Exploring the Technical Debt Landscape

Ipek Ozkaya
ozkaya@sei.cmu.edu

Robert Nord
rn@sei.cmu.edu

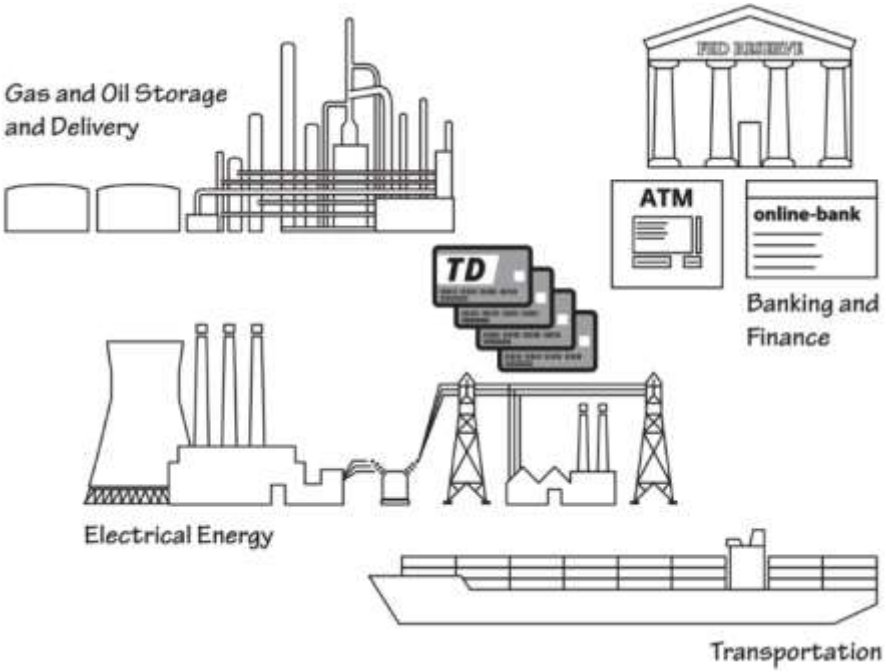
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

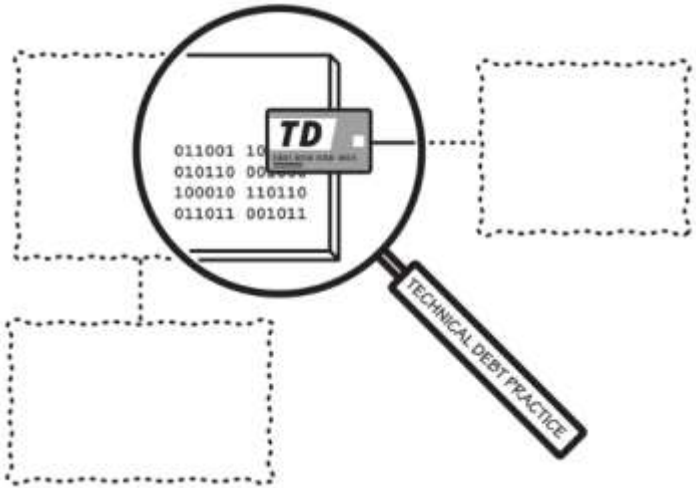
Document Markings

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.

