Using Technical Debt to Improve Software Sustainability and Find Software Vulnerabilities

Ipek Ozkaya
Using Technical Debt to Improve Software Sustainability and Find Software Vulnerabilities

October 25, 2016

© 2016 Carnegie Mellon University

DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Distribution Statements

Copyright 2016 Carnegie Mellon University

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.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

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

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.

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 declared at acquisition time

By the time the government owns the system the accumulation of detection and redo is very expensive.

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

![Diagram showing the relationship between defects, vulnerabilities, and technical debt.](image)

- 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?

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

- 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?
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.

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.

# 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.

<table>
<thead>
<tr>
<th># Types of Design Flaws</th>
<th>Non-vuln files</th>
<th>Vuln files</th>
<th>% have vulns.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8544</td>
<td>47</td>
<td>0.5%</td>
</tr>
<tr>
<td>1</td>
<td>7357</td>
<td>141</td>
<td>2%</td>
</tr>
<tr>
<td>2</td>
<td>2345</td>
<td>91</td>
<td>4%</td>
</tr>
<tr>
<td>3</td>
<td>194</td>
<td>10</td>
<td>5%</td>
</tr>
</tbody>
</table>
Holistic Analysis
Qualitative and Quantitative Analysis

Chromium security issues

Classifying TD from Issues labeled Security

<table>
<thead>
<tr>
<th></th>
<th>Not TD</th>
<th>TD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design Flaws</td>
<td>50</td>
<td>15</td>
</tr>
<tr>
<td>No Design Flaws</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

79 issues are labeled security
- 21 are classified as technical debt
- 65 trace to files containing design flaws
### Partial Evidence

#### Classifying TD from Issues labeled Security

<table>
<thead>
<tr>
<th></th>
<th>Not TD</th>
<th>TD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defect: 26</td>
<td>50</td>
<td>15</td>
</tr>
<tr>
<td>Feature: 1</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>Design Problem: 23</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **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

#### Detecting Design Flaws in Code

- Design Flaws
- No Design Flaws
Supplement Static Analysis with Developer Knowledge

Classifying TD from Issues labeled Security

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

Detecting Design Flaws in Code

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**

<table>
<thead>
<tr>
<th>Priority 1</th>
<th>Priority 2</th>
<th>Priority 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical Debt Issues</td>
<td>22%</td>
<td>56%</td>
</tr>
<tr>
<td>Not Technical Debt Issues</td>
<td>24%</td>
<td>50%</td>
</tr>
</tbody>
</table>

**Developer discussion**

- Watchers vs. Days Open
  - Non-Technical Debt
  - Technical Debt
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

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.

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.
## The Technical Debt Community

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


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/