

Research Review 2017

Technical Debt Analysis through Software Analytics

Dr. Ipek Ozkaya

Principal Researcher

Copyright 2017 Carnegie Mellon University. All Rights Reserved.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

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.

Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM17-0795

Technical Debt Analytics

Technical debt describes a universal software development phenomenon: design or implementation constructs that are expedient in the short term, but set up a technical context that may make future changes costly.

The Problem

Government acquirers need capabilities to assess the technical debt in software and its cost impact.

Managing technical debt relies on an ability to:

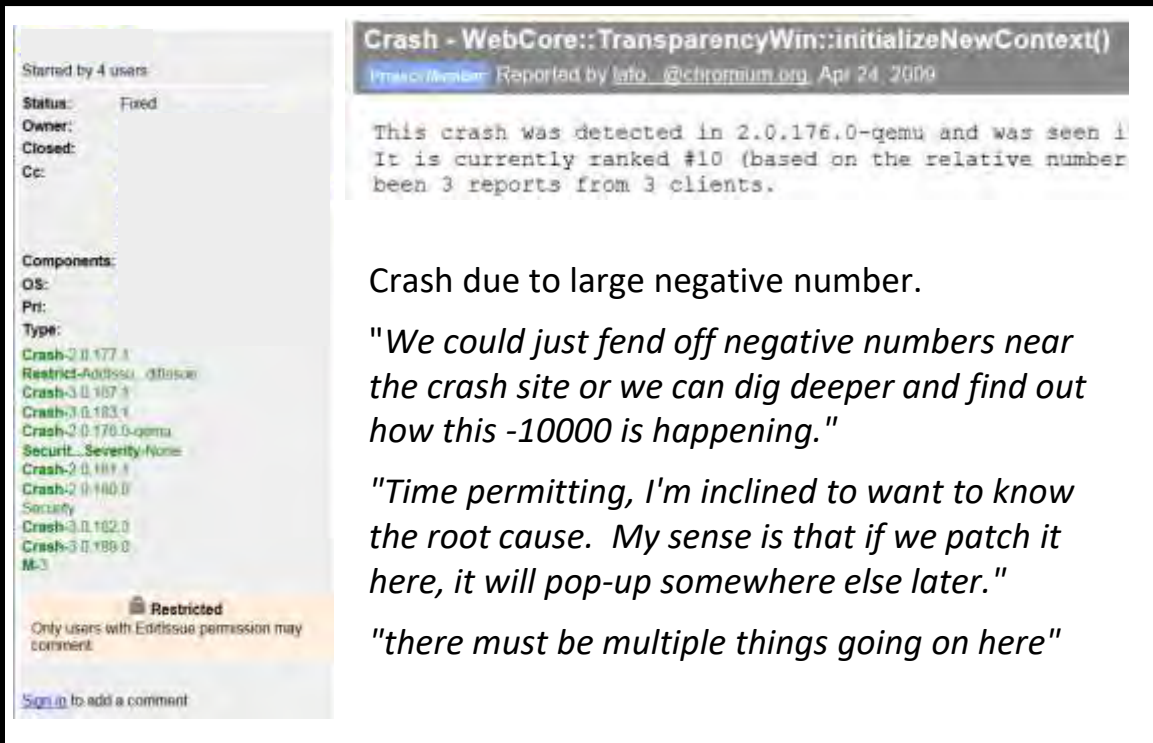
1. identify design decisions
2. quantify consequences of such unintentional and intentional design decisions.

The Solution

Develop software analytics capabilities integrating data from multiple, commonly available sources.

Combine machine learning, refactoring and code analysis, and data mining techniques to create technical debt items that identify problematic issues with potential long-term adverse consequences.

How do Experts Know Technical Debt



Started by 4 users:

Status: Fixed
Owner:
Closed:
Cc:

Components:
OS:
Pri:
Type:

Crash-2.0.177.1
Restrict-Address (3/10/0)
Crash-3.0.107.1
Crash-3.0.183.1
Crash-2.0.170.0-gem
Security-Severity:None
Crash-2.0.181.1
Crash-2.0.140.0
Security
Crash-3.0.102.0
Crash-3.0.189.0
M-3

Restricted
Only users with EditIssue permission may comment.

