Keeping one-step ahead of Architectural Smells:
A Machine Learning Application

Dr. Antonela Tommasel

ISISTAN, CONICET-UNICEN, Argentina

antonela.tommasel@isistan.unicen.edu.ar

CONICET

ISISTAN

Who am I?

- Dr. Antonela Tommasel
  - PhD in Computer Sciences at UNICEN, December 2017
- Work at ISISTAN, CONICET-UNICEN.
- Teacher at UNICEN.
- Research Interests:
  - Recommender systems
  - Text Mining
  - Social Media
  - Machine Learning
  - …

Table of Contents

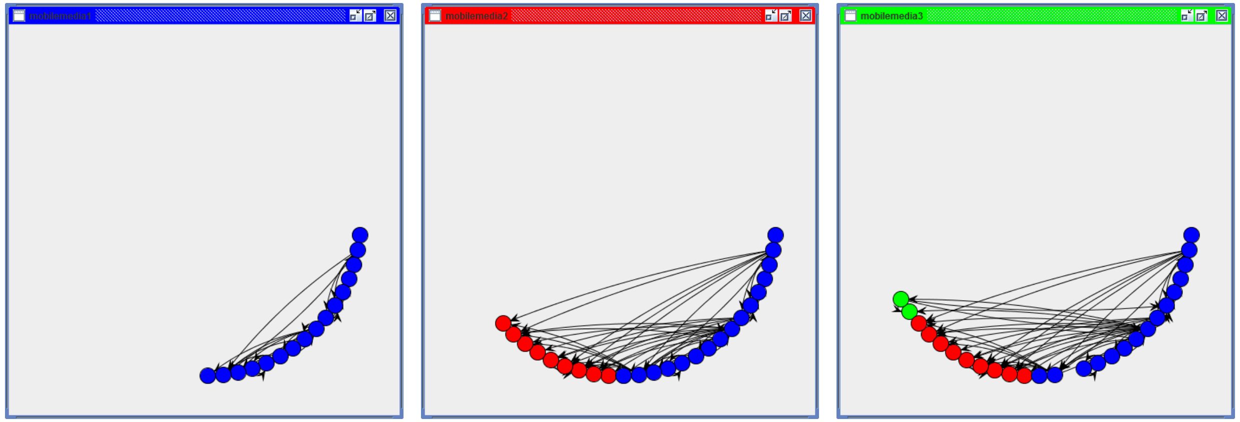
Table of Contents

Software Evolution & Dependencies

- As software systems evolve, the **amount and complexity of the interactions** amongst their components often increases.
  - More coupling.
  - **“*Undesired*” dependencies** amongst certain components (e.g., layer bridging, direct access to databases, cycles).
  - <u>Degradation</u> of intended design.

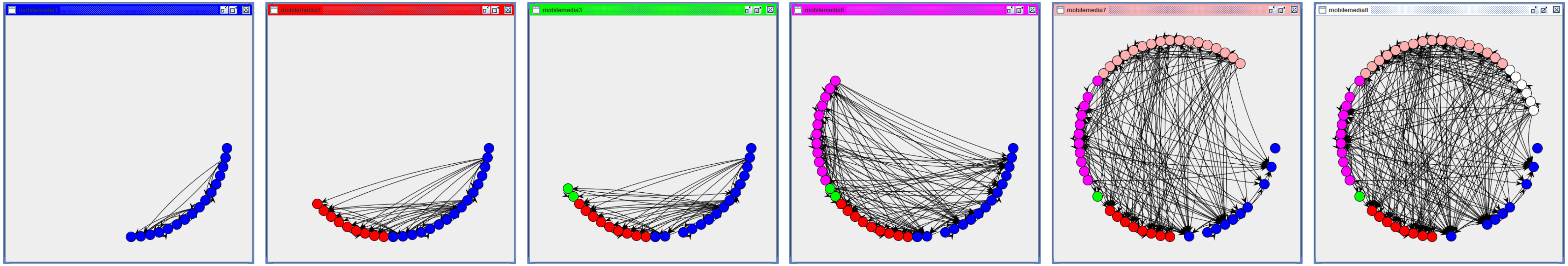
Software Evolution & Dependencies

- As software systems evolve, the **amount and complexity of the interactions** amongst their components often increases.
    - More coupling.
    - **“*Undesired*” dependencies** amongst certain components (e.g., layer bridging, direct access to databases, cycles).
    - <u>Degradation</u> of intended design.

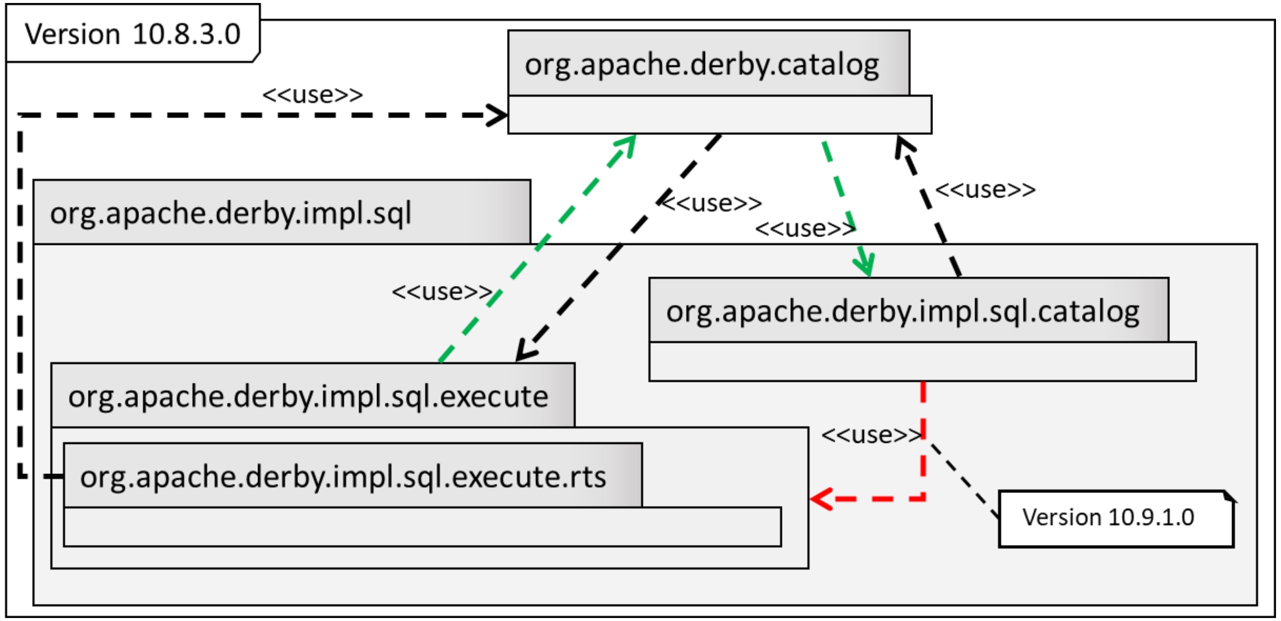


Software Evolution & Dependencies

- As software systems evolve, the **amount and complexity of the interactions** amongst their components often increases.
  - More coupling.
  - **“*Undesired*” dependencies** amongst certain components (e.g., layer bridging, direct access to databases, cycles).
  - <u>Degradation</u> of intended design.

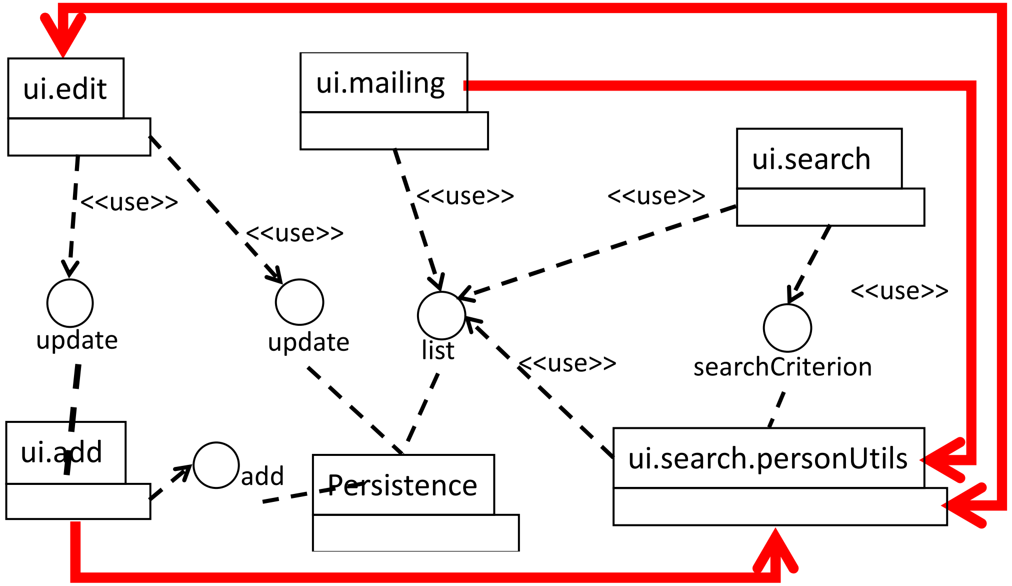


Software Evolution & Dependencies

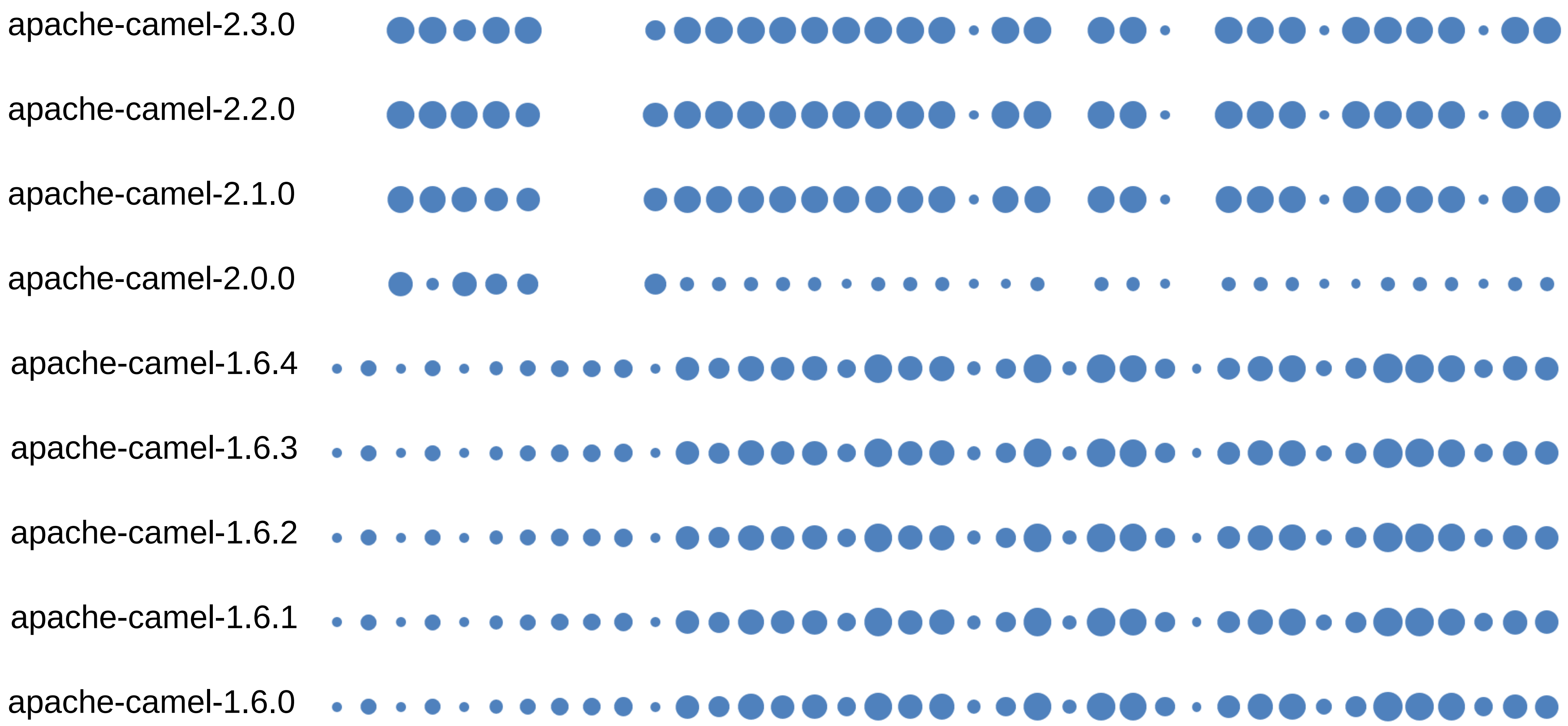


Apache Derby
Introduction of a Cyclic dependency

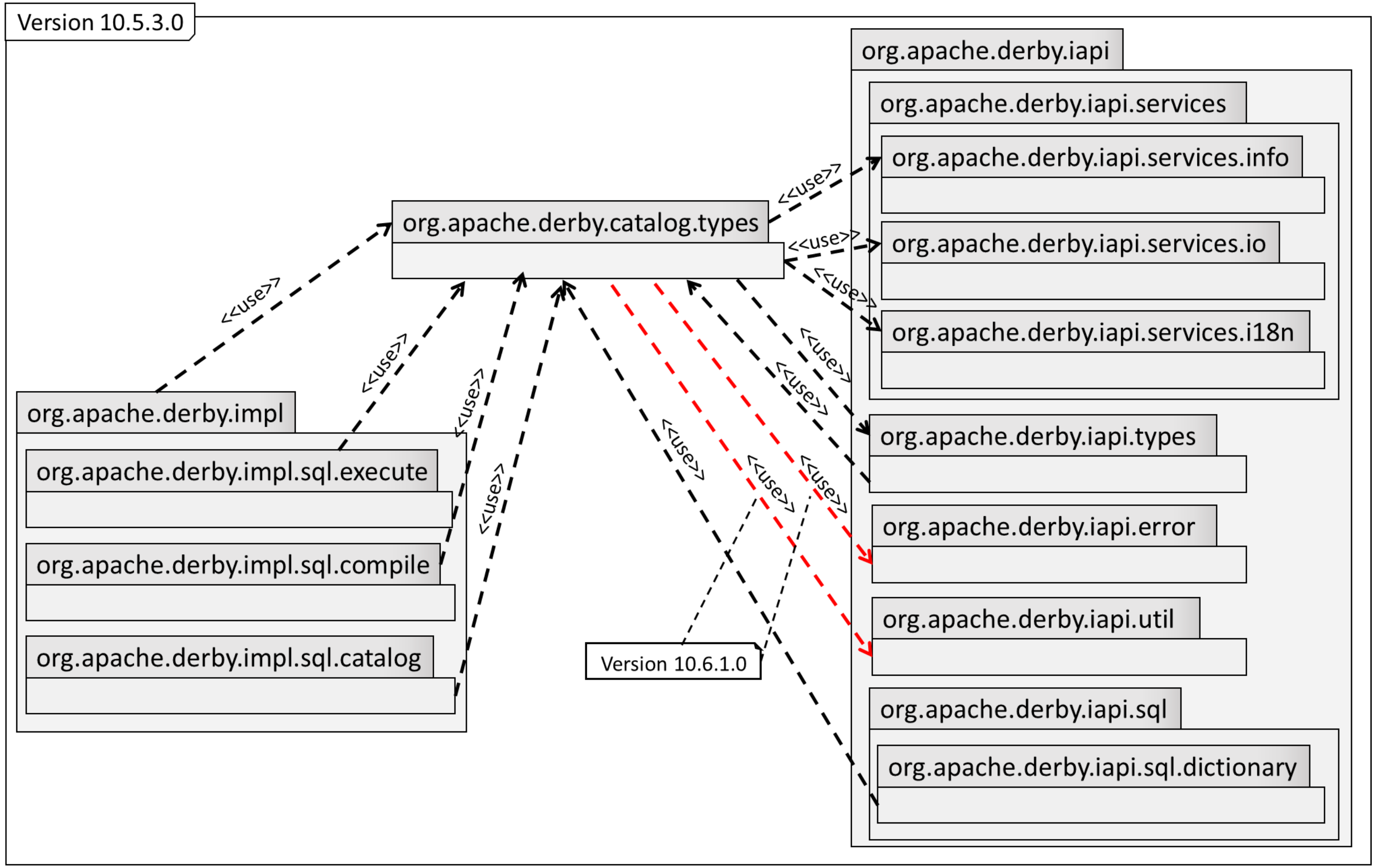
SubscriberDB
Violation of architectural rules (based on a given architecture)



Software Evolution & Dependencies



Software Evolution & Dependencies



Apache Derby
Introduction of a
Hub-like dependency

Software Evolution & Dependencies

- Conscious efforts must be made to stop (or alleviate) degradation.

  - Plan for corrective actions (e.g., refactoring).
  - Monitor system health (e.g., via metrics).
  - Conformance checks.

Software Evolution & Dependencies

- Conscious efforts must be made to stop (or alleviate) degradation.

  - Plan for corrective actions (e.g., refactoring).
  - Monitor system health (e.g., via metrics).
  - Conformance checks.

The early detection of such symptoms is **important** for developers, so that they can **plan ahead** for actions that **preserve** the **quality** of the system.

What can we do about it?

- Different tools available
  - LattixDSM, SonarQube, SonarGraph, JITTAC.

What can we do about it?

- Different tools available
  - LattixDSM, SonarQube, SonarGraph, JITTAC.

- Identification of problems **once they occurred** in the system!
  - Tools normally perform a dependency analysis of the source code.
  - Compute metrics/indicators, ranking of smells (e.g., by severity).
  - Show all these symptoms to developers.

What can we do about it?

However, developers may be **reluctant** to fix **problems**, when they were **already introduced** in the code.

Particularly, quality-related problems.
Schedule pressures, “it still works”, loss of context.

What can we do about it?

However, developers may be **reluctant** to fix **problems**, when they were **already introduced** in the code.

Particularly, quality-related problems.
Schedule pressures, “it still works”, loss of context.

Predict when a dependency-related problem is likely to manifest!

Social Network Analysis to the Rescue!

- Although there are approaches for <u>computing coupling metrics</u>, very **few** of them have dealt with the **prediction** of **dependency** relations amongst software components.

Social Network Analysis to the Rescue!

- Although there are approaches for computing coupling metrics, very **few** of them have dealt with the **prediction** of **dependency** relations amongst software components.

- A particular graph-based approach is social networks analysis (SNA), which has been used for modelling both nature and human phenomena.

SNA techniques can predict links that yet do not exist between pairs of nodes in a network.

Social Network Analysis to the Rescue!

…and Software Engineering?

- **Evidence** that the **topological** features of **dependency graphs** can reveal interesting properties of the software system under analysis.

- Nonetheless, ***SNA*** techniques has not yet greatly exploited in the Software Engineering community.

Social Network Analysis to the Rescue!

…and Software Engineering?

- **Evidence** that the **topological** features of **dependency graphs** can reveal interesting properties of the software system under analysis.

- Nonetheless, ***SNA*** techniques has not yet greatly exploited in the Software Engineering community.

We argue that **Social Network Analysis** techniques need to be revisited with respect to **software dependency prediction**!

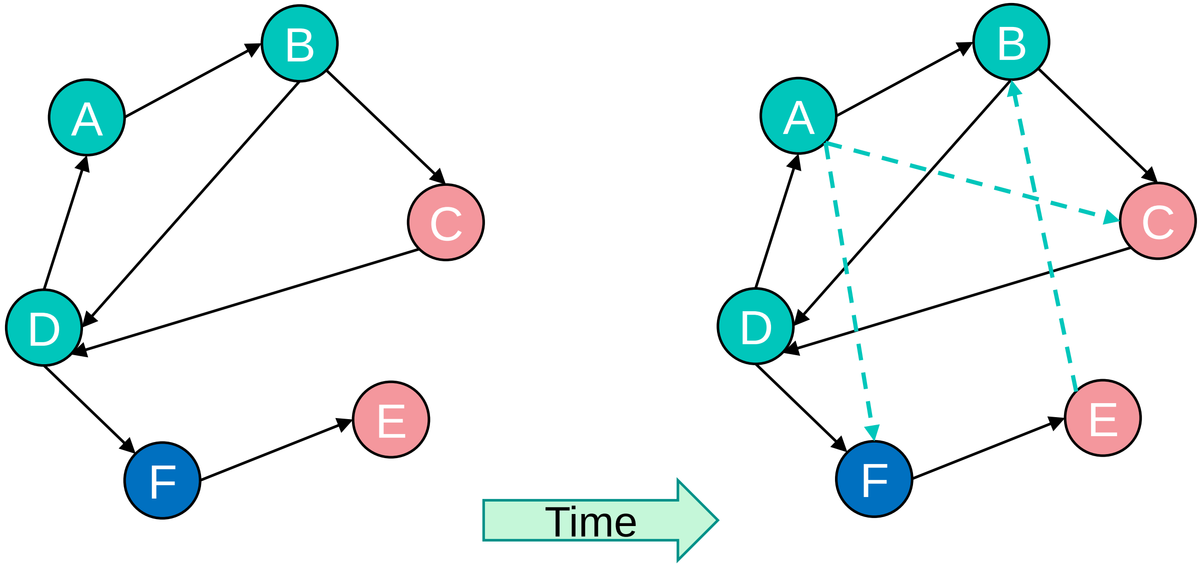
Table of Contents

First Try: Link Prediction Techniques

- **Link Prediction** studies the **evolution of a network/graph** using models of network <u>features.</u>
  - Infer “missing” links between pairs of nodes.
  - Based on the observable links of the network and their attributes.

  - **<u>Homophily Principle (HP)</u>**: interactions between similar nodes occur at a higher rate than interactions between dissimilar nodes.

