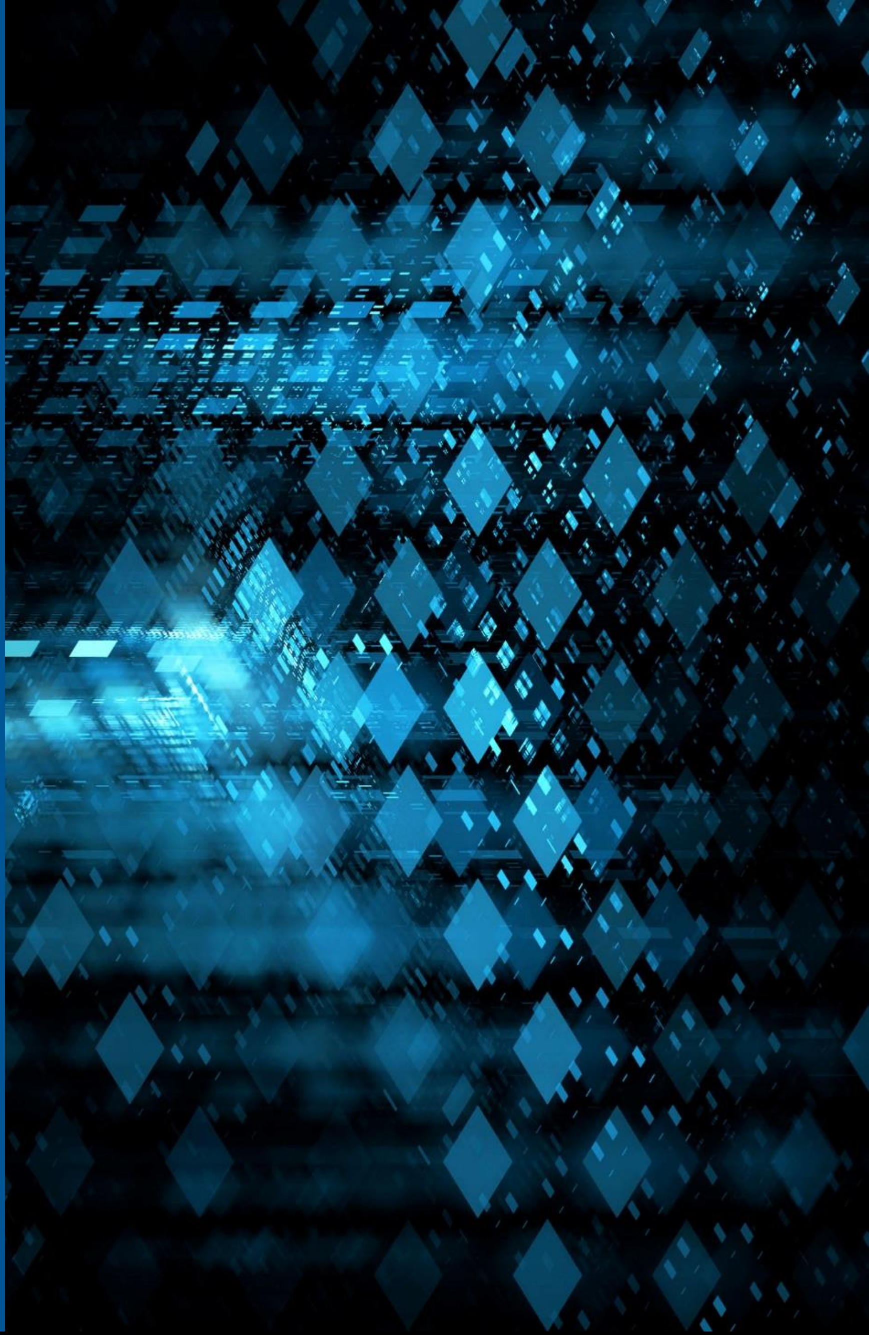
Software Vulnerabilities, Defects, and Design Flaws: A Technical Debt Perspective

Robert L. Nord, Ipek Ozkaya, Forrest Shull

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213



Is technical debt real?

Popular media is recognizing major software failures as technical debt.