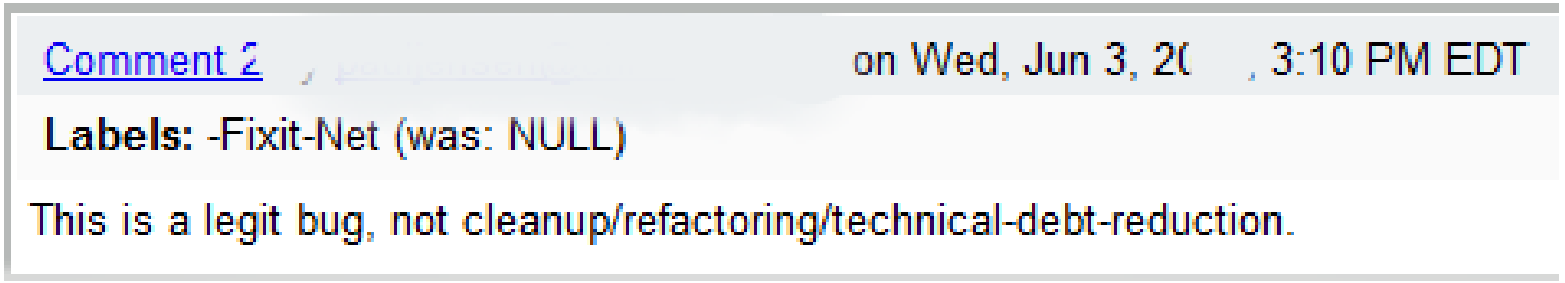
DM19-1048

ALL SYSTEMS HAVE TECHNICAL DEBT!





Software engineers know technical debt when they see it!



Software engineers know technical debt when they see it!

Comment 7 by

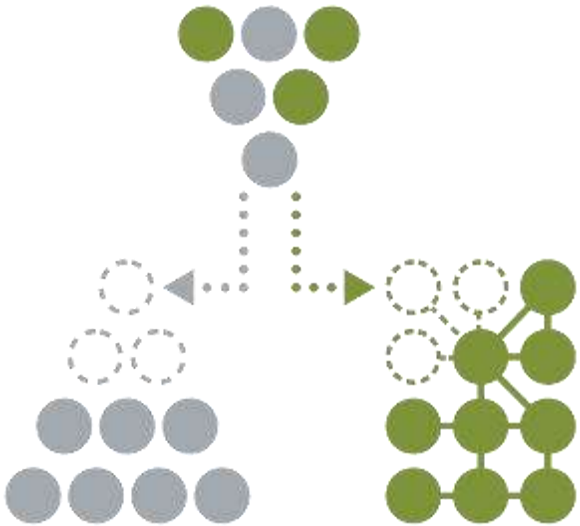
Labels: -Merge-Rejected Merge-Requested (was: NULL)

I'd like to re-request merge for this.

The change looks larger and more complex than it really is: it's mostly plumbing and changes to method signatures adding to the line count. It's been in canary for a week without issue.

Landing this will enable WebView to shed some technical debt, which is quite a big benefit for us.

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, primarily maintainability and evolvability.

**Kruchten, P. Nord, R. Ozkaya, I. Managing Technical Debt, 2019 Pearson*

Crumbling under the load

A successful company in the maritime equipment industry successfully evolved its products for 16 years, amassing 3 million lines of code.

Over these 16 years, it launched many different products; new technologies evolved; staff turned over; and new competitors entered the industry.

