Improving Software Sustainability Through Data-Driven Technical Debt Management

Ipek Ozkaya

October 7, 2015

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

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.

DM-0002839

What is Technical Debt?

We define technical debt as a software design issue that:

- 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.
  - We initially focus on detecting indicators in the form of violating known architectural pattern and maintainability rules to trace such symptoms

What is Technical Debt: Examples

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

Modifiability violation, pattern conformance

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

Modifiability violation, pattern conformance

"A simple API call turned into a nightmare <due to not following guidelines>"

Framework, pattern conformance

DoD Perspective of the Problem

By the time the government owns the system the accumulation of

"Contractor developed our software tool and delivered the code to the government for maintenance. The code was poorly designed and documented therefore there was a very long learning curve to make quality changes. We continue to band aid over 1 million lines of code under the maintenance contract. As time goes by, the tool becomes more bloated and harder to repair."

Cont
inten
or ur
incur

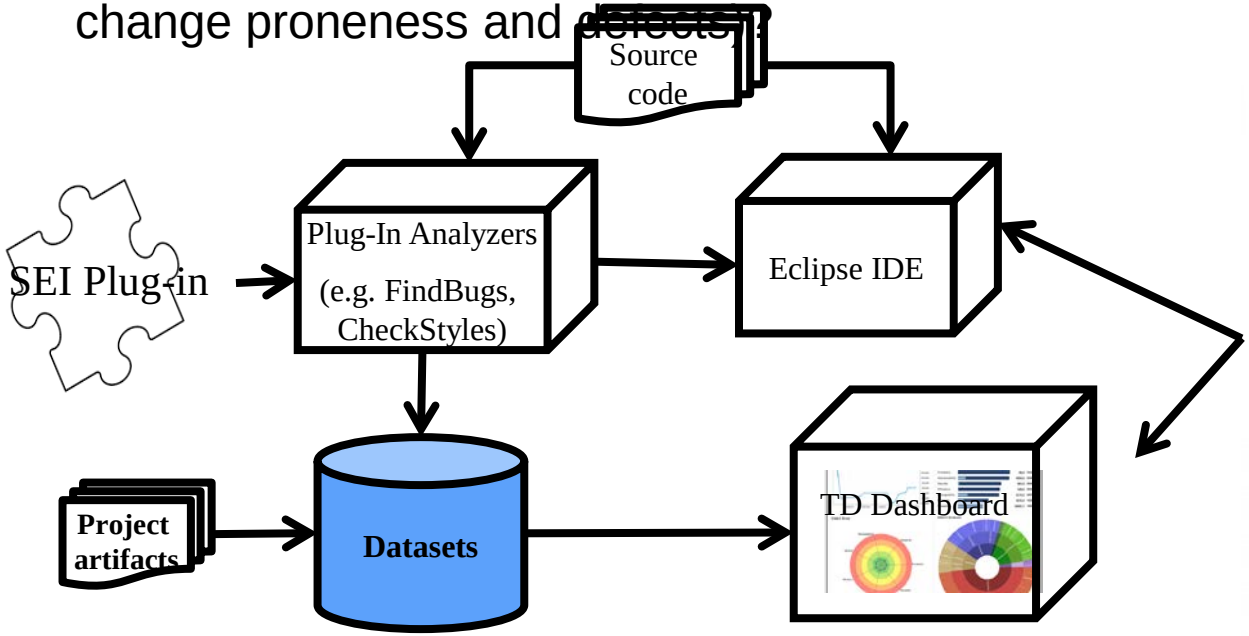
Our goal is to enable better sustainment decision making through

- identifying **indicators** that signify major contributors to technical debt
- analyzing **data sets** to build correlations between these indicators and project measures, such as defect and change proneness