- United Airlines failure *(July 8, 2015, “network connectivity”)*
- New York Stock Exchange glitch *(July 8, 2015, “configuration issue”)*
- Healthcare.gov *(February 2015, “users cannot access functionality”)*

Researchers conservatively estimate $361,000 of technical debt / 100 KLOC as the cost to eliminate structural-quality problems that seriously threaten an application’s business viability.

Are we being fooled by scare tactics?

How do we understand the real problem, and why should we care?

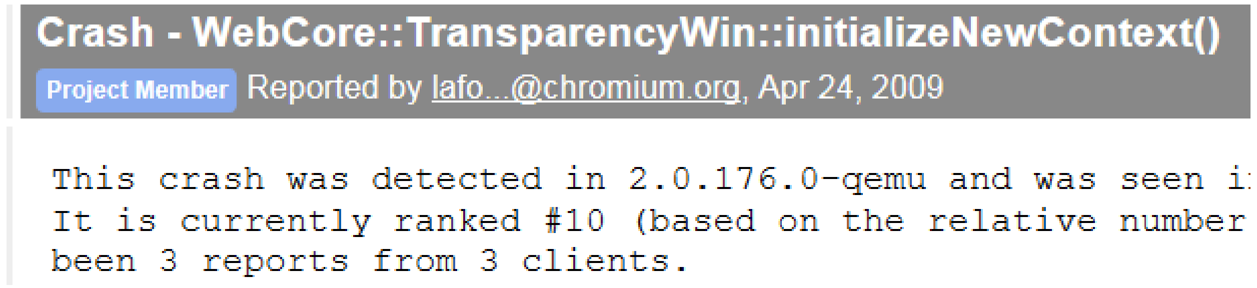
Technical Debt Defined

Our legacy software has code without exception handling, which made sense for lower capacity processors, today we can’t find and track these issues. These areas in the code have become 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).

Technical Debt in Security Issues



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 have been 28 reports from 7 clients… 18 reports from 6 clients.”

“Hmm ... reopening. The test case crashes a debug build, but not the production build. I have confirmed that the original source code does crash the production build, so there must be multiple things going on here.”