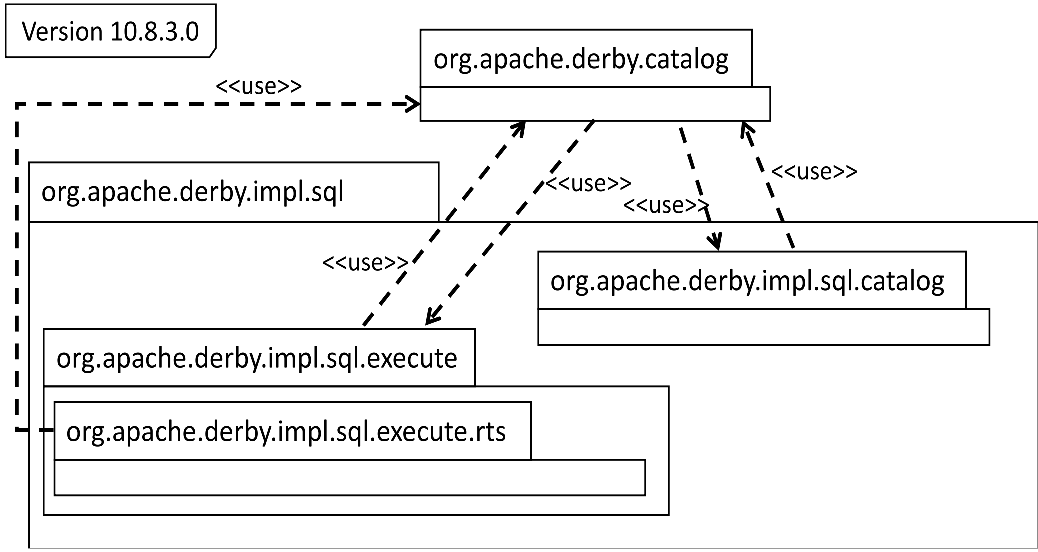
  - Most techniques rely on graph topological features that **assess similarity between pairs of nodes.**



First Try: Link Prediction Techniques

…but we need a graph

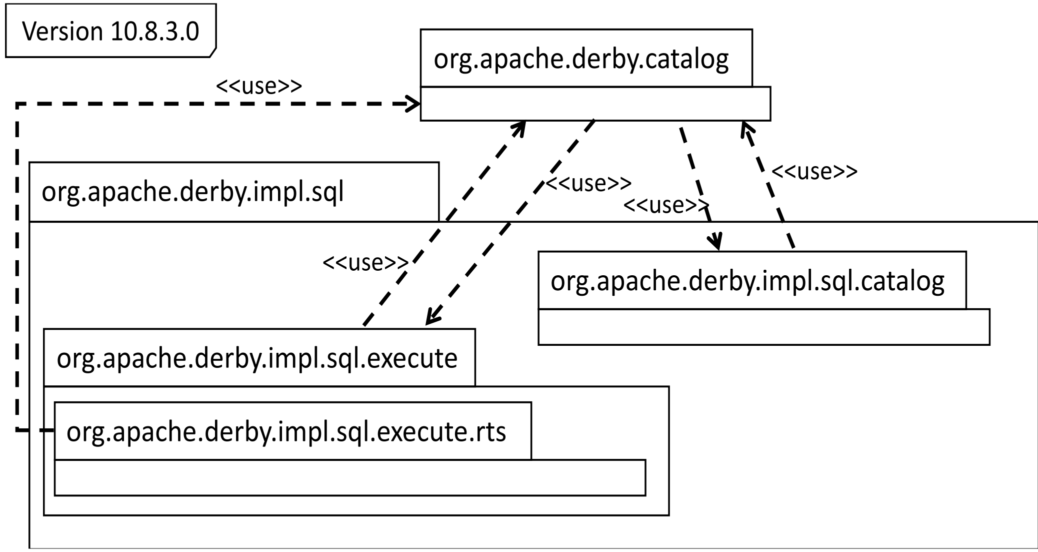
- Build a graph $DG(V, E)$ for system version n, where:
  - Each node v in V is Java package, and each edge *e* in *E* is a usage relationship between a pair of packages v_1 and $v_{2.}$



First Try: Link Prediction Techniques

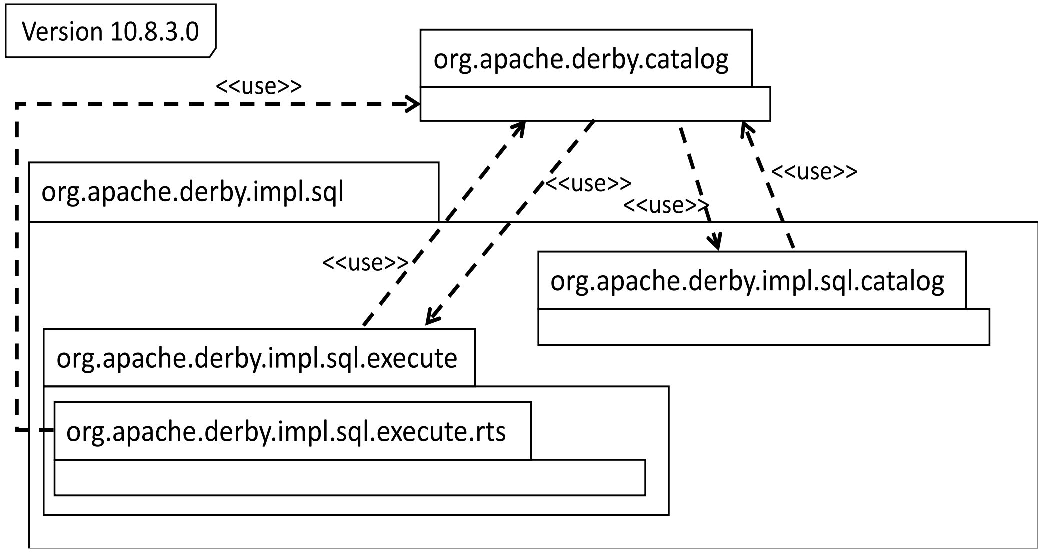
…but we need a graph

- Build a graph $DG(V, E)$ for system version n, where:
  - Each node v in V is Java package, and each edge *e* in *E* is a usage relationship between a pair of packages v_1 and $v_{2.}$
  - <u>Assumption 1</u>: The package structure remains stable over versions.
  - <u>Assumption 2</u>: Similar packages have a high chance to establish usage dependencies.
  - Compute $score(v_1, v_2)$ to assess similarity.

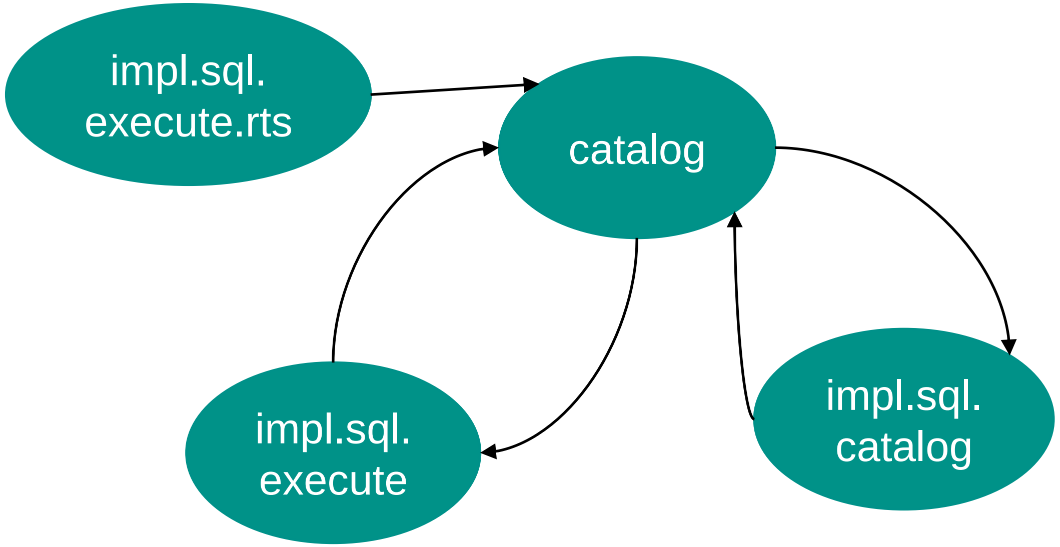


First Try: Link Prediction Techniques

…but we need a graph



- Build a graph $DG(V, E)$ for system version n, where:
  - Each node v in V is Java package, and each edge *e* in *E* is a usage relationship between a pair of packages v_1 and $v_{2.}$
  - Assumption 1: The package structure remains stable over versions.
  - Assumption 2: Similar packages have a high chance to establish usage dependencies.
  - Compute $score(v_1, v_2)$ to assess similarity.



First Try: Link Prediction Techniques

Measuring Similarity Between Nodes

Standard Topological Similarity Metrics

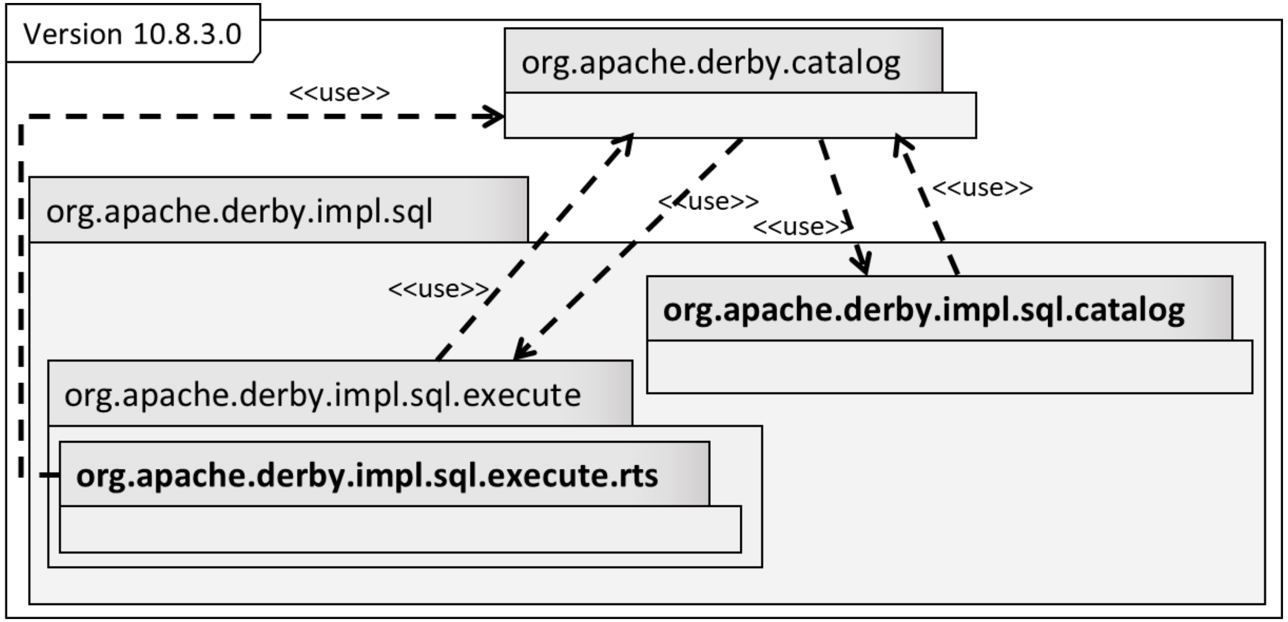
Common Neighbours	Adamic Adar	Kats Score	SimRank	…

Source-code Similarity Metrics

Kunczynsky	Relative Matching	Russel Rao	…

First Try: Link Prediction Techniques

Measuring Similarity Between Nodes



$$Common\ Neighbors = |\mathrm{T}(x) \cap \mathrm{T}(y)|$$

$$|\mathrm{T}(org.apache.derby.impl.sql.execute.rts) \cap \mathrm{T}(org.apache.derby.impl.sql.catalog)|$$

$$\{org.apache.derby.catalog\} \qquad \{org.apache.derby.catalog\}$$

$$\{org.apache.derby.catalog\}$$

$$Common\ Neighbors = 1$$

First Try: Link Prediction Techniques

What do we want?

To what extent LP can leverage on information from software versions to predict likely dependencies in the next version, for those pairs of modules that exist in the analysed versions.

First Try: Link Prediction Techniques

What do we want?

To what extent LP can leverage on information from software versions to predict likely dependencies in the next version, for those pairs of modules that exist in the analysed versions.

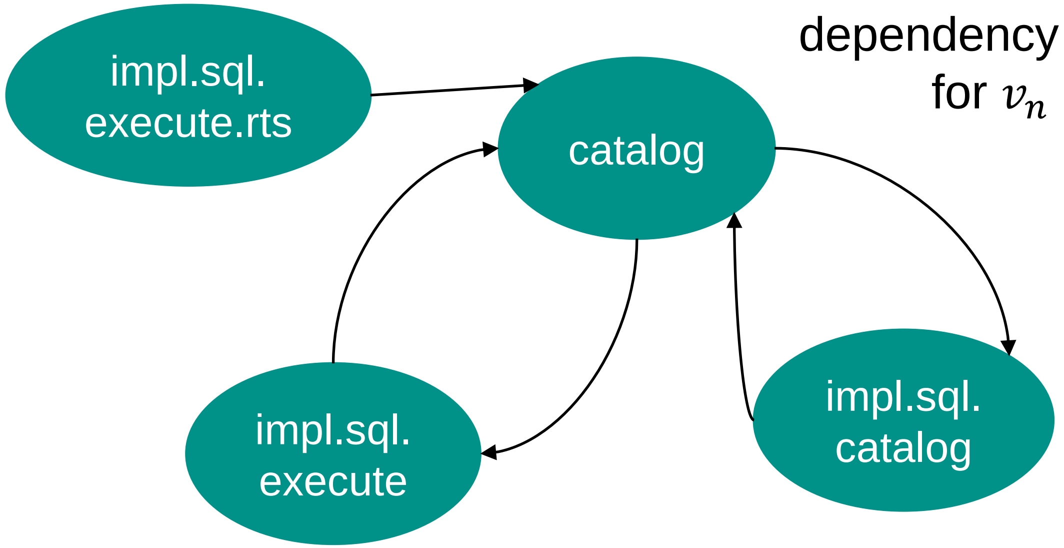
- For a package p, a **ranking** of packages is built, based on their chance of having a future dependency with p, according to a similarity metric.

- For pairs of consecutive versions, the quality of predictions was evaluated in terms of *precision* (i.e., the ratio of actual dependencies discovered to the total number of predictions) for the top-N dependencies of the ranking.

First Try: Link Prediction Techniques

What do we want?

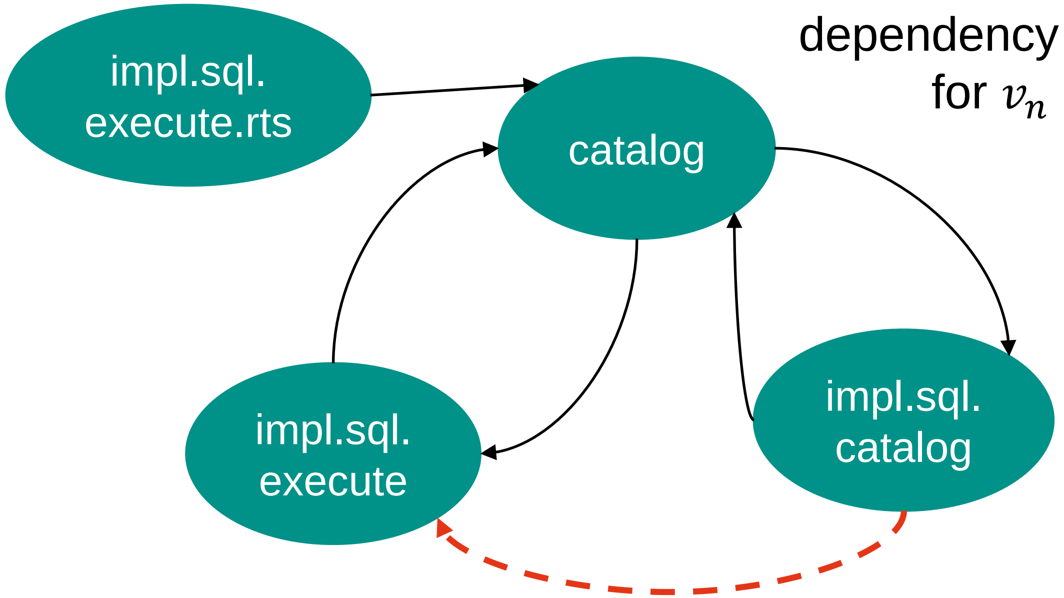
dependency graph
for v_n



First Try: Link Prediction Techniques

What do we want?

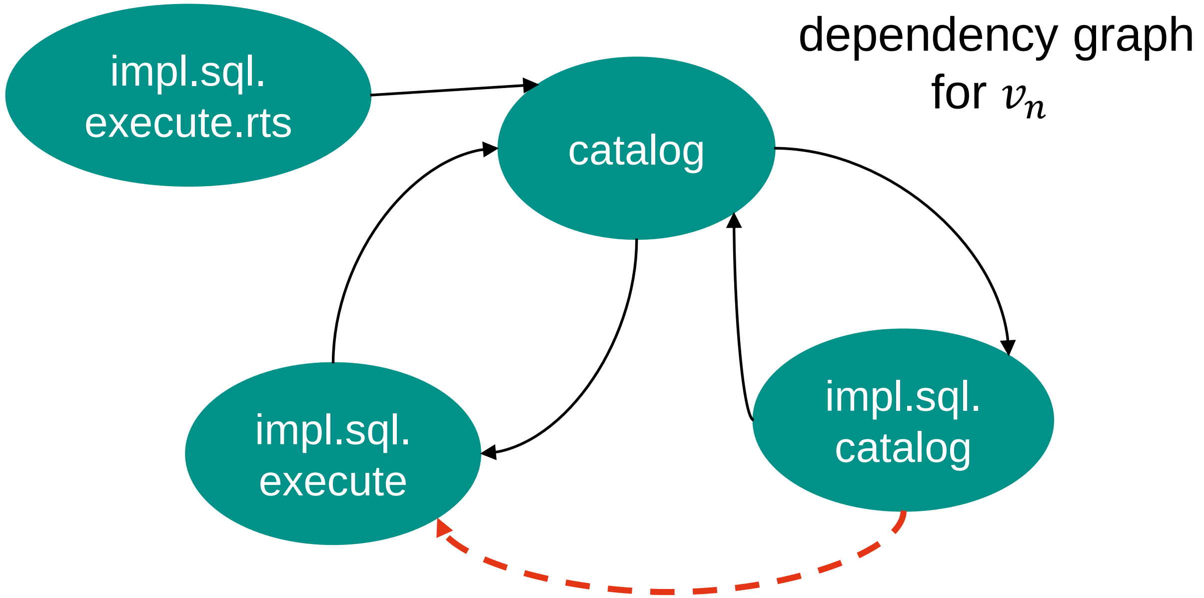
dependency graph
for v_n



Will this dependency
appear on v_{n+1}?

First Try: Link Prediction Techniques

What do we want?



Output for v_{n+1}

Ranking	Common Neighbours	Adamic-Adar
1	impl.sql	**impl.sql.execute**
2	**impl.sql.execute**	impl.sql
3	impl.sql.conn	impl.sql.con
4	impl.db	impl.db
5	impl.store.raw.data	impl.jdbc

First Try: Link Prediction Techniques

Study Settings

- We analysed **package dependencies** in general.
  - Unrelated to specific design problems.

- Dependencies between **classes** were **ignored**.

First Try: Link Prediction Techniques

Study Settings

- We analysed **package dependencies** in general.
  - Unrelated to specific design problems.

We are going to tackle this later!

- Dependencies between **classes** were **ignored**.

First Try: Link Prediction Techniques

Study Settings

Would require to predict new packages

- We analysed **package dependencies** in general.
  - Unrelated to specific design problems.
- Dependencies between **classes** were **ignored**.
- For Link Prediction to produce reasonable outputs, a pair of consecutive versions:
  - v_n and v_{n+1} have approximately the same number of packages.
  - v_{n+1} adds new dependencies between known packages.
  - New dependencies in v_{n+1} between new packages are disregarded.

First Try: Link Prediction Techniques

Study Settings

Two small Java systems.

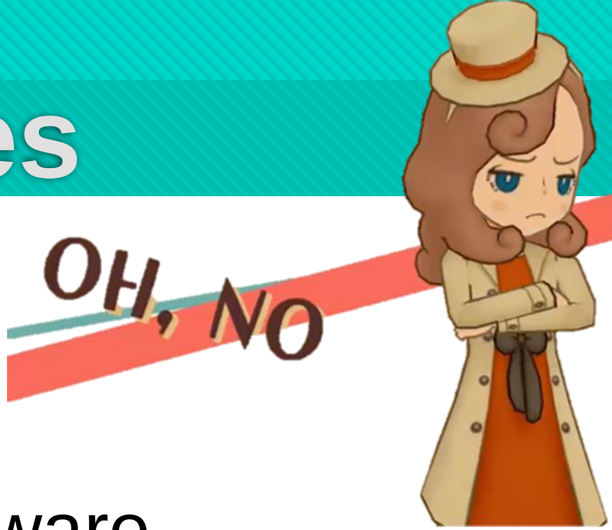
- HealthWatcher (HW)
  - 49 KLOC
- SubscriberDB (SDB)
  - 10 KLOC

	#c	#p	#deps
HWv1	88	19	67
HWv2	92	20	70 (+8,-5)
HWv3	104	21	75 (+5)
HWv4	106	22	85 (+10)
HWv5	108	22	86 (+7,-2)
HWv6	112	23	91
HWv7	116	23	91
HWv8	120	24	96 (+5)
HWv9	132	24	97 (+1)
HWv10	135	25	101 (+4)

	#c	#p	#deps
SDBv1	98	14	30
SDBv2	167	16	47 (+17)
SDBv3	192	17	50 (+4,-1)
SDBv4	193	17	50
SDBv5	193	17	50
SDBv6	193	17	50
SDBv7	195	17	50
SDBv8	195	17	51 (+1)
SDBv9	195	17	51
SDBv10	195	17	51

First Try: Link Prediction Techniques

So far not so good



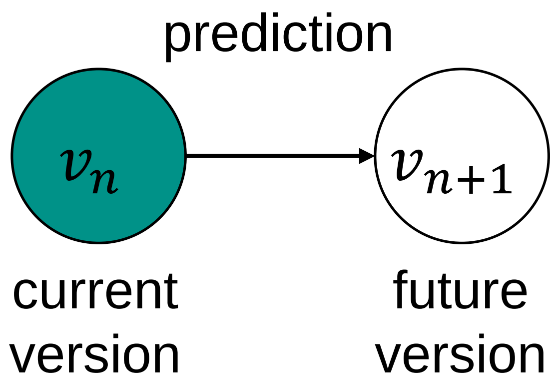
- Unfortunately, **ranking-based** LP is not enough for software dependencies.
  - Precision of predicted links is rather low (0.14-0.25 at most).

  - The **Homophily Principle** <u>does not always hold</u> for Java packages.
    - e.g., dependencies might still appear between dissimilar packages.

  - Two similar packages can **intentionally** be designed to not become dependent on each other.
    - e.g., based on business logic or modularity considerations.

Second Try: Apply Machine Learning Models

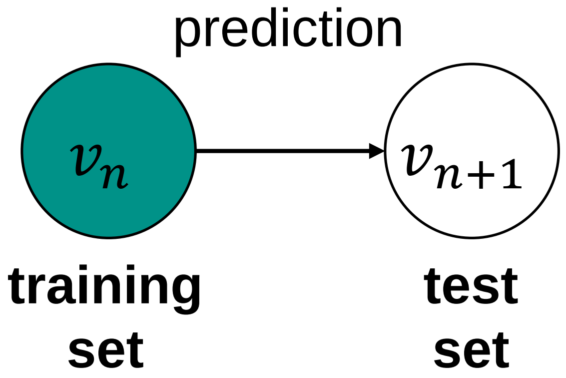
To what extent Link Prediction can leverage on information from the current version to predict dependencies in the next version?



Use statistical techniques to give computer systems the ability to "learn" (on a specific task) with data, without being explicitly programmed

Second Try: Apply Machine Learning Models

To what extent Link Prediction can leverage on information from the current version to predict dependencies in the next version?



Use statistical techniques to give computer systems the ability to "learn" (on a specific task) with data, without being explicitly programmed

Second Try: Apply Machine Learning Models

We need a “dataset”

A **binary classifier** is trained using the topological information provided by a given graph version.

Second Try: Apply Machine Learning Models

We need a “dataset”

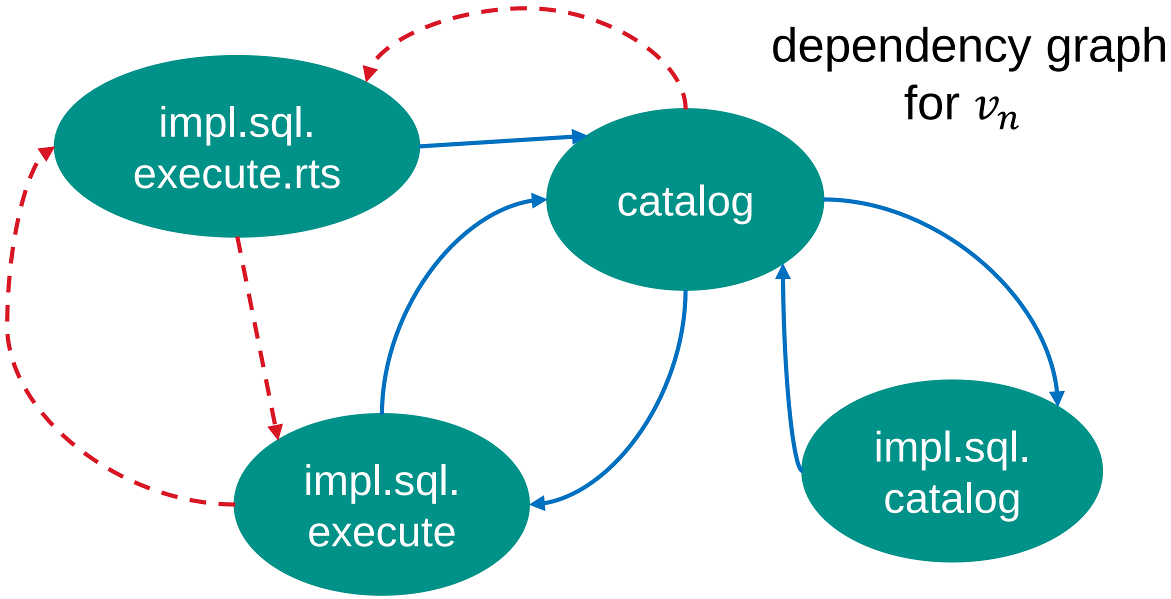
A **binary classifier** is trained using the topological information provided by a given graph version.

- An instance for the classifier consists of:
    - A pair of **nodes**.
    - A list of **features** (e.g., structural metrics) for the pair.
    - A **label** indicating if the nodes are linked (positive class) or not (negative class).

