Automatically Analyzing the Consistency and Preciseness of Class Names

Luuk Stevens
Universiteit van Amsterdam
luuk.stevens@student.uva.nl

Thesis supervisor: Jurgen Vinju

July 24, 2012

Abstract

The consistency and preciseness of class names is important for program comprehension. The goal of this research is to automatically analyze the consistency and preciseness of these names, so that the comprehensibility and maintainability of software can be increased. This may ultimately result in lower overall cost of software projects.

The main research question is: Can the consistency and preciseness of class names in object-oriented software be analyzed automatically? To answer this question an analysis process is developed, that groups "similar" classes together. The classes in these groups are then analyzed for inconsistent and imprecise names. When a class is considered inconsistent or imprecise, a renaming suggestion is given. The suggestions are manually evaluated to determine how successful the proposed analysis process is.

The results of this research suggest that the proposed analysis process does not yet create renaming suggestions that are usable in practice. We successfully found the cause of this problem. Results of a follow-up research indicate that this problem can (partially) be solved, likely improving the renaming suggestions proposed by the analysis process.

Contents

Preface

I would like to thank Jurgen Vinju for giving me the opportunity to perform my master thesis research at the "Centrum Wiskunde & Informatica". Furthermore, I would like to thank Dennis van Leeuwen and Jouke Stoel for sharing their idea's regarding identifier names, formal concept analysis and other topics related to this research.

Chapter 1

Introduction

In this chapter we introduce the automatic analysis of the consistency and preciseness of class names in object-oriented software. First, the relevancy and goal of this research is discussed, followed by a review of related work and a discussion of the research questions, motivation of the research questions and a description of the structure of this document.

1.1 Relevance and Goal

Maintenance costs often dominate the cost of software projects [7]. Source code has to be readable, since it is otherwise hard to comprehend. Especially, since the programmers maintaining software are often not those who constructed it. Program identifiers play an important role in source code comprehension [11, 14]. Since a programmer has to comprehend (part of) the source code before it can be maintained, the quality of identifiers thus influence the cost of a software project.

The detection (and correction) of inconsistent and imprecise class names can result in more understandable code and improve communication between programmers. This is because programmers using a term (class name) in a consistent and precise manner, are more likely talking about the same concept, reducing the chance of miscommunication. This improvement is likely to result in decreased development time, increased maintainability, and ultimately lower overall costs of software projects.

1.2 Related Work

Although in this research class names are analyzed, Høst and Østvold already proposed a method for "debugging" method names, of which some theories could be modified and applied to the analysis of the consistency and preciseness of class names. Høst and Østvold used part-of-speech analysis in combination with nano patterns (binary properties of Java methods, that are automatically detectable or detectable by programmers [15]) to find "naming bugs" [8]. The main goal of their method is to detect inconsistency in names of methods with similar implementation properties. Results of this research look promising in providing an automated analysis process for finding “bugs” in method names, although the effectiveness (usefulness of the renaming suggestions) of the proposed process is not yet analyzed by expert inquiry.

Singer found that there is a correspondence between the suffix of a class name (The last word in the name. For example, Buffer in StringBuffer) and the micro patterns occurring in classes with names with that suffix [16]. This makes it possible to find ambiguity or inconsistencies in the naming of classes.

The work of Singer provides a good starting point for analyzing the consistency of class names by analyzing the suffix. This thesis contributes by extending Singers method with the analysis of the consistency of the whole class name. Furthermore, the detection of imprecise naming of compound words could be integrated in the analysis. Renaming suggestion for classes that are named imprecisely can be given using a method based on the work of Høst and Østvold. For example, suggesting the more precise name "CustomException" for a class that is called "Exception".

1.3 Research Questions, Motivation and Structure

In this section we introduce the necessary definitions, present the research questions and discuss the structure of this document.

1.3.1 Definitions

It is necessary to present some definitions, before the research questions can be presented. We need to define how we will abstract over class names and what we mean with inconsistency and impreciseness. The definitions are as follows:

- *Phrase:* Based on the definition of Høst and Østvold: A phrase is a non-empty list of parts. A part p may be a token (word in the name) or a tag (word type, such as a noun or adjective). A phrase that consists solely of tokens is concrete; all other phrases are abstract.

