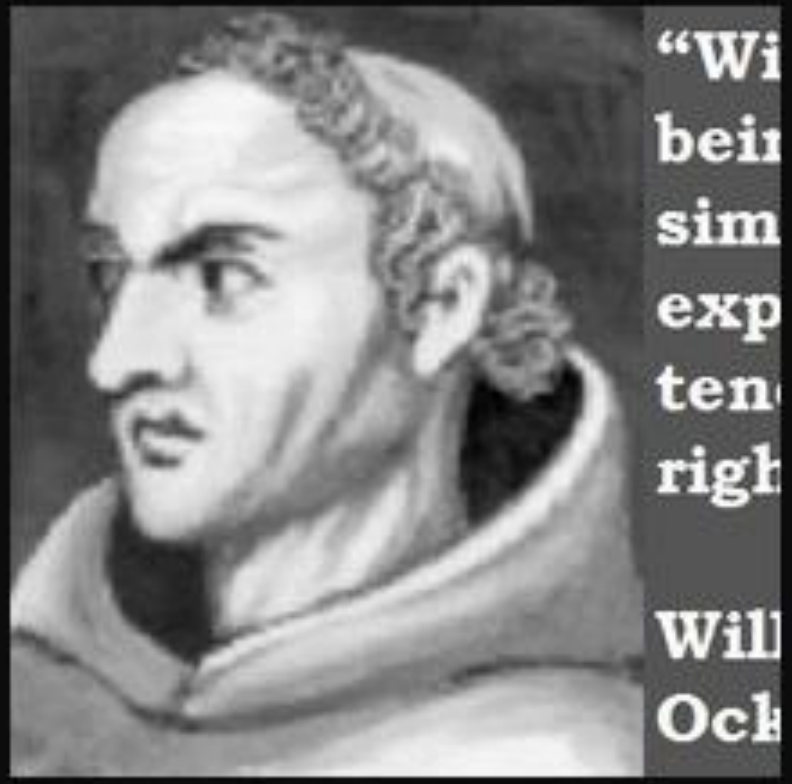
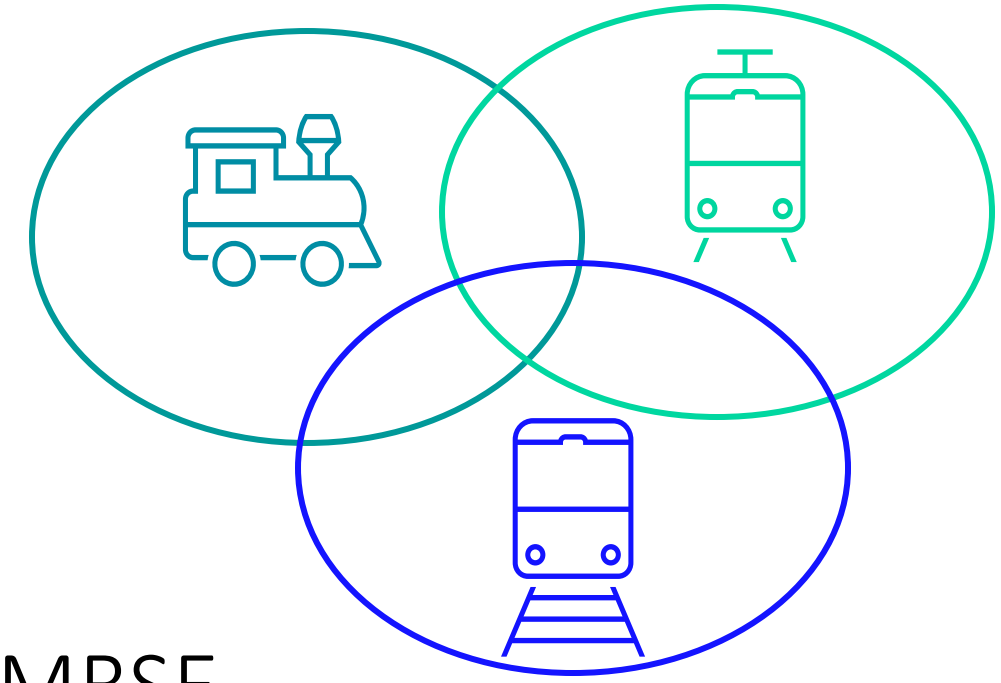
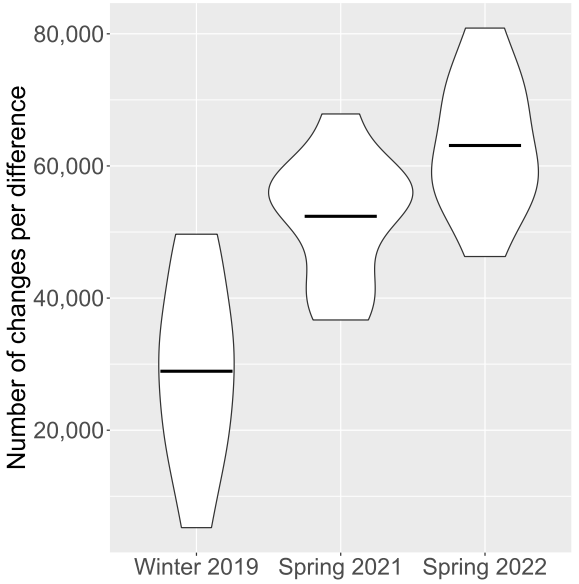


Never increase, beyond what is necessary, the number of entities required to explain anything.

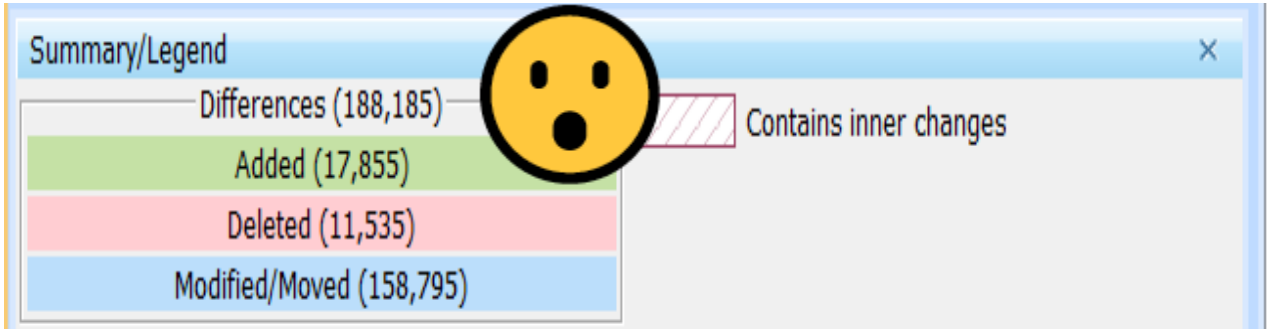
Problem Setting – MBSE meets Managed Cloning



