Feature-Based Software Design Pattern Detection

Based on the research by Najam Nazar, Aldeida Aleti, and Yaokun Zheng

Prepared by: Mohsen Elahifard

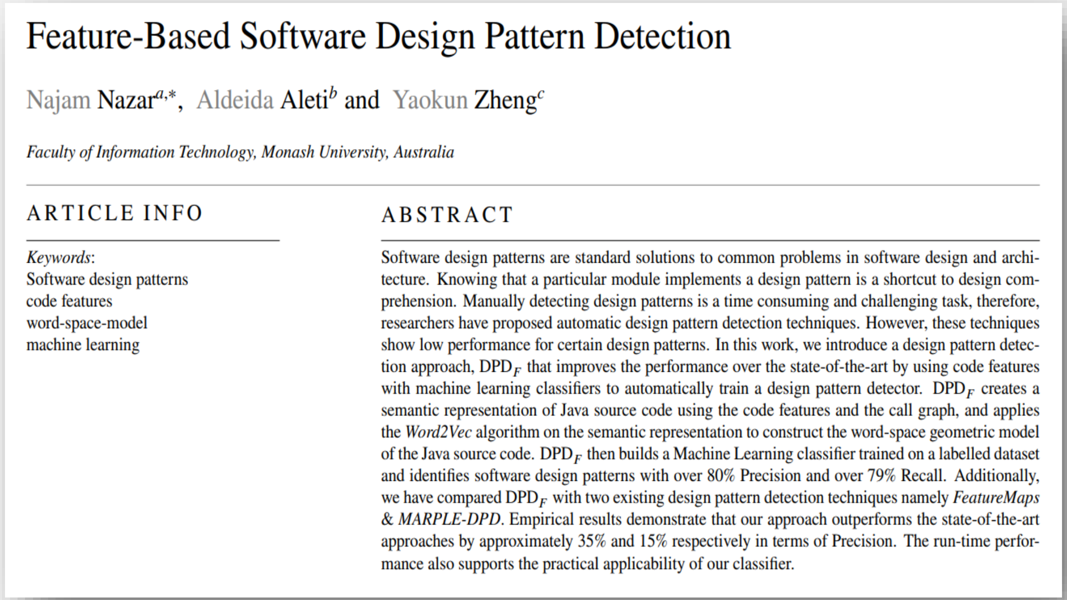
Course: Principles and Patterns in Software Engineering
Instructor: Dr. Morteza Zakeri-Nasrabadi

December 2025

Amirkabir University of Technology
(Tehran Polytechnic)

About the Paper

- Title: Feature-Based Software Design Pattern Detection
- Authors: Najam Nazar, Aldeida Aleti, Yaokun Zheng
- Published in: Journal of Systems and Software, Volume 185, March 2022, Article 111179



Feature-Based Software Design Pattern Detection

Najam Nazar[a,*], Aldeida Aleti[b] and Yaokun Zheng[c]

Faculty of Information Technology, Monash University, Australia

ARTICLE INFO

Keywords:
Software design patterns
code features
word-space-model
machine learning

ABSTRACT

Software design patterns are standard solutions to common problems in software design and architecture. Knowing that a particular module implements a design pattern is a shortcut to design comprehension. Manually detecting design patterns is a time consuming and challenging task, therefore, researchers have proposed automatic design pattern detection techniques. However, these techniques show low performance for certain design patterns. In this work, we introduce a design pattern detection approach, DPD_F that improves the performance over the state-of-the-art by using code features with machine learning classifiers to automatically train a design pattern detector. DPD_F creates a semantic representation of Java source code using the code features and the call graph, and applies the *Word2Vec* algorithm on the semantic representation to construct the word-space geometric model of the Java source code. DPD_F then builds a Machine Learning classifier trained on a labelled dataset and identifies software design patterns with over 80% Precision and over 79% Recall. Additionally, we have compared DPD_F with two existing design pattern detection techniques namely *FeatureMaps* & *MARPLE-DPD*. Empirical results demonstrate that our approach outperforms the state-of-the-art approaches by approximately 35% and 15% respectively in terms of Precision. The run-time performance also supports the practical applicability of our classifier.

Abstract

- Software design patterns provide standard solutions to common design problems and accurate identification of design patterns helps in design comprehension, but manually detecting them is challenging and time-consuming. Existing automatic detection methods often perform poorly for certain patterns.
- This work proposes DPD_F, a feature-based approach for Java code that uses code features, call graphs, and Word2Vec embeddings to train a machine learning classifier. DPD_F achieves over 80% precision and 79% recall, outperforming previous methods like FeatureMaps and MARPLE-DPD, while its runtime performance supports practical applicability.

Introduction: Design Patterns (Importance & Challenges)

- Design pattern detection is an active research field and has gained enormous attention in recent years.
- Software design patterns are recurring solutions to common problems, widely adopted to improve software quality, reuse, and refactoring.
- Recognizing that a module implements a design pattern helps software maintenance.
- Manual detection is challenging and time-consuming due to increasing software complexity and varied coding styles.
- Automatic detection methods exist but often perform poorly in identifying certain patterns.
- Capturing semantic and lexical information from source code is difficult and not fully exploited yet.

Introduction: Proposed Approach – DPD_F

- DPD_F (Feature-Based Design Pattern Detection) uses 15 code features (structural + lexical).
- Builds a call graph and generates Software Syntactic and Lexical Representation (SSLR) of Java source code.
- Applies Word2Vec to construct a word-space geometric model.
- Trains a machine learning classifier to detect 12 commonly used GoF design patterns.

Introduction: Corpus, Evaluation & Key Contributions

- DPD_F Corpus: 1,300 Java files labeled by experts using CodeLabeller.
- Evaluated using Precision, Recall, F1-Score.
- Outperforms state-of-the-art approaches FeatureMaps & MARPLE-DPD (35% & 15% higher Precision).
- The paper introduces DPD_F, which combines code features to improve detection accuracy, and demonstrates superior performance and practical applicability over existing methods.

Preliminaries: Design Patterns

- Focuses on 12 GoF patterns, which covers all three categories:
  - Creational: Builder, Abstract Factory, Factory Method, Prototype, Singleton
  - Structural: Adapter, Decorator, Facade, Proxy
  - Behavioral: Memento, Observer, Visitor
- Patterns analyzed using the proposed DPD_F approach.

Preliminaries: Code Features

- Code features are static attributes of source code (structural & lexical), e.g., class/method names, inheritance, lines of code, complexity (if/else blocks), object-oriented attributes.
- Features capture behavioral, structural, and creational aspects of code.
- Used to identify design patterns more accurately than low-level entities.

Preliminaries: Word Embeddings

- Word-space models encode meaning of words as vectors in high-dimensional space.
- Built using co-occurrence statistics or predictive models like Word2Vec.
- Key principle: Distributional Hypothesis – words in similar contexts have similar meanings.
- Enables measuring semantic similarity between code constructs (e.g., class or method names).
- Word embeddings proven effective in NLP and source code analysis.

Preliminaries: Machine Learning

- In machine learning, systems learn from data and improve automatically.
- The types of machine learning algorithms differ in their approach, the form of input/output data, and the task or problem they aim to solve.
- Major types: Supervised, Unsupervised, Semi-supervised.
- Supervised ML used in this study for classification:
  - Train classifiers on labeled data
  - Predict whether a class contains a design pattern
- Classifiers used: Random Forest (RF) and Support Vector Machines (SVM).
- Each class in code labeled as pattern-containing or not.
- DPD_F classifier builds a model to classify classes based on extracted code features and embeddings, which enables automatic, accurate detection of software design patterns in Java source code.

Study Design: Research Questions

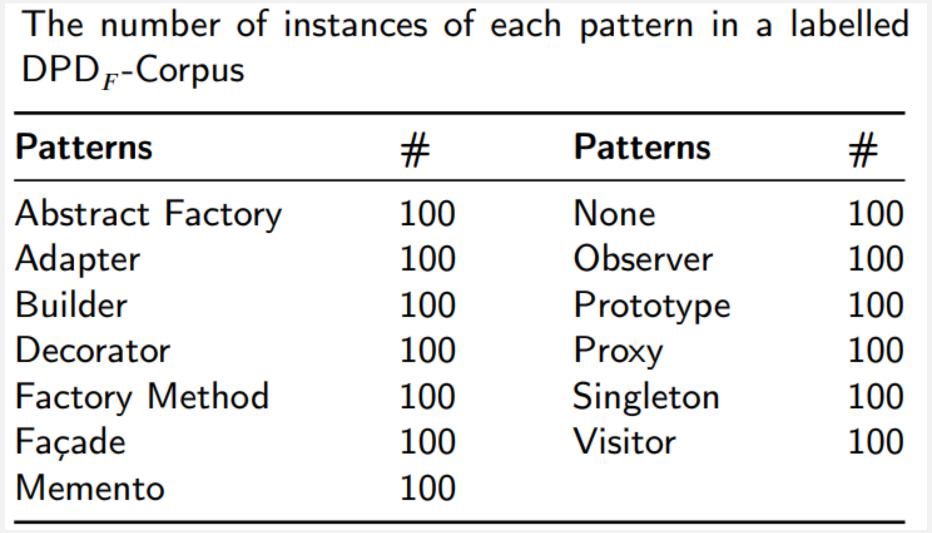
- This study investigates the relationship between extracted code features and software design patterns using Word2Vec and supervised ML.
- RQ1: Is DPD_F effective in detecting software design patterns?
  - Evaluation through Precision, Recall, and F1-Score.
- RQ2: What is the error rate of DPD_F?
  - Analyzing misclassification through confusion matrix.
- RQ3: How well does DPD_F perform compared to existing approaches?
  - Benchmark against FeatureMaps (2019) and MARPLE-DPD (2015).

