Managing Technical Debt throughout the Software Development Lifecycle

Ipek Ozkaya

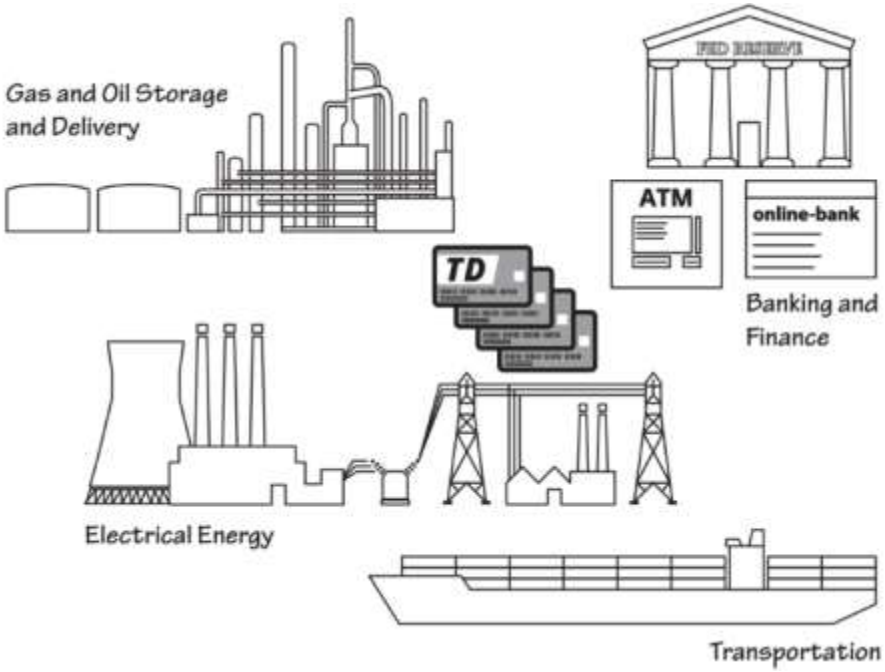
GBSD Training

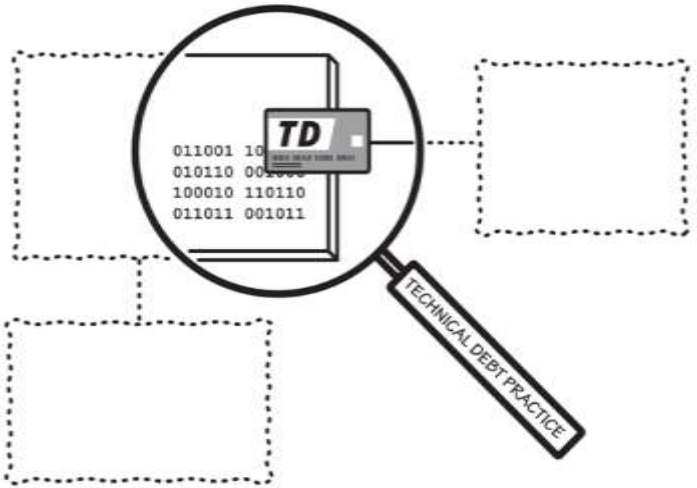
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

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.

DM22-0178

ALL SYSTEMS HAVE TECHNICAL DEBT!





Goals of the Training

What is technical debt?

- Its definition and relationship to defects and vulnerabilities
- Misconceptions of what technical debt is

Context matters!

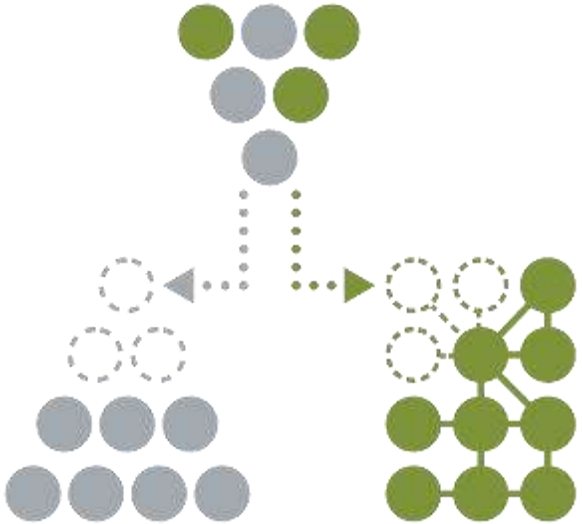
Recognizing technical debt

- Understanding causes of technical debt
- Detecting technical debt

Making technical debt visible

Technical debt within the acquisition lifecycle

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 prohibitively costly.

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

Kruchten, P.; Nord, R.L.; & Ozkaya, I. *Managing Technical Debt: Reducing Friction in Software Development.* Pearson Addison-Wesley. 2019. ISBN-10: 013564593X.

Common Consequences of Technical Debt

- 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 and other systematic issues make the system less usable.
- Outdated technology and platforms require lengthy convoluted solutions and added complexity in maintaining or extending the systems.

Technical Debt is Common

Unmanaged technical debt costs organizations time and money!