Death by a thousand cuts

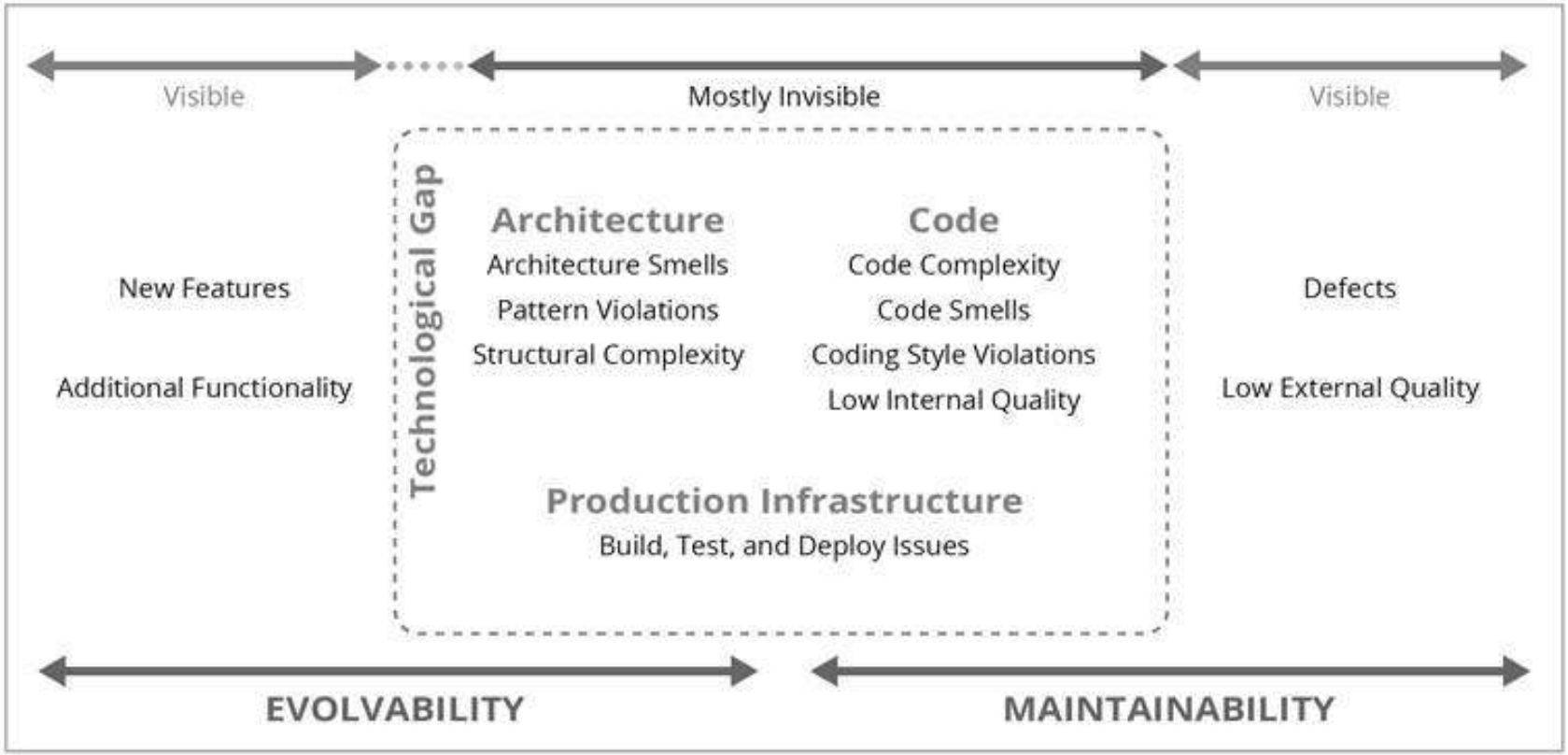
An IT-service organization landed several major contracts, some of which in emerging software development markets.

While the IT-service projects were similar in nature, the organization assumed that its new developers were interchangeable across projects.