Study Design: Methodology

- Goal: Automatically detect design patterns from Java source code.
- Steps:
    - Collect Java source dataset (corpus)
    - Extract code features → semantic–syntactic representation (SSLR)
    - Apply Word2Vec → geometric embedding model
    - Train supervised classifiers → detect design pattern instances

Study Design: Corpus Construction – DPD_F-Corpus

- Public datasets were insufficient or limited.
- A new labeled dataset (DPD_F-Corpus) was built.
- Derived from GitHub Java Corpus (14,436 projects → 2M+ Java files).
- Filtered irrelevant files (tests, UI assets, etc.).
- Final dataset: 1,300 Java files.
- Ensures balanced data for ML training/testing.

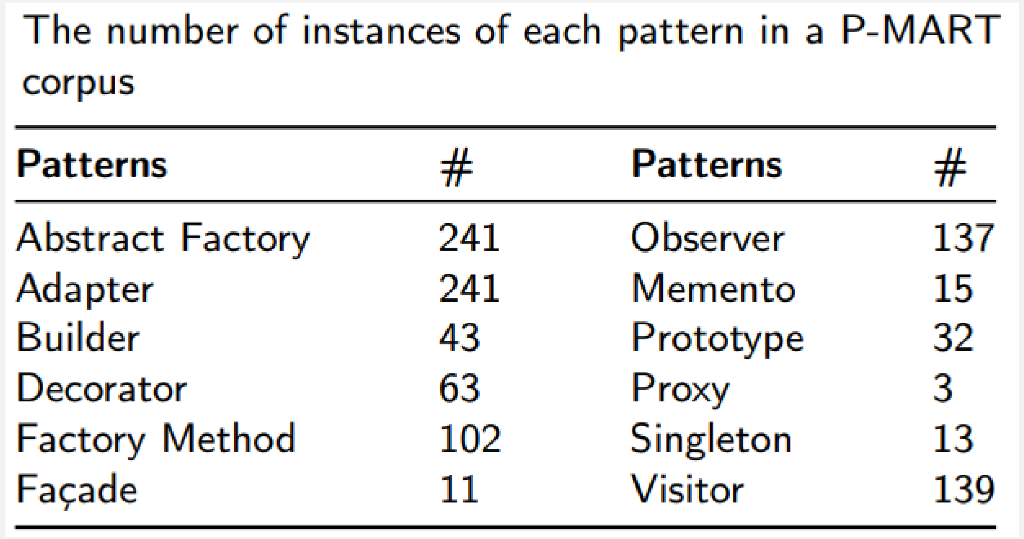


The number of instances of each pattern in a labelled DPD_F-Corpus

Patterns	#	**Patterns**	#
Abstract Factory	100	None	100
Adapter	100	Observer	100
Builder	100	Prototype	100
Decorator	100	Proxy	100
Factory Method	100	Singleton	100
Façade	100	Visitor	100
Memento	100		

Study Design: Benchmark Corpus — P-MART

- P-MART used to compare against existing studies.
- 4,242 files from 9 known Java projects: QuickUML, Lexi, … .
- Contains uneven pattern distribution.
- 1,039 files relevant to 12 target patterns used for benchmarking.
- Includes well-studied domains → supports fair comparison.



The number of instances of each pattern in a P-MART corpus

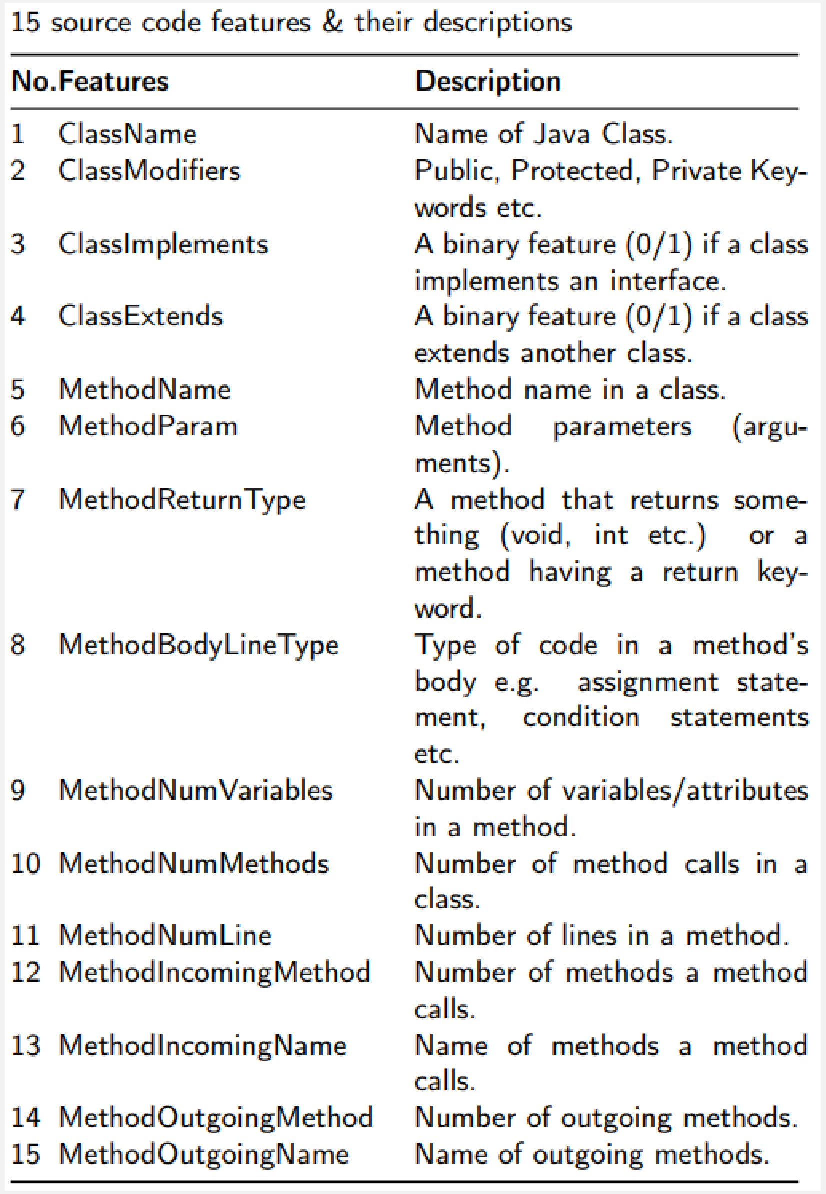
Patterns	#	Patterns	#
Abstract Factory	241	Observer	137
Adapter	241	Memento	15
Builder	43	Prototype	32
Decorator	63	Proxy	3
Factory Method	102	Singleton	13
Façade	11	Visitor	139

Study Design: Data Labeling & Rater Agreement

- Annotation using CodeLabeller tool.
  - Annotators: ≥2 years Java + software engineering experience.
  - Each file labeled by at least three experts.
  - Includes None-label files to evaluate false positives.
- Cohen’s Kappa for DPD_F-Corpus: 0.74 → medium-to-high agreement reliability.

Study Design: Feature Extractions

- Feature extraction converts raw source code into structured, analyzable information. Design patterns rely on class structures, relationships, and communication flows. Extracted features must capture these relationships to enable pattern detection.
- Extracted features are both semantic and syntactic.

