Managing Technical Debt

Ipek Ozkaya

February 17, 2021

Discussion with GBSD

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.

DM21-0143

Agenda

- What is technical debt?
- Managing technical debt
- Getting started

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.

Common Consequences of Technical Debt

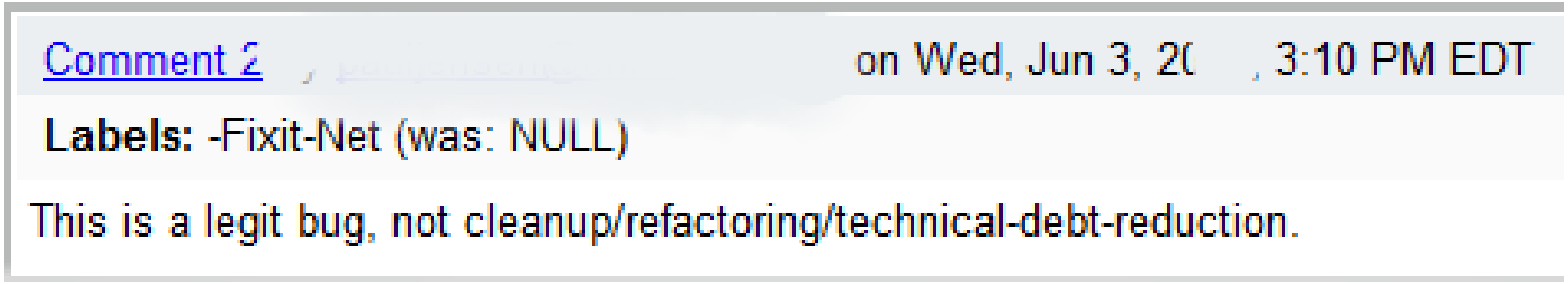
- Teams spend almost all of their time fixing defects, and new capability development is continuously slipping.
- Integration of products built by different teams reveals that incompatibilities cause many failure conditions and lead to significant out-of-cycle rework.
- Progress toward milestones is 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 length convoluted solutions and added complexity in maintaining or extending the systems.

Technical Debt is Common

Unmanaged technical debt costs organizations time and money!

- In 2010 Gartner estimated the total amount of technical debt worldwide could reach $1 trillion
- Government's old technology deficit that needs to be replaced is estimated to be up to $7.5 billion
- U.S. Department of Veterans Affairs, spending 75% of its technology budget to maintain outdated legacy systems
- 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)

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.

A Typical Example

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).

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 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

Managing Technical Debt

Organizations needs to address the following challenges continuously:

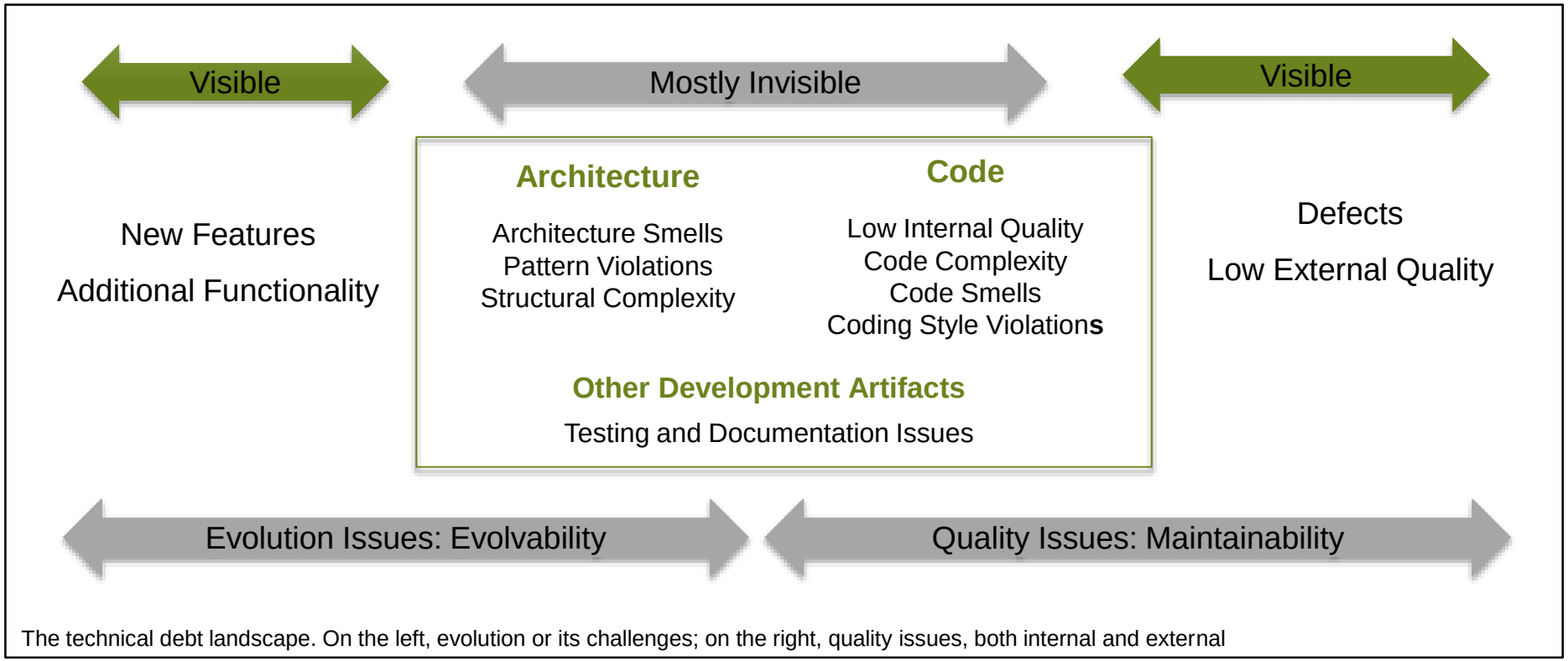
1. Recognizing technical debt
2. Making technical debt visible
3. Deciding when and how to resolve debt
4. Living with technical debt

1. 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

From Symptoms to Specifics



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.