MBSE
Product Family
For Train Control Software



Drift

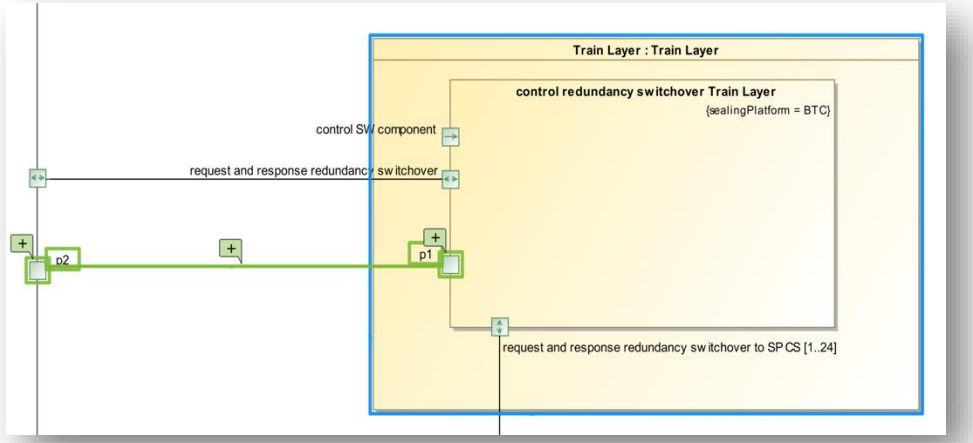


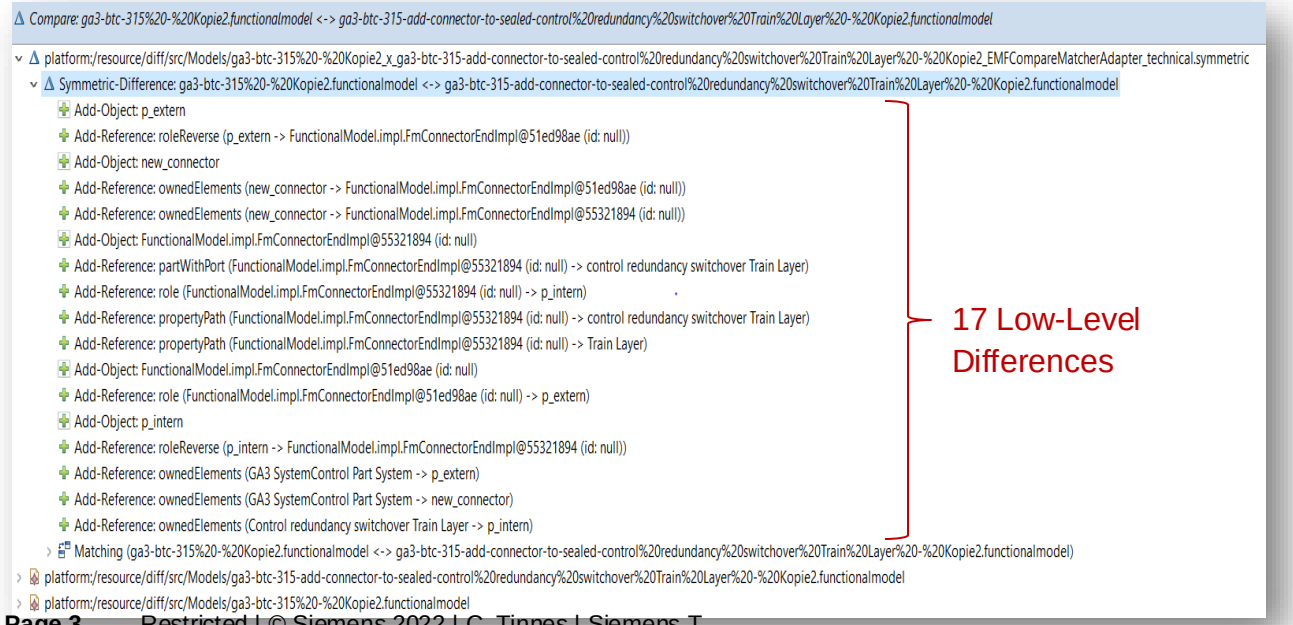
Tinnes, Christof, et al. "Sometimes You Have to Treat the Symptoms: Tackling Model Drift in an Industrial Clone-and-Own Software Product Line." *2022* 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22). ACM, 2022.

Semantic Lifting to the Rescue

Example Difference:

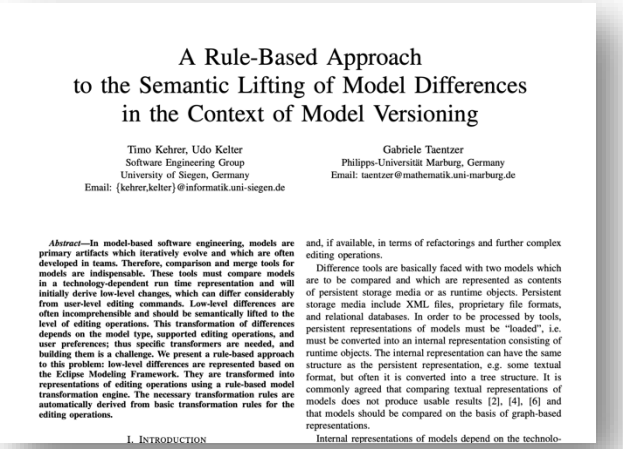
A new `Port` and `Connector` is added to a sealed `Software Component`





17 Low-Level Differences

Semantic Lifting



A Rule-Based Approach
to the Semantic Lifting of Model Differences
in the Context of Model Versioning

Timo Kehrer, Udo Kelter
Software Engineering Group
University of Siegen, Germany
Email: {kehrer,kelter}@informatik.uni-siegen.de

Gabriele Taentzer
Philipps-Universität Marburg, Germany
Email: taentzer@mathematik.uni-marburg.de

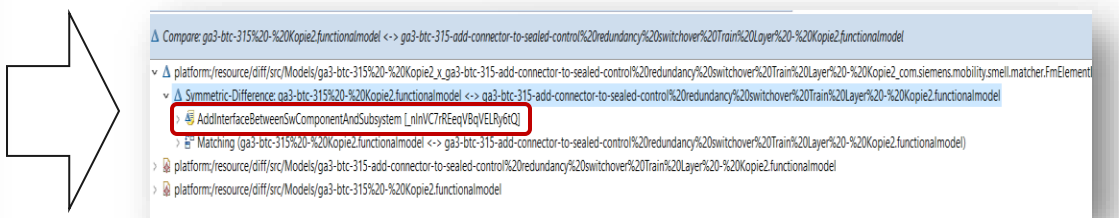
Abstract—In model-based software engineering, models are primary artifacts which iteratively evolve and which are often developed in teams. Therefore, comparison and merge tools for models are indispensable. These tools must compare models in a technology-dependent run time representation and will initially derive low-level changes, which can differ considerably from user-level editing commands. Low-level differences are often incomprehensible and should be semantically lifted to the level of editing operations. This transformation of differences depends on the model type, supported editing operations, and user preferences; thus specific transformers are needed, and building them is a challenge. We present a rule-based approach to this problem: low-level differences are represented based on the Eclipse Modeling Framework. They are transformed into representations of editing operations using a rule-based model transformation engine. The necessary transformation rules are automatically derived from basic transformation rules for the editing operations.

I. Introduction

and, if available, in terms of refactorings and further complex editing operations.

Difference tools are basically faced with two models which are to be compared and which are represented as contents of persistent storage media or as runtime objects. Persistent storage media include XML files, proprietary file formats, and relational databases. In order to be processed by tools, persistent representations of models must be "loaded", i.e. must be converted into an internal representation consisting of runtime objects. The internal representation can have the same structure as the persistent representation, e.g. some textual format, but often it is converted into a tree structure. It is commonly agreed that comparing textual representations of models does not produce usable results [2], [4], [6] and that models should be compared on the basis of graph-based representations.

