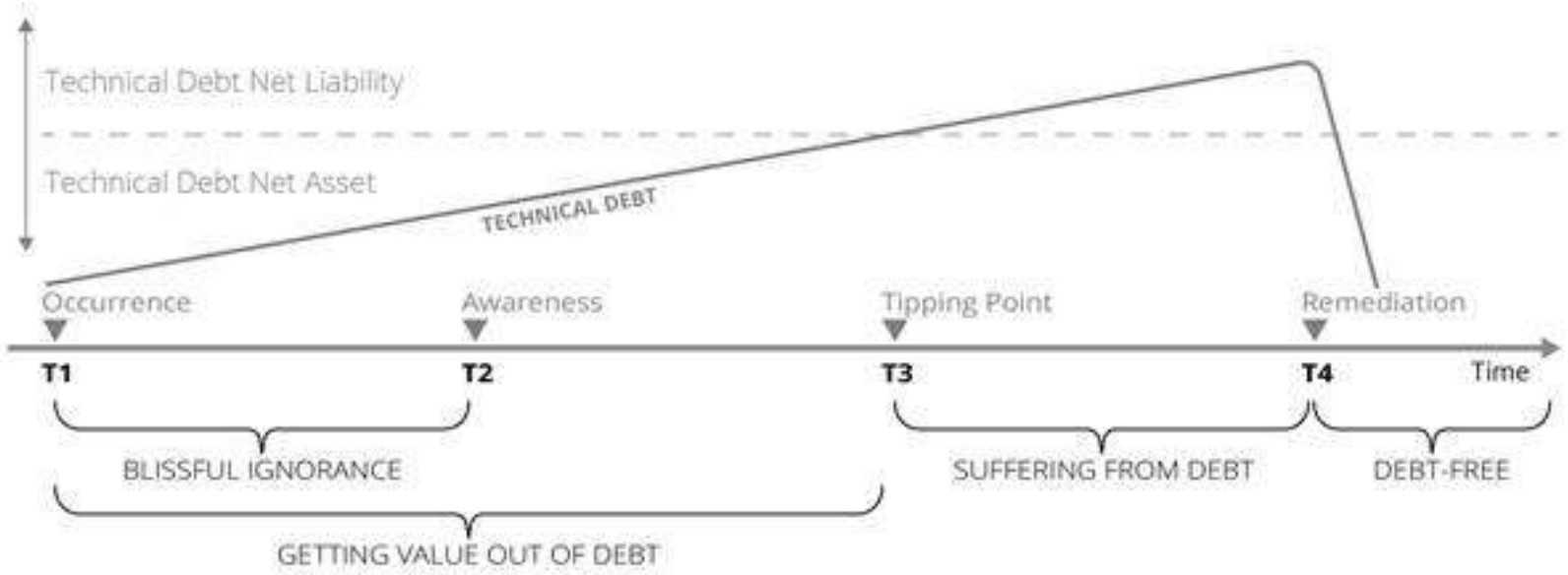
Living with Technical Debt



All long-lived, large scale system have technical debt!

- Allocate time for managing technical debt at every iteration.
- Invest in a sound development and testing infrastructure that includes automated quality measurement.
- Differentiate strategic technical debt from technical debt that emerges from low code quality or poor engineering practices.

Manage the Technical Debt Timeline



Discussion