Second Try: Apply Machine Learning Models

We need a "dataset"

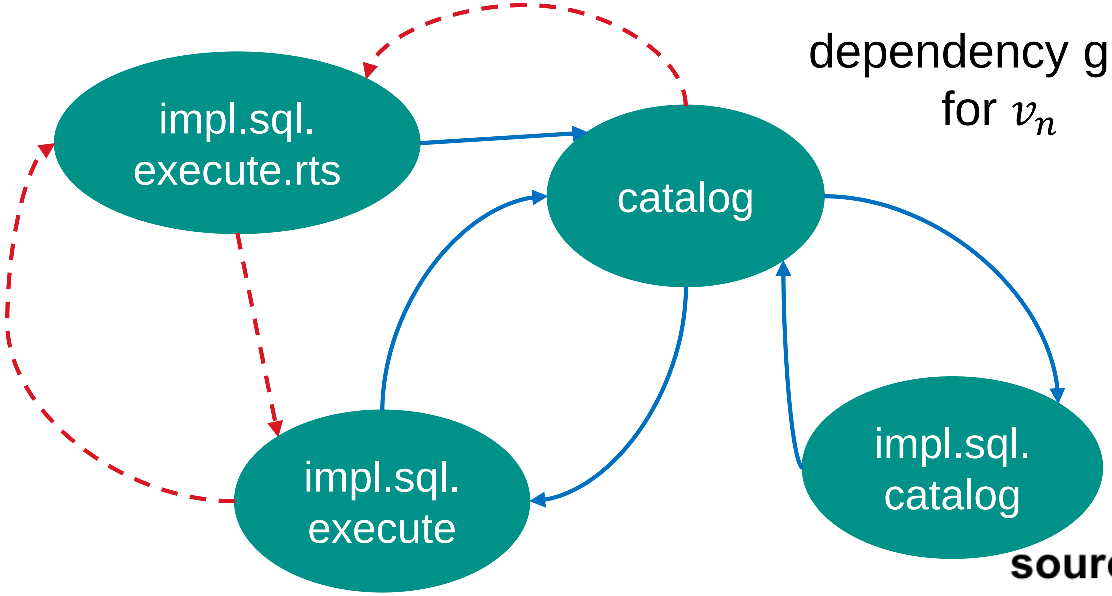
- **Existing dependencies** are used to compute features for instances of the positive class.
- **Missing dependencies** are used to compute features for instances of the negative class.



- Both training and test sets need to be defined.
  - The training set considers the known structure of v_n.
  - The test set considers the full graph of v_{n+1}.

Second Try: Apply Machine Learning Models

We need a “dataset”



dependency graph
for v_n

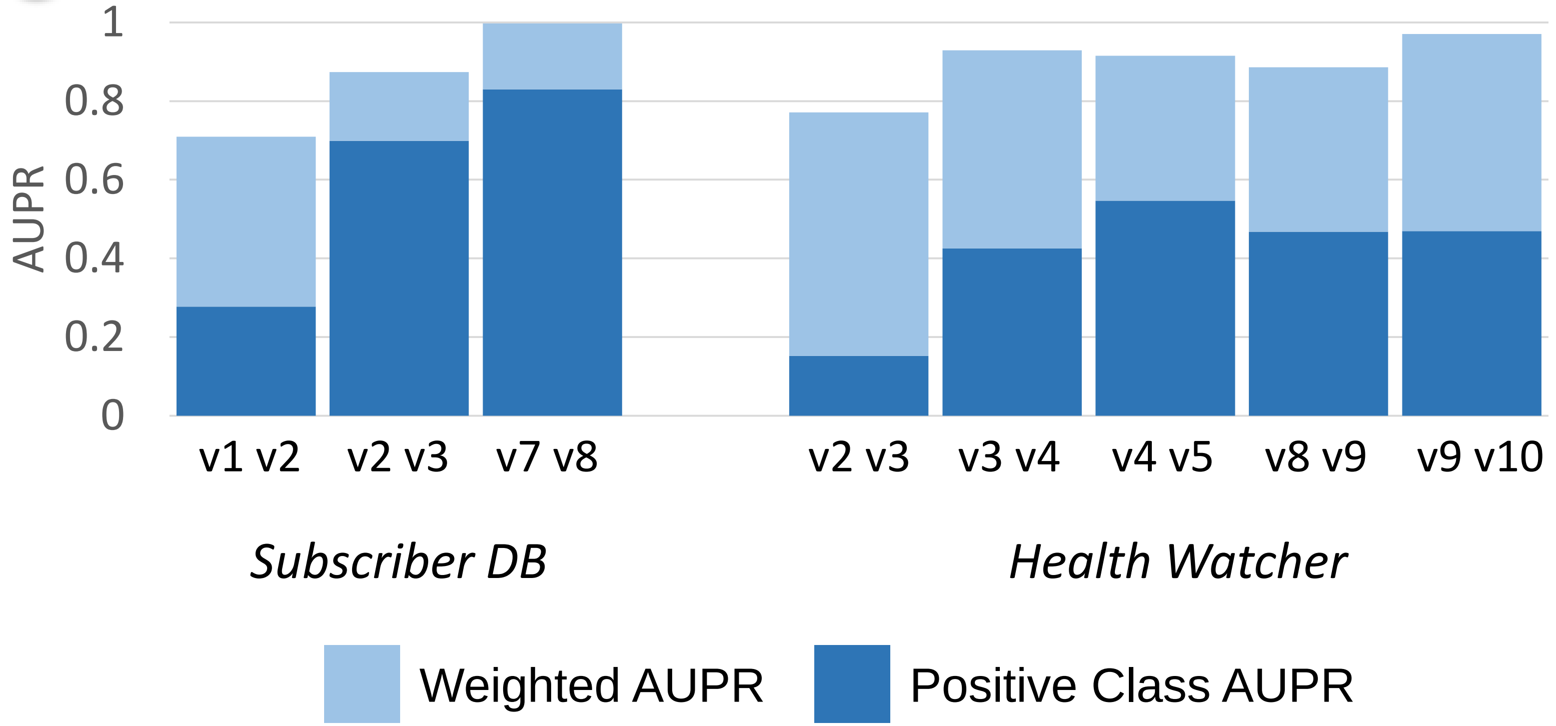
Output v_{n+1}

impl.sql.catalog uses impl.sql.execute.rts?
yes/no

source	target	source uses target?	Common Neighbours	Adamic Adar
catalog	impl.sql.execute	yes	1	3.09
catalog	impl.sql.execute.rts	no	0.25	3.75
impl.sql.catalog	catalog	yes	0	1.66
impl.sql.catalog	impl.sql.execute.rts	no	0.33	3.09
impl.sql.execute.rts	catalog	yes	0.33	3.09

Second Try: Apply Machine Learning Models

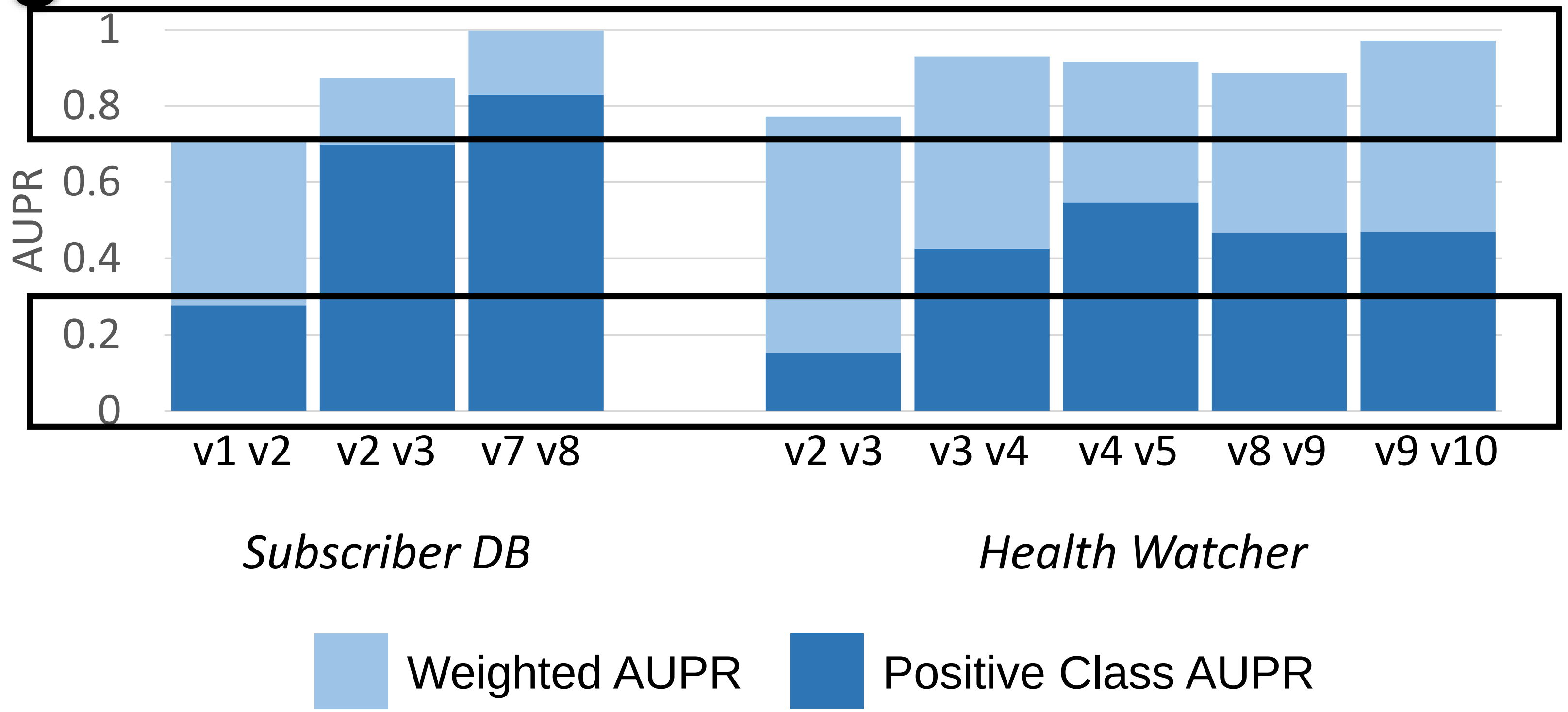
How did it go?



- The predictions were considered over selected versions.
  - The first item is the version for the training set.
  - The second one is the version for the test set

Second Try: Apply Machine Learning Models

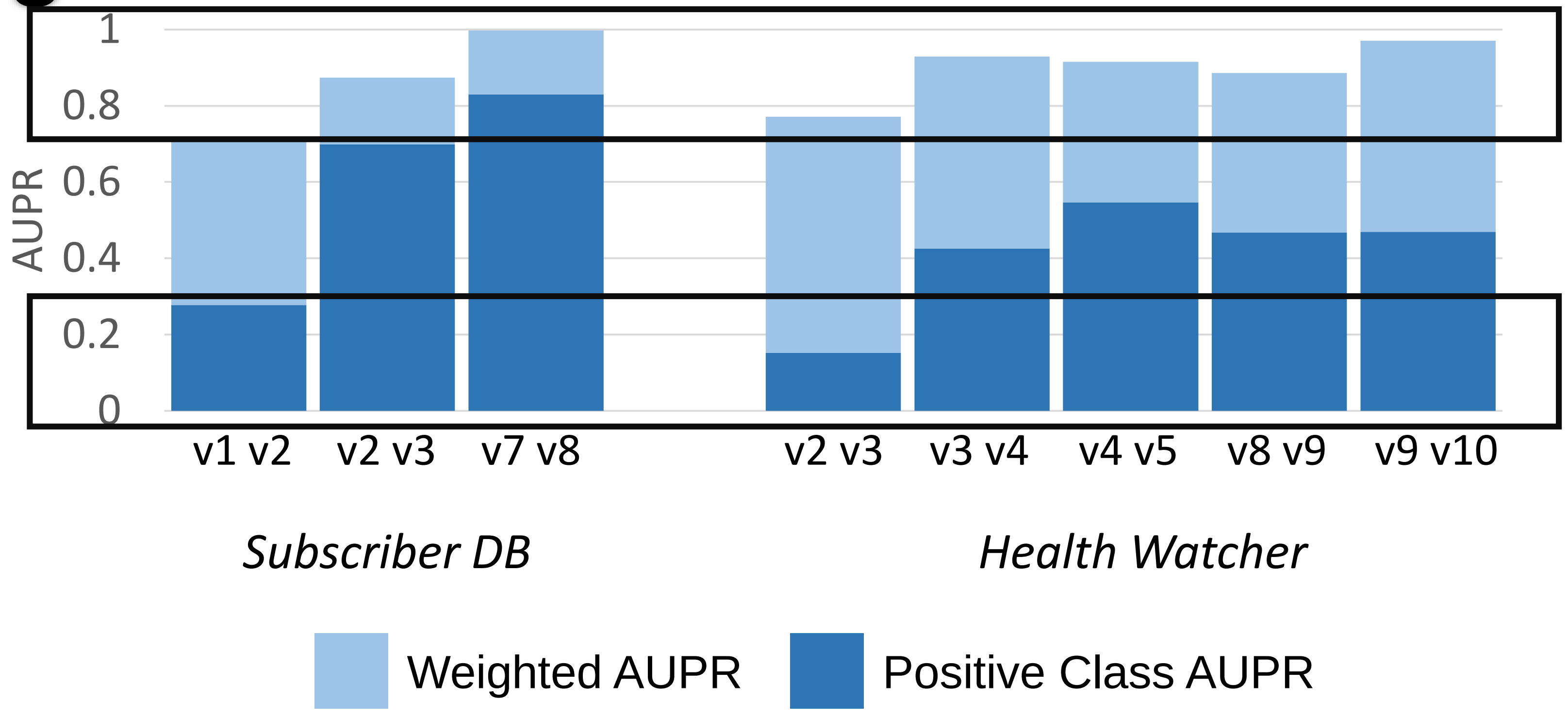
How did it go?



The classifier finds all new dependencies (high recall) but it also mistakenly reports non-existing dependencies (low precision)

Second Try: Apply Machine Learning Models

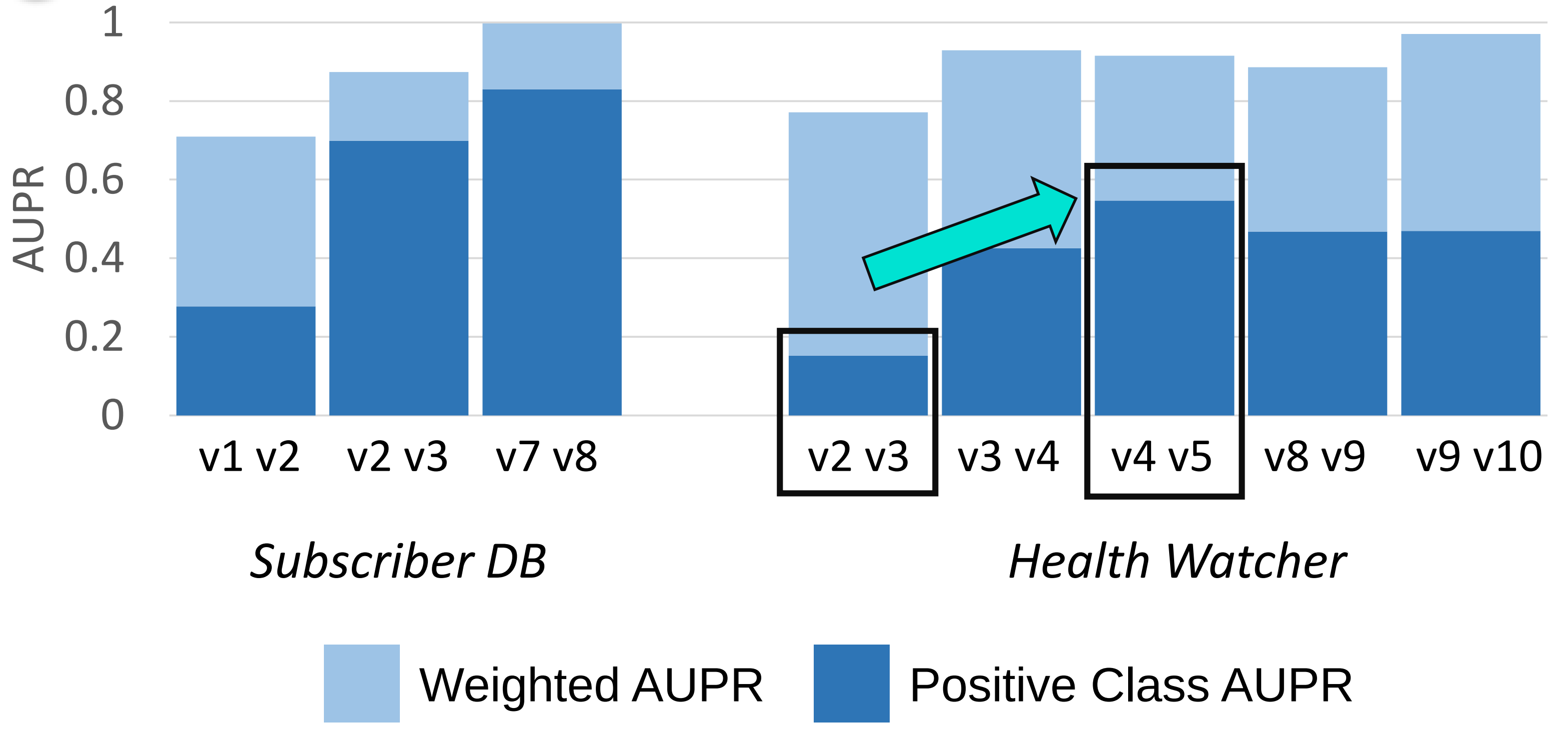
How did it go?



- Better values for the **weighted class** (both positive and negative instances).
  - Average precision values of 0.85 (SDB) and 0.96 (HW)
- However, **precision for the positive class was far from ideal**!
  - Average values of 0.74 (SDB) and 0.23 (HW)

Second Try: Apply Machine Learning Models

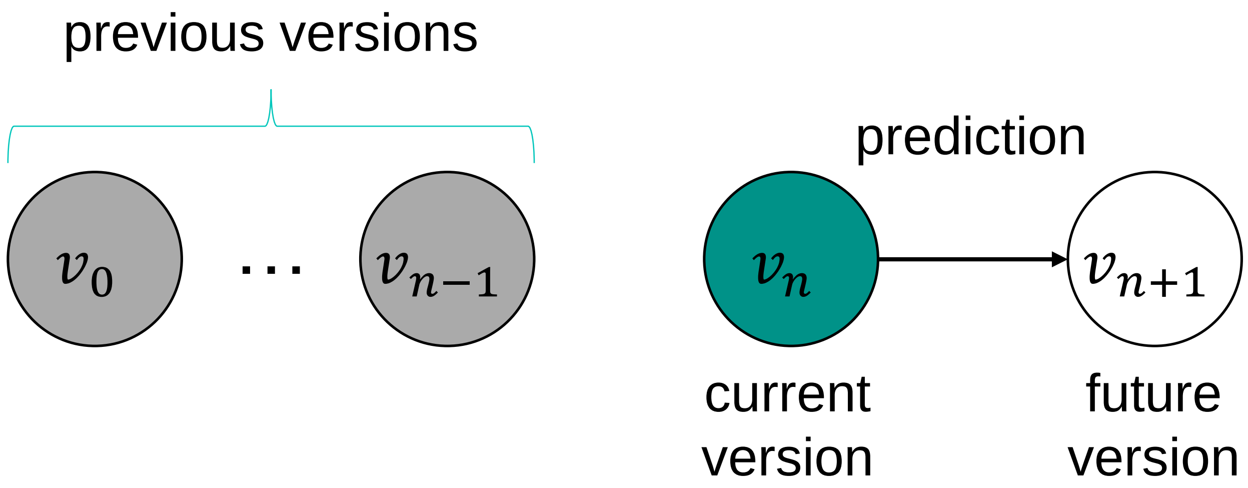
How did it go?



- Variations imply it might be difficult to differentiate between dependencies and non-dependencies due to similar structural characteristics.
    - **Need to consider additional information for having good predictions.**

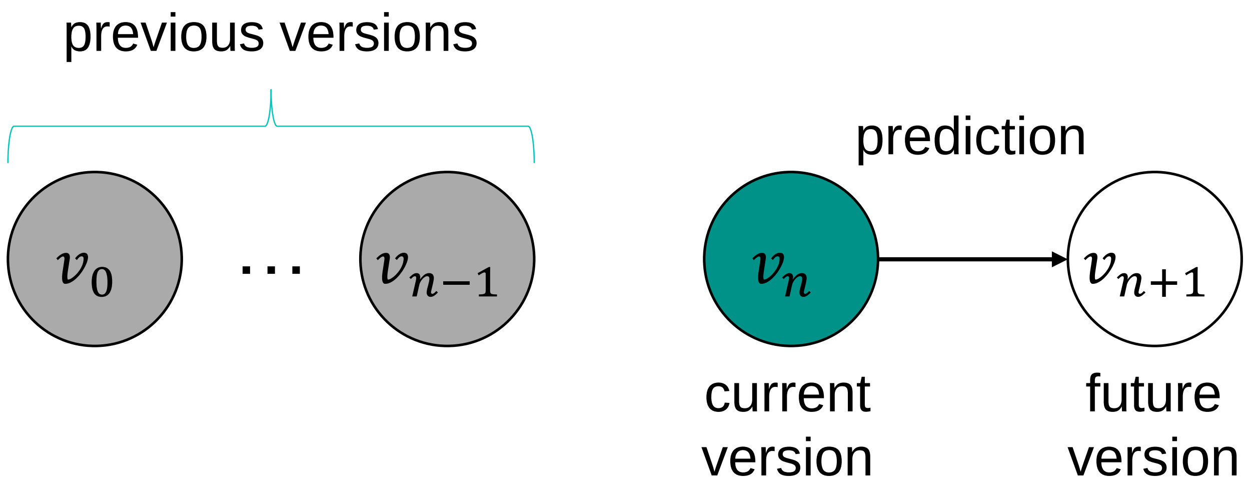
Third Try: Time Series Forecasting

To what extent Link Prediction can leverage on information from past versions to predict dependencies in the next version?



Third Try: Time Series Forecasting

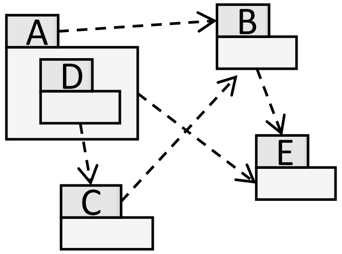
To what extent Link Prediction can leverage on information from past versions to predict dependencies in the next version?



Dynamic SNA (i.e., observations of the graph at different time periods) + topological features = Learn a robust ML model able to predict new links.

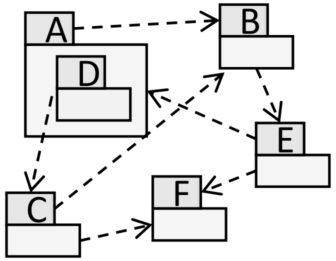
Third Try: Time Series Forecasting

dependency graph for v_{n-1}



source target	source uses target?	Common Neighbours
A – B	true	0.233
A – D	false	0.518
C – B	true	0.289
A – E	true	0.235
…	…	…
B – D	false	0.505

dependency graph for v_n



source target	source uses target?	Common Neighbours
A – B	true	0.353
A – D	false	0.618
C – B	true	0.389
A – E	true	0.385
…	…	…
B – D	false	0.605
E - F	true	0.171
C - F	true	0.1

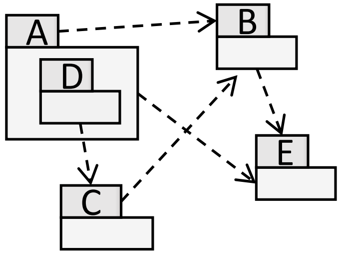
estimation for v_{n+1}

source target	source uses target?	Common Neighbours
A – B	?	0.453
A – D	?	0.718
C – B	?	0.289
A – E	?	0.685
…	…	…
B – D	?	0.805
E - F	?	0.171
C - F	?	0.11

We are **not yet predicting** new dependencies, but **estimating** the features' scores based on previous versions.