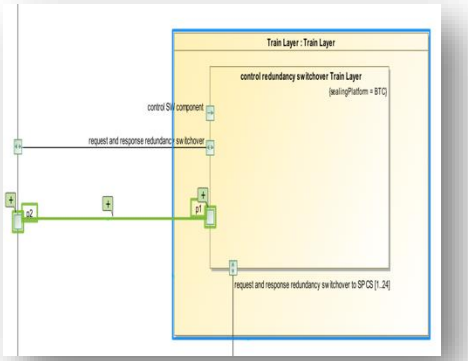
Internal representations of models depend on the technolo-

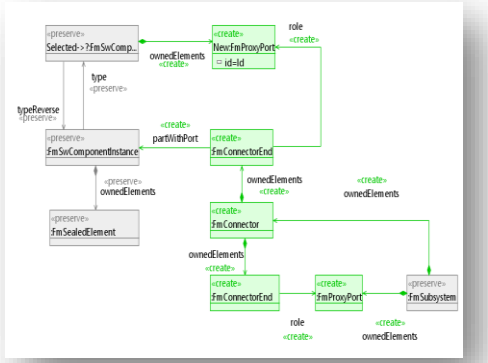


BUT…

1

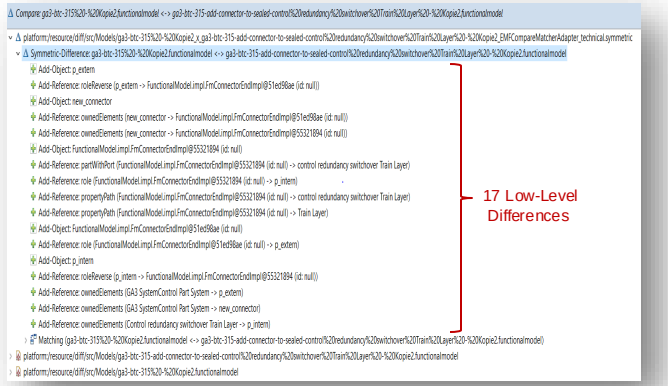
not easy to specify

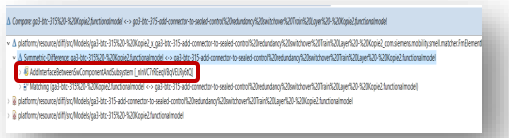




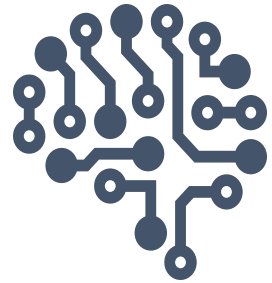
2

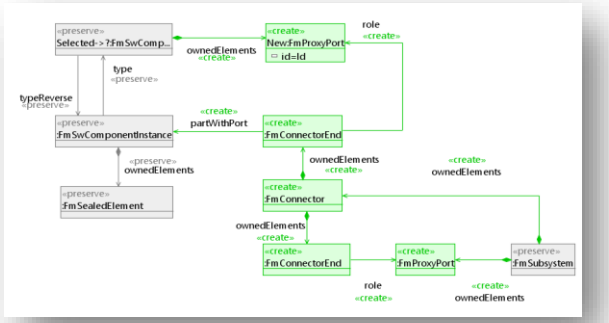
unknown or tacit knowledge





Idea – Mining Editing Patterns





Collection of Model Differences

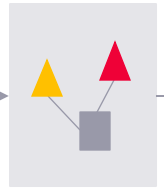
Change Graphs

Pattern Graphs

Edit Rules

Generate Change Graphs

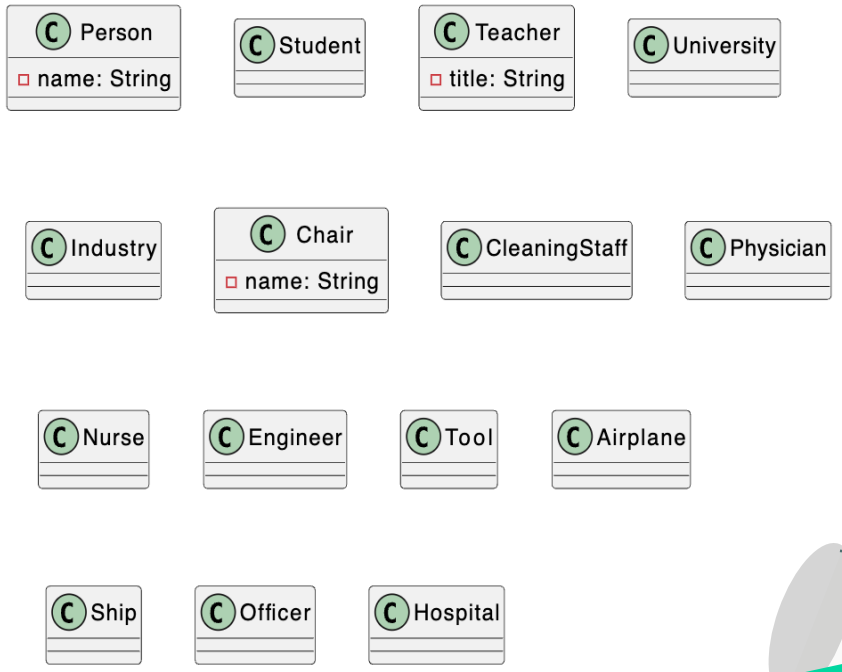
Derive “meaningful” subgraphs



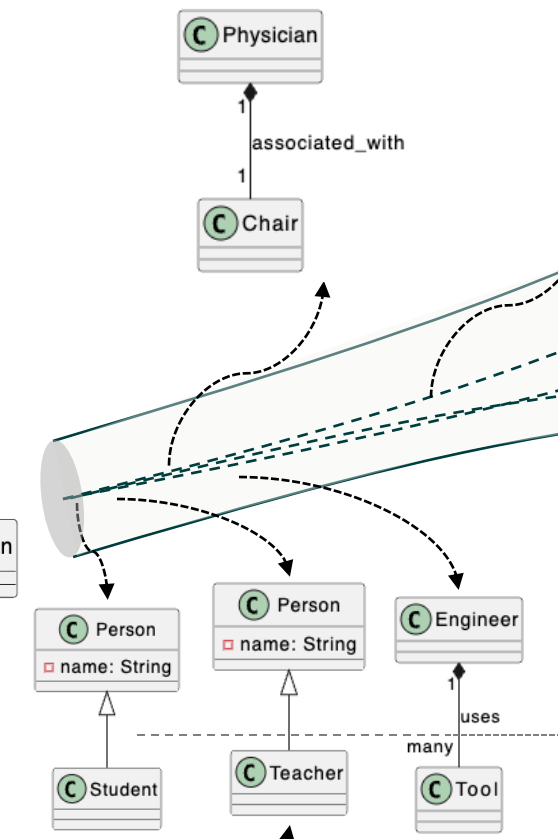
Generate Edit Rules

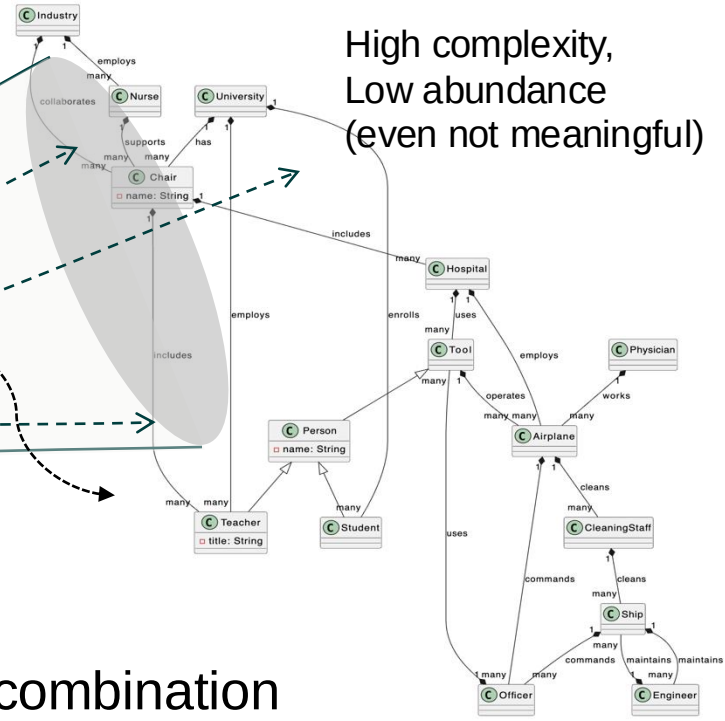
Evolution is Biased

Initial Building Blocks



Real-world evolution is **biased** in the space of possible evolution… Can we make use of this bias?

