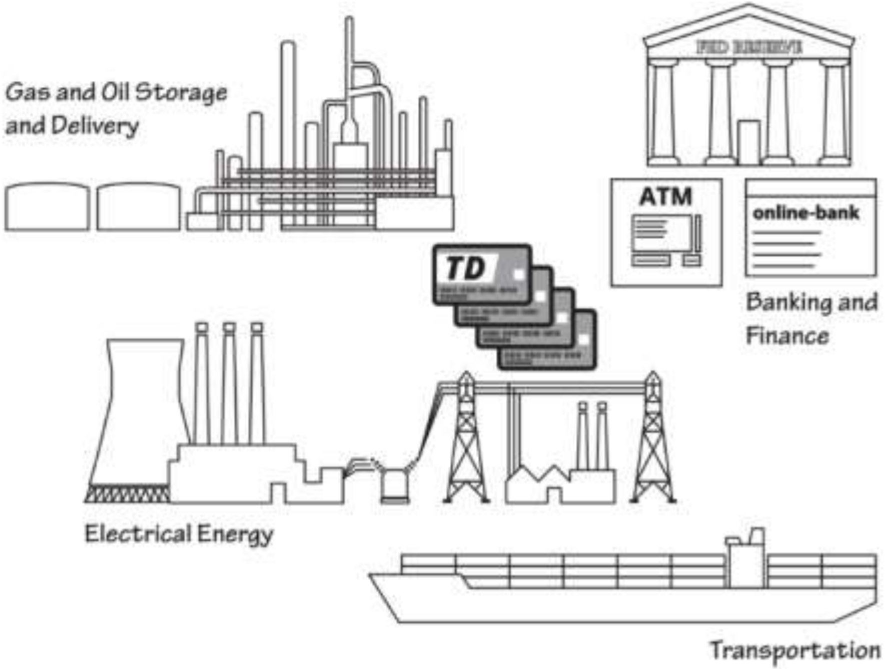
10 Years of Research in Technical Debt and an Agenda for the Future

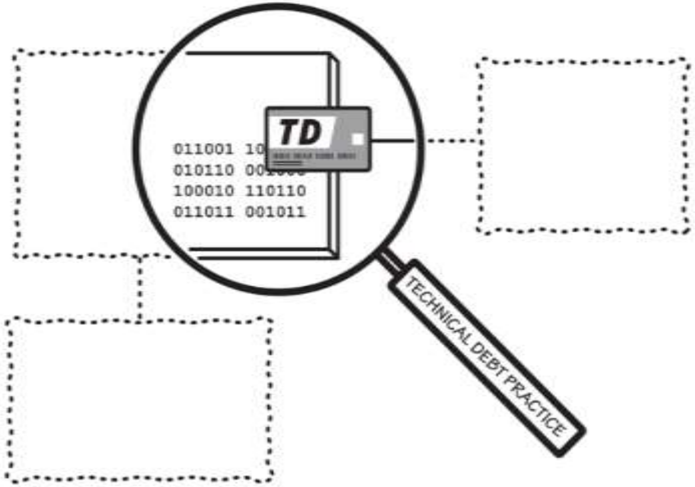
Robert Nord (rn@sei.cmu.edu)

Ipek Ozkaya (ozkaya@sei.cmu.edu)

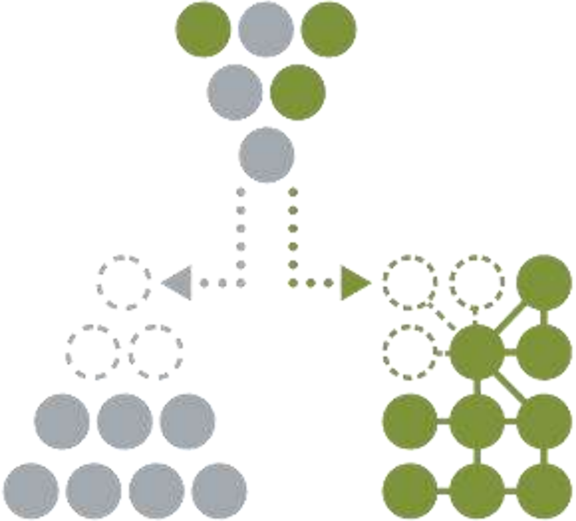
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

ALL SYSTEMS HAVE TECHNICAL DEBT!