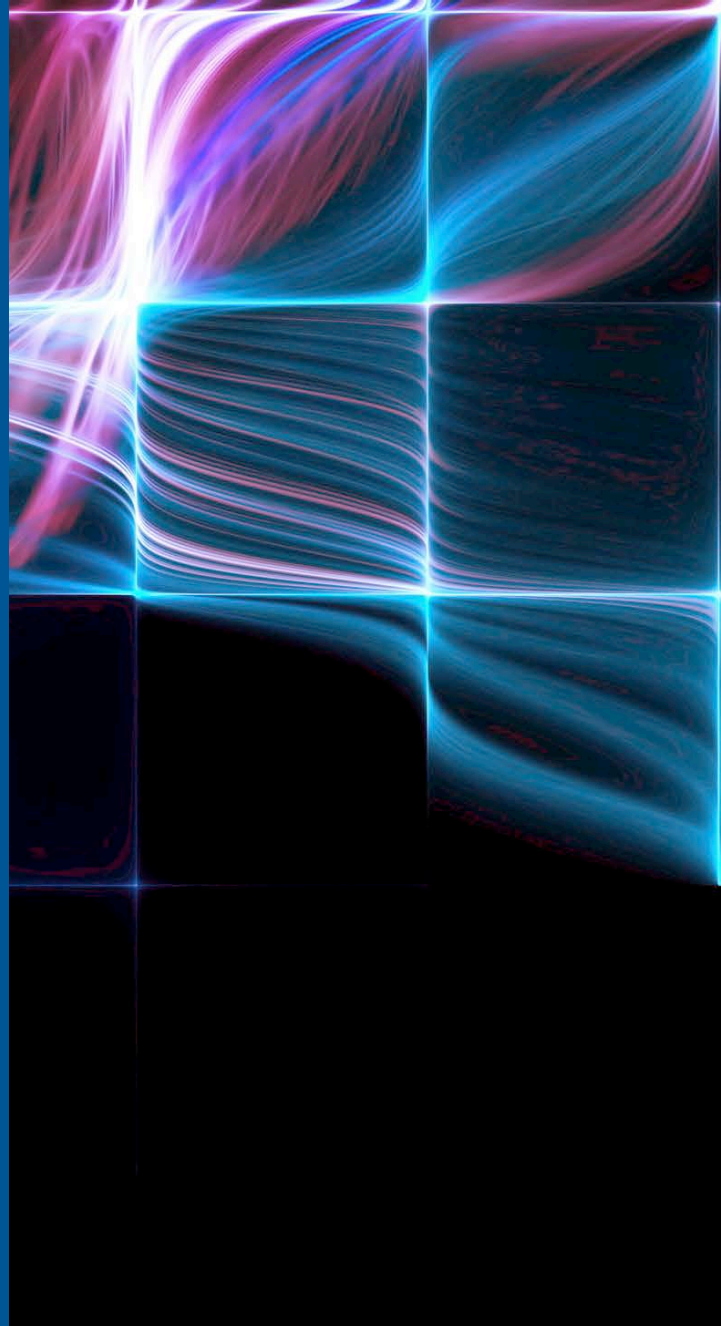
1. time technical debt is incurred
2. time technical debt is recognized
3. time to plan and re-architect
4. time until debt is actually paid-off
5. continuous monitoring

Research Questions

- RQ1: What do our stakeholders care about? Which issues would benefit from being tagged as technical debt?
- RQ2: Can we detect indicators of design issues that result in technical debt?
- RQ3: What are the data needs for correlation?
  - Once we detect them can we map them to externally visible measures (e.g., change proneness and defects)?



Which Issues Would Benefit from Being Tagged as Technical Debt?



RQ1: What Do Stakeholders Care About?

Org	Type	# Surveys out / received
A	Defense Contractor	3,500 / 248
B	Global automation, power robotics	15,000 / 1511
C	Government development/ research lab	200 / 73
D	DoD sustainment	35 / 29
	Total	1861

