Prediction of architectural smells

Fabio Cimmino 807070
Roberto Lotterio 807500

Contents

1 Introduction

Architectural smells [1] [2] are symptoms of bad code or design that can cause different quality problems, such as faults, technical debt [3], or difficulties with maintenance and evolution. Thus, it is very important to be able to understand how they evolved in the past and to predict their future evolution. In this project we propose a different machine learning models which predict architectural smells based on dependency features.
The work is divided into two parts: in the first part we analyzed a single version of 60 projects at the class and package level; in the second part we analyzed 10 versions of 10 projects at the class level.
All the analysis steps were implemented on the KNIME Analytics platform [4].

2 Study design

A review of the sections of this report is presented below, which have been carried out following the steps carried out during the realization of the project. We initially collected 60 projects and 10 versions of 10 projects as reported in Section 3. We then analyzed the projects collected to create the initial datasets. This phase is reported in Section 4.
Section 5 reports the analyzes carried out to find the best model which consist of features selection and comparison of models using a statistical test. Finally, the validation phase was carried out on the test set with the model deemed best by analyzing the performance as explained in Section 6.
In Sections 7 and 8 we have respectively summarized the results and conclusions of the project.

3 Data collection and description

The Java projects taken into account were found through the Qualitas Corpus [5] website. The sixty projects (single version) taken into consideration are reported in Table 1. For each project the domain of the application, the version, the number of classes, packages and lines of code are indicated.

Table 1: Analyzed Projects

Domain	Project	Version	NOC	NOP	TLOC
Database	axion	1.0-M2	257	13	24163
	cayenne	3.0.1	2991	185	192431
	db-derby	10.9.1.0	3010	217	651118
	hsqldb	2.0.0	644	26	143870
	squirrel-sql	3.1.2	73	2	6944
	hibernate	4.2.0	7119	856	431693
Graphic	displaytag	1.2	320	32	20498
	drawswf	1.2.9	311	34	27674
	itext	5.0.3	583	34	78348
	jasperreports	3.7.4	1709	61	169821
	jext	5.0	761	59	60160
	marauroa	3.8.1	247	41	17733
	megamek	0.35.18	1859	37	242836
IDE	checkstyle	5.6	533	42	36641
	colt	1.2.0	381	24	35919
	drjava	stable-20100913-r5387	1210	30	89477
	eclipse SDK	3.7.1			
	jpf	1.5.1	140	10	13342
	nakedobjects	4.0.0	2975	496	133936
	trove	2.1.0	72	4	5845
Middleware	informa	0.7.0	223	26	13874
	jena	2.6.3	1279	48	65774
	jspwiki	2.8.4	582	70	60250
	jtopen	9.4	1915	15	342032
	openjms	0.7.7-beta-1	616	66	39435
	oscache	2.3	115	22	7624
	picocontainer	2.10.2	206	15	9253
	xmojo	5.0.0	22	9	2809
	quartz	1.8.3	269	51	28557
	QuickServer	2.1.0	196	28	18339
	sunflow	0.07.2	209	22	21970
	tapestry	5.1.0.5	2119	139	97206
Parser	ant	1.8.2	1608	122	127507
	antlr	3.4	381	20	47443
	apache-maven	3.0.5	837	143	65685
	javacc	5.0	107	8	14633

Continued on next page

Table 1 – *Continued from previous page*

Domain	**Project**	**Version**	**NOC**	**NOP**	**TLOC**
	jparse	0.96	75	4	24796
	nekohtml	1.9.14	64	7	7647
	xalan	2.7.1	1402	86	183709
	xerces	2.10.0	947	53	125973
Testing	cobertura	1.9.4.1	160	34	54555
	emma	2.0.5312	290	27	21492
	findbugs	1.3.9	1432	67	110782
	fitjava	1.1	95	5	3457
	jmeter	2.5.1	1038	175	94778
	junit	4.10	171	28	6580
	log4j	2.0-beta	606	61	32658
	pmd	4.2.5	872	88	60739
Tool	freecs	1.3.20111225	146	12	22645
	heritrix	1.14.4	656	48	64916
	james	2.2.0	306	31	27087
	jfreechart	1.0.13	1037	69	143062
	jgraph	5.13.0.0	298	34	31818
	jgraphpad	5.10.0.2	375	22	24208
	jmoney	0.4.4	83	4	8197
	jsXe	04_beta	251	14	18494
	pooka	3.0-080505	491	28	44474
	proguard	4.9	648	35	62618
	webmail	0.7.10	115	19	10147

Table 2 shows the 10 projects considering 10 versions for each, reporting the initial and final version with the relative number of classes and packages.

Project	**Verison**	**NOC**	**NOP**
ant	1.7.2	802	108
	1.10.7	1242	134
camel	2.17	2466	181
	3.0.1	3849	206
clojure	1.1.0	1525	113
	1.10.0	3487	197
maven	2.0	104	12
	3.6.3	489	27
junit	4.4	154	23
	4.13	388	34
log4j	2.4	735	69
	2.12	1203	85
logback	0.2.5	89	12
	1.2.0	239	23
lombok	0.9.3	649	48
	1.18.0	940	72
mockito	2.22.0	614	41
	3.2.0	634	43
slf4j	1.1.0	23	4
	2.0.0	65	10

Table 2: Number of classes and packages for the first and last version of each project analyzed

4 Datasets creation

Once downloaded, each version was analyzed by Arcan [6] to extract the depenency features:

- **edge_id:** number that identifies the dependency between two classes or packages
- **from:** name of the class or package from which the edge starts
- **to:** name of the class or package where the edge arrives
- **type:** kind of dependency
- **weight:** number of dependencies of the underlying code

- **CD:** binary number indicating the presence or absence of a Cyclic Dependency. It refers to a subsystem (component) that is involved in a chain of relations that break the desirable acyclic nature of a subsystems dependency structure.
- **HL:** binary number indicating the presence or absence of a Hub-like Dependency. This smell arises when an abstraction has (outgoing and ingoing) dependencies with a large number of other abstractions.
- **UD:** binary number indicating the presence or absence of a Unstable Dependency. It describes a subsystem (component) that depends on other subsystems that are less stable than itself.