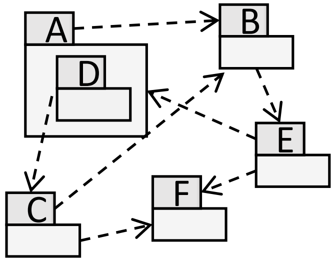
Third Try: Time Series Forecasting

dependency graph for v_{n-1}



source target	source uses target?	Common Neighbours
A – B	true	0.233
A – D	false	0.518
C – B	true	0.289
A – E	true	0.235
…	…	…
B – D	false	0.505

dependency graph for v_n



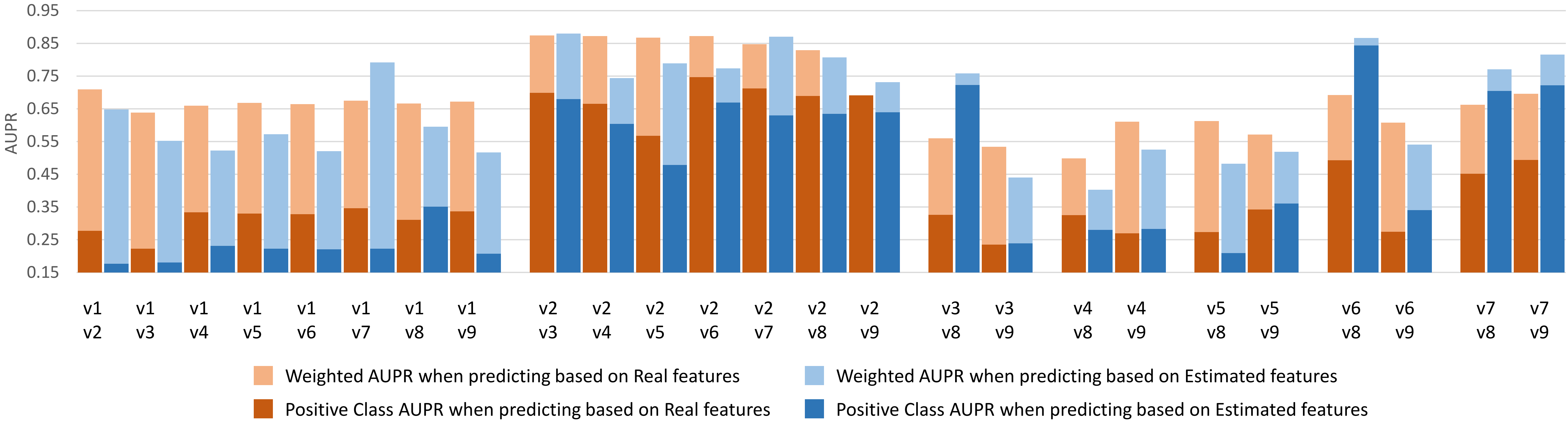
source target	source uses target?	Common Neighbours
A – B	true	0.353
A – D	false	0.618
C – B	true	0.389
A – E	true	0.385
…	…	…
B – D	false	0.605
E - F	true	0.171
C - F	true	0.1

estimation for v_{n+1}

source target	source uses target?	Common Neighbours
A – B	?	0.453
A – D	?	0.718
C – B	?	0.289
A – E	?	0.685
…	…	…
B – D	?	0.805
E - F	?	0.171
C - F	?	0.11

- Prediction is based on a classifier **trained** with the **last known version of the system**, v_n.
- The **test** set considers **the estimated feature** scores for v_{n+1}.

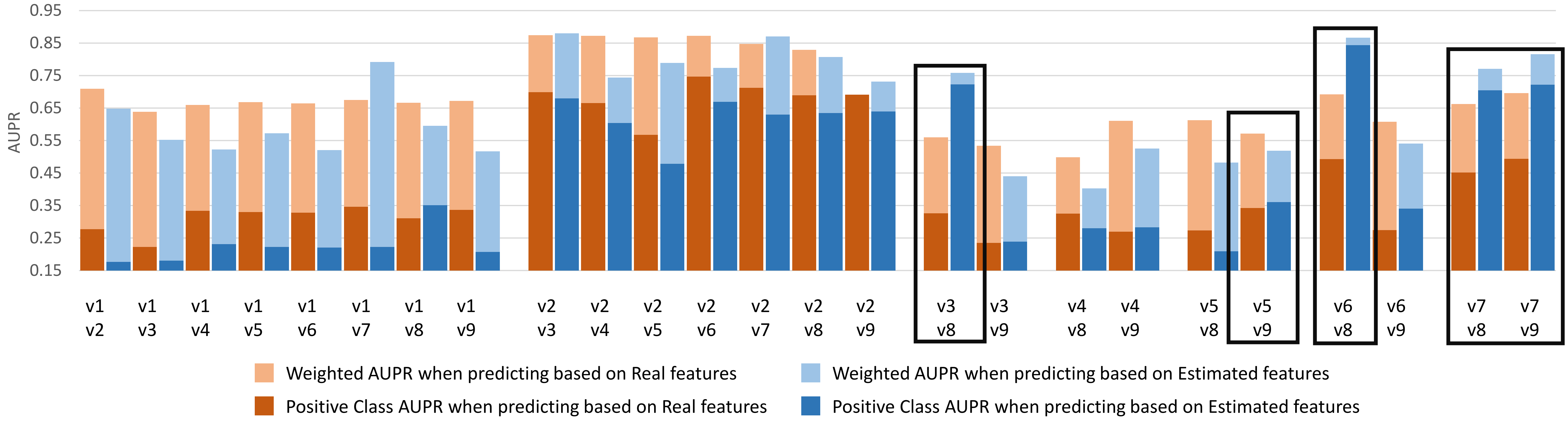
Third Try: Time Series Forecasting



- The versions represent the span for the estimations.
  - v1-v3 means that v1, v2 and v3 served to estimate the features for v4 (test set).

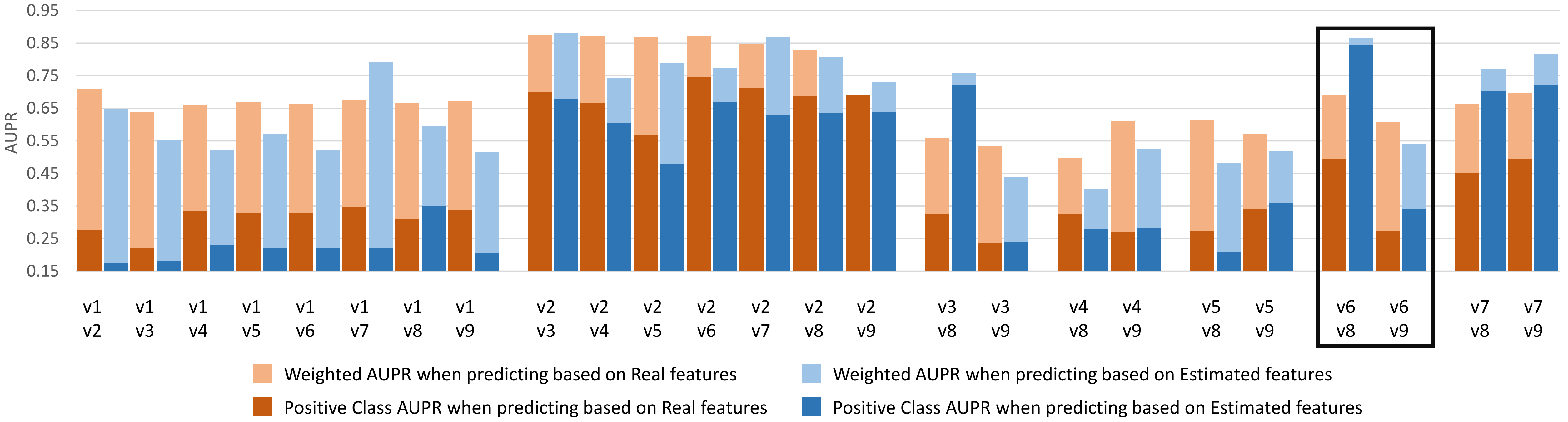
- Each pair represents the span of estimations, with **real** versus **estimated** features.

Third Try: Time Series Forecasting



- Better values for the **positive class!**
  - Average values of 0.84.
  - Estimated features are “better predictors” than real features.

Third Try: Time Series Forecasting



- The choice of versions for forecasting was relevant!
  - More versions sometimes decreases the quality of the predictions.

- This effect could be related to the structural changes in earch version.

Lessons Learned

What did we learn?

- We wanted to
  - Assess the LP performance in dependency graphs
  - Assess the kind of information required for having reasonable predictions.

- **Naïve LP techniques are not adequate for the task.**

- **Leveraging on information from previous versions gives reasonable predictions**, although not all versions seem useful.

Lessons Learned

What did we learn?

- We wanted to
  - Assess the LP performance in dependency graphs
  - Assess the kind of information required for having reasonable predictions.

- **Naïve LP techniques are not adequate for the task.**
- **Leveraging on information from previous versions gives reasonable predictions**, although not all versions seem useful.

Machine Learning techniques have the potential for Link Prediction applied to software dependencies

Lessons Learned

What do we do now?

- Despite the potential of LP techniques, further investigation is needed.
- A systematic study with more systems is required to corroborate our initial findings.
- The features currently used can be extended.

Lessons Learned

What do we do now?

- Despite the potential of LP techniques, further investigation is needed.
- A systematic study with more systems is required to corroborate our initial findings.
- The features currently used can be extended.

Develop **customized LP algorithms for dependency-related problems**

(e.g., layering violations, cyles, hub-like configurations)

Table of Contents

Dependency-based Smells

- An architectural **bad smell** is a commonly used set of **architectural design decisions** that **negatively** impacts system lifecycle properties.
  - E.g. understandability, testability, extensibility, and reusability.

Dependency-based Smells

- An architectural **bad smell** is a commonly used set of **architectural design decisions** that **negatively** impacts system lifecycle properties.
  - E.g. understandability, testability, extensibility, and reusability.

- **Dependency-based smells** involve **interactions** amongst system components.
  - Occur when one or more components **violate design principles** or rules.
  - Often manifest themselves as **undesired dependencies** in the source code.

Dependency-based Smells

- An architectural **bad smell** is a commonly used set of **architectural design decisions** that **negatively** impacts system lifecycle properties.
  - E.g. understandability, testability, extensibility, and reusability.

- **Dependency-based smells** involve **interactions** amongst system components.
  - Occur when one or more components **violate design principles** or rules.
  - Often manifest themselves as **undesired dependencies** in the source code.

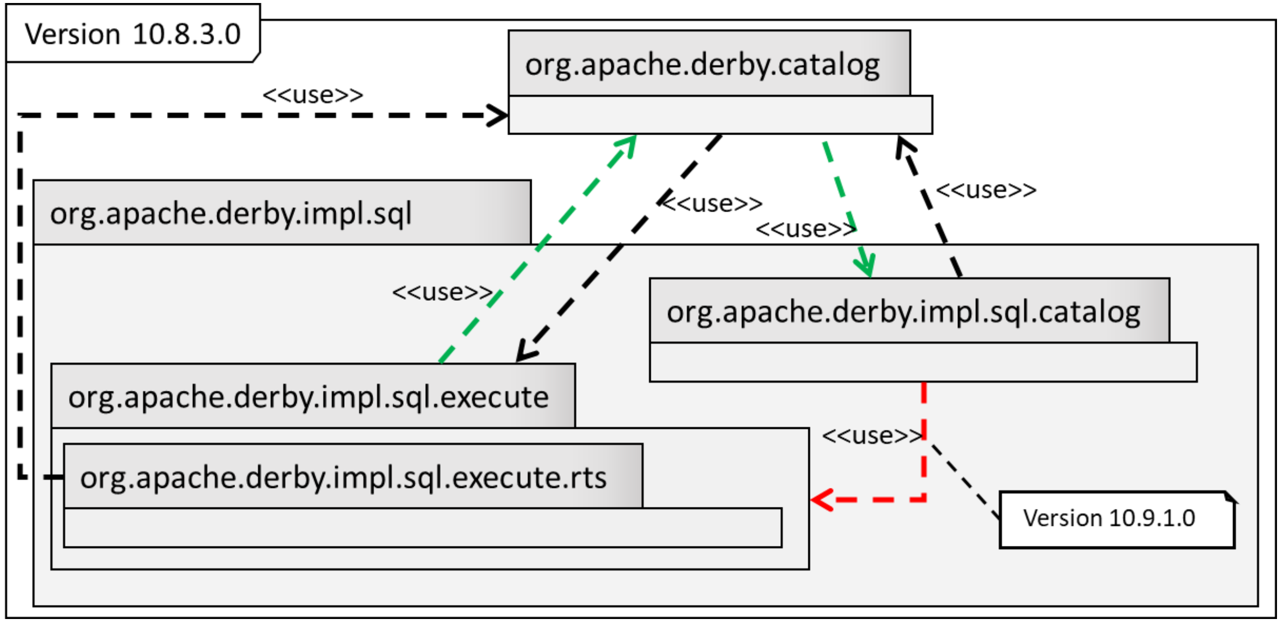
Cyclic Dependencies

Hub Like Dependencies

Dependency-based Smells

Cyclic Dependencies

- Various components directly or indirectly depend on each other to function properly.
- A case of an *undesired dependency*.
  - Breaks the desirable acyclic nature of a subsystem's dependency structure.



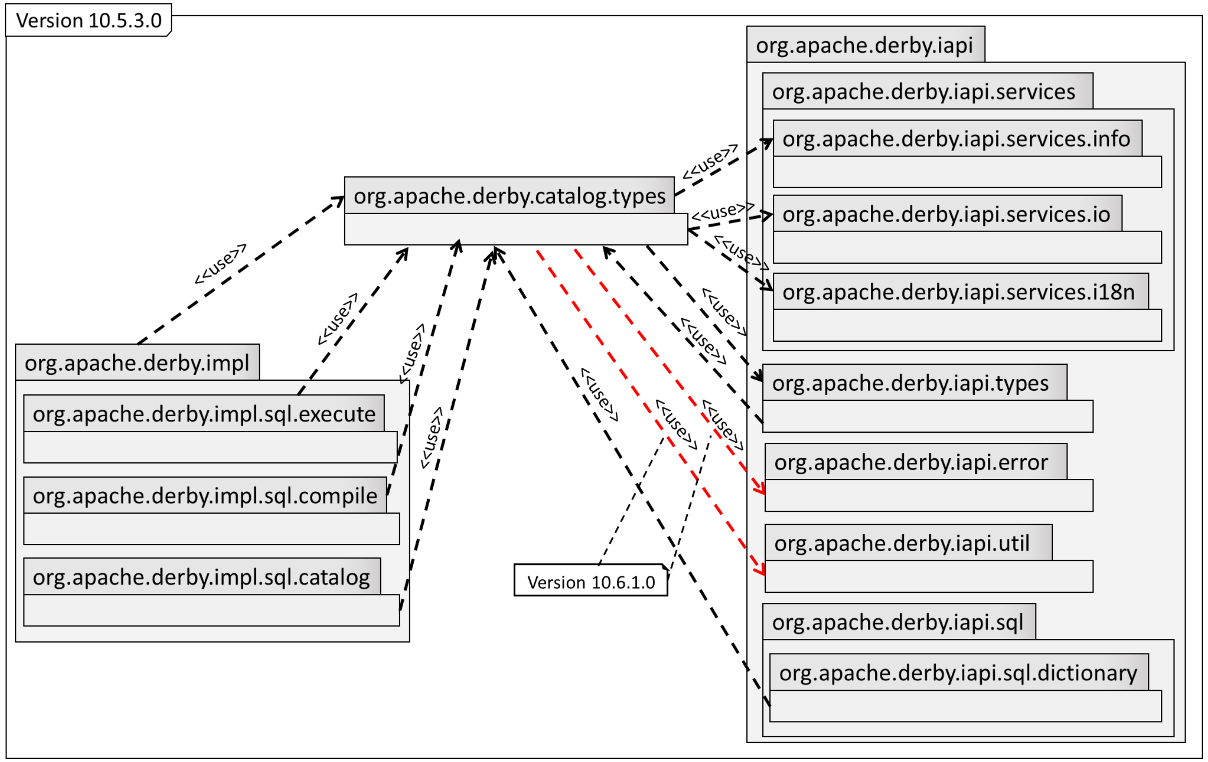
- Components involved in a cycle can be **hard** to **maintain**, **test** or **reuse** in isolation.
- Cycles might have different shapes.
  - **Different harmful levels for the system health than others.**

Dependency-based Smells

Hub-like Dependencies

- A component has outgoing and ingoing dependencies with a large number of other components.

- Detecting hubs:
  1. Computes the median of the number of incoming and outgoing dependencies of all packages.
  2. For each package: Are both its incoming and outgoing dependencies greater than the incoming and outgoing medians?
  3. incoming - outgoing dependencies < than a fraction of the total dependencies of that package.



Dependency-based Smells

Once again, we resort to Machine Learning!

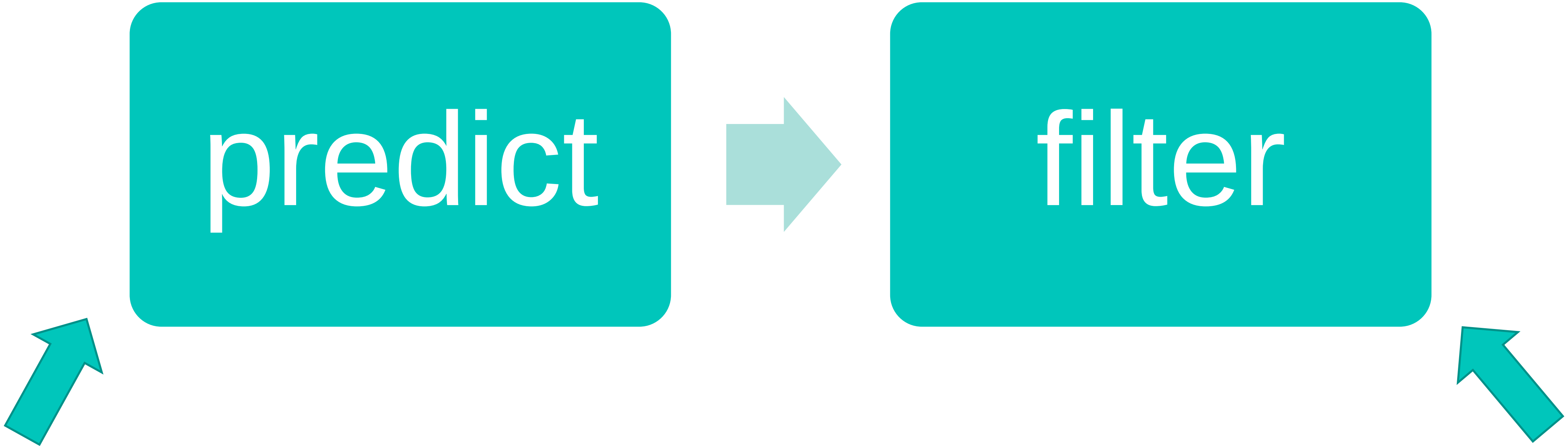
Dependency-based Smells

Once again, we resort to Machine Learning!

predict → filter

Dependency-based Smells

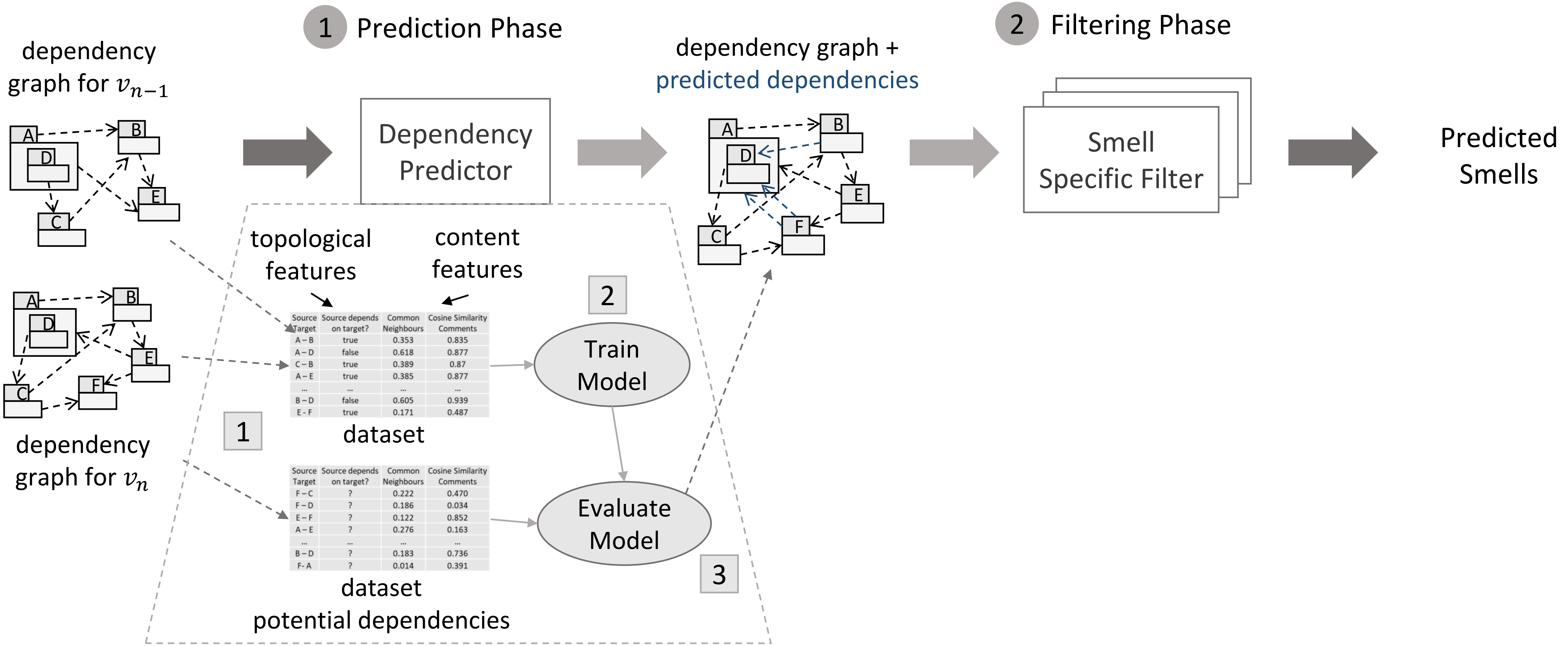
Once again, we resort to Machine Learning!



predict the appearance of new dependencies in the next system version

filter the predicted dependencies according to the characteristics of specific types of smells.

