CREST Workshop

Got Technical Debt? Surfacing Elusive Technical Debt in Issue Trackers

Stephany Bellomo, Robert Nord, Ipek Ozkaya, Mary Poppeck

Nov 28-29th, 2016

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
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-0004232
Do issue trackers reveal technical debt?

- **RQ1**: Do developers use the term *technical debt explicitly* when discussing problems in their issue trackers?
- **RQ2**: Can *implicit* technical debt items be discovered systematically within issue trackers?
- **RQ3**: What are the distinguishing *characteristics* of technical debt items discovered in issue trackers?
### Overview of Data Sets

<table>
<thead>
<tr>
<th>Data set</th>
<th>Source</th>
<th>Filter criteria</th>
<th># Records analyzed</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Setup</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(instrument development)</td>
<td>Chromium</td>
<td>Google issue tracker</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>Connect</td>
<td>Jira</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Technical debt survey</td>
<td>Examples (as text)</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Phase 1</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TD categorization</td>
<td>Connect</td>
<td>Jira</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>Phases 2–4</td>
<td>Connect</td>
<td>286</td>
</tr>
<tr>
<td></td>
<td>TD classification, analysis, and evaluation</td>
<td>Project A</td>
<td>Defects/CRs Sep. 2010 to Dec. 2014</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Project B</td>
<td>All year 2013</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Chromium</td>
<td>Milestone 48 Stars (watchers) &gt; 3</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td>1,264</td>
</tr>
</tbody>
</table>

- Initial phased focused on exploring RQ1 (explicit declaration) and survey examples
- Core research phases 1-4
- Mix of open source and project data
- Created manageable sized data sets for manual analysis

Data sets are also published
Multi-phased analysis approach

1. **Categorization:** Extract reoccurring concepts from samples; create initial categorization

2. **Classification:** Systematically classify data sets using categorization

3. **Evaluation:** Validate effectiveness of classification with project stakeholders

4. **Analysis:** Analyze the technical debt items for characteristics

**Outputs:**
- Classified data set, refined classification guidance
- Stakeholder confirmation of findings
- Demographic statistical analysis; unstructured data affinity grouping and analysis

**Examples**
- Are these really TD?
- Does the classification make sense?
Technical Debt Classification Rules (Described as a Decision Tree)

- In current project, we are using method with larger datasets and machine learning
Technical Debt Breakout

Defects/Vulnerability (377)
- 100 new features
- 151 documentation
- 28 not enough info

Other (279)
- 100 new features
- 151 documentation
- 28 not enough info

Technical Debt (51)

Deployment & Build
- Out-of-sync build dependencies 3 CN
- Version conflict 1 CN
- Dead code in build scripts 1 CN
- 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

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

CH = Chromium, PA = Project A, PB = Project B, CN = CONNECT
Examples

Not Technical Debt

[Project A #25] Correct the values for subsystem A to reflect the subsystem B values

[Project B #265] Update alert authoring UI – ‘event window’ should be close to ‘any rule’ checkbox

[Project B #1513] Refactor onclicks in nodes.html into query events

Technical Debt

[Project A #18] approximately 340 records exist in the database twice … so much time had elapsed in some cases the duplicate was endorsed.

[Chromium #367158] Currently, we have a lot of duplicate/boilerplate code in this test. We should try to simplify this test so that it’s easier to maintain and read.
Example of a Technical Debt Item

<table>
<thead>
<tr>
<th>Name</th>
<th>Connect #Gateway-1631: Empty Java package (dead code)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development artifact</td>
<td>The re-architecture of the source code to support multiple NwHIN specifications has introduced a new Java packaging scheme.</td>
</tr>
<tr>
<td>Symptoms</td>
<td>Numerous empty Java package folders present across multiple projects.</td>
</tr>
<tr>
<td>Consequences</td>
<td>No impact to functionality; however, may lead to confusion for users implementing enhancements or modifications to the source code.</td>
</tr>
<tr>
<td>Analysis</td>
<td>New and existing classes have been moved into these new package folders; however, the previous package folders have been left in place with no class files.</td>
</tr>
</tbody>
</table>

**Our Assertion:** Technical debt can be made **visible earlier** when tracked similarly to defects, consequently managed more effectively and strategically.
RQ3: Are there any quantifiable characteristics

Are TD issues open longer?

Do TD issues generate more developer discussion?

Do TD issues have higher priority?

<table>
<thead>
<tr>
<th></th>
<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>
<td>22%</td>
</tr>
<tr>
<td>Not Technical Debt Issues</td>
<td>24%</td>
<td>50%</td>
<td>26%</td>
</tr>
</tbody>
</table>

Technical Debt Research at the SEI
March 15, 2016
© 2016 Carnegie Mellon University
Approved for Public Release; Distribution is Unlimited
Our Emerging Definition of Technical Debt

Technical debt is design work relating to software units that have evidence of present or anticipated accumulation of extra work.

• Exists in an executable system artifact, such as code, build scripts, automated test suites;

• Is traced to several locations in the system, implying ripple effects of impact of change;

• Has a quantifiable effect on system attributes of interest to developers, such as increasing number of defects, negative change in maintainability and code quality indicators are symptoms of technical debt.
Summary of Findings

• Using this method we manually identified 51 examples of technical debt records in several issue tracker datasets.

• Existing definitions focus on the explicit shortcuts, however, the issues we found are mostly implicit - result of unintentional design choices.
  • We presented an emerging definition from our work.

• We found no searchable characteristics when we analyzed the technical debt records.
  • Consequently, text analysis is necessary.

• We observed developers do not identify the consequences of technical debt in issue trackers
  • Suggested a template for improving this.
Future Vision: Towards Technical Debt Analytics

**Problem:** Managing the consequences of technical debt relies on an ability to (1) identify unintentional decisions and (2) quantify the consequences of such decisions.

**Solution:** Develop tools that integrate data from multiple, commonly available sources to surface problematic decisions and quantify consequences.

**Approach:** Combine techniques from machine learning, code analysis, and data mining to identify problematic design issues.