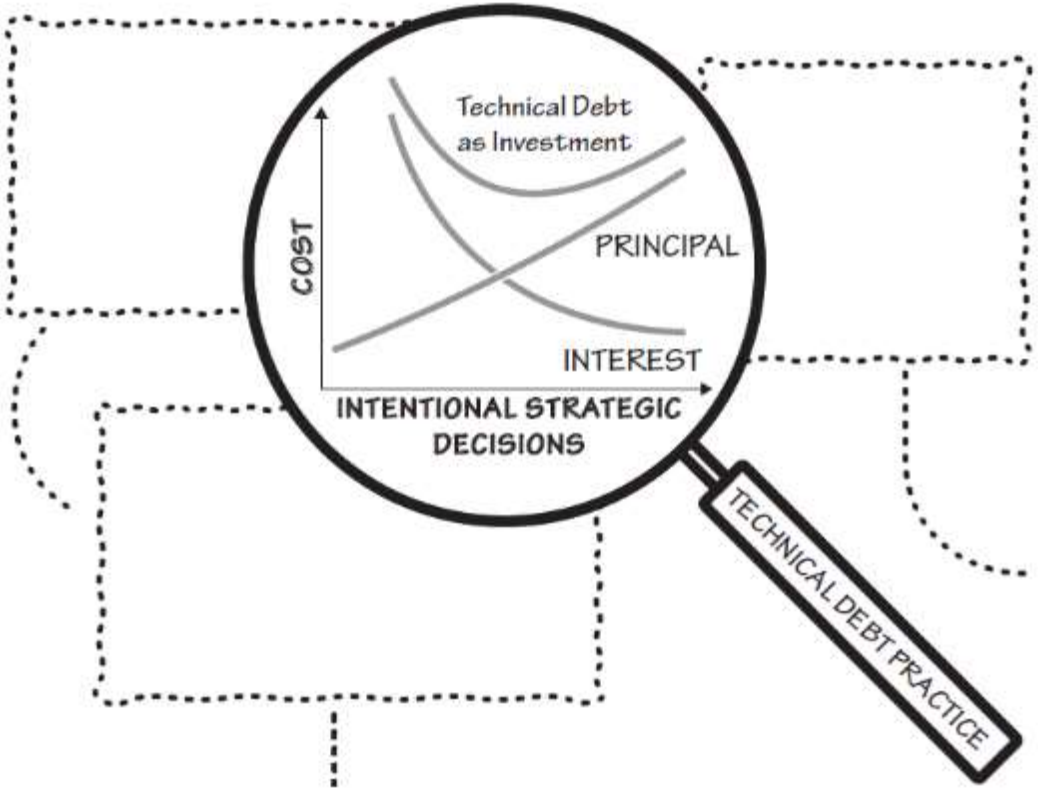
Hitting the Wall

Two large global financial institutions merged. The new management determined that a duct-tape and rubber-band approach, mixing the two IT systems in some kind of chimera, would not work. They had to walk away from years of accumulated technical debt in the original systems.

The Technical Debt Landscape



Principle: Technical debt is not synonymous with bad quality



We need 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

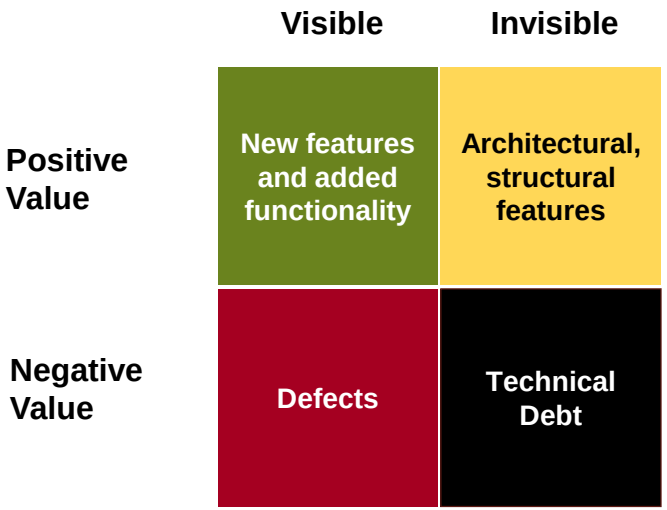
Technical debt is a software design issue that:

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

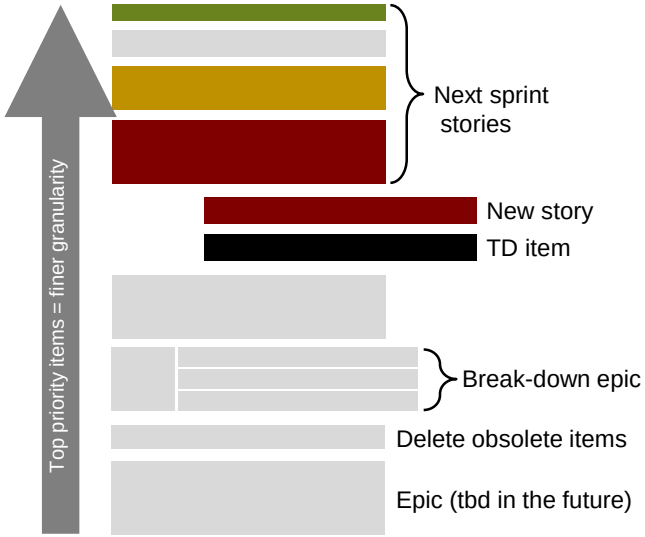
Is traced to **several locations** in the system, implying issues are not isolated but propagate throughout the system artifacts;

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

Essential Software Development Artifacts include Technical Debt



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

What is the nature of YOUR debt

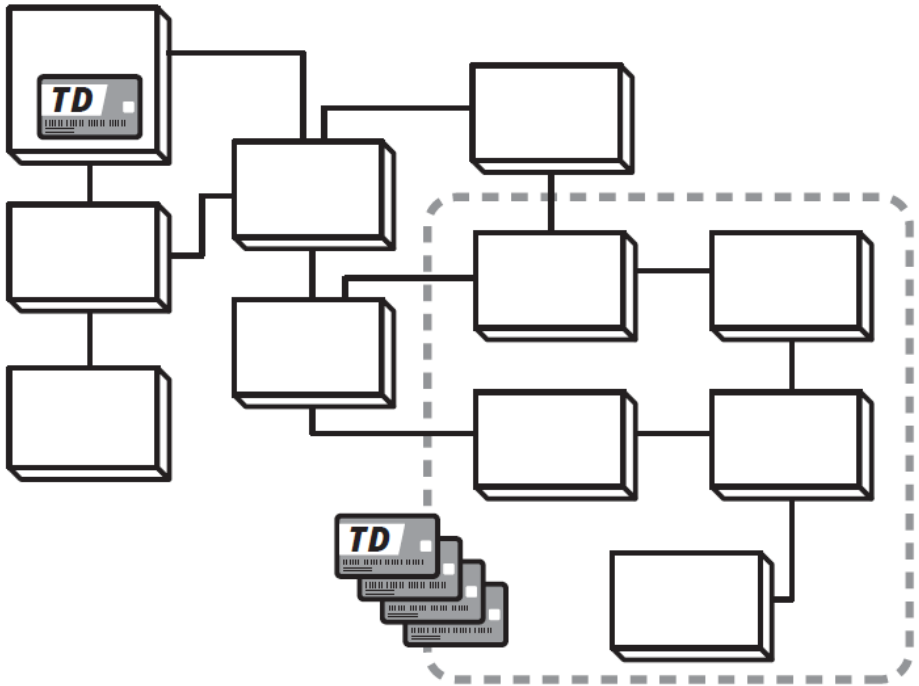
Deployment & Build	Out-of-sync build dependencies
	Version conflict
	Dead code in build scripts
Code Structure	Event handling
	API/Interfaces
	Unreliable output or behavior
	Type conformance issue
	UI design
	Throttling
	Dead code
	Large file processing or rendering
	Memory limitation
	Poor error handling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

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.