- *Inconsistency:* A class name is considered inconsistent, if (a part of) its phrase is significantly different from other classes with similar implementations. What significantly different is, is determined using statistics.

- *Impreciseness:* A class name is considered imprecise, if a longer phrase (with more parts) is more common for class names of classes with similar implementations. What significantly different is, is for preciseness also determined using statistics.

Note, that with this definition of inconsistency we try to find synonyms. An example of an inconsistently named class is "CustomError", when there are many "similar" classes that are named "<Noun>-Exception", or maybe even more concrete "CustomException". In this case "CustomException" is more consistent than "CustomError".

An example of a imprecisely named class is a class named "Exception", when many "similar" classes have a longer name, like "<Noun>-Exception", or maybe even more concrete "CustomException". In this case "CustomException" is more precise than "Exception".

1.3.2 Primary Research Questions

In this section the main, or primary, research questions are defined and motivated. We close this section with a note on information hiding, which is related to these questions.

Research Questions

Using the definitions from the previous section, the main research questions are defined as follows:

- Can the consistency and preciseness of class names in object-oriented software be analyzed automatically?
- How "effective" is the analysis process proposed?

With effective we mean, in how many cases we can argue that a renaming suggestion proposed by the analysis process is more suitable than the original class name.

Motivation

The motivation for the main research questions is, that Høst and Østvold presented a quite sophisticated approach for the analysis of consistency (and preciseness) of methods names. Singer analyzed only the suffixes of classes, while both consistency and preciseness are important for program comprehension according to Deissenboeck. We think an approach based on the work of Høst and Østvold might work, by using micro patterns to group similar classes together (instead of nano patterns for methods, as Høst and Østvold did). Furthermore, Singer already found that there is a correlation between the suffix of class names and micro patterns. This relation might also hold for other parts of the class name. By analyzing the consistency and preciseness of classes using an approach based on the work Høst and Østvold, we will investigate if more usable renaming suggestions could be generated.

Note, that we do not propose or investigate a general definition or guideline for good naming. In this research we focus on consistency and preciseness, because inconsistent and imprecise naming is found harmful for program comprehension by Deissenboeck [4]. He uses the word "conciseness" where we use preciseness. With conciseness Deissenboeck refers conciseness of meaning, not the length of a word. We will use the term preciseness, to avoid confusion with conciseness as in concise code or concise (short) names.

A Note on Implementation Hiding

One might argue that it is not appropriate to use the implementation of classes for the analysis of the correctness of its name, since the class name is an abstraction of its implementation. And this abstraction should not "leak" how a class is implemented. Note however, that we will not analyze the implementation of a class to determine *how* a class is implemented, but to determine *what* a class implements. We try to make a distinction between the different concepts classes implement. A properly named class is very likely to have a name that contains the name of the concept it implements. This is discussed more in more detail in Section 2.1.

1.3.3 Secondary Research Questions

In this section the secondary research questions of a follow-up research are defined and motivated.

Research Questions

Initially, our goal was to perform a second study, involving experienced programmers judging the renaming suggestions produced by the proposed analysis process. The goal of this case study would be to more accurately and objectively determine how effective this process is. However, the results of the main research indicate that the proposed process does not yet produce renaming suggestions usable enough to perform an extensive case study. Therefore, we present the following research question for our second study, which will later be formulated more concretely:

- Can the renaming suggestions produced by the proposed analysis process be improved?

Motivation

The motivation for the case study, involving experienced programmers (which we did not perform, but such a study could still be useful for future research), is that there is no accurate data on how effective the proposed “debugging” methods from related research are (For example, the methods of Høst & Østvold and Singer). There are many factors that can make the proposed methods ineffective. For example, micro or nano patterns could not be powerful enough to distinguish sufficiently between the differences in the implementations of classes or methods for the purpose of name analysis. Furthermore, programmers could intentionally implement two classes with the same name differently. With providing data on the effectiveness of the method proposed in this research, we would have aimed to set a point of reference for future research.