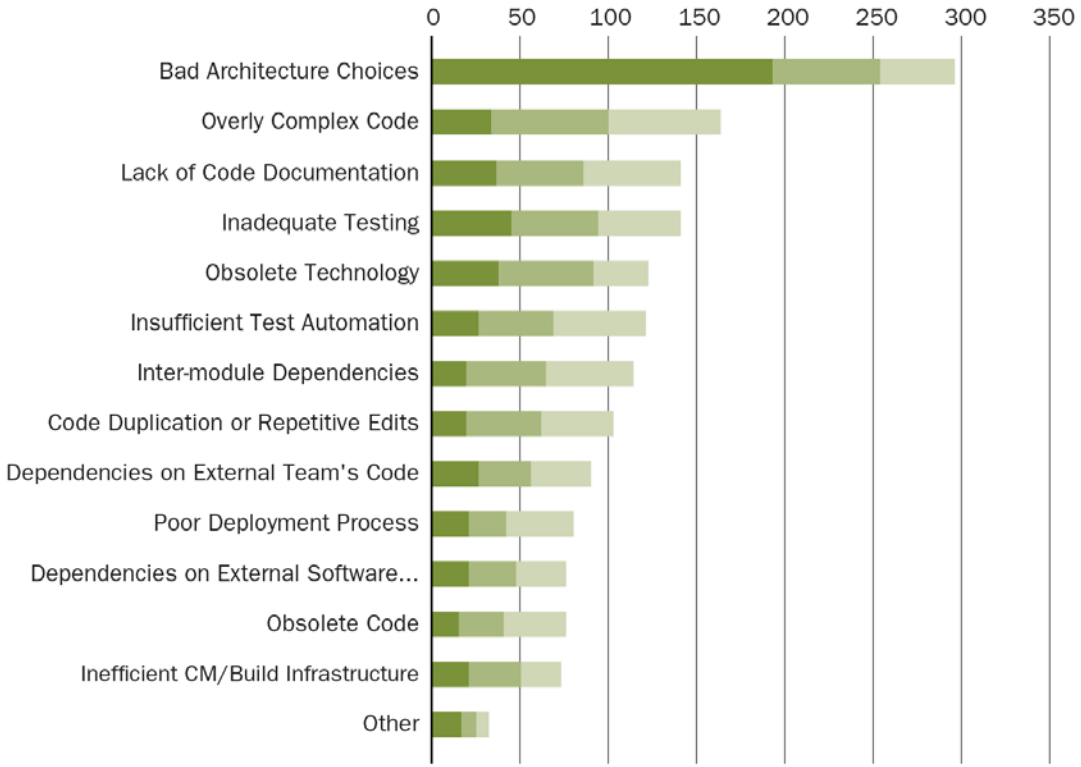
Collaborated with two global development organizations and two government development and sustainment labs to answer:

- Is there a commonly shared definition of technical debt among professional software engineers?
- Are issues with architectural elements among the most significant sources of technical debt?
- Are there practices and tools for managing technical debt?

Findings – 1

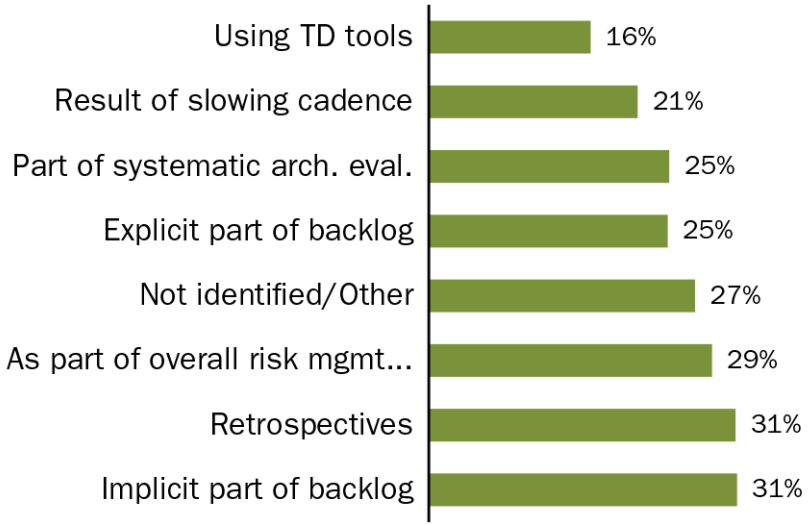
Technical debt is just not an abstract metaphor!

Bad architectural choices rated as the top contributor to technical debt, followed by overly complex code and inadequate testing. 56% of the respondents ranked architecture among their top 3 pain points.