For the analysis of the single versions of 60 projects we have updated the three CD, HL and UD features into one feature as a target for prediction. So the target is a categorical variable that can take the following values: NOTHING, CD, UD, HL, CD-HL, CD-UD, HL-UD, CD-UD-HL.

4.1 Dataset for 60 class-level projects

The dataset for the single versions of the 60 projects at the class level is composed of the following features:

from	to	type	weight	Target

The edge id has been removed as it is not useful for prediction purposes. Furthermore, only the CD and HL features were used for the creation of the target variable since the Unstable Dependency is not detected at class level.

4.2 Dataset for 60 package-level projects

The dataset for the single versions of the 60 projects at the package level is composed of the following features:

from	to	weight	Target

The edge id was removed for the same reason as above. We did not consider the Type feature since the dependency type is established only at the class level.

4.3 Dataset for 10 version of a project

For the creation of this dataset we used a sliding window approach, using two versions as a window and the next as a target, as previously calculated. For the 10 versions, the minor versions of each project have been downloaded in ascending order. We decided to use the minor versions and not the release versions to capture a greater difference between versions (example: 1.1.0, 1.2.0, ... , 1.10.0).
We observed that although the versions considered were not continuous, the changes over time were minimal. So we decided to add metrics as attributes with the intent to increase differences between versions. For each class the following metrics were calculated with Arcan:

- Fan-in (FI): measure of the number of functions or methods that call some other function or method
- Fan-out (FO): number of functions that are called by function
- Coupling between objects (CBO): count of the number of classes that are coupled to a particular
- Lack of Cohesion of Methods (LCOM): numbers of pairs of methods that shared references to instance variables

Considering the dependency relationships, these metrics are calculated for both the "from" class and the "to" class. the dataset is formed in the following way:

from	to	type	version 1	version 2	metrics v1	metrics v2	target v3
from	to	type	version 2	version 3	metrics v2	metrics v3	target v4
. . .	. . .		. . .	. . .	. . .	. . .	. . .
from	to	type	version 8	version 9	metrics v8	metrics v9	target v10

By "version N" we mean the features "weight", "CD" and "HL" related to version N.

5 Quest for the best model

This section describes the procedure that allows you to determine the best classifier for the single versions of the 60 projects (at the class and package level) and for the 10 projects each consisting of 10 versions. The prediction performances of three models were compared. The classifiers chosen are the following:

- Bayes Net: K2 algorithm [7]
- Random Forest: random forest algorithm with information gain ratio as split criterion (Copersmith method [8])
- Ibk (K-Nearest Neighbors) [9] with K=1

The search for the optimal classifier was structured in the following way: we divided the dataset into two partitions, the first consists of 80% and was used to train and compare the models to find the best one. The second partition consisted of the remaining 20% of the dataset and was used to test the best model. We used stratified partitioning to keep the classes balanced. To compare the models, we made a features selection to find the best set of features, then the models were trained and finally their performances were compared with a test.

5.1 Features selection

The goal of the features selection is to test each possible subset of features finding the one which minimizes the error rate. We used the wrapper approach as a features selection method. This approach uses a predictive model to score feature subsets. Each new subset is used to train a model, which is tested on a hold-out set. Counting the number of mistakes made on that hold-out set (the error rate of the model) gives the score for that subset. Subsequently the models built on the best set of features were validated.
The part dedicated to training was further divided into 25% and 75%. Features selection was made on 25% while 75% is used for validation: for every model, the validation consisted in comparing the performances of the model when built with or without feature selection.

5.1.1 Features validation for the dataset of 60 project

Table 3 and Table 4 shows the results of the feature selection validation respectively at class and package level. The first column indicates the machine learning model, the second indicates the number of optimal and total features, the third column reports the error rate committed by the wrapper at that feature level and the last column reports how the accuracy achieved by the models on the validation set varies depending on that feature level.

Model	**# Features**	**Error Rate**	**Accuracy**
Bayes Net	3	0.167	0.88
	All	0.170	0.88
Random Forest	1	0.183	0.85
	All	0.485	0.47
IBK	2	0.168	0.87
	All	0.178	0.85

Table 3: Features validation for dataset at class level

Model	**# Features**	**Error Rate**	**Accuracy**
Bayes Net	All	0.394	0.73
Random Forest	1	0.436	0.64
	All	0.489	0.49
IBK	All	0.429	0.62

Table 4: Features validation for dataset at package level

As for the class level dataset, as shown in Table 1, it is possible to observe that Bayes Net and IBK do not have a significant difference despite the features selection. Instead Random Forest after the features selection manages to increase its accuracy from 47% to 85%.
While as regards the package level dataset, as shown in Table 2, for Bayes Net and IBK it was not possible to identify an optimal subset of features.

5.1.2 Features validation for the 10 version of 10 projects

This section reports the results of the features selection carried out on the 10 versions of the 10 projects using the sliding window approach. As in the previous section, we use the wrapper method for features selection by comparing the three machine learning models (BayesNet, Random Forest, IBk).