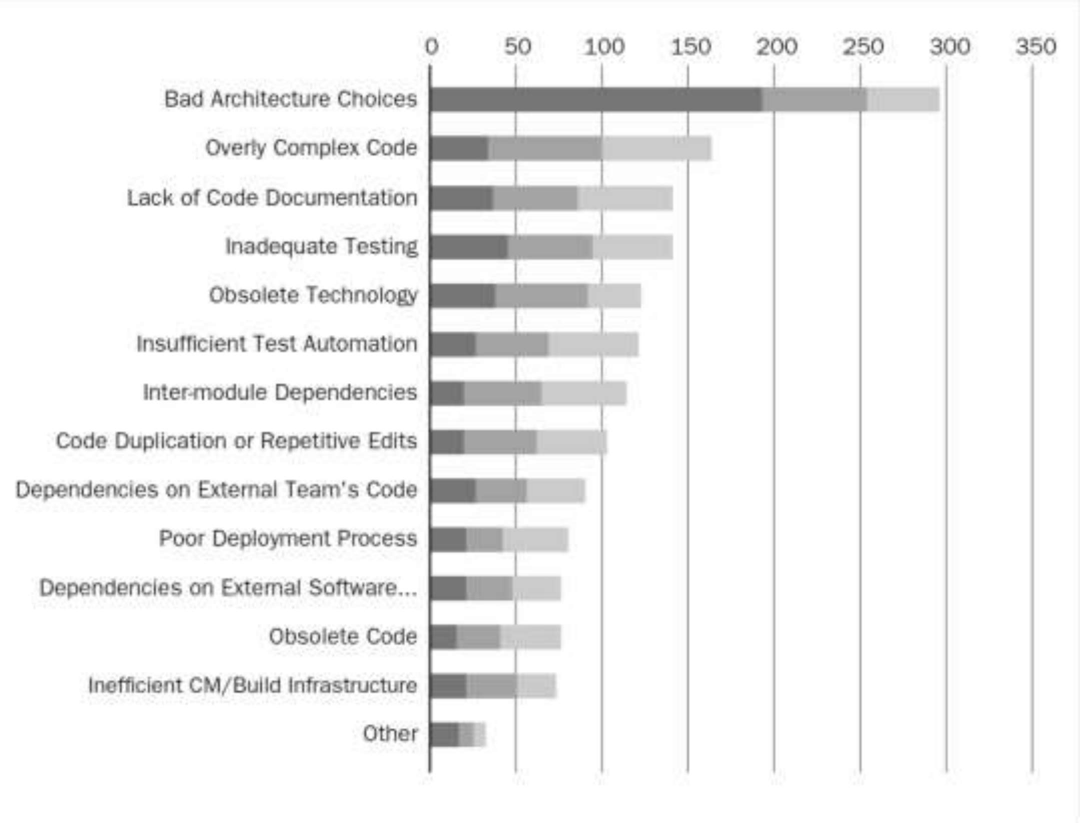


Technical Debt: A Definition



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.

Software Architecture and Design Tradeoffs Matter



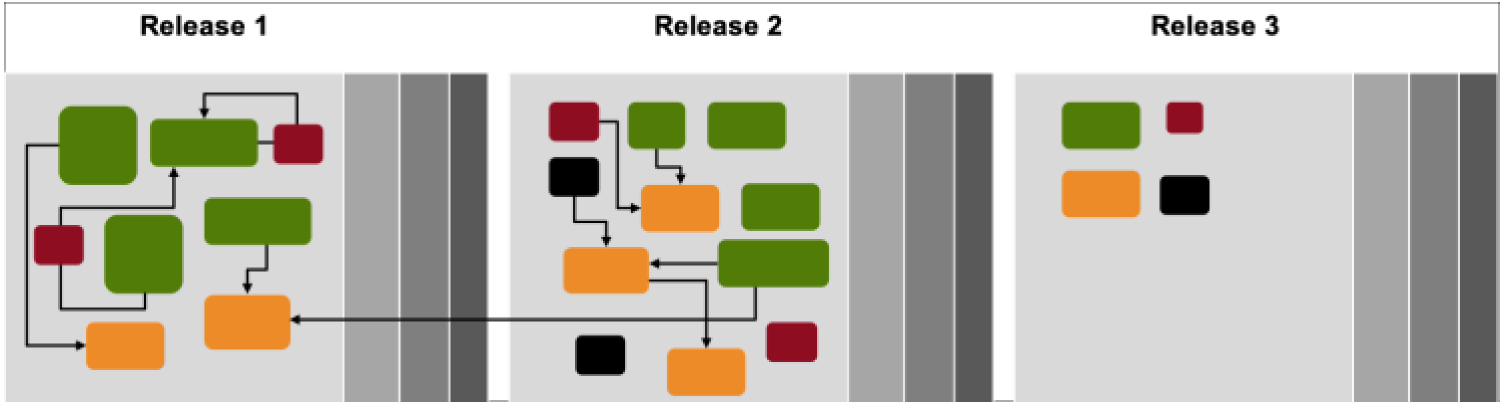
Results from over 1800 developers from two large industry and one government software development organizations reinforce that unattended architecture decisions and practices are at the root of technical debt.

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.

Our vision in 2012

Provide a way for the software engineering community to recognize that technical debt has its roots in architecture rework.

Our 2012 ICSA paper* presented a dependency analysis framework for measuring architecture rework as a proxy for technical debt.