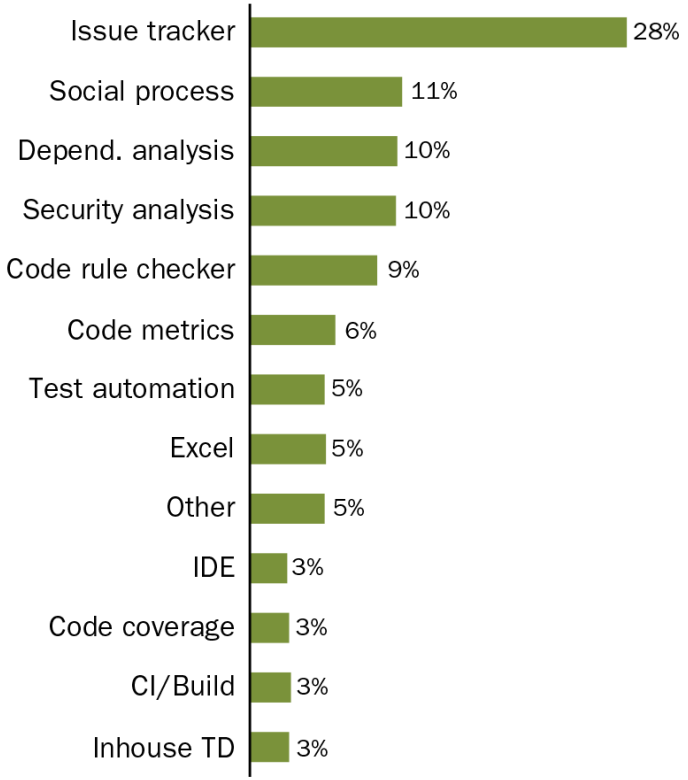


Findings – 2

75% of respondents said that dealing with the consequences of technical debt has **consumed a painful chunk of project resources.**



Current **tools do not capture** the key areas of accumulating problems in technical debt.



Results

Significance

- First of its kind broad, practice-based study with impact on research, government, and industry.
- The finding that bad architecture choices are most significant contributor to debt is influencing other's research.
- Enabling us to create engagement where we conduct detailed artifact analysis with two of our collaborators.

Publications

"Measure it? Manage it? Ignore it? Software Practitioners and Technical Debt" N. Ernst, S. Bellomo, I. Ozkaya, R. Nord, I. Gorton, FSE 2015

