Using Technical Debt to Improve Software Sustainability and Find Software Vulnerabilities

Ipek Ozkaya

Distribution Statements

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM-0004041

What is Technical Debt?

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 are now 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** effect on system attributes of interest to developers (e.g., increasing defects, negative change in maintainability and code quality indicators).

DoD Perspective of the Problem

By the time the government owns the

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 are now hard to track and have become security nightmares.

lo is

Developers intentionally or unintentionally incur debt

Developers recognize, but do not declare or fix the debt

An optimal time to rearchitect or refactor the system passes

technical debt is used strategically and declare at acquisition time

1. time technical debt is incurred
2. time technical debt is recognized
3. time to plan and re-architect
4. time until debt is actually paid-off
5. continuous monitoring

Our goal is to enable better sustainment decision making through technical debt analytics

- What indicators signify major contributors to technical debt?
- Are software components with accrued technical debt more likely to be vulnerability-prone?
- Can we build correlations between these indicators and project measures, such as defects, vulnerabilities and change proneness?

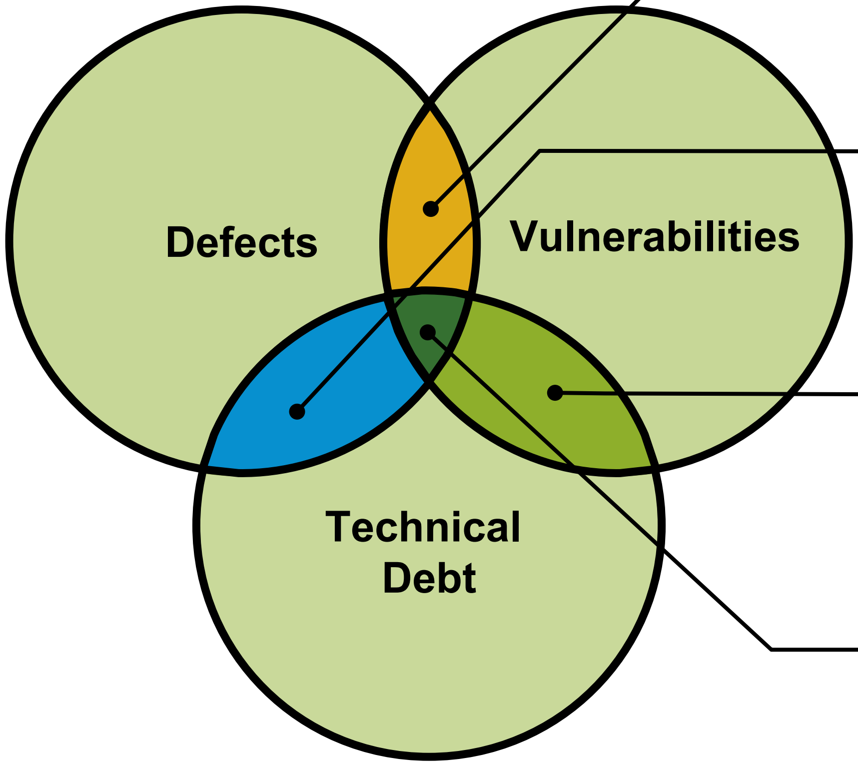
Why do we need a new term?