- Government's old technology deficit that needs to be replaced is estimated to be up to $7.5 billion in 2016 (https://federalnewsradio.com/reporters-notebook-jason-miller/2016/09/39000-shows-modernization-effort-matters-much/)
- U.S. Department of Veterans Affairs spending 75% of its technology budget to maintain outdated legacy systems (https://techcrunch.com/2017/06/19/how-the-presidents-american-tech-council-should-tackle-reforming-government-tech/)
- Major government programs often in the news for lack of their technical debt management (https://www.military.com/daily-news/2016/02/17/f35-deficiencies-decreasing-hundreds-remain-program-manager.html)
- High-profile industry failures are often associated with technical debt (for example, United Airlines network connectivity failure and New York Stock Exchange glitches of July 8, 2015: https://venturebeat.com/2015/07/09/3-takeaways-from-3-big-tech-outages-nyse-united-and-wsj/)

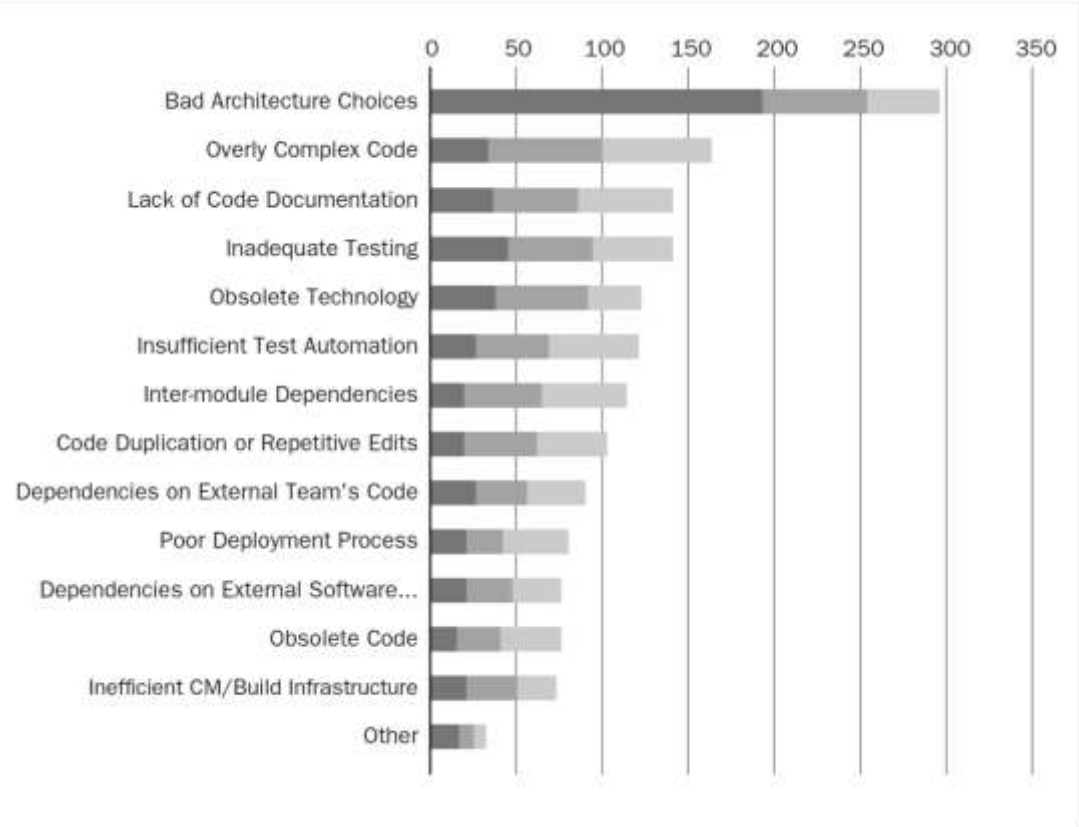
A Typical Technical Debt Example

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 decision** made to solve a problem but may not stand the test of time and will cause rework. Technical debt, therefore

- Exists in an executable system artifact, such as code, build scripts, data model, automated test suites
- Often 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)

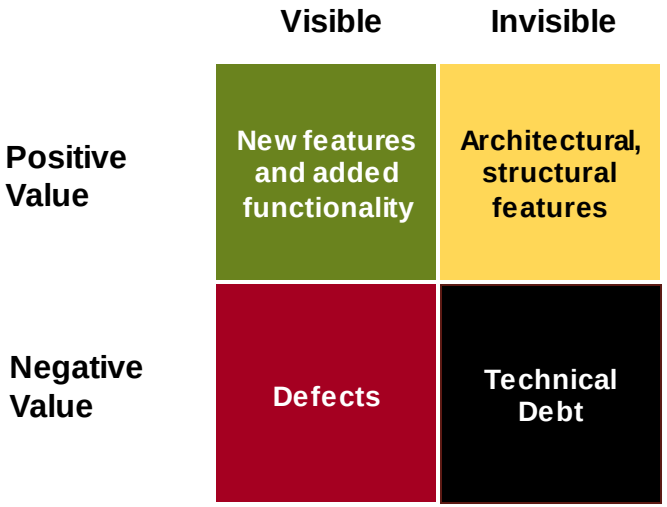
Software Architecture and Design Tradeoffs Matter



Results from over 1800 developers from two large industry and one government software development organizations reinforce that unattended architecture decisions and practices are at the root of technical debt.

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

Essential Elements of Software Development



Kruchten, P.; Nord, R.L.; & Ozkaya, I. *Managing Technical Debt: Reducing Friction in Software Development.* Pearson Addison-Wesley. 2019. ISBN-10: 013564593X.

