Software Solutions Symposium 2017

March 20–23, 2017

The Relationship Between Design Flaws and Software Vulnerabilities: A Technical Debt Perspective

Robert Nord and Ipek Ozkaya

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

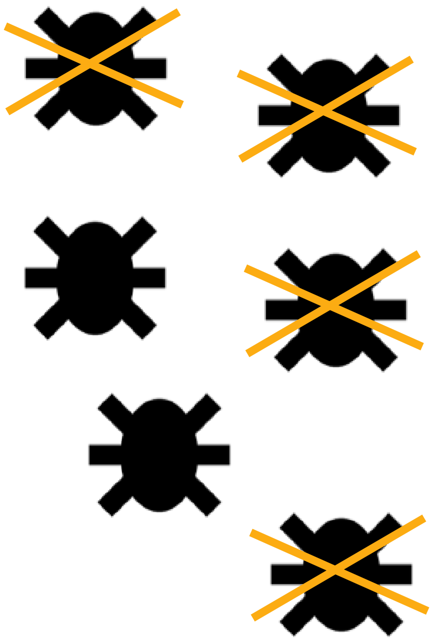
"*We could just fend off negative numbers near the crash site or we can dig deeper and find out how this -10000 is happening.*"

"*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."

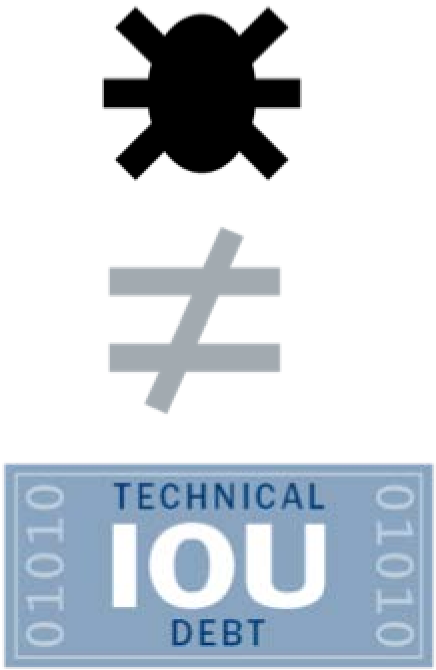
Misconception: Eliminating defects eliminates technical debt



This view suffers from the following shortcomings:

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

Correction: Defects are key symptoms of technical debt



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

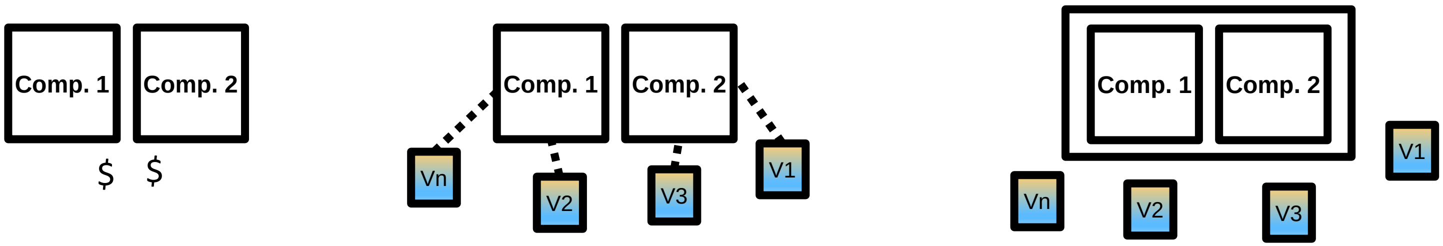
The quantity of resources and processes that go into defect management indicates the accumulating side effects of technical debt.

Question

Design Time
(debt introduced)

Operation Time
(are there vulnerabilities?)