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

Principle: Architecture debt has the highest cost of ownership



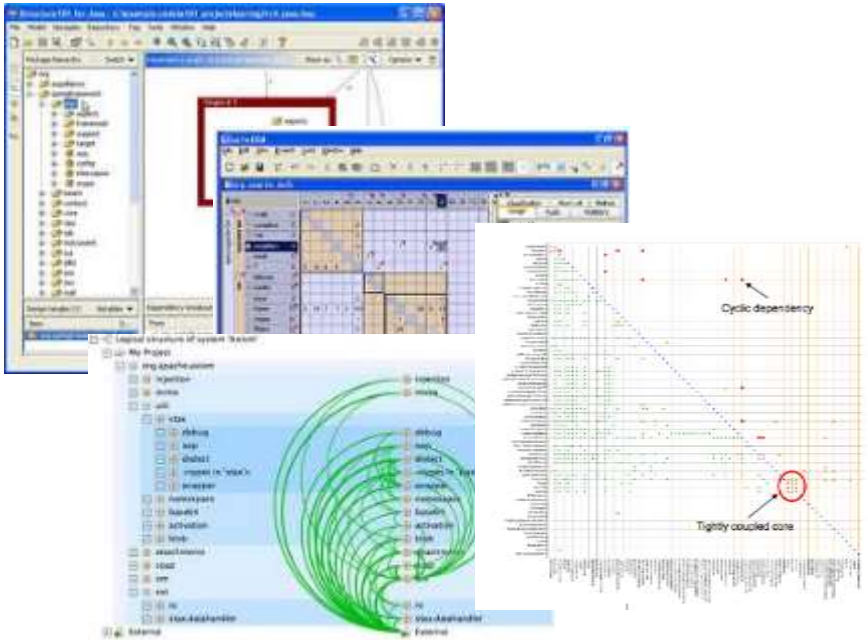
Experiences from Google:

C. Sadowski, E. Aftandilian, A. Eagle, L. Miller-Cushon, C. *Jaspan Lessons from building static analysis tools at Google*. Commun. ACM 61(4): 58-66 (2018)

D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, Vinay Chaudhary, M. Young, J. Crespo, D. Dennison: *Hidden Technical Debt in Machine Learning Systems*. NIPS 2015: 2503-2511

J. D. Morgenthaler, M. Gridnev, R. Sauciuc, S. Bhansali: *Searching for build debt: experiences managing technical debt at Google*. MTD@ICSE 2012: 1-6

Take Advantage of Tool Support



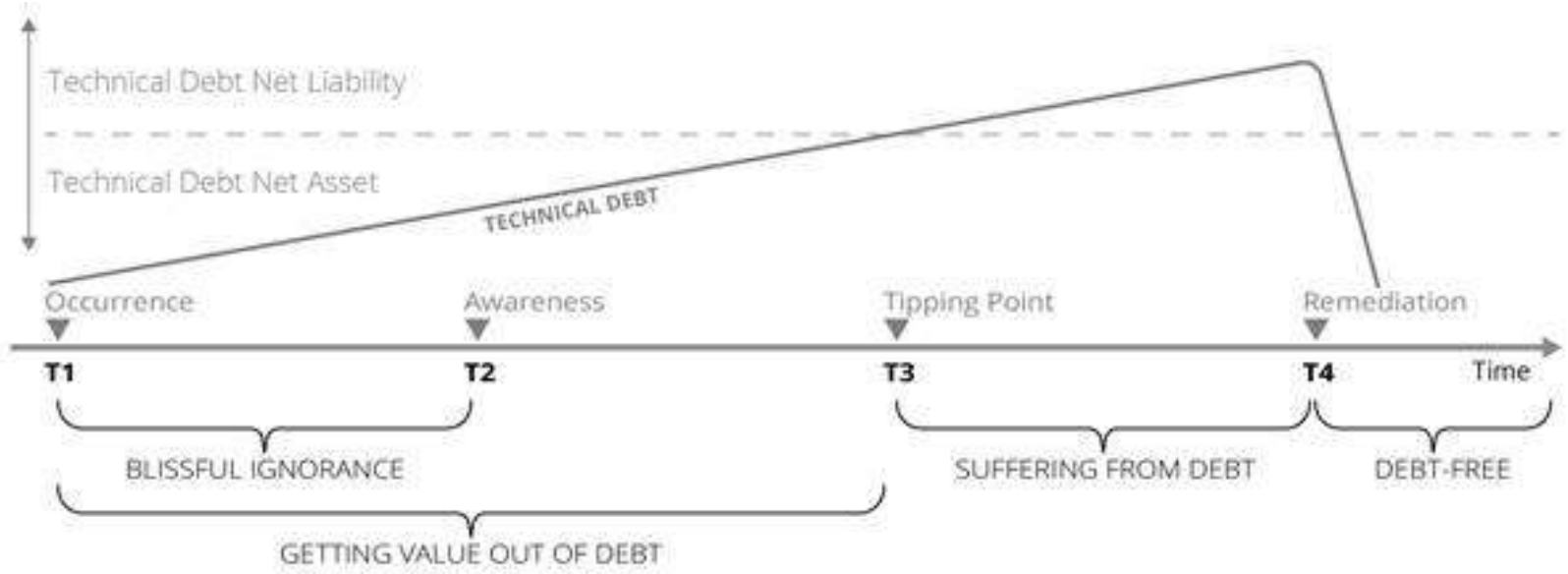
Tools can help assess aspects of software complexity and structural quality.