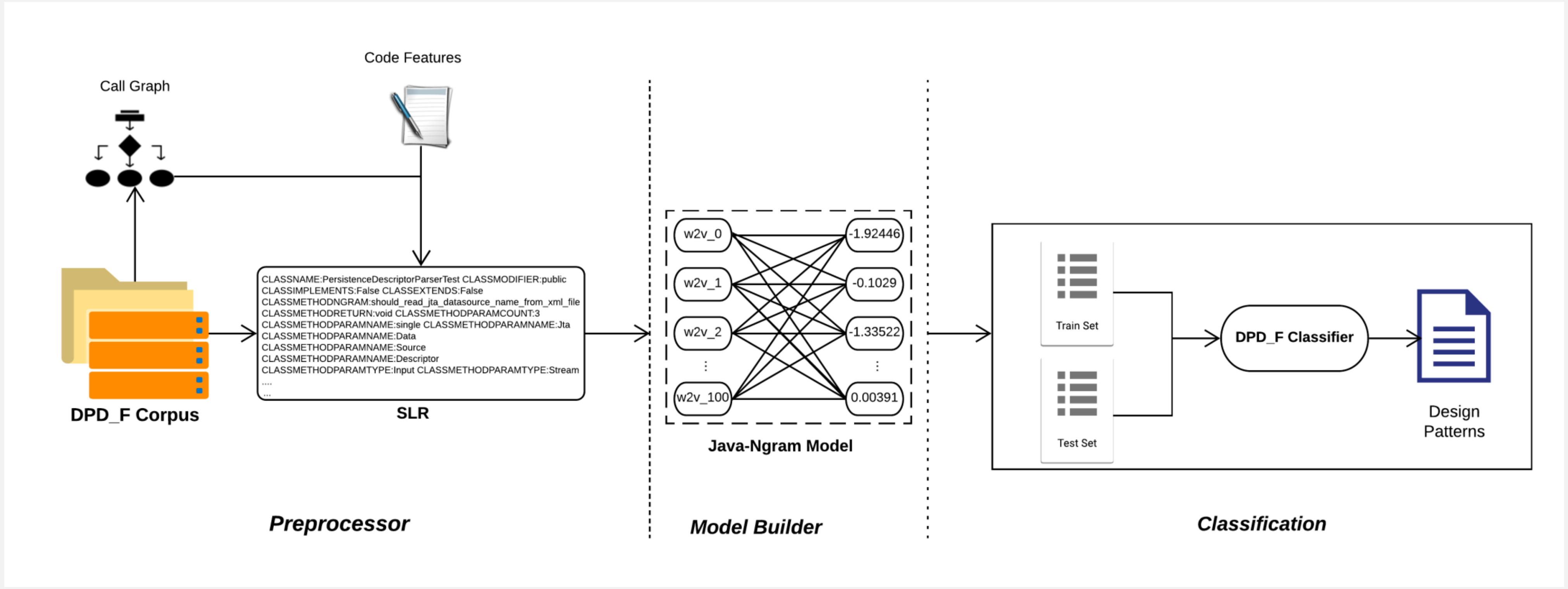


15 source code features & their descriptions

No.	Features	Description
1	ClassName	Name of Java Class.
2	ClassModifiers	Public, Protected, Private Keywords etc.
3	ClassImplements	A binary feature (0/1) if a class implements an interface.
4	ClassExtends	A binary feature (0/1) if a class extends another class.
5	MethodName	Method name in a class.
6	MethodParam	Method parameters (arguments).
7	MethodReturnType	A method that returns something (void, int etc.) or a method having a return keyword.
8	MethodBodyLineType	Type of code in a method's body e.g. assignment statement, condition statements etc.
9	MethodNumVariables	Number of variables/attributes in a method.
10	MethodNumMethods	Number of method calls in a class.
11	MethodNumLine	Number of lines in a method.
12	MethodIncomingMethod	Number of methods a method calls.
13	MethodIncomingName	Name of methods a method calls.
14	MethodOutgoingMethod	Number of outgoing methods.
15	MethodOutgoingName	Name of outgoing methods.

Study Design: DPD$_F$

- DPD$_F$ consists of three main phases: Preprocessing, Model Building, and Machine-Learning Classification.



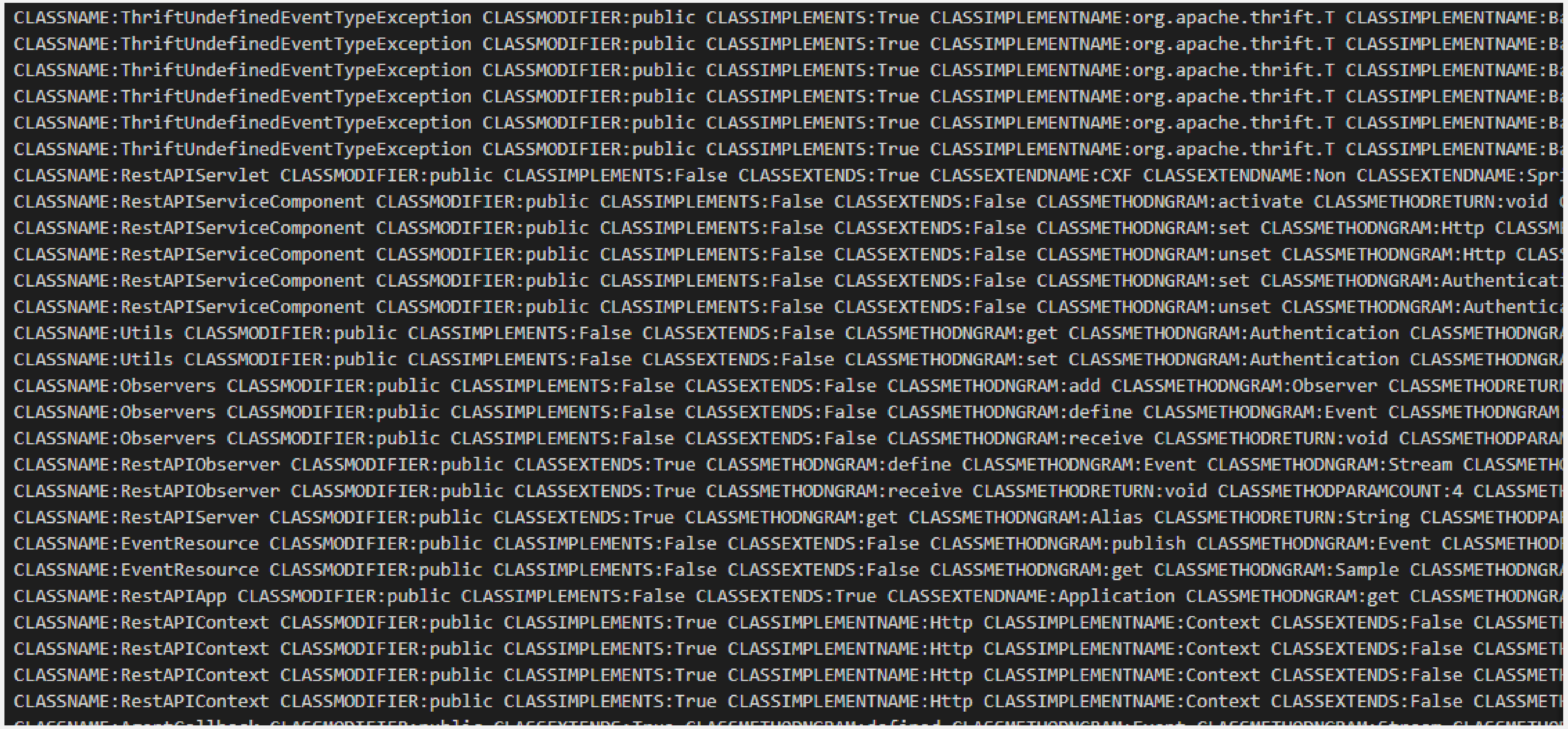
```
CLASSNAME:PersistenceDescriptorParserTest CLASSMODIFIER:public
CLASSIMPLEMENTS:False CLASSEXTENDS:False
CLASSMETHODNGRAM:should_read_jta_datasource_name_from_xml_file
CLASSMETHODRETURN:void CLASSMETHODPARAMCOUNT:3
CLASSMETHODPARAMNAME:single CLASSMETHODPARAMNAME:Jta
CLASSMETHODPARAMNAME:Data
CLASSMETHODPARAMNAME:Source
CLASSMETHODPARAMNAME:Descriptor
CLASSMETHODPARAMTYPE:Input CLASSMETHODPARAMTYPE:Stream
....
...
```

Study Design: Preprocessing

- Treating code as plain text loses execution and relation information; SCG restores caller–callee context.
- SCG better captures behavioral/creational aspects of patterns compared to AST alone.
- SSLR merges syntactic (signatures, modifiers, inheritance) and semantic (interactions, roles) information. It encodes classes, methods, interfaces, relationships, and call flows as sentence-like tokens. It includes extracted features serialized into natural language lines.
- Example item: “Class A implements Interface I; Method m calls n(); Method m has 3 variables.”.
- *Parser & call-graph generator implemented in Python using Plyj (Java7 parser).*

Study Design: Preprocessing (cont.)