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



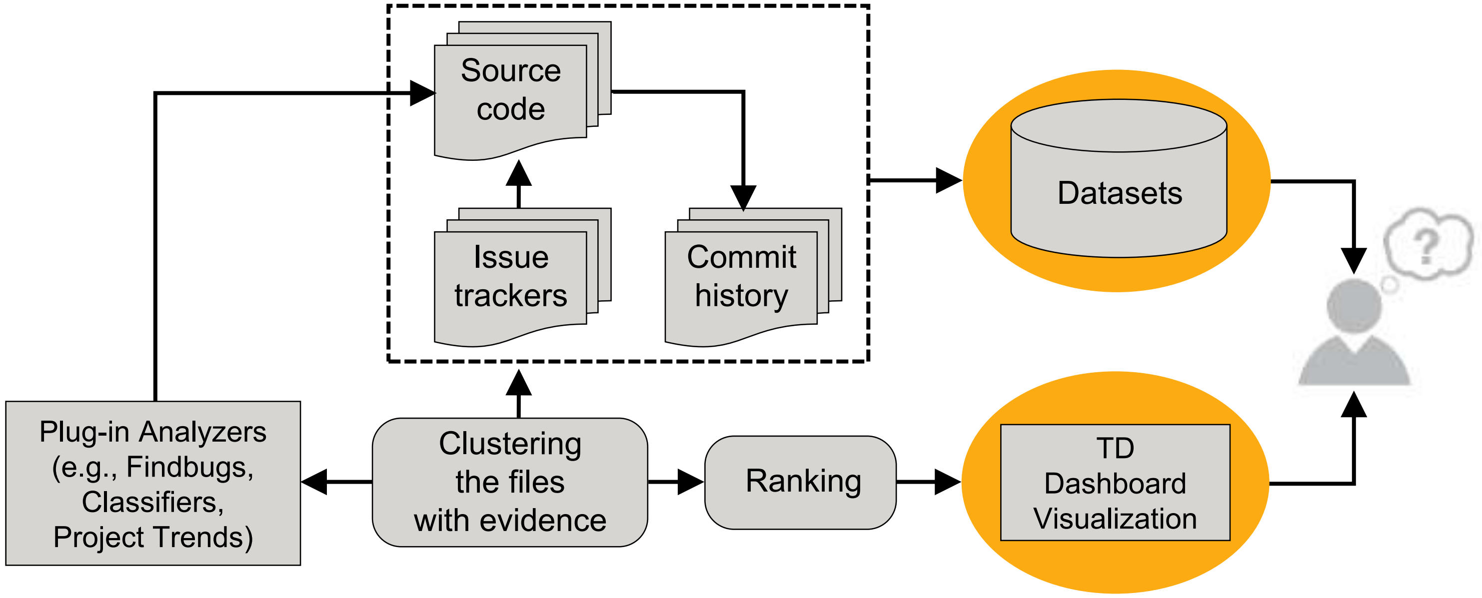
Defect proneness implies increased vulnerability risks.

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

Technical debt increases vulnerability risks.

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

Towards Technical Debt Analytics



- Extracting evidence from the issue trackers
- Extracting evidence from code and commit history
- Holistic analysis

Extracting Evidence from the Issue Trackers

Do Issue Trackers Reveal Technical Debt?

	Data set	Source	Filter criteria	# Records analyzed
Technical debt classification, analysis, and evaluation **Total: 727 issues**	Connect	Jira	March 2012	286
	Project A	Jira	Defects/CRs Sep. 2010 to Dec. 2014	86
	Project B	FogBugz	All year 2013	193
	Chromium	Google issue tracker	M(ilestone): 48 Stars (watchers) > 3	163