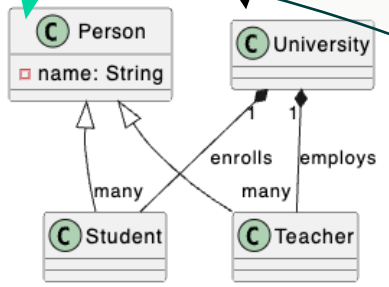


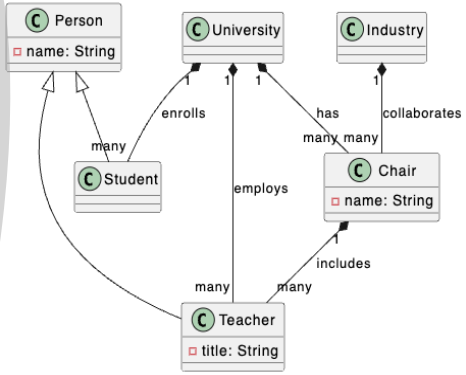


High complexity,
Low abundance
(even not meaningful)

Unbiased combination

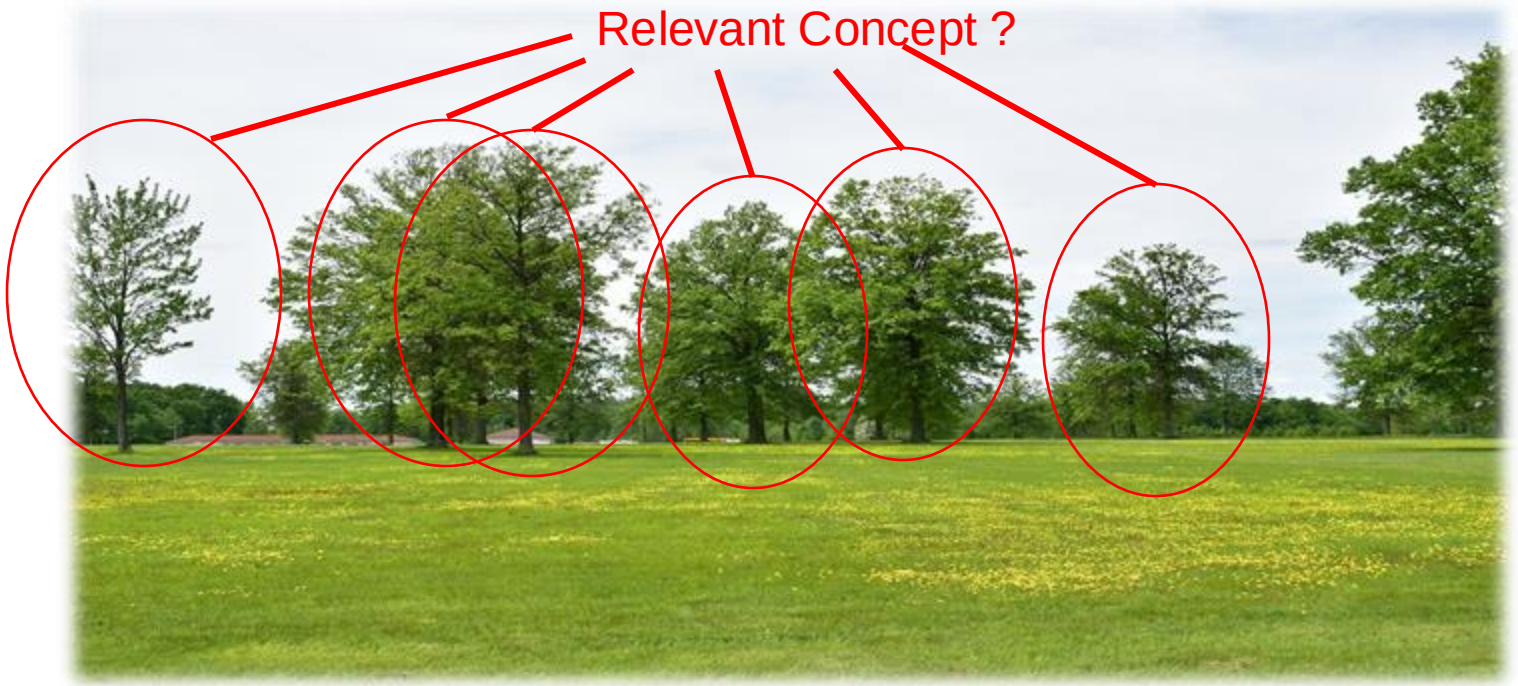
Biased combination





High complexity,
High abundance

What is a Pattern? Exampels



Relevant Concept ?

В лесу много зеленых деревьев. Деревья часто бывают зелеными.

Relevant Concept ?

Relevant Concept ?

В лесу много зеленых деревьев. Деревья часто бывают зелеными.