- Here is an example (subset) of the SSLR file:



```
CLASSNAME:ThriftUndefinedEventTypeException CLASSMODIFIER:public CLASSIMPLEMENTS:True CLASSIMPLEMENTNAME:org.apache.thrift.T CLASSIMPLEMENTNAME:Ba
CLASSNAME:ThriftUndefinedEventTypeException CLASSMODIFIER:public CLASSIMPLEMENTS:True CLASSIMPLEMENTNAME:org.apache.thrift.T CLASSIMPLEMENTNAME:Ba
CLASSNAME:ThriftUndefinedEventTypeException CLASSMODIFIER:public CLASSIMPLEMENTS:True CLASSIMPLEMENTNAME:org.apache.thrift.T CLASSIMPLEMENTNAME:Ba
CLASSNAME:ThriftUndefinedEventTypeException CLASSMODIFIER:public CLASSIMPLEMENTS:True CLASSIMPLEMENTNAME:org.apache.thrift.T CLASSIMPLEMENTNAME:Ba
CLASSNAME:ThriftUndefinedEventTypeException CLASSMODIFIER:public CLASSIMPLEMENTS:True CLASSIMPLEMENTNAME:org.apache.thrift.T CLASSIMPLEMENTNAME:Ba
CLASSNAME:ThriftUndefinedEventTypeException CLASSMODIFIER:public CLASSIMPLEMENTS:True CLASSIMPLEMENTNAME:org.apache.thrift.T CLASSIMPLEMENTNAME:Ba
CLASSNAME:RestAPIServlet CLASSMODIFIER:public CLASSIMPLEMENTS:False CLASSEXTENDS:True CLASSEXTENDNAME:CXF CLASSEXTENDNAME:Non CLASSEXTENDNAME:Spri
CLASSNAME:RestAPIServiceComponent CLASSMODIFIER:public CLASSIMPLEMENTS:False CLASSEXTENDS:False CLASSMETHODNGRAM:activate CLASSMETHODRETURN:void C
CLASSNAME:RestAPIServiceComponent CLASSMODIFIER:public CLASSIMPLEMENTS:False CLASSEXTENDS:False CLASSMETHODNGRAM:set CLASSMETHODNGRAM:Http CLASSME
CLASSNAME:RestAPIServiceComponent CLASSMODIFIER:public CLASSIMPLEMENTS:False CLASSEXTENDS:False CLASSMETHODNGRAM:unset CLASSMETHODNGRAM:Http CLASS
CLASSNAME:RestAPIServiceComponent CLASSMODIFIER:public CLASSIMPLEMENTS:False CLASSEXTENDS:False CLASSMETHODNGRAM:set CLASSMETHODNGRAM:Authenticat
CLASSNAME:RestAPIServiceComponent CLASSMODIFIER:public CLASSIMPLEMENTS:False CLASSEXTENDS:False CLASSMETHODNGRAM:unset CLASSMETHODNGRAM:Authentica
CLASSNAME:Utils CLASSMODIFIER:public CLASSIMPLEMENTS:False CLASSEXTENDS:False CLASSMETHODNGRAM:get CLASSMETHODNGRAM:Authentication CLASSMETHODNGRA
CLASSNAME:Utils CLASSMODIFIER:public CLASSIMPLEMENTS:False CLASSEXTENDS:False CLASSMETHODNGRAM:set CLASSMETHODNGRAM:Authentication CLASSMETHODNGRA
CLASSNAME:Observers CLASSMODIFIER:public CLASSIMPLEMENTS:False CLASSEXTENDS:False CLASSMETHODNGRAM:add CLASSMETHODNGRAM:Observer CLASSMETHODRETURN
CLASSNAME:Observers CLASSMODIFIER:public CLASSIMPLEMENTS:False CLASSEXTENDS:False CLASSMETHODNGRAM:define CLASSMETHODNGRAM:Event CLASSMETHODNGRAM
CLASSNAME:Observers CLASSMODIFIER:public CLASSIMPLEMENTS:False CLASSEXTENDS:False CLASSMETHODNGRAM:receive CLASSMETHODRETURN:void CLASSMETHODPARAM
CLASSNAME:RestAPIObserver CLASSMODIFIER:public CLASSEXTENDS:True CLASSMETHODNGRAM:define CLASSMETHODNGRAM:Event CLASSMETHODNGRAM:Stream CLASSMETHO
CLASSNAME:RestAPIObserver CLASSMODIFIER:public CLASSEXTENDS:True CLASSMETHODNGRAM:receive CLASSMETHODRETURN:void CLASSMETHODPARAMCOUNT:4 CLASSMETH
CLASSNAME:RestAPIServer CLASSMODIFIER:public CLASSEXTENDS:True CLASSMETHODNGRAM:get CLASSMETHODNGRAM:Alias CLASSMETHODRETURN:String CLASSMETHODPAR
CLASSNAME:EventResource CLASSMODIFIER:public CLASSIMPLEMENTS:False CLASSEXTENDS:False CLASSMETHODNGRAM:publish CLASSMETHODNGRAM:Event CLASSMETHODN
CLASSNAME:EventResource CLASSMODIFIER:public CLASSIMPLEMENTS:False CLASSEXTENDS:False CLASSMETHODNGRAM:get CLASSMETHODNGRAM:Sample CLASSMETHODNGRA
CLASSNAME:RestAPIApp CLASSMODIFIER:public CLASSIMPLEMENTS:False CLASSEXTENDS:True CLASSEXTENDNAME:Application CLASSMETHODNGRAM:get CLASSMETHODNGRA
CLASSNAME:RestAPIContext CLASSMODIFIER:public CLASSIMPLEMENTS:True CLASSIMPLEMENTNAME:Http CLASSIMPLEMENTNAME:Context CLASSEXTENDS:False CLASSMETH
CLASSNAME:RestAPIContext CLASSMODIFIER:public CLASSIMPLEMENTS:True CLASSIMPLEMENTNAME:Http CLASSIMPLEMENTNAME:Context CLASSEXTENDS:False CLASSMETH
CLASSNAME:RestAPIContext CLASSMODIFIER:public CLASSIMPLEMENTS:True CLASSIMPLEMENTNAME:Http CLASSIMPLEMENTNAME:Context CLASSEXTENDS:False CLASSMETH
CLASSNAME:RestAPIContext CLASSMODIFIER:public CLASSIMPLEMENTS:True CLASSIMPLEMENTNAME:Http CLASSIMPLEMENTNAME:Context CLASSEXTENDS:False CLASSMETH
CLASSNAME:AgentCallback CLASSMODIFIER:public CLASSEXTENDS:True CLASSMETHODNGRAM:defined CLASSMETHODNGRAM:Event CLASSMETHODNGRAM:Stream CLASSMETHO
```

Study Design: Model Building

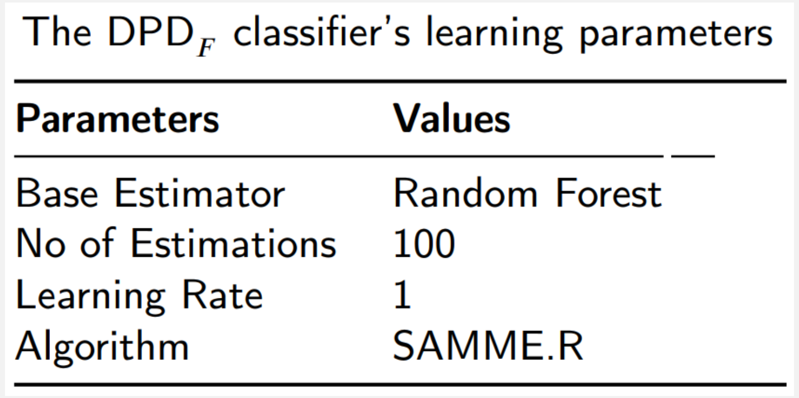
- SSLR files from preprocessing are treated like natural-language documents. Java classes and methods segmented into n-grams, considered as words.
- Word2Vec generates distributed vector embeddings representing n-gram context.
- Two architectures exist: CBOW and Skip-gram.
- Final embedding size = 100 dimensions per n-gram.
- File representation = average of n-gram vectors + concatenation with extracted features.
- Output: Java Embedded Model (JEM) → compact vector representation of each class.

Study Design: Model Building (cont.)

