Software knowledge analytics

as a role model for making sense of the world

INAUGURAL VLOEBERGHS CHAIR LECTURE

Ralf Lämmel, Uni Koblenz, May 2022

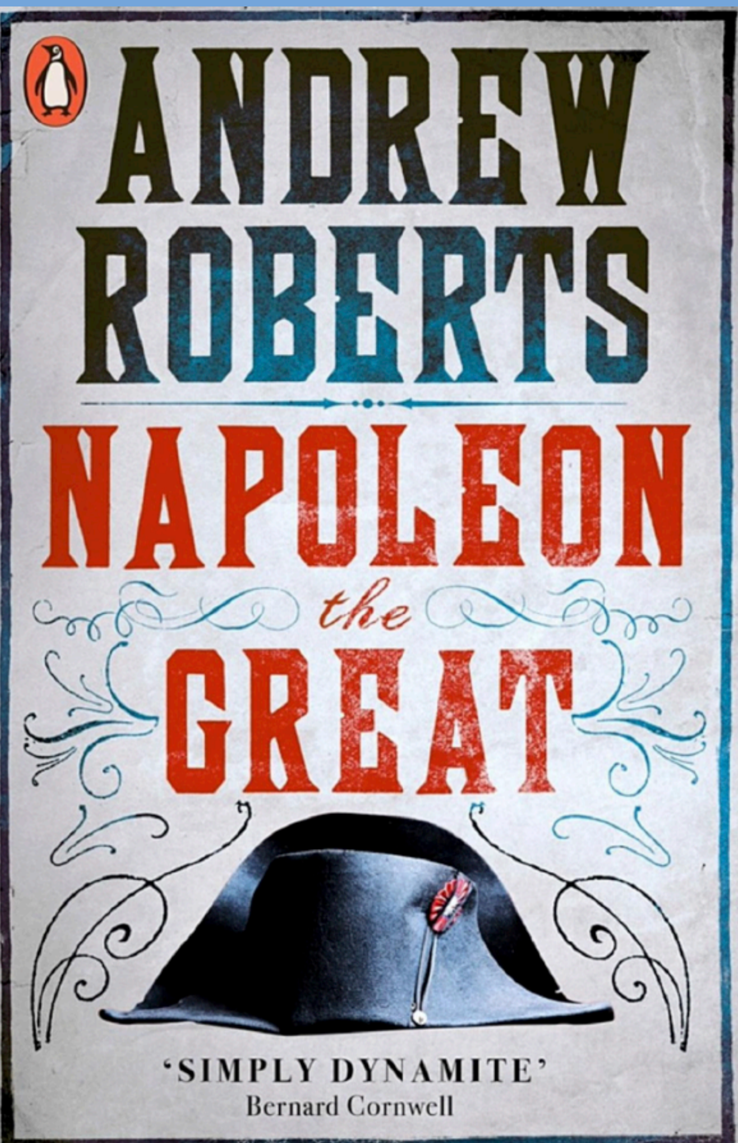
Making sense of the world?

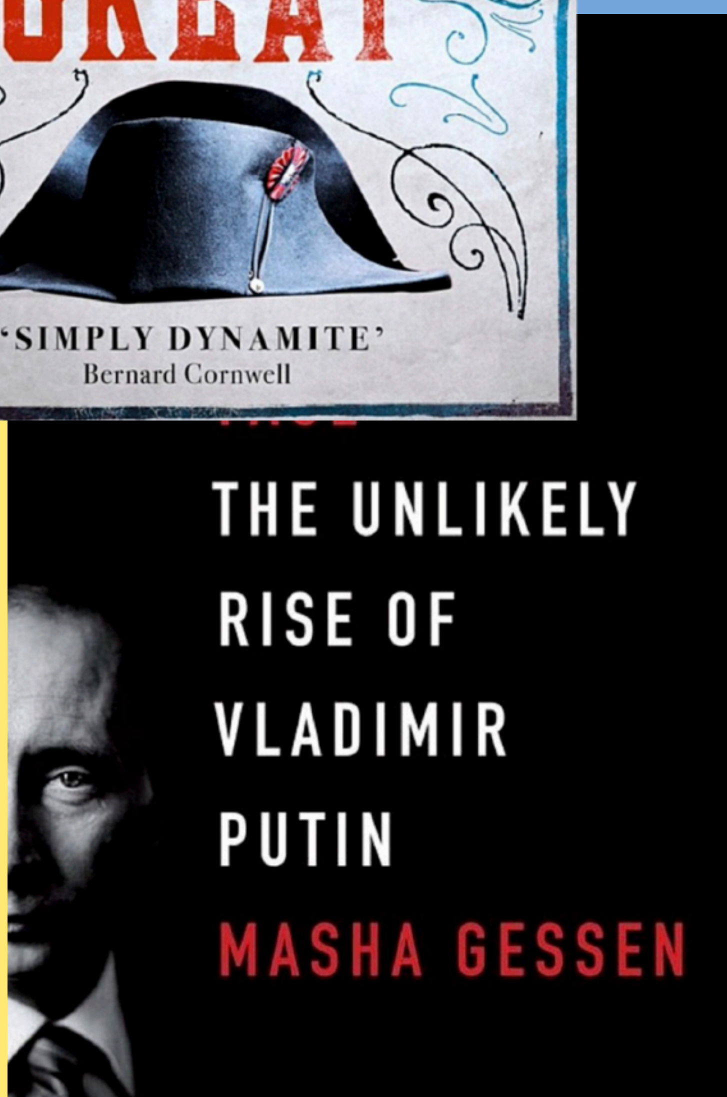
Source: https://www.nature.com/articles/d41586-021-00257-y

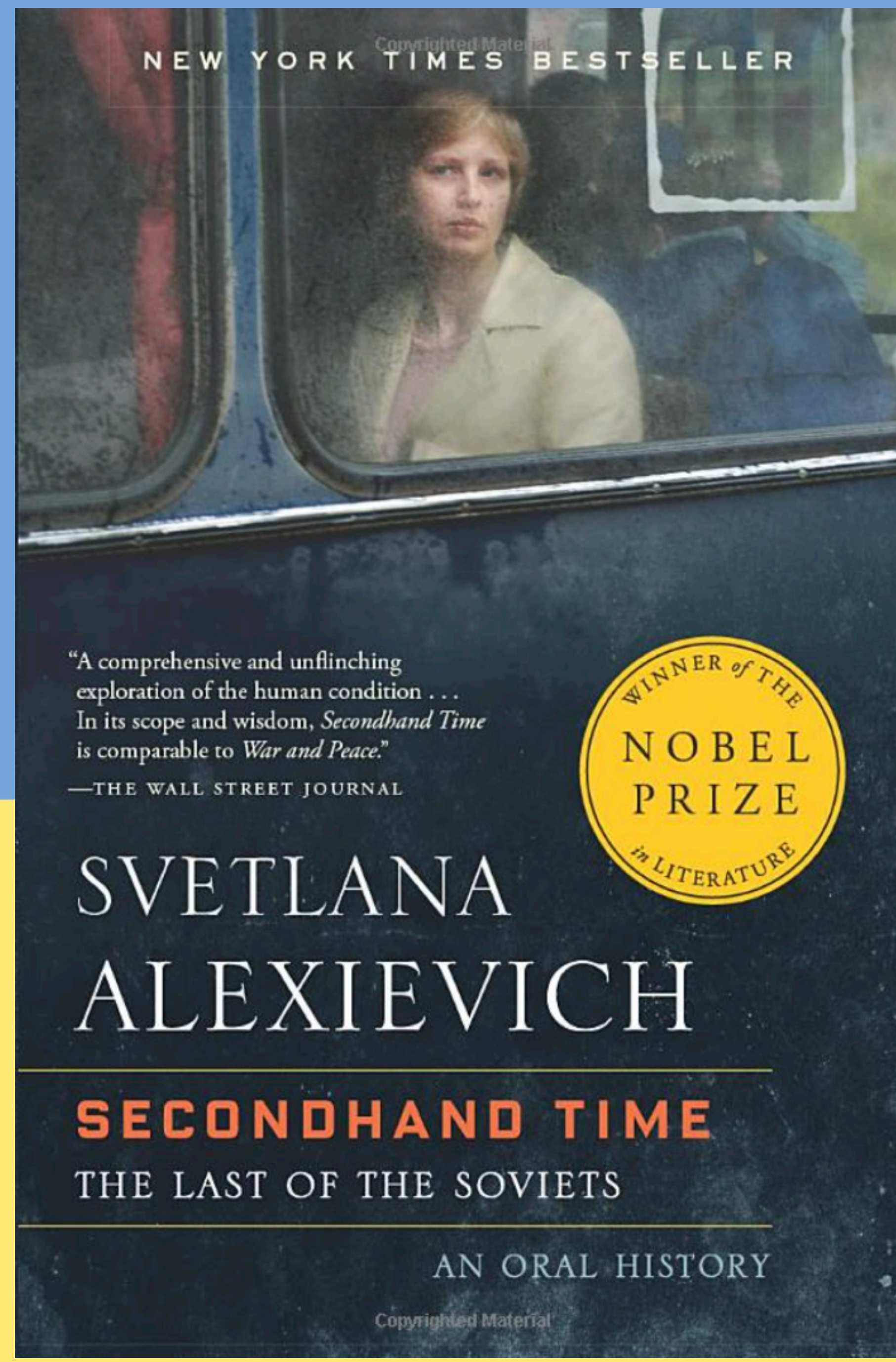
https://www.forbes.com/sites/brucelee/2021/05/30/nashville-shop-sells-not-vaccinated-yellow-star-patches-here-are-the-responses/?sh=10a8dc153435

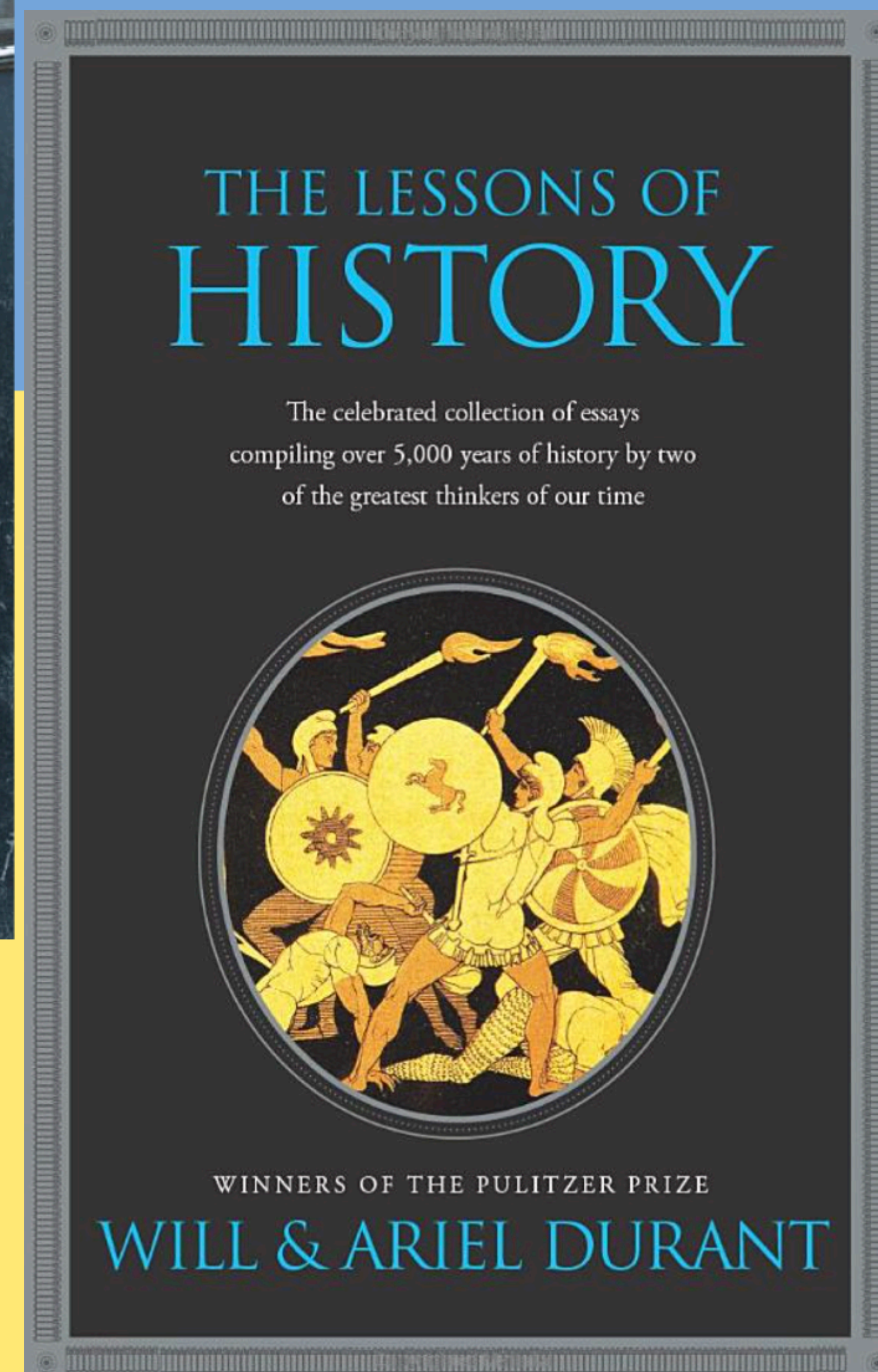
Source: https://it.wikipedia.org/wiki/Don%27t_Look_Up

Making sense of the world?









Software knowledge analytics

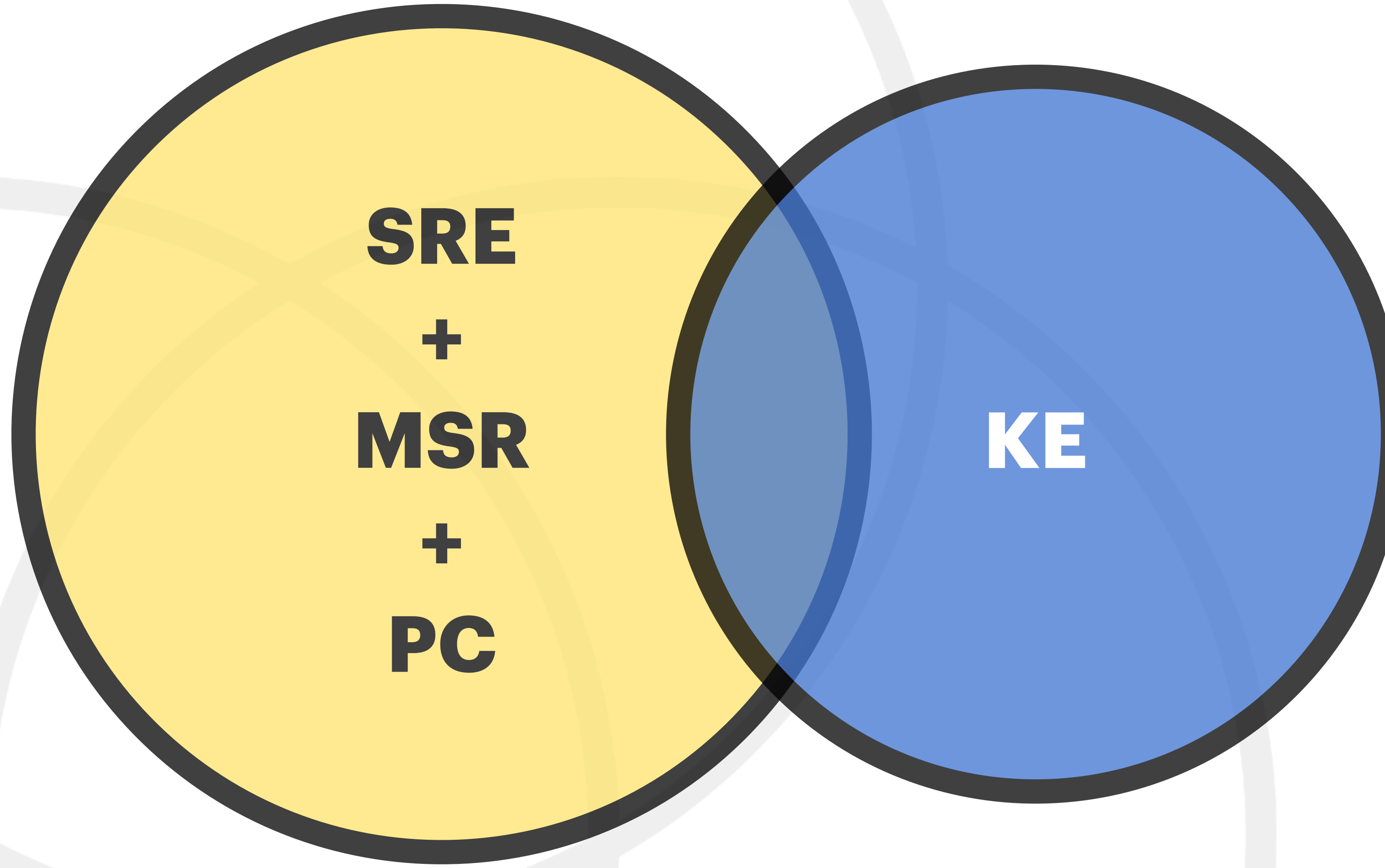
as a role model for

making sense of the world

The Venn diagram of *software knowledge analytics*

Software Reverse Engineering (SRE) “is the practice of analyzing a software system, either in whole or in part, to extract design and implementation information.”

[https://dblp.org/rec/reference/icsec/CipressoS10.html]



Mining Software Repositories (MSR) is the field that “analyzes the rich data available in software repositories, such as version control repositories, mailing list archives, bug tracking systems, issue tracking systems, etc. to uncover interesting and actionable information about software systems, projects and software engineering”.

[https://dblp.org/rec/reference/icsec/CipressoS10.html]

Program comprehension (PC) "is that activity by which software engineers come to an understanding of the behavior of a software system using the source code as the primary reference”.

[https://dblp.uni-trier.de/rec/journals/ac/BennettRW02.html]

Knowledge engineering (KE) “refers to all technical, scientific and social aspects involved in building, maintaining and using knowledge-based systems”.

[https://en.wikipedia.org/wiki/Knowledge_engineering]

A brief history of time

Epoché	Since	Innovation
Programming language theory	1970	Mathematical approach to defining syntax and semantics
Programming language processors	1980	Rapid implementation of language analyses and transformations
Empirical software engineering	1990	Scientific approach to software engineering
Software language engineering	2000	General engineering approach to languages across technical space
Mining software repositories	2000	Scientific approach to analyzing software projects
Software language science	2008	Scientific approach to software language comprehension
Linguistic software architecture	2010	Conceptualized representation of software projects
Knowledge graphs	2015	Semantic data extraction and integration

Table of contents

- *Showcases*
- *Principles*
- *Challenges*

of **Software Knowledge Analytics**

Thus,
this is a bit of a
meta-mythological
presentation on
the subject.

Showcases of
Software Knowledge Analytics

- Software language usage
- Software technology usage
- Software developer profiling
- Work-item prediction
- Ownership management
- ...

Software language usage

A Showcase of Software Knowledge Analytics

Motivation

Actually, we did this empirical research to support other research on query language integration.

- Graph databases are an interesting trend
- Used for knowledge graphs, such as Wikidata
- And in a *Big Data* context at Google et al.

What is the usage of graph-related query languages in open-source projects?