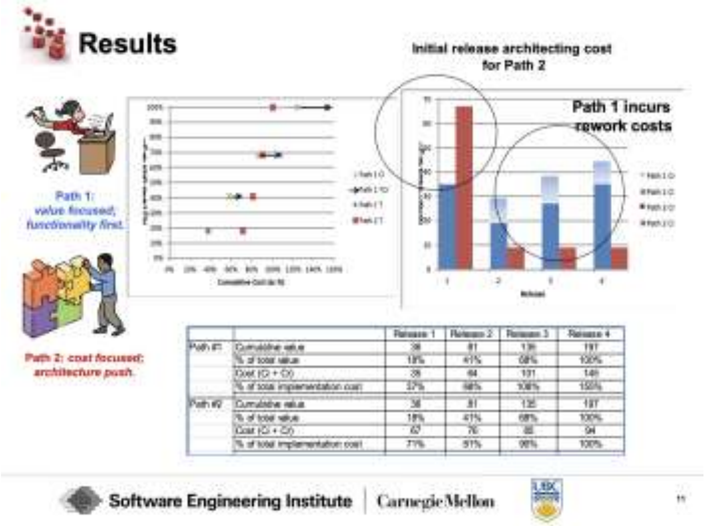


legend

New features and added functionality	Architectural, structural features
Defects	Technical Debt

* In search of a metric for managing architectural technical debt
RL Nord, I Ozkaya, P Kruchten, M Gonzalez-Rojas - 2012 Joint Working IEEE/IFIP Conference on Softw are Architecture, 2012

Let’s Recall the Paper and the Presentation



Results

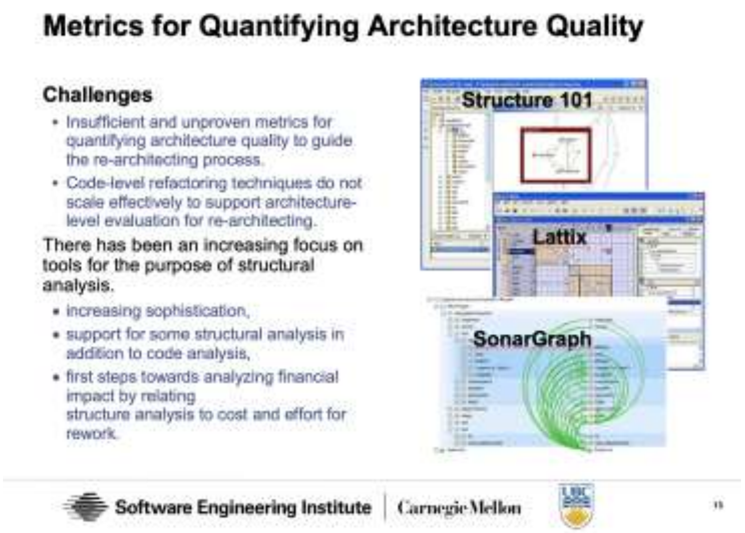
Initial release architecting cost for Path 2
Path 1 incurs rework costs
Path 1: value focused; functionality first.
Path 2: cost focused; architecture push.
Cumulative Cost (in %)
Release
1
2
3
4

		Release 1	Release 2	Release 3	Release 4
Path #1	Cumulative value	36	81	136	197
	% of total value	18%	41%	68%	100%
	Cost (Ci + Cr)	38	64	101	149
	% of total implementation cost	37%	68%	108%	150%
Path #2	Cumulative value	36	81	135	197
	% of total value	18%	41%	68%	100%
	Cost (Ci + Cr)	67	76	85	94
	% of total implementation cost	71%	81%	90%	100%

Software Engineering Institute | Carnegie Mellon

We demonstrated the potential rework created as a consequence of the tension between architecture decisions and delivering priority functional requirements.

We exemplified use of one metric, propagation cost, but emphasized that quantifying rework is not trivial.



Metrics for Quantifying Architecture Quality

Challenges

- Insufficient and unproven metrics for quantifying architecture quality to guide the re-architecting process.
- Code-level refactoring techniques do not scale effectively to support architecture-level evaluation for re-architecting.

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

- 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.

Software Engineering Institute | Carnegie Mellon

The Work that Followed

Research recognizing the **connection of architecture and design roots of technical debt**

- e.g. Martini and Bosch 2014-2015, Lin, Liang, Avgeriou 2015

Research investigating propagation cost (Pc) and other **architecture related metrics and architecture smell detection**

- e.g. Abad 2015, Ampatzoglou 2015, MacCormack 2016, Azadi 2019, Verdecchia 2020

Self-admitted technical debt research which identified conversations in code comments providing further examples of technical debt and architecture.

- e.g. Maldonado and Shihab 2017

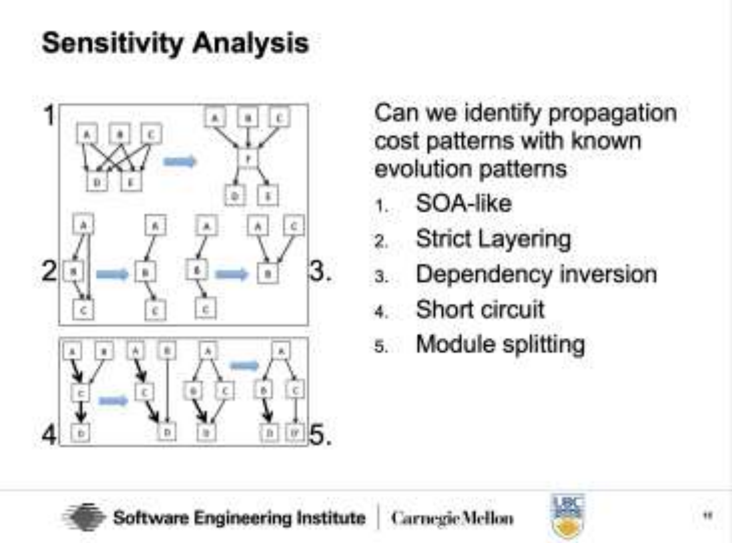
Work that focused on understanding **how to manage technical debt** and **architecture evolution,** including systematic literature studies