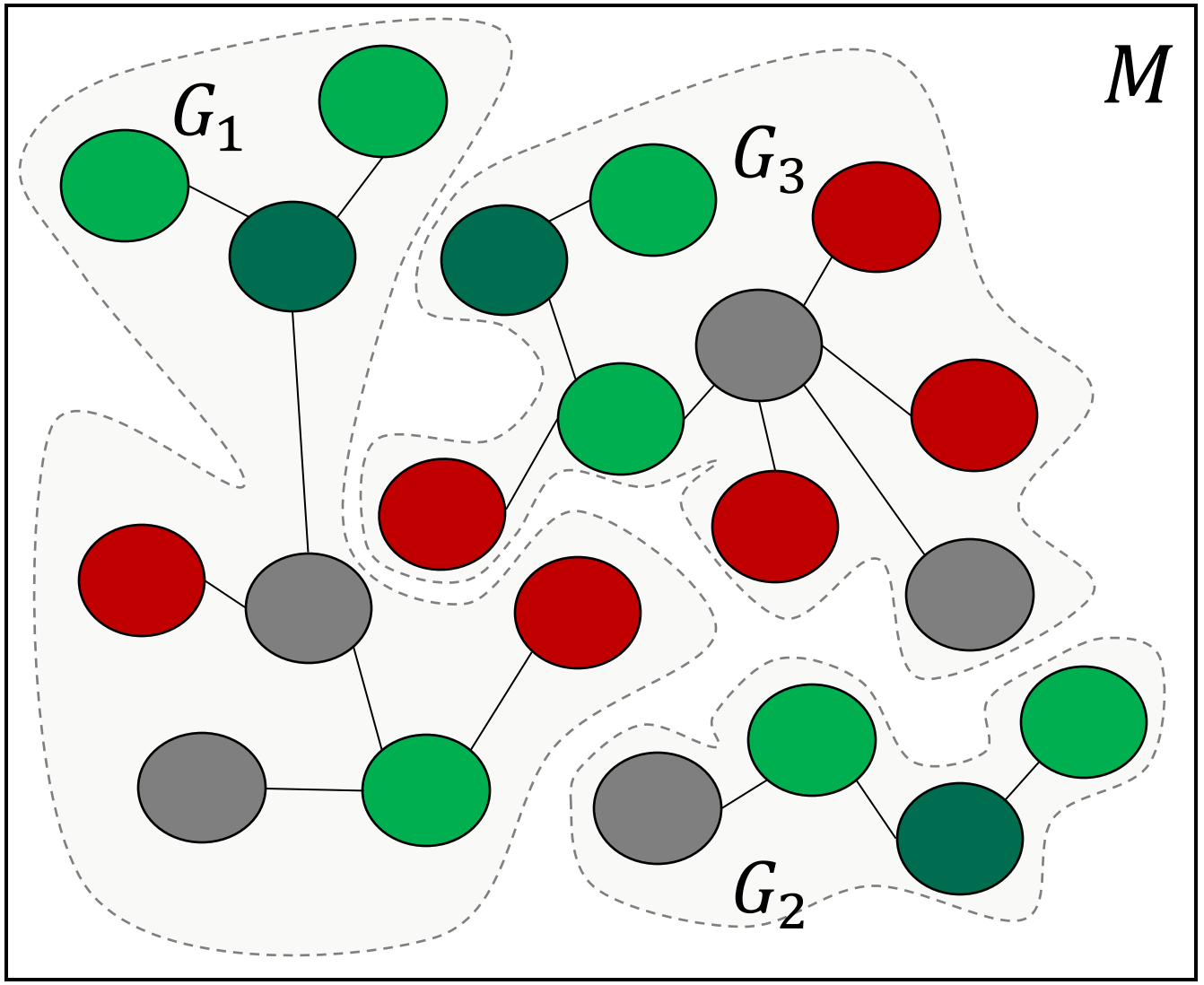
the pattern subgraph
~ edit operation

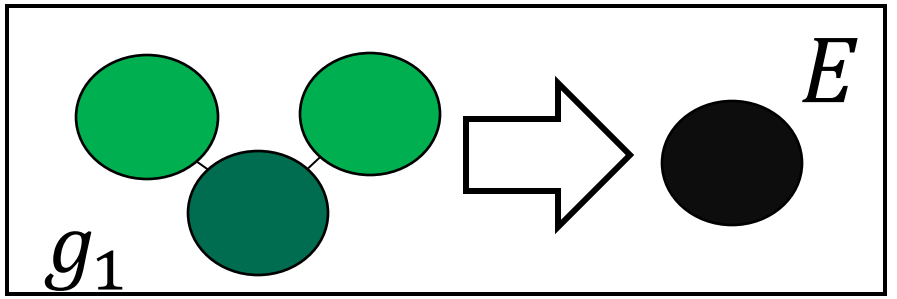
number of occurences
of the pattern

number of vertices

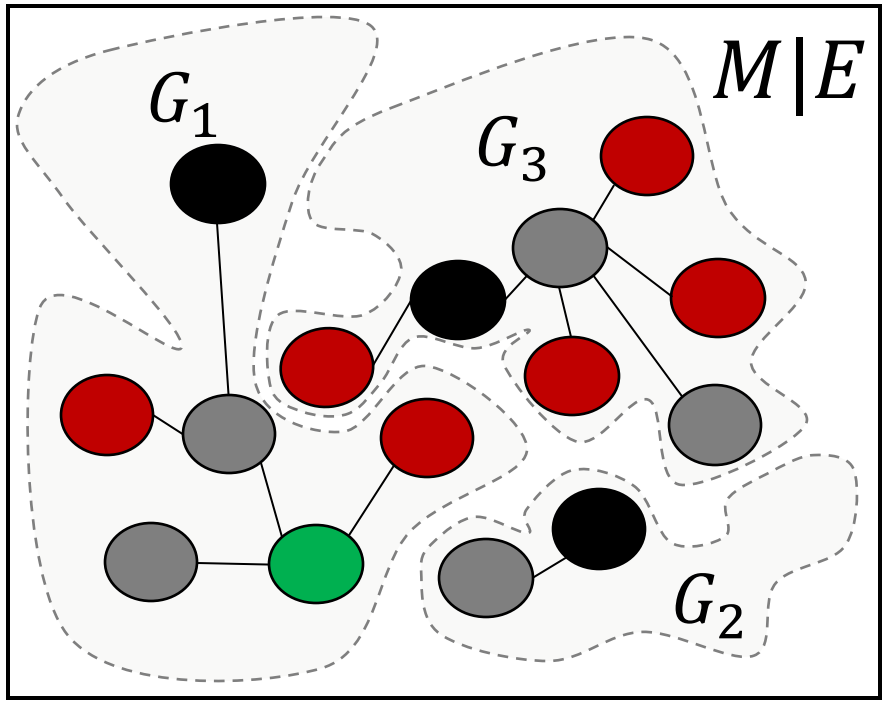
number of edges

$$\mathrm{compr}(g) = \big(\mathrm{supp}(g) - 1\big) \cdot \big(|V_g| + |E_g|\big)$$

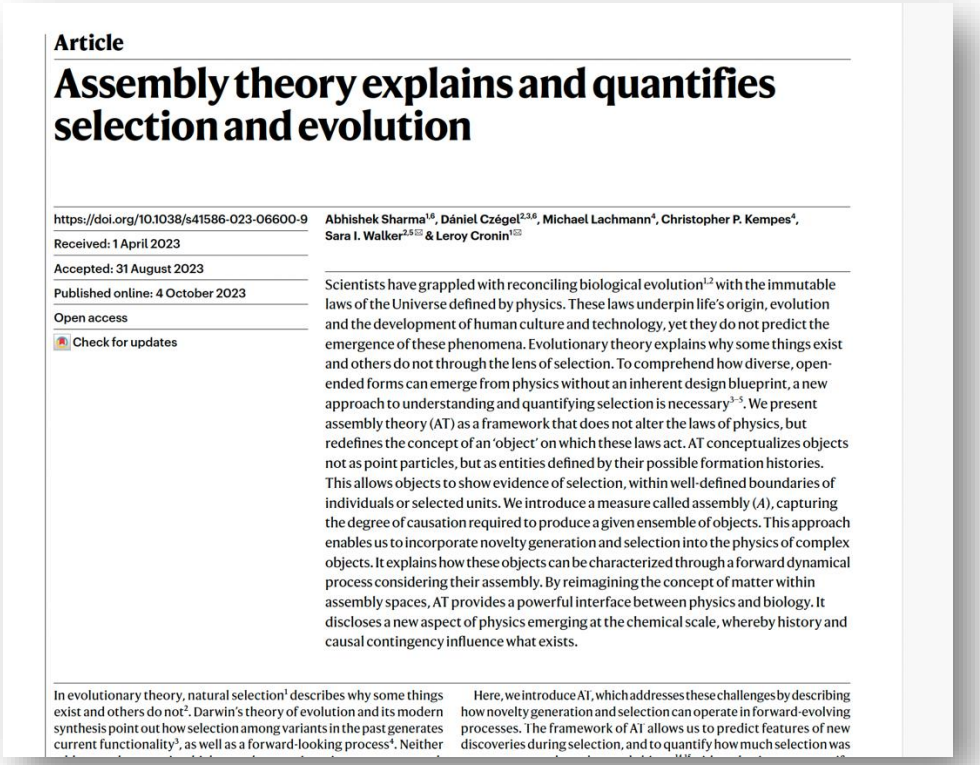




+



Assembly Theory: “Amount of Selection”



Article

Assembly theory explains and quantifies selection and evolution

https://doi.org/10.1038/s41586-023-06600-9
Received: 1 April 2023
Accepted: 31 August 2023
Published online: 4 October 2023
Open access
Check for updates

Abhishek Sharma[1,6], Dániel Czégel[2,3,6], Michael Lachmann[4], Christopher P. Kempes[4], Sara I. Walker[2,5] & Leroy Cronin[1]

Scientists have grappled with reconciling biological evolution[1,2] with the immutable laws of the Universe defined by physics. These laws underpin life's origin, evolution and the development of human culture and technology, yet they do not predict the emergence of these phenomena. Evolutionary theory explains why some things exist and others do not through the lens of selection. To comprehend how diverse, open-ended forms can emerge from physics without an inherent design blueprint, a new approach to understanding and quantifying selection is necessary[3–5]. We present assembly theory (AT) as a framework that does not alter the laws of physics, but redefines the concept of an 'object' on which these laws act. AT conceptualizes objects not as point particles, but as entities defined by their possible formation histories. This allows objects to show evidence of selection, within well-defined boundaries of individuals or selected units. We introduce a measure called assembly (A), capturing the degree of causation required to produce a given ensemble of objects. This approach enables us to incorporate novelty generation and selection into the physics of complex objects. It explains how these objects can be characterized through a forward dynamical process considering their assembly. By reimagining the concept of matter within assembly spaces, AT provides a powerful interface between physics and biology. It discloses a new aspect of physics emerging at the chemical scale, whereby history and causal contingency influence what exists.