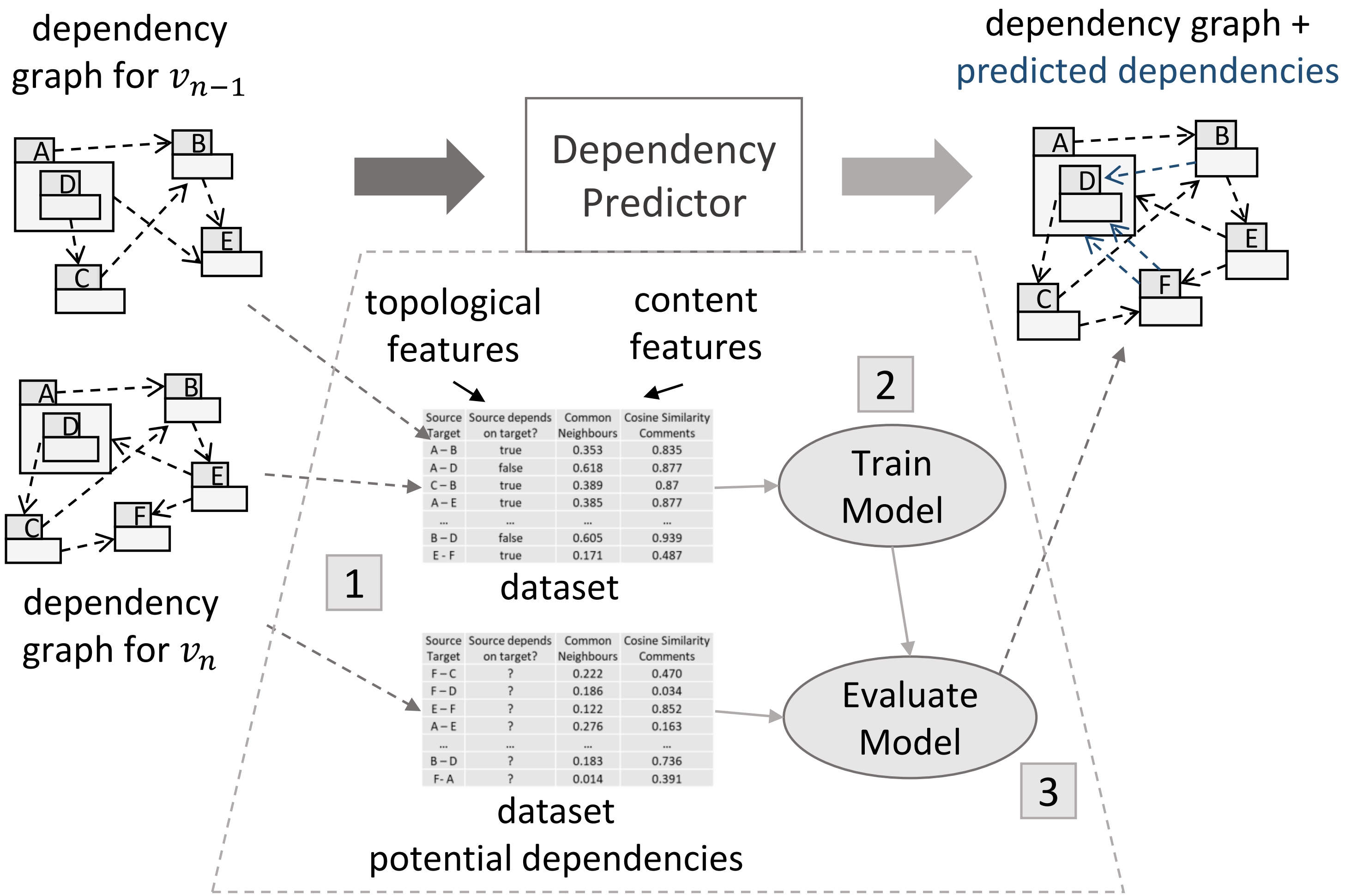
Dependency-based Smells



Source Target	Source depends on target?	Common Neighbours	Cosine Similarity Comments
A – B	true	0.353	0.835
A – D	false	0.618	0.877
C – B	true	0.389	0.87
A – E	true	0.385	0.877
...	...	...	...
B – D	false	0.605	0.939
E - F	true	0.171	0.487

Source Target	Source depends on target?	Common Neighbours	Cosine Similarity Comments
F – C	?	0.222	0.470
F – D	?	0.186	0.034
E – F	?	0.122	0.852
A – E	?	0.276	0.163
...	...	...	...
B – D	?	0.183	0.736
F- A	?	0.014	0.391

Prediction Phase

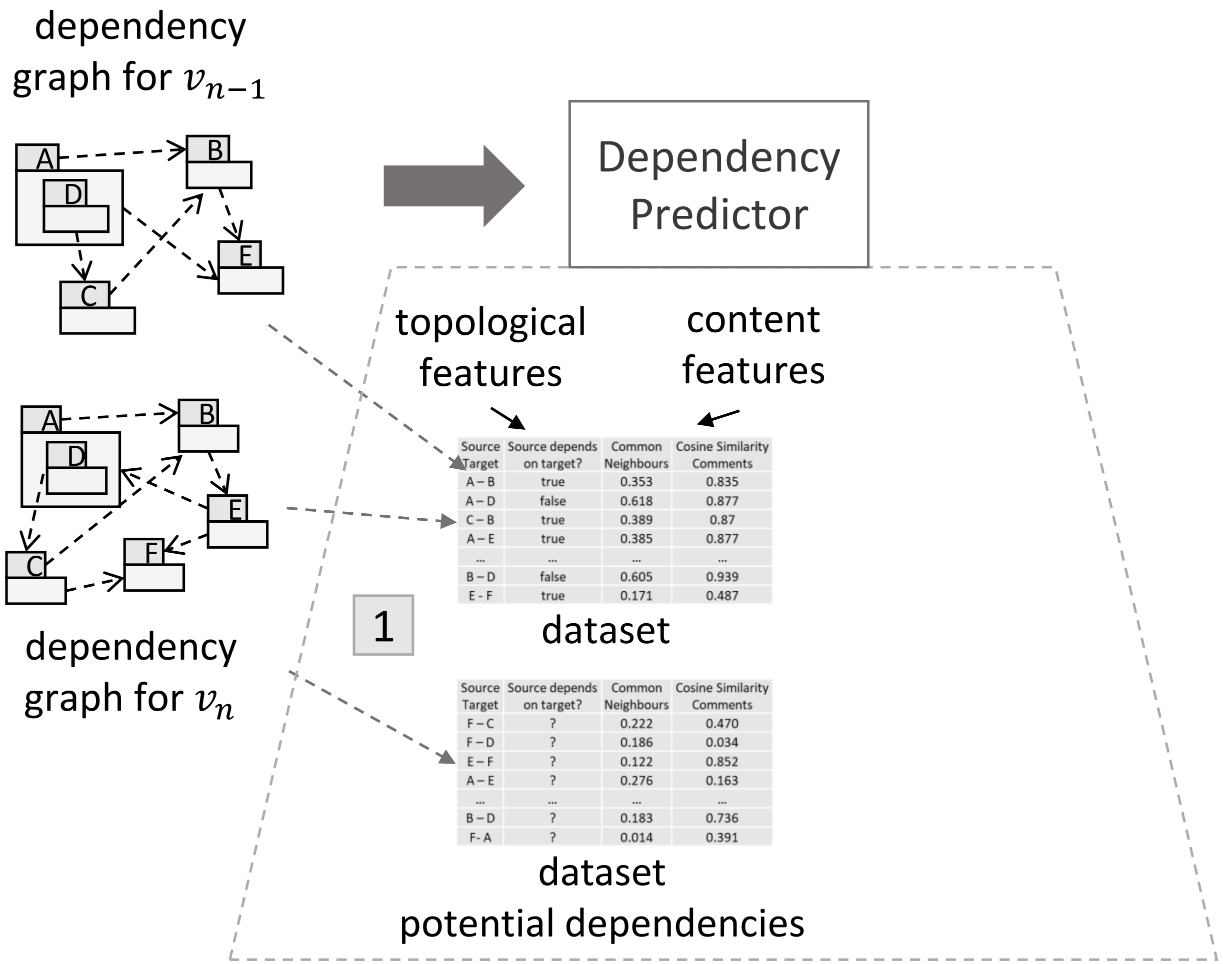


Source Target	Source depends on target?	Common Neighbours	Cosine Similarity Comments
A – B	true	0.353	0.835
A – D	false	0.618	0.877
C – B	true	0.389	0.87
A – E	true	0.385	0.877
...	...	...	...
B – D	false	0.605	0.939
E - F	true	0.171	0.487

Source Target	Source depends on target?	Common Neighbours	Cosine Similarity Comments
F – C	?	0.222	0.470
F – D	?	0.186	0.034
E – F	?	0.122	0.852
A – E	?	0.276	0.163
...	...	...	...
B – D	?	0.183	0.736
F- A	?	0.014	0.391

- Individual dependencies are inferred based on training a binary classification model.
- Dependency graphs of the current (v_n) and previous versions (v_{n-1}) are used as inputs.
- The output is the set of dependencies that are likely to appear in the next system version v_{n+1}.
- This phase is **smell independent**.
  - Only identifies dependencies that might prefigure different smells in the second phase.

Prediction Phase



Source Target	Source depends on target?	Common Neighbours	Cosine Similarity Comments
A – B	true	0.353	0.835
A – D	false	0.618	0.877
C – B	true	0.389	0.87
A – E	true	0.385	0.877
…	…	…	…
B – D	false	0.605	0.939
E - F	true	0.171	0.487

dataset

Source Target	Source depends on target?	Common Neighbours	Cosine Similarity Comments
F – C	?	0.222	0.470
F – D	?	0.186	0.034
E – F	?	0.122	0.852
A – E	?	0.276	0.163
…	…	…	…
B – D	?	0.183	0.736
F- A	?	0.014	0.391

dataset
potential dependencies

The prediction phase internally involves **3** steps.

Step 1

- The instance-based representation are constructed, based on both topological and content-based features.
  - Existing dependencies → positive class.
  - Missing dependencies → negative class.
- The training set includes:
  - Existing dependencies in v_{n-1}.
  - Missing dependencies in v_{n-1}.
  - Existing dependencies in v_n.

Prediction Phase

- **<u>Content-based features</u>** are an **alternative** (and **complementary**) similarity criterion to topological features.

- Natural language processing routines are used to transform texts into their **bag-of-words representations** by considering different aspects of the original texts.
    - Restricted to only the appearing nouns, adjective or verbs…
    - Remove punctuation…

Dependency-based Smells

Prediction Phase

- <u>**Content-based features**</u> are an **alternative** (and **complementary**) similarity criterion to topological features.

- Natural language processing routines are used to transform texts into their **bag-of-words representations** by considering different aspects of the original texts.
  - Restricted to only the appearing nouns, adjective or verbs…
  - Remove punctuation…

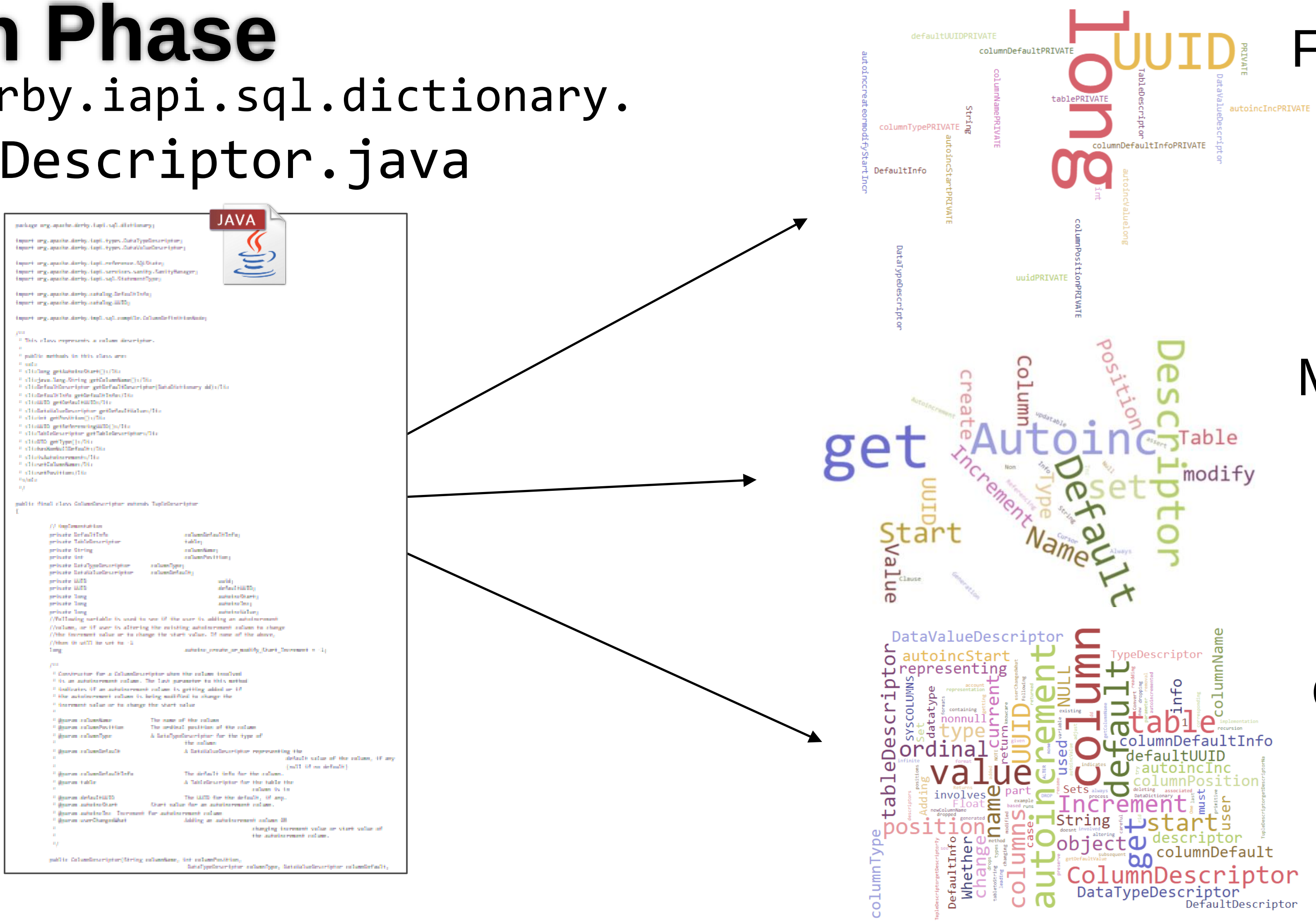
- The bag-of-words class representations can be used to assess the similarity amongst the classes.
  - Cosine similarity is commonly used.

- Each Java class c as a bag-of-words containing the most representative tokens that characterize its source code.
  - Either considering the name of the field attributes of the classes, the name of the declared methods or the class comments and documentation.

Dependency-based Smells

Prediction Phase

org.apache.derby.iapi.sql.dictionary.
ColumnDescriptor.java



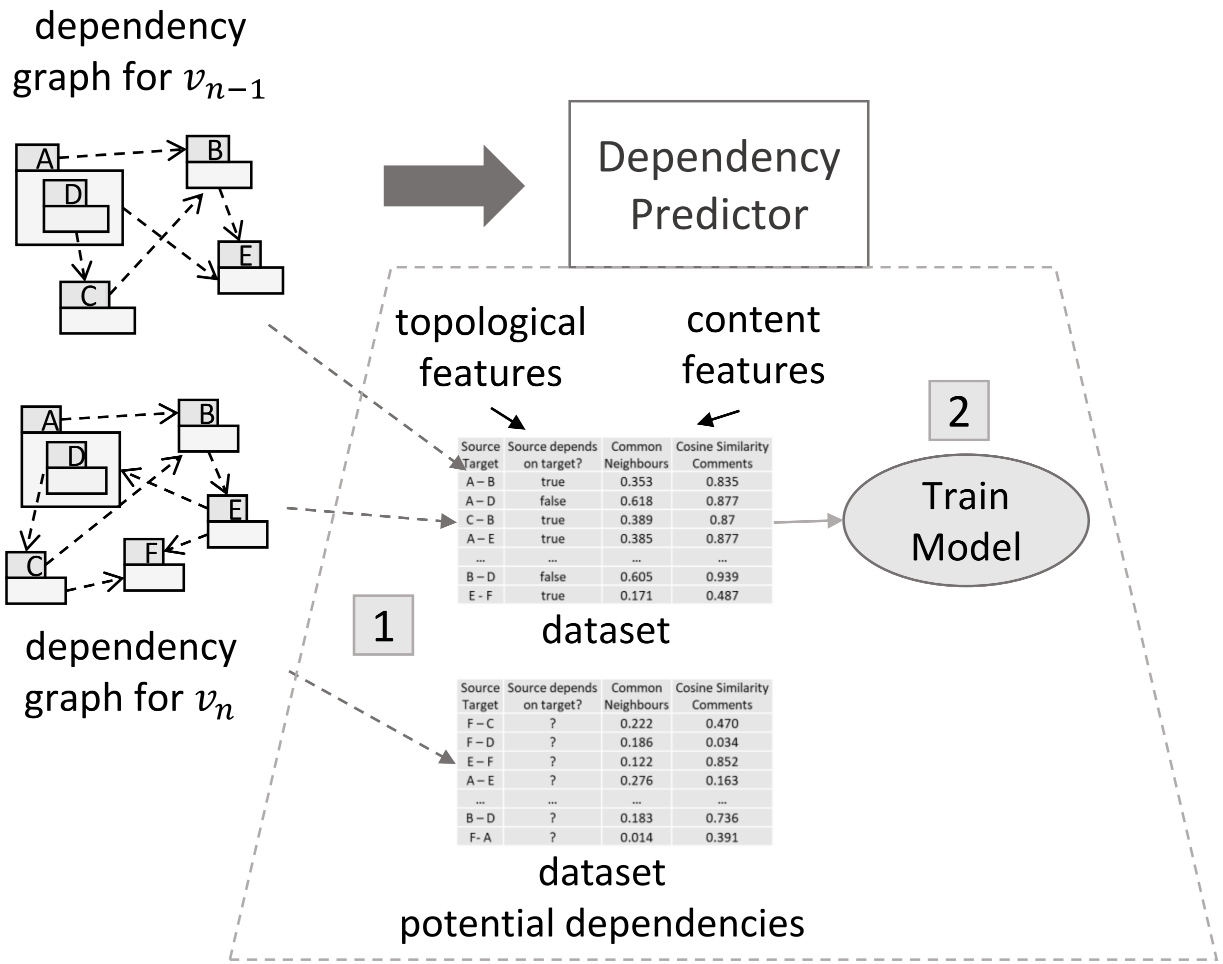
Fields

Method Names

Comments

Dependency-based Smells

Prediction Phase



Source Target	Source depends on target?	Common Neighbours	Cosine Similarity Comments
A – B	true	0.353	0.835
A – D	false	0.618	0.877
C – B	true	0.389	0.87
A – E	true	0.385	0.877
...	...	...	...
B – D	false	0.605	0.939
E - F	true	0.171	0.487

Source Target	Source depends on target?	Common Neighbours	Cosine Similarity Comments
F – C	?	0.222	0.470
F – D	?	0.186	0.034
E – F	?	0.122	0.852
A – E	?	0.276	0.163
...	...	...	...
B – D	?	0.183	0.736
F- A	?	0.014	0.391

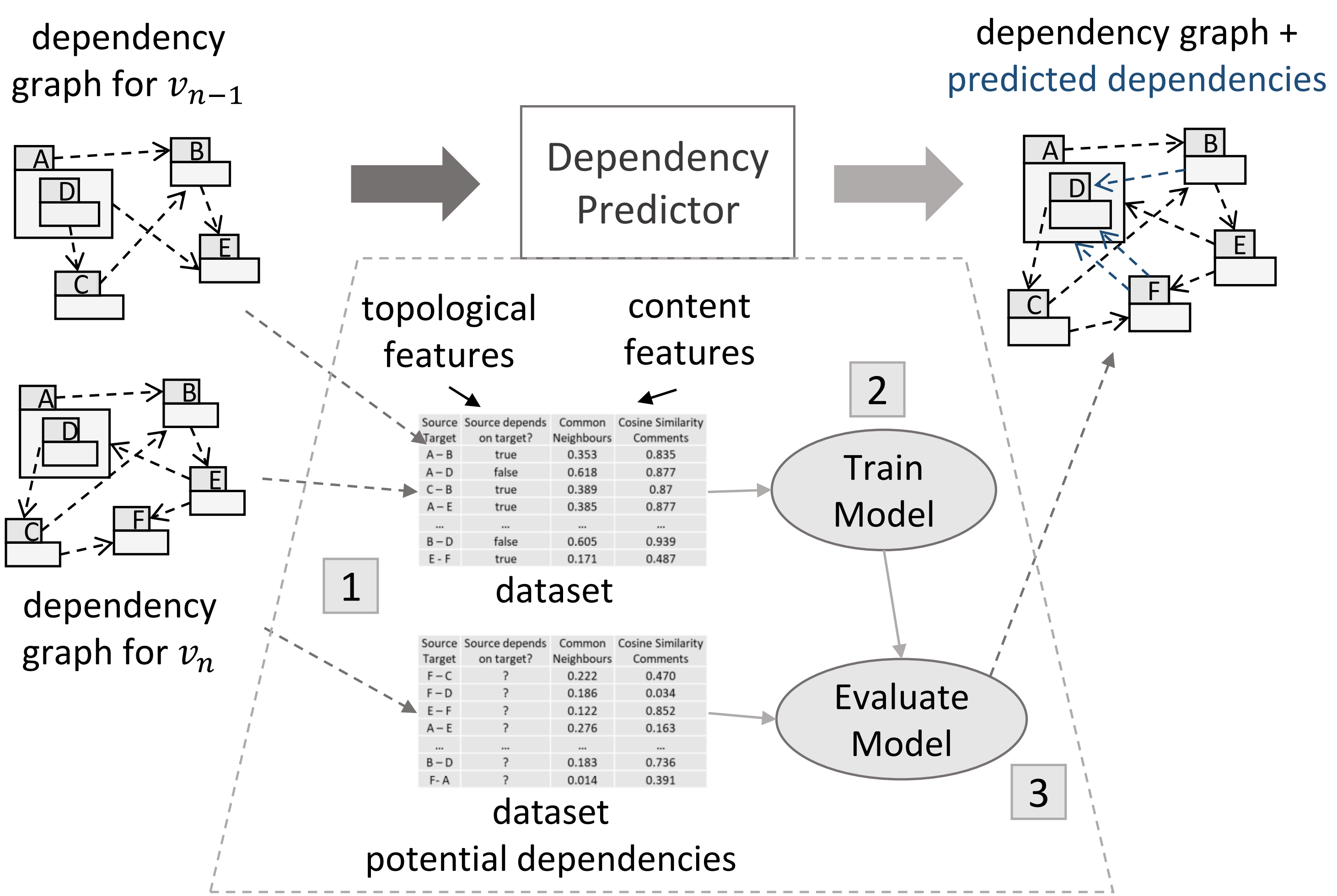
The prediction phase internally involves **3** steps.

Step 2

- The classification model is built.
- The classifier is trained for properly learning instances of both the positive and negative classes.
  - Includes information of dependencies in v_{n-1} being guaranteed that are not going to appear in v_n.

Dependency-based Smells

Prediction Phase



The prediction phase internally involves **3** steps.

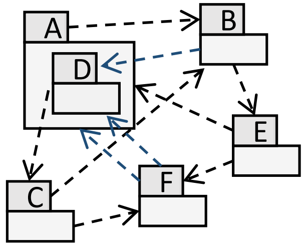
Step 3

- Dependencies are predicted.
- Only potential dependencies considering the packages already existing in v_n are considered.

Dependency-based Smells

Filtering Phase

dependency graph +
predicted dependencies



Smell
Specific Filter

Predicted
Smells

- The prediction of a dependency is not enough to predict the appearance of an architectural smell.
  - **Not every predicted dependency might cause an smell to emerge.**

- Predicted dependencies undergo a filtering process.
  - Filters are **smell-dependent**.

Filtering Phase - Cycles

- Considers only predicted dependencies that would lead to the closure of new cycles in v_{n+1}.

Two variants:

- *All predicted dependencies are simultaneously considered.*
  - Allows to detect cycles needed more than one dependency to be closed.
- *Dependencies are individually analysed.*
  - Allows to detect cycles needed only one dependency to be closed.

Dependency-based Smells

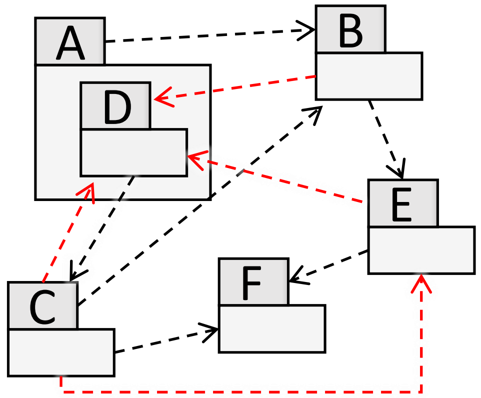
Filtering Phase - Cycles

- Considers only predicted dependencies that would lead to the closure of new cycles in v_{n+1}.

Two variants:

- *All predicted dependencies are simultaneously considered.*
  - Allows to detect cycles needed more than one dependency to be closed.

- *Dependencies are individually analysed.*
  - Allows to detect cycles needed only one dependency to be closed.



Dependency-based Smells

Filtering Phase - Cycles

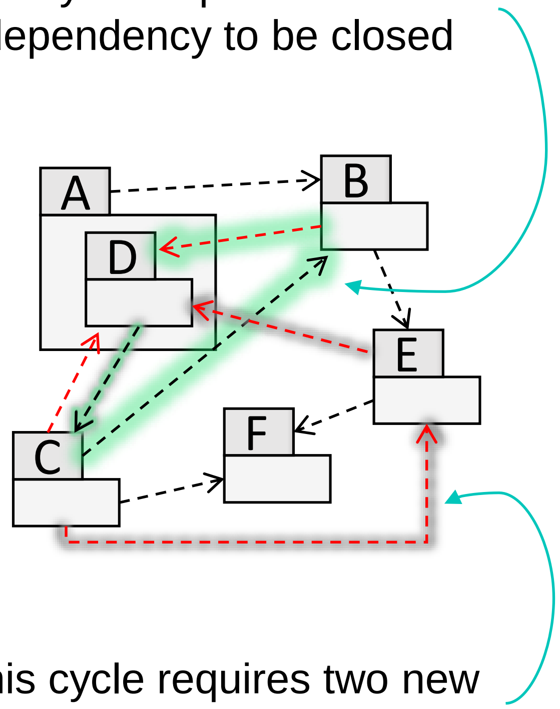
- Considers only predicted dependencies that would lead to the closure of new cycles in v_{n+1}.

Two variants:

- *All predicted dependencies are simultaneously considered.*
  - Allows to detect cycles needed more than one dependency to be closed.

- *Dependencies are individually analysed.*
  - Allows to detect cycles needed only one dependency to be closed.

this cycle requires one new dependency to be closed



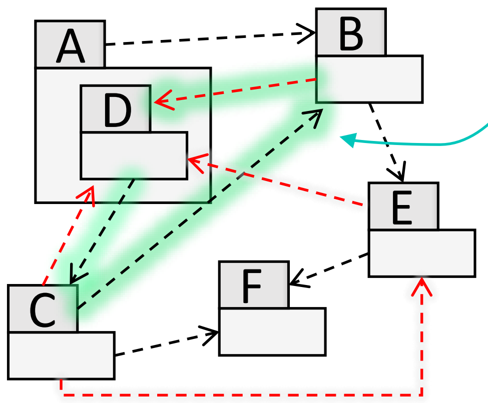
this cycle requires two new dependencies to be closed

Dependency-based Smells

Filtering Phase - Cycles

this cycle requires one new dependency to be closed

- Considers only predicted dependencies that would lead to the closure of new cycles in v_{n+1}.