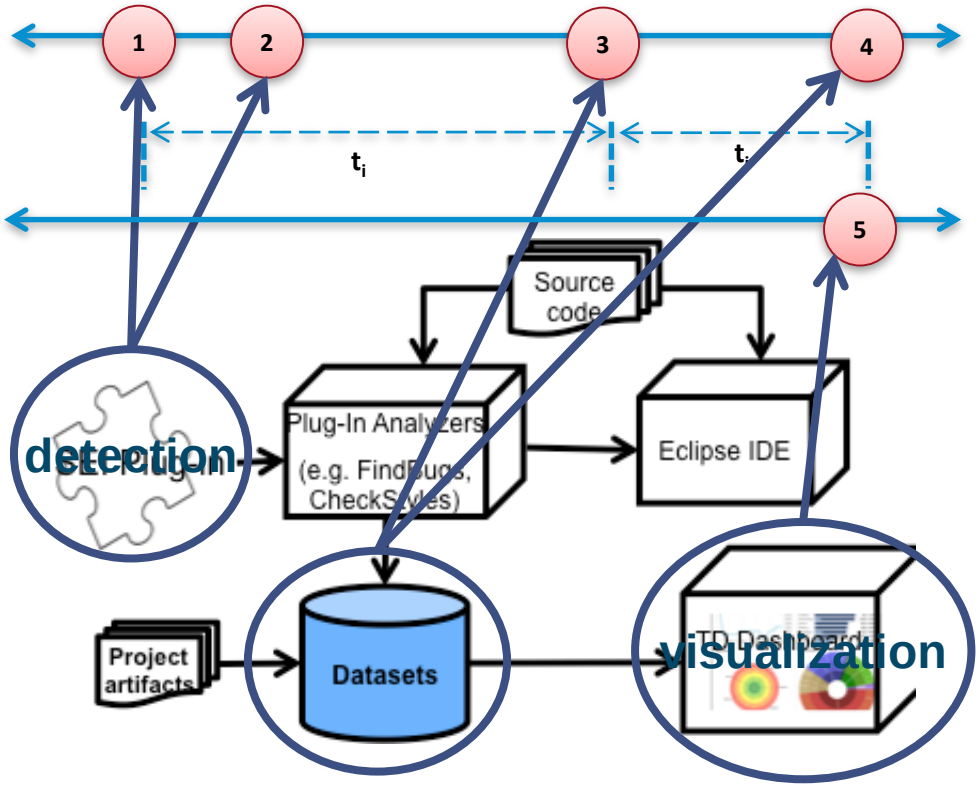
ACM SIGSOFT Distinguished Paper Award

Can We Detect Indicators of Design Issues that Result in Technical Debt?

RQ2: Can We Detect Indicators of Design Issues?

What code and design indicators can be repeatably discovered that correlate with project measures that allow us to manage technical debt?

- combines *static code analysis, architectural abstractions, empirical field studies,* and *conceptual correlation modeling* to test qualitative causal assumptions.



1. technical debt is incurred
2. technical debt is recognized
3. plan and re-architect
4. debt is actually paid-off
5. continuous monitoring

Tool Support

Any tool for experimentation should

- have a low threshold of entry for organizations
- be easy to extend by others

Selected SonarQube as our prototype environment

- Pros
  - API that we and others can extend
  - built-in analysis frameworks for code analysis to extend with rules
- Cons
  - incorporates an existing technical debt measurement framework that is code quality level and not validated. This results in confusion

*Previously had analyzed Cast, Lattix and Structure101. Ran experiments with SonarQube and research prototype from Drexel University, Titan

Findings – Detecting Modularity

Initial results on analyzing sample project (Connect version 4.4) point to architecture root cause of technical debt.

- Files that have the most modularity issues make up 16% of the overall system
- These files on the other hand represent a substantial percentage of the bugs
  - Looking at StartDate 6/20/12 EndDate 9/15/13, Files represent 84% of the bugs,
  - Looking at StartDate 9/16/13 EndDate 12/8/14, Files represent 47% of the bugs,
- A reduction in issues may imply a major refactoring.

Results

Significance

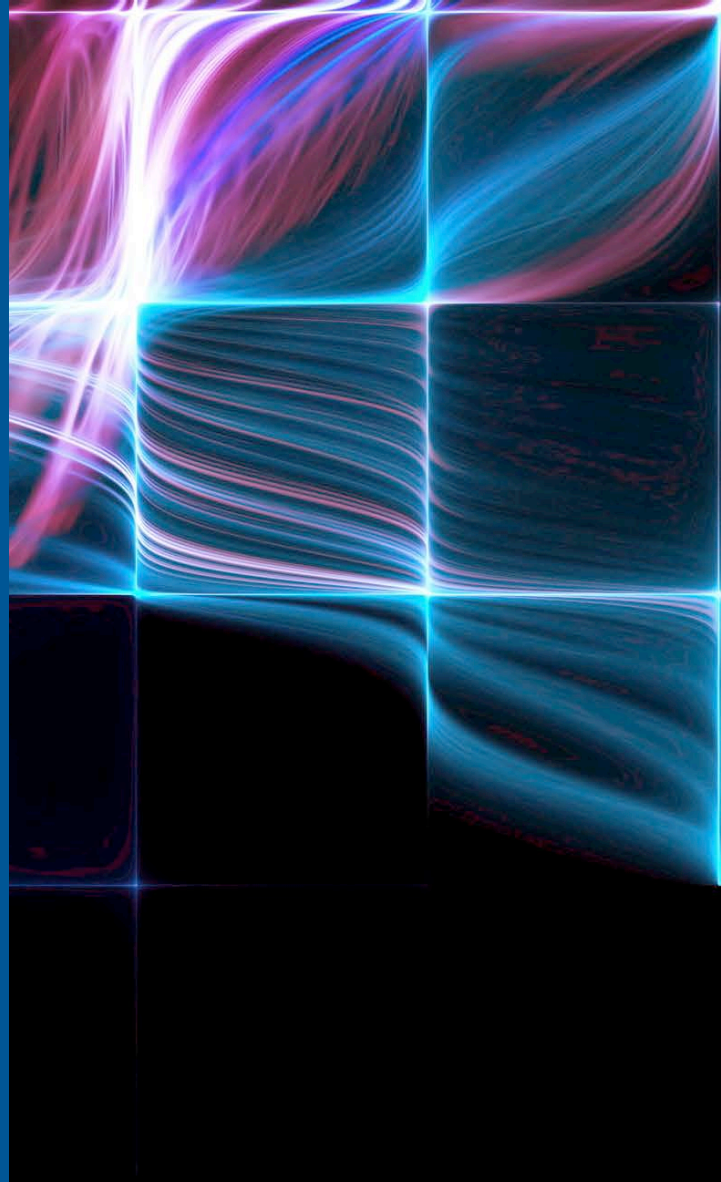
- Focuses typical code detection techniques on architecturally significant design issues
- Starts building the validation environment