In evolutionary theory, natural selection[1] describes why some things exist and others do not[2]. Darwin's theory of evolution and its modern synthesis point out how selection among variants in the past generates current functionality[3], as well as a forward-looking process[4]. Neither

Here, we introduce AT, which addresses these challenges by describing how novelty generation and selection can operate in forward-evolving processes. The framework of AT allows us to predict features of new discoveries during selection, and to quantify how much selection was

We can show:

„Objects“ maximizing „**selection**“

=

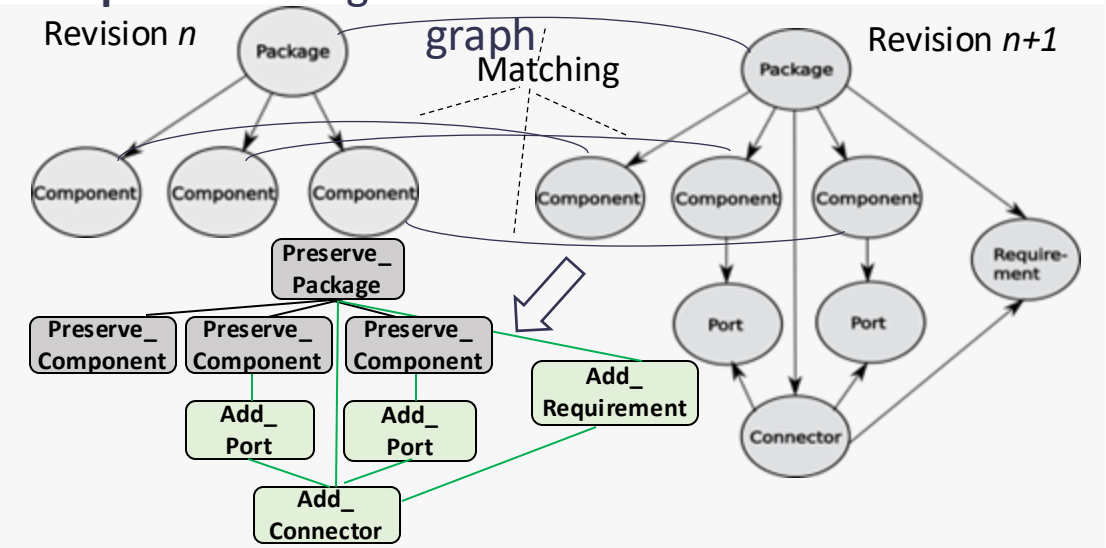
„Objects“ **defined by** maximal compression

$$A = \sum_{i=1}^{N} e^{a_i} \left(\frac{n_i - 1}{N_T} \right)$$

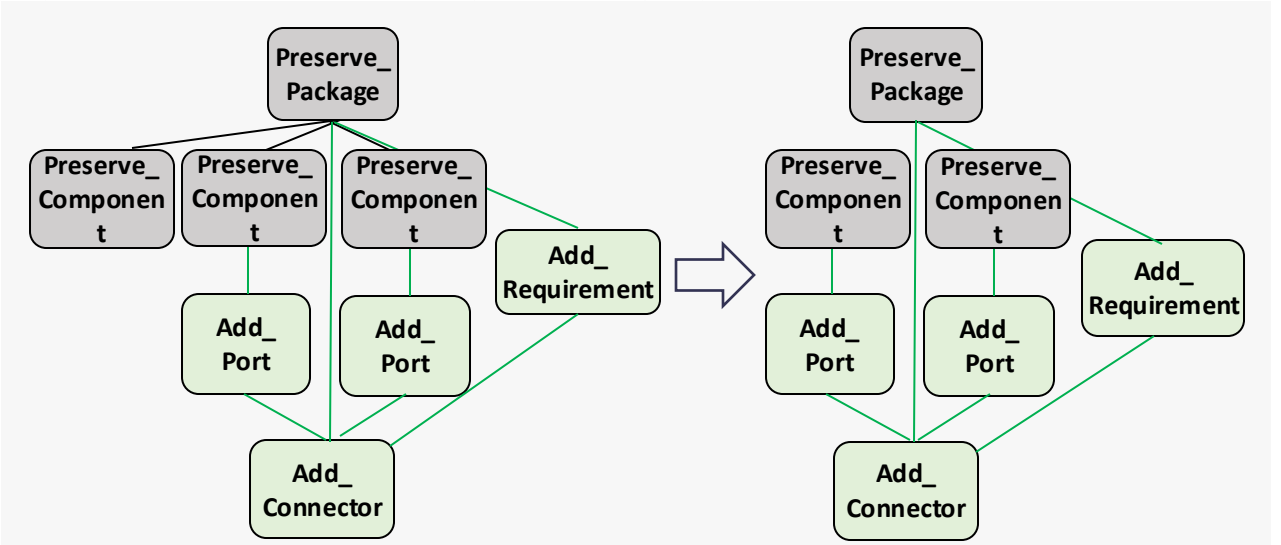
A – **assembly** of the ensemble
a_i - **assembly** index of object I
n_i - **copy number** of object i
N – **total number of unique object**
N_T - **total number of objects** in the ensemble

Ockham – Compression-based Discovery of Editing Patterns

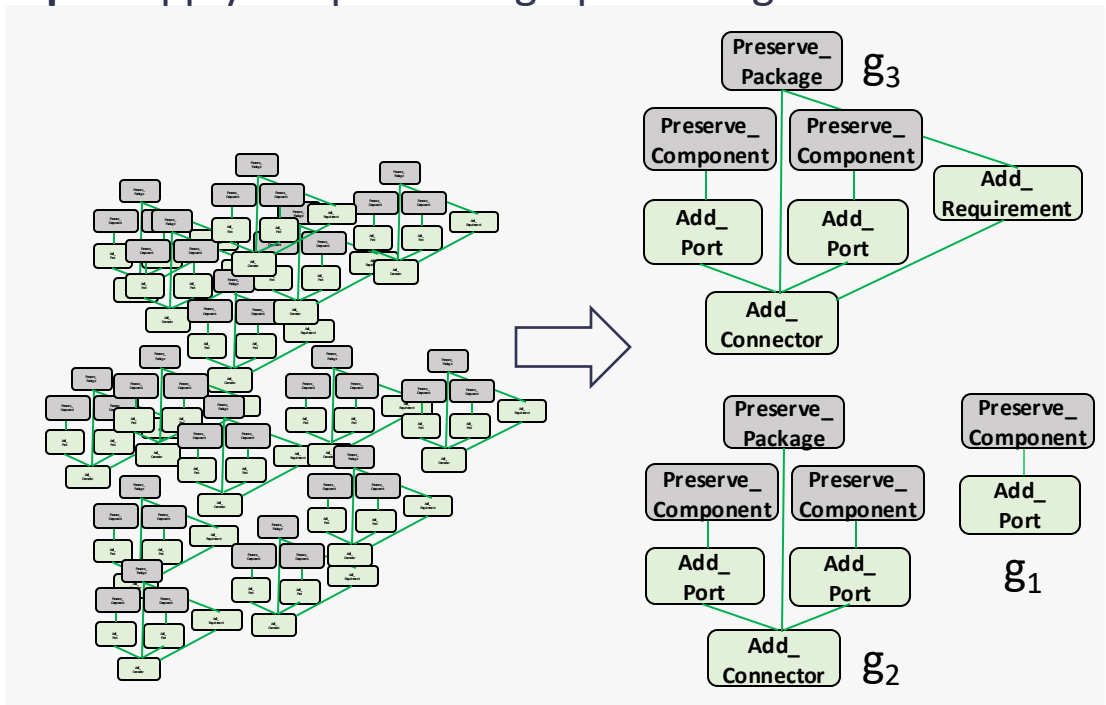
Step 1: Matching and derivation of difference