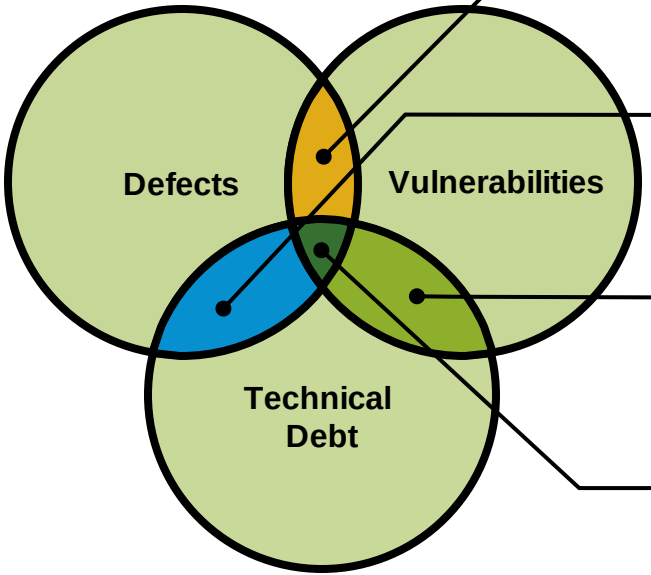
The Need to Identity Technical Debt

Defect – error in coding or logic that causes a program to malfunction or to produce incorrect and/or unexpected results

Vulnerability – system weakness in the intersection of three elements:

- system flaw
- attacker access to the flaw
- attacker capability to exploit the flaw

Technical debt – design or implementation construct traced to several locations in the system, that make future changes more costly



Defect proneness implies increased vulnerability risks.

Technical debt, as it lingers in the system, increases defect proneness.

Technical debt increases vulnerability risks.

Some issues just overlap, making them hard to tease apart!

Common Misconceptions

Technical debt is not simply bad quality.

Lack of process is not technical debt; rather, it results in technical debt.

New features not yet implemented are not technical debt.

Everything not yet done is not technical debt.

Context Matters

Unmanaged Technical Debt Puts Programs at Risk

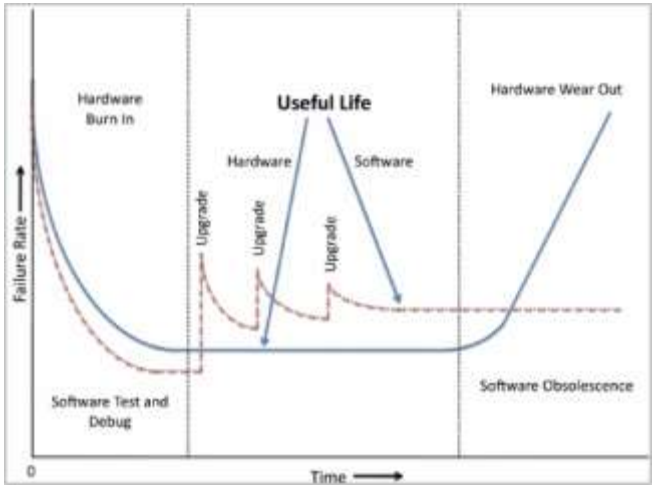
Example critical quality attributes for mission critical systems:

Safety – Architectural Design must address Hazards: Prevention, Detection, Isolation & Recovery (PDI&R), plus Operational Transparency w/Diagnostic Visibility (OTwDV).

Security – Architectural Design must address Cybersecurity Threats & Hazards (PDI&R, OTwDV).

Reliability – Architectural Design must minimize disruption during upgrades (Sprints, Defect Resolution, Tech Insert &Tech Refresh); it must also provide Fault Tolerance including Fault DI&R and OTwDV.

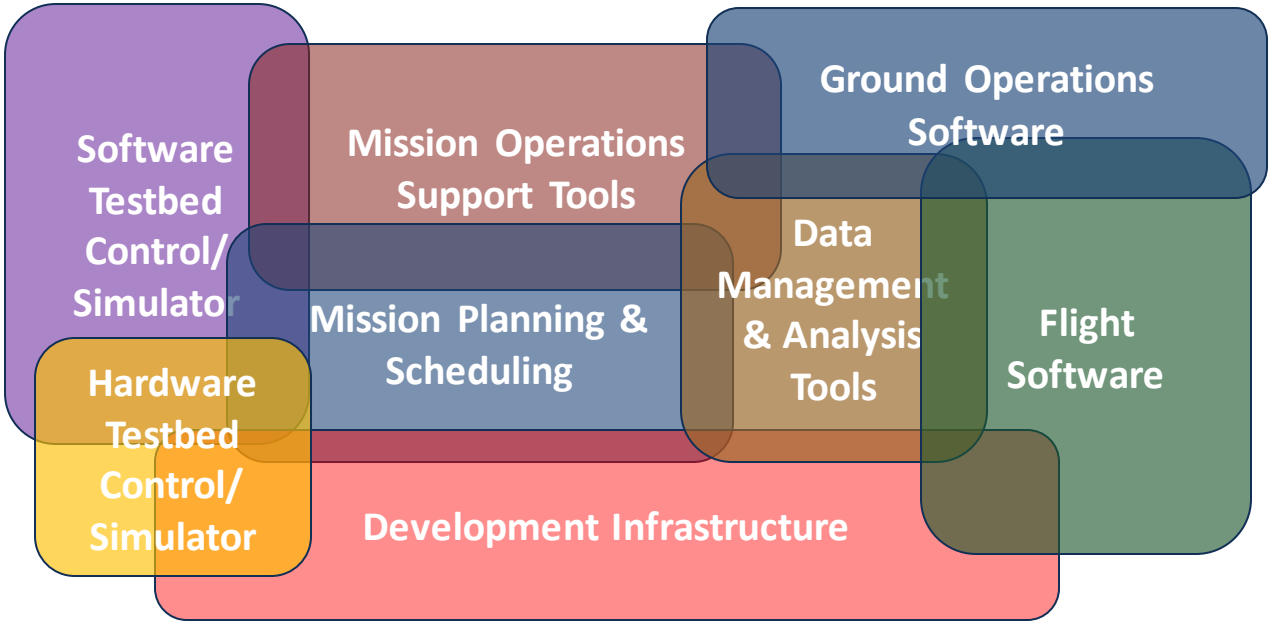
Certifiability – Architectural Design must address Certification Requirements (Certification Requirements Analysis, Requirements Decomposition and Traceability, System Testability under Certification Test Conditions, …) including General Testability.