Software language usage

A Showcase of Software Knowledge Analytics

Target Languages

1. **SPARQL** RDF query language (W3C recomm.)
2. **Cypher** Neo4j/openCypher property graph QL
3. **Gremlin** Apache Tinkerpop graph traversal
4. **GraphQL** Graph query and REST replacement

As baseline comparisons: **XQuery** (W3C) and **SQL**

Software language usage

A Showcase of Software Knowledge Analytics

Query example

GraphQL

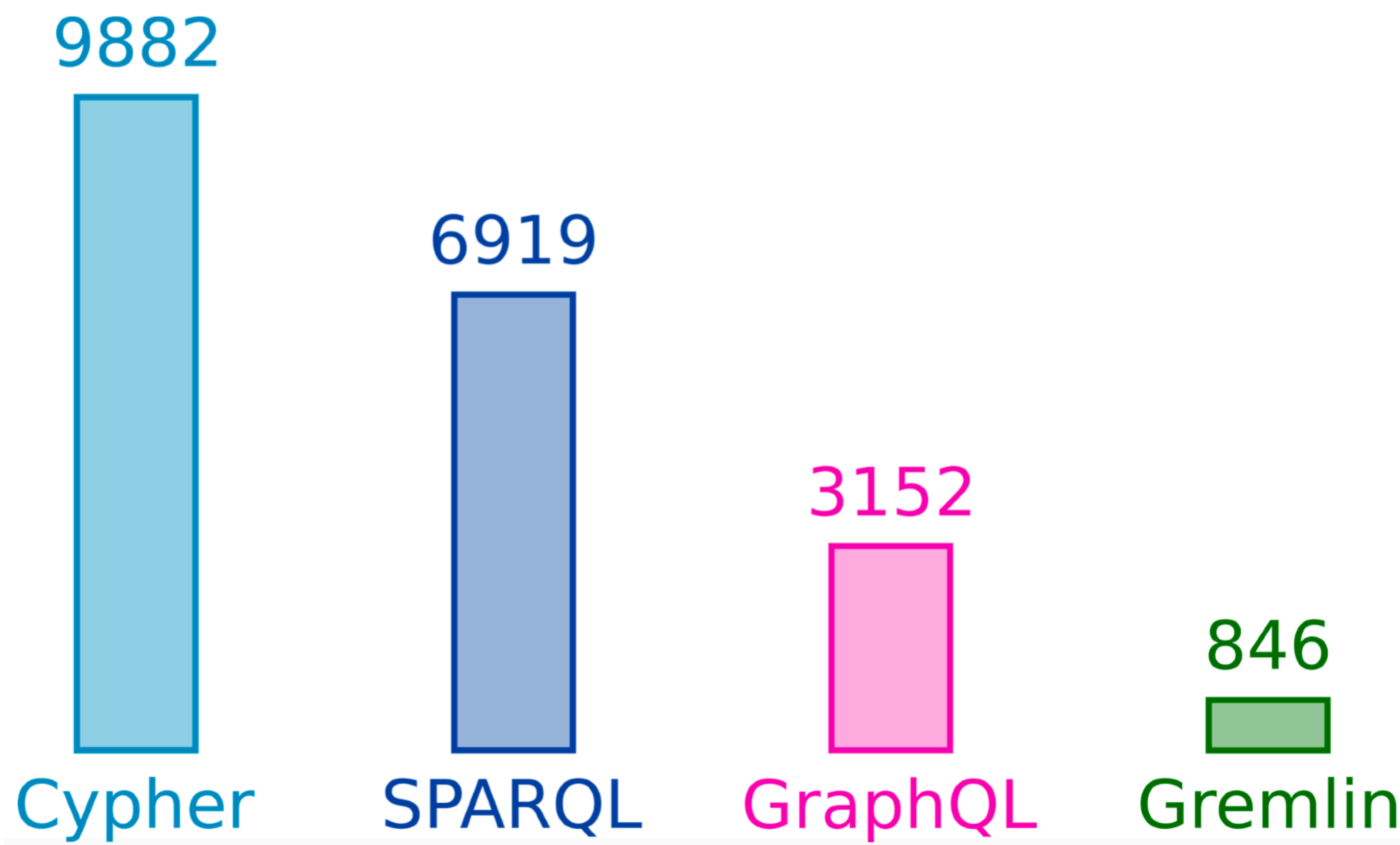
Graph query langugage and REST replacement

```
person {
  name
  knows {
    name
  }
}
```

Software language usage

A Showcase of Software Knowledge Analytics

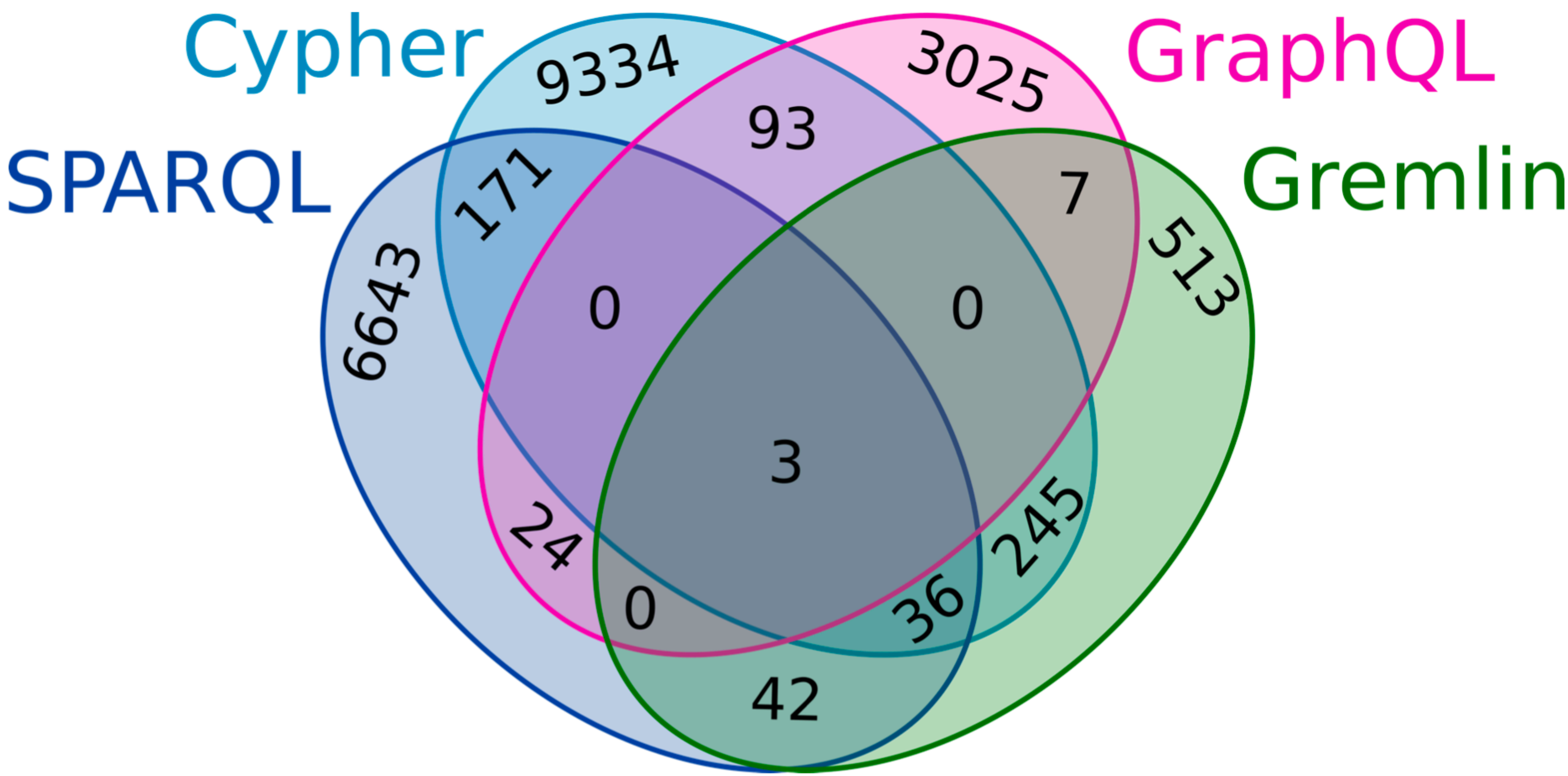
Projects per query language



Software language usage

A Showcase of Software Knowledge Analytics

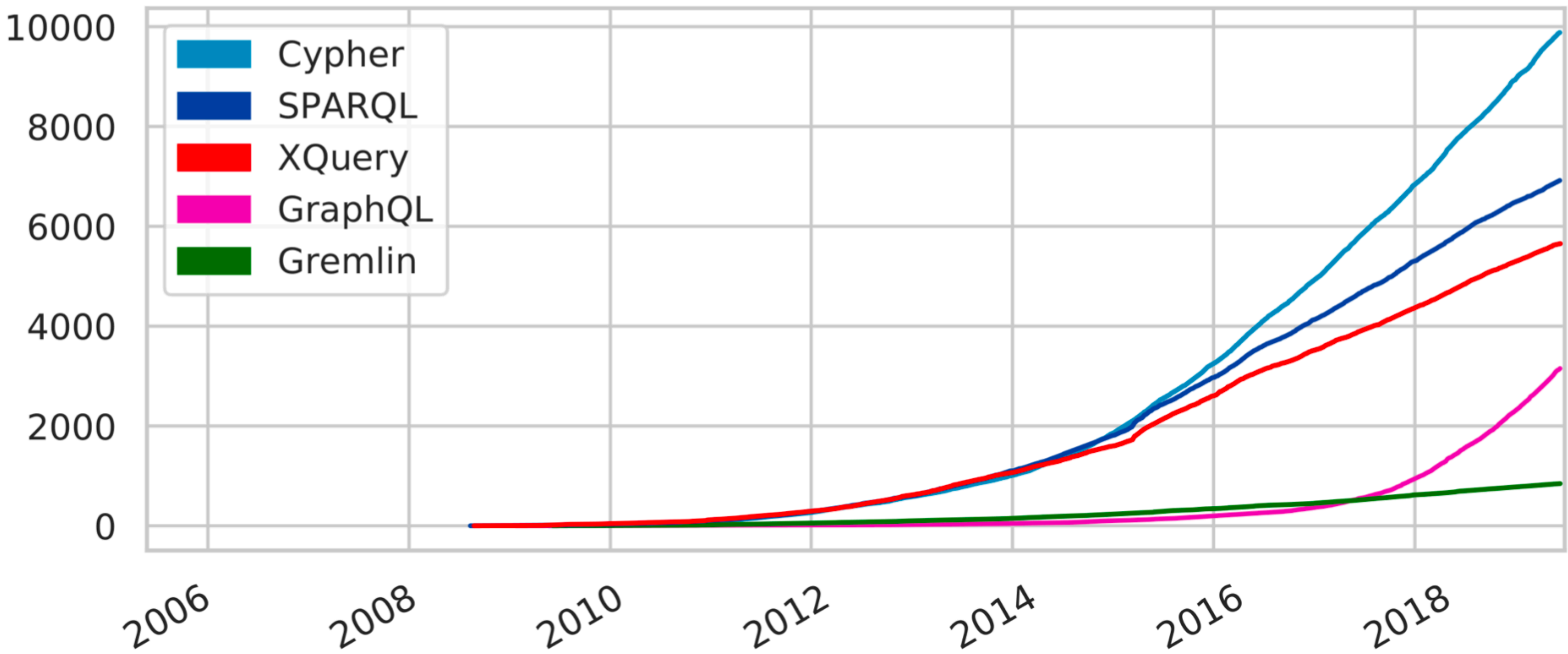
Projects per query language



Software language usage

A Showcase of Software Knowledge Analytics

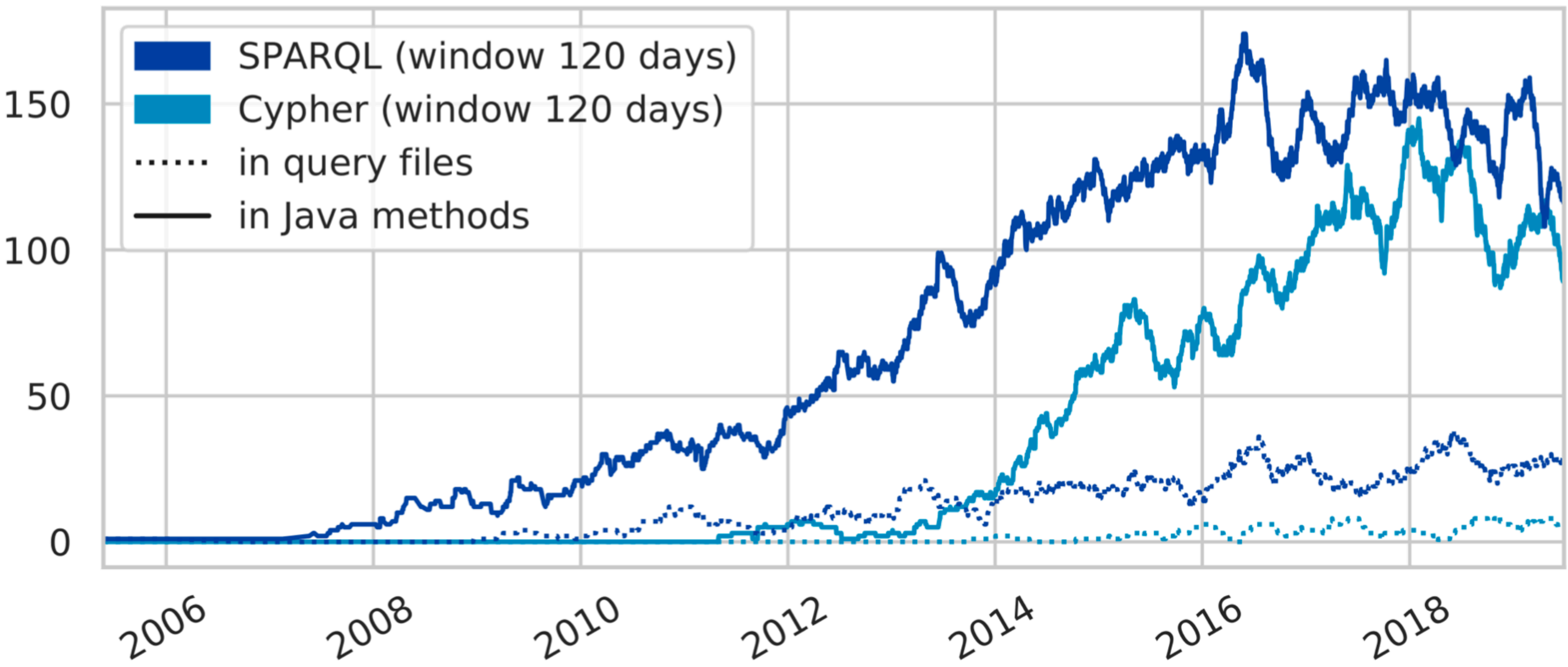
Project count over time



Software language usage

A Showcase of Software Knowledge Analytics

Query coding activities over all repositories



Software language usage

A Showcase of Software Knowledge Analytics

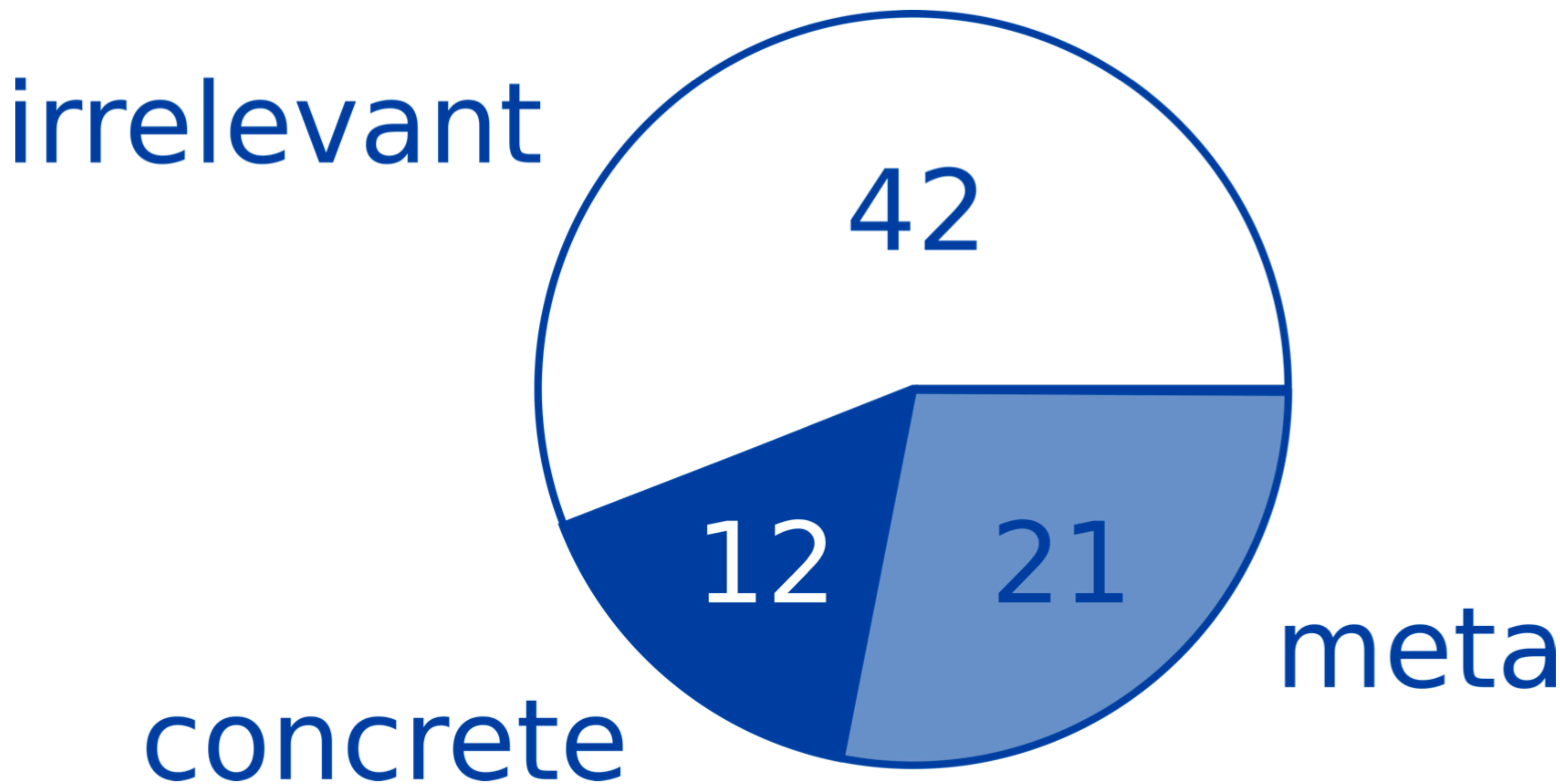
Manual labeling of query coding activities

Label	Description	Example
concrete	Applications using concrete instance data	Museum Exhibit Managment System
meta	Applications using graph structure queries	Database Exploration Tool
irrelevant	Libraries, Frameworks and other uses	SPARQL to SQL compiler