Two variants:

- *All predicted dependencies are simultaneously considered.*
  - Allows to detect cycles needed more than one dependency to be closed.

the cycle requiring two dependencies is not going to be found

- *Dependencies are individually analysed.*
  - Allows to detect cycles needed only one dependency to be closed.

Filtering Phase - Hubs

- Only the nodes incidental to the predicted edges that fit with the hub definition are actually predicted.
  - Allow the detection of those nodes becoming hubs due to the addition of new dependencies.
  - Disregard nodes that might become hubs due to changes in the overall structure of the dependency graph.

Three variants:

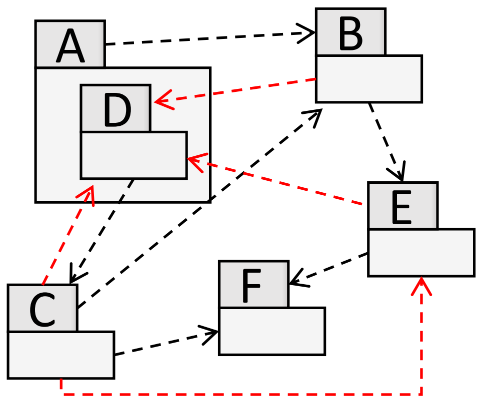
- *Dependencies are individually analysed.*
- *Dependencies are grouped per node.*
- *All predicted dependencies are simultaneously considered.*

Filtering Phase - Hubs

- Only the nodes incidental to the predicted edges that fit with the hub definition are actually predicted.
  - Allow the detection of those nodes becoming hubs due to the addition of new dependencies.
  - Disregard nodes that might become hubs due to changes in the overall structure of the dependency graph.

Three variants:

- *Dependencies are individually analysed.*
- *Dependencies are grouped per node.*
- *All predicted dependencies are simultaneously considered.*

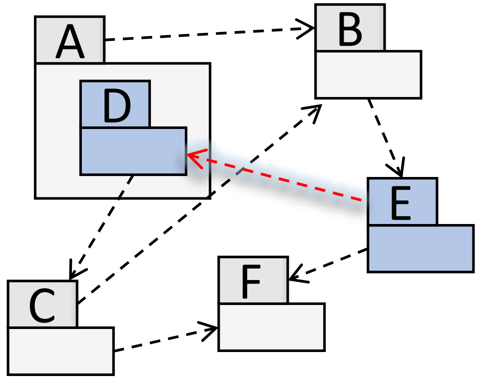


Filtering Phase - Hubs

- Only the nodes incidental to the predicted edges that fit with the hub definition are actually predicted.
  - Allow the detection of those nodes becoming hubs due to the addition of new dependencies.
  - Disregard nodes that might become hubs due to changes in the overall structure of the dependency graph.

Three variants:

- *Dependencies are individually analysed.*
- *Dependencies are grouped per node.*
- *All predicted dependencies are simultaneously considered.*

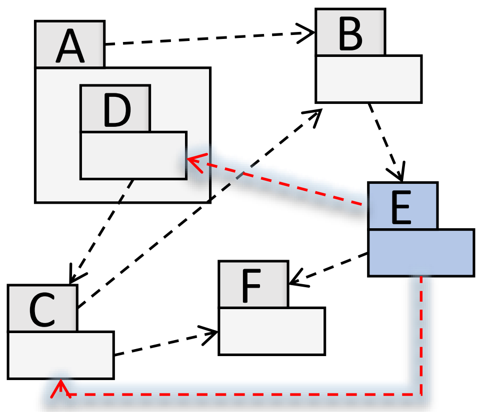


Filtering Phase - Hubs

- Only the nodes incidental to the predicted edges that fit with the hub definition are actually predicted.
  - Allow the detection of those nodes becoming hubs due to the addition of new dependencies.
  - Disregard nodes that might become hubs due to changes in the overall structure of the dependency graph.

Three variants:

- *Dependencies are individually analysed.*
- *Dependencies are grouped per node.*
- *All predicted dependencies are simultaneously considered.*

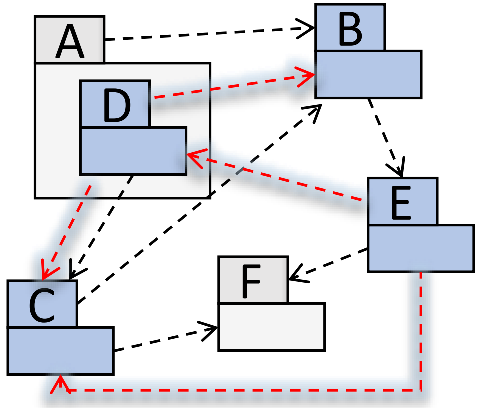


Filtering Phase - Hubs

- Only the nodes incidental to the predicted edges that fit with the hub definition are actually predicted.
  - Allow the detection of those nodes becoming hubs due to the addition of new dependencies.
  - Disregard nodes that might become hubs due to changes in the overall structure of the dependency graph.

Three variants:

- *Dependencies are individually analysed.*
- *Dependencies are grouped per node.*
- *All predicted dependencies are simultaneously considered.*



Dependency-based Smells

Study Settings

Two medium Java systems.

- **Apache Derby**
  - 14 versions.
  - 40 KLOC.
- Apache Ant
  - 18 versions.
  - 60 KLOC.

	#c	#p	#deps	#cycles	cycle length	#hubs	hub degree
derby 10.5.1.1	1344	96	767 +10	234 +4	11.59	28 +2	27.67
derby 10.5.3.0	1344	96	768 +1	234	11.59	28	27.71
derby 10.6.1.0	1387	98	804 +36	254 +6	12.99	29 +3	28.34
derby 10.6.2.1	1387	98	805 +1	255 +1	13.02	29	28.34
derby 10.7.1.1	1389	98	807 +4,-2	257 +2	12.98	29	28.44
derby 10.8.1.2	1395	97	837 +31,-1	305 +22	15.17	30 +1	30.03
derby 10.8.3.0	1395	96	841 +3	306 +1	15.13	30	30.06
derby 10.9.1.0	1406	96	851 +20,-10	280 +5	13.43	29 +1	30.62
derby 10.10.1.1	1453	100	938 +89,-2	291 +10	13.32	29 -1	32.89

Dependency-based Smells

Study Settings

Two medium Java systems.

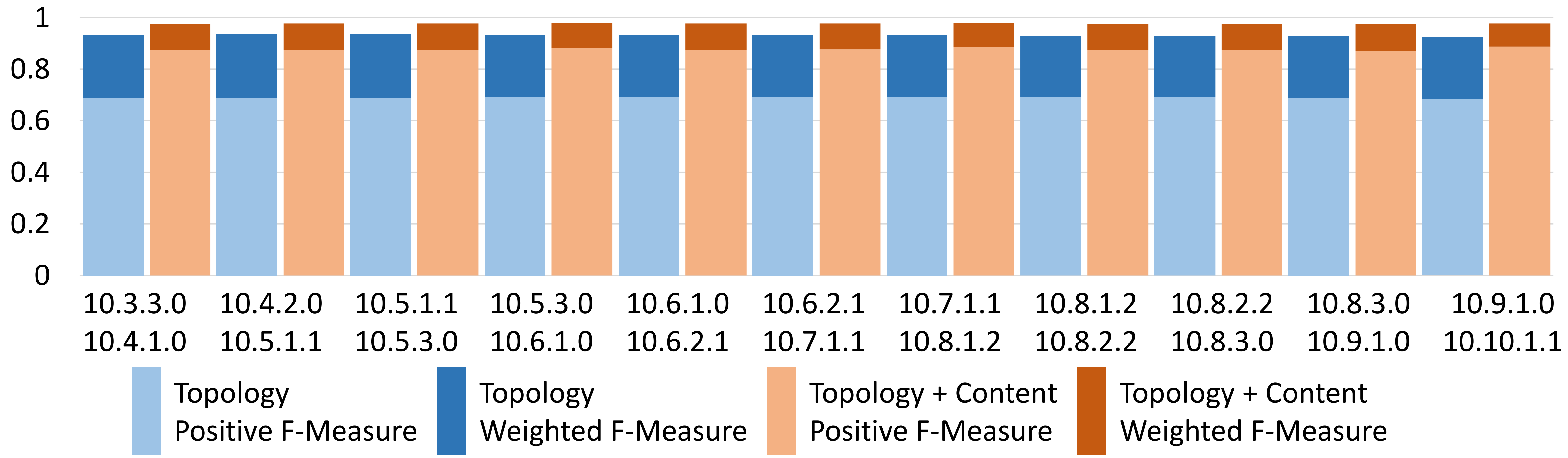
- Apache Derby
  - 14 versions.
  - 40 KLOC.

- **Apache Ant**
  - 18 versions.
  - 60 KLOC.

	#c	#p	#deps	#cycles	cycle length	#hubs	hub degree
ant 1.6.0	352	24	90 +20,-1	30 +1	3.73	9 +2	14.22
ant 1.6.2	369	24	92 +2	43 +2	4.12	9	14.67
ant 1.6.3	380	25	97 +5	43 +1	4.7	9	15.33
ant 1.7.1	502	29	137 +46,-6	63 +5	5.1	12 +1	17.42

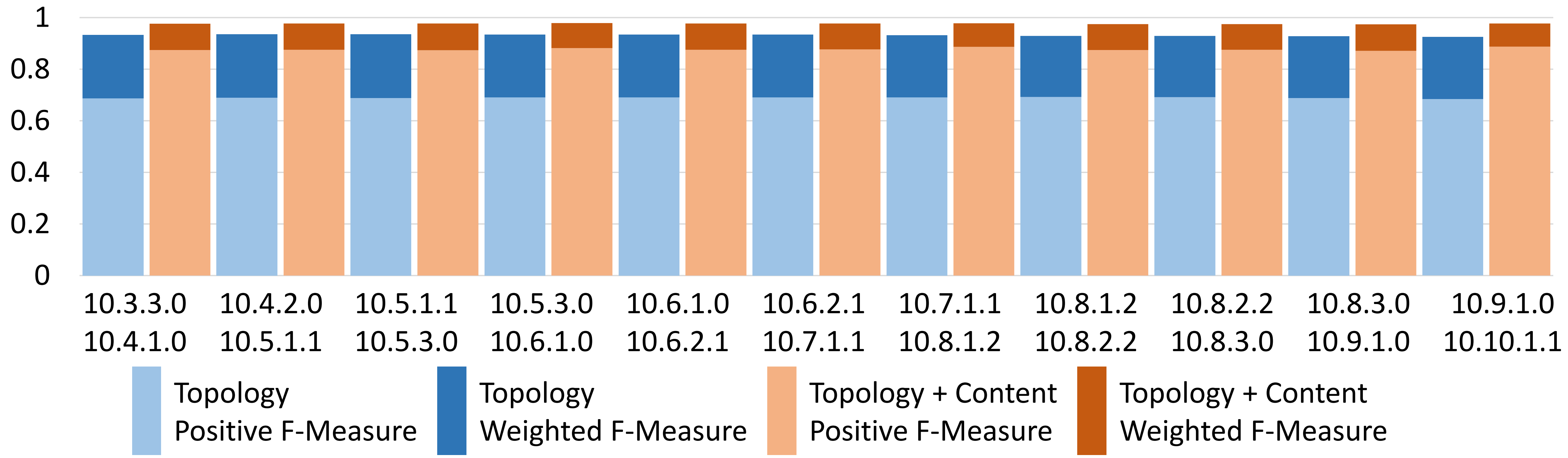
Dependency-based Smells

How did it go? – Prediction Phase



- Compares results of considering either topological or topological + content features.
- Results are presented for those sets of versions in which new dependencies between already existing packages were added.

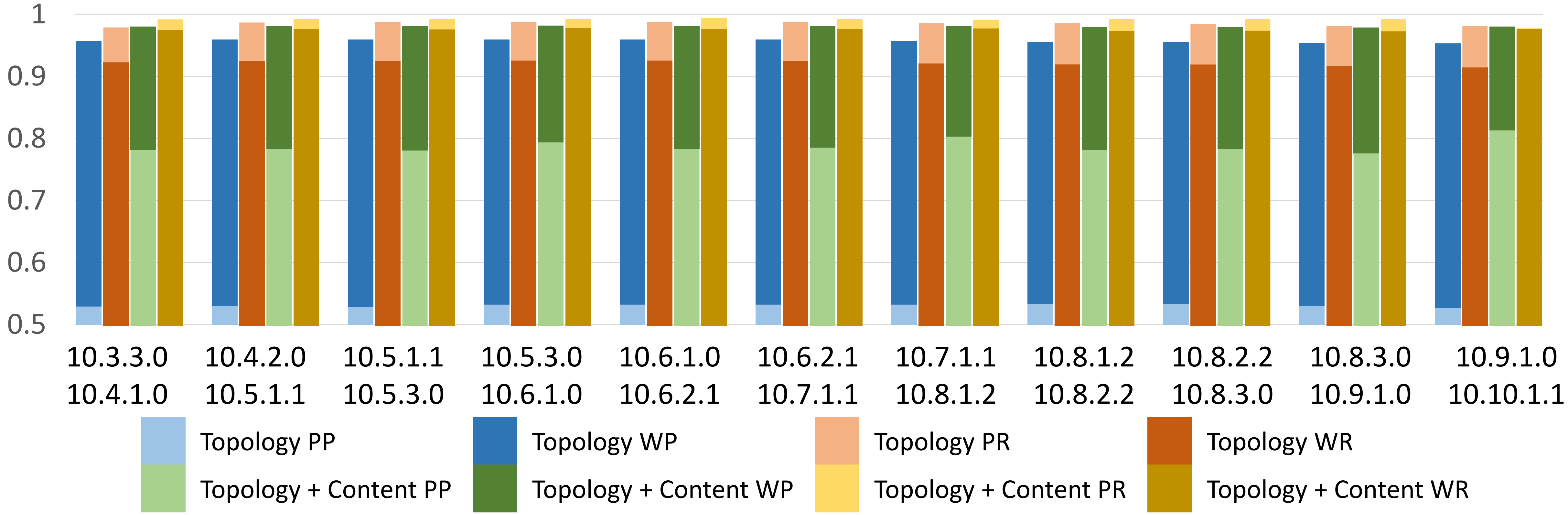
How did it go? – Prediction Phase



- Adding content-based features increased the quality of the predicted dependencies.
  - Average improvements of 27%.

Dependency-based Smells

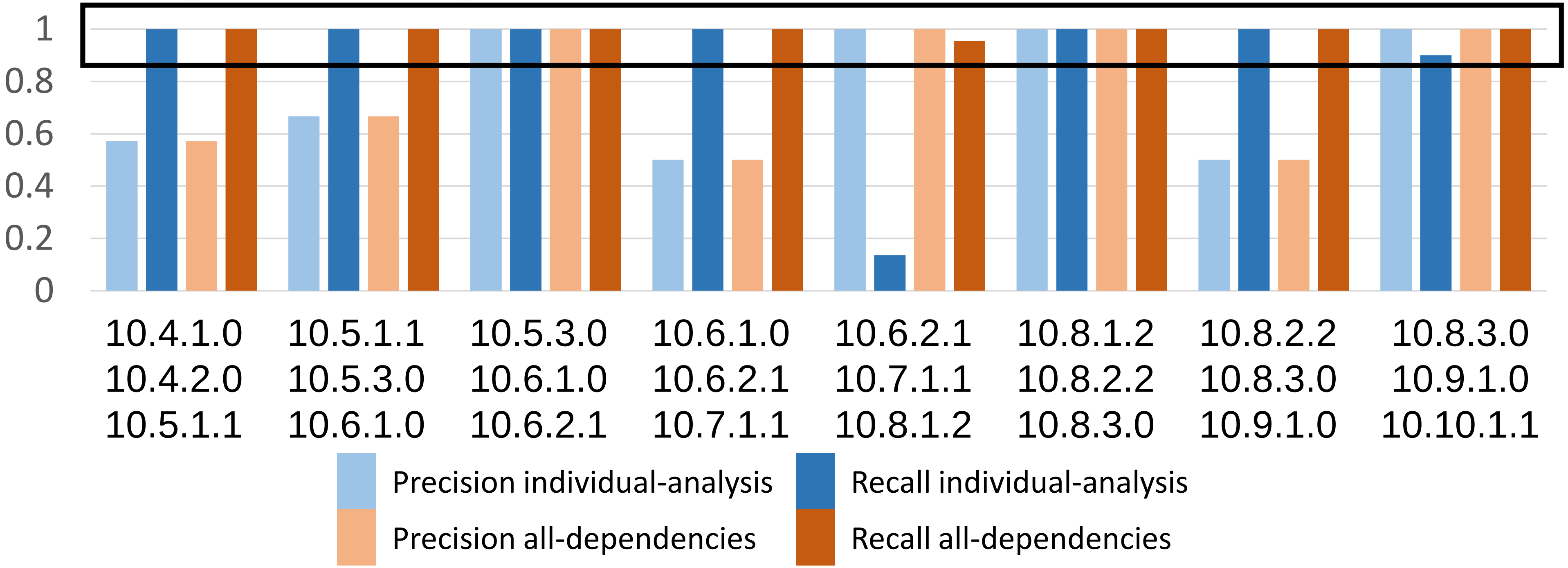
How did it go? – Prediction Phase



- High F-Measure values are due to a **high recall** and a **moderate precision**.
- The trained model is capable of finding most future dependencies, but it also predicts false dependencies.

Dependency-based Smells

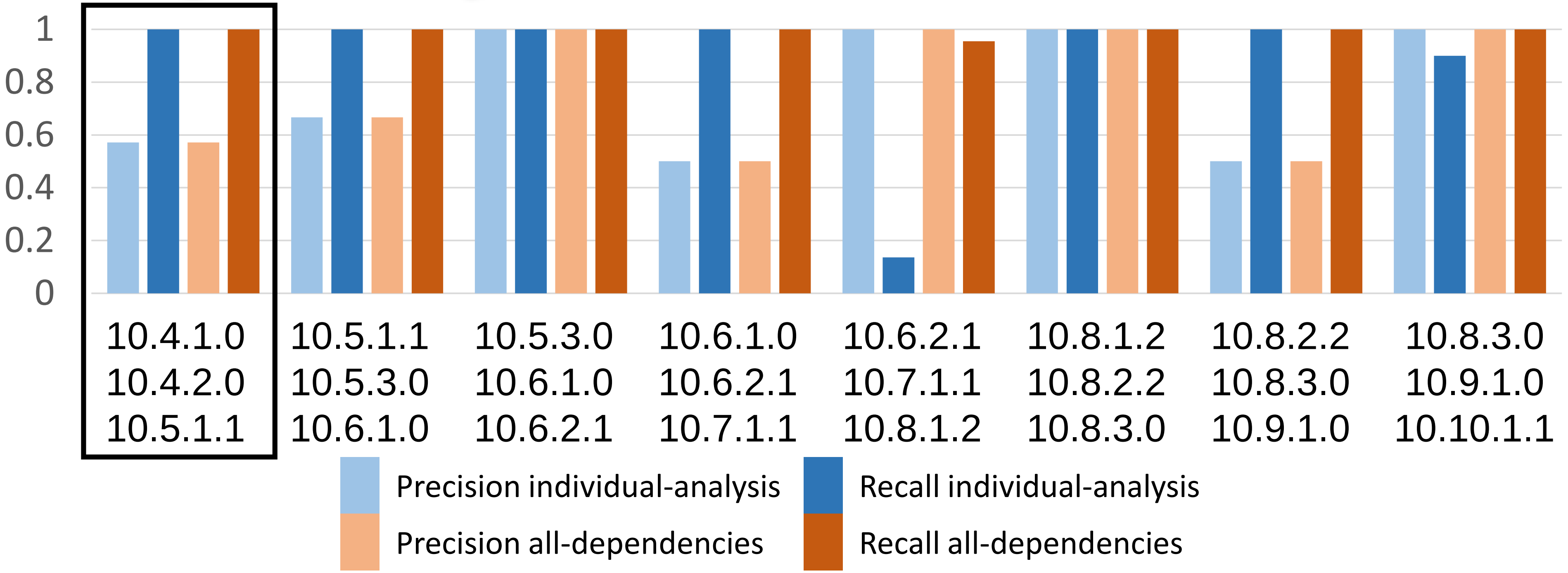
How did it go? – Cycle Prediction



- In most cases recall is almost perfect (almost every new dependency leading to the closure of a quasi-cycle was found).
- Precision indicates that some mistaken dependencies are also predicted.
  - At most 5 mistaken predictions (0.06% of total dependencies).

Dependency-based Smells

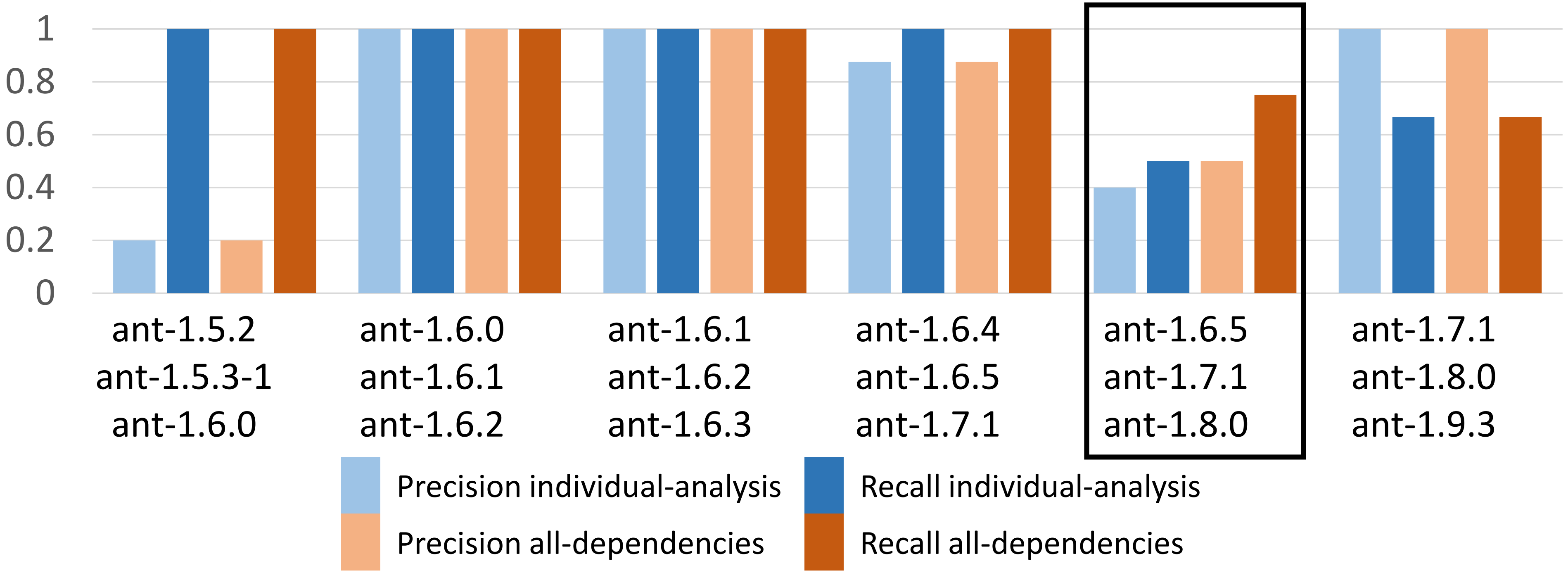
How did it go? – Cycle Prediction



- Similar performance for both variants.
  - Quasi-cycles are closed by adding only one dependency or by multiple dependencies that also individually close cycles.