Publication

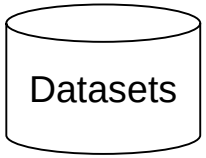
"A Case Study in Locating the Architectural Roots of Technical Debt," R. Kazman, Y. Cai, R. Mo, Q. Feng, L. Xiao, S. Haziyev, V. Fedak, A.Shapochka, ICSE 2015, (Florence, Italy), May 2015.

"Hotspot Patterns: The Formal Definition and Automatic Detection of Architecture Smells," R. Mo, Y. Cai, R. Kazman, L. Xiao, WICSA 2015, (Montreal, Canada), May 2015.

What Are the Data Needs for Correlation?



RQ3: Data Needs for Correlation

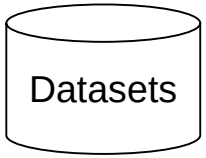


What are the data needs? Based on our practice studies:

Independent examples	Analysis inputs	Outputs/ Dependent variables
Replicated functionality	code clones	$$$
Functionality depending on in-house algorithm	dependencies	$$$
Coupling between two modules	dependencies	$$$
Code doesn't need to be developed at safety criticality level	dependencies/ designated criticality	$$$
High stress test scenario generated major failure	complexity	$$$

Closer look into the dependent variables show that additional effort is either spent on defects/issues or propagating changes.

Data Test Beds – 1



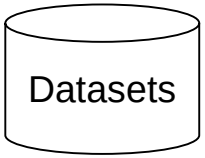
DoD relevant communication terminal

Qualitative: knowledge about major refactorings over program lifetime, and assessments from acquisition team and contractor, access to the SEI team

Quantitatively:

- Outcomes that show when technical debt was increasing vs. "bought down"? (e.g., defect proneness, change proneness, cost of change)
- Early indicators of technical debt growth (e.g., deviation from good system design principles, deviation from reference architecture)
- Internal to the SEI, history of 2006-2011 period including some versions of code, project performance metrics (defects, PDR/CDR analysis results).

Data Test Beds – 2



Government open source health IT exchange

Qualitative: architecture evaluation (ATAM) results from 2011, access to the development team, specific issues the team tagged as technical debt, documentations