Model	# Features	Error Rate	Accuracy
Bayes Net	12	0.021	0.97
	All	0.036	0.96
Random Forest	7	0.020	0.97
	All	0.034	0.97
IBK	1	0.025	0.96
	All	0.042	0.96

Table 5: Features validation for 10 version of Ant

Model	# Features	Error Rate	Accuracy
Bayes Net	8	0.018	0.98
	All	0.051	0.95
Random Forest	8	0.017	0.98
	All	0.034	0.98
IBK	2	0.019	0.98
	All	0.037	0.97

Table 6: Features validation for 10 versions of Camel

Model	# Features	Error Rate	Accuracy
Bayes Net	8	0.010	0.98
	All	0.040	0.97
Random Forest	8	0.007	0.97
	All	0.040	0.98
IBK	5	0.010	0.98
	All	0.047	0.97

Table 7: Features validation for 10 versions of jUnit

Model	# Features	Error Rate	Accuracy
Bayes Net	6	0.005	0.98
	All	0.025	0.97
Random Forest	10	0.005	0.99
	All	0.016	0.99
IBK	7	0.009	0.98
	All	0.018	0.98

Table 8: Features validation for 10 versions of Log4j

Model	# Features	Error Rate	Accuracy
Bayes Net	All	0	0.98
RandomForest	All	0	1
IBK	5	0	1
	All	0.05	1

Table 9: Features validation for 10 versions of Maven

Model	# Features	Error Rate	Accuracy
Bayes Net	7	0.003	1
	All	0.027	0.98
Random Forest	15	0.002	1
	All	0.004	1
IBK	2	0.003	1
	All	0.006	0.99

Table 10: Features validation for 10 versions of Clojure

Model	# Features	Error Rate	Accuracy
Bayes Net	6	0	1
	All	0.011	0.94
Random Forest	7	0	0.98
	All	0.046	0.98
IBK	5	0	0.99
	All	0.046	0.98

Table 11: Features validation for 10 versions of Logback

Model	# Features	Error Rate	Accuracy
Bayes Net	5	0.002	0.99
	All	0.014	0.98
Random Forest	19	0	0.98
	All	0.007	0.99
IBK	19	0	0.99
	All	0.016	0.99

Table 12: Features validation for 10 versions of Lombok

Model	# Features	Error Rate	Accuracy
Bayes Net	10	0	0.99
	All	0.010	0.99
Random Forest	4	0	1
	All	0.001	1
IBK	14	0.001	0.99
	All	0.008	0.99

Table 13: Features validation for 10 versions of Mockito

Model	# Features	Error Rate	Accuracy
Bayes Net	13	0	0.85
	All	0.111	0.96
Random Forest	5	0	0.92
	All	0.111	0.92
IBK	6	0	0.92
	All	0.444	0.96

Table 14: Features validation for 10 versions of Slf4j

5.2 Model comparison

Every model was trained and tested on the same partition used for the validation of the feature selection phase, by using the K-fold cross validation procedure (10 folds, stratified sampling) with the chosen subset of features. We have compared the various models in pairs by applying the Single Sample T-test on the difference in their error rates obtained for each fold of the cross validation. We carried out this statistical test to verify that the difference between the model with the highest accuracy compared to the rest is statistically significant. For every test the confidence interval was set to 95%.

5.2.1 Model comparison for 60 projects

Below is a comparison of the models for the datasets of the 60 projects at the class and package level. For both datasets the first table indicates the accuracy obtained in cross validation while in the second we report the results of the T-test.

Model	**Accuracy**
Bayes Net	0.893
Random Forest	0.852
IBk	0.874

Models	**Lower Bound**	**Upper Bound**
Bayes Net vs. Random Forest	- 0.0422	- 0.0388
Bayes Net vs. IBk	- 0.0204	- 0.0168
Random Forest vs. IBk	0.00204	0.0234

Table 15: Model comparison at class level

Model	**Accuracy**
Bayes Net	0.761
Random Forest	0.649
IBk	0.623

Models	**Lower Bound**	**Upper Bound**
Bayes Net vs. Random Forest	- 0.1207	- 0.1027
Bayes Net vs. IBk	- 0.1488	- 0,1268
Random Forest vs. IBk	- 0.0342	- 0.018

Table 16: Model comparison at package level

As can be seen from Table 15, the best class-level model is BayesNet with an accuracy of 0.893. It can be said that BayesNet is actually the best model since the confidence interval between BayesNet and the other two models does not include 0 (i.e. we have a statistically significant difference).Even at the package level, the best model is BayesNet and is also confirmed by the results of the T-test (Table 16).

5.2.2 Model comparison for 10 versions of 10 projects

As in the Subsection 4.2.1, in this subsection the results of the comparisons between the various models are reported considering for each of the 10 projects the 10 versions.

Model	Accuracy
Bayes Net	0.969
Random Forest	0.971
IBk	0.974

Models	Lower Bound	Upper Bound
Bayes Net vs. Random Forest	0.0003	0.0037
Bayes Net vs. IBk	0.0031	0.0073
Random Forest vs. IBk	0.0012	0.0052

Table 17: Model comparison for 10 versions of Ant

As for Ant (Table 17), the best model is IBk with an accuracy of 0.974. This observation is also confirmed by the T-test.

Model	Accuracy
Bayes Net	0.978
Random Forest	0.980
IBk	0.979

Models	Lower Bound	Upper Bound
Bayes Net vs. Random Forest	0.0016	0.0034
Bayes Net vs. IBk	- 0.0005	0.0021
Random Forest vs. IBk	- 0.0024	- 0.0009

Table 18: Model comparison for 10 versions of Camel

For Camel (Table 18) the best model is Random Forest with an accuracy of 0.98. This observation is also confirmed by the T-test. Although the comparison between IBk and BayesNet is not statistically significant, we do not take this comparison into consideration since the model with the highest accuracy is Random Forest.

Model	Accuracy
Bayes Net	0.996
Random Forest	0.997
IBk	0.996

Models	Lower Bound	Upper Bound
Bayes Net vs. Random Forest	0.0009	0.0018
Bayes Net vs. IBk	0.0002	0.0009
Random Forest vs. IBk	- 0.0013	- 0.0003

Table 19: Model comparison for 10 versions of Clojure

As for Clojure (Table 19), the best model is Random Forest with an accuracy of 0.997. This observation is also confirmed by the T-test.