Software language usage

A Showcase of Software Knowledge Analytics

75 labeled repositories



Software language usage

A Showcase of Software Knowledge Analytics

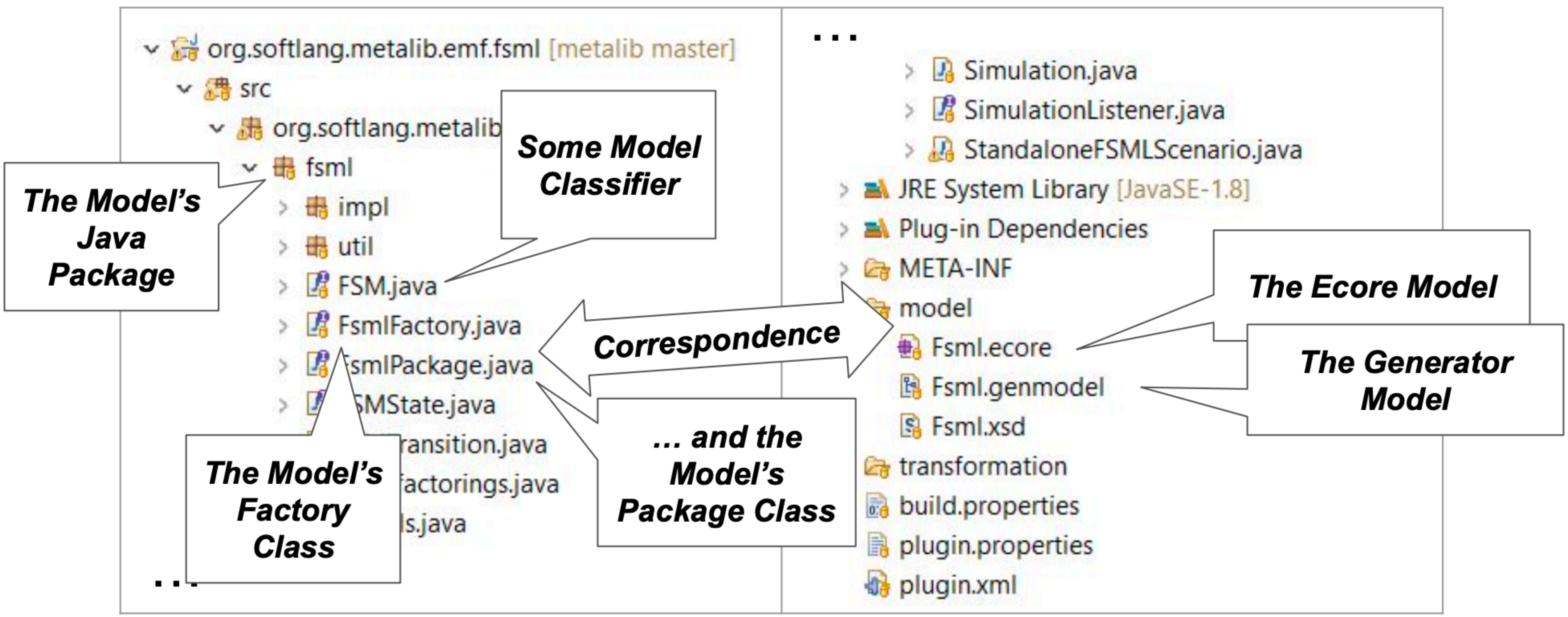
See also:

Philipp Seifer, Johannes Härtel, Martin Leinberger, Ralf Lämmel, Steffen Staab: **Empirical study on the usage of graph query languages in open source Java projects**. SLE 2019: 152-166

Software technology usage

A Showcase of Software Knowledge Analytics

Motivation: What is an EMF pattern of usage?

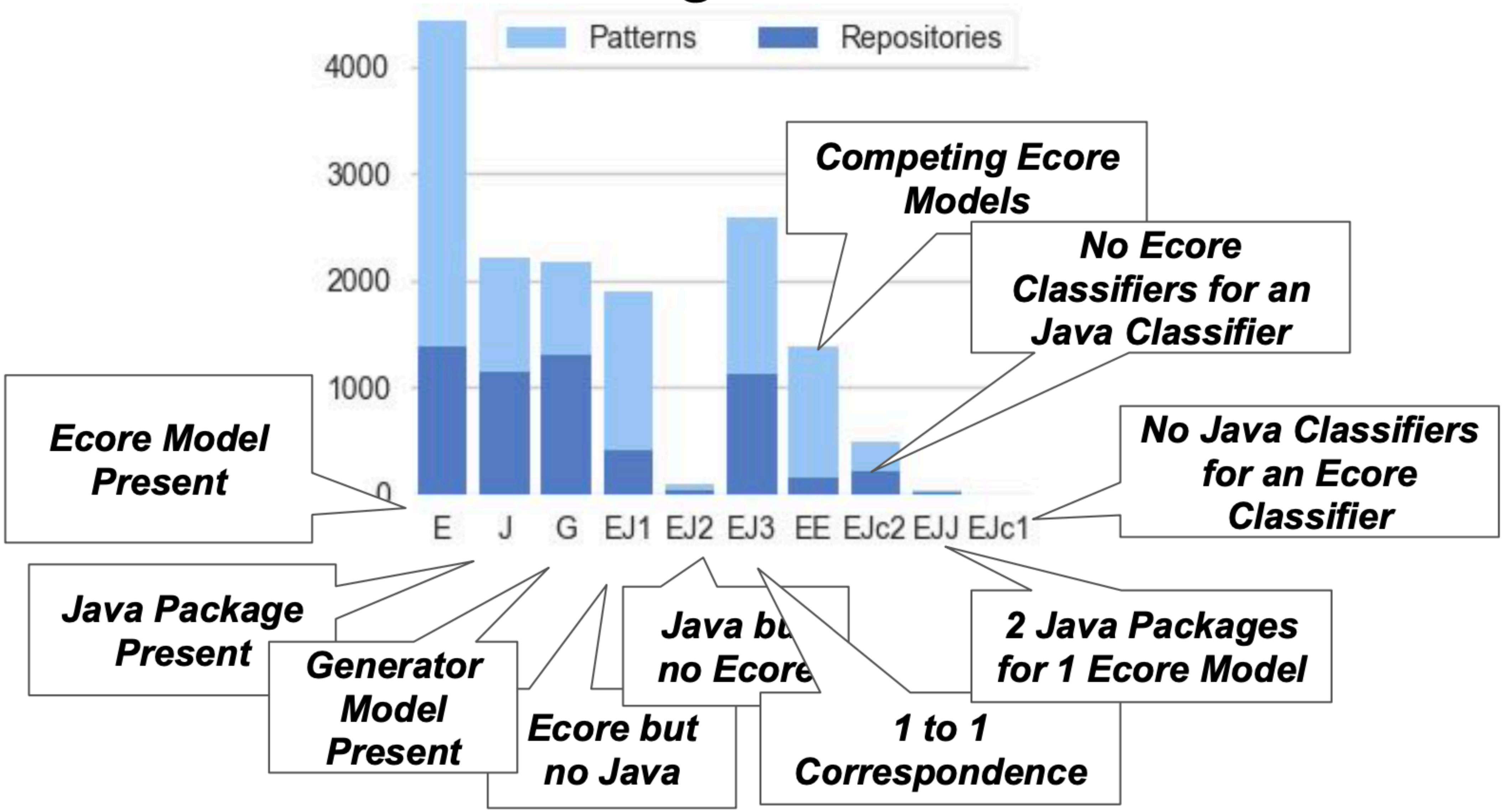


Software technology usage

A Showcase of Software Knowledge Analytics

Patterns found in a recovery project

Which Patterns of Usage can be found?

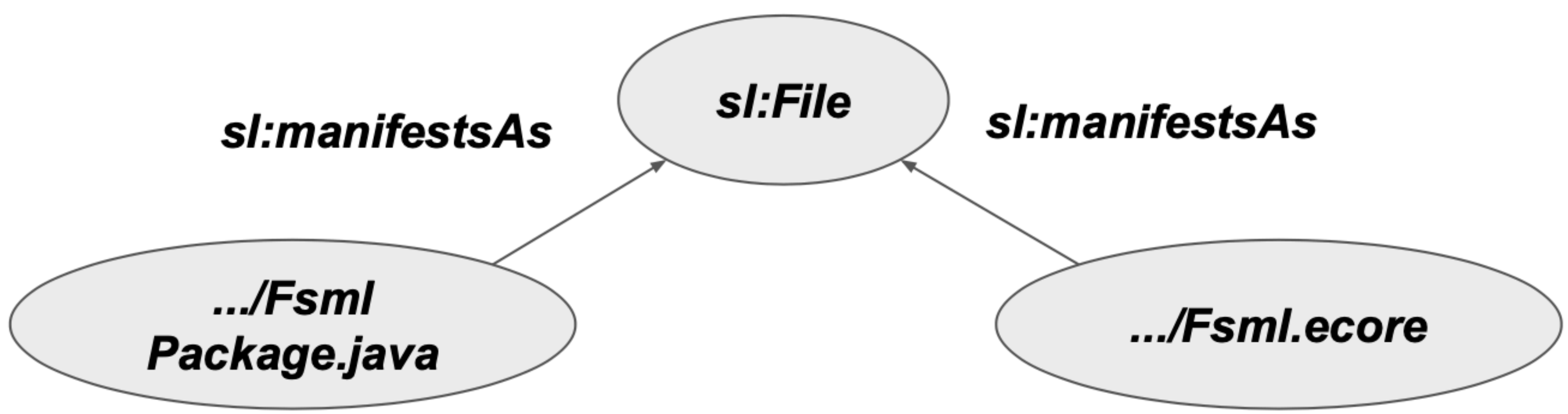


Software technology usage

A Showcase of Software Knowledge Analytics

Logic-based recovery of patterns

Initial tripels



Software technology usage

A Showcase of Software Knowledge Analytics

Logic-based recovery of patterns

Rule application

Classifying ?x – a file with extension 'java' – as element of language Java.

```
(?x, sl:manifestsAs, sl:File) Extension(?x, "java") -> (?x, sl:elementOf, sl:Java).
```

Software technology usage

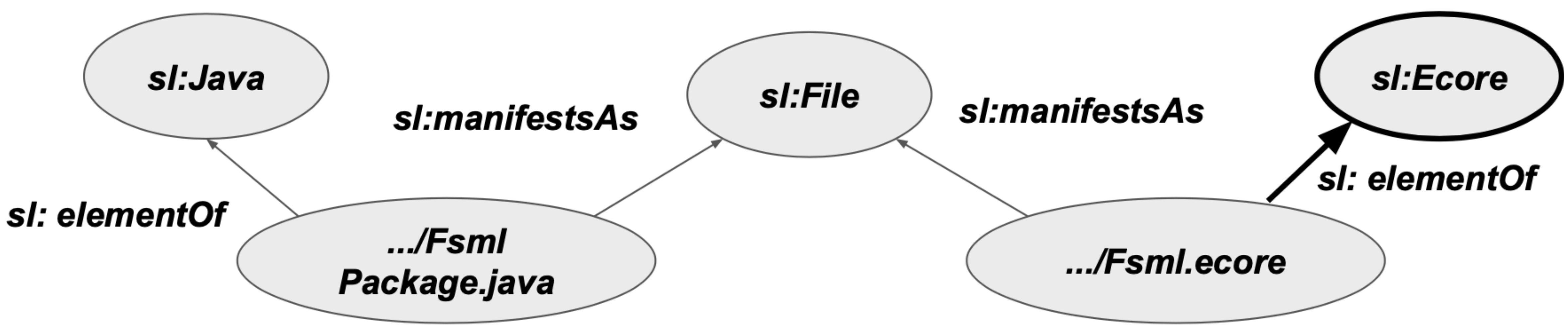
A Showcase of Software Knowledge Analytics

Logic-based recovery of patterns

Rule application cont'd

Classifying ?x – a file with extension 'ecore' – as element of language Ecore.

```
(?x, sl:manifestsAs, sl:File) Extension(?x,"ecore") -> (?x, sl:elementOf, sl:Ecore).
```



Software technology usage

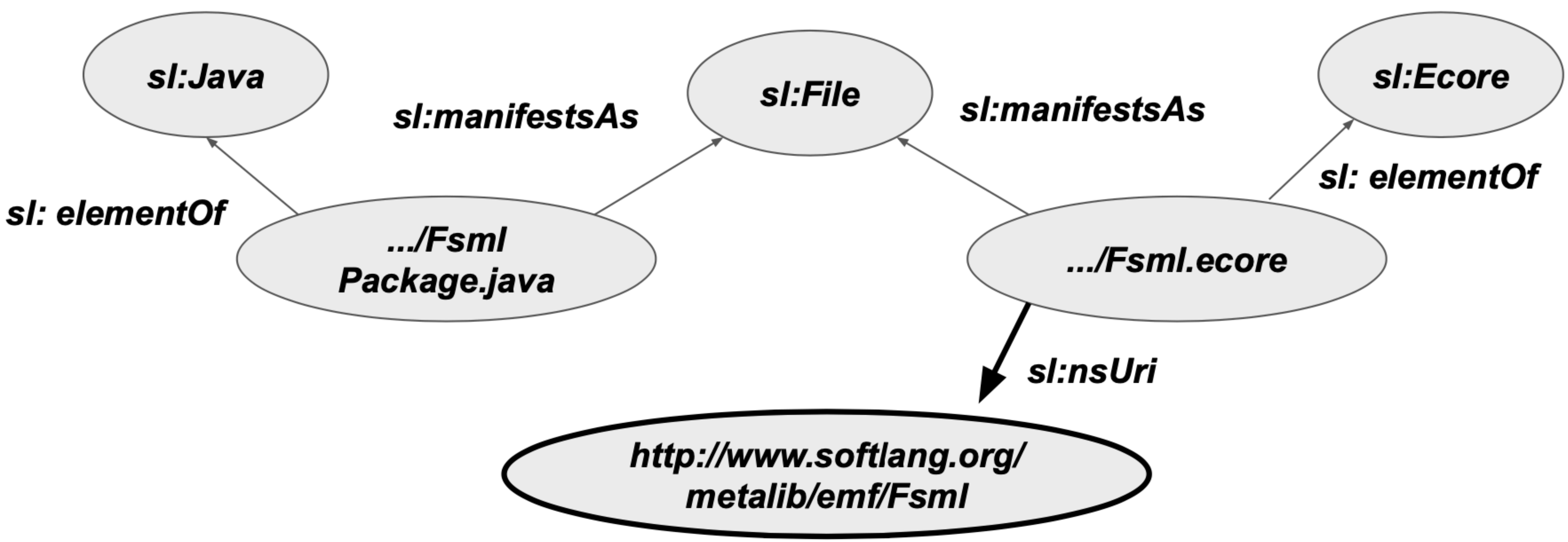
A Showcase of Software Knowledge Analytics

Logic-based recovery of patterns

Rule application cont'd

Extracting a nsUri in an ecore file by an XPath on the XML AST.

```
(?x, sl:elementOf, sl:Ecore) UriXml(?x, "/ecore:EPackage/@nsURI",?nsUri) -> (?x, sl:nsUri, ?nsUri).
```



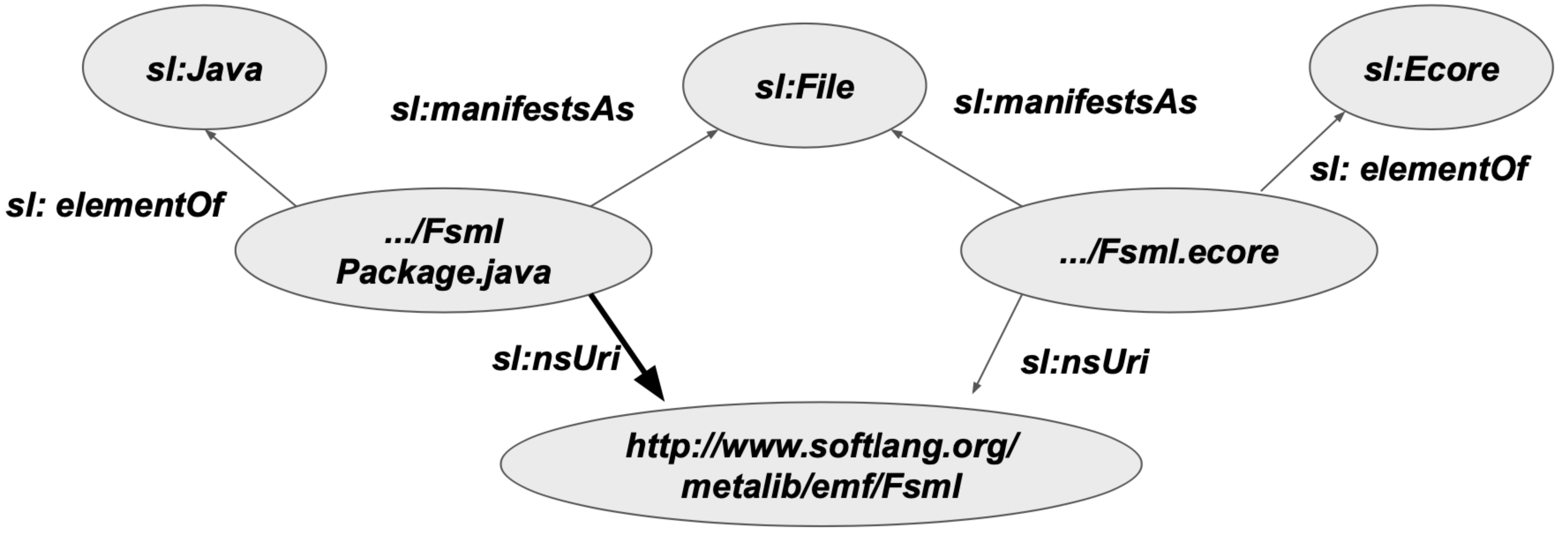
Software technology usage

A Showcase of Software Knowledge Analytics

Logic-based recovery of patterns
Rule application cont'd

Extracting a nsUri in a Java file with suffix 'Packag.java' by an XPath on Java AST.

```
(?x, sl:elementOf, sl:Java) Match(?x,".*Package.java")
UriJava(?x, "type[1]/members/variables[name/identifier='eNS_URI']/initializer/value/value", ?nsUri) ->
    (?x, sl:nsUri, ?nsUri).
```



Software technology usage

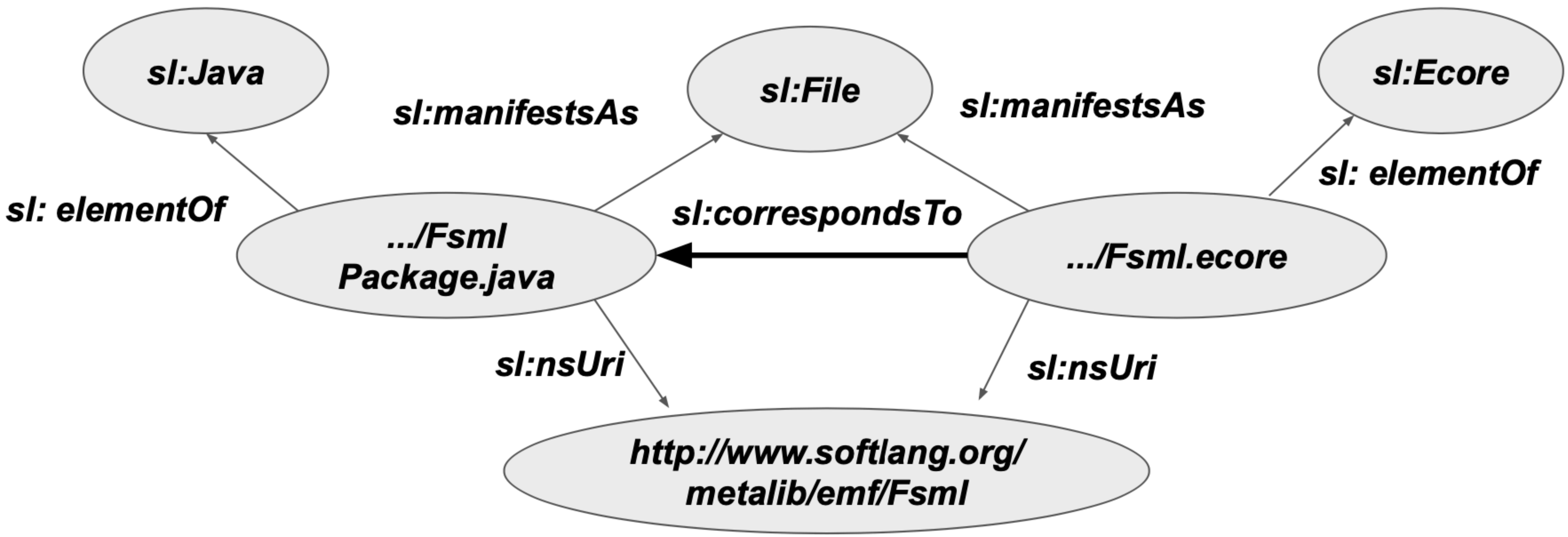
A Showcase of Software Knowledge Analytics

