Towards Improving Software Architecture Degradation Mitigation by Machine Learning

Sebastian Herold

Christoph Knieke
Mirco Schindler
Andreas Rausch

Karlstad University, Sweden

Clausthal University of Technology, Germany

IARIA

ADAPTIVE ESES 2020

29th October, 2020

Contact: sebastian.herold@kau.se

KARLSTADS UNIVERSITET

TU Clausthal
Clausthal University of Technology

About the presenter

- Short resume
  - 2015- : Associate Professor in Computer Science at Karlstad University
  - 2014-2015: Research Fellow at Lero – The Irish Software Research Centre
  - 2011-2014: Post-doctoral researcher at Clausthal University of Technology

- Research Interests
  - Software Architecture, in particular degradation
  - Software Evolution and Modernization

What is this about?

Software Architecture Degradation

“The continuous divergence between the intended (prescriptive) and the as-implemented (descriptive) architecture.”

Machine Learning

THAT WAS SURPRISINGLY EASY. HOW COME THE ROBOTIC UPRISING USED SPEARS AND ROCKS INSTEAD OF MISSILES AND LASERS?
IF YOU LOOK TO HISTORICAL DATA, THE VAST MAJORITY OF BATTLE-WINNERS USED PRE-MODERN WEAPONRY.

Thanks to machine-learning algorithms, the robot apocalypse was short-lived.

“Software learning to perform a certain class of task better over time based on previous experience.”

Motivation

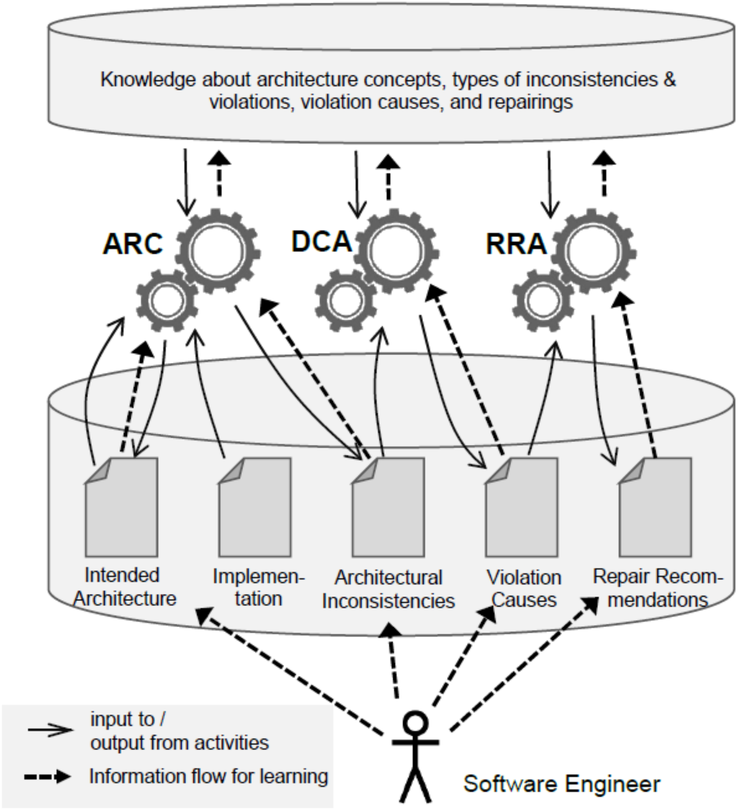
- Typical activities in architecture degradation mitigation
  - Architecture recovery
  - Consistency checking / degradation detection
  - Degradation analysis / comprehension
  - Degradation repairing

- All activities are labour-intensive and intellectually challenging

Motivating question:

How can we make use of machine learning to mitigate architecture degradation more efficiently?

Core of the approach


Knowledge about architecture concepts, types of inconsistencies & violations, violation causes, and repairings
ARC
DCA
RRA
Intended Architecture
Implemen-tation
Architectural Inconsistencies
Violation Causes
Repair Recom-mendations
input to / output from activities
Information flow for learning
Software Engineer