Model	Accuracy
Bayes Net	0.979
Random Forest	0.982
IBk	0.977

Models	Lower Bound	Upper Bound
Bayes Net vs. Random Forest	- 0.0009	0.0068
Bayes Net vs. IBk	- 0.0065	0.0028
Random Forest vs. IBk	- 0.0085	- 0.0010

Table 20: Model comparison for 10 versions of Junit

As for Junit (Table 20), the best model is Random Forest. This observation is confirmed by the comparison with IBk but not with BayesNet. Since there is no statistically significant difference between Random Forest and BayesNet, so it is not possible to establish which is the best model, we decided to take the one with the highest accuracy.

Model	**Accuracy**
Bayes Net	0.984
Random Forest	0.989
IBk	0.985

Models	**Lower Bound**	**Upper Bound**
Bayes Net vs. Random Forest	0.0031	0.0068
Bayes Net vs. IBk	- 0.0023	0.0033
Random Forest vs. IBk	- 0.0060	- 0.0029

Table 21: Model comparison for 10 versions of Log4j

As for Log4j (Table 21), the best model is Random Forest with an accuracy of 0.989. This observation is also confirmed by the T-test.

Model	**Accuracy**
Bayes Net	0.985
Random Forest	0.985
IBk	0.980

Models	**Lower Bound**	**Upper Bound**
Bayes Net vs. Random Forest	- 0.0085	0.0085
Bayes Net vs. IBk	- 0.0138	0.0037
Random Forest vs. IBk	- 0.0127	0.0026

Table 22: Model comparison for 10 versions of Logback

As for Logback (Table 22) BayesNet and Random Forest have the same accuracy. By performing the T-test between the various models, it cannot be concluded that no model is actually better than another. So we decided to take BayesNet as the best model.

Model	**Accuracy**
Bayes Net	0.990
Random Forest	0.986
IBk	0.990

Models	**Lower Bound**	**Upper Bound**
Bayes Net vs. Random Forest	- 0.0084	- 0.0007
Bayes Net vs. IBk	- 0.0038	0.0023
Random Forest vs. IBk	0.0013	0.0062

Table 23: Model comparison for 10 versions of Lombok

As for Lombok (Table 23) BayesNet and IBk are the best. Both have a statistically significant difference with Random Forest but not with each other. Since BayesNet is more efficient, we choose the latter as the best model.

Model	**Accuracy**
Bayes Net	0.992
Random Forest	0.999
IBk	0.999

Models	**Lower Bound**	**Upper Bound**
Bayes Net vs. Random Forest	0.0025	0.0116
Bayes Net vs. IBk	0.0025	0.0116
Random Forest vs. IBk	0	0

Table 24: Model comparison for 10 versions of Maven

Regarding Maven (Table 24) Random Forest and IBk were identical as they have the same accuracy and both are statistically different from BayesNet. The confidence interval of the T-test between Random Forest and IBk turns out to be 0 since in each fold of the cross validation they have the same error rate.

Model	**Accuracy**
Bayes Net	0.995
Random Forest	0.997
IBk	0.995

Models	**Lower Bound**	**Upper Bound**
Bayes Net vs. Random Forest	0.0001	0.0032
Bayes Net vs. IBk	- 0.0019	0.0021
Random Forest vs. IBk	- 0.0026	- 0.0004

Table 25: Model comparison for 10 versions of Mockito

As for Mockito (Table 25) Random Forest is the best model. This observation is also confirmed by the T-test.

Model	Accuracy
Bayes Net	0.897
Random Forest	0.936
IBk	0.897

Models	Lower Bound	Upper Bound
Bayes Net vs. Random Forest	- 0.0229	0.0979
Bayes Net vs. IBk	- 0.0971	0.0971
Random Forest vs. IBk	- 0.1438	0.0688

Table 26: Model comparison for 10 versions of Slf4j

As for Slf4j (Table 26), none of the models is statistically better than the others. However, we decided to choose Random Forest as the best model as it has the highest accuracy.

6 Validation

In this section, the models which achieved the best performances are validated. Subsection 5.1 contains the results obtained on the dataset at class and package level for the 60 projects. Subsection 5.2 contains the results regarding the 10 versions of the 10 projects.
In order to calculate the model performances, the following metrics were used:

6.1 Performance at class and package level for the 60 projects

The best class-level model turned out to be BayesNet. We then tested this model on the test set of the first partition that was made in our workflow. The accuracy obtained is 0.89. The Figure 1 shows the graph of the recall, precision and F-measure for each class. Table 27 shows the confusion matrix.

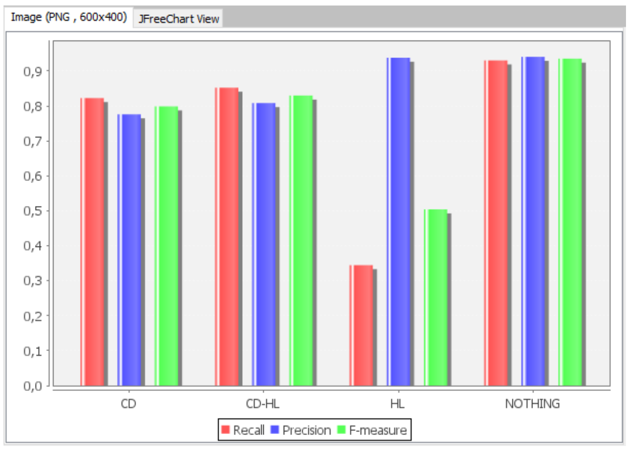

Figure 1: Classes metrics at class level

Actual/Predicted	**Nothing**	**CD-HL**	**CD**	**HL**
Nothing	9048	1933	17	0
CD-HL	2477	33223	4	2
CD	128	27	953	10
HL	3	138	205	182

Table 27: Confusion matrix at class level

Even at the package level, the best model is BayesNet with an accuracy of 0.75. The Figure 2 shows the graph of the recall, precision and F-measure for each class. Table 28 shows the confusion matrix.