This is only a starting point!

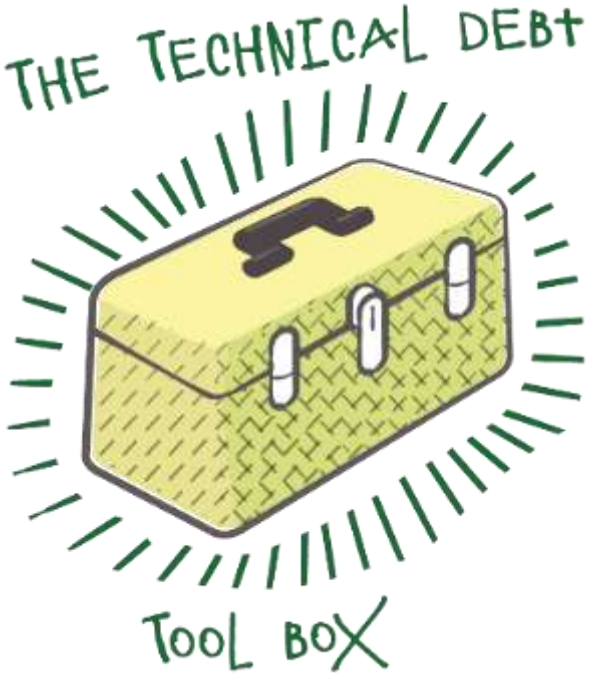
Information from these tools needs to be coupled with an understanding of:

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

Manage the Technical Debt Timeline



Your Technical Debt Tool Box



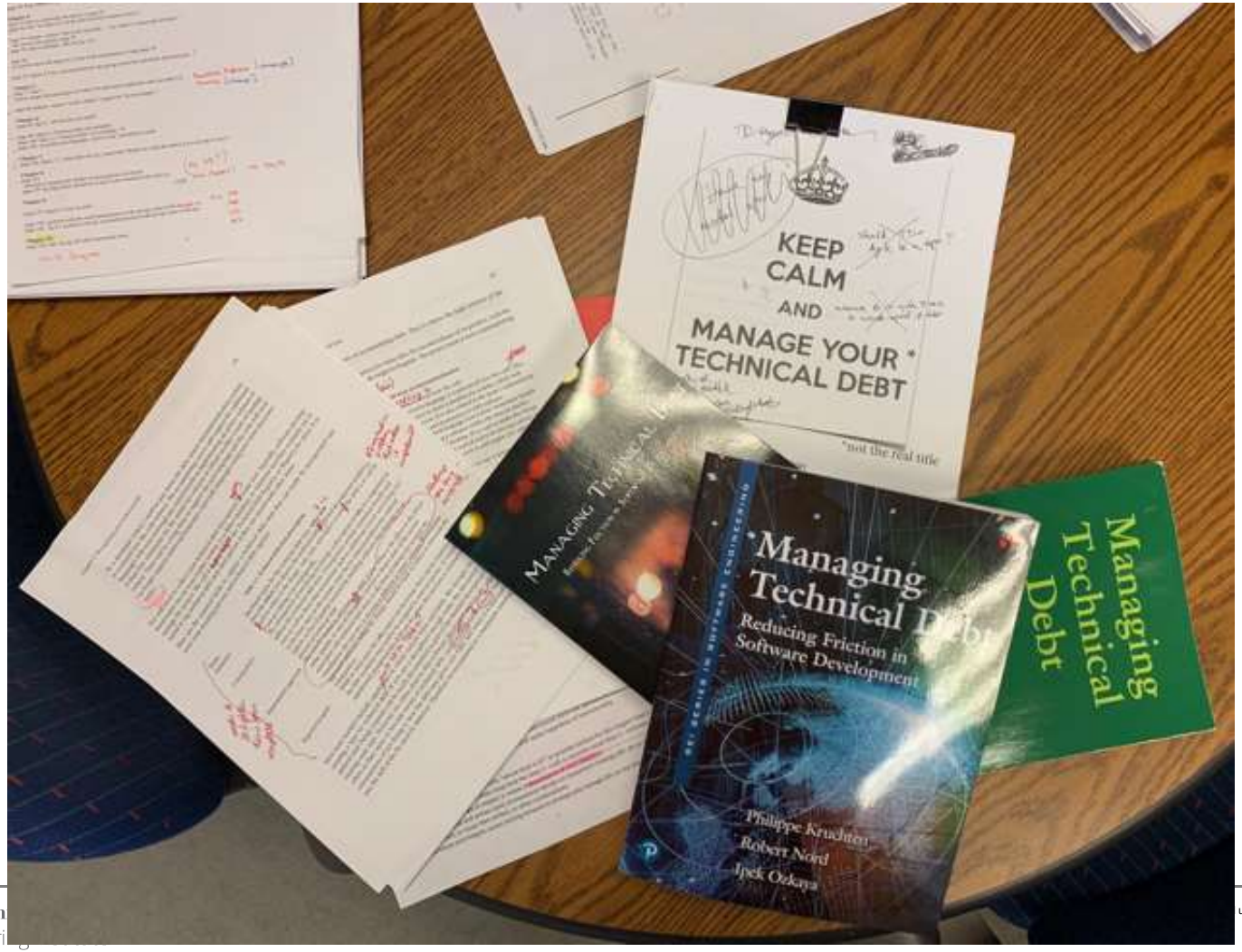
Become aware

Assess the information

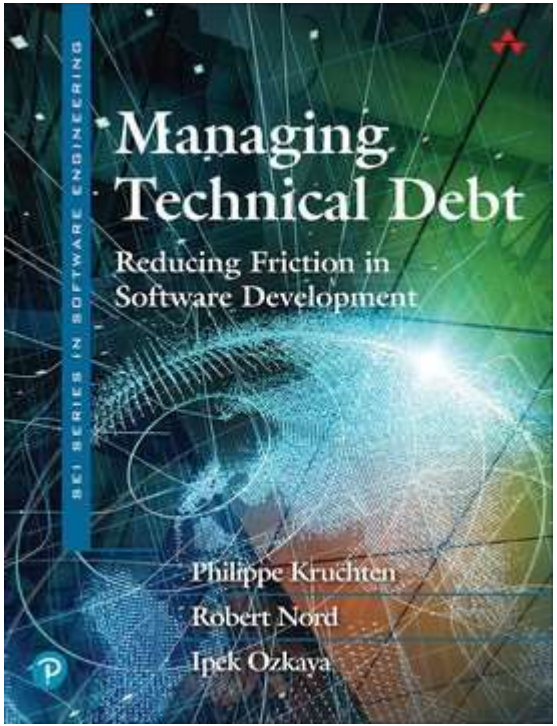
Build a registry

Decide what to fix

Take action


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

For more stories and practices ….



Philippe Kruchten
pbk@ece.ubc.ca

Robert Nord
rn@sei.cmu.edu

Ipek Ozkaya
ozkaya@sei.cmu.edu