Logic-based recovery of patterns

Rule application completed

Inferring Correspondence between java and ecore with the same nsUri.

```
(?ecore, sl:elementOf, sl:Ecore) (?java, sl:elementOf, sl:Java)
(?ecore, sl:nsUri, ?nsUri) (?java, sl:nsUri, ?nsUri) ->
    (?ecore, sl:correspondsTo, ?java).
```



Software technology usage

A Showcase of Software Knowledge Analytics

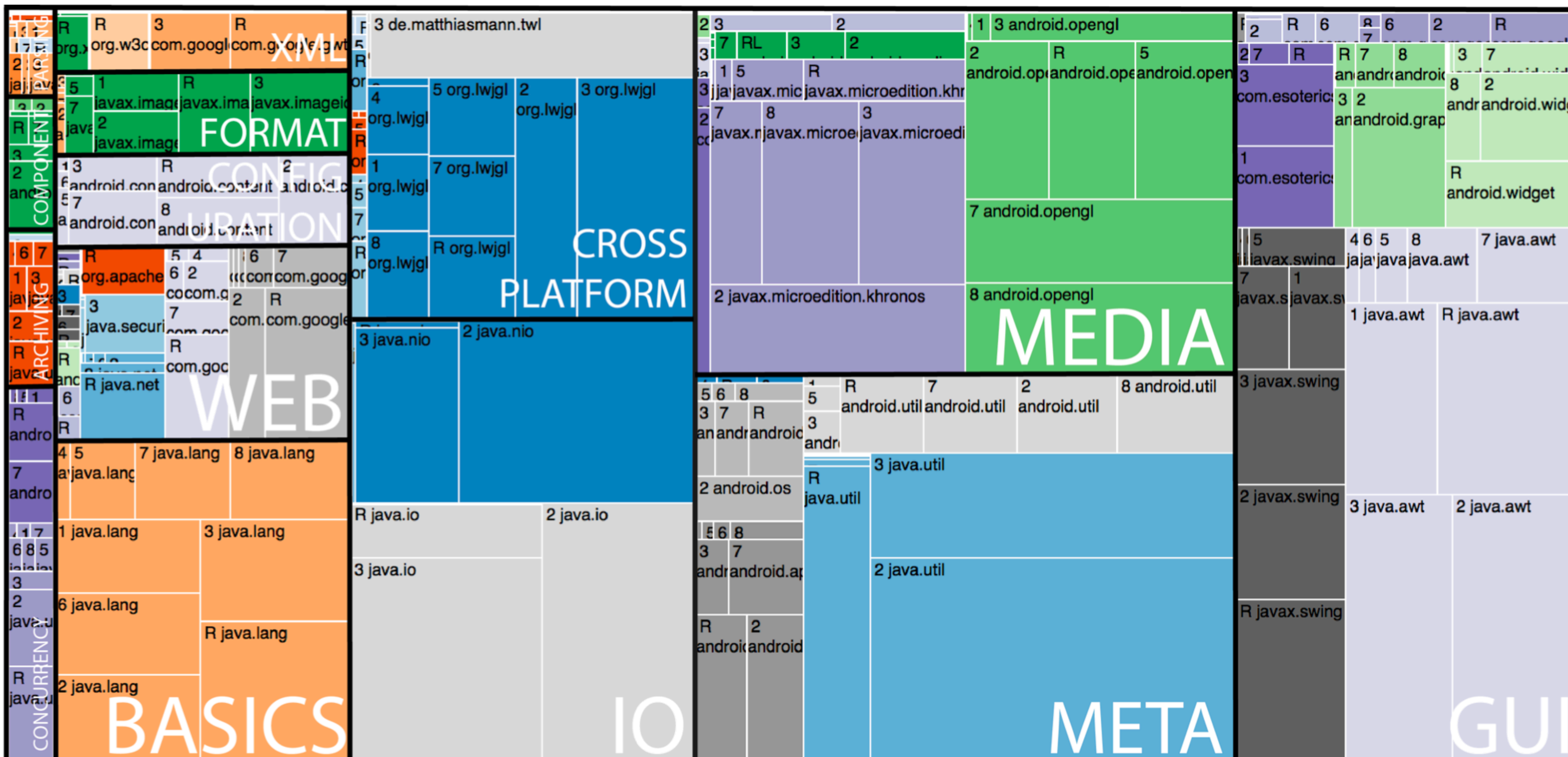
See also:

Marcel Heinz, Johannes Härtel, Ralf Lämmel: **Reproducible Construction of Interconnected Technology Models for EMF Code Generation**. J. Object Technol. 19(2): 8:1-25 (2020)

Software developer profiling

A Showcase of Software Knowledge Analytics

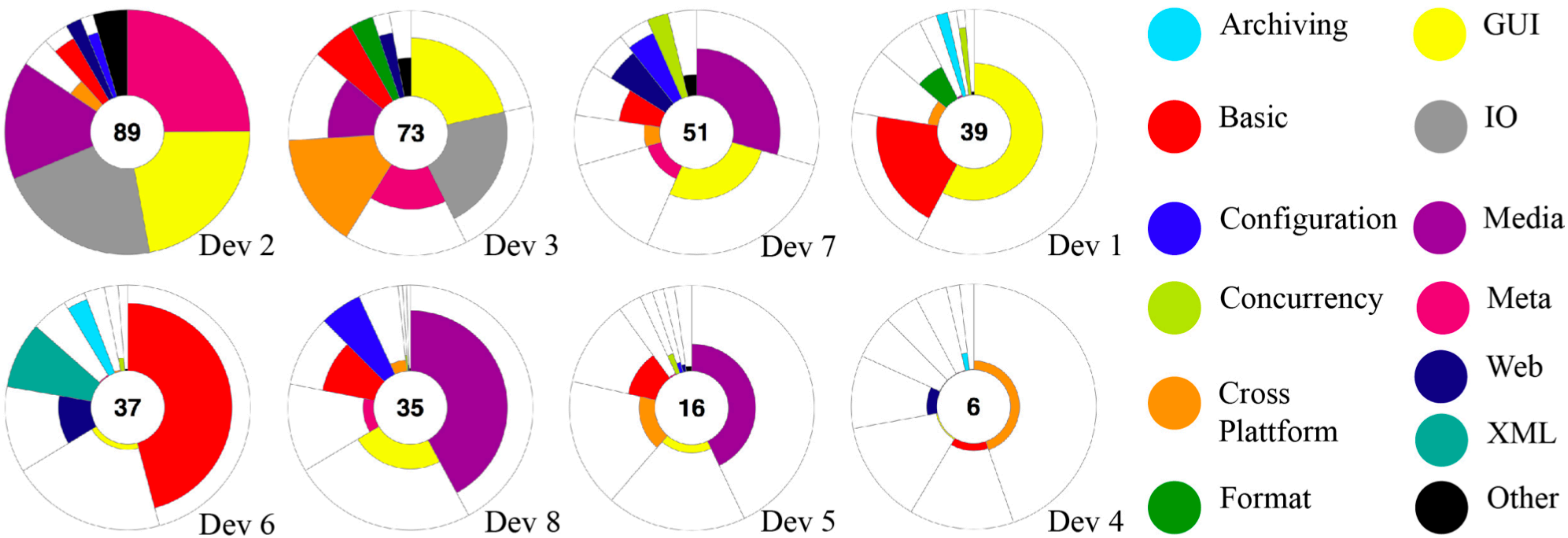
Software projects involve many technical domains (APIs).



Software developer profiling

A Showcase of Software Knowledge Analytics

Software developer profiles also feature technical domains (APIs).



Software developer profiling

A Showcase of Software Knowledge Analytics

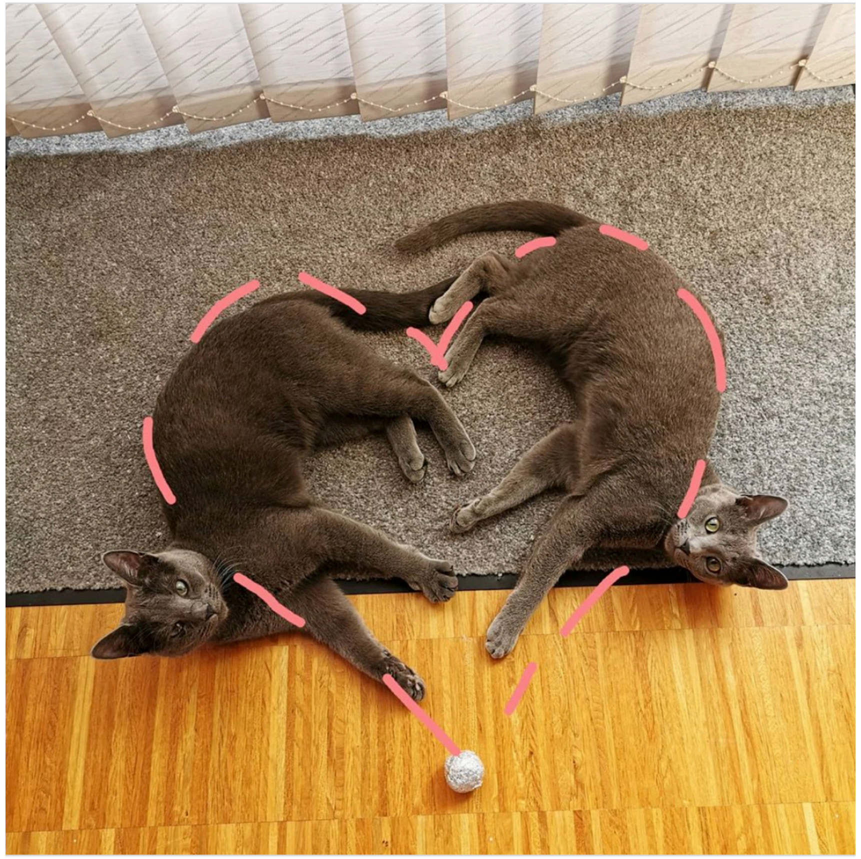
Related research questions:

- How to abstract usefully such API profiles?
- How dissimilar are API profiles across developers?
- How stable are API profiles over time?
- Can we use those profiles, for example, for bug assignment?
- ...

Work-item prediction

A Showcase of Software Knowledge Analytics

Scenarios of work-item prediction I/II



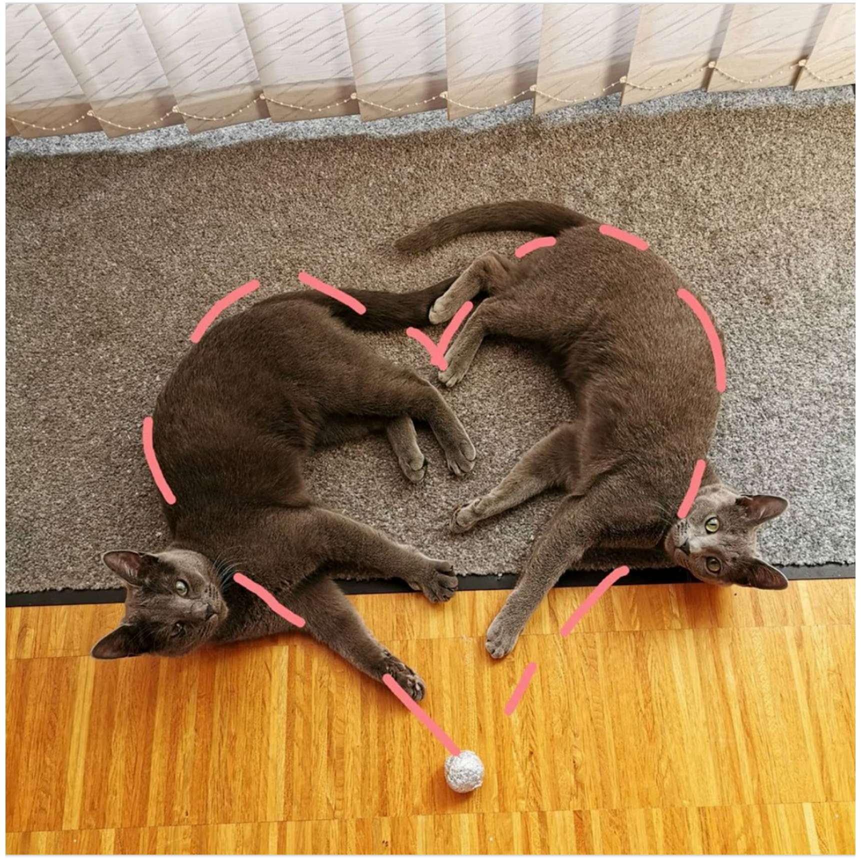
The ‘**Incident Response**’ Scenario:

- *Work item*: **Alert** for suboptimal performance
- *Question*: The workflow steps to follow in response
- *Automation*: Record steps in past instances
- *Challenge*: To know when someone is responding

Work-item prediction

A Showcase of Software Knowledge Analytics

Scenarios of work-item prediction II/II



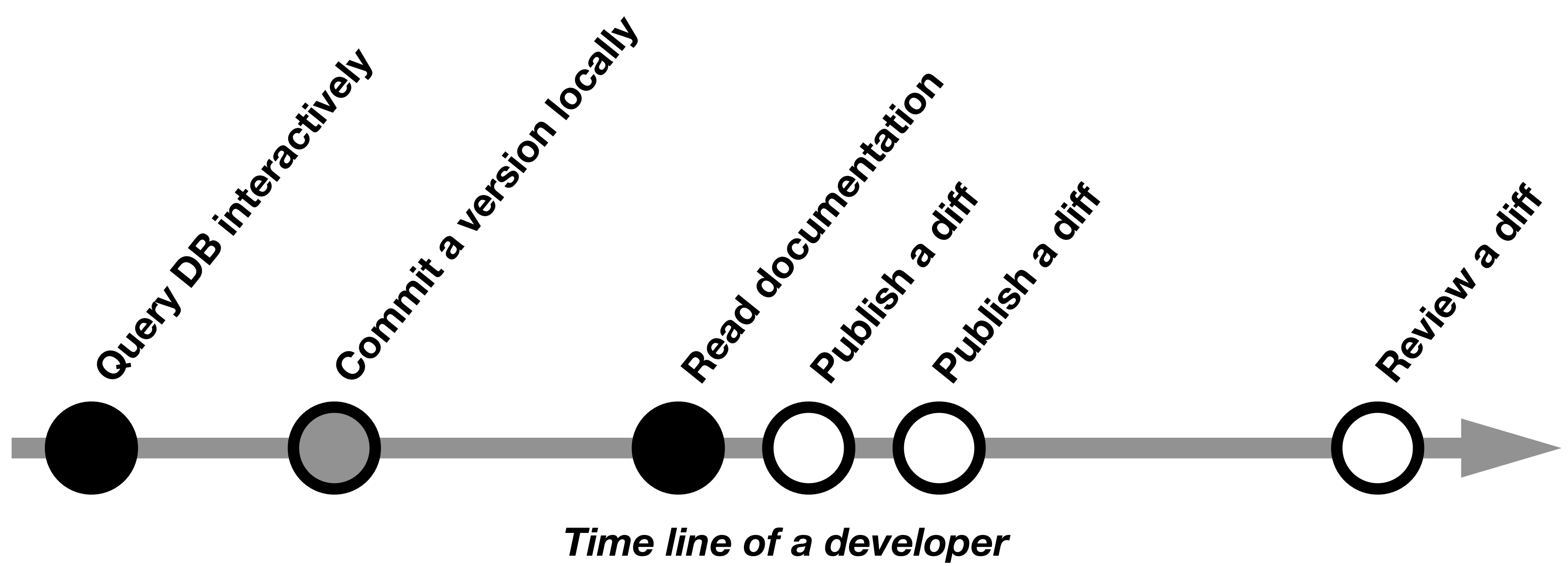
The ‘**Aggregate Performance**’ Scenario:

- *Work item*: A **diff** (a system change)
- *Question*: Time spent on diff
- *Automation*: Record all activities on diff
- *Challenge*: To know when someone is working on the diff

Work-item prediction

A Showcase of Software Knowledge Analytics

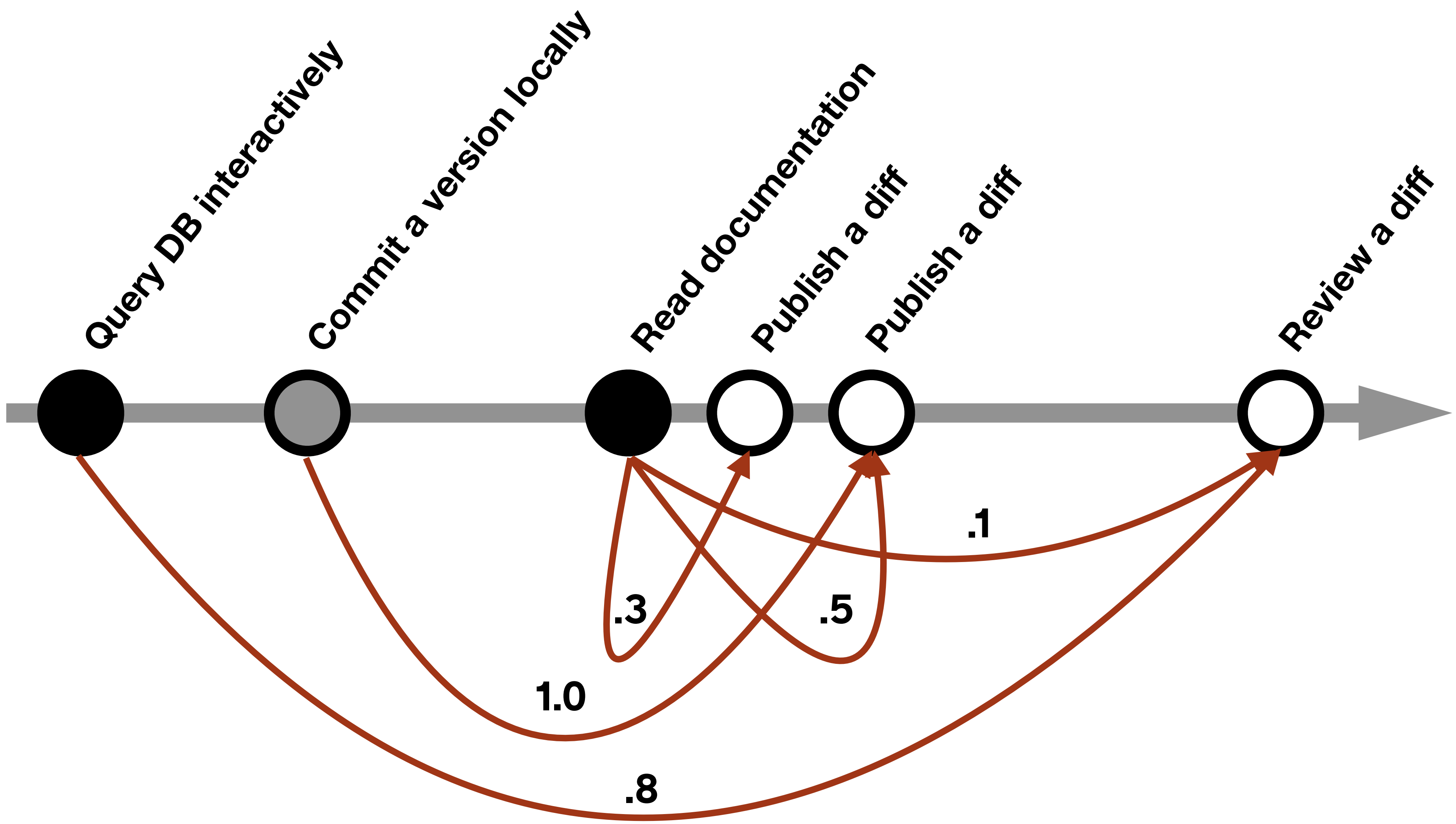
Dark matter in developer workflow analysis



Work-item prediction

A Showcase of Software Knowledge Analytics

Probabilistic work-item prediction



Work-item prediction

A Showcase of Software Knowledge Analytics

Why do we have dark matter?