- e.g. Fontana 2016, Guo 2016, Letouzey 2012, Rios 2018, Besker 2018

The good, the bad, and the opportunity

The paper made the architecture roots of technical debt very visible.

“Architectural technical debt” in our title unintentionally implied a technical debt taxonomy.



Sensitivity Analysis

Can we identify propagation cost patterns with known evolution patterns

1. SOA-like
2. Strict Layering
3. Dependency inversion
4. Short circuit
5. Module splitting

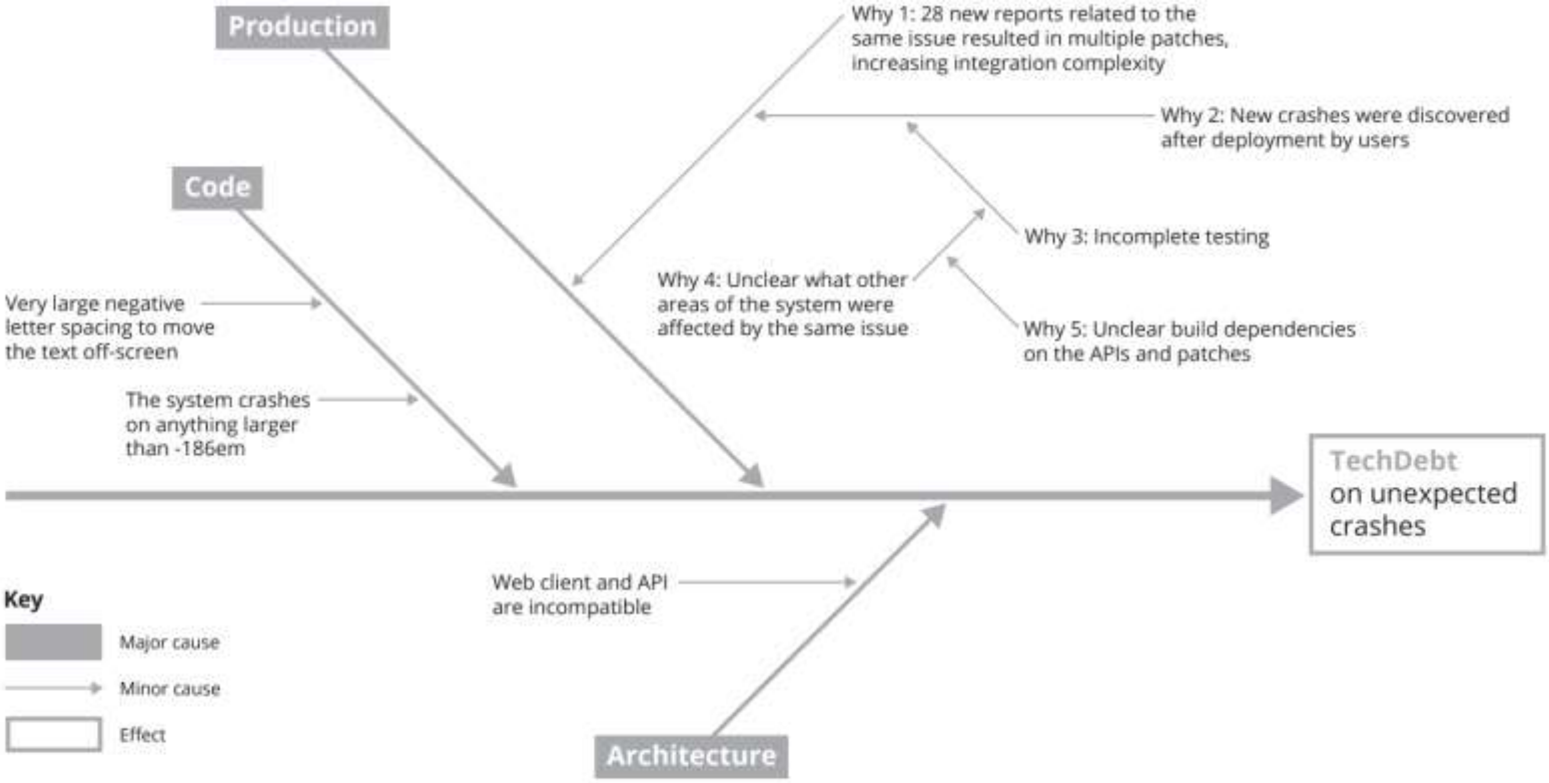
Software Engineering Institute | Carnegie Mellon

11

Not enough people attended the presentation to pick up on our tease to investigate the relationship between design patterns and metrics.