1.3.4 Structure

Chapter 2 describes the research regarding the main research questions. The next chapter, chapter 3, describes the follow-up research regarding the secondary research question. We close this thesis with a conclusion and a discussion of directions for future research.

Chapter 2

Analyzing Consistency and Preciseness

In this chapter we address the first research questions: Can the consistency and preciseness of class names in object-oriented software be analyzed automatically? And, how effective is the analysis process proposed?

First, an introduction to the theories used during this experiment is given. Next, the experiment is discussed in the methods section. Then, the approach section describes the approach taken to automatically analyze the consistency and preciseness of class names. Finally, we close this chapter with an overview and a discussion of the results.

2.1 Introduction

Before we can discuss the experiment and the analysis approach that will be taken to automatically analyze the consistency and preciseness of class names, it is necessary to introduce relevant concepts and their relations. A theoretical framework of these concepts and relations is shown in figure 2.1.

In this framework we use two terms that might need an introduction: "implementation semantics" and "semantic profile". With implementation semantics we mean: The "true" meaning of the source code of a class. Programmers can determine this meaning by reasoning about the source code.

With semantic profile we mean: An abstraction of the implementation semantics. Classes with similar implementation semantics would have the same, or a similar, semantic profile. This profile enables the automatic grouping of classes with similar implementation semantics.

2.1.1 Relation Between the Class Name and Implementation Semantics

In the introduction of this document we assumed that inconsistent and imprecise naming could be detected by analyzing the name and implementation semantics of classes. This assumes that there is a relation between the name of a class and the semantics of its implementation (Shown in figure 2.1 as name-implementation relation). In other words, classes with similar names are expected to have similar implementation properties.

We assume this, because class names are mostly not arbitrary chosen by programmers, but tell something about the concept or function of a class. In the "Java Code Conventions" [17] is stated that class names should be simple and descriptive. Furthermore, Robert C. Martin argues in his book "Clean Code" that identifier names should reveal intent and that only one word per concept should be used (For example, not mixing Controller and Manager) [12].

That programmers largely follow these guidelines is supported by the work of Singer and Kirkham. They found a correlation between the last word of a class name (suffix) and patterns in the implementation of the class [16].

2.1.2 Abstraction of the Implementation Semantics

We want to automatically determine what classes have similar implementation semantics, and therefore are expected to have similar names. We need an abstraction, since machines cannot determine the true meaning of classes. Abstractions will leave out details, thus there will always be a semantic gap between the semantic profile (abstraction) and the "true meaning" (implementation semantics) of class implementations (Shown as the semantic gap in 2.1). However, if we abstract appropriately, there will also be a relation between the class name and the semantics profile of a class (Shown in figure 2.1 as name-abstraction relation). How we will analyze the class names and create an semantic profile of classes is discussed in section 2.3.