Dependency-based Smells

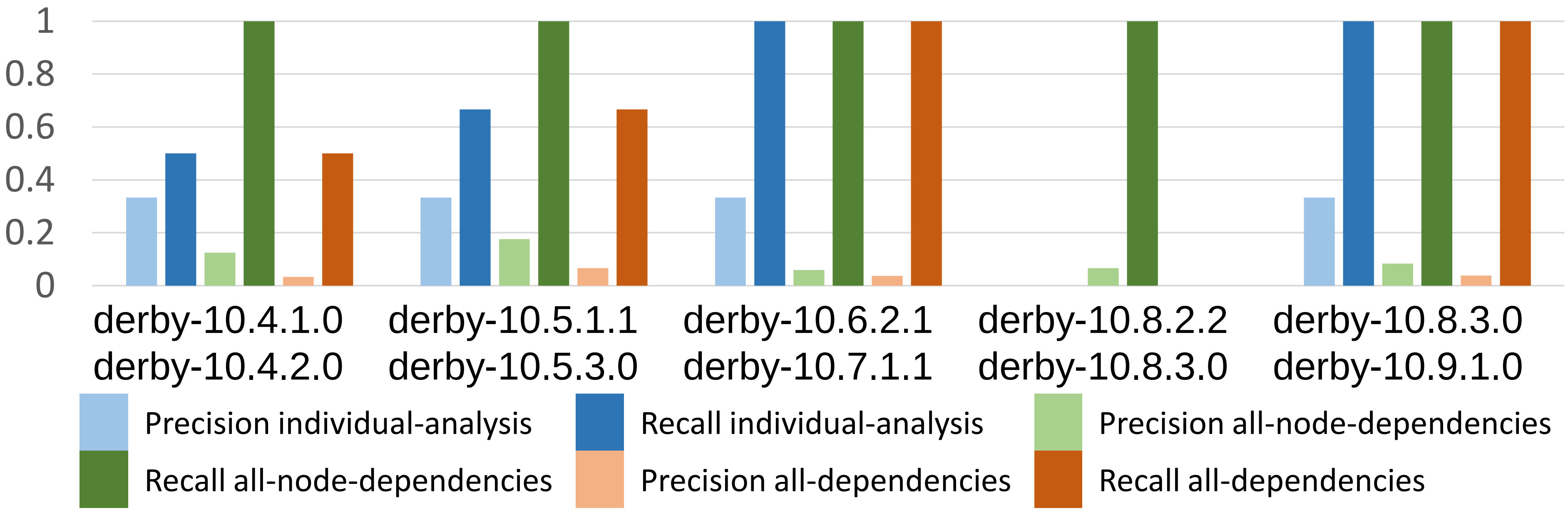
How did it go? – Cycle Prediction



- Differences between the variants could be explained by the existence of quasi-cycles needing +1 dependency to be closed.
  - Precision of individual-analysis is not affected, but recall decreases.

Dependency-based Smells

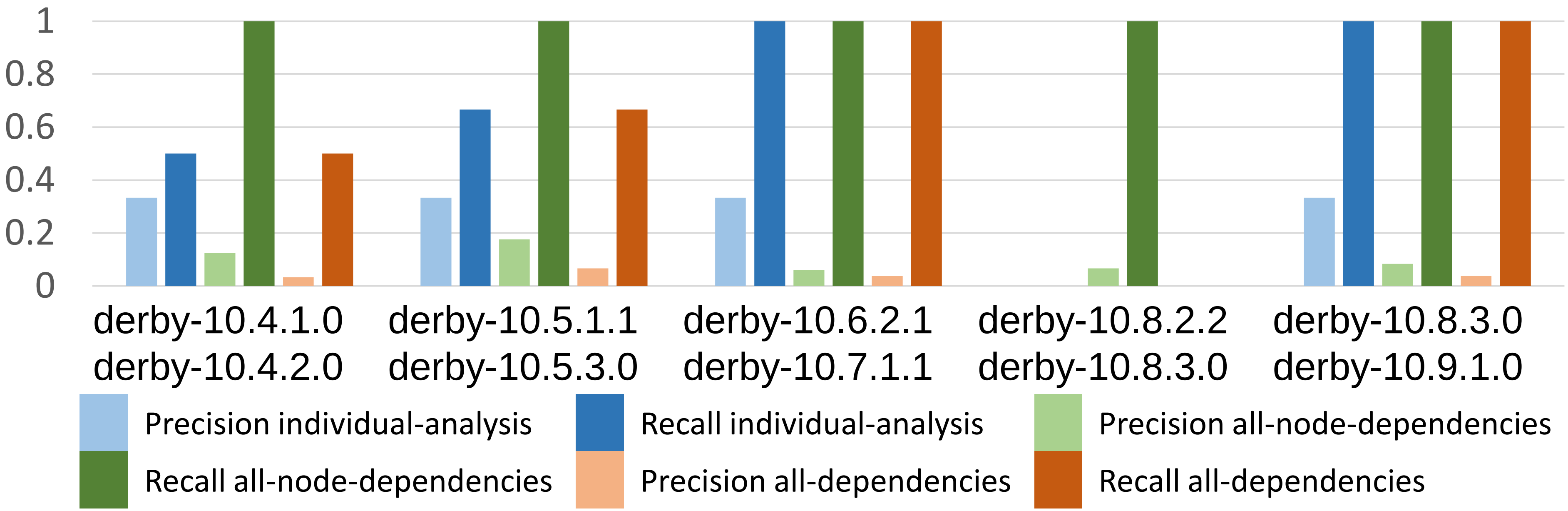
How did it go? – Hub Prediction



- The performance of the variants differ.
    - individual-analysis. ↓recall (highest number of missed nodes) ↑ precision (fewest mistaken predictions)
    - all-node. ↑ recall → precision (mistaken predictions) → neighbourhood more important than overall structure
    - all-dependencies. ↓recall ↓precision

Dependency-based Smells

How did it go? – Hub Prediction



- At least one missed smell.

  - Mistaken predictions in the first phase.
  - Hubs might not only depend on the addition of new edges but on the overall graph structure.
  - Hubs might also depend on the unknown structure of the graph (dependencies added between yet unknown packages).

Lessons Learned

- An initial evaluation with two types of smells showed a good performance!
  - High recall, low precision.
- Including content-based features improves dependency prediction.
- The choice of the filter variant (for a given smell type) can affect both recall and precision.
  - We preferred good recall over precision in the analysed cases.
- Smell predictions depended on both the **current** overall system structure and **version** history.

Lessons Learned

What do we do now?

- Perform a systematic study with more systems and other dependency-based smells.

- The prediction capabilities are sensitive to the prediction model.
  - Analyse and extend the set of features used.
  - Considering software specific-metrics?

- Smells might not be harmful.
  - How can we train a model to discard them?

Lessons Learned

What do we do now?

- Perform a systematic study with more systems and other dependency-based smells.

- The prediction capabilities are sensitive to the prediction model.
  - Analyse and extend the set of features used.
  - Considering software specific-metrics?

Increase precision!

- Smells might not be harmful.
  - How can we train a model to discard them?

Table of Contents

Time-series Smell Prediction

- Most link prediction approaches have been proposed based on **static** network representations.
  - A **snapshot** of the network is available and the goal is to predict the future links.

Time-series Smell Prediction

- Most link prediction approaches have been proposed based on **static** network representations.
  - A **snapshot** of the network is available and the goal is to predict the future links.

- Nonetheless, networks are **dynamic** and perhaps nondeterministic.
  - Changes in the underlying structure and parameters over time.

- In these cases, additional information could be extracted from the **history of network evolution**.

Time-series Smell Prediction

- Most link prediction approaches have been proposed based on **static** network representations.
  - A **snapshot** of the network is available and the goal is to predict the future links.

- Nonetheless, networks are **dynamic** and perhaps nondeterministic.
  - Changes in the underlying structure and parameters over time.

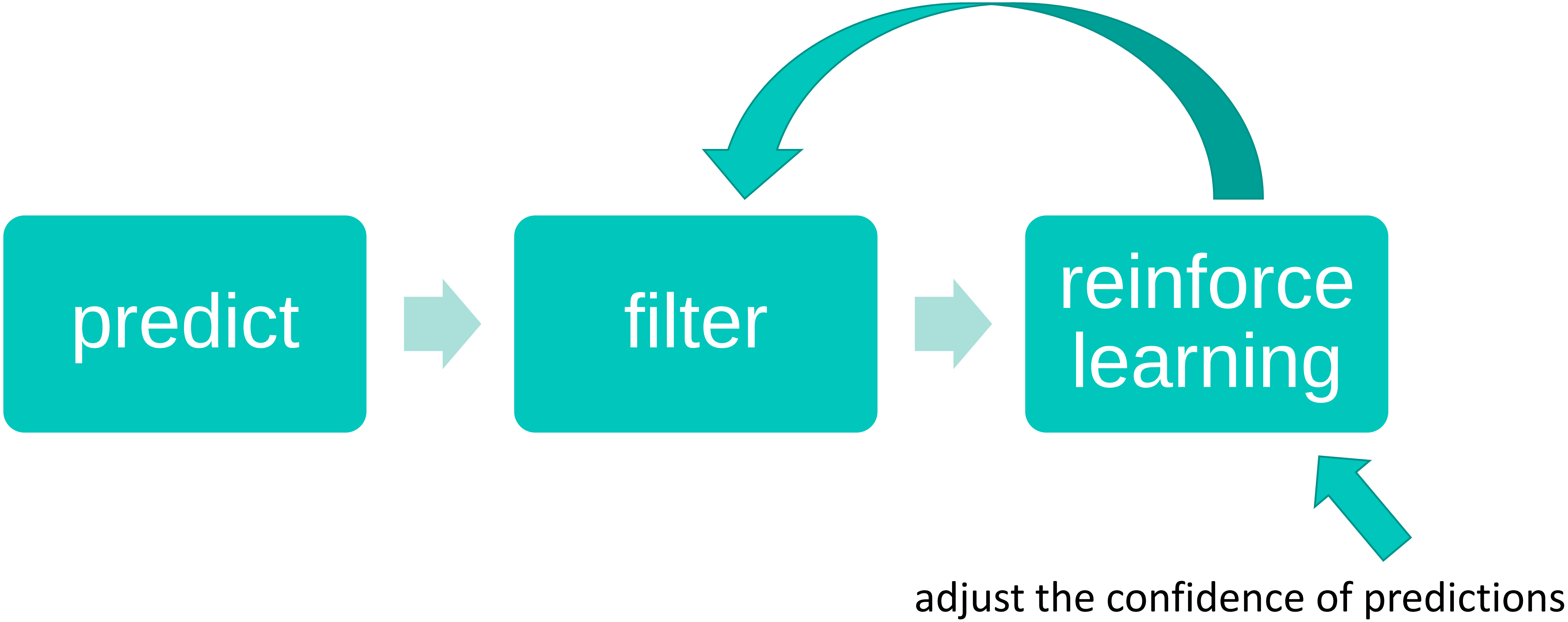
- In these cases, additional information could be extracted from the **history of network evolution**.

Link prediction techniques could be enriched by including **time series** information and **reinforcement learning** mechanisms.

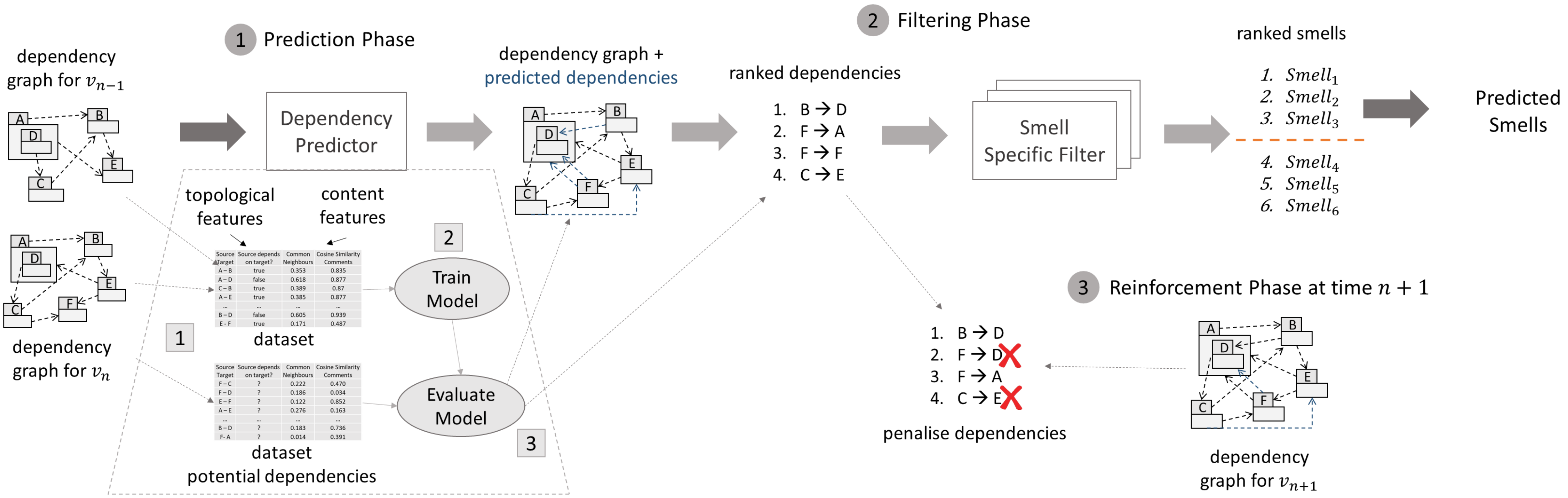
Time-series Smell Prediction

predict → filter

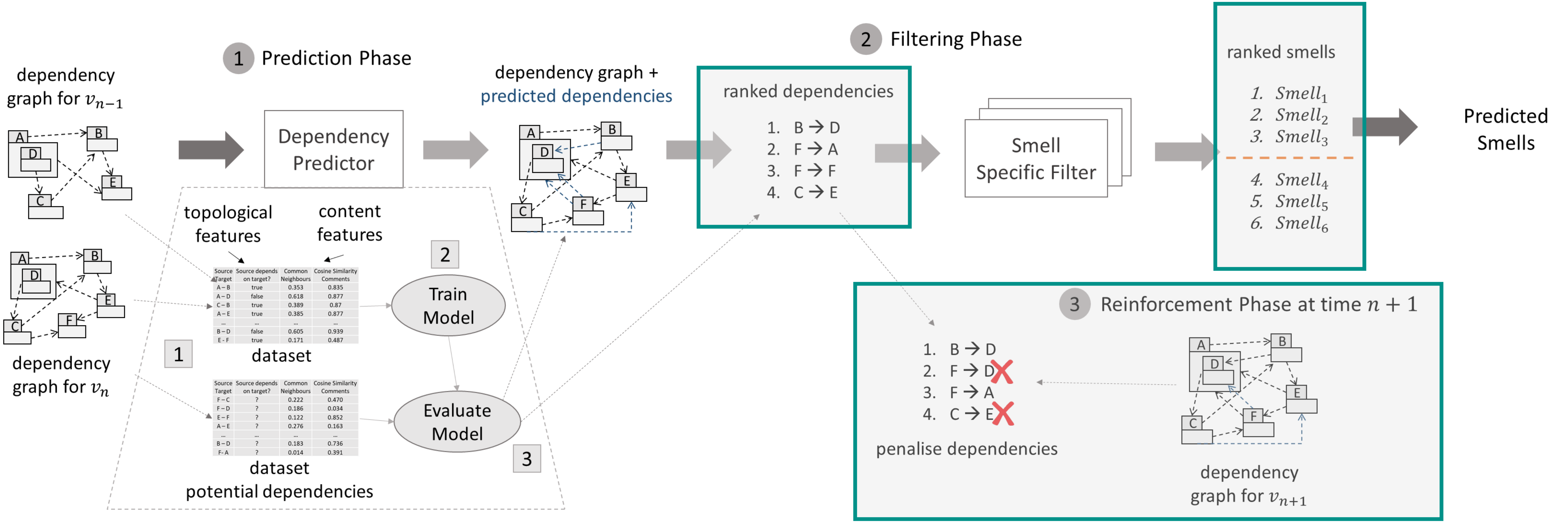
Time-series Smell Prediction



Time-series Smell Prediction



Time-series Smell Prediction



Time-series Smell Prediction

Leverages on the history of software versions to estimate the confidence of predictions.

Three phases:

Time-series Smell Prediction

Leverages on the history of software versions to estimate the confidence of predictions.

Three phases:

1. Considering the information of two software versions, it predicts the appearance of new dependencies in the next system version.

Time-series Smell Prediction

Leverages on the history of software versions to estimate the confidence of predictions.

Three phases:

1. Considering the information of two software versions, it predicts the appearance of new dependencies in the next system version.

2. Smells are filtered and **ranked** according to:
   - The characteristics of the specific types of smells.
   - The confidence score of the predicted dependencies.

Time-series Smell Prediction

Leverages on the history of software versions to estimate the confidence of predictions.

Three phases:

1. Considering the information of two software versions, it predicts the appearance of new dependencies in the next system version.

2. Smells are filtered and **ranked** according to:
    - The characteristics of the specific types of smells.
    - The confidence score of the predicted dependencies.

3. When the next system version is known, the confidence of predicted dependencies is updated to reflect the actual changes in the actual dependency graph.
    - Applies an adaptation of reinforcement learning.

Time-series Smell Prediction

Filtering & Ranking

- Up to now, all predicted smells were presented to the developer, which resulted in the mistaken prediction of smells.

Time-series Smell Prediction

Filtering & Ranking

- Up to now, all predicted smells were presented to the developer, which resulted in the mistaken prediction of smells.
- Once smells are predicted and prioritised, we need to define which of them are going to be presented.
- **Choosing the number of smells to recommend might not be easy!**

Filtering & Ranking

Several alternatives:

- Set a **fixed threshold** and always recommend the same number of smells.
    - Threshold could be based on relevancy scores, a percentage of instances or the number of predicted items.

    - This has several drawbacks.
        - Ignores the characteristics of the task at hand.
        - Might fail to acknowledge the possibility of rankings presenting different scores distributions.

Time-series Smell Prediction

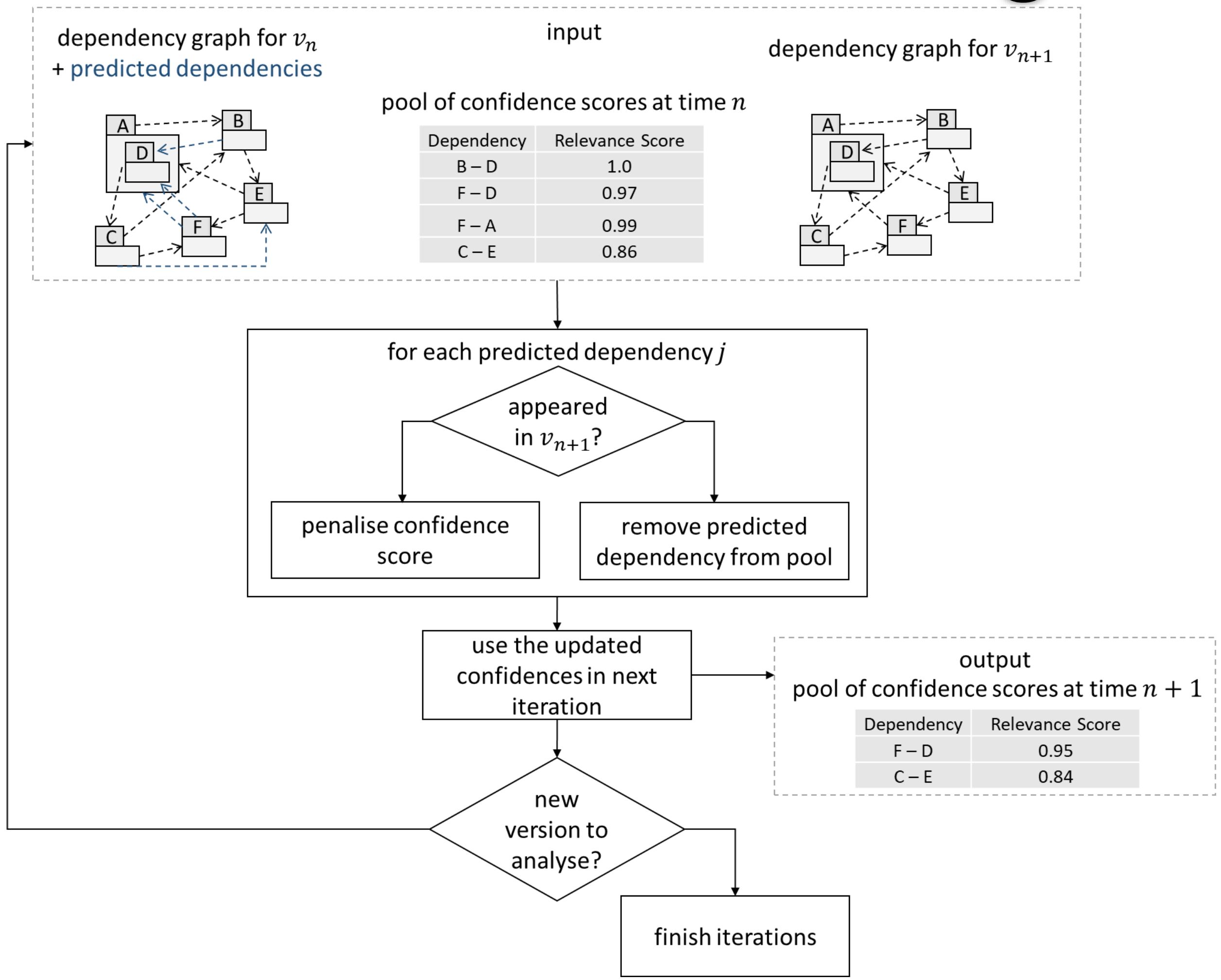
Filtering & Ranking

Several alternatives:

- The number of smells to predict will be chosen according to the **history of discoverable smells in the previous versions**.

- The average number of predictable smells in the previous versions of the system plus its standard deviation.

Reinforcement Learning

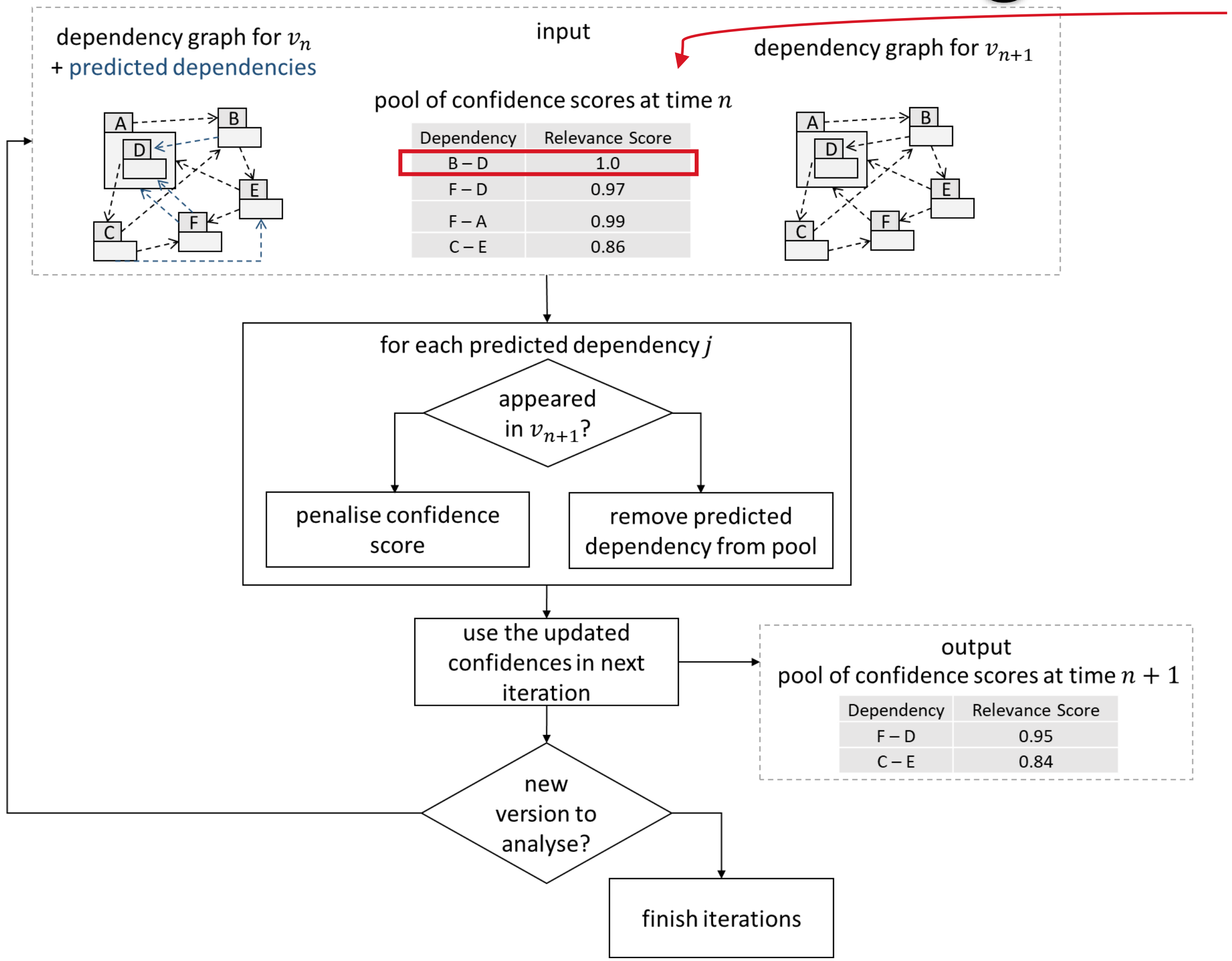


- When the following software version is known, the reinforcement learning phase <u>updates the relevance of dependencies</u> based on the structure of the newest system version.