- Despite formal syntax, code names & comments carry natural-language semantics.
- Word embeddings capture similarity between classes/methods and preserve both syntactic structure and semantic relationships learned from context.
- *Implementation of this step is carried out using the Gensim Word2Vec library.*

Study Design: Classification

- Each instance represented by:
  - Project ID + Class name + 100-dimensional embedding + Pattern label.
- Trained for multi-class pattern detection.
- Input: augmented feature vectors + embeddings from Word2Vec step.
- Training method: Stratified K-Fold Cross-Validation (K = 10).
- Stratification keeps pattern distribution balanced in all folds.
- Evaluation with standard metrics
- *Implemented (Random Forest + StratifiedKFold) and evaluated using Scikit-learn .*

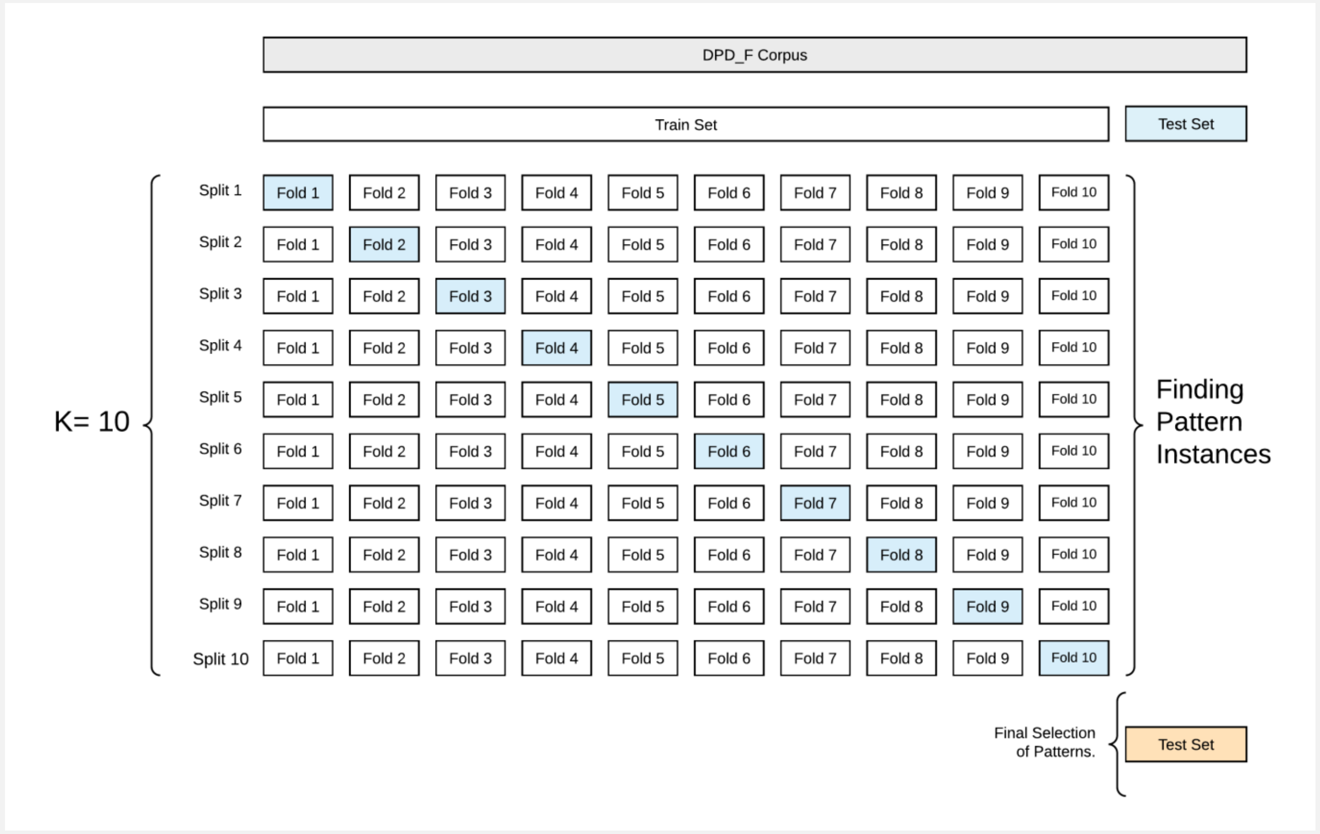


The DPD$_F$ classifier's learning parameters

Parameters	**Values**
Base Estimator	Random Forest
No of Estimations	100
Learning Rate	1
Algorithm	SAMME.R

Study Design: Classification (cont.)

- Here is an illustration of the K-Fold Stratification procedure using 10-fold Cross-Validation with 90/10 train-test splits on the DPD_F corpus:



Results and Evaluations

- We evaluate DPD$_F$ using standard machine learning metrics:

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

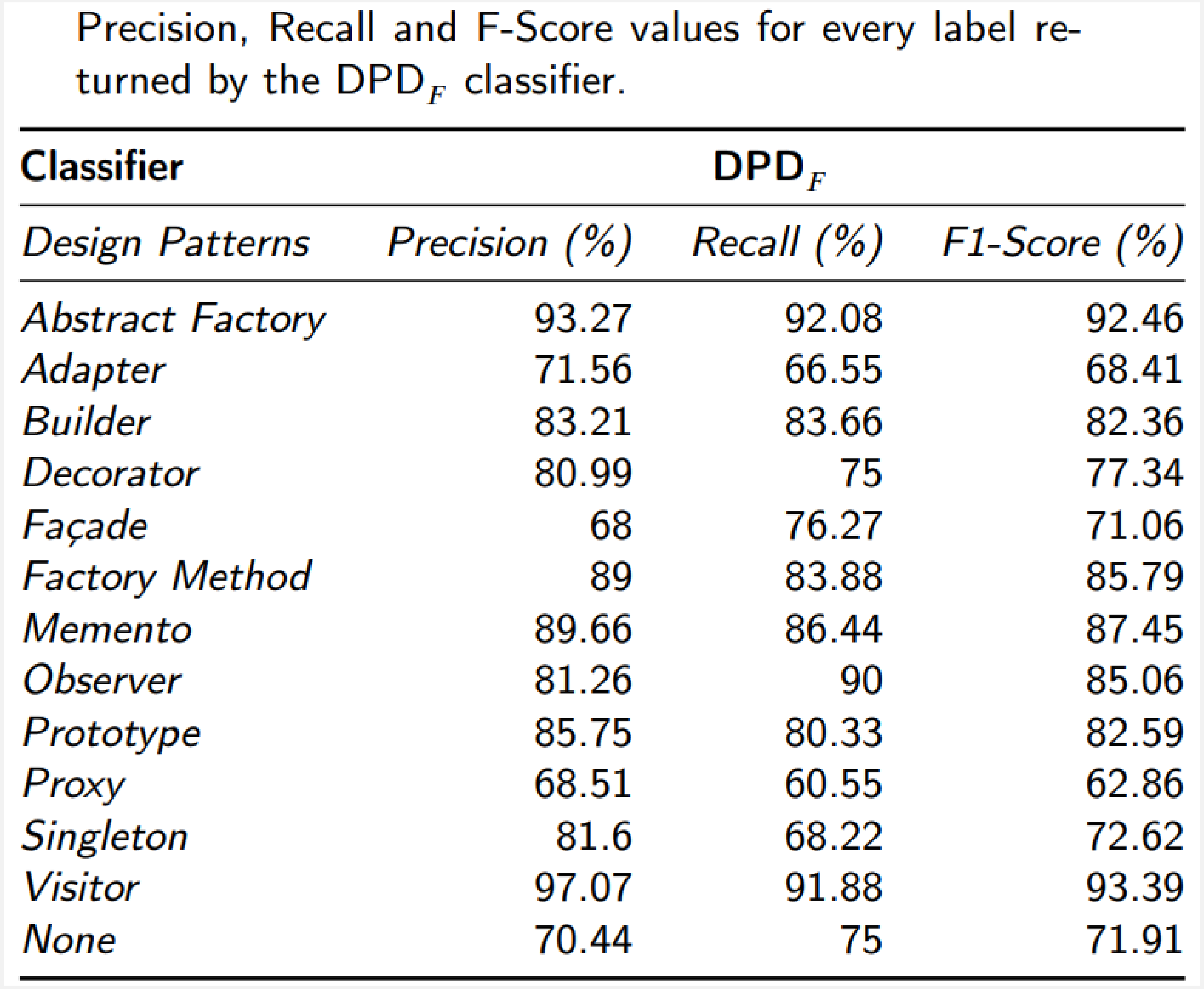
$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{F1} - \text{Score} = \frac{2 \times P \times R}{P + R}$$

Results and Evaluations (cont.)

- RQ1: Is DPD_F effective in detecting software design patterns?
  - Classifier performance remains consistent across runs.
  - Final scores are computed using weighted average across folds.
  - DPD_F achieves 80+% Precision and 79+% Recall overall.
  - Most patterns are accurately identified by the model.
  - Visitor achieves the highest Precision (~97%).
  - Facade & Proxy show lower Precision (~68%), likely due to structural complexity and contextual ambiguity.