R. L. Nord, I. Ozkaya, R. S. Sangwan, J. Delange, M. A. Gonzalez, P. Kruchten: *Variations on Using Propagation Cost to Measure Architecture Modifiability Properties*. ICSM 2013: 400-403

Technical Debt Causal Chain



Production

Why 1: 28 new reports related to the same issue resulted in multiple patches, increasing integration complexity

Why 2: New crashes were discovered after deployment by users

Code

Why 3: Incomplete testing

Why 4: Unclear what other areas of the system were affected by the same issue

Why 5: Unclear build dependencies on the APIs and patches

Very large negative letter spacing to move the text off-screen

The system crashes on anything larger than -186em

TechDebt on unexpected crashes

Web client and API are incompatible

Architecture

Key

Major cause

Minor cause

Effect

A Typical Example

Technical debt is a **software design decision** made to solve a problem but may not stand the test of time and cause rework.

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

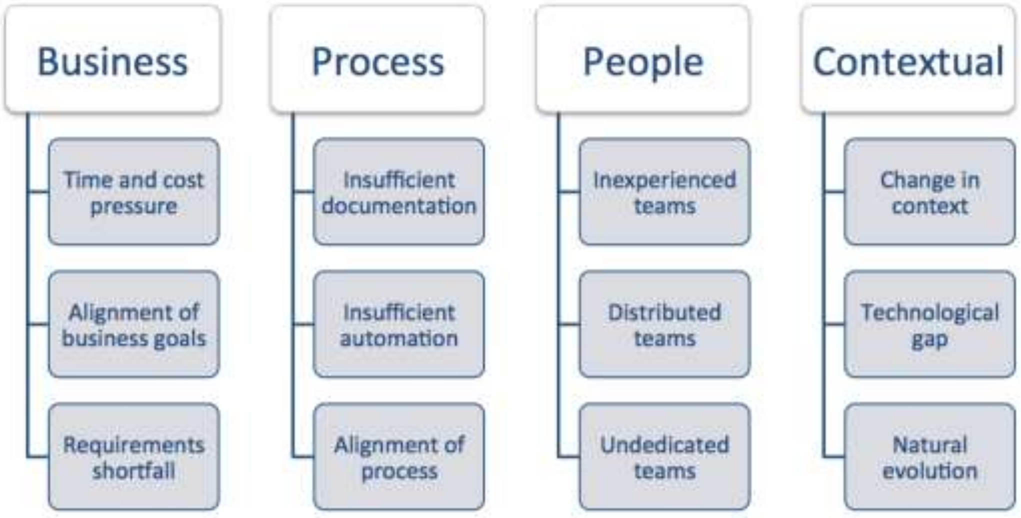
“A decade ago processors were not as powerful. To optimize for performance, we would not insert **code for exception handling** when we knew we would not **divide by zero** or hit an **out of bounds memory** condition. These **areas now are hard to track** and have become **security nightmares**.”

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

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

Context Matters

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 our recommended first step to ensure causes of technical debt and its symptoms are understood.

Approach: Set of interviews and stakeholder-focused meetings to analyze project context, supported by artifact review as needed.

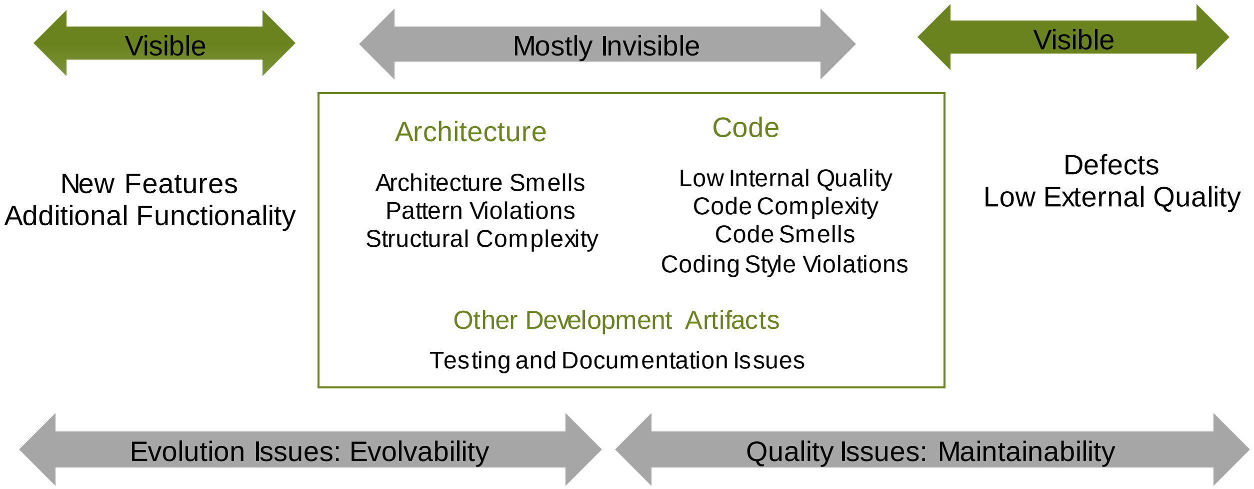
Outcomes: A scorecard and supporting data which includes a list of potential and actual causes of technical debt, impact rating for each, and approaches for identifying relevant technical debt.