All these attributes are mission critical, will be impacted by technology evolution, and have critical cost and assurance impact if the system needs to be reworked unintentionally due to technical debt.

All Software Matters!

Mission-critical tactical systems are composed of many intertwined hardware and software components! All of the software involved in these components create opportunities for technical debt to accumulate and propagate to the rest of the system!



Goals of the Training

What is technical debt?

- Its definition and relationship to defects and vulnerabilities
- Understanding misconceptions of what technical debt is

Context matters!

Recognizing technical debt

- Understanding causes of technical debt
- Detecting technical debt

Making technical debt visible

Technical debt within the acquisition lifecycle

Separate What Causes Technical Debt from the Actual Debt

Common causes of technical debt

Techniques and approaches to eliminate the causes will be different from those to identify and remove technical debt. Understanding and eliminating causes help avoiding future technical debt.

Technical Debt Exposure Workshop and Analysis

Purpose: A systematic approach to navigate through the state of a software development project focusing on key areas including vision, architecture, development practices, and organization.

- This is a recommended first step to ensure causes and actual debt are understood and separated to ensure appropriate management and avoidance strategies are implemented in a timely manner.

Approach: Set of interviews and day-long multiple stakeholder focused meetings, supported by artifact review as needed. Ideally conducted before major milestones and as a program is going through major transitions.

Outcomes: A scorecard and supporting data which includes a list of potential and actual causes of technical debt, impact rating for each and a mapping to the timeline and approaches for identifying technical debt.

Sample scorecard

Business Vision	business goals	+
	success strategies	+
	resources	=
	customer communication	=
	consequences of business decisions	=
	feedback cycles	=
	...	...
Architecture	architecturally significant requirements	–
	architecture fitness	–
	architecture issues	=
	short-term and long-term architecture goals	–
	impact of technology change	=
	build, integration, test, and deployment alignment	–
	...	...
Development	development infrastructure	=
	quality assurance	=
	development tools	–
	done criteria	–
	code maintenance and evolution	–
	software development processes and practices	=
	...	
Organization	collaboration	=
	change management	+
	cost of delay and rework	=
	uncertainty	–
	development team resources	=
	new employee onboarding	+
	team communication	+
	...	...

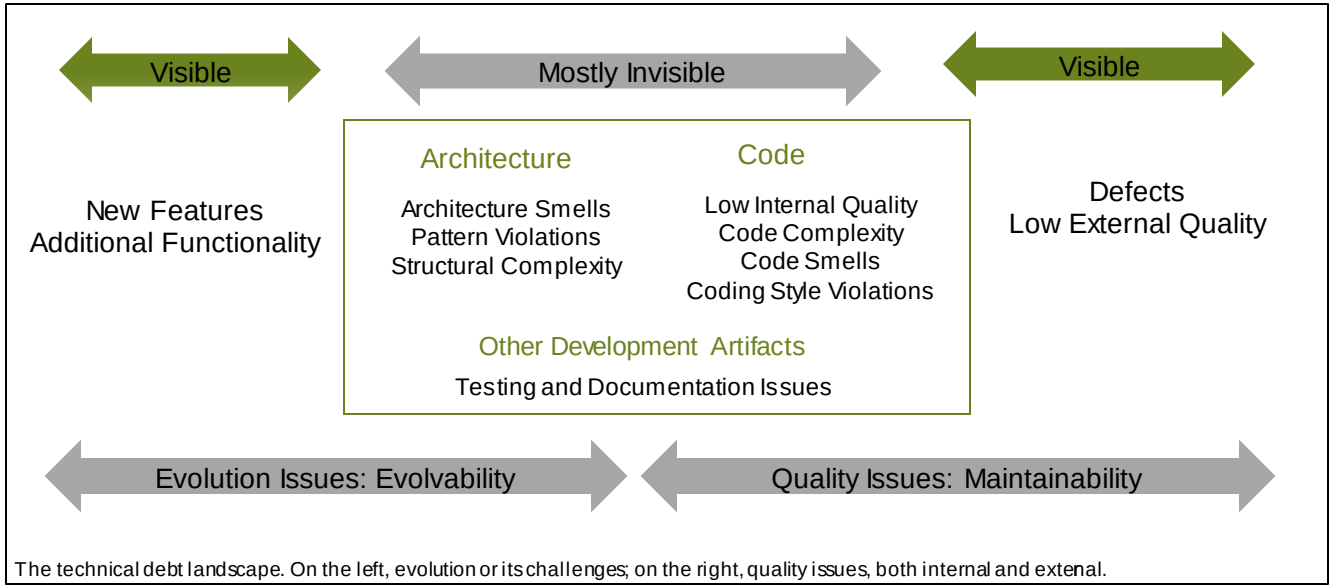
LEGEND	
+	*Issues are managed to minimize technical debt exposure*
=	*Can improve, can contribute to technical debt*
–	*Significant issues contributing to technical debt*