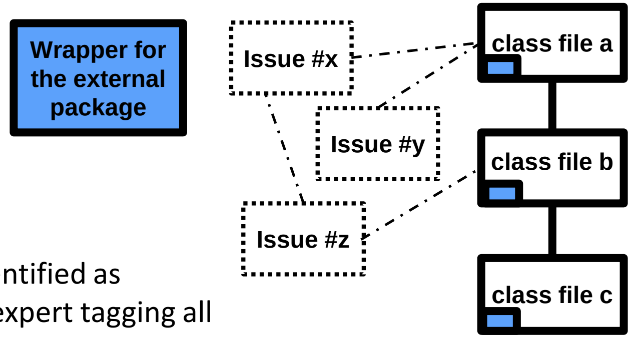
Maintenance / Evolution Time
(will fixing debt fix vulnerabilities?)



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

Design Root Cause

A **technical debt–aware**, graph-based **data structure** representing a view of the system enriched with information from multiple software development artifacts



Current approach: Run static analysis to identify coding violations.

Vision: Enrich with architectural information.

All these files have integer overflows that cause crashes. One of the files participates in an architecture violation (cross-module cycles, improper hierarchy).

Developers create a patch every place they see the similar integer overflow issue.

The improper hierarchy also has a causal relationship with bug churn; hence it represents technical debt.

Multiple issues identified as technical debt by expert tagging all related to crash based on integer overflow, resulting in a patch.

Root cause of the integer overflow is thought to be caused by an external package.

16 files participate in the original problem.

Identifying the design cause brings in 8 more that provide a more accurate picture of impact – total bug and change churn impact increases by at least 30%.

Example Data Set

Chromium version:	17.0.963.46
Released:	February 8, 2012
Files:	18,730; 11k files with bugs 289 files with vulnerabilities
Issue range:	Feb 1, 2010 – Feb 8, 2012
Issues:	#bug: 14k; #security: 79

Chromium project

- Began in 2008
- Complex web-based application that operates on sensitive information and allows untrusted input from both web clients and servers.

Approach

Identify software vulnerabilities
- security label
- identify indicator from issue: CWE
- trace to file

Identify technical debt
- apply classification rules to issues
- extract design problem and rework from issues
- trace to file
- indicator from file: bugs and churn
- indicator from file: design flaws

Model relationships
- design concepts
- technical debt indicators

Test for correlations between technical debt prone files and files with known vulnerabilities.

Issue
Name
Status
Priority
Label: Security, Impact, Severity
Type: Bug, Bug-Security