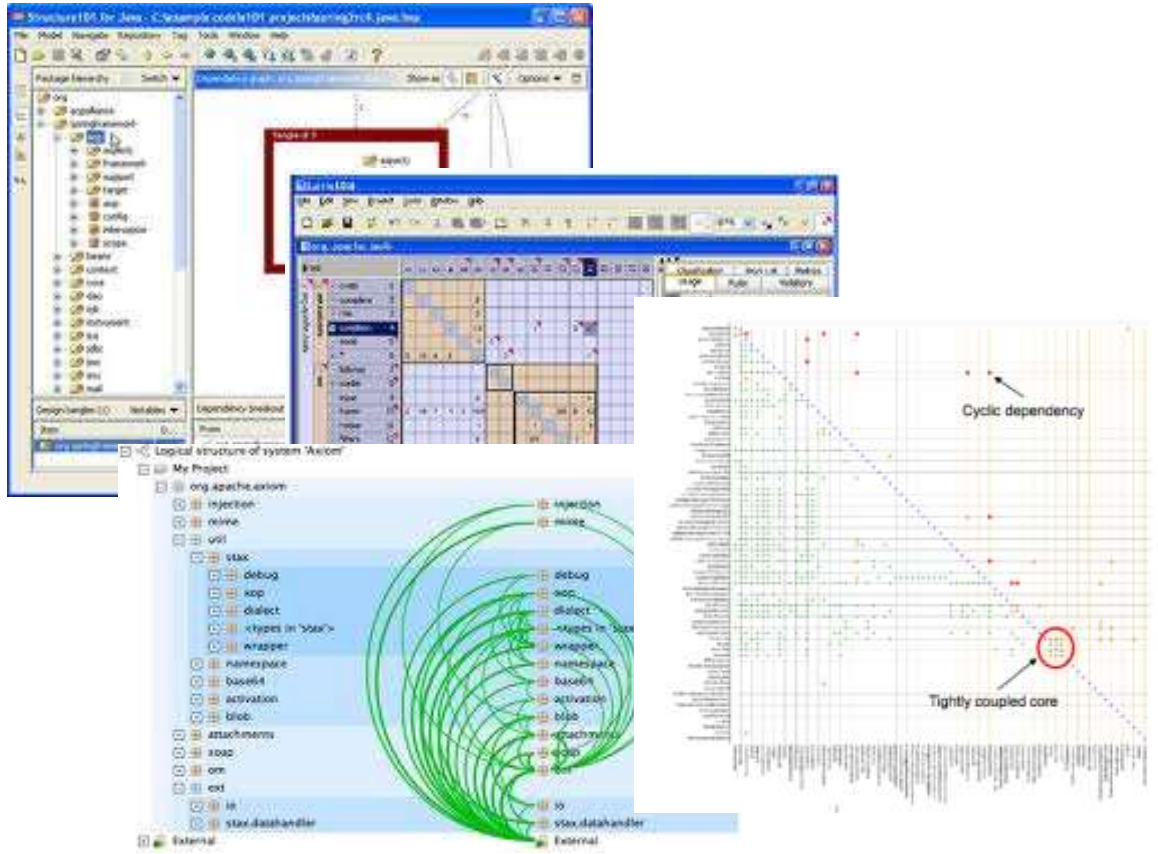
Taking Advantage of Tool Support

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

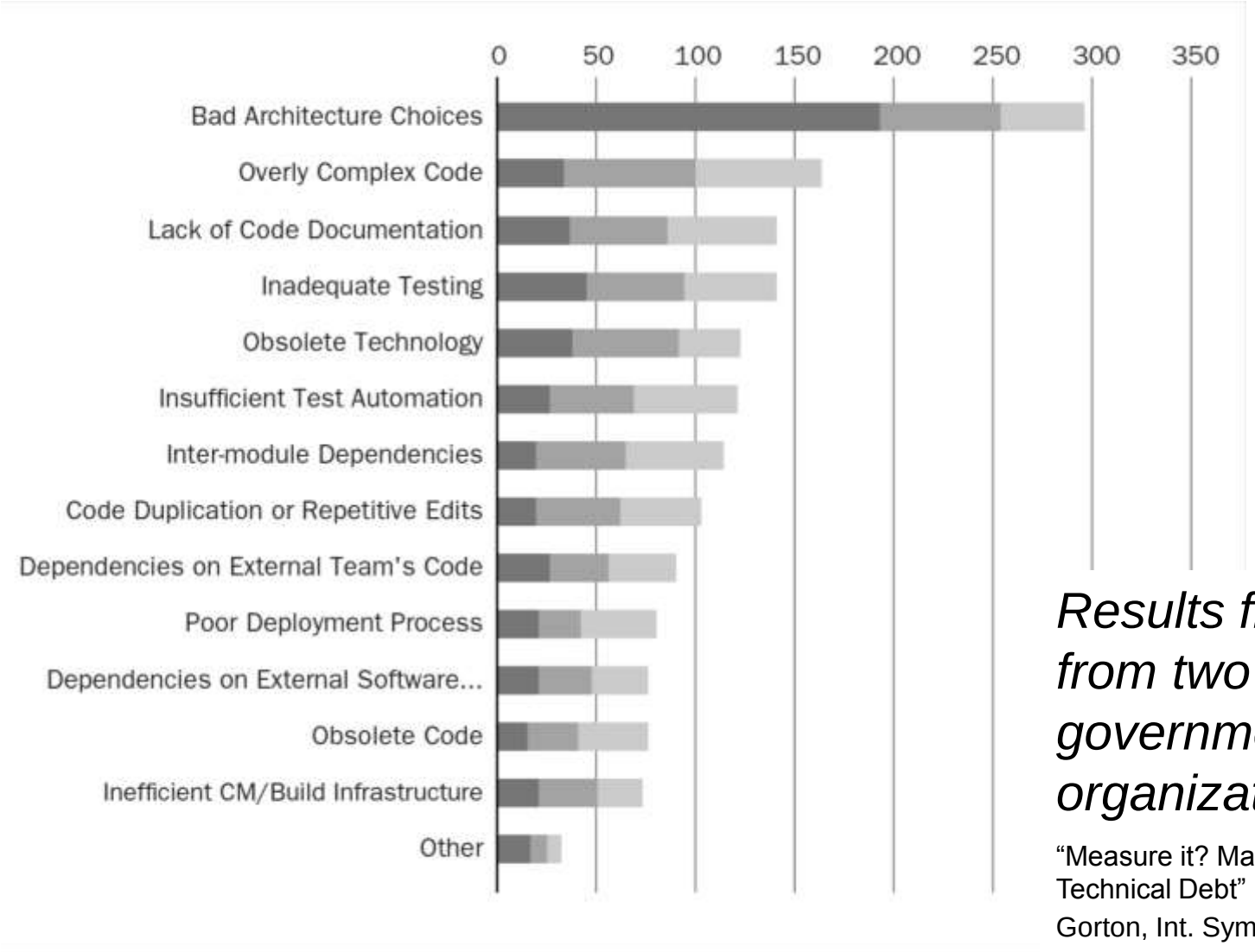
This is only a starting point!