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

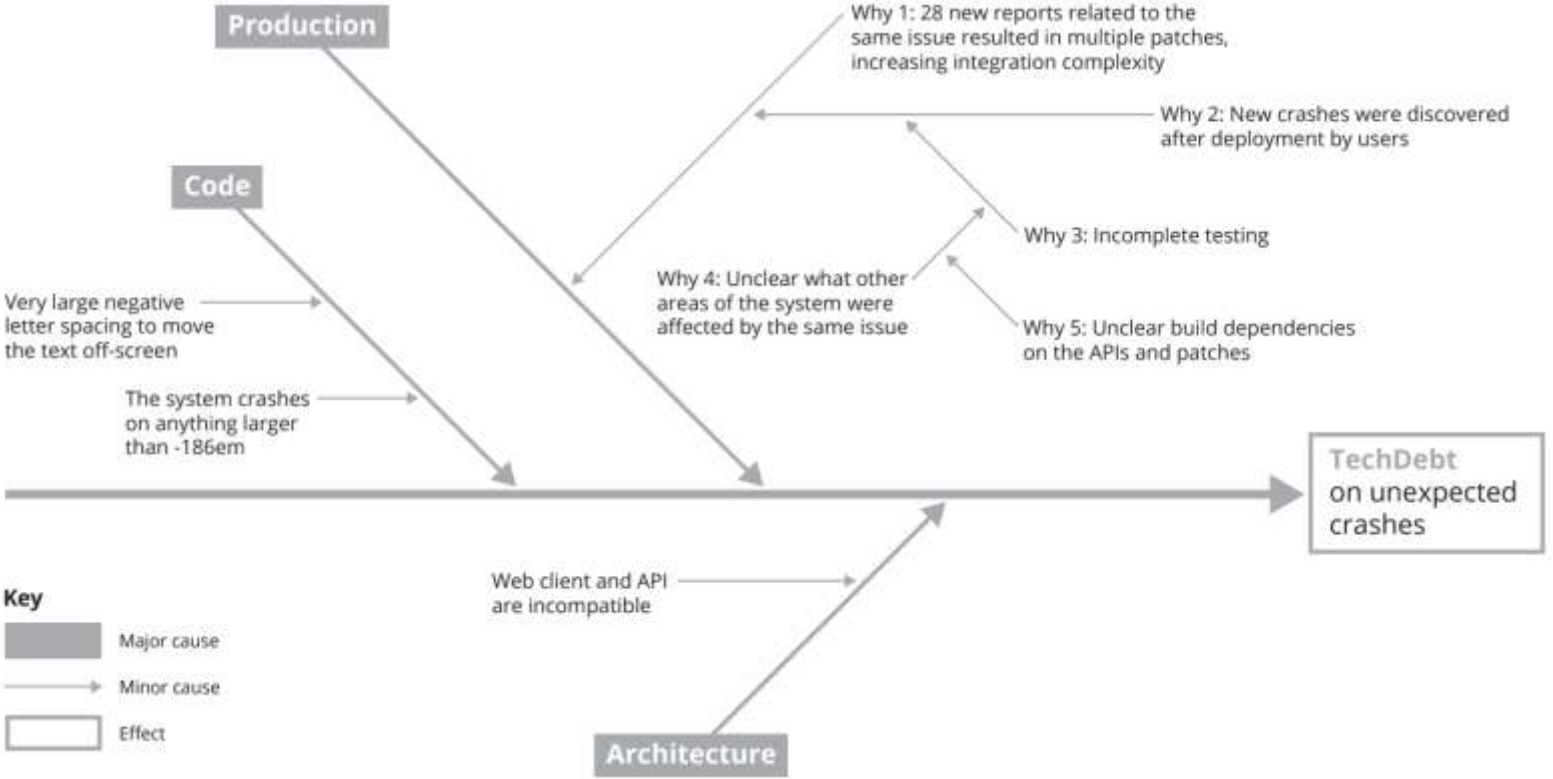
The Technical Debt Landscape



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*. Volume 29. Number 6. Nov/Dec 2012.

An Example Technical Debt Causal Chain



Detecting Technical Debt

1. *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
2. *Detect technical debt from symptoms* that signal architecture issues.
3. *Detect technical debt from architecture* during design reviews and analysis of decisions
4. *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

Examples of Technical Debt's Cybersecurity Impact, Robert Nord, Ipek Ozkaya, Carol Woody, SEI Technical Note for GBSD. July 2021.

Goals of the Training

What is technical debt?

- Its definition and relationship to defects and vulnerabilities
- Misconceptions of what technical debt is

Context matters!