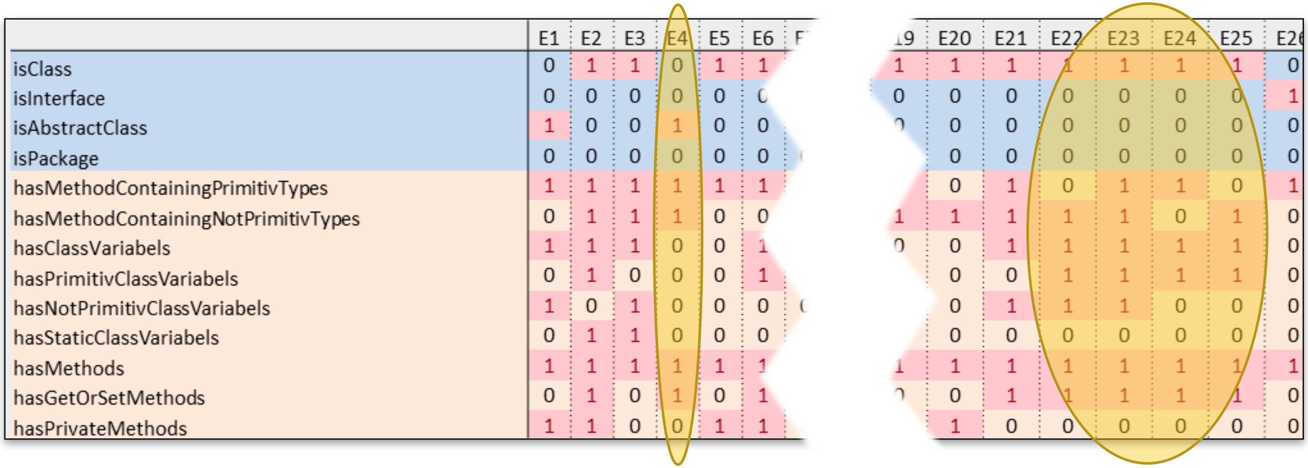
- **A**rchitecture **R**ecovery and **C**onsistency
  - Recover intended architecture from code
  - Check consistency between intended architecture and code
- **D**egradation **C**ause **A**nalysis
  - Identify the reasons for/causes of architectural inconsistencies, identify actual violations
- **R**ecommending **R**epair **A**ctions
  - Recommend refactoring of implementation or architectural adaptations to resolve degradation

ML in Architecture Recovery & Consistency

- Architecture Recovery often understood as: clustering implementation elements into architectural components
- Clustering a typical task in unsupervised learning

- There are more complex architectural concepts like patterns and guidelines
- Architectural decisions affect more than the system decomposition
- Result of recovery is most often not the intended but the implemented architecture

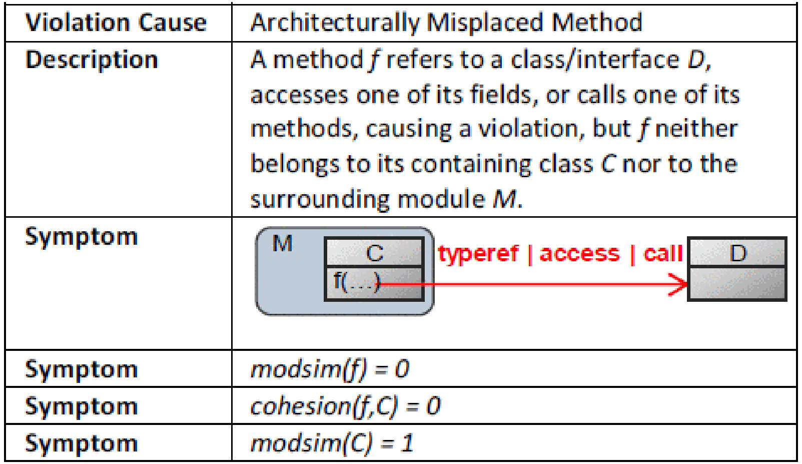
Main idea: express architectural concepts in terms of code-based features



	E1	E2	E3	E4	E5	E6	F	19	E20	E21	E22	E23	E24	E25	E26
isClass	0	1	1	0	1	1		1	1	1	1	1	1	1	0
isInterface	0	0	0	0	0	0		0	0	0	0	0	0	0	1
isAbstractClass	1	0	0	1	0	0		0	0	0	0	0	0	0	0
isPackage	0	0	0	0	0	0	(		0	0	0	0	0	0	0
hasMethodContainingPrimitivTypes	1	1	1	1	1	1			0	1	0	1	1	0	1
hasMethodContainingNotPrimitivTypes	0	1	1	1	0	0		1	1	1	1	1	0	1	0
hasClassVariabels	1	1	1	0	0	1		0	0	1	1	1	1	1	0
hasPrimitivClassVariabels	0	1	0	0	0	1			0	0	1	1	1	1	0
hasNotPrimitivClassVariabels	1	0	1	0	0	0	(		0	1	1	1	0	0	0
hasStaticClassVariabels	0	1	1	0	0	0			0	0	0	0	0	0	0
hasMethods	1	1	1	1	1	1		1	1	1	1	1	1	1	1
hasGetOrSetMethods	0	1	0	1	0	1		0	0	1	1	1	1	1	0
hasPrivateMethods	1	1	0	0	1	1			1	0	0	0	0	0	0