- Tools don't track work items consistently.
- Tools aren't fully integrated.
- Logging is not designed with workflow analysis in mind.
- Developer workflow is somewhat unstructured.
- Developers engage in a lot of context switching.
- ...

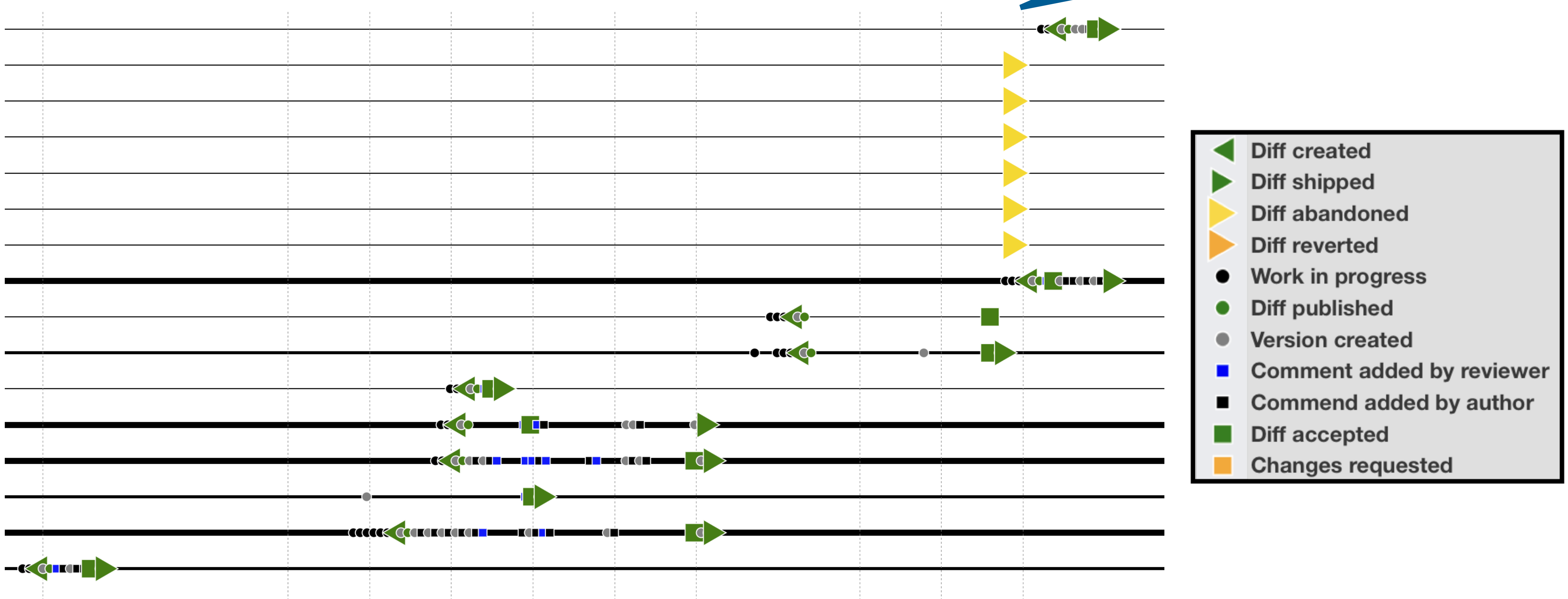
Also known elsewhere as:
Sukriti Goel, Jyoti M. Bhat, and Barbara Weber. 2013.
End-to-End Process Extraction in **Process Unaware Systems**.
In Business Process Management Workshops –
BPM 2012 International Workshops. Revised Papers (Lecture Notes in Business Information Processing), Vol132. Springer, 162–173.

Work-item prediction

A Showcase of Software Knowledge Analytics

Context switching in development

We need more than time proximity and high-confidence events.



Work-item prediction

A Showcase of Software Knowledge Analytics

See also:

Ralf Lämmel, Alvin Kerber, Liane Praza: **Understanding What Software Engineers Are Working on: The Work-Item Prediction Challenge**. ICPC 2020: 416-424

Ownership management

A Showcase of Software Knowledge Analytics

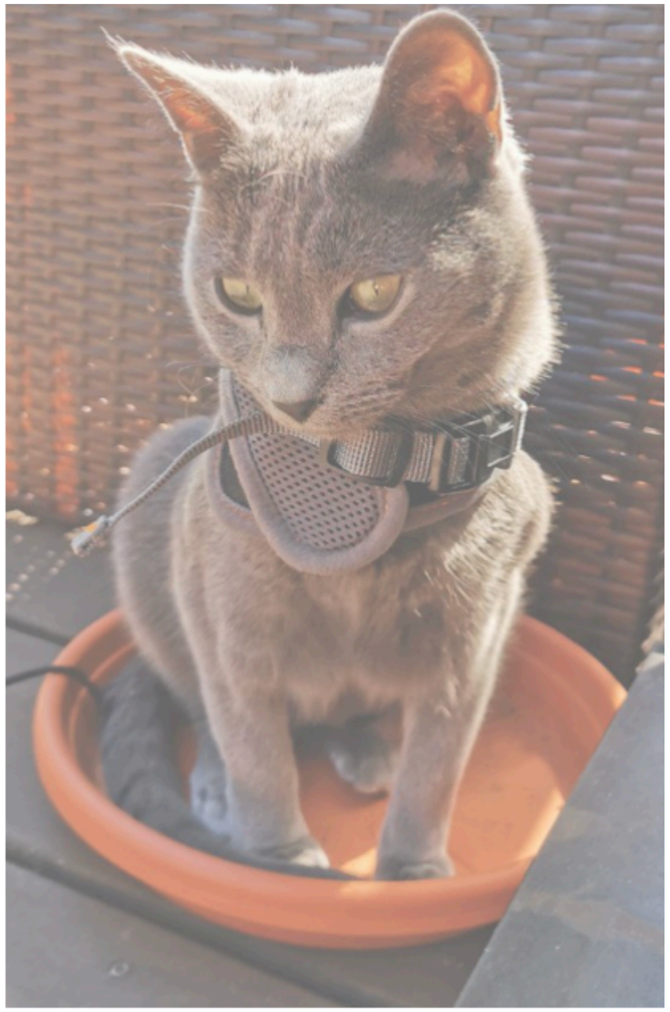
What's ownership management?

"Each asset has the most accountable owner at all times."

Software & data assets:

Hive tables,
Pipelines,
ML models,
Files in repos,
...

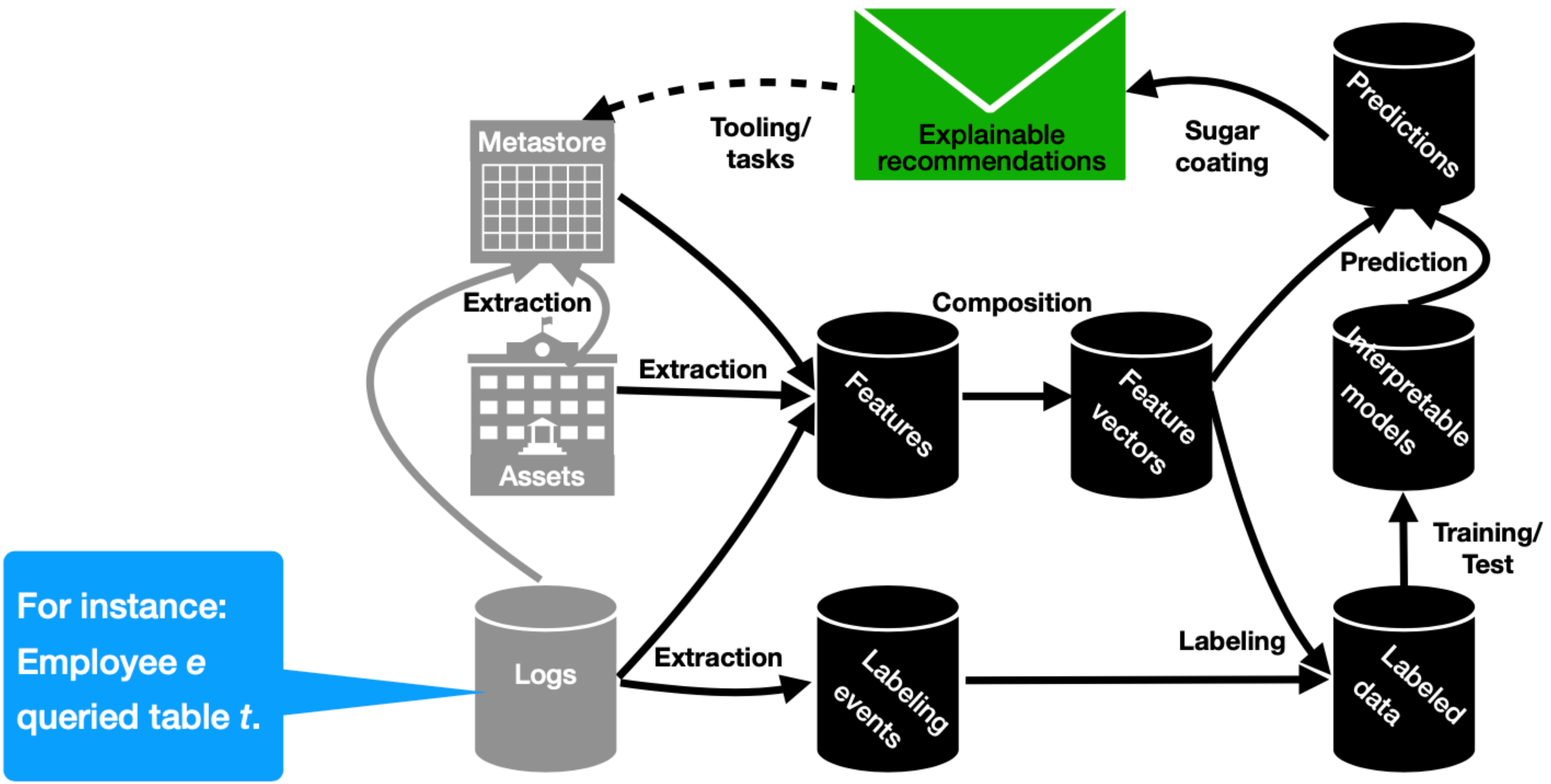
POC for all means regarding
reliability,
security,
privacy,
et al.



Ownership management

A Showcase of Software Knowledge Analytics

Architecture of an ownership recommendation system



Ownership management

A Showcase of Software Knowledge Analytics

Challenges in ownership management

Challenge	Details
Ownership decay	*How to know whether to trust owners on file?*
Asset subclassing	*How to identify and handle specific subsets of assets?*
Team-level ownership	*How to assign teams as owners with individual signal?*
Ranking owner candidates	*What ranking to use to recommend one ore more candidates?*
Whole/part asset relationships	*How to obey those relationships with recommendations?*
Monotonic features	*How to make sure that "more" means "more likely owner"?*
Explainable recommendations	*How to explain recommendations to use so that they accept?*

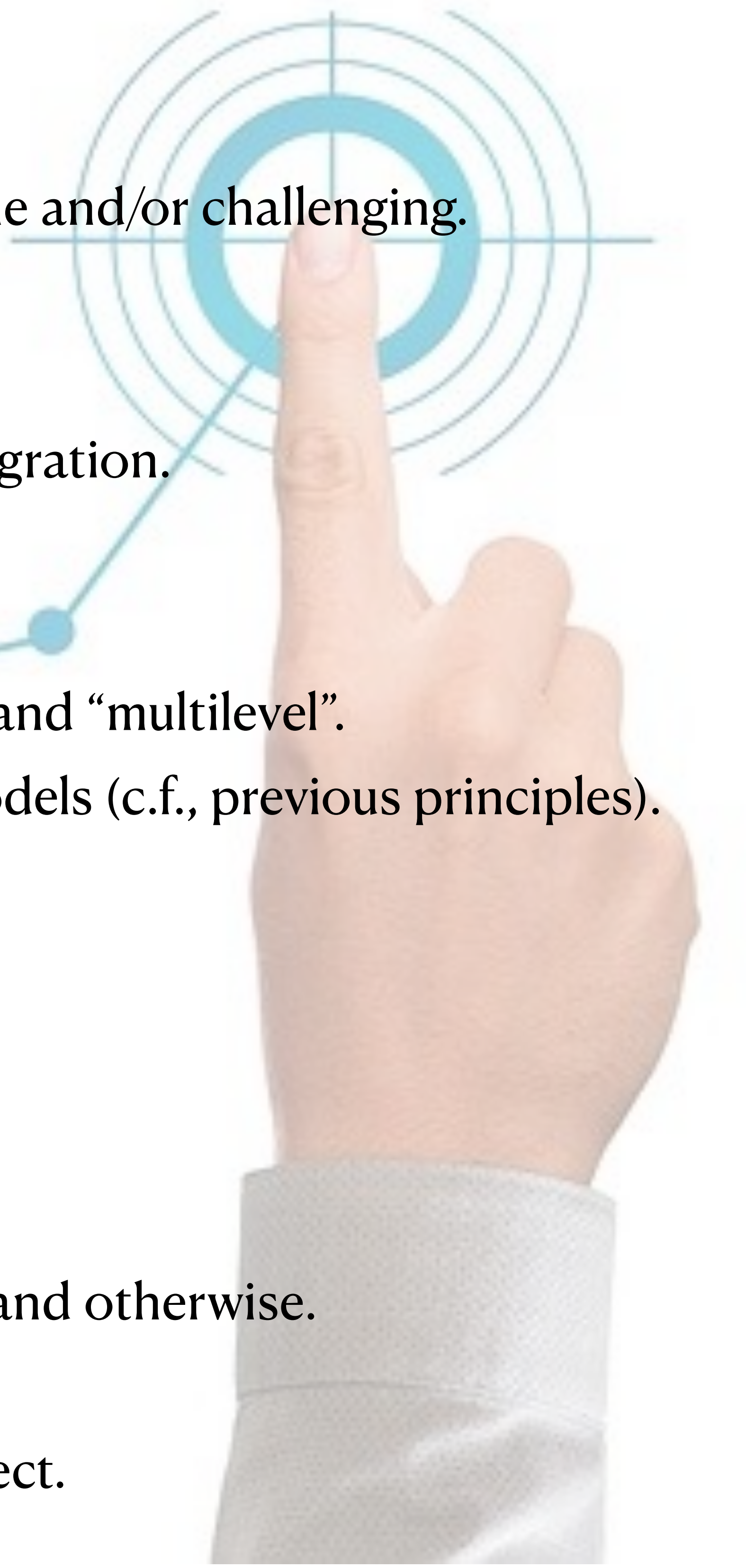
Ownership management

A Showcase of Software Knowledge Analytics

See also:

John Ahlgren, Maria Eugenia Berezin, Kinga Bojarczuk, Elena Dulskyte, Inna Dvortsova, Johann George, Natalija Gucevska, Mark Harman, Shan He, Ralf Lämmel, Erik Meijer, Silvia Sapora, Justin Spahr-Summers: **Ownership at Large: Open Problems and Challenges in Ownership Management**. ICPC 2020: 406-410

Principles of *software knowledge analytics*



- **Hypothesis building**
  - Set up falsifiable hypotheses together with the research questions.
  - Lay out the theory to back up those hypotheses/RQs to be reasonable and/or challenging.
- **Data extraction and integration**
  - Follow an empirical approach — more artifact- than subject-based.
  - Justify chosen data sources and methods of data extraction and integration.
- **Mathematical modeling**
  - Aim at the discovery of mathematical models.
  - Address problems such as “type I error”, “overfitting”, “skewed data”, and “multilevel”.
  - Enable (probabilisitic) reasoning regarding any data, hypotheses, models (c.f., previous principles).
- **Logical reasoning**
  - Enrich data extraction and integration.
  - Perform (part of) the analysis by such reasoning.
- **Semantic (meta)data**
  - Add programmatically useful documentation for all entities involved.
  - Leverage such documentation in logical reasoning for explainability and otherwise.
- **Continuous replication**
  - Enable continuous validation in terms of reproducibility for any project.
  - Enable follow-up projects to layer on top of existing ones soundly.

Hypothesis building

A Principle of Software Knowledge Analytics

Examples of hypotheses

- The greater the number of software engineers per square meter in a country, the smaller the ratio of failing to succeeding software projects in the country.
- Haskell programmers perform better in web programming than C programmers.
- …

Hypothesis building

A Principle of Software Knowledge Analytics

What's a hypothesis?

- A relation between two variables?
  - C.f. independent, dependent, observed, non-observed, identified variables.
- An introduction of the research question?
- A proposal regarding the expected result?
- It's what you propose to prove by your research!
- A hypothesis may change over time, as research progresses.
- ...

Data extraction and integration

A Principle of Software Knowledge Analytics

Forms of extraction

- Scanning
- Parsing
- Static analysis
- Dynamic analysis
- NLP
- Scraping
- …

Image by Simon J from Pixabay

Data extraction and integration

A Principle of Software Knowledge Analytics

Facets of integration

- Joins
- Conversion
- ID recovery
- Metadata
- Traceability links
- ...