Commit History	Code File
Issue	Name
Code	LOC
Version history	Age

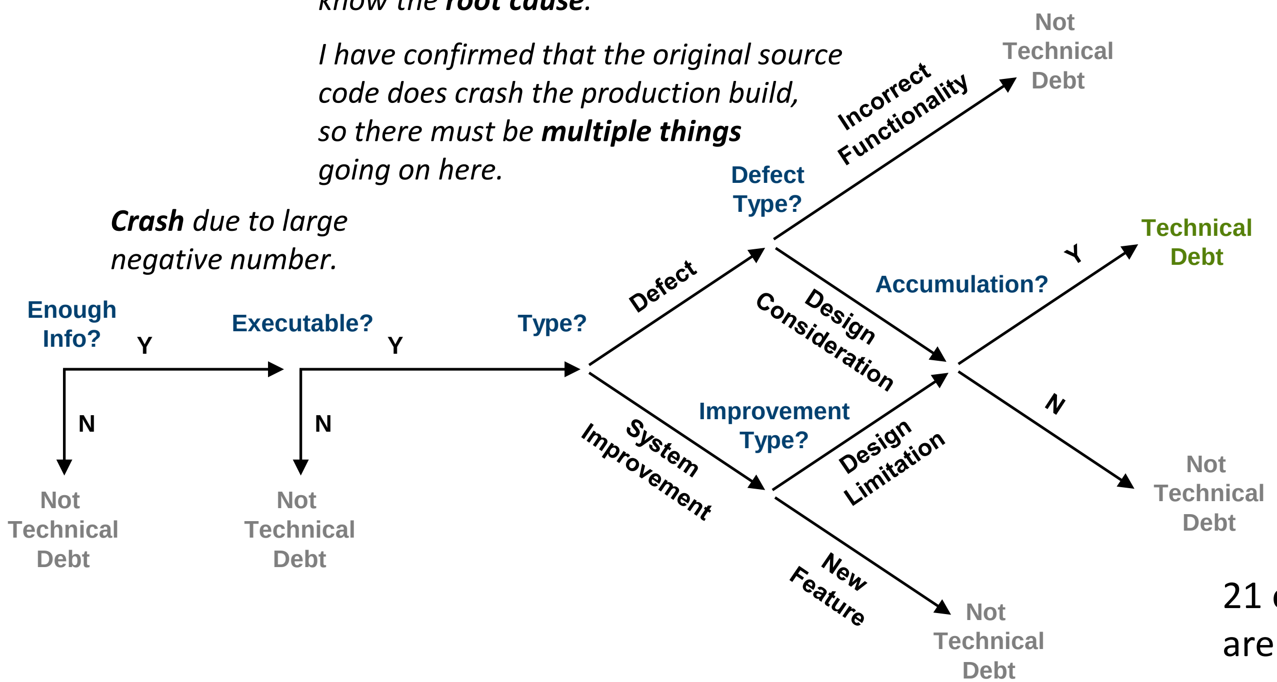
Indicator: Technical Debt Tag

*We could just fend off negative numbers near the crash site or we can **dig deeper***

*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

Manual analysis on four data sets reveal some common issues

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

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 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.

Unstable Interface

		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.

Modularity Violation

Shared secret between files

Should be extracted as design rules

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

Analysis: Design Flaws - 1

Increased rates of design flaws are strongly correlated with increased rates of security bugs.

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

Design flaws extracted using dependency analysis at the class level within files: unstable interface, modularity violation, improper inheritance, cycles.

Feng, Q., Kazman, R., Cai, Y., Mo, R., Xiao, L. Towards an architecture-centric approach to security analysis. *Proceedings of the 13th Working IEEE/IFIP Conference on Software Architecture*. IEEE, 2016.

Analysis: Design Flaws - 2

Moreover, being involved in more 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%*
4	1	0	*0%*