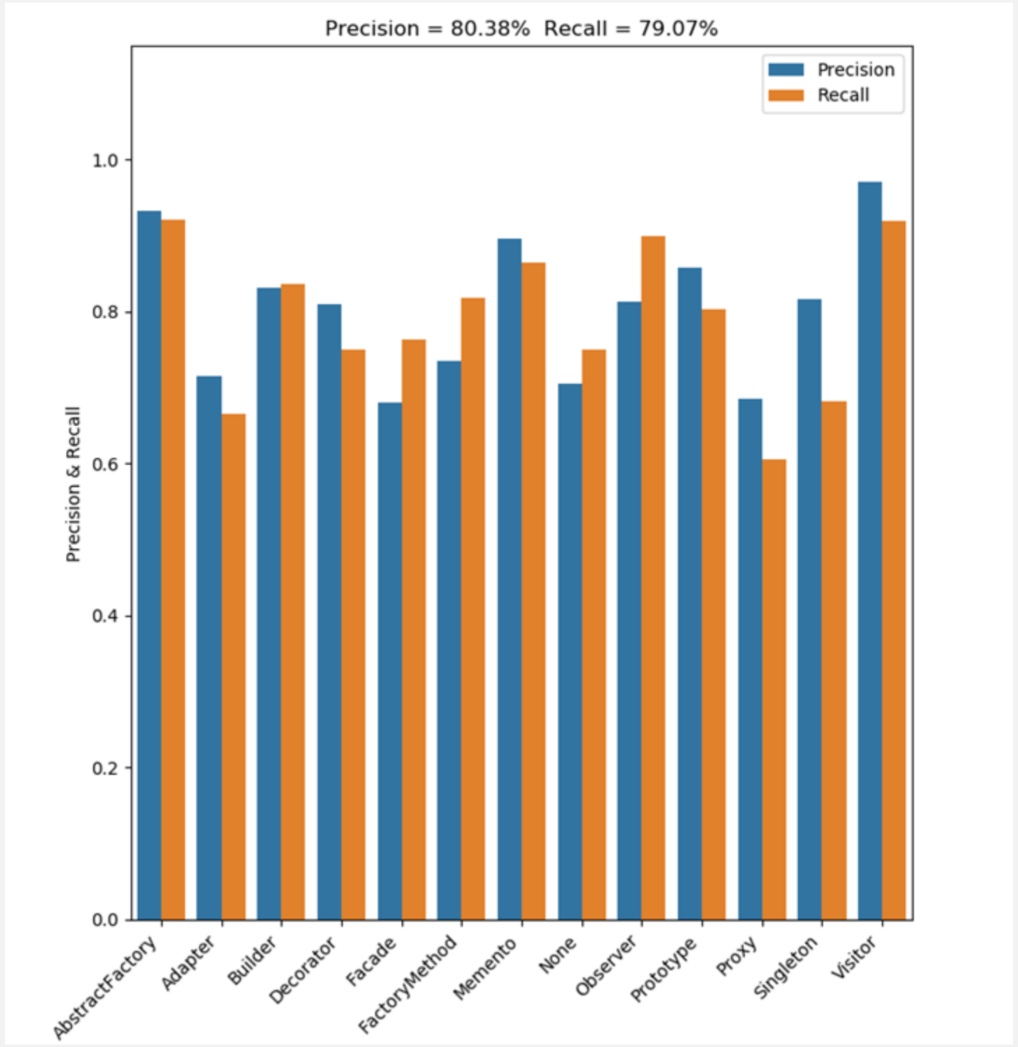
Results and Evaluations (cont.)



Precision, Recall and F-Score values for every label returned by the DPD$_F$ classifier.

Classifier		**DPD$_F$**	
Design Patterns	*Precision (%)*	*Recall (%)*	*F1-Score (%)*
Abstract Factory	93.27	92.08	92.46
Adapter	71.56	66.55	68.41
Builder	83.21	83.66	82.36
Decorator	80.99	75	77.34
Façade	68	76.27	71.06
Factory Method	89	83.88	85.79
Memento	89.66	86.44	87.45
Observer	81.26	90	85.06
Prototype	85.75	80.33	82.59
Proxy	68.51	60.55	62.86
Singleton	81.6	68.22	72.62
Visitor	97.07	91.88	93.39
None	70.44	75	71.91

Results and Evaluations (cont.)

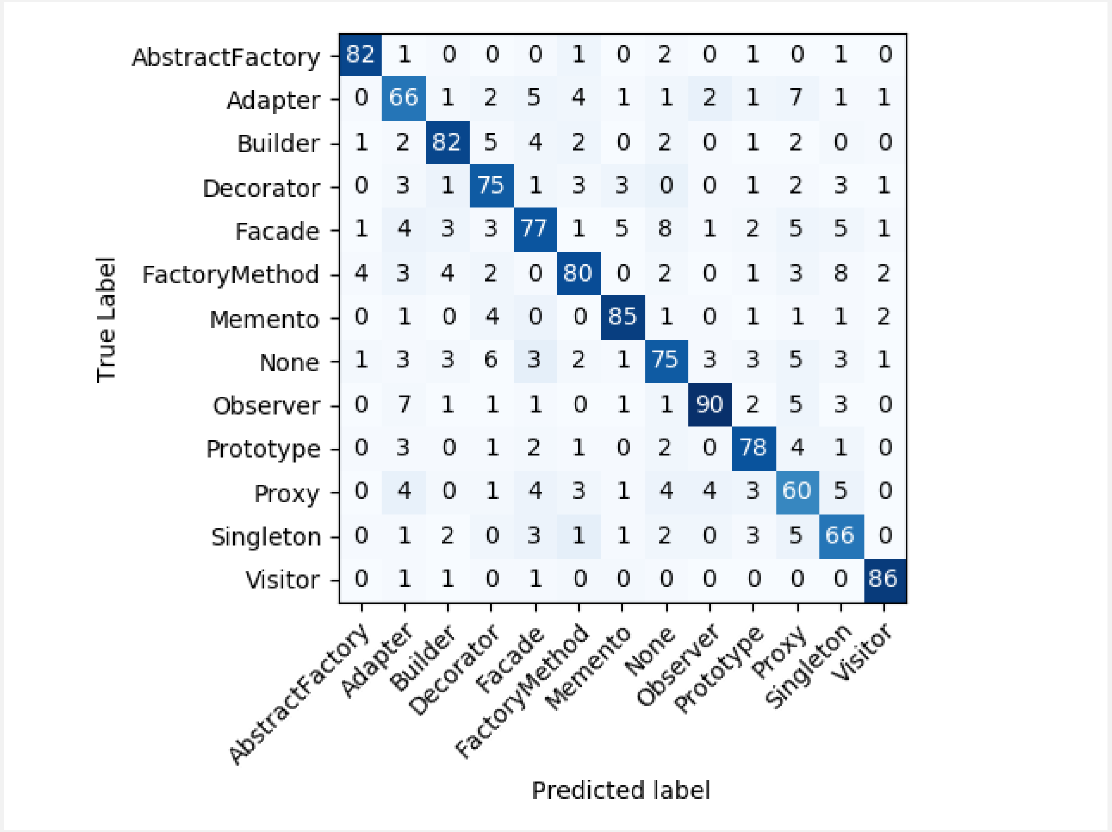


Results and Evaluations (cont.)

- RQ2: What is the error rate of DPD_F?
  - Misclassification rate computed using the confusion matrix.
  - Represents how often pattern instances are incorrectly predicted.
  - DPD_F shows strong correct detection for most labels.
  - Adapter–Observer overlap (7 Observer cases → predicted as Adapter).
  - Lower true-prediction counts in Singleton, Proxy, Adapter.
  - Error rate < 20% across dataset.

Results and Evaluations (cont.)

- Here is the confusion matrix of DPD_F:



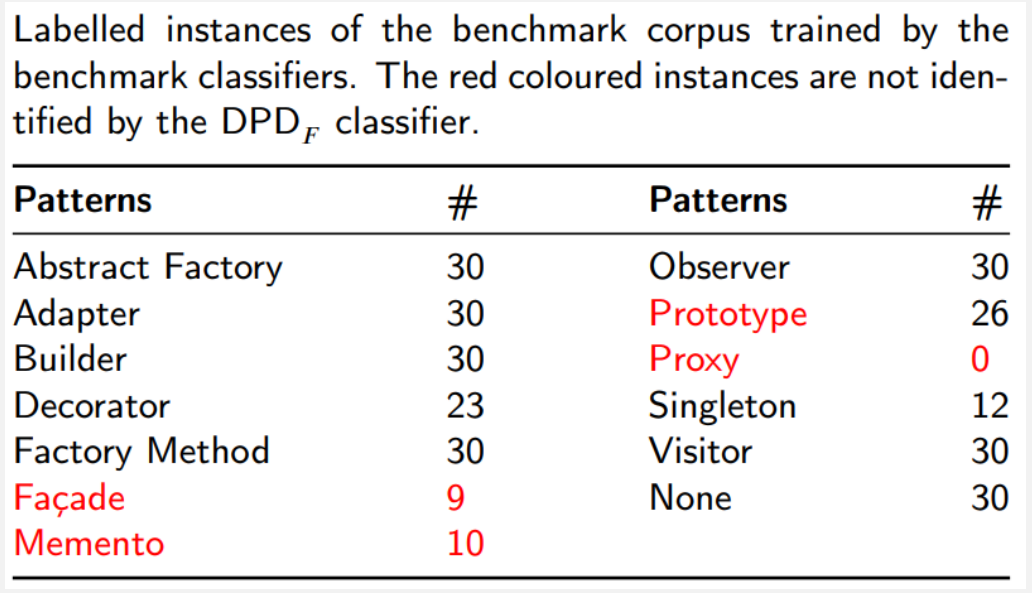
	AbstractFactory	Adapter	Builder	Decorator	Facade	FactoryMethod	Memento	None	Observer	Prototype	Proxy	Singleton	Visitor
AbstractFactory	82	1	0	0	0	1	0	2	0	1	0	1	0
Adapter	0	66	1	2	5	4	1	1	2	1	7	1	1
Builder	1	2	82	5	4	2	0	2	0	1	2	0	0
Decorator	0	3	1	75	1	3	3	0	0	1	2	3	1
Facade	1	4	3	3	77	1	5	8	1	2	5	5	1
FactoryMethod	4	3	4	2	0	80	0	2	0	1	3	8	2
Memento	0	1	0	4	0	0	85	1	0	1	1	1	2
None	1	3	3	6	3	2	1	75	3	3	5	3	1
Observer	0	7	1	1	1	0	1	1	90	2	5	3	0
Prototype	0	3	0	1	2	1	0	2	0	78	4	1	0
Proxy	0	4	0	1	4	3	1	4	4	3	60	5	0
Singleton	0	1	2	0	3	1	1	2	0	3	5	66	0
Visitor	0	1	1	0	1	0	0	0	0	0	0	0	86