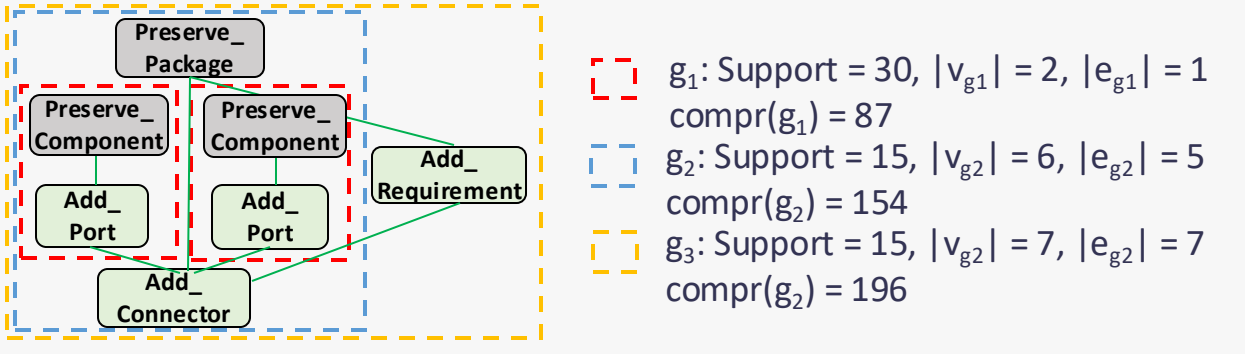
Step 2: Derivation of simple change graph (SCG)



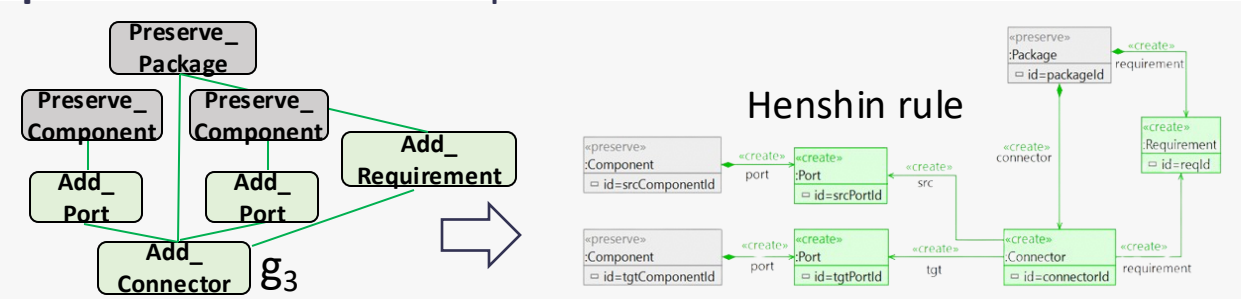
Step 3: Apply Frequent Subgraph Mining



Step 4: Select most relevant subgraphs (ranking)

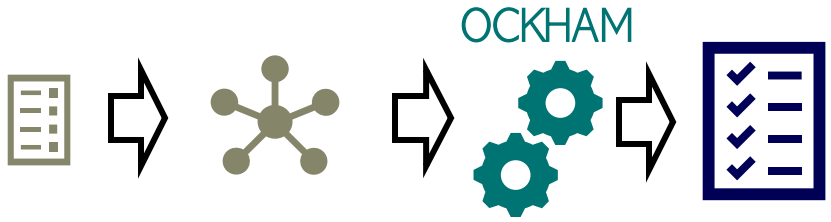


Step 5: Generate the edit operations

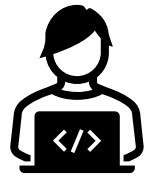


Experiments and Results

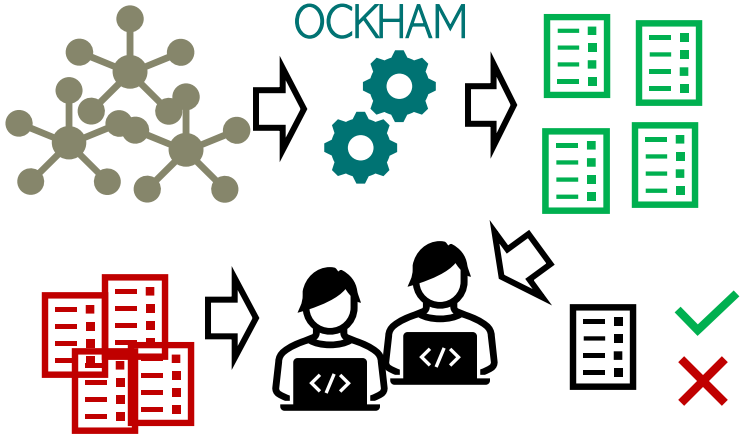
Controled Experiment



	MAP@2	MAP@5	MAP@10	MAP@∞
Compression	0.955	0.969	0.969	0.969
Frequency	0.013	0.127	0.152	0.190