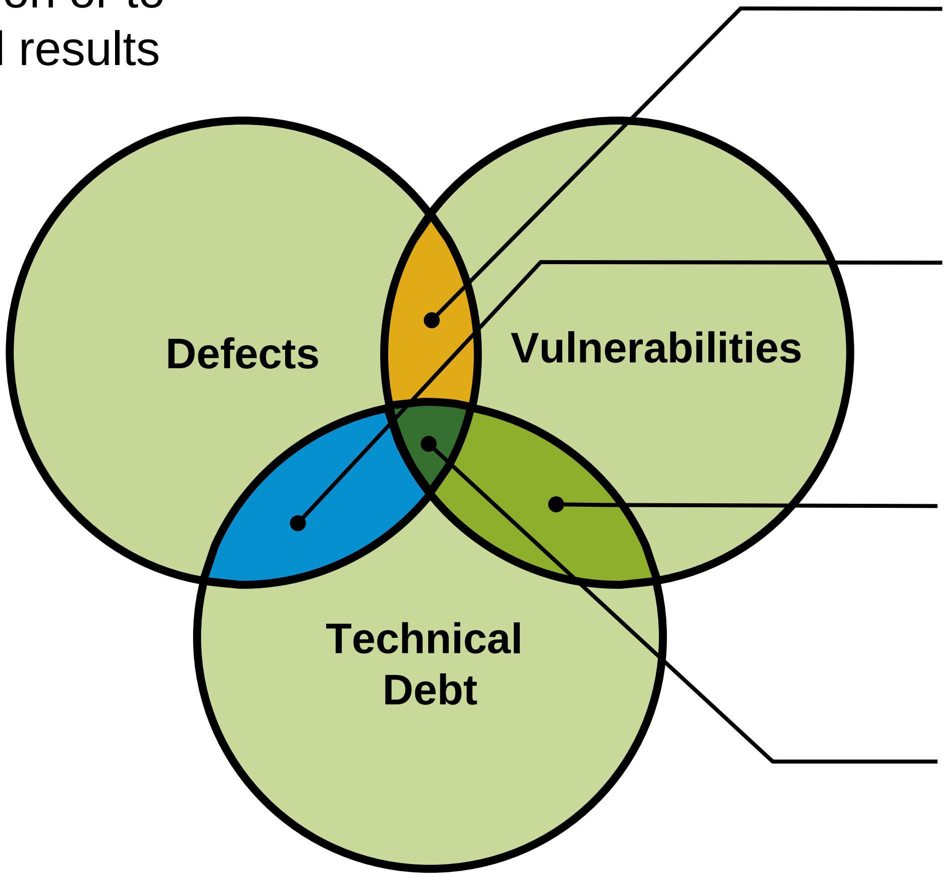
Impacts of Technical Debt

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



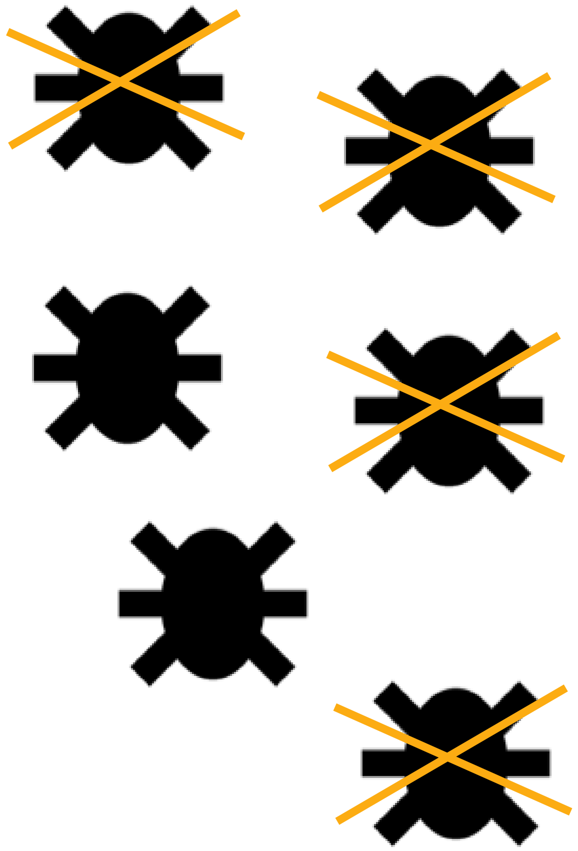
Defect proneness implies increased vulnerability risks.

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!

Misconception: Eliminating defects & vulnerabilities eliminates technical debt



This view suffers from the following shortcomings:

- Focuses only on customer-visible, functional aspects of system problems
- Results in overlooking underlying contributors to defects and vulnerabilities that can be design issues
- Fails to recognize accumulating interest of technical debt that defects and vulnerabilities might be signaling

Correction: Defects and vulnerabilities are key symptoms of technical debt

Defects and vulnerabilities, especially recurring ones that have been open for a long time or reopen and that accumulate around particular aspects of the system, are symptoms of technical debt to address.

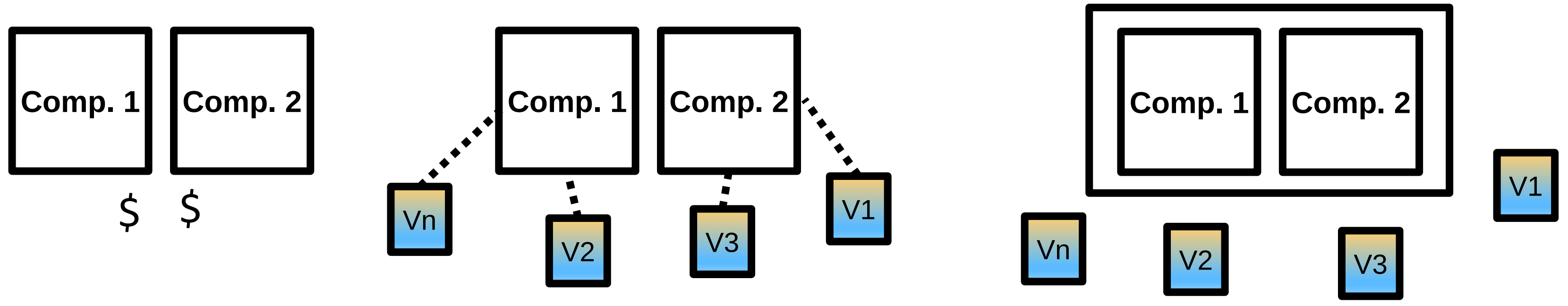
The quantity of resources and processes that go into defect and vulnerability management indicates the accumulating side effects of technical debt.