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
- ….



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.

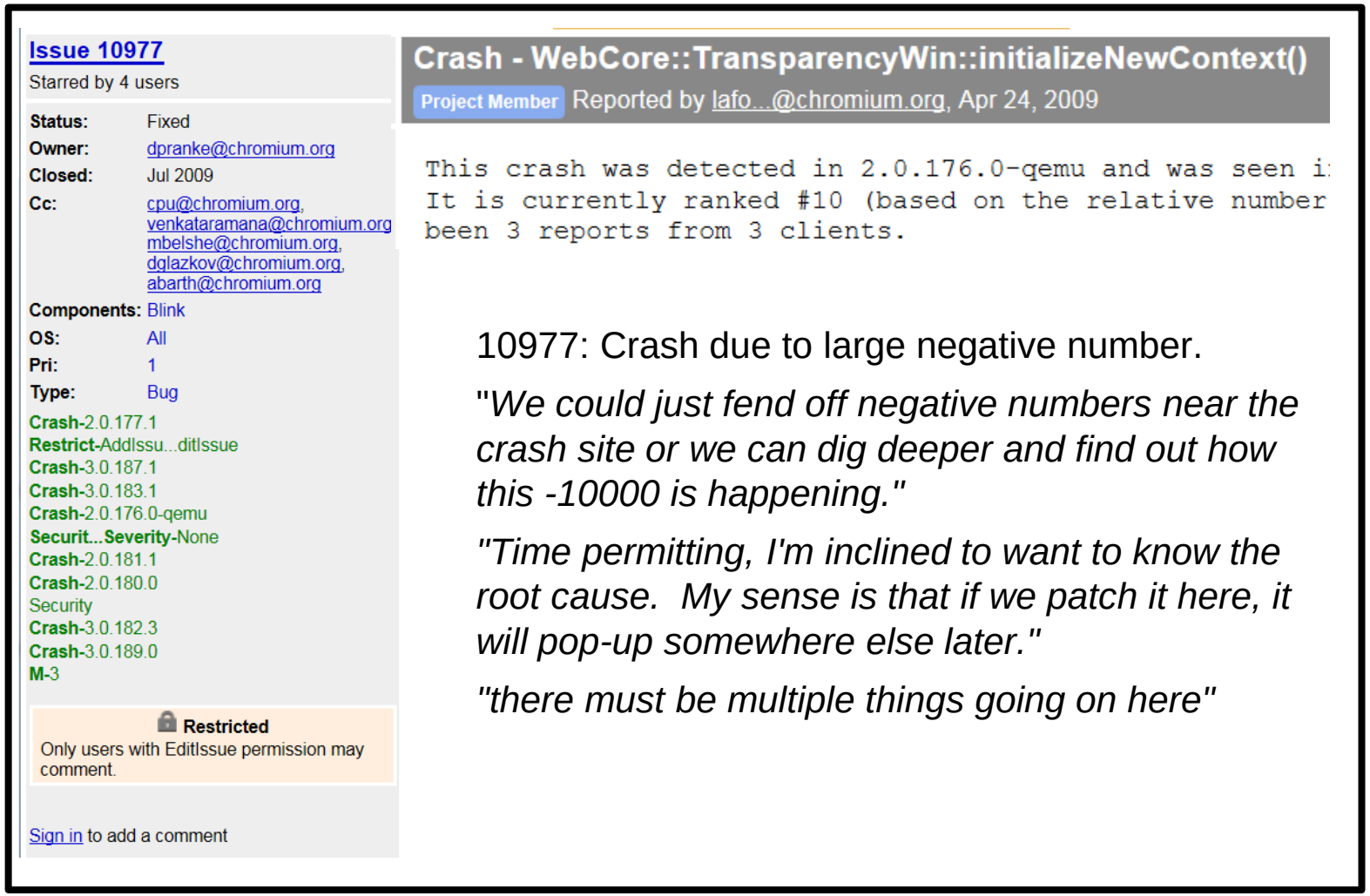
2. Making Technical Debt Visible

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

Communicating Technical Debt

Developers already discuss technical debt, even if not using the vocabulary.



Issue 10977

Starred by 4 users

Status:	Fixed
Owner:	dpranke@chromium.org
Closed:	Jul 2009
Cc:	cpu@chromium.org, venkataramana@chromium.org mbelshe@chromium.org, dglazkov@chromium.org, abarth@chromium.org
Components:	Blink
OS:	All
Pri:	1
Type:	Bug

Crash-2.0.177.1
Restrict-AddIssu...ditIssue
Crash-3.0.187.1
Crash-3.0.183.1
Crash-2.0.176.0-qemu
Securit...Severity-None
Crash-2.0.181.1
Crash-2.0.180.0
Security
Crash-3.0.182.3
Crash-3.0.189.0
M-3

Restricted
Only users with EditIssue permission may comment.

Sign in to add a comment