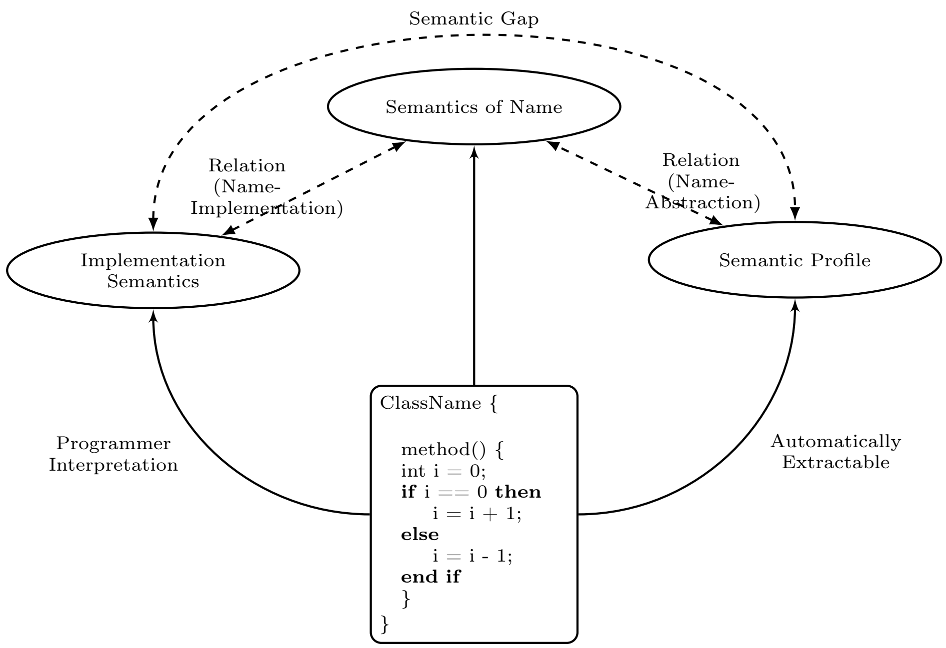


Figure 2.1: A theoretical framework of class semantics as used for the analysis of the concsistency and preciseness of class names.

2.2 Research Method

In this section the research methods are discussed.

2.2.1 Design of the Analysis Process

The first step is to design and implement a process for the automatic analysis of the consistency and preciseness of class names. The approach chosen will be based on the theories introduced in this chapter, and will be discussed in section 2.3.

Designing this process is not straightforward, and success is not guaranteed. The analysis of the consistency and preciseness of class names is not researched before, to the extend of this study. Questions we need to answer are:

- How do we determine how classes are generally named?
- How do we group "similar" classes together effectively?
- How do we construct an algorithm that analyzes the consistency and preciseness of a class name given a set of "similar" classes?

Success is not guaranteed, because:

- An inappropriate method may chosen to create a semantic profile of classes.
- Programmers might intentionally name classes with different implementations similarly.

We will motivate each decision made during the design of the analysis process in section 2.3.

2.2.2 Case Study

To determine how well our approach works, we perform a small exploratory case study. Results will be obtained by analyzing the classes of one application, and consist of references to classes (class names with package prefix) that are found inconsistently or imprecisely named and their renaming suggestions.

2.2.3 Evaluating the Results

There will always be a semantic gap between the implementation semantics and the semantic profile of a class. The generated renaming suggestions are based on the abstraction of the implementation semantics. In order to be useful, there must be a relation between the renaming suggestion and the real implementation semantics. Since these semantics can only be interpreted by human, the validation of the results will depend on human judgement.

We will evaluate the results by taking a sample set of classes that are found to be inconsistent or imprecise, and evaluate the renaming suggestions. We will reason and judge whether the renaming suggestion fits the class better than its original name. Manual evaluation of the renaming suggestions will entail some threats to validity (Section 2.2.4).

2.2.4 Threats to Validity

A threat to validity is that we could be biased during the evaluation of the renaming suggestions, since this process depends on human judgement only. To mitigate this thread, the motivation for the judgement of every suggestion is given. Furthermore, the results in this chapter are only meant to get an impression of how well the proposed analysis method works.

2.3 Design of the Analysis Process

In this section the designed process for the analysis of the consistency and preciseness of class names is discussed. The process is based on the work of Høst and Østvold [8]. The following steps must be taken to complete the approach shown in figure 2.2:

- A natural language analysis must be performed on the class names. In this phase the class names are decomposed into individual words, and the type of each word is determined using part-of-speech (POS) tagging. This is discussed in section 2.3.1.

- Semantic profiles of the classes must be created, by analyzing Java .class files. This is discussed in section 2.3.2.

- A corpus of Java applications must be analyzed to determine how classes implementing a certain concept are generally named by Java programmers. This is discussed in section 2.3.3.

- Algorithms must be constructed to analyze the consistency and preciseness of class names with a similar semantic profile, and generate improvement suggestions. This is discussed in section 2.3.4.