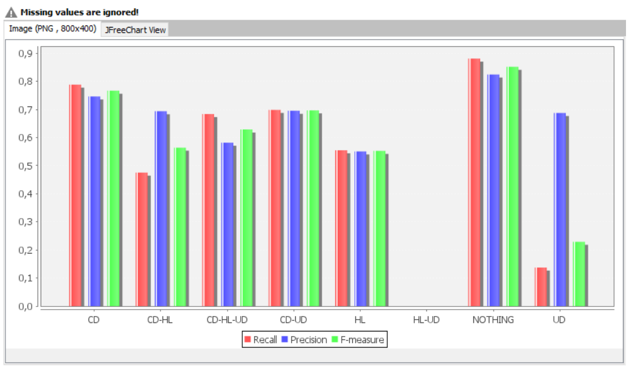

Figure 2: Classes metrics at package level

Actual/Predicted	CD-HL-UD	CD	NOTHING	CD-HL	CD-UD	HL	UD	HL-UD
CD-HL-UD	160	6	8	27	16	17	0	0
CD	0	697	121	3	62	1	0	0
NOTHING	1	101	1474	1	82	8	6	0
CD-HL	33	20	3	77	1	28	0	0
CD-UD	18	101	70	0	454	3	4	0
HL	41	0	17	3	0	76	0	0
UD	0	9	91	0	38	0	22	0
HL-UD	22	0	4	0	0	5	0	0

Table 28: Confusion matrix at package level

6.2 Performance for 10 version of 10 projects

In each subsection below the results of the 10 projects obtained on the test set using the best model are reported.

6.2.1 Ant

The best model for Ant turned out to be IBk. The accuracy obtained is 0.97.

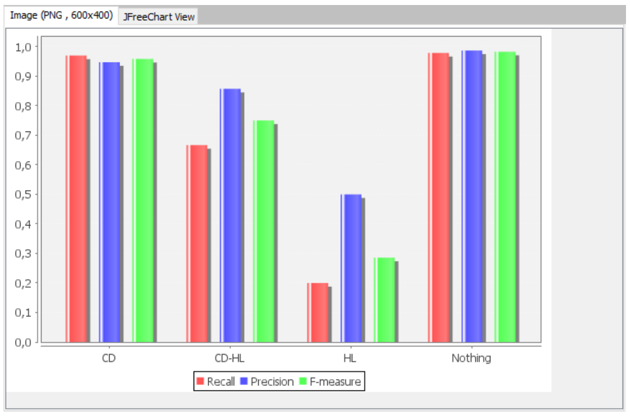

Figure 3: Classes metrics for 10 version of Ant

Actual/Predicted	Nothing	CD	CD-HL	HL
Nothing	4660	101	1	0
CD	58	1923	1	1
CD-HL	0	6	12	0
HL	4	0	0	1

Table 29: Confusion matrix of Ant

6.2.2 Camel

The best model for Camel turned out to be Random Forest. The accuracy obtained is 0.98.

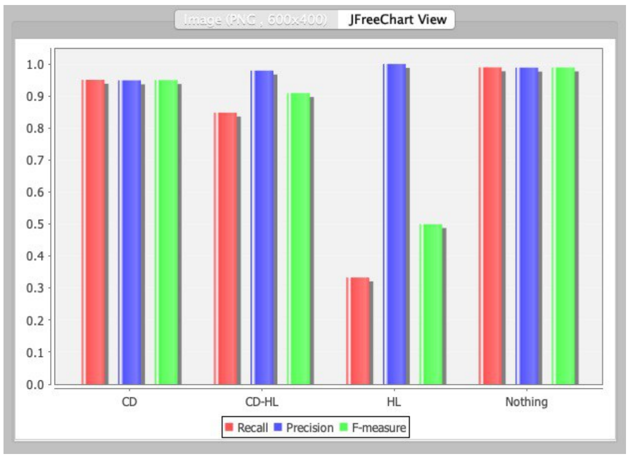

Figure 4: Classes metrics for 10 version of Camel

Actual/Predicted	**Nothing**	**CD**	**CD-HL**	**HL**
Nothing	13354	141	0	0
CD	150	2925	2	0
CD-HL	0	17	95	0
HL	4	0	0	2

Table 30: Confusion matrix of Camel

6.2.3 Clojure

The best model for Clojure turned out to be Random Forest. The accuracy obtained is 0.99.

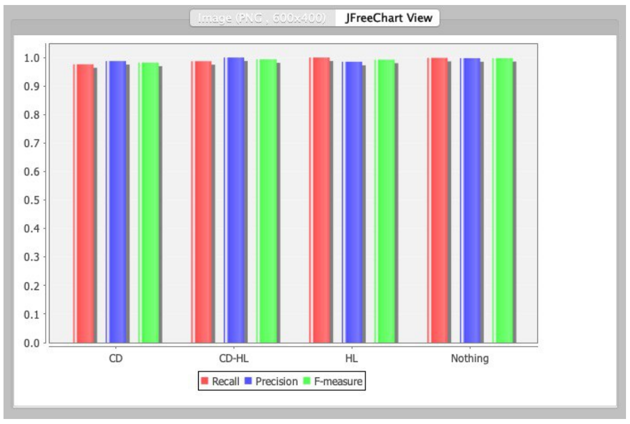

Figure 5: Classes metrics for 10 version of Clojure

Actual/Predicted	**Nothing**	**HL**	**CD**	**CD-HL**
Nothing	11457	0	15	0
HL	0	131	0	0
CD	29	0	1194	0
CD-HL	0	2	0	154

Table 31: Confusion Matrix of Clojure

6.2.4 Junit

The best model for Junit turned out to be Random Forest. The accuracy obtained is 0.98.