Mathematical modeling

A Principle of Software Knowledge Analytics

Examples

- Logistic regression models for observed variables
- Confidence intervals for identified variables
- N-grams for language corpuses
- Bayesian models for probability distribution of observations
- Decision trees for feature-based predictions
- Performance models for ML model

Logical reasoning

A Principle of Software Knowledge Analytics

Examples

- Description logic-based reasoning
- Datalog style deductive databases
- Logic-based verification
- Constraint systems

Semantic metadata

A Principle of Software Knowledge Analytics

Examples

- Ontological classifiers
- Ontological relationships
- Traceability links
- Versions / variants / scopes
- Truth values / sources
- …

Semantic metadata

A Principle of Software Knowledge Analytics

A metadata bug on Wikidata

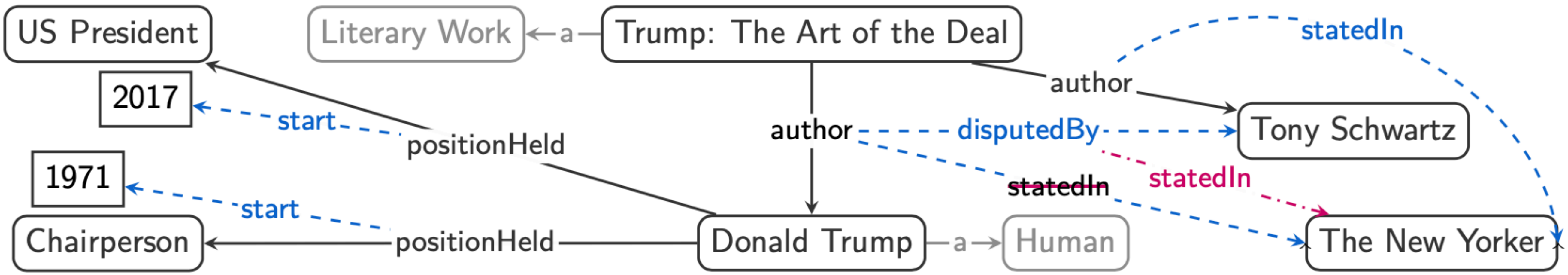


Figure — Knowledge graph excerpt from Wikidata. We use labels instead of IDs and rename or simplify some for brevity. *We use dark solid for the data graph, dashed for metadata, and light solid for the data schema. In dash-dotted we annotate a fix clarifying the scope of —statedIn→.*

Continuous replication

A Principle of Software Knowledge Analytics

Aka “Fight the replication crisis”

- Replication in a narrow sense
  - Validation of an analysis and the interpretation of results
  - Aka reproducibility
- Replication in a broad sense
  - Exact replication (with different data)
  - Generalized replication (with revised methodology)
- Continuous replication
  - Keep analyses alive and enable replication in a broad sense

Challenges of Software Knowledge Analytics

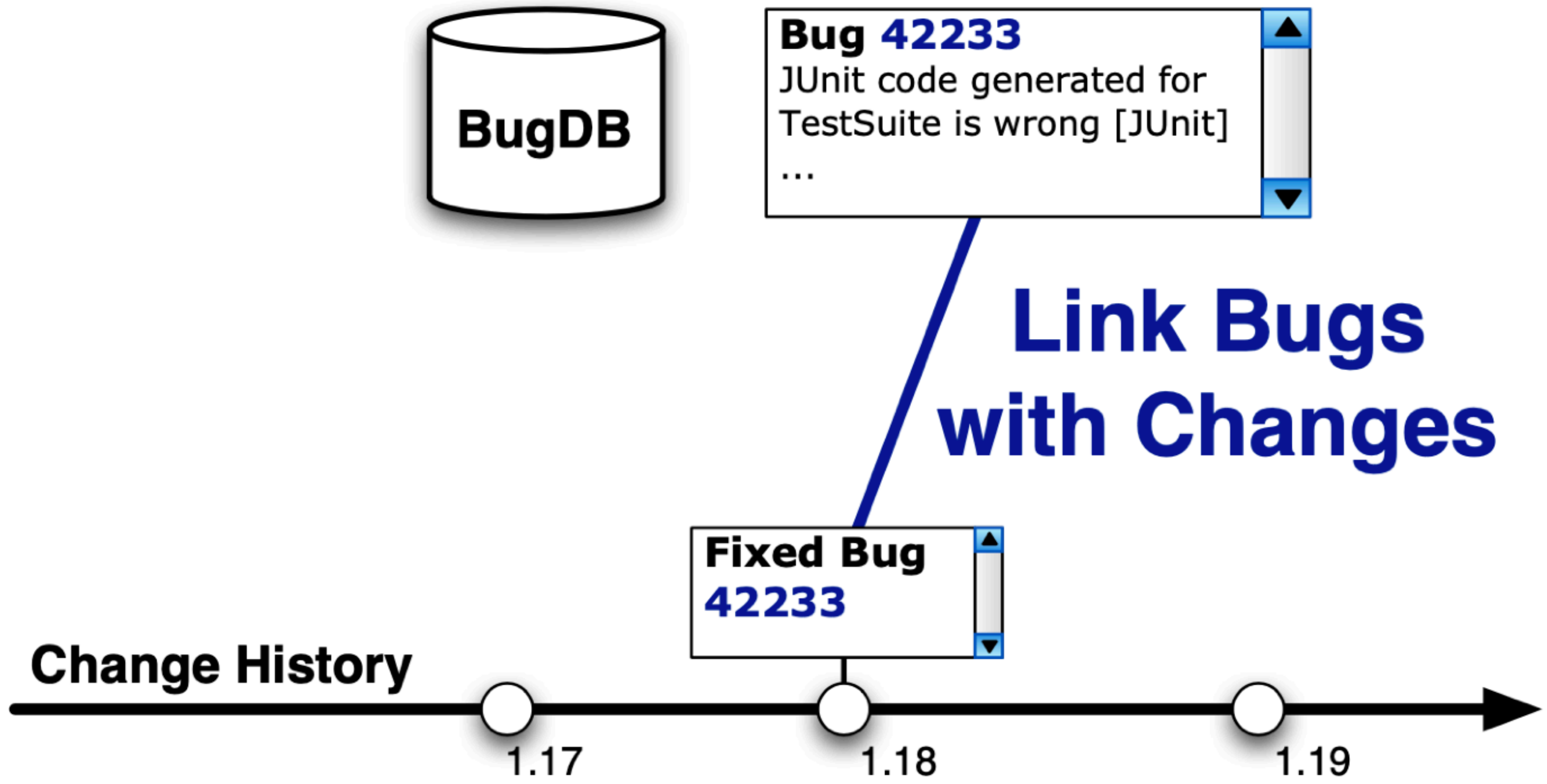
- Handling weak data
- Scaling for evolving data
- Ontology engineering
- Knowledge graph population
- Managing threats to validity

Handling weak data

A Challenge of Software Knowledge Analytics

- Example: Observing variables with **uncertainty**
  - Instance — Linking bug fixes to bug-inducing changes

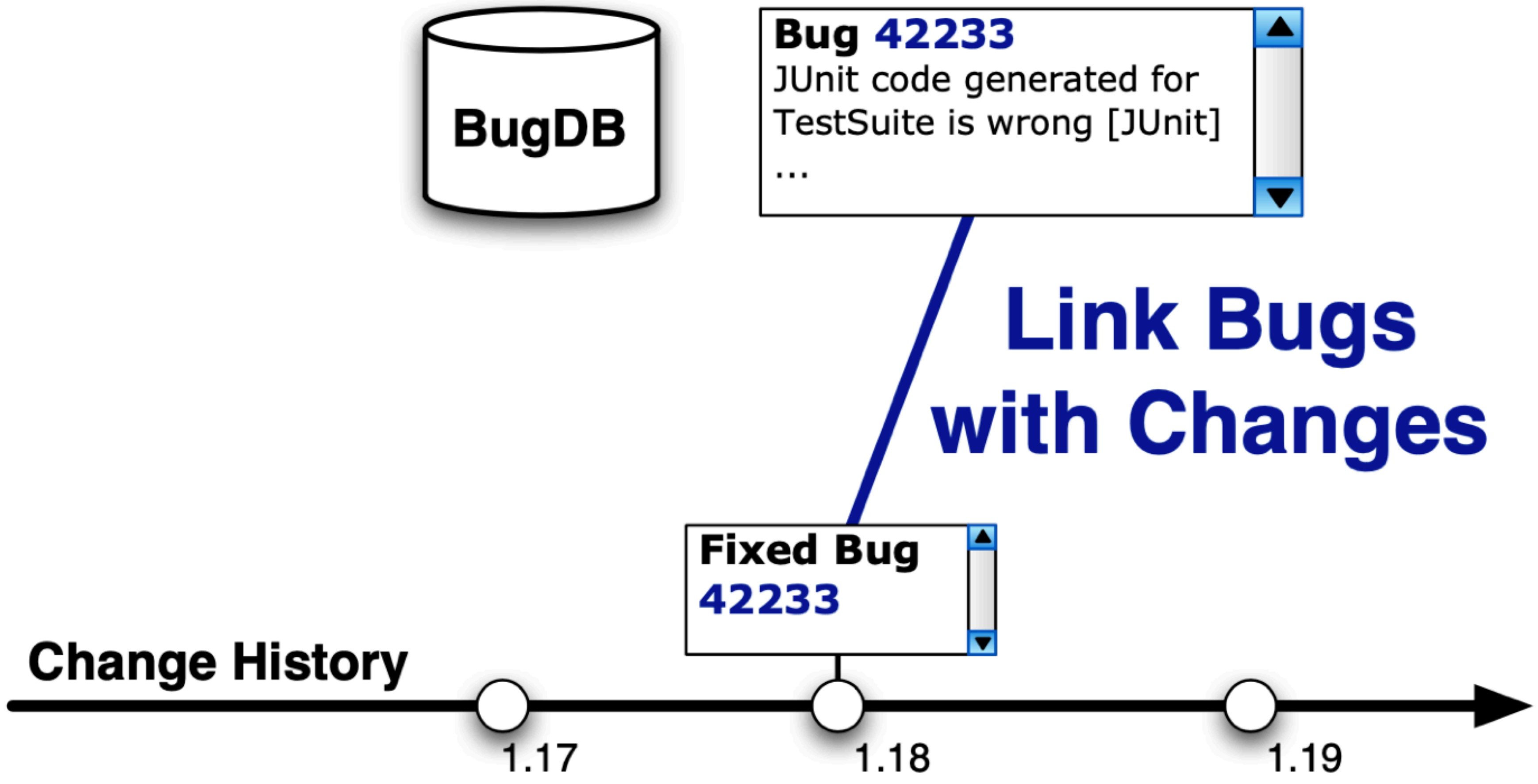
SZZ



1. We start with a bug report in the bug database, indicating a *fixed problem*.
2. We extract the associated change from the version archive, thus giving us the *location* of the fix.
3. We determine the *earlier change* at this location that was applied before the bug was reported.

Source: Jacek Sliwerski, Thomas Zimmermann, Andreas Zeller: When do changes induce fixes? (On Fridays.) MSR 2005

SZZ



- How to do know for sure whether a commit fixes a bug?
- ... what part of the commit fixes the bug?
- ... when that part was changed in the past?
- ... what other changes are spurious?

SZZ

Handling weak data with loads of heuristics

The Mapping and Selection Mechanisms of the Studied SZZ Implementations

	Description	Mapping Mechanism	Selection Mechanism
B-SZZ	First SZZ implementation proposed by Śliwerski et al. [9].	The *annotate* function is used to prepend the last change that modified each line of code within a file in a given change. Next, each line of code is scanned in order to identify the last change that modified the lines that were involved in the bug-fixing change. Such changes are potential bug-introducing changes.	†‡Potential bug-introducing changes that are dated after the bug report date are removed.
AG-SZZ	B-SZZ improvement proposed by Kim et al. [11].	The annotation-graph is used to represent evolution of each line of code within source files. A depth-first search of the annotation-graph is used to find the potential bug-introducing changes.	†‡Changes such as comments, format changes, blank lines, and code movement are not flagged as potential bug-introducing changes.
MA-SZZ	It is built on top of the AG-SZZ but it is aware of meta-changes. This implementation is proposed in this paper.		†‡Potential bug-introducing changes that are meta-changes are removed.
R-SZZ	B-SZZ improvement proposed by Davies et al. [27]. We build R-SZZ on top of MA-SZZ in this paper.		The latest potential bug-introducing change is indicated as bug-introducing.
L-SZZ	B-SZZ improvement proposed by Davies et al. [27]. We build L-SZZ on top of MA-SZZ in this paper.		The largest potential bug-introducing change is indicated as bug-introducing.

In addition to the selection mechanisms described directly in each row, the selection mechanisms of the prior rows that have the † symbol are also inherited. For example, L-SZZ inherits all of the previous selection mechanisms except the one from R-SZZ. Finally, the ‡ symbol indicates that all of the potential bug-introducing changes are returned by that SZZ implementation.

Source: Daniel Alencar da Costa et al.:
A Framework for Evaluating the Results of the SZZ Approach for Identifying Bug-Introducing Changes.
IEEE Trans. Software Eng. 43(7): 641-657 (2017)

Handling weak data

A Challenge of Software Knowledge Analytics

- Another type of weak data
  - “**Weak supervision** is a branch of machine learning where noisy, limited, or imprecise sources are used to provide supervision signal for labeling large amounts of training data in a supervised learning setting.”

Source: https://en.wikipedia.org/wiki/Weak_supervision, 2022-05-16

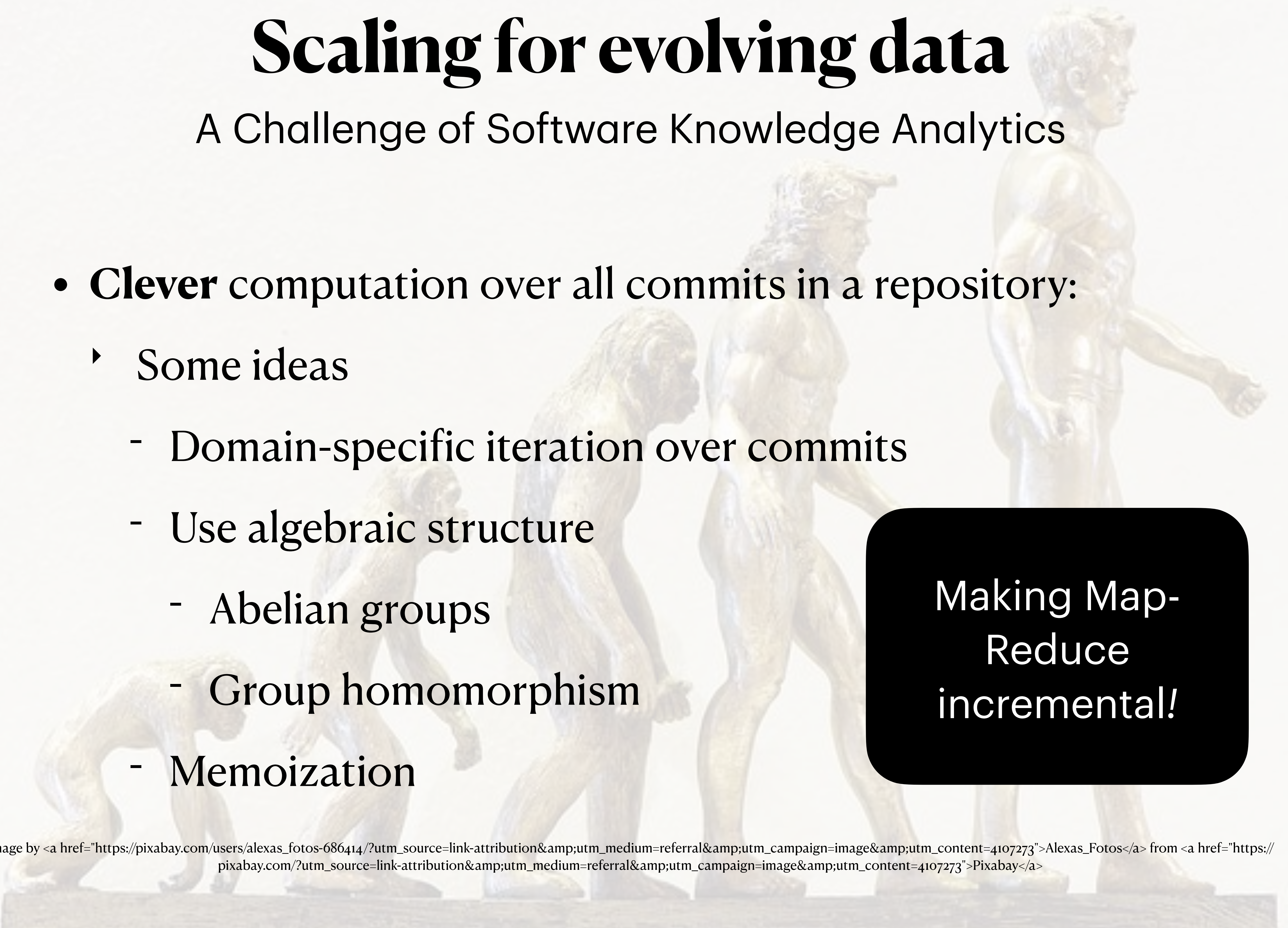
Scaling for evolving data

A Challenge of Software Knowledge Analytics

- Example: Computation over all commits in a repository:
  - Instance — **Naive** MCC evolution of files
    - Iterate over all commits
    - **Materialize all files**
    - **Compute MCC on all files**
    - **Compose per-file maps for all commits**

Scaling for evolving data

A Challenge of Software Knowledge Analytics

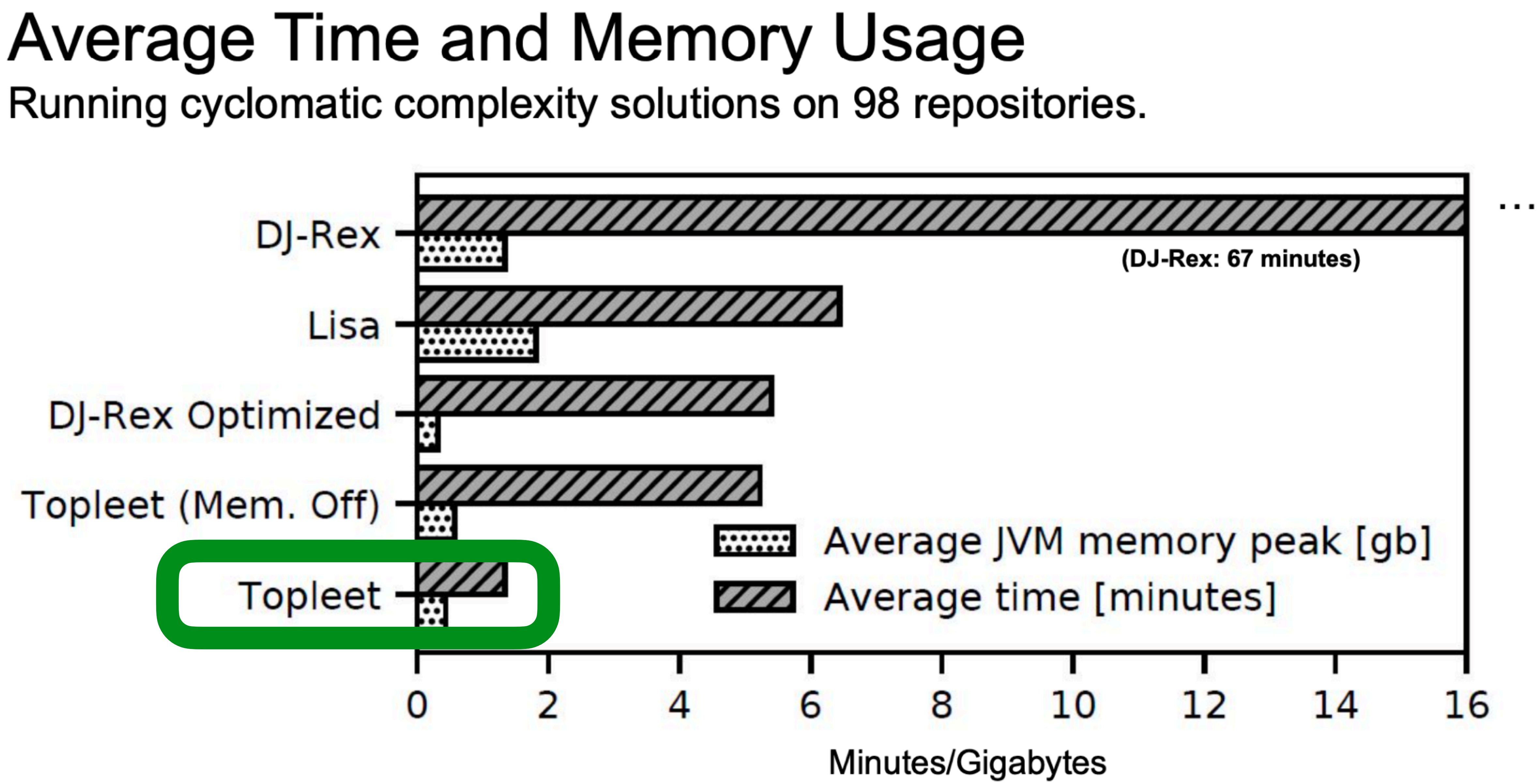


- **Clever** computation over all commits in a repository:
  - Some ideas
    - Domain-specific iteration over commits
    - Use algebraic structure
      - Abelian groups
      - Group homomorphism
    - Memoization

Making Map-Reduce incremental!