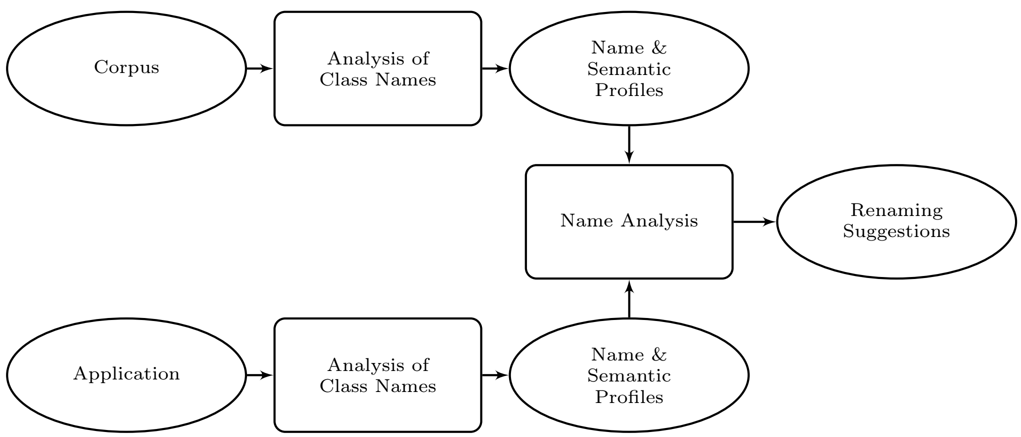


Figure 2.2: An overview of the approach that will be taken to analyze the consistency and preciseness of class names. The ovals represent inputs or (intermediate) products. The rectangles processing steps.

2.3.1 Analysis of Class Names

Our goal is to automatically generate renaming suggestions for classes that are named inconsistently or imprecisely. These suggestions will be given on token (word) level. For example, suggesting "Null-Pointer-Exception" for a class that is called "Null-Pointer-Error" (last token is considered inconsistent). Furthermore, we want to be able to give abstract suggestions, in case we can not give concrete suggestions with a certain statistical certainty. For example, suggesting "<noun>-Exception" instead of "Exception" (name is considered imprecise).

From the above examples we can conclude that we not only need to split the class names into separate tokens (by splitting at the upper case characters), but we also have to identify the word type (part-of-speech, or POS tagging), since not all words used in class names are nouns [3]. Knowing the word type lets us distinguish between suggestions like "<noun>-factory" and "<adjective>-factory".

Tokenizing

Although splitting camel cased class names into separate words seems quite straightforward, not all programmers strictly follow Java naming conventions (For example, using underscores to separate words). Furthermore, some class names can be split ambiguously (For example, J2SELibrary, J-2-SE Library or J2SE-Library). It is not necessary for this research to strive for perfect tokenization, but more accurate results will probably result in more usable data. Therefore the Intt tokenizer is used [2], because manual verification of the output of different tokenizers from related research indicates that this is the most accurate tokenizer (See also Appendix A).

Tagging

The grammatical roles of the words in the class names are determined using the POS tagging process. As for the tokenizer, a more accurate tagger will likely result in more usable data. Results from Appendix A indicate that the tagger of S. Butler is the most accurate. Therefore his tagger is used during this research.

2.3.2 Extracting Semantic Profiles

In this section is discussed how semantic profiles are extracted from classes.

Theory

Micro patterns are machine traceable patterns on class level. These patterns are similar to Design Patterns, but stand at a lower, closer to the implementation, level of abstraction. Gil and Maman proposed 27 micro patterns [6]. Five of these patterns are shown in Table 2.1 as an example. See the work of Gil and Maman for a complete list of definitions.

Gil and Maman also performed a static analysis on the occurrence of micro patterns. Their analysis suggests with a high confidence level, that the occurrence of these patterns is not random, but is tied to the specification or the purpose that the software realizes. This suggests that the same concepts, classes with similar implementation semantics (involving the same design decisions), probably contain the same micro patterns.

This statement is supported by the work of Singer and Kirkham. They found that there is a correlation between the suffix of class names (name reflects the intend or concept) and micro patterns (implementation semantics) [16].

The above findings suggest that the occurrence of micro patterns can be used as a semantic profile for classes, since: these patterns are tied to the purpose of classes, are machine traceable and there is a relation between the occurrence of patterns and (part of) the name of classes.

Name	Description
Pool	A class which declares only static final fields, but no methods.
Stateless	A class with no fields, other than static final ones.
Data Manager	A class where all methods are either getters or setters.
Sink	A class whose methods do not propagate calls to any other class.

Table 2.1: Examples of micro patterns defined by Gil and Maman.