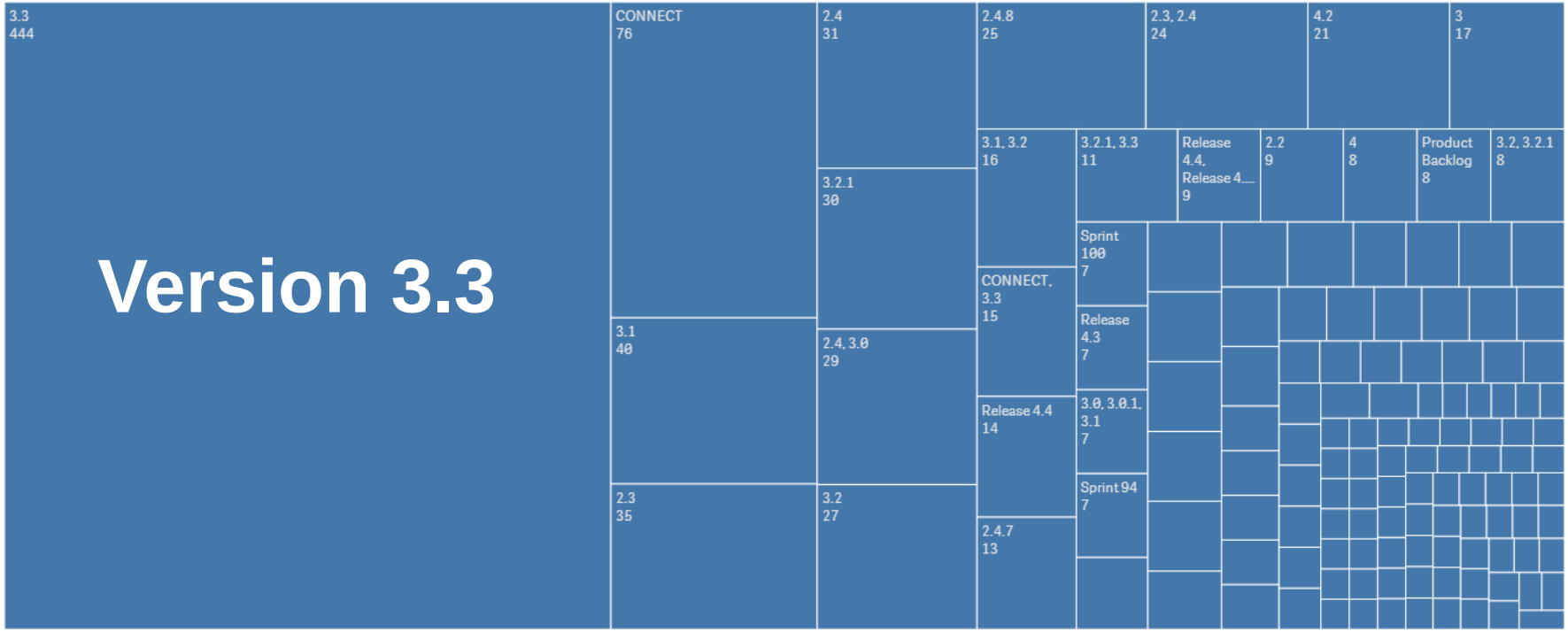
Quantitatively:

- Jira data (commits, check-ins and check-outs over 10 releases)
- Issues data base
- Code repository in GitHub

Analyzing Connect Data from Jira and GitHub

How are issues distributed across releases:

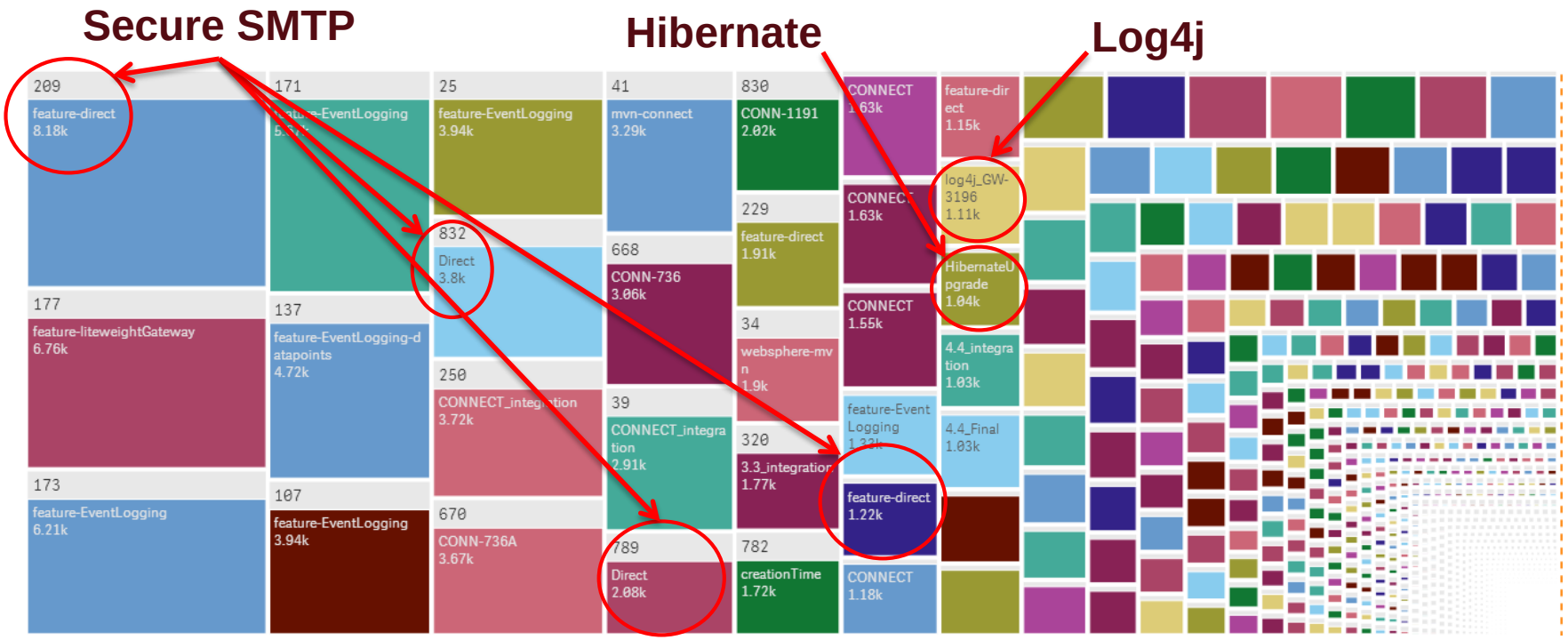
- Version 3.3 has an order of magnitude more issues



Analyzing Connect Data from Jira and GitHub

Which files are affected by what types of issues:

- Classifying files based on issues can help understand the impact of change



Analyzing Connect Data from Jira and GitHub

```
14      // BalanceInquiry constructor
15      public BalanceInquiry(int userAccountNumber, Screen atmScreen,
16              BankDatabase atmBankDatabase, Keypad atmKeypad) {
17          super(userAccountNumber, atmScreen, atmBankDatabase);
18          keypad = atmKeypad;
19      } // end BalanceInquiry constructor
20
21      // performs the transaction
22      public void execute() {
23          logger.info("This is a test\nlog over two lines");
```

Avoid using newlines in log messages ···

a minute ago ▾ L23

Minor ○ Open Not assigned Not planned

log4j, sei

```
24
25          // get references to bank database and screen
26          BankDatabase bankDatabase = getBankDatabase();
27          Screen screen = getScreen();
28
29          // get the available balance for the account involved
30          double availableBalance =
31                  bankDatabase.getAvailableBalance(getAccountNumber());
```

Research and Transition

Extensions to the open source technical debt model and tooling to include other key quality attributes concerns, e.g. security, architectural technical debt management tooling

Relationship of technical debt management and testing

Extensions to the data sets of rules for detecting likely sources of technical debt, along with correlations to cost to fix, cost to implement a new feature, and defects with other case studies

Courses and case studies, published data sets

Team

SEI Team Members

- Ipek Ozkaya, PhD, SSD
- Rod Nord, PhD, SSD
- Stephany Bellomo, MSc., SSD
- Neil Ernst, PhD, SSD
- Ian Gorton, PhD, SSD
- Rick Kazman, PhD, SSD
- Forrest Shull, PhD, SSD/ERO
- Harry Levinson, SSD/CTS

Research Collaborators

- Philippe Kruchten, PhD, Univ. of British Columbia, funded
- Raghu Sangwan, PhD, Penn State, funded
- Managing Technical Debt research community, inf. Sharing
- Industry, DoD, and tool vendor partners

Contact Information

Ipek Ozkaya
Principal Researcher
SSD/SEAP
Telephone: +1 412 268 3551
Email: ozkaya@sei.cmu.edu

U.S. Mail
Software Engineering Institute
Customer Relations
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
USA