Image by Alexas_Fotos from Pixabay

Naive versus clever



Source: Johannes Härtel, Ralf Lämmel:
Incremental Map-Reduce on Repository History.
SANER 2020: 320-331

Ontology engineering

A Challenge of Software Knowledge Analytics

- Example: Classify and associate entities in software domain:
  - Instance — **Software languages and their usage.**
    - What are the relevant entity types?
    - … relationship types?
    - What's the meaning of the relationships?
    - How to identify the entities ("instances")?

Table 1: Entity types in relevant papers.

Paper	Artifact	Function	Record	System	Technology	Language	Inf. resource	Fragment	Collection	Trace	Concept	Others
[1]	x	x	x				x					x
[2]	x	x	x		x					x		x
[3]	x			x	x						x	x
[4]					x	x	x				x	x
[5]	x						x	x		x		x
[6]	x		x									
[7]	x	x	x									
[8]	x									x		
[9]	x						x		x			
[10]	x	x	x		x	x		x	x			
[11]	x	x	x							x		
[12]				x								x
[13]	x	x								x		x
[14]	x	x										

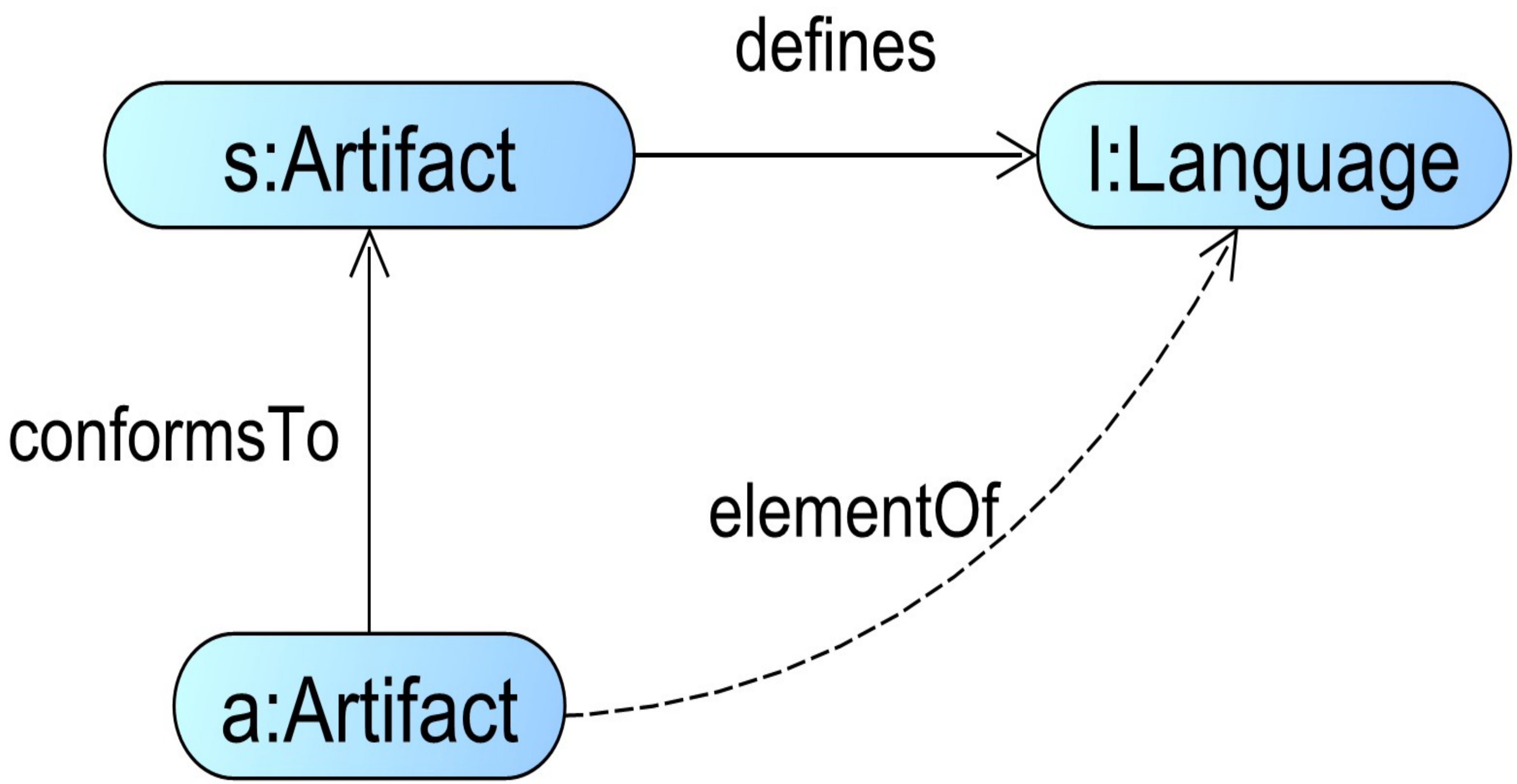
Source: Marcel Heinz, Ralf Lämmel, Andrei Varanovich: **Axioms of Linguistic Architecture**. MODELSWARD 2017: 478-486

Table 2: Relationship types in relevant papers.

Paper	Conformance	Definition	Correspondence	Implementation	Usage	Membership	Typing	Dependency	Abstract rel.	Others
[1]					x					x
[2]	x									
[3]	x	x	x	x	x			x		x
[4]				x	x		x	x		x
[5]				x			x			x
[6]						x	x		x	
[7]										x
[8]									x	
[9]		x							x	
[10]	x	x	x	x		x	x	x	x	
[11]	x	x					x		x	
[12]		x								x
[13]							x			
[14]	x								x	

Source: Marcel Heinz, Ralf Lämmel, Andrei Varanovich: **Axioms of Linguistic Architecture**. MODELSWARD 2017: 478-486

Understanding Membership



- $\text{elementOf}(a, l) \Rightarrow \text{Artifact}(a) \wedge \text{Language}(l) \ldots$
- $\text{elementOf}(a, l) \Leftarrow \exists s.\text{defines}(s, l) \wedge \text{conformsTo}(a, s).$

Source: Marcel Heinz, Ralf Lämmel, Andrei Varanovich: **Axioms of Linguistic Architecture**. MODELSWARD 2017: 478-486

Understanding Membership

- $\text{Specification}(a) \Rightarrow \text{Artifact}(a)$.
- $\text{Language}(l) \Rightarrow \exists s.\text{Specification}(s) \wedge \text{defines}(s, l)$ ...
- $\text{defines}(a, e) \Rightarrow \text{Artifact}(a) \wedge \text{Entity}(e)$.
- $\text{conformsTo}(a, s) \Rightarrow \text{Artifact}(a) \wedge \text{Artifact}(s)$.
- $\text{conformsTo}(a, s) \Leftarrow (\forall p_a.\text{partOf}(p_a, a) \wedge \exists p_s.\text{partOf}(p_s, s)$
  $\wedge \text{ conformsTo}(p_a, p_s))$
  $\vee \exists t.\text{defines}(s, t) \wedge \text{elementOf}(a, t)$.

Source: Marcel Heinz, Ralf Lämmel, Andrei Varanovich: **Axioms of Linguistic Architecture**. MODELSWARD 2017: 478-486

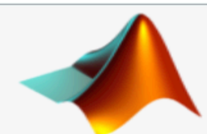
Language classification on Wikipedia/Dbpedia

https://en.wikipedia.org/wiki/MATLAB — ① URL

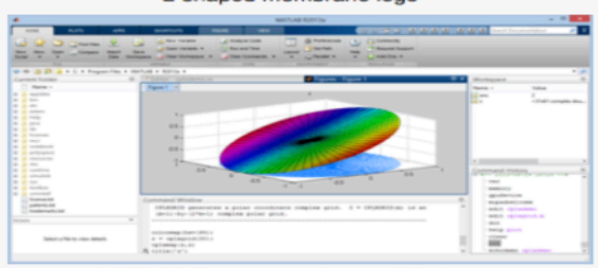
MATLAB (*matrix laboratory*) is a multi-paradigm numerical computing environment and proprietary programming language developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, C#, Java, Fortran and Python.

② Summary

MATLAB



L-shaped membrane logo[1]



MATLAB R2013a running on Windows 8

Developer(s)	MathWorks
Initial release	1984; 34 years ago
Stable release	R2018a / 15 March 2018; 3 months ago
Preview release	None [±]
Written in	C, C++, Java
Operating system	Windows, macOS, and Linux[2]
Platform	IA-32, x86-64
Type	Numerical computing
License	Proprietary commercial software
Website	mathworks.com/products/matlab

MATLAB

Paradigm	multi-paradigm: functional, imperative, procedural, object-oriented, array
Designed by	Cleve Moler
Developer	MathWorks
First appeared	late 1970s
Stable release	9.4 (R2018a) / March 14, 2018; 3 months ago
Preview release	None [±]
Typing discipline	dynamic, weak
Filename extensions	.m
Website	mathworks.com/products/matlab
Influenced by	
APL · EISPACK · LINPACK · PL/0 · Speakeasy[3]	
Influenced	
Julia[4] · Octave[5] · Scilab[6]	

MATLAB Programming at Wikibooks

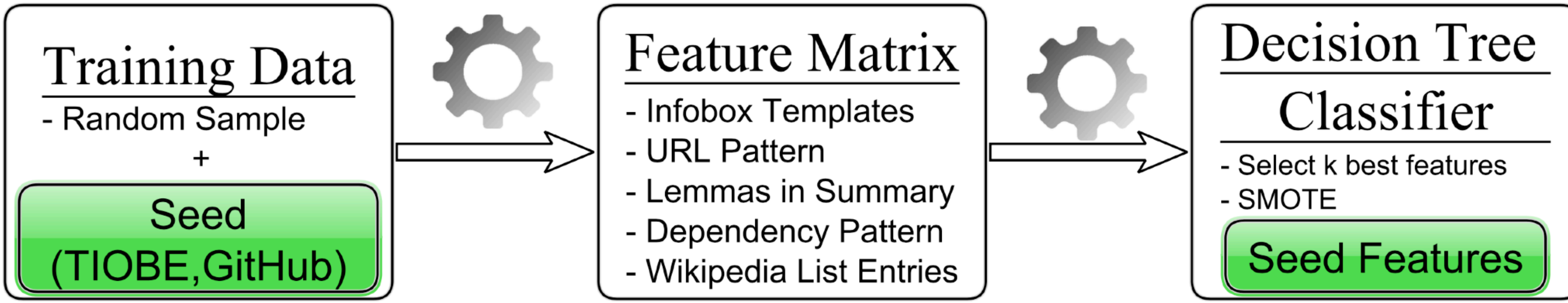
③ Infoboxes

④ Category Graph

Categories: Image processing software | Array programming languages | C software | Computer algebra system software for Linux | Computer algebra system software for MacOS | Computer algebra system software for Windows | Computer algebra systems | Computer vision software | Cross-platform software | Data mining and machine learning software | Data visualization software | Data-centric programming languages | Dynamically typed programming languages | Econometrics software | High-level programming languages | IRIX software | Linear algebra | Mathematical optimization software | Numerical analysis software for Linux | Numerical analysis software for MacOS | Numerical analysis software for Windows | Numerical linear algebra | Numerical programming languages | Numerical software | Parallel computing | Plotting software | Proprietary commercial software for Linux | Proprietary cross-platform software | Regression and curve fitting software | Software modeling language | Statistical programming languages | Time series software

Source: Marcel Heinz, Ralf Lämmel, Mathieu Acher: **Discovering Indicators for Classifying Wikipedia Articles in a Domain - A Case Study on Software Languages**. SEKE 2019: 541-706

ML approach to Wikipedia-based classification



Source: Marcel Heinz, Ralf Lämmel, Mathieu Acher: **Discovering Indicators for Classifying Wikipedia Articles in a Domain - A Case Study on Software Languages**. SEKE 2019: 541-706

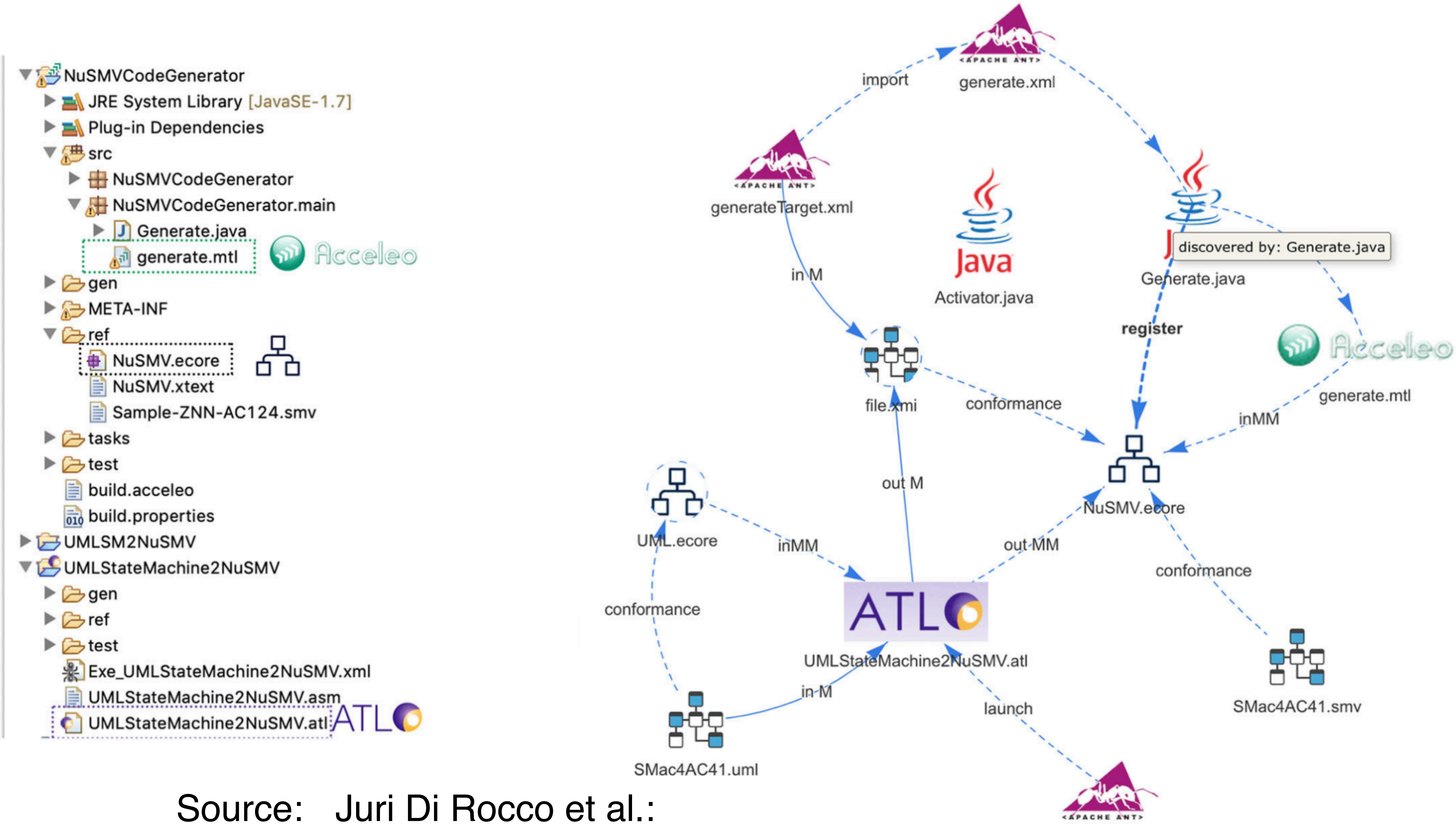
Knowledge graph population

A Challenge of Software Knowledge Analytics

- Example: Extract technology usage from repository:
  - Instance — Megamodels for model transformation:
    - Identify models/metamodels/transformations.
    - Draw links between those identities.