Tool

Maman already constructed a tool to extract micro patterns from Java byte code. However, ironically we found the source code of this tool hard to understand, and therefore hard to verify (with the pattern descriptions from the paper) and modify. Therefore a new tool is constructed.

The newly implemented tool analyzes Java .class files to extract micro patterns using the ASM library [13]. ASM is used, because it is an easy to use library for the analysis of Java .class files.

Both tools output an array of booleans for each class, indicating which micro patterns occur in that particular class. The output of both tools are compared for a set of test classes (minimum implementations of classes containing a certain micro pattern). By comparing the output of the two programs, some remarkable observations are made. Some of them indicated faults in our tool (which we have corrected), but some suggest ambiguity in the work of Gil and Maman (ambiguous pattern definitions) or even inconsistencies between their work and their tool. The observations made by comparing the two tools can be found in Appendix B. We also had to make some assumptions during the implementations of the tool, which can also be found in Appendix B.

If the tool of Maman really contains faults, the conclusions made by Gil and Maman about micro patterns using statistical analysis may have become obsolete. They used statistical analysis to determine the individual value of each pattern, and the randomness of their occurrence. The outcome of their analysis is directly dependent on the accuracy of the tool used.

Note, that we would rather have analyzed Java source code, since this is the code the programmer works with. However, analyzing the source code seemed to be very impractical, because many projects use libraries, of which the source code is often not provided. Since some patterns involve inheritance, this means that the super classes of classes inheriting form these library classes cannot be analyzed.

We chose to analyze open source Java projects, since the sources of these projects are widely available. Furthermore, Java is an industrial language in wide use.

2.3.3 Extracting Names and Micro Patterns from a Corpus

To determine what class names are inconsistent or imprecise, we must first determine how classes containing certain micro patterns are generally named by Java programmers. We do this by analyzing the class names and micro patterns from a large corpus of Java applications from different development teams and different application domains. For this experiment the Qualitas Corpus is used [18].

This corpus contains 109 Java applications from different application domains. For every application the source code and the binary files are provided. In addition, every project contains a .property file which defines some metadata about the project. This meta-data includes what code (packages) is part of the system, and what parts are not (For example, infrastructure code and 3rd party libraries are not part of a system).

It would have been nice to have used the whole corpus, since one of its purposes is reproducibility (other researchers could download the corpus and repeat the experiment, producing the same results as a previous experiment). However, not all projects are complete, and are missing 3rd party libraries. These projects are excluded from the analysis. For reproducibility purposes the used projects are shown in table 2.2.

There are three things to note about the corpus. First, the corpus contains a couple of Java EE applications. These applications contain dependencies on the Java EE API. There are multiple implementations of this API. For example the Apache Tomcat and the Oracle Glassfish application servers contain an implementation. To keep analysis independent of implementation details, the Java EE API provided by Oracle is used. This API does however not include implementations of the methods.

Second, some applications depend on other projects in the corpus. One of the projects that is used by many other projects is JUnit. The jar-files of these projects must be loaded when a depending project is analyzed.