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*

The Technical Debt Landscape



Visible

Mostly Invisible

Visible

New Features
Additional Functionality

Architecture

Architecture Smells
Pattern Violations
Structural Complexity

Code

Low Internal Quality
Code Complexity
Code Smells
Coding Style Violations

Other Development Artifacts

Testing and Documentation Issues

Defects
Low External Quality

Evolution Issues: Evolvability

Quality Issues: Maintainability

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.
Managing Technical Debt Reducing Friction in Software Development, Pearson Addison-Wesley, 2019.

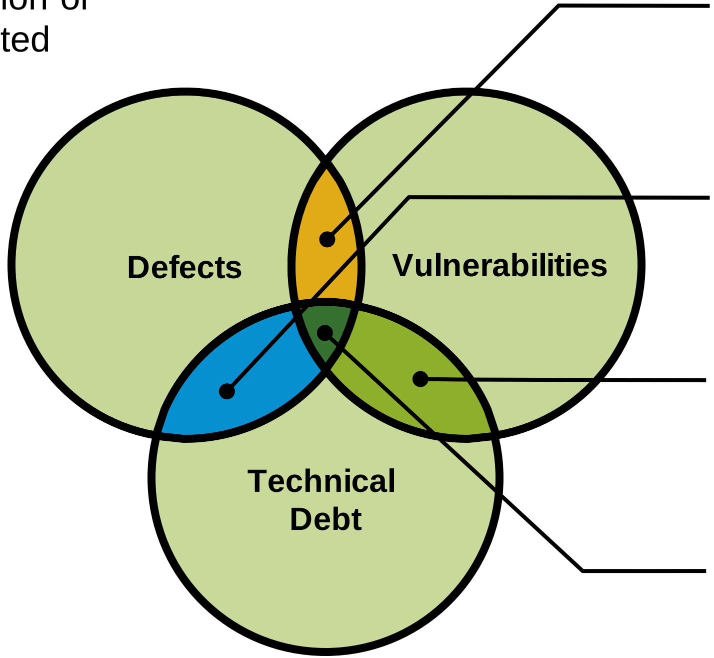
Technical Debt Fills a Missing Gap in Software Development

defect – error in coding or logic that causes a program to malfunction or to produce incorrect/ 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.

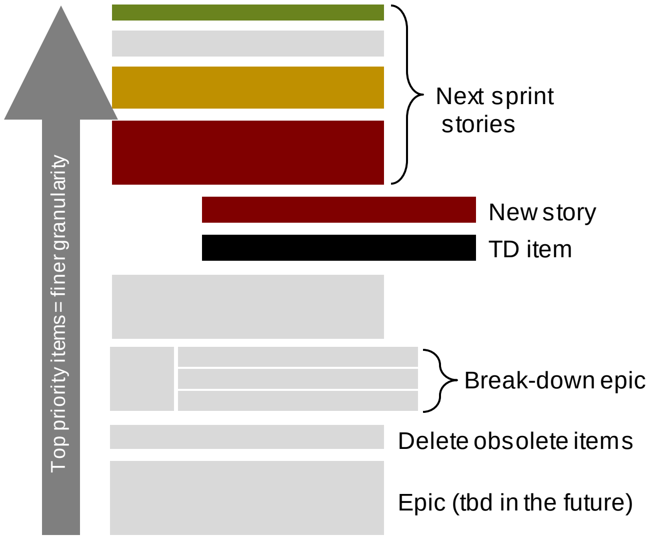
Technical debt increases vulnerability risks.

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

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



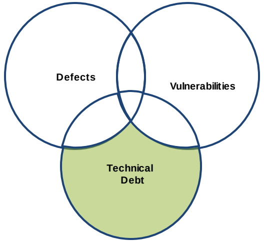
Detecting Technical Debt

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. July 2021.

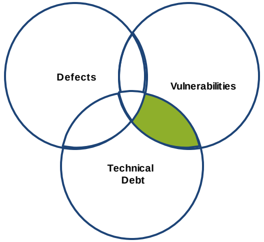
Technical Debt Item Examples – Detect from Code