- Includes additional information regarding the evolution of the network.

Time-series Smell Prediction

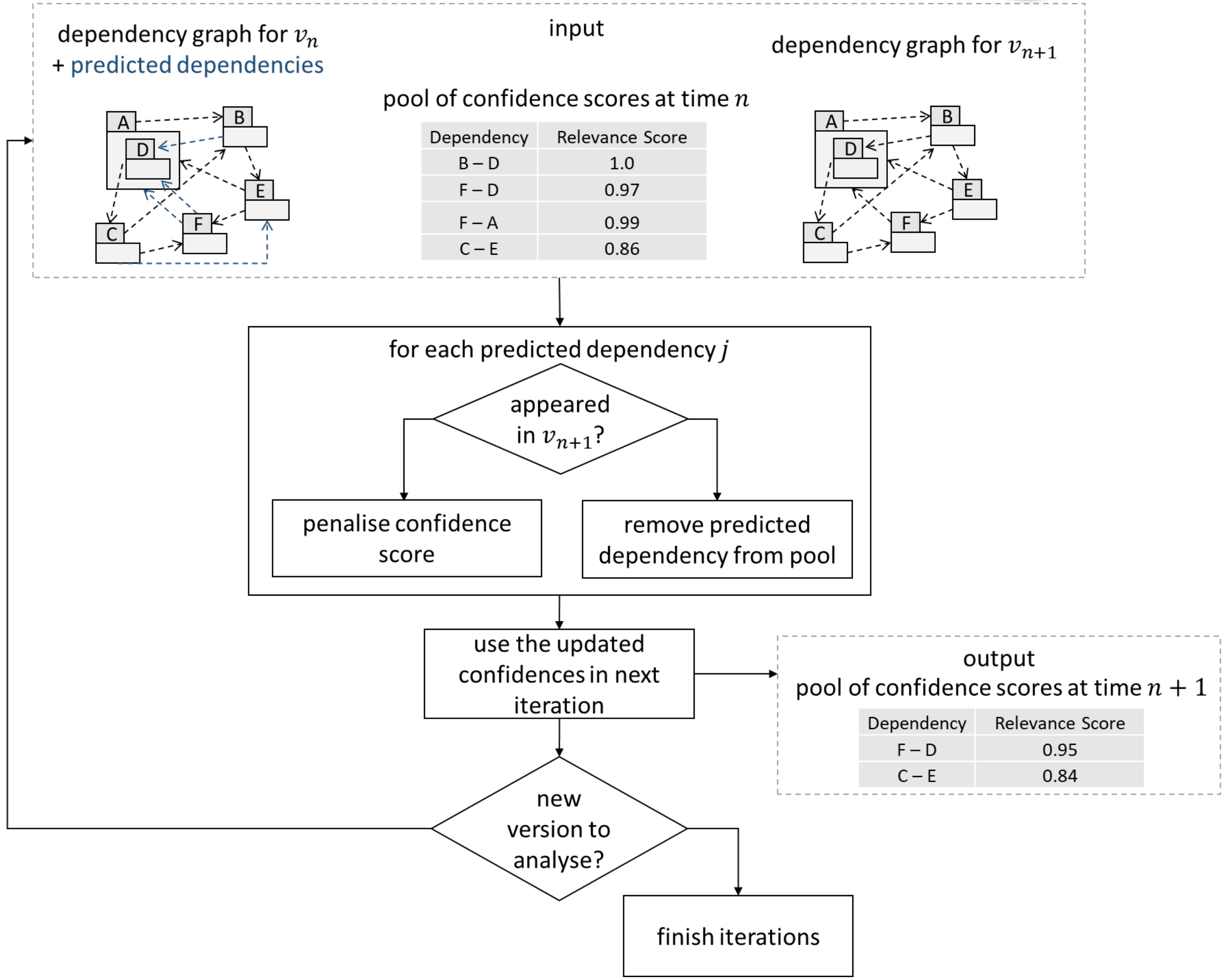
Reinforcement Learning



- A pool of predicted dependencies is maintained.
- In every iteration, new predicted dependencies are added to the pool and associated to a learning automaton that updates the confidence of the predicted dependency according to changes in the environment.
- The learning automaton starts with a confidence of 1.

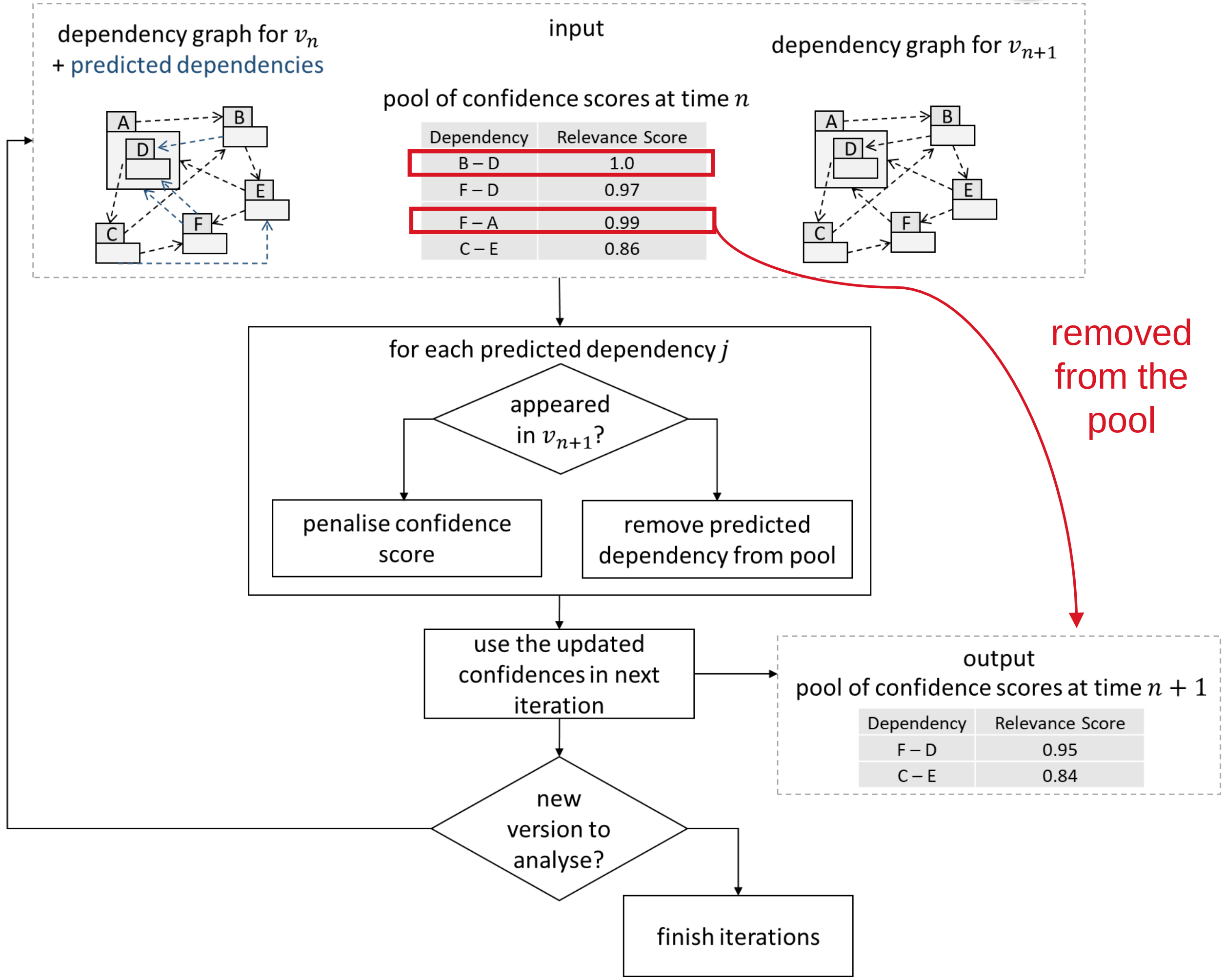
Time-series Smell Prediction

Reinforcement Learning



For each predicted dependency there are **two** possibilities.

Reinforcement Learning

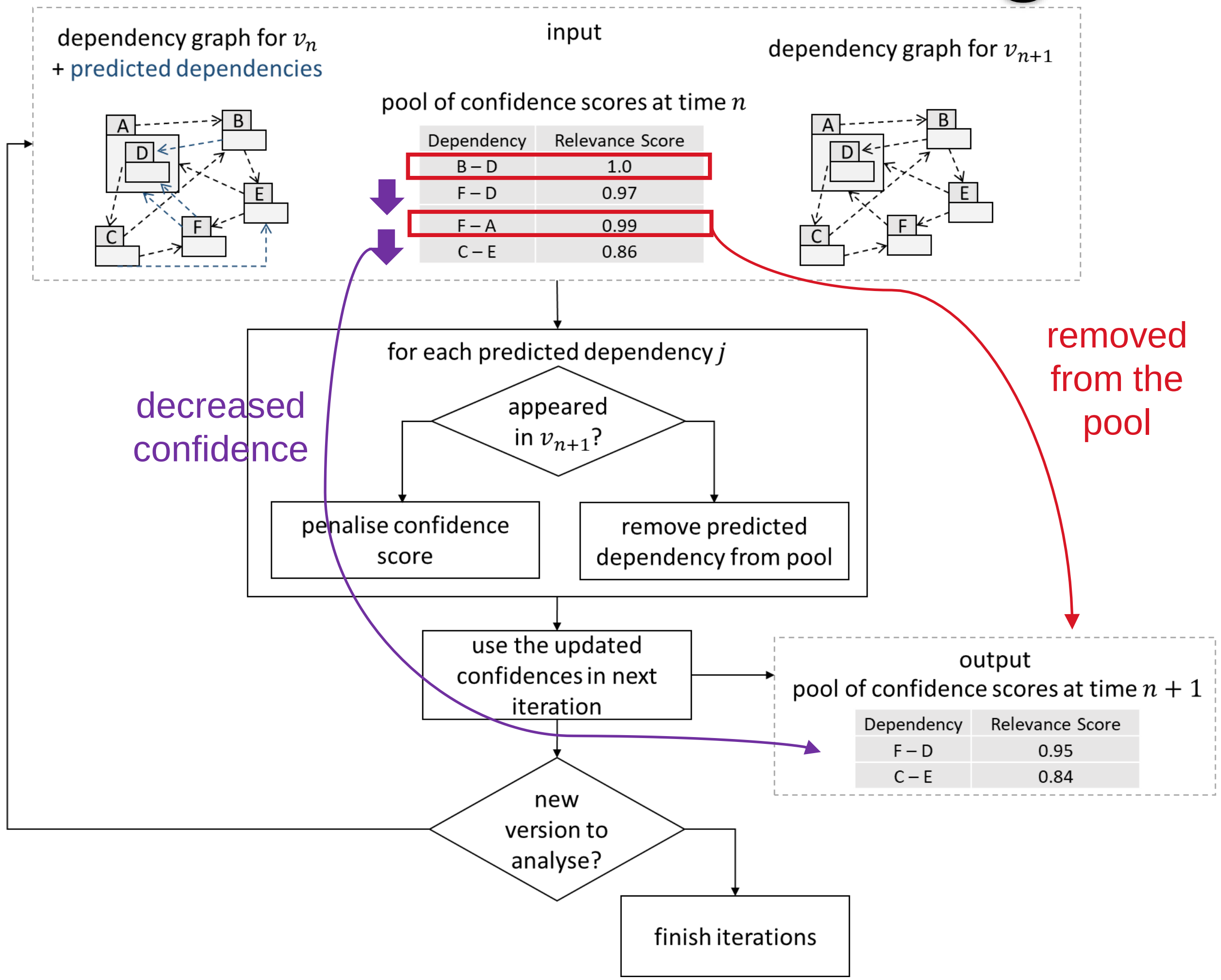


For each predicted dependency there are **two** possibilities.

1. The dependency **appears** on the new software version → It is removed from the pool.

Time-series Smell Prediction

Reinforcement Learning



For each predicted dependency there are **two** possibilities.

1. The dependency **appears** on the new software version → It is removed from the pool.

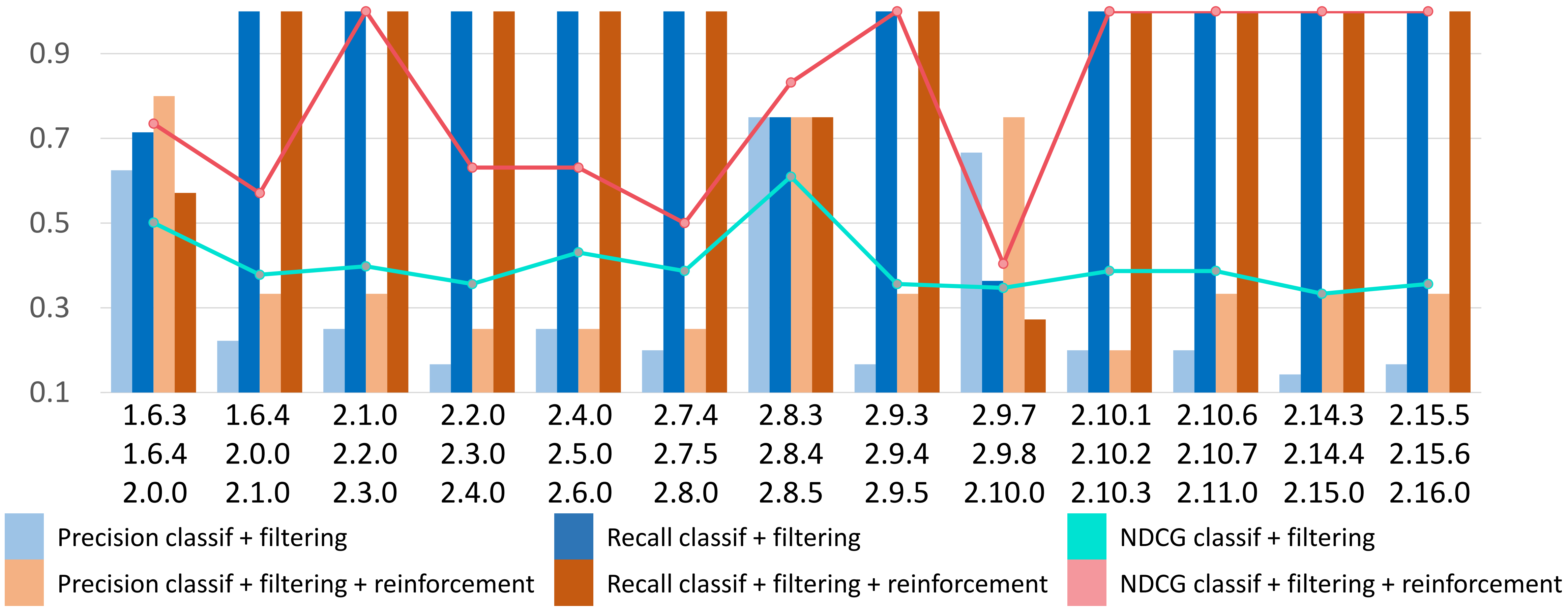
2. The dependency **does not appear** on the new software version → The associated the learning automaton decreases its confidence to penalise the incorrect prediction.

$$C_{n+1} = 1 - b * C_n$$

Time-series Smell Prediction

How did it go?

Apache Camel

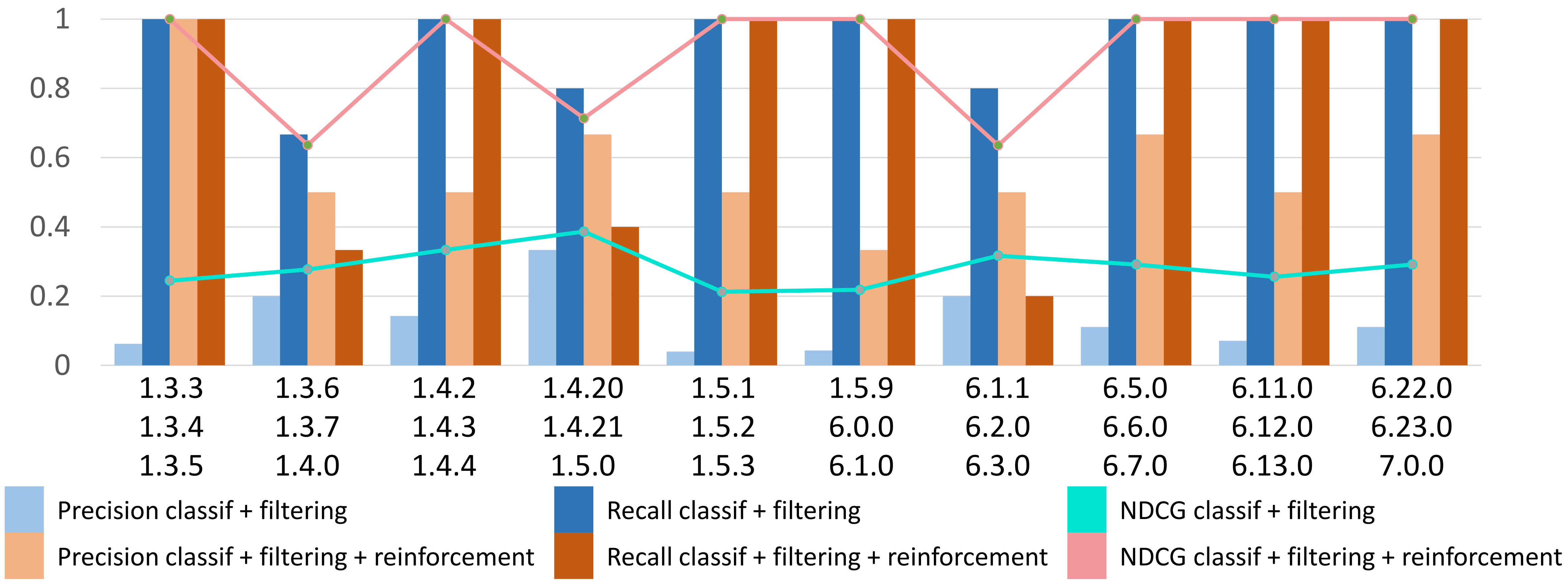


We still need to tailor the size of the ranking.

Time-series Smell Prediction

How did it go?

Apache Wicket

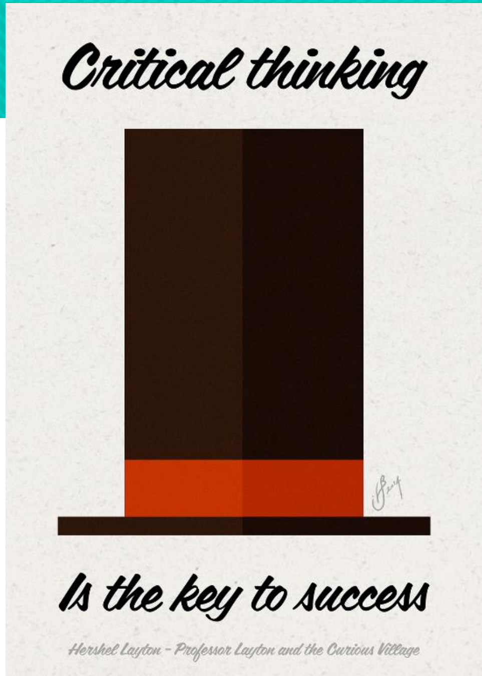


We still need to tailor the size of the ranking.

Table of Contents

Summary

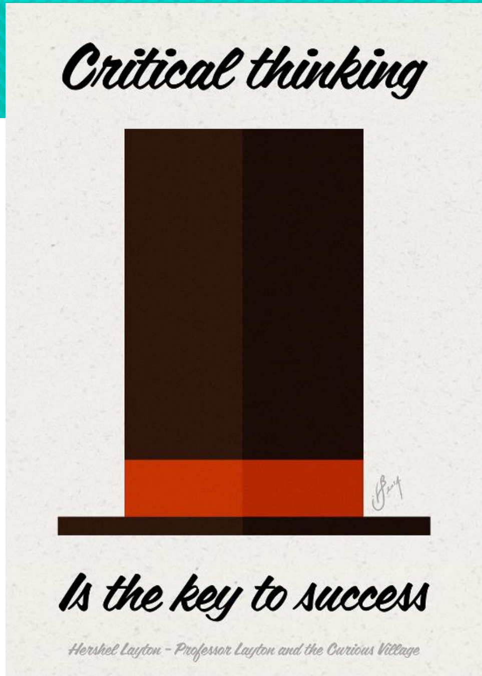
- As software systems evolve **“*undesired*” dependencies appear.**
  - Degradation of intended design.



Summary

- As software systems evolve **“*undesired*” dependencies appear.**
  - <u>Degradation</u> of intended design.

Machine Learning can help **predict dependencies**.

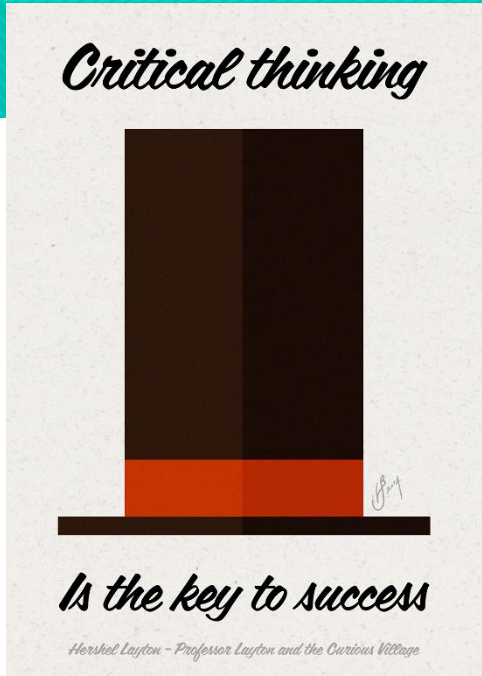


Summary

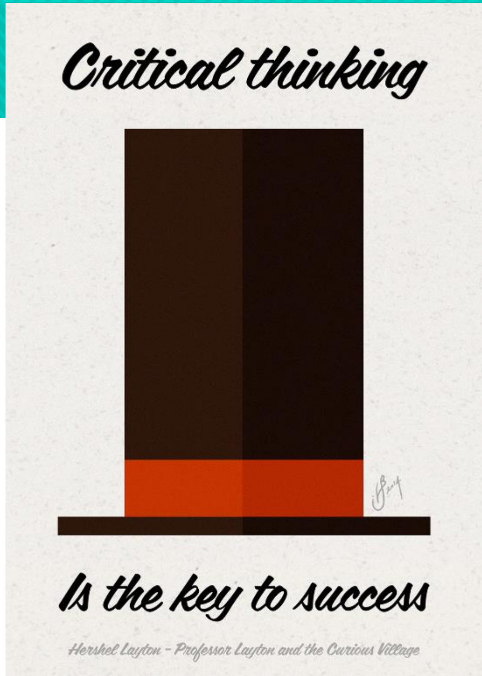
- As software systems evolve “***undesired***” **dependencies appear.**
  - <u>Degradation</u> of intended design.

Machine Learning can help **predict dependencies**.

Predicted dependencies can be used to **predict smells**.



Summary



- As software systems evolve “***undesired***” **dependencies appear.**
  - <u>Degradation</u> of intended design.

Machine Learning can help **predict dependencies**.

Predicted dependencies can be used to **predict smells**.

Plan ahead for actions that **preserve** the **quality** of the system.

We are far from finished…

"Now this is not the end. It is not even the beginning of the end.
But it is, perhaps, the end of the beginning."

- Can communities help boost predictions?
- More features.
  - Design metrics? OO metrics? Global characteristics of smells?
- Analyse other dependency-based problems!
  - Analyse other types of smells?
- Can we predict the appearance of new nodes (e.g. new packages, classes)?
- Can we predict the disappearance of dependencies?
- How about a tool?

… and a lot more!

Questions?

Two and a Half Papers

- Diaz-Pace, J.A., Tommasel, A., and Godoy, D. “**Can Network Analysis Techniques help to Predict Design Dependencies? An Initial Study**”. In Proceedings of the IEEE INTERNATIONAL CONFERENCE ON SOFTWARE ARCHITECTURE (ICSA 2018). Seattle USA. April, 2018. https://arxiv.org/abs/1808.02776v1

- Diaz-Pace, J.A., Tommasel, A., and Godoy, D. “**Towards Anticipation of Architectural Smells using Link Prediction Techniques**”. In Proceedings of the 18th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2018). Madrid, Spain. September, 2018. http://arxiv.org/abs/1808.06362

Keeping one-step ahead of Architectural Smells:
A Machine Learning Application

Dr. Antonela Tommasel

ISISTAN, CONICET-UNICEN, Argentina

antonela.tommasel@isistan.unicen.edu.ar

CONICET

ISISTAN