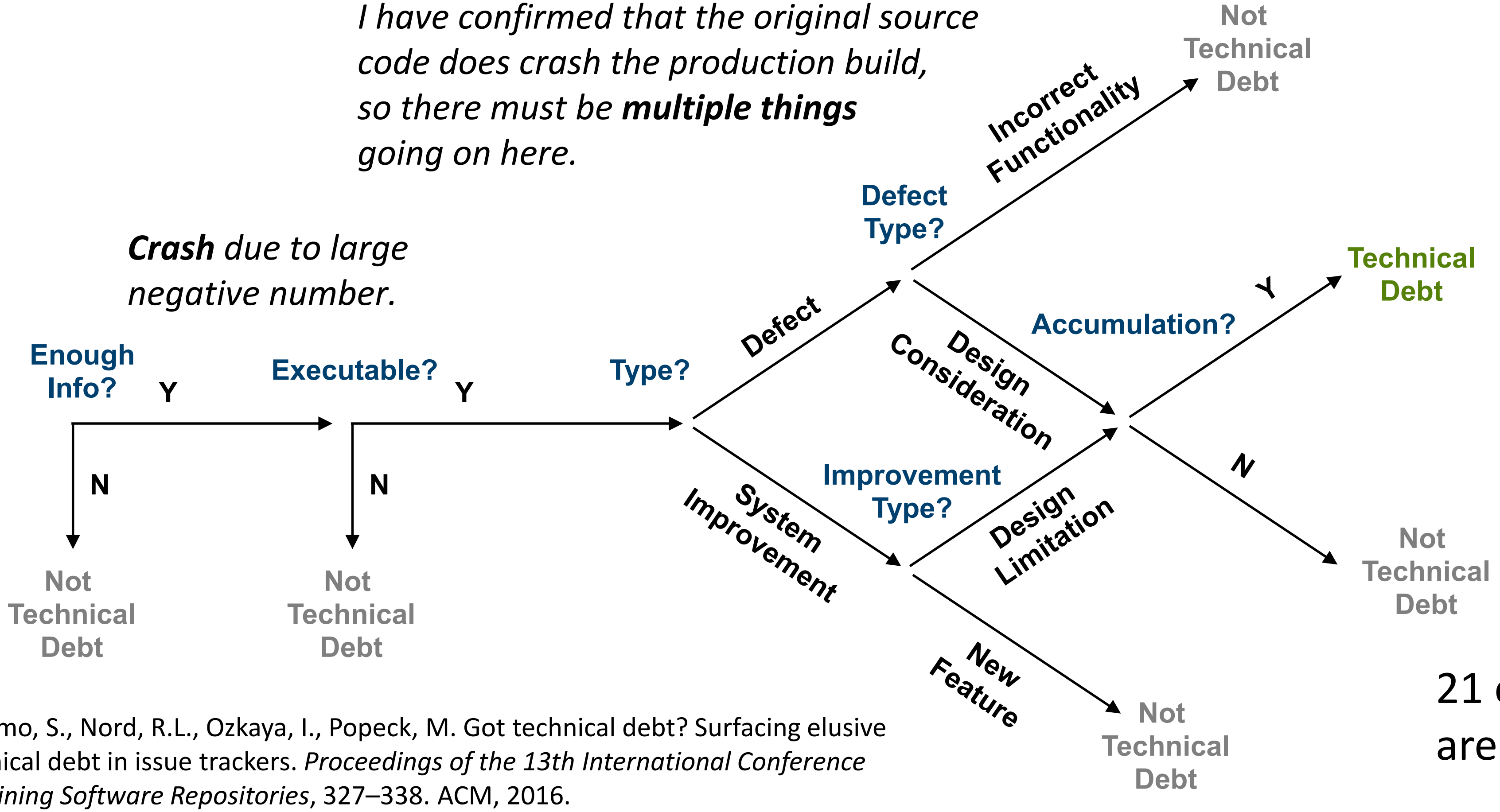
- Do developers use the term *technical debt* explicitly when discussing issues and tasks in their issue trackers?
- Can technical debt items be discovered systematically within issue trackers?
- What are the distinguishing characteristics of technical debt items discovered in issue trackers?

Indicator: Technical Debt Tag

*Time permitting, I'm inclined to want to know the **root cause**.*

*I have confirmed that the original source code does crash the production build, so there must be **multiple things** going on here.*

***Crash** due to large negative number.*



*There have been **28 reports** from 7 clients… **18 reports** from 6 clients*

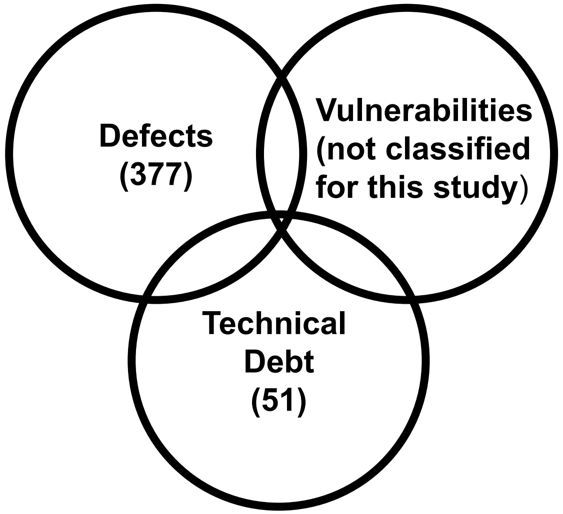
*My sense is that if we patch it here, it will **pop-up somewhere else later**.*

*hmm ... **reopening**. the test case crashes a debug build, but not the production build.*

21 of 79 issues labeled security are classified as technical debt.

Bellomo, S., Nord, R.L., Ozkaya, I., Popeck, M. Got technical debt? Surfacing elusive technical debt in issue trackers. *Proceedings of the 13th International Conference on Mining Software Repositories*, 327–338. ACM, 2016.