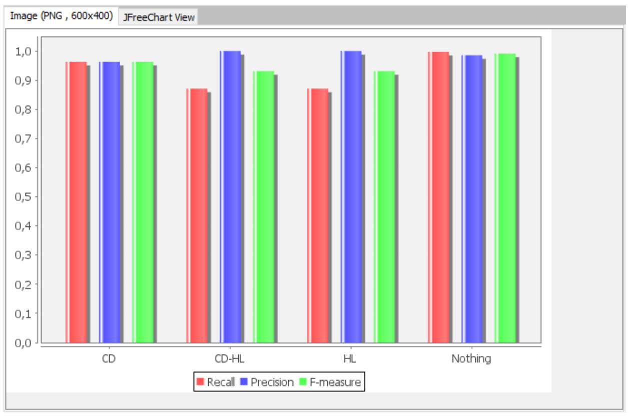

Figure 6: Classes metrics for 10 version of Junit

Actual/Predicted	CD-HL	CD	NOTHING	HL
CD-HL	27	4	0	0
CD	0	156	6	0
NOTHING	0	2	685	0
HL	0	0	4	27

Table 32: Confusion matrix of Junit

6.2.5 Log4j

The best model for Log4j turned out to be Random Forest. The accuracy obtained is 0.98.

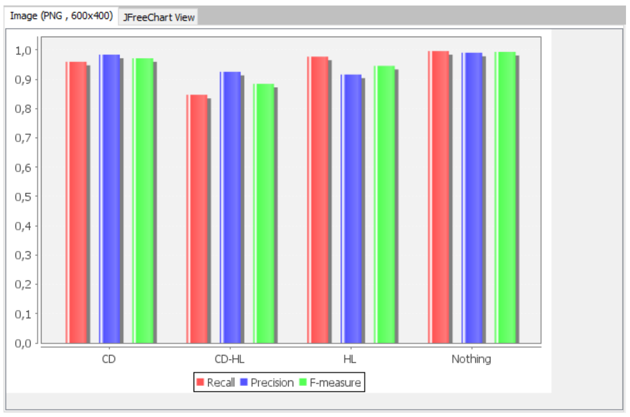

Figure 7: Classes metrics for 10 version of Log4j

Actual/Predicted	Nothing	CD	CD-HL	HL
Nothing	3007	0	9	1
CD	0	50	2	7
CD-HL	25	4	701	0
HL	2	0	0	88

Table 33: Confusion matrix of Log4j

6.2.6 Logback

The best model for Logback turned out to be BayesNet. The accuracy obtained is 0.98.

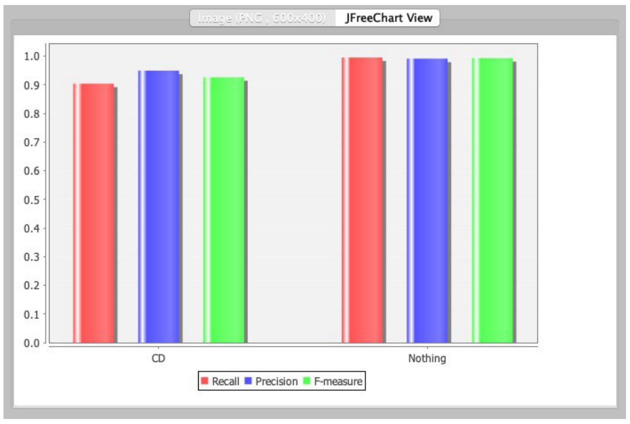

Figure 8: Classes metrics for 10 versions of Lokback

Actual/Predicted	**Nothing**	**CD**
Nothing	242	1
CD	2	119

Table 34: Confusion matrix of Logback

6.2.7 Lombok

The best model for Logback turned out to be BayesNet. The accuracy obtained is 0.98.

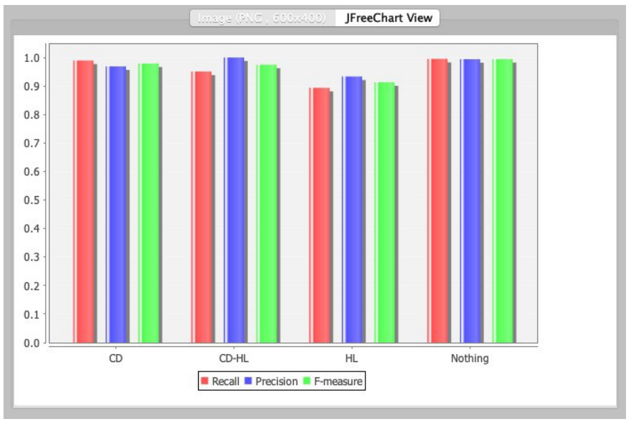

Figure 9: Classes metrics for 10 versions of Lombok

Actual/Predicted	**Nothing**	**CD**	**CD-HL**	**HL**
Nothing	1029	0	4	1
CD	0	58	1	2
CD-HL	2	0	186	0
HL	4	0	1	42

Table 35: Confusion matrix of Logback

6.2.8 Maven

The best model for Maven turned out to be Random Forest. The accuracy obtained is 0.99.

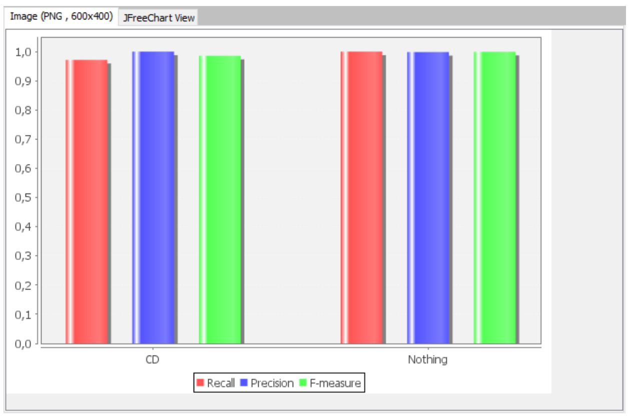

Figure 10: Classes metrics for 10 versios of Maven

Actual/Predicted	**Nothing**	**CD**
Nothing	625	0
CD	1	34

Table 36: Confusion matrix of Maven

6.2.9 Mockito

The best model for Mockito turned out to be Random Forest. The accuracy obtained is 0.99.