Crash - WebCore::TransparencyWin::initializeNewContext()

Project Member Reported by lafo...@chromium.org, Apr 24, 2009

```
This crash was detected in 2.0.176.0-qemu and was seen i
It is currently ranked #10 (based on the relative number
been 3 reports from 3 clients.
```

10977: Crash due to large negative number.

"*We could just fend off negative numbers near the crash site or we can dig deeper and find out how this -10000 is happening.*"

"Time permitting, I'm inclined to want to know the root cause. My sense is that if we patch it here, it will pop-up somewhere else later."

"there must be multiple things going on here"

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

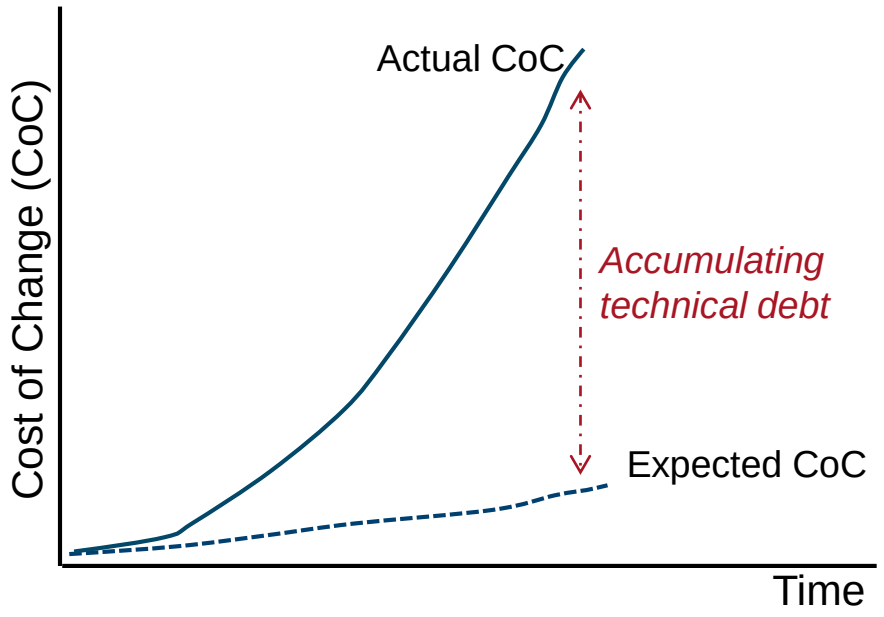
Stephany Bellomo, Robert L. Nord, Ipek Ozkaya, Mary Popeck: Got technical debt?: surfacing elusive technical debt in issue trackers. MSR 2016: 327-338