Recognizing technical debt

- Understanding causes of technical debt
- Detecting technical debt

Making technical debt visible

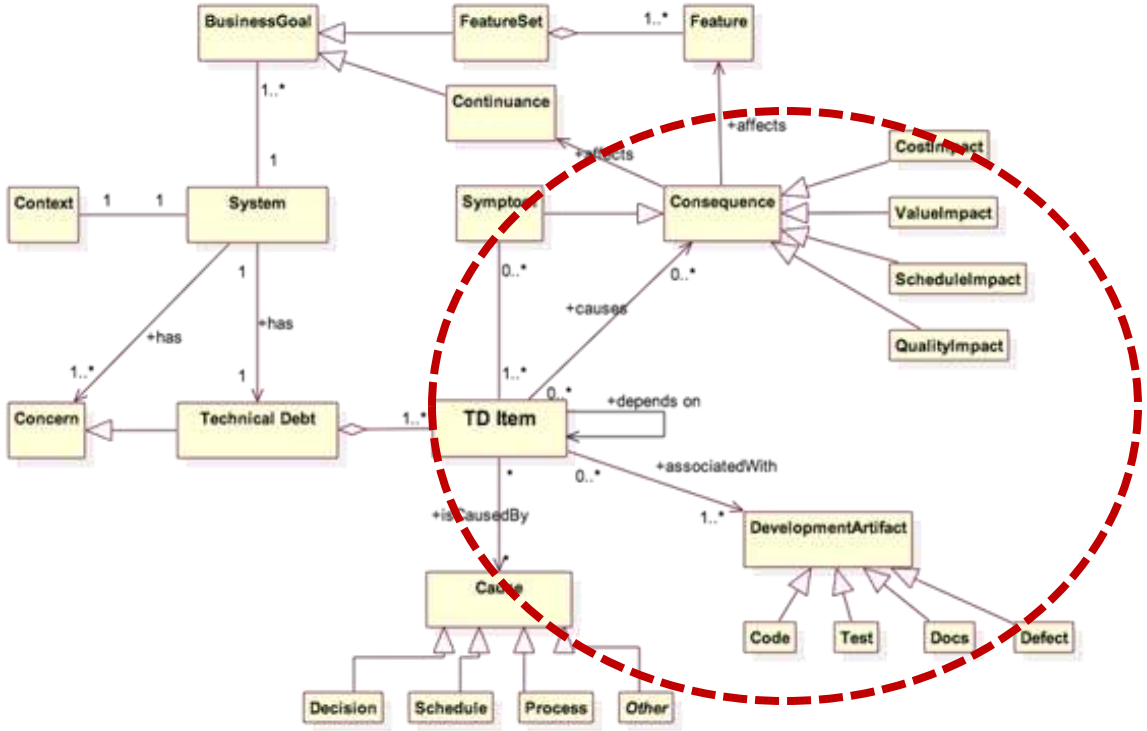
Technical debt within the acquisition lifecycle

Making Technical Debt Visible

Making technical debt visible implies communicating and tracking technical debt in a manner that

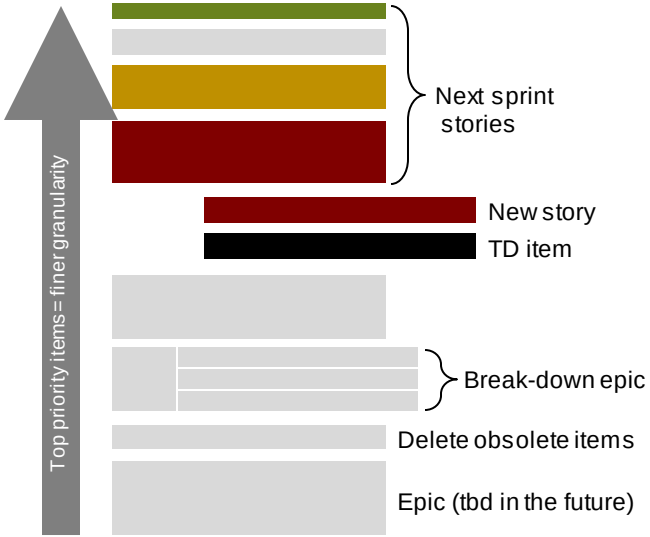
- Is timely
- Concretely identifies what and where
- Includes experienced and potential consequences
- Involves all relevant stakeholders

Technical Debt Items and Their Context



Kruchten, P.; Nord, R.L.; & Ozkaya, I. *Managing Technical Debt: Reducing Friction in Software Development.* Pearson Addison-Wesley. 2019. ISBN-10: 013564593X.

Technical Debt Examples – Detect from Code



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, re-architecture 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

Technical Debt Examples – Detect from Symptoms

Name	Unexpected crashes due to API incompatibility
Summary	The source code uses a very large negative letter-spacing in an attempt to move the text offscreen. The system handles up to -186 em fine, but crashes on anything larger. A similar issue was fixed with a patch, but there were several other similar reports. My sense is that if we patch it here, it will pop up somewhere else later.
Consequences	We already had 28 reports from seven clients. And it definitely leaves the software vulnerable. Finding the root cause can be time consuming given that existing patches did not resolve the issue.
Remediation approach	The external web client and our software likely has an API incompatibility, but further analysis is needed. The course of action is to verify where the root of this problem is and see if we can fix it on our side. If the external web client team needs to fix it, we would need to negotiate.
Reporter / assignee	DevSecOps Team / External Web Client Team

Technical Debt Examples – Detect from Architecture

Name	Publish/subscribe design likely to not meet real-time timing requirements as message load increases
Summary	We are not able to guarantee delivery of messages in priority order with the current pub/sub design. We already need to deal with increasing number of messages, which is slowing down performance. We may likely have missed messages as well.
Consequences	We will not be able to meet the real-time requirements, hence not pass the tests in the lab, which will delay production.
Remediation approach	Given the size of the code base, switching to a different messaging approach is too risky. However, we can supplement our existing pub/sub architecture with a message prioritization approach with some local architectural changes
Reporter / assignee	Reported during system integration test /assigned to the architecture team

Example Quality Attribute Roots of Technical Debt

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

← Deployability, buildability

← Maintainability, modifiability

← Data model

← Testability

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

Recommendations for Managing *TechDebt* Items

Create a dedicated technical debt item label in your issue tracker.