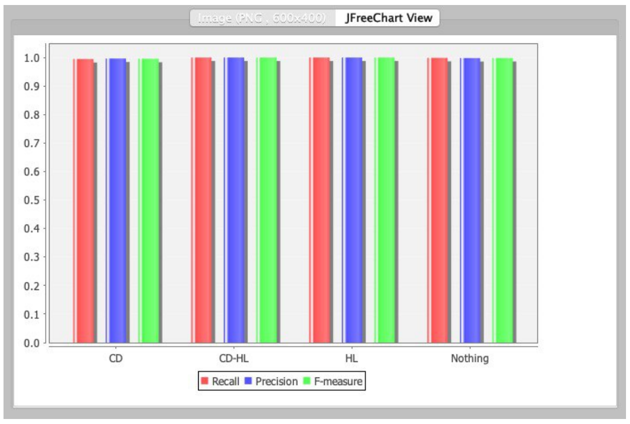

Figure 11: Classes metrics for 10 versions of Mockito

Actual/Predicted	**CD**	**Nothing**	**CD-HL**	**HL**
CD	571	3	0	0
Nothing	2	1655	0	0
CD-HL	0	0	92	0
HL	0	0	0	96

Table 37: Confusion Matrix of Mockito

6.2.10 Slf4j

The best model for Slf4j turned out to be Random Forest. The accuracy obtained is 0.76.

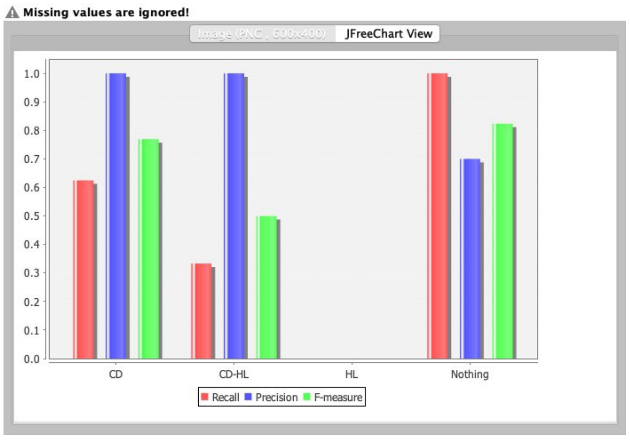

Figure 12: Classes metrics for 10 versions of Slf4j

Actual/Predicted	**Nothing**	**CD**	**CD-HL**	**HL**
Nothing	14	0	0	0
CD	3	5	0	0
CD-HL	2	0	1	0
HL	1	0	0	0

Table 38: Confusion matrix of Sfl4j

7 Discussion of the results obtained

In this section, for each dataset analyzed, a summary of the results obtained will be reported.

7.1 Results for 60 projects at class level

For the 60 projects analyzed at class level, the features selection proved to be of little use for BayesNet and IBk, given that the number of initial features obtaining an accuracy almost unchanged with respect to the model built on all the features. Instead, the features selection with Random Forest has been effective since accuracy has increased by 38%, compared to the model with all the features, using "to" as the only features. We can conclude that the "to" features are more relevant than the others to identify one or more ASs on a dependency.
From the results obtained in the final validation phase, it emerged that the Recall of the "HL" class is low: the model therefore more often than not recognizes an "HL" dependency as "CD-HL" or "CD".

7.2 Results for 60 projects at package level

As regards the dataset of the 60 projects at the package level, the features selection has not had positive results, in fact for both Bayes Net and IBk the highest accuracy was obtained considering all the features. With Random Forest the highest accuracy was obtained considering only the "from" feature, increasing by 15%. Comparing the models for this dataset, the best was BayesNet, supported by the T-test from which positive feedback was obtained. Observing the results of the validation it can be noticed that the classifier, despite having had an accuracy of 75%, is not able to distinguish the "HL-UD" dependencies, often classifying them as "CD-HL-UD".
In general it can be said that if the classifier has to discriminate more than one AS on a dependency, it can only partially discriminate them (example: CD-HL is identified only as CD).

7.3 Results for 10 version of 10 projects

The datasets of the 10 versions of 10 different projects all report similar results as regards the feature selection: it was not very useful for the purpose of improving the accuracy since the initial accuracy with all the features was already very high. the improvements were limited to 1-3 maximum percentage points, reaching an accuracy of 1 in more cases.
As for the machine learning models chosen for each of the 10 projects, the best in seven out of ten cases was Random Forest, followed by two cases in which the best turned out to be BayesNet, while in one case IBk was chosen. In some cases, however, it was not possible to establish whether there was a difference between the various models since according to the t-test there was no statistically significant difference. In those cases, however, we decided to use the model that had obtained the highest accuracy in the training step.
The accuracy obtained during the final validation phase is very high. So it can be said that if an AS is present in one version, with high probability, it will continue over time in the various subsequent versions.