3. Deciding When and How to Resolve Debt

This is essentially a business decision balancing the cost of remediating debt (making a change) and the cost of accepting debt (no change).

- Cost of remediation is simply the cost of all needed changes
- Cost of accepting debt is more complex
  - Work arounds
  - Delays or inability to add capability
  - Recurring maintenance overhead
  - User satisfaction

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.

Developing 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

Replace

- The cost of reducing the debt far exceeds replacing the system
- Replacing the system has added benefit of aligning with new markets and technology

Commitment to invest, incremental refactoring

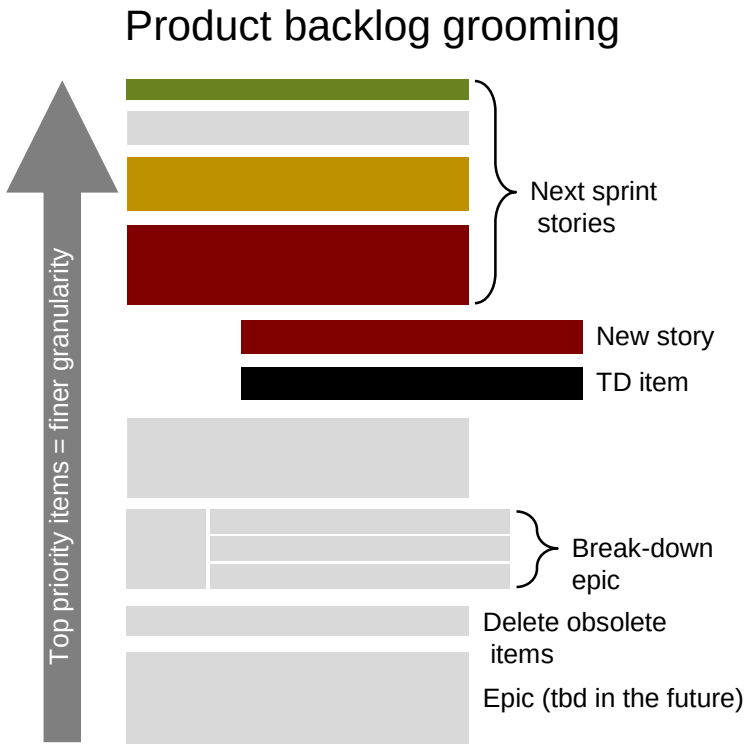
- Refactoring over multiple releases or focused refactoring release
- Balancing new feature development with refactoring effort to continue to generate value

4. 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 Technical Debt Management to Release Planning



Discussion of backlog items should include an explicit focus on any technical debt items.

Resolving technical debt should be an explicit part of planning (allocate one sprint, integrate into multiple sprints, etc.).

Technical debt should be explicitly recorded, similar to new user stories, defects, and the like.

Technical Debt Management within 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 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 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

Ask The Hard Questions Early and Often

What are our dominant sources of debt?

- Start with code, architecture, and technology as areas of investigation

How can debt be visualized with effective tool support?

