

In Search of a Metric for Managing Architectural Technical Debt

A 10 Year Retrospective

Robert Nord (rn@sei.cmu.edu)

Ipek Ozkaya (ozkaya@sei.cmu.edu)

Philippe Kruchten (pbk@ece.ubc.ca)

Marco Gonzalez-Rojas (emarkux@protonmail.com)

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

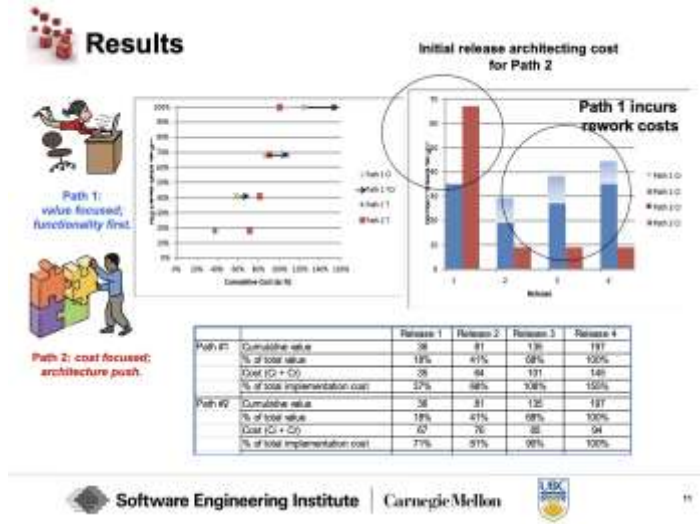
Our vision in 2012

Provide a way for the software engineering community to recognize that technical debt has its roots in architecture rework.

The 2012 ICSA paper presented a dependency analysis framework for measuring architecture rework as a proxy for technical debt.



Let's Recall the Paper and the Presentation



We exemplified use of one metric, propagation cost, but emphasized that quantifying rework is not trivial.

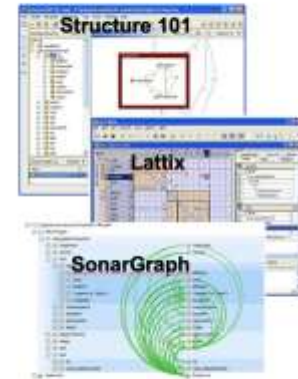
Metrics for Quantifying Architecture Quality

Challenges

- Insufficient and unproven metrics for quantifying architecture quality to guide the re-architecting process.
- Code-level refactoring techniques do not scale effectively to support architecture-level evaluation for re-architecting.

There has been an increasing focus on tools for the purpose of structural analysis.

- increasing sophistication,
- support for some structural analysis in addition to code analysis,
- first steps towards analyzing financial impact by relating structure analysis to cost and effort for rework.



We demonstrated the potential rework created as a consequence of the tension between architecture decisions and delivering priority functional requirements.

The Work that Built on the Paper

Research recognizing the **connection of architecture and design roots of technical debt**

- e.g. Martini and Bosch 2014-2015, Lin, Liang, Avgeriou 2015

Research investigating propagation cost (Pc) and other **architecture related metrics and architecture smell detection**

- e.g. Abad 2015, Ampatzoglou 2015, MacCormack 2016, Azadi 2019, Verdecchia 2020

Self-admitted technical debt research which identified conversations in code comments providing further examples of technical debt and architecture.

- e.g. Maldonado and Shihab 2017

Work that focused on understanding **how to manage technical debt and architecture evolution**, including systematic literature studies

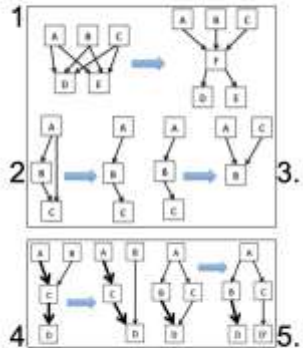
- e.g. Fontana 2016, Guo 2016, Letouzey 2012, Rios 2018, Besker 2018

The good, the bad, and the opportunity

The paper made the architecture roots of technical debt very visible.

“Architectural technical debt” in our title unintentionally implied a technical debt taxonomy.

Sensitivity Analysis



Can we identify propagation cost patterns with known evolution patterns

1. SOA-like
2. Strict Layering
3. Dependency inversion
4. Short circuit
5. Module splitting

Not enough people attended the presentation to pick up on our tease to investigate the relationship between design patterns and metrics.

R. L. Nord, I. Ozkaya, R. S. Sangwan, J. Delange, M. A. Gonzalez, P. Kruchten: *Variations on Using Propagation Cost to Measure Architecture Modifiability Properties*. ICSM 2013: 400-403

Going forward

Both commercial software industry (e.g. agile at scale/DevOps practices) and regulated software environments (e.g. Adaptive Acquisition Framework) today recognize technical debt management as a core software engineering practice.

TechDebt conference (www.techdebtconf.org) is a way to connect to ongoing research.

Open questions include:

- How to quantify rework with a variety of metrics to guide how and when to refactor systems to resolve technical debt?
- How can rework quantification be related to operational practices, e.g. how should technical debt be recorded and prioritized?
- How can empirical data and analysis be used to improve iterative and incremental architecture practices to manage technical debt?

THANK YOU!



Copyright 2022 Carnegie Mellon University, Marco Gonzalez-Rojas and Philippe Kruchten .

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.

References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

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.

DM22-0211