Question

Design Time
(debt introduced)

Operation Time
(are there defects & vulnerabilities?)

Maintenance / Evolution Time
(will fixing debt fix defects & vulnerabilities?)



Are software components with accrued technical debt more likely to be defect and vulnerability defect prone?

Technical Debt Indicators: Design flaws

Technical debt examples

"We have a model-view controller framework. Over time we violated the simple rules of this framework and had to retrofit many functions later."

Modularity violation, pattern conformance

"There were two highly coupled modules that should have been designed separately from the beginning"

Modularity violation, pattern conformance

"A simple API call turned into a nightmare [due to not following guidelines]"

Framework, pattern conformance

Example design flaws:

Unstable Interface

Modularity Violation

Improper Inheritance

Cycle

Xiao, L., Cai, Y., Kazman, R. Design rule spaces: A new form of architecture insight. *Proceedings of the 36rd International Conference on Software Engineering*, 967–977. ACM, 2014.

Indicator: Design Flaws - 1

<u>Unstable Interface</u>, Modularity Violation, Improper Inheritance, Cycle

		1	2	3	4	5	6	7	8	9	10	11
1	ui.gfx.size.cc	(1)	Use,3	,2	**,3**	,3	,1		,1	,2		
2	ui.gfx.size.h	Call,3	(2)	,5	**,4**	,2		,1	,2	,1	,1	
3	ui.gfx.point.h	,2	,5	(3)	**,5**	,3		,1	,1	,2	,1	,1
4	ui.gfx.rect.h	**Call,3**	**Call,4**	**Call,5**	**(4)**	Call,6	,2	,2	,2	,5	,2	,2
5	ui.gfx.rect.cc	Call,3	Call,2	Call,3	**Call,6**	(5)	,1	,1	,1	,3	,1	,2
6	webkit.plugins.ppapi.ppapi_plugin_instance.cc	Call,1	Call,	Call,	**Call,2**	Call,1	(6)	,1	,5	,2	,2	,2
7	content.renderer.paint_aggregator.cc		Call,1	Call,1	**Call,2**	Call,1	,1	(7)	,2	,2	,2	,1
8	content.renderer.render_widget.cc	Call,1	Call,2	Call,1	**Call,2**	Call,1	Call,5	Call,2	(8)	,3	,1	,1
9	ui.gfx.rect_unittest.cc	,2	Call,1	,2	**Call,5**	Call,3	,2	,2	,3	(9)	,2	,2
10	webkit.plugins.webview_plugin.cc		,1	,1	**Call,2**	,1	,2	,2	,1	,2	(10)	,1
11	ui.gfx.blit.cc		Call,	Call,1	**Call,2**	Call,2	,2	,1	,1	,2	,1	(11)

Xiao, L., Cai, Y., Kazman, R. “Design rule spaces: A new form of architecture insight.”
Proceedings of the 36rd International Conference on Software Engineering, 967–977. ACM, 2014.

Indicator: Design Flaws - 2

Unstable Interface, Modularity Violation, Improper Inheritance, Cycle

Shared secret between files

Should be extracted as design rules

		1	2
1	ContextConfig.java	(1)	**,31**
2	TldConfig.java	**,31**	(2)

Xiao, L., Cai, Y., Kazman, R. “Design rule spaces: A new form of architecture insight.”
Proceedings of the 36rd International Conference on Software Engineering, 967–977. ACM, 2014.