Ensure that you are only identifying technical debt symptoms through static analysis within the CI/CD tool chain, but also uncovering the technical debt related to the symptoms.

Add technical debt identification at multiple points in the software development lifecycle, including architecture reviews, cybersecurity analysis, and code quality analysis.

Monitor technical debt based on cost of rework, new technical debt items identified, and high-priority technical debt items that need to be resolved.

Triage identified technical debt items regularly based on their priority and decision to when and if to resolve.

Goals of the Training

What is technical debt?

- Its definition and relationship to defects and vulnerabilities
- Misconceptions of what technical debt is

Context matters!

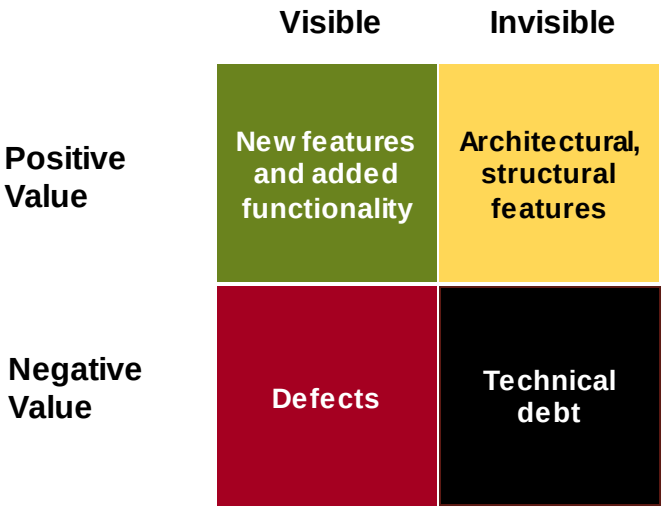
Recognizing technical debt

- Understanding causes of technical debt
- Detecting technical debt

Making technical debt visible

Technical debt within the acquisition lifecycle

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.

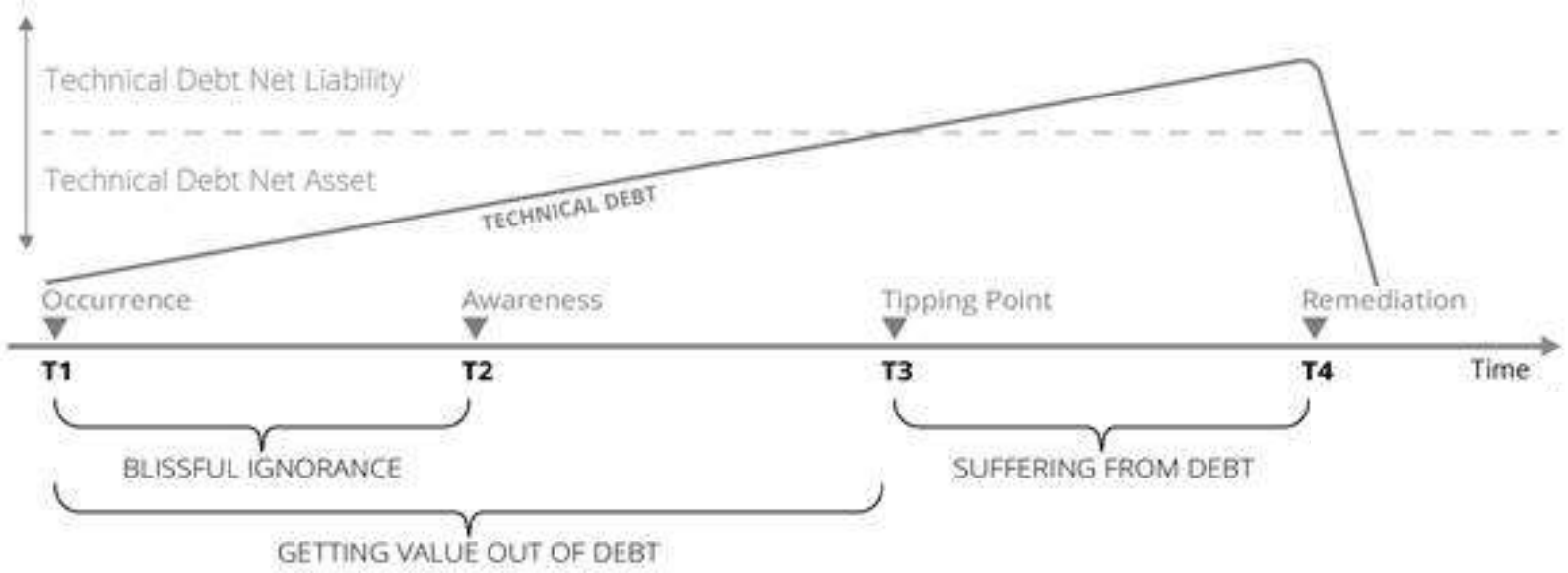
Incorporate Tracking into Existing Practices

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 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.