Technical Debt in Issue Trackers



Deployment & Build	Out-of-sync build dependencies	3	CN
	Version conflict	1	CN
	Dead code in build scripts	1	CN
Code Structure	Event handling	5	2CH, 3PB
	API/Interfaces	5	2CH, 1CN, 2PB
	Unreliable output or behavior	5	4CH, 1PA
	Type conformance issue	3	CN
	UI design	3	PB
	Throttling	2	1CH, 1PB
	Dead code	2	CN
	Large file processing or rendering	2	CH
	Memory limitation	2	CH
	Poor error handling	1	PA
	Performance appending nodes	1	CH
	Encapsulation	1	PB
	Caching issues	1	CN
Data Model	Data integrity	6	PA
	Data persistence	3	PB
	Duplicate data	2	PA
Regression Tests	Test execution	1	CH
	Overly complex tests	1	CH

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

Extracting Evidence from Code and Commit History

Combined Rules for Detecting Technical Debt

Code rules:

Duplicate code

Out of sync versions

Out of sync build dependencies

Dead code

Architectural rules (design flaws):

Dependency propagation

Test coverage

Cross-module cycles

Cross-package cycles

Unstable interface

Detecting Technical Debt = We are making the implicit statement that "As these TD issues stay in the system they are more likely to cause more bugs and will cost more to fix later. Not all issues will cost more to fix later."

Analysis: Design Flaws

Co-existence of different types of design flaws correlates with the presence of vulnerabilities.

# Types of Design Flaws	Non-vuln files	Vuln files	% have vulns.
0	8544	47	0.5%
1	7357	141	2%
2	2345	91	4%
3	194	10	5%

R. L. Nord, I. Ozkaya, E. J. Schwartz, F. Shull, R. Kazman: Can Knowledge of Technical Debt Help Identify Software Vulnerabilities? CSET @ USENIX Security Symposium 2016

Holistic Analysis

Qualitative and Quantitative Analysis

Chromium security issues

Classifying TD from Issues labeled Security

Detecting Design Flaws in Code

	Not TD	TD
Design Flaws	50	15
No Design Flaws	8	6

79 issues are labeled security

- 21 are classified as technical debt
- 65 trace to files containing design flaws

Partial Evidence

Detecting Design Flaws in Code

Classifying TD from Issues labeled Security

	Not TD	TD
Design Flaws	**50** Defect: 26 Feature: 1 Design Problem: 23	15
No Design Flaws	8	6

67577: *"This is a 2-liner. I'll take it, if only to get our rampant security bug list down by one."*
Flaw: modularity violation

64108: *"feature was never fully implemented, we may not have put in proper checks to prevent this."*
Flaws: modularity violation, cycle