[Sign in](#) to add a comment

Crash - WebCore::TransparencyWin::initializeNewContext()
[Track/Number] Reported by [Info. @chromium.org](#), Apr 24, 2009

This crash was detected in 2.0.176.0-gem and was seen 1
It is currently ranked #10 (based on the relative number
been 3 reports from 3 clients.

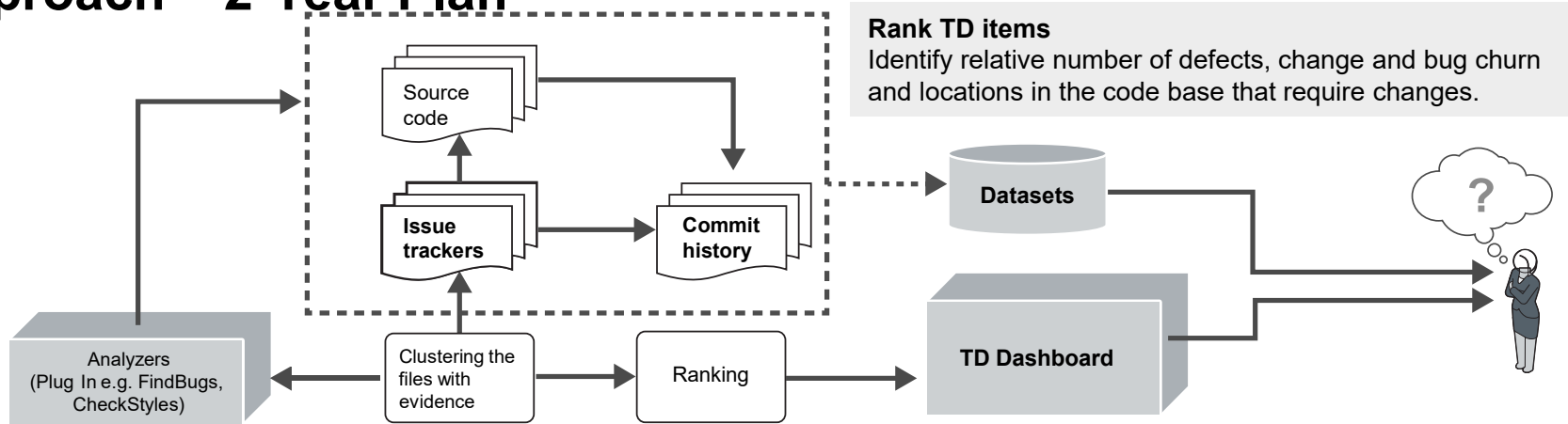
Crash due to large negative number.

"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 must be multiple things going on here"

Approach – 2-Year Plan



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 (e.g., build dependencies, dead code).

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 (e.g. same design concept, same files, co-changing files) and extract impacted additional files with related violations.

Our Progress

Applying ML to Technical Debt Extraction from Issue Trackers

Deployment & Build	Out-of-sync build dependencies
	Version conflict
	Dead code in build scripts
Code Structure	Event handling
	API/Interfaces
	Unreliable output or behavior
	Type conformance issue
	Dead code
	Large file processing or rendering
	Encapsulation
Data Model	Caching issues
	Data integrity
	Data persistence
	Duplicate data
Regression Tests	Test execution
	Overly complex tests

Experimenting using supervised algorithms and improve our training set:

- Experts consistently agree, while some algorithms clue on refactoring action phrases such as “style errors” that are not design related
- Developers do identify long lasting design issues as technical debt, but not consistently.