Manage the Technical Debt Timeline



Technical Debt Management and the Acquisition Lifecycle

Include language on how technical debt will be managed in contracts, including

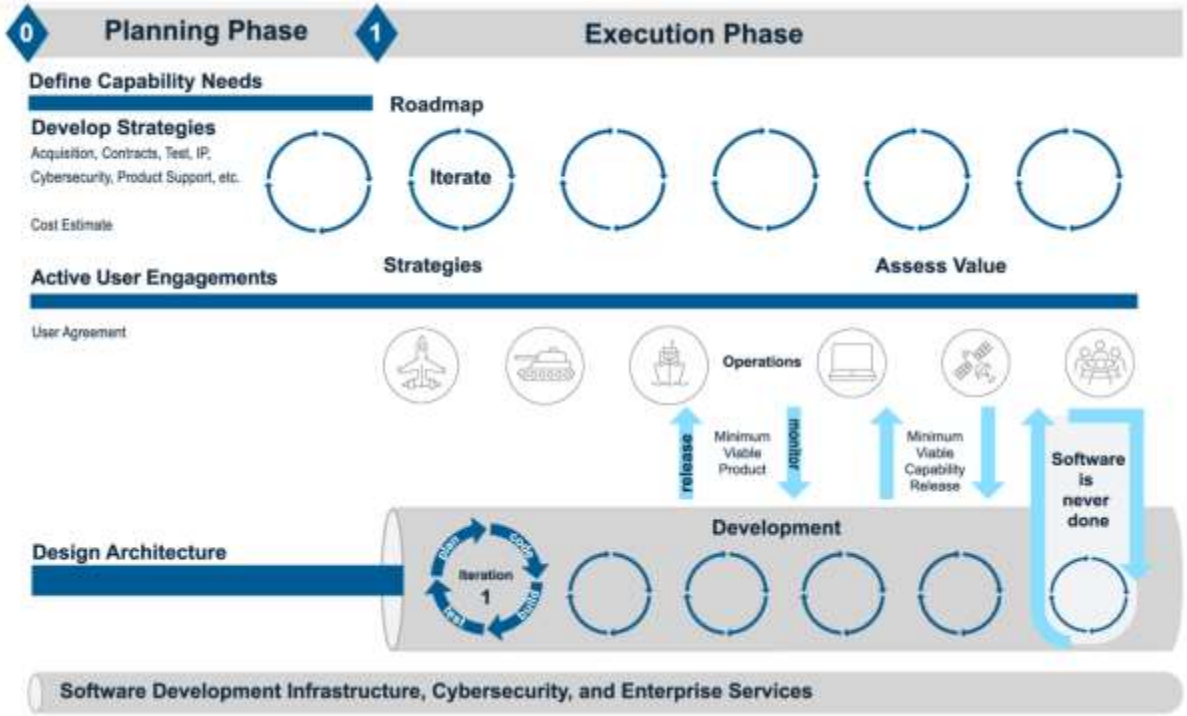
- Percentage of resources to be withheld until high-priority technical debt is resolved
- Data to be shared throughout the development lifecycle
- Ongoing analysis to be conducted and its results shared
- Incentives to share technical debt the contractor takes on

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

Request evidence from contractors and continuously assess where you are on the technical debt timeline.

- Helpful data includes commit histories, defect logs, testing results, architecture conformance measures, and software quality analyses.

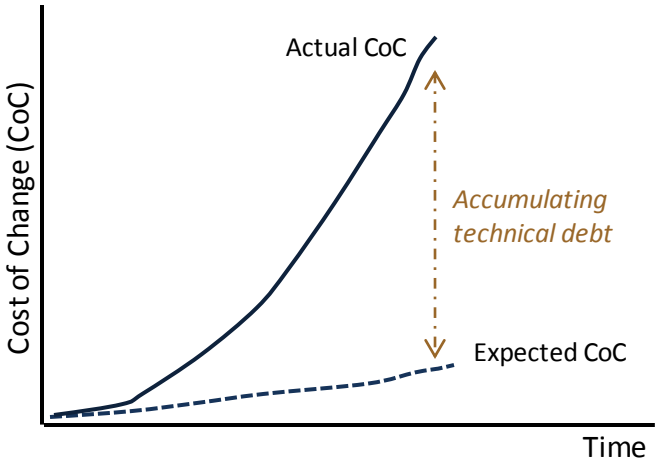
Acquisition Pathways – Software



Programs will maximize use of automated software testing and security accreditation, continuous integration and continuous delivery of software capabilities, and frequent user feedback and engagement. Programs will consider the program's lifecycle objectives and actively manage technical debt. ….

(https://aaf.dau.edu/aaf/software/)

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 tradeoff decisions about remediation.

Next Steps

Build a case to manage debt:

- Identify instances of technical debt over the history of the project.
- Correlate identified instances with expected symptoms of accruing interest.

Use data to identify instances:

- Insights into the planned design (from team members)
- Narrative information about the project history (specifically, any major refactorings that have occurred)
- Code repository; architecture documentation; defect and vulnerability instances

Use data to detect symptoms:

- History of change and defect reports on a project; ideally traceable to versions of the code under configuration management, and to the code modules involved
- Classification of defects and/or changes by type
- Effort by change and/or defect fix

Resources

International Conference on Technical Debt

- Each year the conference brings industry, researchers, and tool vendors together to discuss approaches to improve managing technical debt (e.g., https://conf.researchr.org/home/TechDebt-2022).
- Kruchten, P.; Nord, R.L.; & Ozkaya, I. *Managing Technical Debt: Reducing Friction in Software Development.* Pearson Addison-Wesley. 2019. ISBN-10: 013564593X.

Contact Information

Dr. Ipek Ozkaya

Technical Director

Engineering Intelligent Software Systems

Software Engineering Institute

Carnegie Mellon University

ozkaya@sei.cmu.edu