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

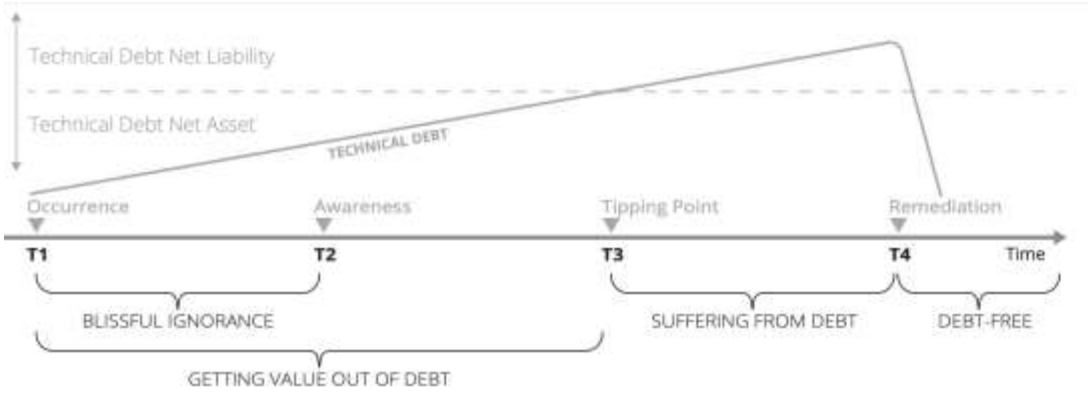


Technical Debt Item Examples – Detect from Symptoms

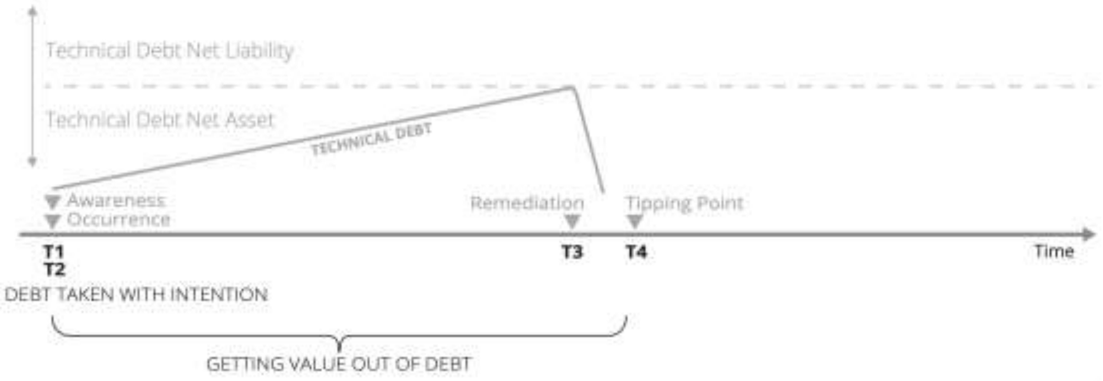
Name	Unexpected crashes due to API incompatibility
Summary	The source code uses a very large negative letter-spacing in an attempt to move the text offscreen. The system handles up to -186 em fine, but crashes on anything larger. A similar issue was fixed with a patch, but there were several other similar 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



Manage the Technical Debt Timeline



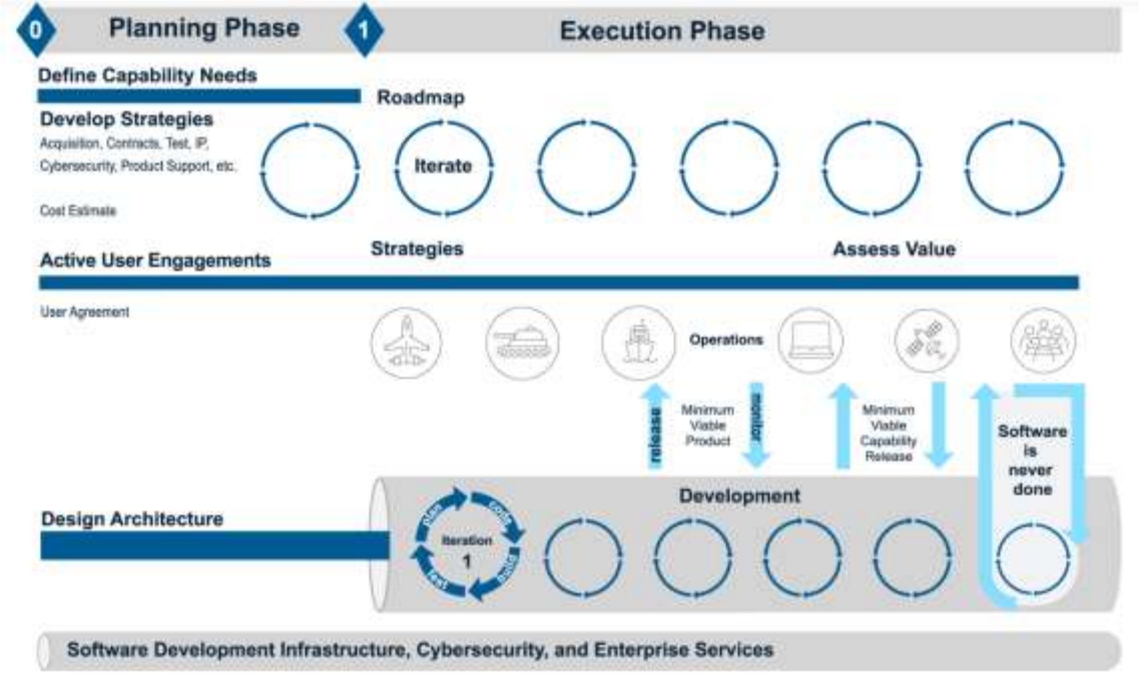
Unintentional and not well managed technical debt



Intentional and well managed technical debt

Acquisition Pathways – Software

An iterative and incremental, architecture focused process which includes proactive technical debt management is recommended.