Qualitative and Quantitative Analysis

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

Design Flaws and Future Consequences

Classifying TD from Issues labeled Security

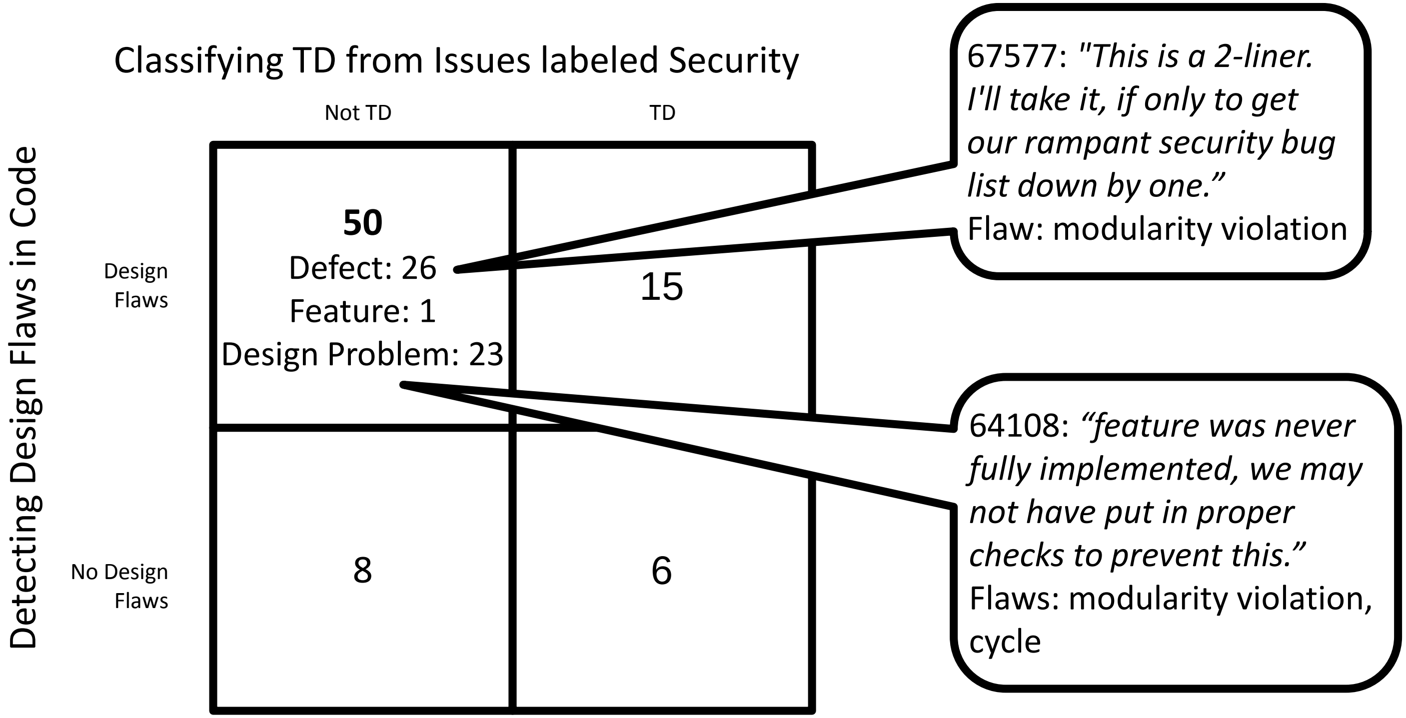
Detecting Design Flaws in Code

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

66931: *"Is it a workaround … the root bug …long term fix"*
Flaw: modularity violation

68766: *I've got my bandaid fix all reviewed and ready to check in once the tree reopens. But this problem sounds nasty enough that we definitely need a real fix."*
Flaws: modularity violation, cycle, improper inheritance

Partial Evidence



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”

70589: “My plan is to back out the brokenness, and fix it properly later”

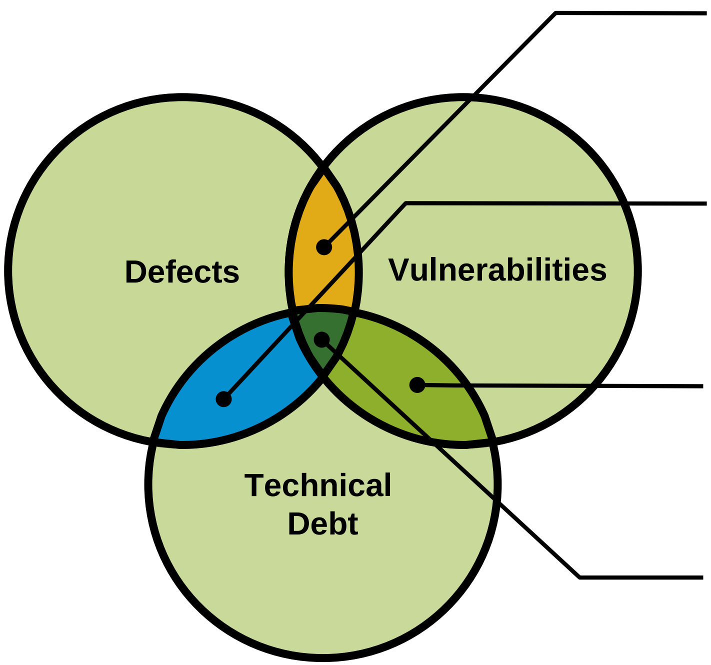
Take-aways

The more types of **design flaws** a file is involved in, the higher the likelihood of it also having **vulnerabilities**; files with vulnerabilities tend to have more code churn.

When they address **security** issues, software developers use **technical debt concepts** to discuss **design limitations** and their consequences on future work.

Take-aways

Technical debt can be made **visible earlier** when tracked similarly to defects and vulnerabilities, consequently managed more effectively and strategically. Organizations can start today.



Not all defects are vulnerabilities, but defect proneness does imply increased vulnerability risks

Defects are not technical debt, but technical debt as it lingers in the system increases defect proneness.

Similarly, technical debt increases vulnerability risks.

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

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

Software Solutions Symposium 2017

Robert Nord rn@sei.cmu.edu

Ipek Ozkaya ozkaya@sei.cmu.edu