Image by Pete Linforth from Pixabay

Raw data versus knowledge graph



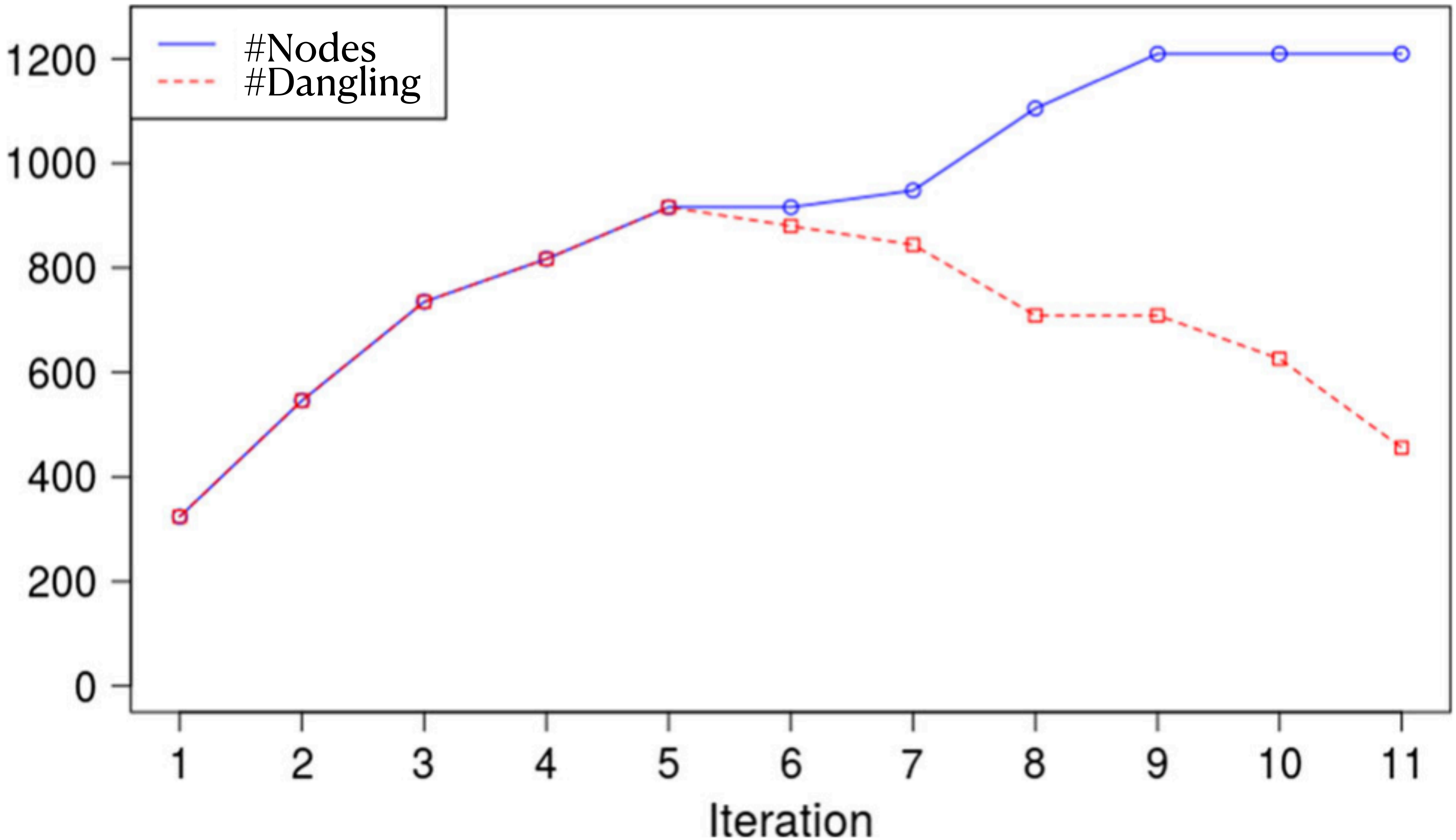
Source: Juri Di Rocco et al.:
Understanding MDE projects: megamodels to the rescue for architecture recovery.
Softw. Syst. Model. 19(2): 401-423 (2020)

Increasing number of extraction heuristics

Iteration	Applied heuristics	#Nodes	#Edges	#Dangling nodes
1	EH	324	0	324
2	EH, AH	546	0	546
3	EH, AH, KH	735	0	735
4	EH, AH, KH, LH	817	0	817
5	EH, AH, KH, LH, ANH	916	0	916
6	EH, AH, KH, LH, ANH, APH	916	37	880
7	EH, AH, KH, LH, ANH, APH, LTH	948	212	844
8	EH, AH, KH, LH, ANH, APH, LTH, ANATLH	1105	831	709
9	EH, AH, KH, LH, ANH, APH, LTH, ANATLH, JH	1210	831	709
10	EH, AH, KH, LH, ANH, APH, LTH, ANATLH, JH, TOTEMH	1210	1039	626
11	EH, AH, KH, LH, ANH, APH, LTH, ANATLH, JH, TOTEMH, KM3ECOREH	1210	1112	456

EH EcoreHeuristic, *AH* ATLHeuristic, *KH* KM3Heuristic, *LH* LauncherHeuristic, *ANH* ANTHeuristic, *APH* ATLWithPathHeuristic, *LTH* LauncherATLHeuristic, *ANATLH* ANTWithATLHeuristic, *JH* JavaHeuristic, *TOTEMH* ATLWithTOTEMHeuristic, *KM3ECOREH* KM32ECOREHeuristic

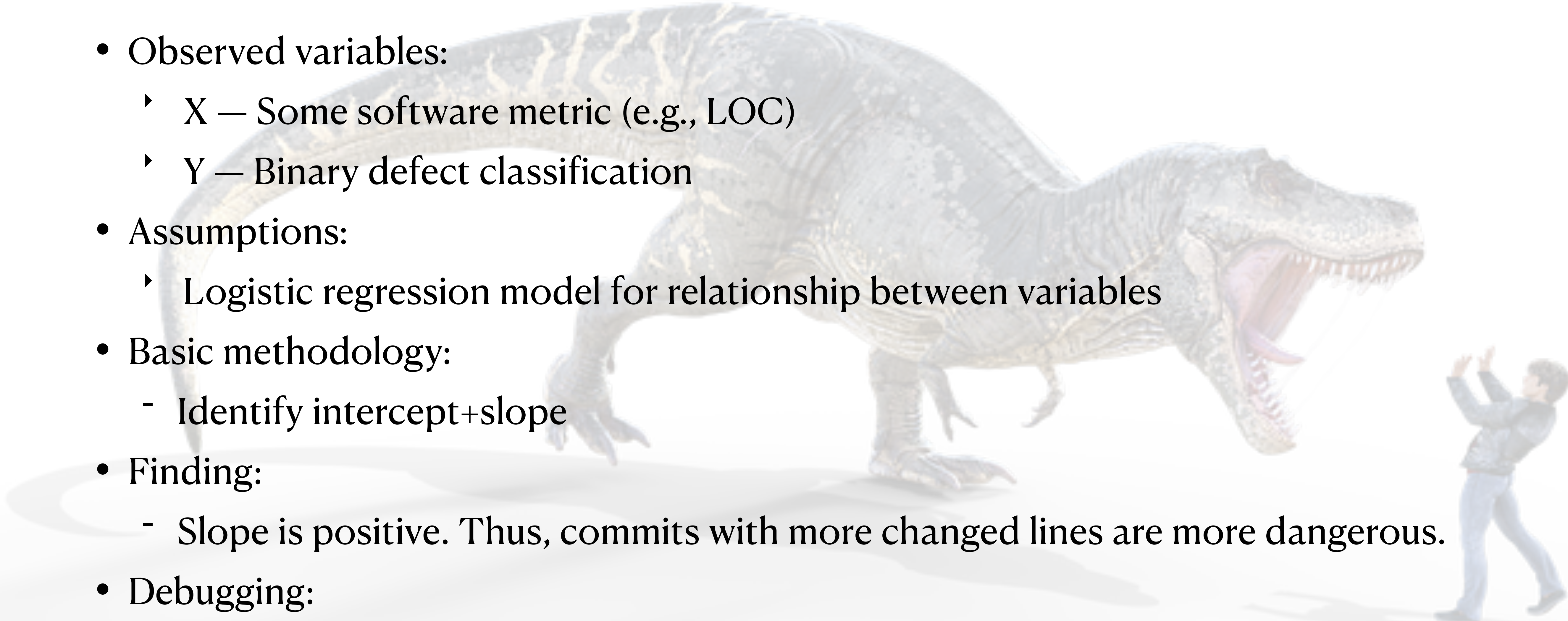
Nodes recovered



Managing threats to validity

A Challenge of Software Knowledge Analytics

Example: Debugging a software defect analysis



- Observed variables:
  - X — Some software metric (e.g., LOC)
  - Y — Binary defect classification
- Assumptions:
  - Logistic regression model for relationship between variables
- Basic methodology:
  - Identify intercept+slope
- Finding:
  - Slope is positive. Thus, commits with more changed lines are more dangerous.
- Debugging:
  - Replace some observed and unobserved variables by synthetic data.

Managing threats to validity

A Challenge of Software Knowledge Analytics

Example: Debugging a software defect analysis

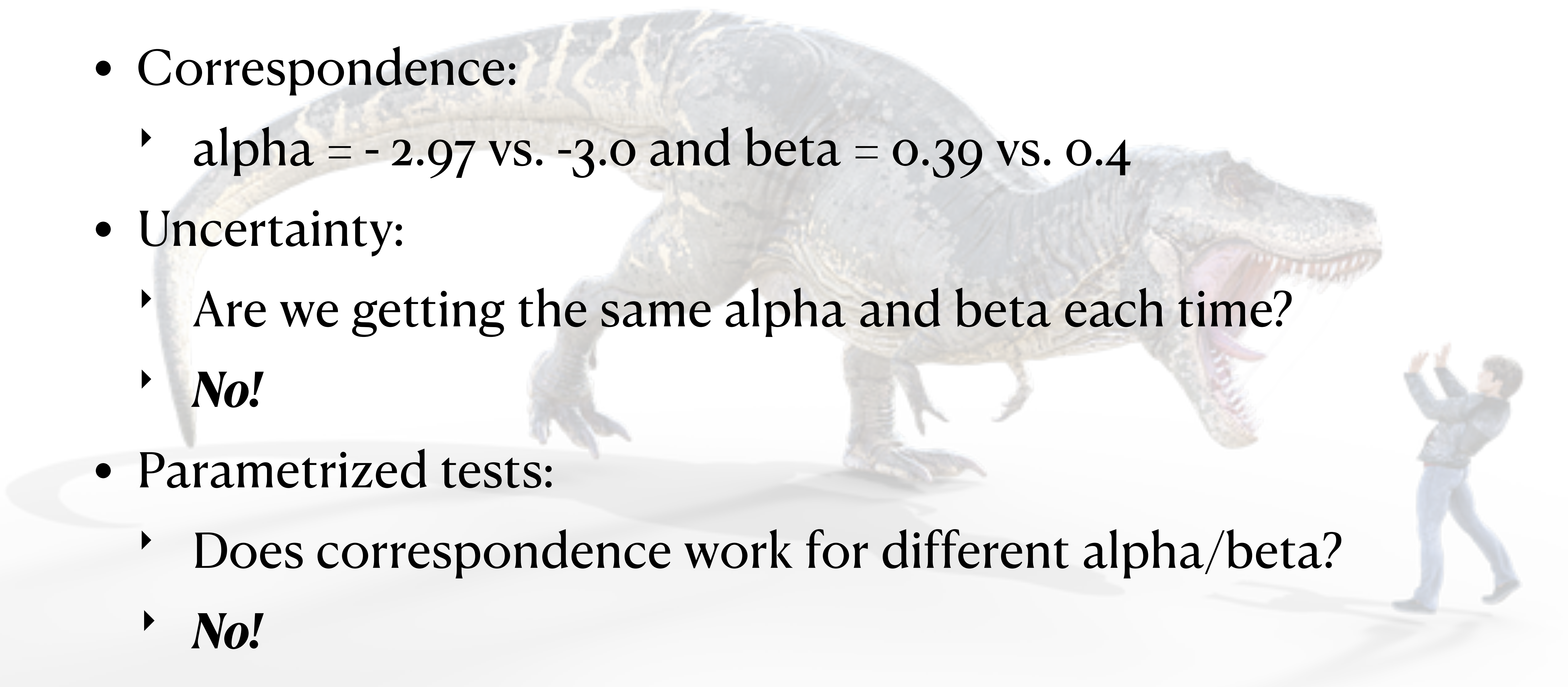
R code which substitutes variables of the original methodology by synthetic variables

```
# Kept observed variables.
N ← N # Number of commits.
X ← X # (vector) Keep the original variable X.

# Substituted unobserved variables.
alpha ← -3.0
beta ← 0.4
prob ← 1 / (1 + exp(-(alpha + beta * X))) # (vector)
      Assumption of the logistic regression model on
      the relation between X and Y.

# Substituted observed variable Y.
Y ← rbinom(N, size = 1, prob = prob) # (vector)
      Assumption on the output distribution.
```

Managing threats to validity

A Challenge of Software Knowledge Analytics

Results of debugging the software defect analysis

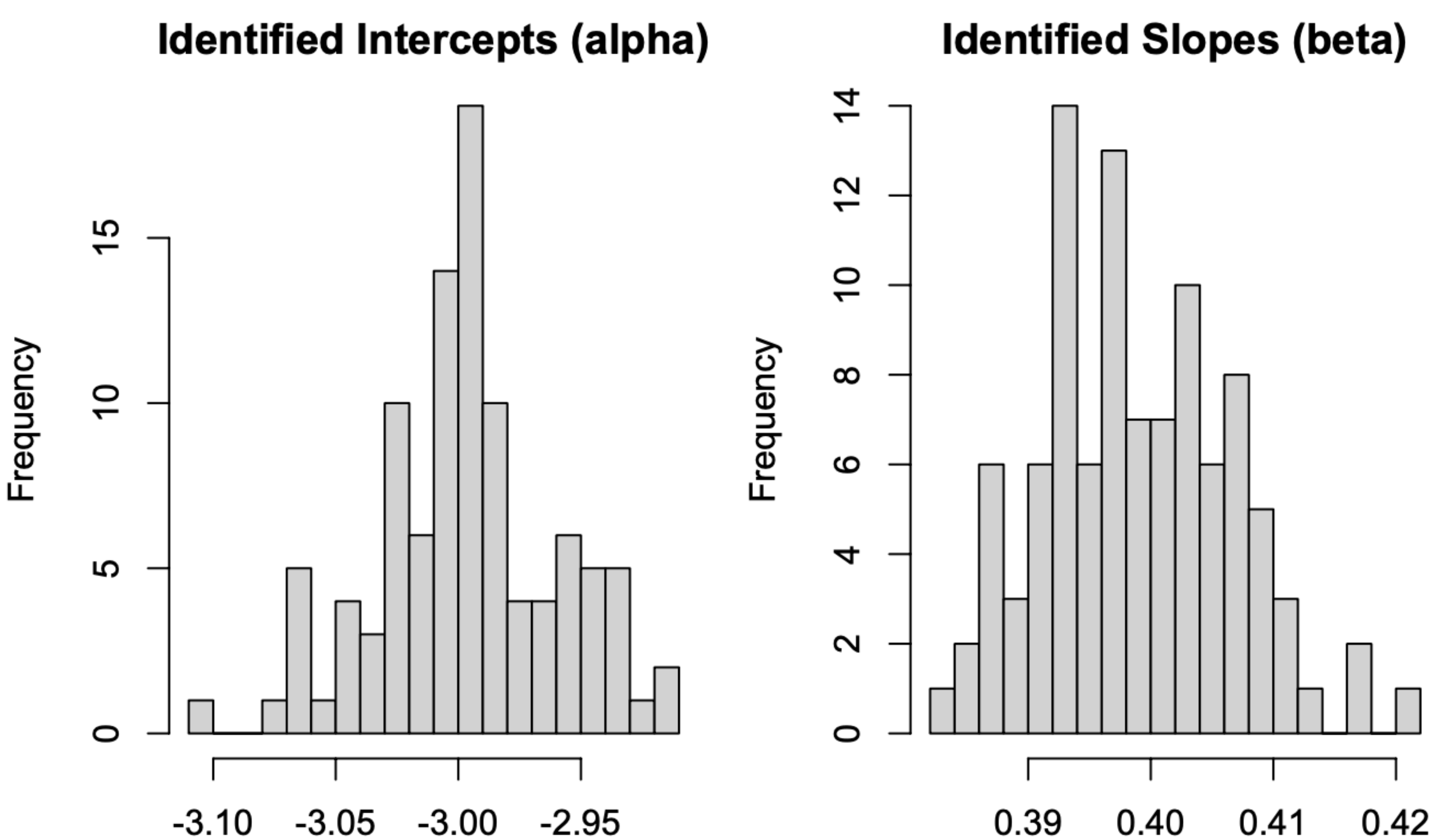


- Correspondence:
  - alpha = - 2.97 vs. -3.0 and beta = 0.39 vs. 0.4
- Uncertainty:
  - Are we getting the same alpha and beta each time?
  - ***No!***
- Parametrized tests:
  - Does correspondence work for different alpha/beta?
  - ***No!***

Managing threats to validity

A Challenge of Software Knowledge Analytics

Uncertainty for software defect analysis



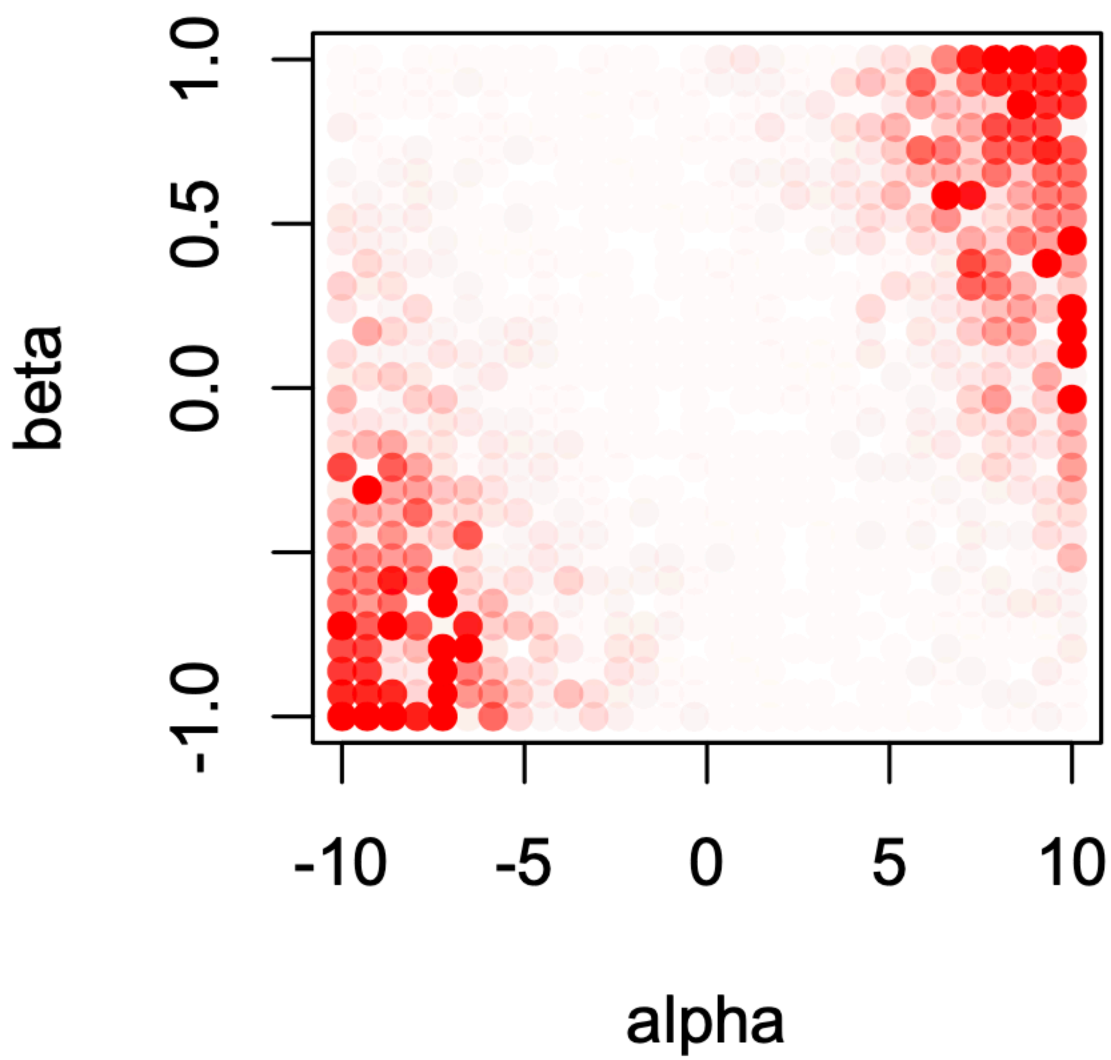
Basic methodology cannot observe *prob!*

Source: Johannes Härtel, Ralf Lämmel: Operationalizing Threats to MSR Studies by Simulation-Based Testing. MSR 2022

Managing threats to validity

A Challenge of Software Knowledge Analytics

Parametrized tests for software defect analysis



Simulated alpha and beta and the corresponding error in the identification, depicted as red dots (red in- creases with error).

Source: Johannes Härtel, Ralf Lämmel:
Operationalizing Threats to MSR Studies by Simulation-Based Testing.
MSR 2022

Table of contents

- *Showcases*
- *Principles*
- *Challenges*

of Software Knowledge Analytics

All
Done

Outlook

Technical lecture topics for the next few days

- **API clustering – An exercise in abstraction**
- **Joint API usage – An exercise in causality**
- **Graph language proliferation – An exercise in (language) usage analysis**
- **Knowledge graph validation – An exercise in reasoning (with contexts & metadata)**
- **Classifier discovery on Wikipedia – An exercise in ML-based knowledge engineering**
- **Developer workflow modeling – An exercise in process mining**
- **Linguistic architecture recovery – An exercise in rule-based reasoning**
- **Simulation of MSR/ESE studies – An exercise in debugging threats to validity**
- **Regression analysis of defect data – An exercise in multilevel modeling**
- **API developer profiles – An exercise in hypothesis building and validation**

Image by Alexas_Fotos from Pixabay