Results and Evaluations (cont.)

- RQ3: How well does DPD_F perform compared to existing approaches?
  - Lack of standard public benchmarks makes direct validation difficult.
  - Most prior studies do not release source code or datasets.
  - Approach to comparison:
    - Two relevant DP-detection studies selected: Thaller et al. (2019) & Fontana et al. (2011).
    - Selection based on use of code features, ML classification, or both.
    - Reproduced results manually due to unavailable implementations/datasets.

Results and Evaluations (cont.)

- Benchmarking Strategy:
  - Apply DPD_F model on P-MART corpus → compare against reported results.
  - Apply existing approaches to DPD_F-Corpus → evaluate cross-performance.
- Dataset preparation:
  - 290 Java files extracted & labelled → Labelled P-MART Corpus.


Labelled instances of the benchmark corpus trained by the benchmark classifiers. The red coloured instances are not identified by the DPD_F classifier.

Patterns	#	**Patterns**	#
Abstract Factory	30	Observer	30
Adapter	30	Prototype	26
Builder	30	Proxy	0
Decorator	23	Singleton	12
Factory Method	30	Visitor	30
Façade	9	None	30
Memento	10		

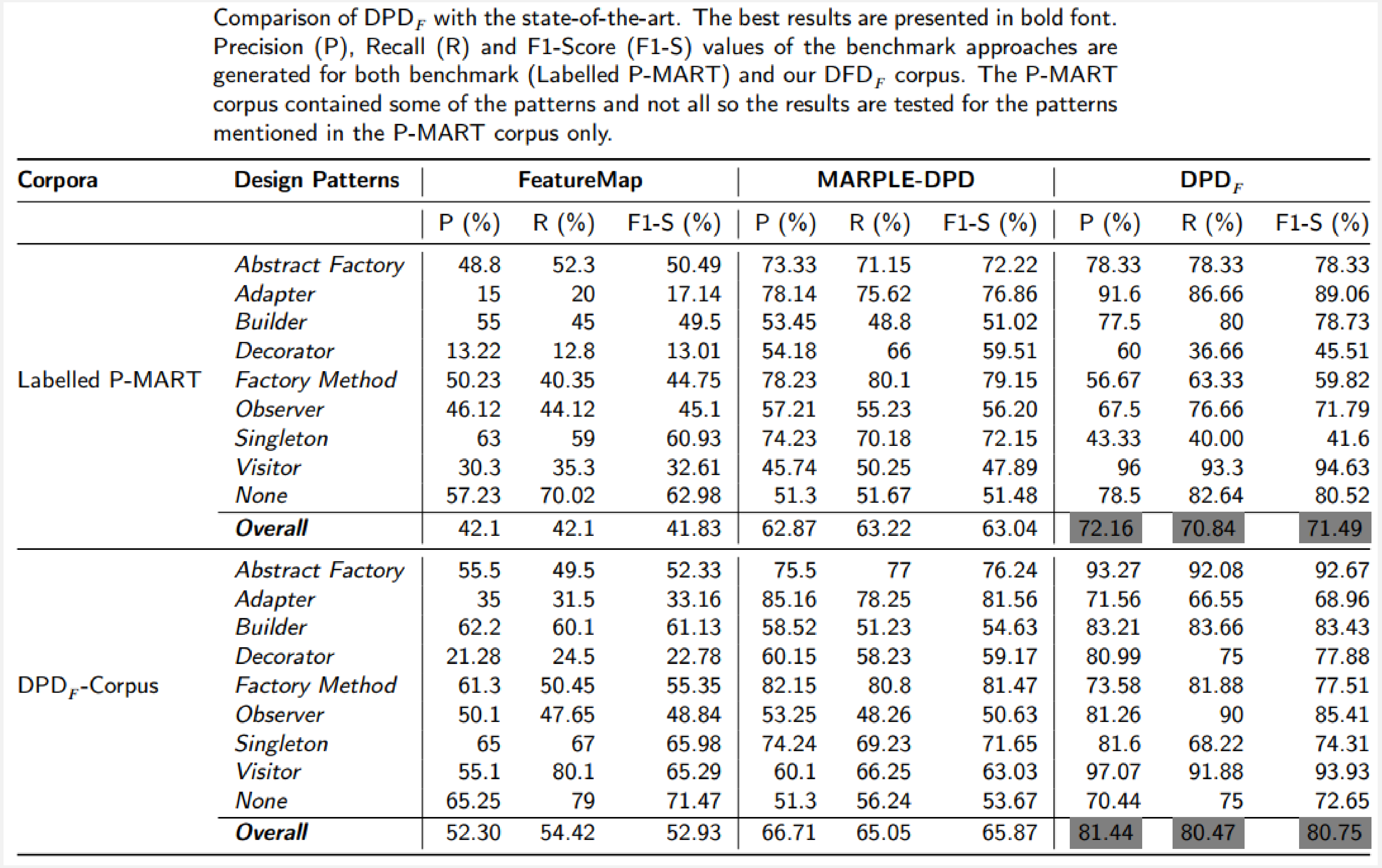
Results and Evaluations (cont.)

- How DPD$_F$ compares to prior pattern-detection research?
  - FeatureMaps — Thaller et al. (2019)
    - Used Feature-Maps with Random Forest + CNN.
    - DPD$_F$ replicated using only RF to fairly align with their baseline.
    - Original work detected only Decorator & Singleton, but extended to all patterns in benchmark corpus.
    - DPD$_F$ > FeatureMaps:
      - +30% Precision, +30% Recall on benchmark corpus.
      - On DPD$_F$-Corpus → ~30% Precision gain & ~26% Recall gain.

Results and Evaluations (cont.)

- How DPD_F compares to prior pattern-detection research?
  - MARPLE-DPD — Fontana et al. (2011) + Zanoni et al. (2015)
    - Extracts patterns mechanically using basic elements & metrics.
    - Different classifiers originally; DPD_F replicated using only RF for fair comparison.
    - Weighted Precision & Recall calculated for all patterns.
    - DPD_F > MARPLE-DPD:
      - ~10% Precision improvement on benchmark corpus
      - ~15% Precision improvement on DPD_F corpus
    - Patterns with <30 labelled instances (Facade, Memento, Prototype) excluded for K=10 CV.
    - Singleton sometimes better detected by MARPLE due to very few instances in corpus. Larger dataset improves classifier training.

Results and Evaluations (cont.)



Comparison of DPD$_F$ with the state-of-the-art. The best results are presented in bold font. Precision (P), Recall (R) and F1-Score (F1-S) values of the benchmark approaches are generated for both benchmark (Labelled P-MART) and our DFD$_F$ corpus. The P-MART corpus contained some of the patterns and not all so the results are tested for the patterns mentioned in the P-MART corpus only.