ML in Degradation Cause Analysis

- Idea: express causes as structural patterns + metrics



Violation Cause	Architecturally Misplaced Method
Description	A method *f* refers to a class/interface *D*, accesses one of its fields, or calls one of its methods, causing a violation, but *f* neither belongs to its containing class *C* nor to the surrounding module *M*.
Symptom	M C f(…) typeref | access | call D
Symptom	*modsim(f) = 0*
Symptom	*cohesion(f,C) = 0*
Symptom	*modsim(C) = 1*

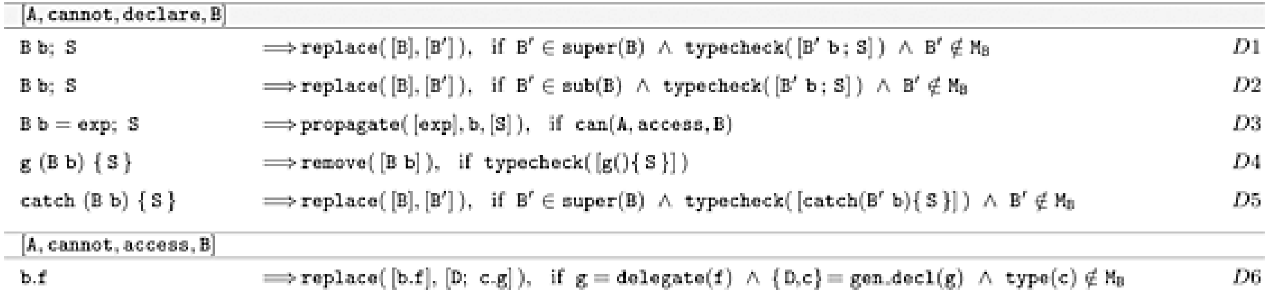
S. Herold, M. English, J. Buckley, S. Counsell and M. Ó. Cinnéide, "Detection of violation causes in reflexion models," *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER).*

- Challenges
  - Interpretation of metric values for violations instances – symptoms express “ideal” case
  - Overlap with other violation causes

- Use of ML techniques
  - Classification of “safe” violation cause instances
  - Discovery of novel violation causes

ML in Recommending Repair Actions

- Refactoring recommender



⌈A, cannot, declare, B⌉		
B b; S	$\Longrightarrow$ replace(⌈B⌉, ⌈B′⌉), if $B' \in$ super(B) $\wedge$ typecheck(⌈B′ b ; S⌉) $\wedge$ $B' \notin M_B$	$D1$
B b; S	$\Longrightarrow$ replace(⌈B⌉, ⌈B′⌉), if $B' \in$ sub(B) $\wedge$ typecheck(⌈B′ b ; S⌉) $\wedge$ $B' \notin M_B$	$D2$
B b = exp; S	$\Longrightarrow$ propagate(⌈exp⌉, b, ⌈S⌉), if can(A, access, B)	$D3$
g (B b) { S }	$\Longrightarrow$ remove(⌈B b⌉), if typecheck(⌈g(){ S }⌉)	$D4$
catch (B b) { S }	$\Longrightarrow$ replace(⌈B⌉, ⌈B′⌉), if $B' \in$ super(B) $\wedge$ typecheck(⌈catch(B′ b){ S }⌉) $\wedge$ $B' \notin M_B$	$D5$
⌈A, cannot, access, B⌉		
b.f	$\Longrightarrow$ replace(⌈b.f⌉, ⌈D; c.g⌉), if g = delegate(f) $\wedge$ {D,c} = gen_decl(g) $\wedge$ type(c) $\notin M_B$	$D6$

Terra, R., Valente, M. T., Czarnecki, K., and Bigonha, R. S. (2015), A recommendation system for repairing violations detected by static architecture conformance checking, *Softw. Pract. Exper.*, 45, 315– 342,

- Fixed, predefined
  - Set of rules
  - Priorities of rules
  - Action parts

- Machine Learning could support
  - Adapt priorities by considering additional features and observing acceptance/rejection of recommendations
  - Observe additional manual actions after accepting recommendations and refine action parts / learn new rules

Conclusion

- We believe there is huge potential for ML in software architecture maintenance and evolution
- Access to data is expected to be a huge challenge
- Validated intended architectures and degradation analysis by experts needed for training

Thanks for your attention!

Any questions?

sebastian.herold@kau.se