	3D / Graphics / Media	
drawswf-1.2.9	galleon-2.3.0	jhotdraw-7.5.1
joggplayer-1.1.4s	sunflow-0.07.2	
	IDE	
checkstyle-5.1	drjava-stable-20100913-r5387	eclipse-SDK-3.6
netbeans-6.9.1		
	SDK	
colt-1.2.0	gt2-2.7-M3	jchempaint-3.0.1
jFin-DateMath-R1.0.1	jpf-1.0.2	
	Database	
axion-1.0-M2	azureus-4.5.0.4	c-jdbc-2.0.2
cayenne-3.0.1	derby-10.6.1.0	hibernate-3.6.0-beta4
hsqldb-2.0.0	squirrel-sql-3.1.2	
	Diagram / Visualisation	
argouml-0.30.2	exoportal-v1.0.2	ireport-3.7.5
jasperreports-3.7.3	jext-5.0	jung-2.0.1
velocity-1.6.4		
	Games	
freecol-0.9.4	marauroa-3.8.1	megamek-0.35.18
	Middleware	
castor-1.3.1	informa-0.7.0-alpha2	jboss-5.1.0
jena-2.6.3	jspwiki-2.8.4	jtopen-7.1
openjms-0.7.7-beta-1	picocontainer-2.10.2	quartz-1.8.3
quickserver-1.4.7	struts-2.2.1	tapestry-5.1.0.5
tomcat-7.0.2	xmojo-5.0.0	
	Parsers / Generators / Make	
ant-1.8.1	antlr-3.2	javacc-5.0
jparse-0.96	maven-3.0	nekohtml-1.9.14
sablecc-3.2	xalan-2.7.1	xerces-2.10.0
	Programming Language	
aspectj-1.6.9	jre-1.6.0	jruby-1.5.2
	Testing	
cobertura-1.9.4.1	emma-2.0.5312	findbugs-1.3.9
fitjava-1.1	fitlibraryforfitnesse-20100806	htmlunit-2.8
jmeter-2.4	jrat-0.6	junit-4.8.2
pmd-4.2.5	quilt-0.6-a-5	
	Tool	
columba-1.0	compiere-330	freecs-1.3.20100406
ganttproject-2.0.9	heritrix-1.14.4	jag-6.1
jedit-4.3.2	jfreechart-1.0.13	jgraph-5.13.0.0
jgraphpad-5.10.0.2	jgrapht-0.8.1	jgroups-2.10.0
jmoney-0.4.4	jsXe-04-beta	mvnforum-1.2.2-ga
proguard-4.5.1	roller-4.0.1	rssowl-2.0.5
sandmark-3.4	webmail-0.7.10	weka-3.7.2

Table 2.2: Corpus of Java applications.

Finally, even though great care is taken to resolve the dependencies in the corpus, some projects seem to contain some dependencies that could not be resolved. It is possible that the developers of these projects broke these dependencies (unintentionally). However, we excluded the project from the corpus if more than 1% of the analyzed classed failed, as the result of unresolved dependencies. 1% is a somewhat arbitrary number, but I found that most projects only contain a couple of unresolved dependencies (less than 1%). Only a couple of project contain a large amount of unresolved dependencies, suggesting missing libraries.

2.3.4 Consistency and Preciseness Analysis Algorithms

The consistency and preciseness analysis process (See figure 2.3) consists of three distinct algorithms, namely:

- *Recurring Suffix Extraction:* Names that are generally used by programmers are extracted from the corpus.
- *Concept Creation:* Classes are grouped into concepts. We define as a concept a set of classes containing the same (sub-)set of micro patterns.
- *Name Analysis:* Each concept is analyzed for inconsistent and imprecise class names.

The process uses the analyzed class names and micro patterns (as discussed in section 2.3.1 and 2.3.2) of a corpus and an application to be analyzed as input. The output of the process is a set of renaming suggestions per inconsistently or imprecisely named class of the application that is analyzed. In this section each of the phases of the analysis process is discussed, including the analysis algorithm.

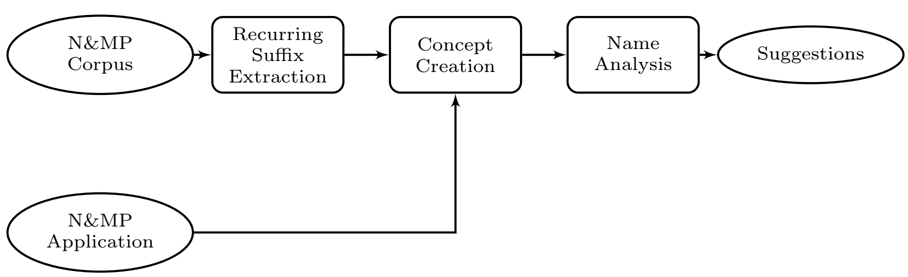


Figure 2.3: The inputs, output and phases of the consistency and preciseness analysis process. N&MP is name and micro patterns abbreviated.

Recurring Suffix Extraction

The class name and micro pattern data of classes from the corpus is extracted, if the suffix of the class name occurs at least twice, and at least in two different applications from the corpus. This simple heuristic filters out all class names that are specific to an application, or used solely by one developer or team. These incidental or idiosyncratic names are not valuable for the analysis of consistency and preciseness, since this information does not tell something about how concepts are generally named. Furthermore, the large amounts of data would slow the analysis down significantly.