Manual analysis on four data sets reveal some common issues

Applying ML to Technical Debt Extraction from Issue Trackers

Approach:

n-gram feature engineering and gradient boosting focusing on

- modeling with boosting algorithms to build the weighted average of many classification trees – iteratively improving the weak classifiers and creating a final strong classifier
- increasing our labelled data set, which includes actual examples where developers themselves declare the issues as technical debt

Model	Precision	Recall	Test count	F-measure	Training Count
1	0.60	0.42	510	0.50	157
2	0.71	0.32	411	0.44	256
3	0.77	0.42	311	0.54	356
4	0.74	0.70	161	0.72	506

- Our models are improving, especially in recall (more true debt is identified)
- Experimenting with improving accuracy (all debt is identified)
- We are engaged with stakeholders eager to try this approach out on their data

Analysis Rules and Design Topics

Approach:

Are some rules more useful than others? Analyzed 466 Java & C# rules across three tools:

- 55% were easily labeled as non-design and 19% were labeled as clearly design, others needed further context
- Design rules go beyond statement level quality checks and are cross system boundaries.
- Currently validating with open source & collaborator projects.

EXAMPLES OF DESIGN, NON-DESIGN, AND HARD TO CLASSIFY RULES

Design Rules

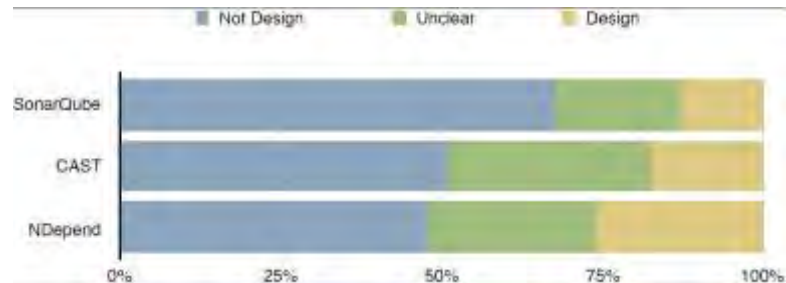
Action Classes should only call Business Classes
Avoid high number of class methods invoked in response to a message
Avoid Classes with a High Lack of Cohesion

Non-Design Rules

Try-catch blocks should not be nested
All script files should be in a specific directory

Hard to Classify

Avoid hiding attributes
Avoid defining singleton or factory when using Spring
Avoid declaring an exception and not throwing it
Lines of code covered by tests



Analysis Rules and Design Topics

Applied design rule extraction to 9 projects:

- Teams consistently rate maintainability rules as low priority
- We are able to identify deep routed design problems such as logging, exception handling, synchronization that should have been acted on earlier

334	Standard outputs should not be used directly to log anything	DR
39	Nested code blocks should not be used	DR
20	@FunctionalInterface annotations should be used to flag Single Abstract Method interfaces	DR
20	clone should not be overridden	DR
19	Classes should not be too complex	DR
17	“Exception” should not be caught when required by called methods	DR
15	Methods should not have too many parameters	DR
10	Classes named like “Exception” should extend “Exception” or a subclass	DR
3	Exception handlers should preserve the original exceptions	DR
2	Throwable and error should not be caught	DR
2	Credentials should not be hard-coded	DR

Impact

Role	Contribution to our work	SEI Impact
DoD PM, sustainment professionals	Challenge problems, project measures	Help develop policy guidance, (e.g., AFLMC work)
Defense contractors	Data, feedback, validation	Develop organizational practices (e.g., Lockheed Martin data sharing)
Industry	Data, feedback, validation	Incentivize teams to identify technical debt (e.g., ABB data sharing)
Tool vendors	Transition partner	Extend tools to label and analyze technical debt items (e.g., Silverthread, Lattix)
Researchers, students, PIs	Technical validity	Community leader (e.g. increased number of PhDs; the first 2-day conference on TechDebt at ICSE 2018)

Summary

Our progress is on target with interim pilot opportunities

- The issue tracker analysis provides an opportunity for uncovering systemic, lingering issues.
- Design rule analysis provides an opportunity to analyze systems with a different perspective and reprioritize the results accordingly.

Future work includes bringing these pieces together for an ongoing software analysis capability

- DoD and other government stakeholders need this capability! There is increasing lack of understanding of the state of the software quality and its consequences.
- The advances in data analytics techniques is an opportunity.
- Technical debt will only be more important to analyze for and manage!

Provide Contact Information

Presenter

Dr. Ipek Ozkaya
Principal Researcher
ozkaya@sei.cmu.edu

Team

Dr. Robert Nord, SSD
Stephany Bellomo, SSD
James Ivers, SSD
Dr. Zach Kurtz, CERT