8 Conclusions

The aim of this project was to use the dependency features, extracted from various projects with Arcan, to predict the architectural smells. the aim of this project was to use the dependency features, extracted from various projects with Arcan, to predict the architectural smells. Our analysis first focused on 60 versions of 60 different projects, at the class and package level. We then analyzed 10 projects each consisting of 10 versions.
For the 60 projects analyzed at class level, the presence or absence of an AS can be predicted with high accuracy by looking at the weight associated with the edge of dependence, the starting node and the arrival node.
While for the 60 projects at the package level we have achieved not very high accuracy. This is due to the fact that considering the projects to a greater granularity, the level of detail that is had at class level is lost with a consequent loss of information. So multiple Architectural Smells are incorporated into the same packages and predicting them becomes a more difficult task.
At last for the 10 versions of the 10 projects, given that we have achieved very high performance, we can conclude that the presence of architectural smells in the project's history allows us to predict the presence of architectural smells in the future. The 2-version sliding window approach, with the inclusion of dependency starting and arrival class metrics, has therefore proved to be an effective method for predicting AS in the future.

Appendices

A Adopted metrics

The metrics used to evaluate the performances are reported in this appendix. The Accuracy is the proportion of true results (both true positives and true negatives) among the total number of cases examined. The formula for calculating Accuracy is as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

The Precision is the ability of a classifier not to label a positive instance which is actually negative. For each class it is defined as the relationship between true positives and the sum of true and false positives. The formula for calculating Precision is as follows:

$$Precision = \frac{TP}{TP + FP}$$

The Recall is the ability of a classifier to find all the positive instances. For each class it is defined as the relationship between true positives and the sum of true positives and false negatives.The formula for calculating Recall is as follows:

$$Recall = \frac{TP}{TP + FN}$$

F-measure is a weighted harmonic average of the Precision and Recall metrics such that the best score is 1 and the worst is 0. As a general rule, the weighted average of F1 should be used to compare classifier models, not overall accuracy.The formula for calculating F-measure is as follows:

$$F\text{-}measure = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

B Single sample T-test: comparing classifiers with same test set

Consider two classification models M_1 and M_2 to be compared using k-folds cross validation procedure. The dataset D is partitioned into K disjoint subsets, exhaustive (a partition of D) and with almost a constant number of records $D_1, D_2, \ldots, D_k$.
We then apply classification techniques to construct classification model M_1 and M_2 from $K-1$ of the partitions and test them on the remaining partition. This step is repeated K times each time using a different partition as the test set. Let:

- $M_{1,k}$ inducer for the model M_1 obtained at the k-th iteration ($k = 1, \ldots, K$)
- $M_{2,k}$ inducer for the model M_2 obtained at the k-th iteration ($k = 1, \ldots, K$)

Each pair of models $M_{1,k}$ and $M_{2,k}$ are tested on the same partition k.
Let:

- $e_{1,k}$ error rate of $M_{1,k}$
- $e_{2,k}$ error rate of $M_{2,k}$

The difference between their error rates during the k-th iteration (fold) can be written as:

$$d_k = e_{1,k} - e_{2,k}$$

If K is sufficiently large, then d_k is normally distribuited with:

$$\text{mean} = d_t^{cv}$$

$$\text{standard deviation} = \sigma^{cv}$$

The overall variance in the observed differences is estimated using the following formula:

$$\sigma^2_{d_{cv}} = \frac{\sum_{k=1}^{K}(d_k - \overline{d})^2}{K \cdot (K-1)}$$

Where:

$$\overline{d} = \frac{1}{K}\sum_{k=1}^{K} d_k$$

We use the student t distribution to compute the confidence interval for the value of the true mean d_t^{cv}:

$$\left(\overline{d} - t_{1-\frac{\alpha}{2}}^{k-1} \cdot \sigma_{d_{cv}}, \overline{d} + t_{1-\frac{\alpha}{2}}^{k-1} \cdot \sigma_{d_{cv}}\right)$$

Where $t_{1-\frac{\alpha}{2}}^{k-1}$ is obtained from a probability table, we get the quantile associated with confidence $1 - \alpha$ and $K - 1$ degrees of freedom.
Concerning the use of the confidence interval to establish statistical significance, the same considerations made for the case of 2 classifiers apply here. If the confidence interval spans the value 0, we conclude that the observed difference is not statistically significat at level α (confidence $1 - \alpha$).

References

[1] Joshua Garcia, Daniel Popescu, George Edwards, and Nenad Medvidovic. Identifying architectural bad smells. In *2009 13th European Conference on Software Maintenance and Reengineering*, pages 255–258. IEEE, 2009.

[2] Martin Lippert and Stephen Roock. *Refactoring in large software projects: performing complex restructurings successfully.* John Wiley & Sons, 2006.

[3] Isela Macia, Roberta Arcoverde, Elder Cirilo, Alessandro Garcia, and Arndt von Staa. Supporting the identification of architecturally-relevant code anomalies. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 662–665. IEEE, 2012.

[4] Michael R Berthold, Nicolas Cebron, Fabian Dill, Thomas R Gabriel, Tobias Kötter, Thorsten Meinl, Peter Ohl, Kilian Thiel, and Bernd Wiswedel. Knime-the konstanz information miner: version 2.0 and beyond. *AcM SIGKDD explorations Newsletter*, 11(1):26–31, 2009.

[5] Ewan Tempero, Craig Anslow, Jens Dietrich, Ted Han, Jing Li, Markus Lumpe, Hayden Melton, and James Noble. Qualitas corpus: A curated collection of java code for empirical studies. In *2010 Asia Pacific Software Engineering Conference (APSEC2010)*, pages 336–345, December 2010.

[6] Francesca Arcelli Fontana, Ilaria Pigazzini, Riccardo Roveda, Damian Tamburri, Marco Zanoni, and Elisabetta Di Nitto. Arcan: A tool for architectural smells detection. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 282–285. IEEE, 2017.

[7] Remco R Bouckaert. Bayesian network classifiers in weka for version 3-5-7. *Artificial Intelligence Tools*, 11(3):369–387, 2008.

[8] Don Coppersmith, Se June Hong, and Jonathan RM Hosking. Partitioning nominal attributes in decision trees. *Data Mining and Knowledge Discovery*, 3(2):197–217, 1999.

[9] D. Aha and D. Kibler. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.