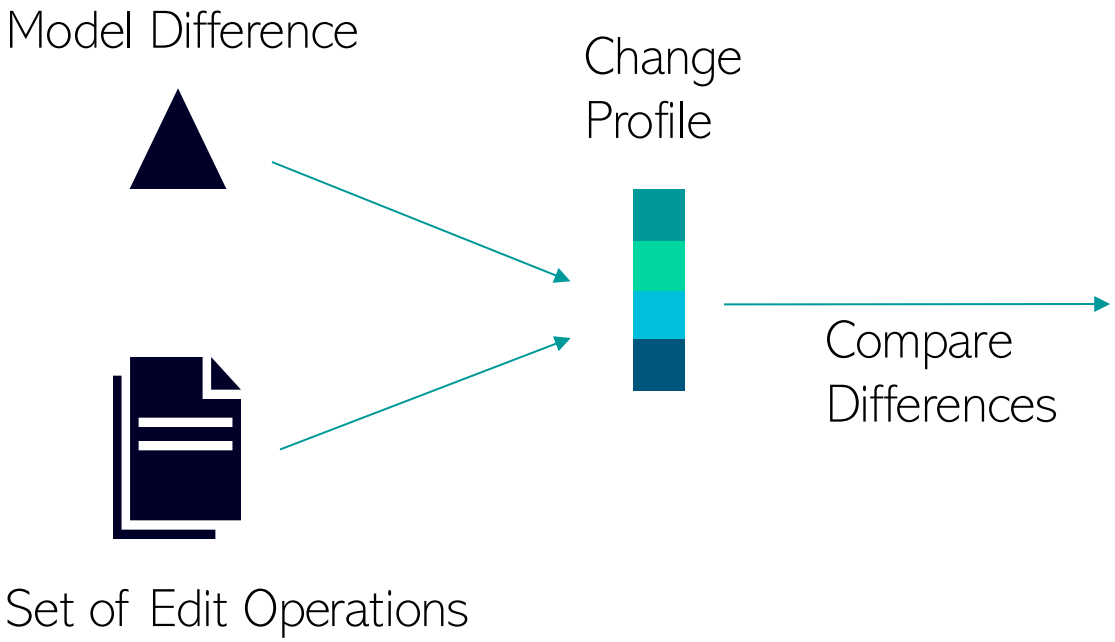
Real-World Experiment

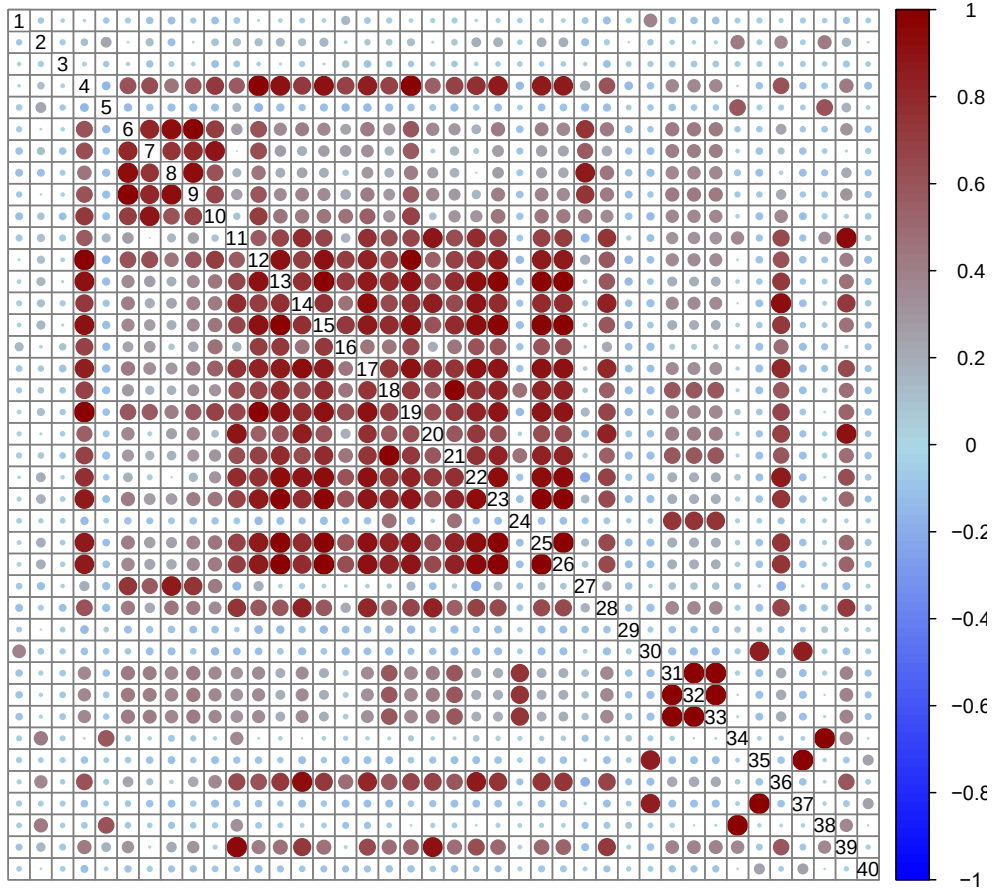


Participant	Mean mined	Mean random	p-value (t-test)	p-value (Wilcoxon)
P1	3.20	1.68	$11.8 \cdot 10^{-5}$	$29.0 \cdot 10^{-5}$
P2	4.04	2.76	$16.6 \cdot 10^{-4}$	$6.43 \cdot 10^{-3}$
P3	4.32	2.60	$9.30 \cdot 10^{-6}$	$5.87 \cdot 10^{-5}$
P4	4.32	1.08	$2.67 \cdot 10^{-15}$	$3.51 \cdot 10^{-10}$
P5	4.48	1.60	$1.17 \cdot 10^{-11}$	$1.15 \cdot 10^{-7}$
Total	**4.072**	**1.944**	$< 2.2 \cdot 10^{-16}$	$< 2.2 \cdot 10^{-16}$

Tinnes, Christof, et al. "Learning domain-specific edit operations from model repositories with frequent subgraph mining." *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021.

Application Example – Change Profiles

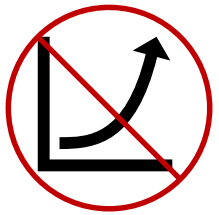




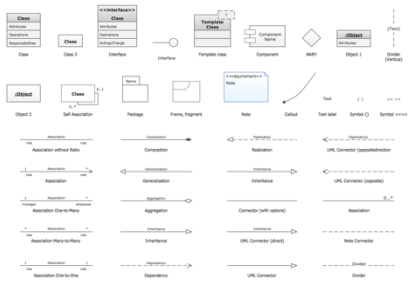
Differences seem to build „clusters“

SIEMENS

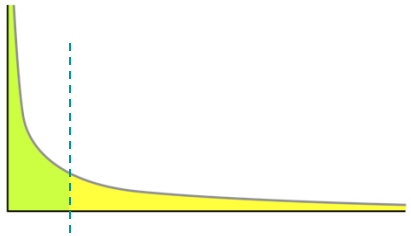
Limitations & Outlook



No scalability of Frequent Subgraph Mining



Limited to “types”– Ockham only captures semantics that is explicitly encoded in the meta-model



No “complete” set, no coverage of the “long tail”

SIEMENS

Database

Patterns/Operations

Distributed Representation (e.g., Neural Network)

Language

Models are abstract.
Models are **description**.
Models are **communication**.

Models —are (100 %)→ Models are —abstract description communication (33 %)→ Models are abstract. Models are **description**. Models are **communication**.

Models —✕ (0%)→ Models are abstract **communication**.

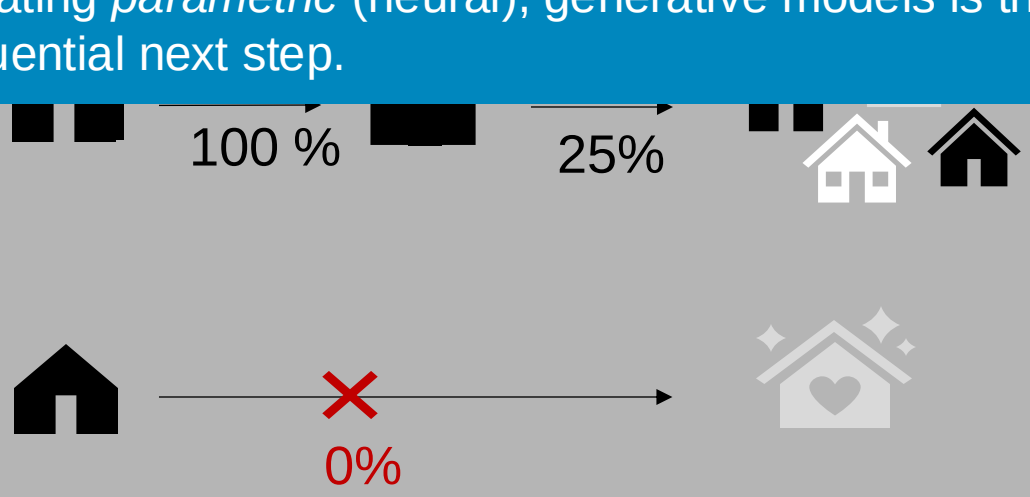
Models —$\mathbb{P} > 0$→ Models are abstract **communication**.

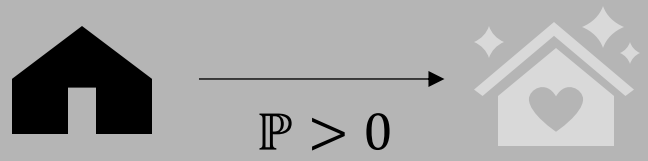
Covering the *Long-Tail* of domain- and project-specific context in software models needs *distributed representations*.

Investigating *parametric* (neural), generative models is therefore a consequential next step.

Language Model

Model





2-Pattern

Generative Model