- Decide on key business metrics and relate them to the product, that will drive what tool, if any, is right
- Request information from contractors; make it part of contracts

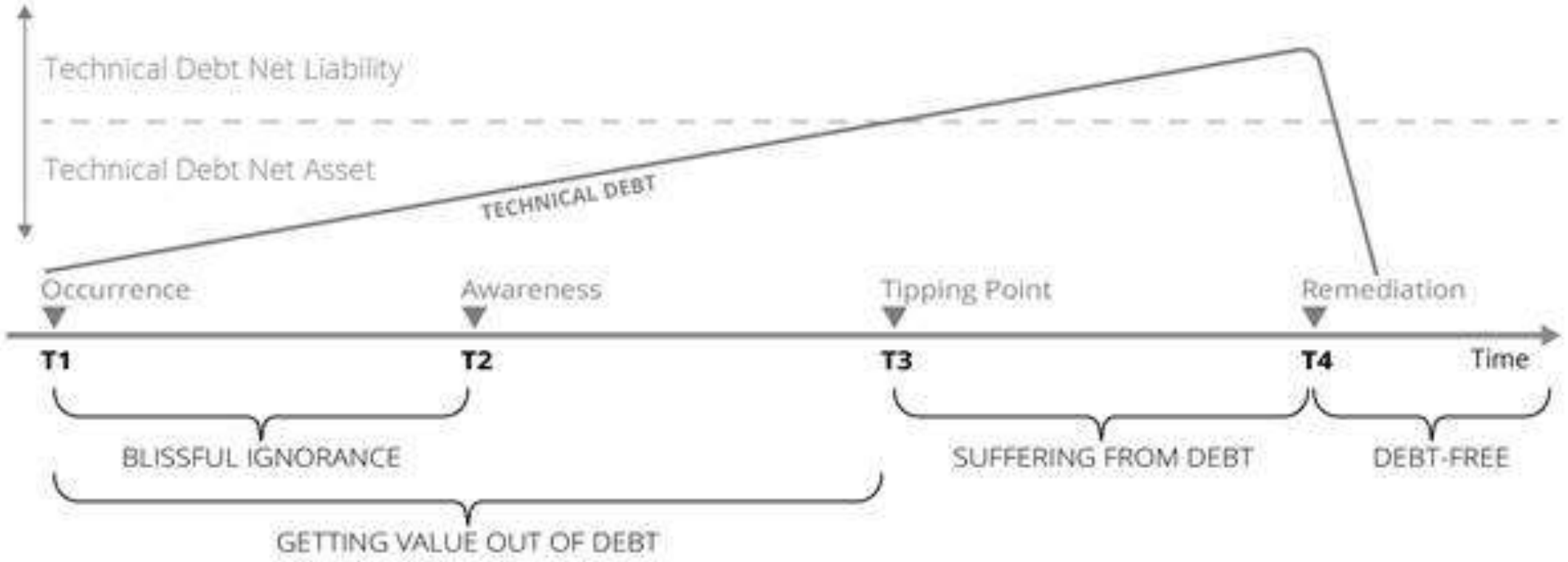
How and when to pay-back debt?

- Make it an ongoing process
- Incentivize development teams to disclose technical debt

How to manage strategic technical debt?

- Optimize for cost of change, both for short-term and long-term

Manage the Technical Debt Timeline



From Folklore to Practice

Building a case to manage debt:

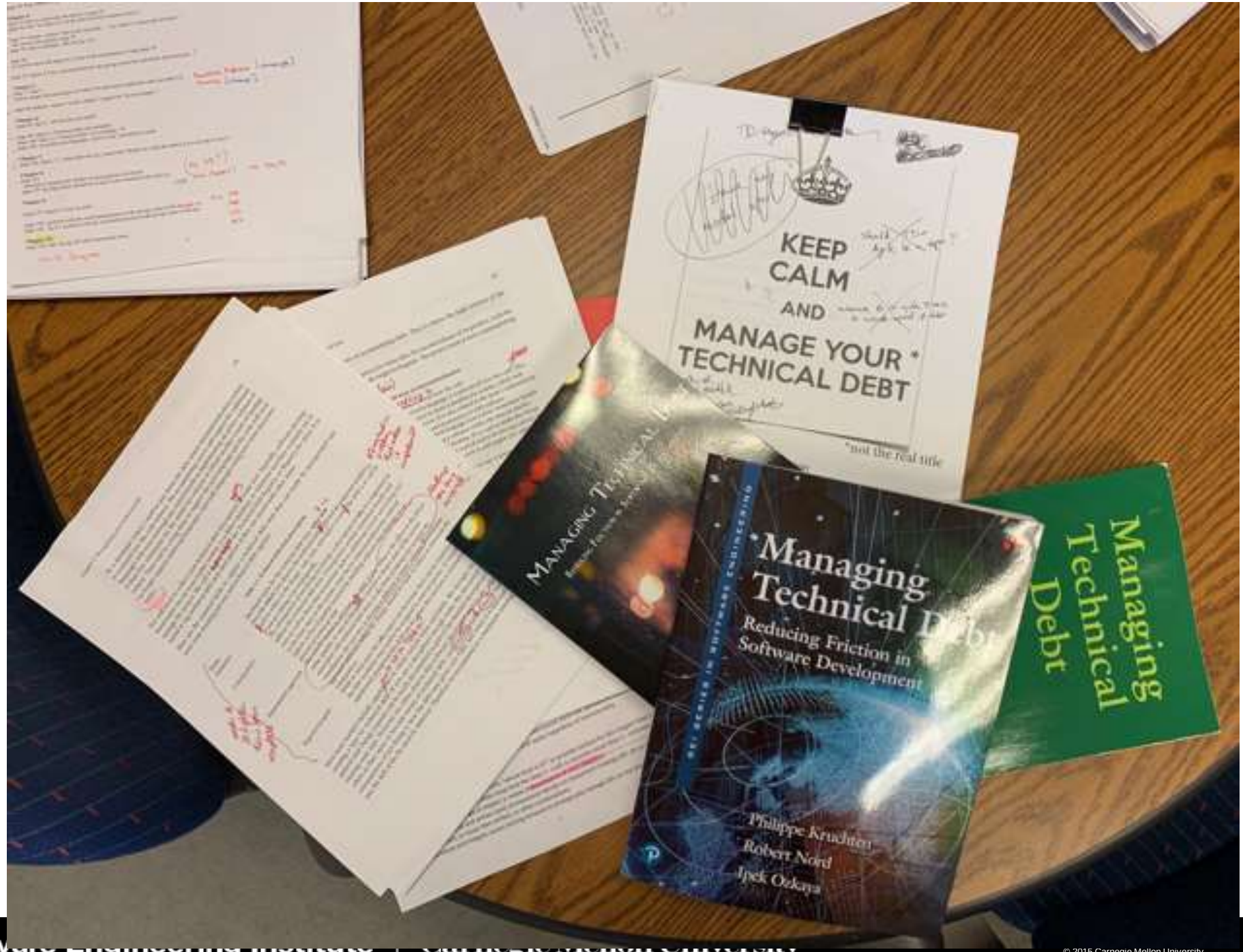
- Can we identify instances of technical debt over the history of the project?
- Do those instances correlate with expected symptoms of accruing interest?

Use data to identify instances:

- Insights into the planned design (from team members).
- Optional: narrative information about the project history (specifically, any major refactorings that have occurred).
- Code repository; optional: architecture documentation.

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.
- Optional: classification of defects / changes by type.
- Effort by change / defect fix.



Questions?

Contact Information

Ipek Ozkaya, PhD
Technical Director
Engineering Intelligent Software Systems
Software Solutions Division
Carnegie Mellon University
Software Engineering Institute

email: ozkaya@sei.cmu.edu

For more on technical debt:
https://www.sei.cmu.edu/research-capabilities/all-work/display.cfm?customel_datapageid_4050=6520
For more on software architecture:
http://www.sei.cmu.edu/architecture/