Supplement Static Analysis with Developer Knowledge

Classifying TD from Issues labeled Security

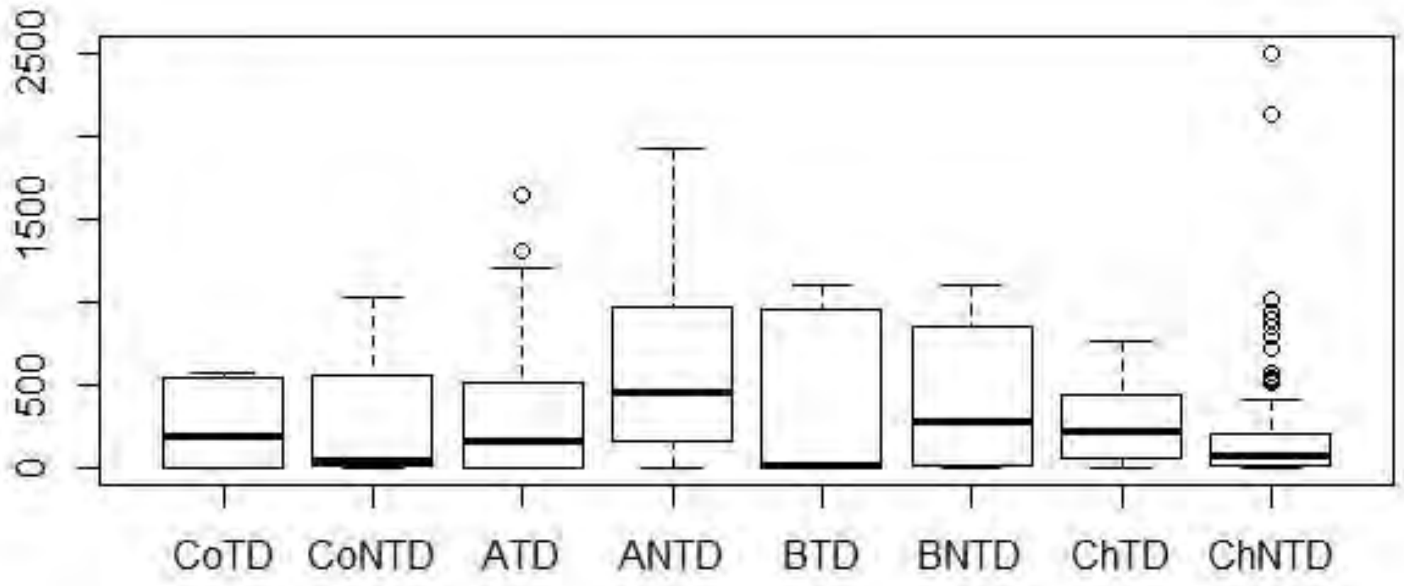
Detecting Design Flaws in Code

	Not TD	TD
Design Flaws	50	15
No Design Flaws	8	6

10977: “we could just fend off … or we can dig deeper” “if we patch it here, it will pop-up somewhere else later”

Are there any quantifiable characteristics?

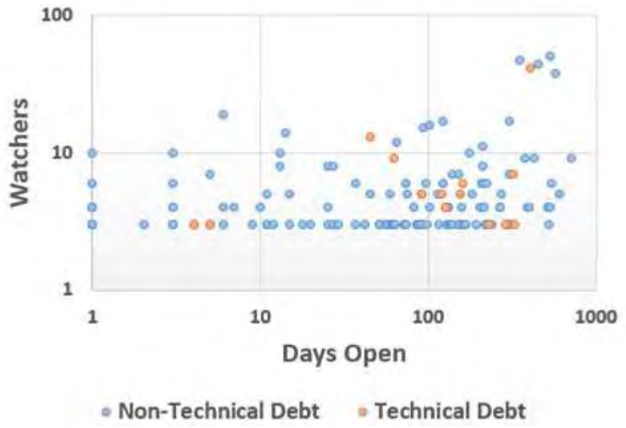
Time to close



Issue priority

	Priority 1	Priority 2	Priority 3
Technical Debt Issues	22%	56%	22%
Not Technical Debt Issues	24%	50%	26%