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/)

Going forward

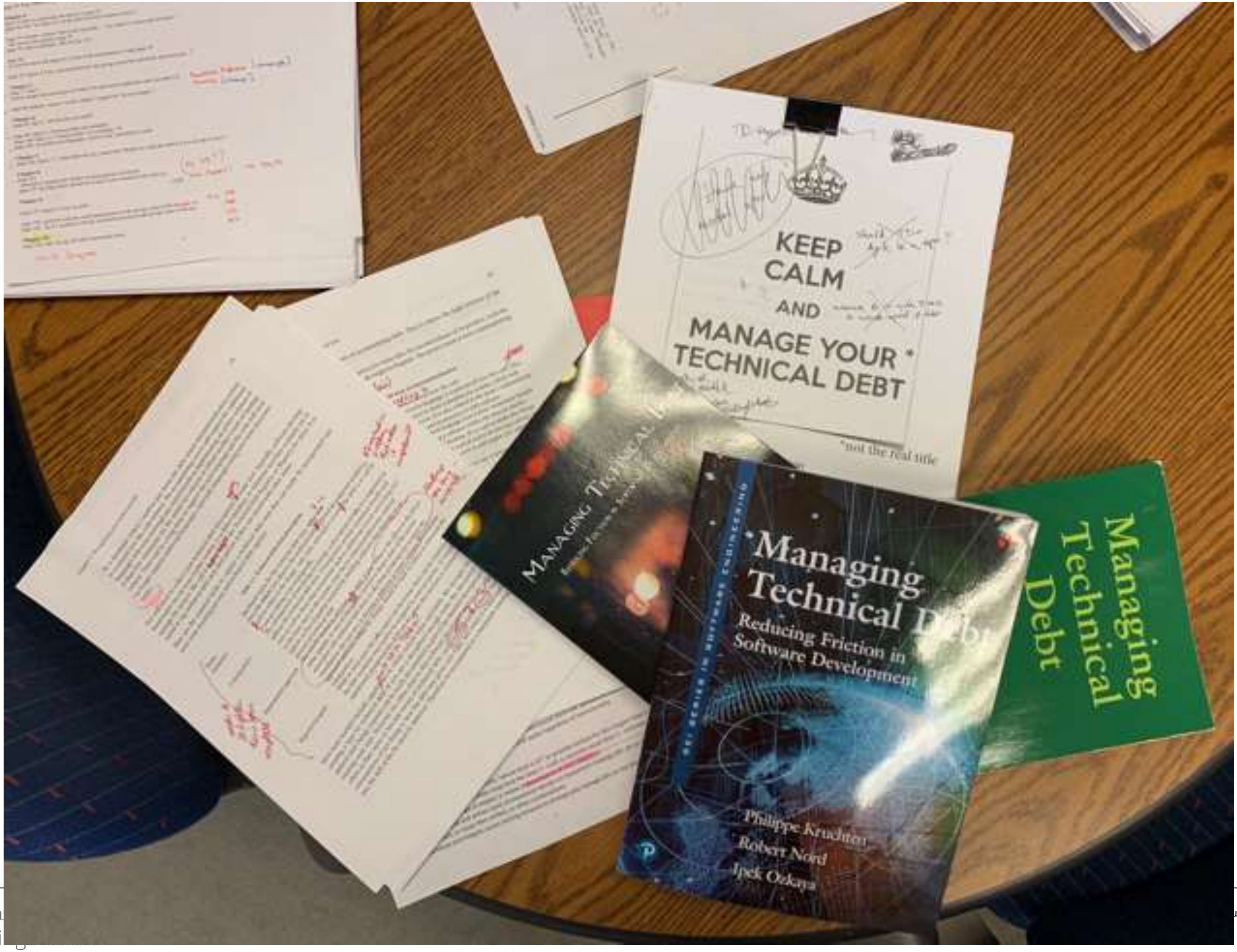
Both commercial software industry (e.g. agile at scale/DevOps practices) and regulated software environments (e.g. Adaptive Acquisition Framework) today recognize technical debt management as a core software engineering practice.

TechDebt conference (www.techdebtconf.org) is a way to connect to ongoing research.

NDAA Section 835 calls for a study on better understanding

Open questions include:

- How to quantify rework with a variety of metrics to guide how and when to refactor systems to resolve technical debt?
- How can rework quantification be related to operational practices, e.g. how should technical debt be recorded and prioritized?
- How can empirical data and analysis be used to improve iterative and incremental architecture practices to manage technical debt?


KEEP CALM AND MANAGE YOUR * TECHNICAL DEBT
*not the real title
Managing Technical Debt
Managing Technical Debt
Reducing Friction in Software Development
Philippe Kruchten
Robert Nord
Ipek Ozkaya
SEI Series in Software Engineering
Managing Technical Debt

THANK YOU!