Corpora	Design Patterns	FeatureMap			MARPLE-DPD			DPD$_F$		
		P (%)	R (%)	F1-S (%)	P (%)	R (%)	F1-S (%)	P (%)	R (%)	F1-S (%)
Labelled P-MART	*Abstract Factory*	48.8	52.3	50.49	73.33	71.15	72.22	78.33	78.33	78.33
	Adapter	15	20	17.14	78.14	75.62	76.86	91.6	86.66	89.06
	Builder	55	45	49.5	53.45	48.8	51.02	77.5	80	78.73
	Decorator	13.22	12.8	13.01	54.18	66	59.51	60	36.66	45.51
	Factory Method	50.23	40.35	44.75	78.23	80.1	79.15	56.67	63.33	59.82
	Observer	46.12	44.12	45.1	57.21	55.23	56.20	67.5	76.66	71.79
	Singleton	63	59	60.93	74.23	70.18	72.15	43.33	40.00	41.6
	Visitor	30.3	35.3	32.61	45.74	50.25	47.89	96	93.3	94.63
	None	57.23	70.02	62.98	51.3	51.67	51.48	78.5	82.64	80.52
	Overall	42.1	42.1	41.83	62.87	63.22	63.04	72.16	70.84	71.49
DPD$_F$-Corpus	*Abstract Factory*	55.5	49.5	52.33	75.5	77	76.24	93.27	92.08	92.67
	Adapter	35	31.5	33.16	85.16	78.25	81.56	71.56	66.55	68.96
	Builder	62.2	60.1	61.13	58.52	51.23	54.63	83.21	83.66	83.43
	Decorator	21.28	24.5	22.78	60.15	58.23	59.17	80.99	75	77.88
	Factory Method	61.3	50.45	55.35	82.15	80.8	81.47	73.58	81.88	77.51
	Observer	50.1	47.65	48.84	53.25	48.26	50.63	81.26	90	85.41
	Singleton	65	67	65.98	74.24	69.23	71.65	81.6	68.22	74.31
	Visitor	55.1	80.1	65.29	60.1	66.25	63.03	97.07	91.88	93.93
	None	65.25	79	71.47	51.3	56.24	53.67	70.44	75	72.65
	Overall	52.30	54.42	52.93	66.71	65.05	65.87	81.44	80.47	80.75

Threats to Validity: Internal Validity

- Counting of instances:
  - Minor differences in methodology vs. benchmark (pattern roles vs. instances in file). DPExample project unavailable → potential minor impact.
  - Mitigation: relabel benchmark corpus using our method.
- Bugs:
  - Possible errors in SSLR generation, Word2Vec, or classifiers. Debugging reduced risk; remaining bugs likely reduce Precision/Recall, not inflate them.
- Data Labelling:
  - Potential disagreements between labelers.
  - Mitigation: ≥3 labelers; high kappa score; plan to hire more in future.

Threats to Validity: External Validity

- Reference set may not cover all Java source code of interest. Classifier performance outside reference set uncertain.
- Mitigation: increase size and diversity of reference set in future work.

Related Work

- Substantial research exists on detecting design patterns from source code.
- Approaches often transform code & patterns into intermediate representations: rules, models, graphs, or languages.
- Tools & ML methods used for pattern detection:
  - Static/dynamic analysis (Eclipse plugins, JStereoType, FUJABA).
  - Graph matching + ML (MARPLE-DPD).
  - Reverse engineering UML/diagrams.
  - Software metrics + ML (Uchiyama et al., Lanza & Marinescu).
  - Deep learning & feature maps (Thaller et al., Hussain et al.).
  - …

Related Work (cont.)

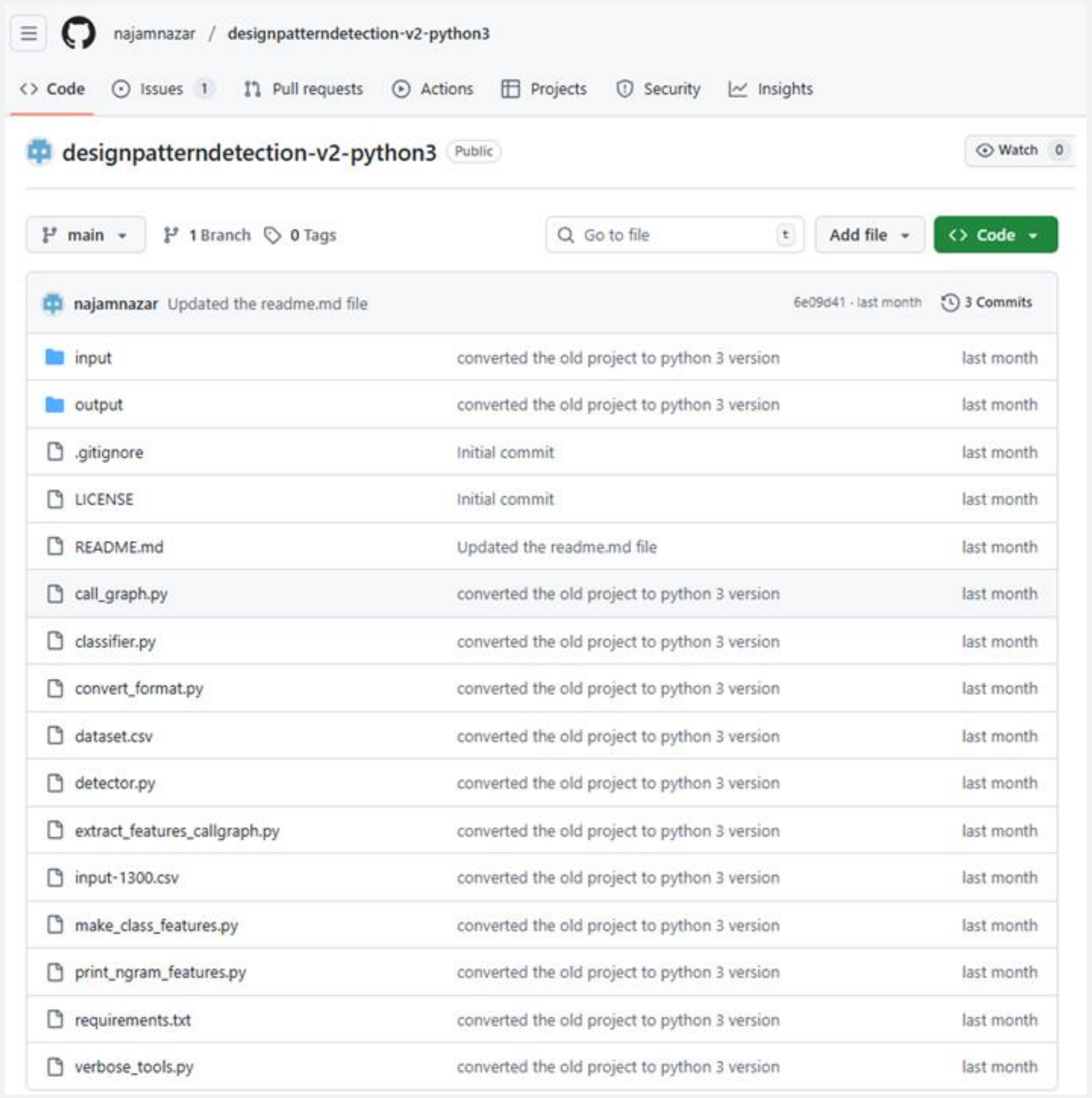
- How DPD_F differs?
  - Uses 15 source code features vs. metrics or feature maps in prior work.
  - Large labelled corpus: 1,300 files from 200+ projects.
  - Classifier achieves ~80% Precision (prior studies: 42–67%).
  - Detects 12 design patterns vs. 2–6 in existing approaches.

Conclusion & Future Directions

- DPD_F: A feature-based + ML powered framework for design-pattern detection.
- Workflow:
  - SSLR generation → Word2Vec embeddings → Supervised classifier.
- Performance:
  - ~80% Precision, ~79% Recall — beats prior work by +30% and +15%.
- Broader coverage:
  - Detects 12 patterns more than earlier studies.
- Future Work:
  - Expand training corpus + introduce additional source-code features → Potential gains for low-accuracy patterns & real-world applicability.

Additional Info

- The complete implementation, annotated reference set, and result files have been released publicly as an open-source project:
  - github.com/najamnazar/designpatterndetection-v2-python3



najamnazar / designpatterndetection-v2-python3

Code Issues 1 Pull requests Actions Projects Security Insights

designpatterndetection-v2-python3 Public Watch 0

main 1 Branch 0 Tags Go to file Add file Code

najamnazar Updated the readme.md file 6e09d41 · last month 3 Commits

input	converted the old project to python 3 version	last month
output	converted the old project to python 3 version	last month
.gitignore	Initial commit	last month
LICENSE	Initial commit	last month
README.md	Updated the readme.md file	last month
call_graph.py	converted the old project to python 3 version	last month
classifier.py	converted the old project to python 3 version	last month
convert_format.py	converted the old project to python 3 version	last month
dataset.csv	converted the old project to python 3 version	last month
detector.py	converted the old project to python 3 version	last month
extract_features_callgraph.py	converted the old project to python 3 version	last month
input-1300.csv	converted the old project to python 3 version	last month
make_class_features.py	converted the old project to python 3 version	last month
print_ngram_features.py	converted the old project to python 3 version	last month
requirements.txt	converted the old project to python 3 version	last month
verbose_tools.py	converted the old project to python 3 version	last month

Thanks for your attention ☺