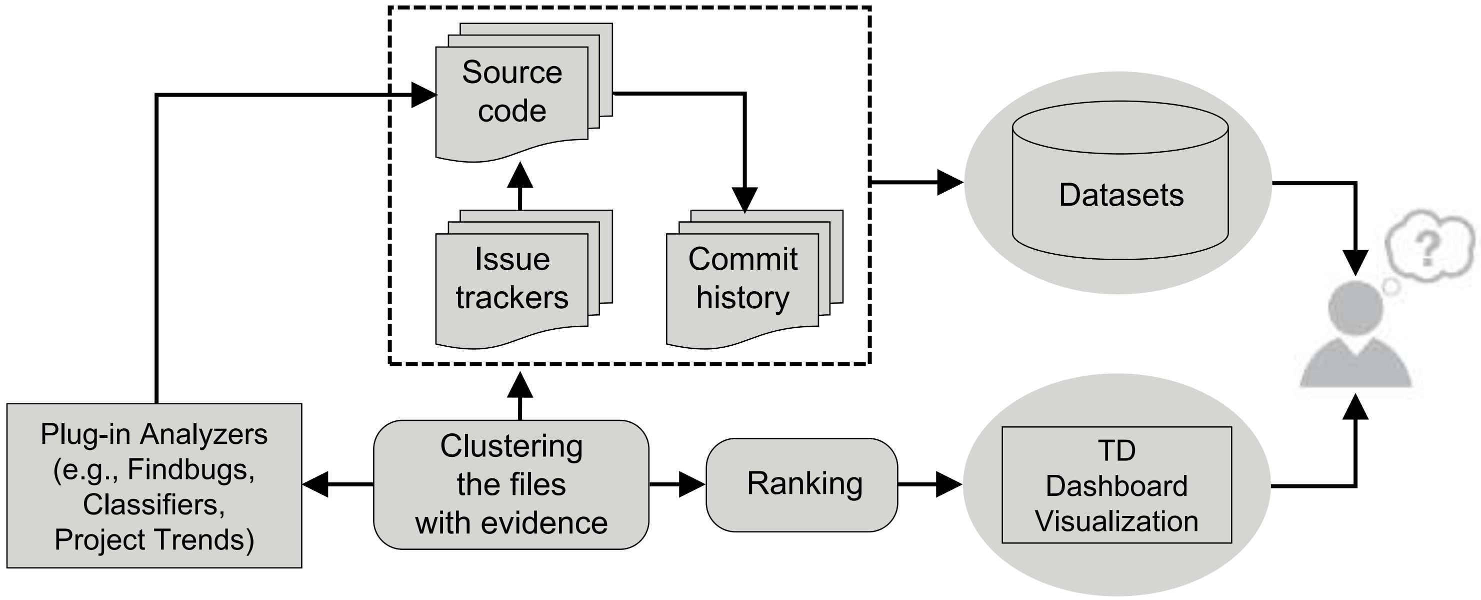
Developer discussion



Summary Findings

- **Design areas** with greater density of technical debt provide significant opportunities for improvement.
- The issues we find are **mostly the result of unintentional design choices**.
- **Correlations** between vulnerabilities and technical debt indicators warrant further research that combines multiple artifacts in analysis.
- Technical debt can be made **visible earlier** when tracked similarly to defects, consequently managed more effectively and strategically.

Towards Technical Debt Analytics



Rank TD items
- Identify relative number of defects, change and bug churn and locations in the code base that require changes.
- Create an initial ranking.

Create a technical debt classifier
- Apply topic modeling algorithms to issue tracker data sets to extract topics related to accumulating rework
- Extract categories of TD related design

Correlate analysis rules with TD topics
- Identify recurring design concepts, their mappings to code analysis rules and their interrelationships
- Run code analyzers to detect quality violations to identify candidate TD items

Consolidate TD items
- Run criteria for consolidations and extract impacted additional files with related violations.

The Technical Debt Community

Role	Impact our research by contributing	Impact your organizational practices
DoD PM, sustainment professionals	Challenge problems, project measures	Ask targeted questions earlier, ask for evidence based on our approach
Defense contractors	Data, feedback, validation of techniques	Invest in secure and maintainable practices, use our approach
Industry	Data, feedback, validation of techniques	Incentivize teams to identify sources of technical debt
Tool vendors	Transition partner	Extend tools to label and analyze technical debt items
Researchers, students, PIs	Technical validity	Extend/challenge our approach, extend, use, and challenge our data sets

Relevant SEI Published Work

R. L. Nord, I. Ozkaya, E. J. Schwartz, F. Shull, R. Kazman: Can Knowledge of Technical Debt Help Identify Software Vulnerabilities? CSET @ USENIX Security Symposium 2016

S. Bellomo, R. L. Nord, I. Ozkaya, M. Popeck: Got Technical Debt? Surfacing Elusive Technical Debt in Issue Trackers, to appear in proceedings of Mining Software Repositories 2016, collocated @ICSE 2016.

R. L. Nord, R. Sangwan, J. Delange, P. Feiler, L, Thomas, I. Ozkaya: Missed Architectural Dependencies: The Elephant in the Room, WICSA 2016.

P. Avgeriou, P. Kruchten, R. L. Nord, I. Ozkaya, C. B. Seaman: Reducing Friction in Software Development. IEEE Software Future of Software Engineering Special Issue 33(1): 66-73 (2016)

L. Xiao, Y. Cai, R. Kazman, R. Mo, Q. Feng: Identifying and Quantifying Architectural Debts, ICSE 2016.

N. A. Ernst, S. Bellomo, I. Ozkaya, R. L. Nord, I. Gorton: Measure it? Manage it? Ignore it? software practitioners and technical debt. ESEC/SIGSOFT FSE 2015: 50-60

Managing Technical Debt Research Workshop Series 2010-2016
https://www.sei.cmu.edu/community/td2016/series/

Team

SEI team members

- Ipek Ozkaya, PhD, SSD (PI)
- Rod Nord, PhD, SSD (PI)
- Stephany Bellomo, MSc., SSD
- Neil Ernst, PhD, SSD
- Rick Kazman, PhD, SSD
- Ed Schwartz, PhD, CERT
- Forrest Shull, PhD, SSD/ERO
- Robert Stoddard, SSD

Research Collaborators

- Philippe Kruchten, PhD, Univ. of British Columbia, funded
- Raghu Sangwan, PhD, Penn State, funded
- Managing Technical Debt research community
- Industry, DoD, and tool vendor partners

Contact Information

Ipek Ozkaya
Principal Researcher
SSD/SEAP
Telephone: +1 412 268 3551
Email: ozkaya@sei.cmu.edu

Managing Technical Debt Project
http://www.sei.cmu.edu/architecture/research/arch_tech_debt/