Analysis

	Classified Not TD	Classified TD
No Design Flaw	8	6
Design Flaw	50	15

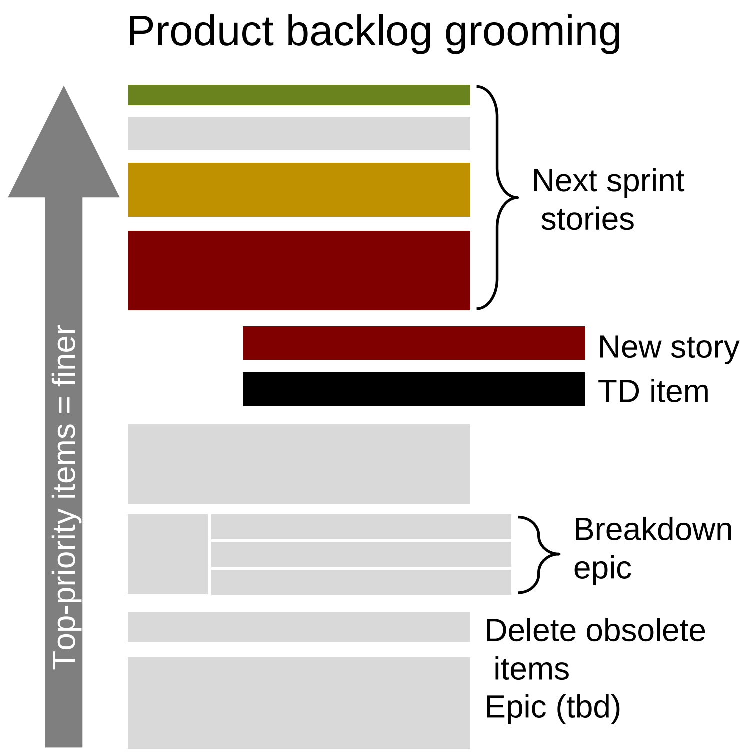
Not all files with design flaws demonstrate technical debt.

Project	Bug/Design Flaw Correlation	Change/Design Flaw Correlation	Sec Bug/Design Flaw Correlation
Chrome	0.987	0.988	0.979

Increased rates of design flaws and code churn are strongly correlated with increased rates of security bugs. The more types of design flaws a file is involved in, the higher the likelihood that it also has vulnerabilities; files with vulnerabilities tend to have more code churn.

Affinity	CWE	#Issues
interface	200: Information Exposure	1
resource arbitration	362: Concurrent Execution using Shared Resource with Improper Synchronization	3
	400: Uncontrolled Resource Consumption	3
invalid result	20: Improper Input Validation	2
	451: User Interface (UI) Misrepresentation of Critical Information	2
	476: NULL Pointer Dereference	1
	704: Incorrect Type Conversion or Cast	1
	825: Expired Pointer Dereference	1
boundary conditions	125: Out-of-bounds Read	1
	703: Improper Check or Handling of Exceptional Conditions	4
	787: Out-of-bounds Write	2
privilege	250: Execution with Unnecessary Privileges	2
	269: Improper Privilege Management	1
	285: Improper Authorization	1

One simple practice



Regardless of the SDLC process, technical debt needs to be explicitly managed:

- Technical debt items should be explicitly recorded, similar to new user stories, defects, and the like.
- Discussion of the items in the backlog should include an explicit focus on any technical debt items to include or postpone.
- Resolving technical debt should be part of planning (allocate one sprint, integrate into multiple sprints, etc.).

Start today!

- Make architecture features and technical debt visible. Incentivize teams, contractors, acquisition personnel to communicate the debt clearly.
- Differentiate strategic technical debt from technical debt that emerges from low code quality.
- Invest in tools and techniques that help elicit and track leading indicators.
- Engage business and technical staff in making tradeoffs.
- Integrate technical debt management into standard operating procedures (e.g., planning, reviews, retrospectives, risk management).

Further Resources

N. Ernst, S. Bellomo, I. Ozkaya, R. L. Nord: What to Fix? Distinguishing between design and non-design rules in automated tools, International Conference on Software Architecture, 2017.

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/td2017/series/

Technical Debt Publications and other resources available at
http://www.sei.cmu.edu/architecture/research/arch_tech_debt/arch_tech_debt_library.cfm

Contact Information

Robert Nord rn@sei.cmu.edu

Ipek Ozkaya ozkaya@sei.cmu.edu

Forrest Shull fshull@sei.cmu.edu

URL: http://www.sei.cmu.edu/architecture/research/arch_tech_debt/