We look at the suffix, because most compound words are noun-noun compounds. For this kind of compounds it is generally accepted that they involve a head concept and a modifier concept. This means that the compound describes a specialization of the head concept. Both in English and in class names, the last word of the compound is almost always the head of the compound. In Blackboard for example, Board is the head concept and Black is the modifier. Thus blackboard is a special kind of board [4]. By looking at the last word of the class name, we thus check if the head concept (the not specialized version of the word) recurs in different application and is thus generally used.

Note, that we chose for a threshold value of 2, because this is the lowest value for which we can speak of recurring suffixes. If the results indicate that a higher value might improve the renaming suggestions, the threshold value can be increased.

Concept Creation

We will create concepts of classes with the same (sub-)sets of micro patterns using Formal Concept Analysis (FCA). This type of analysis is a way to create a hierarchy of formal concepts that represent the set of objects sharing the same values for a certain set of properties. The hierarchy is formed by placing concepts below the concepts that have a super-set of objects. We want to group classes with the same micro patterns together, so that we can analyze if there are specific names used for specific combinations of micro patterns.

We can easily apply FCA to create concepts, because the "name-micro pattern" combinations can be directly used as formal context, which is needed to create formal concepts. A formal context consists of Objects (the classes) with the same set of attributes (the 27 micro patterns), and values indicating which object contains which attributes (boolean values indicating wether a class contains a micro pattern). The objects and attributes could also be swapped (patterns as object, classes as attributes). This has not influence on the analysis.

After performing FCA on the given formal context, classes with the same micro patterns (and the same sub-sets of the containing micro patterns) are grouped together into formal concepts. A concept lattice (hierarchy) of a small amount of classes could look something like figure 2.4.

The figure shows the top concept that contains all the classes as attributes, since all classes contain no patterns as sub-set. The bottom concept contains no classes as attributes, since no class contains al the occurring micro patterns. The concepts in between, represent al the occurring combinations of micro patterns as objects, with the classes containing that combination of micro patterns as attributes.

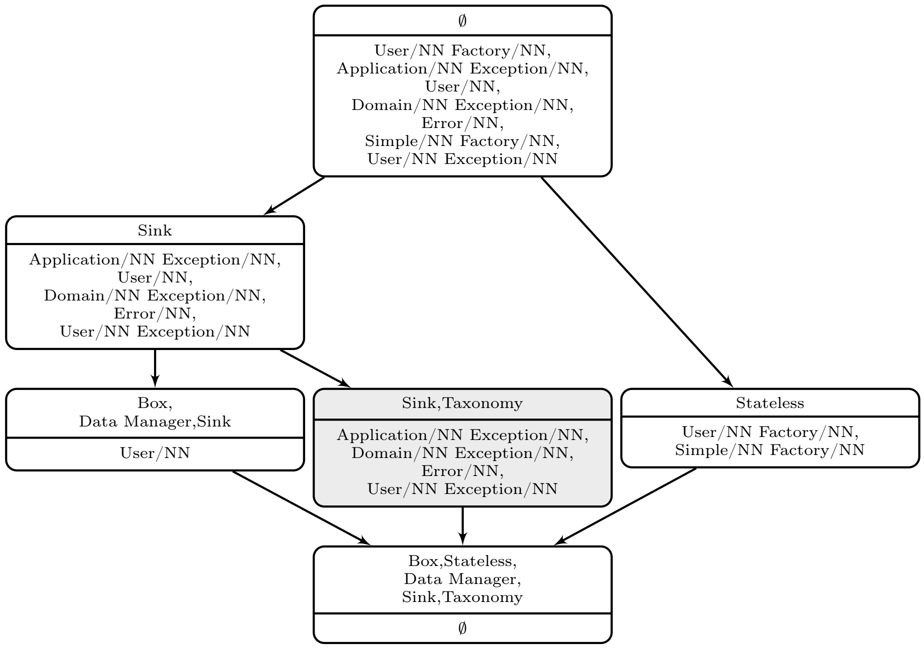


Figure 2.4: Example of a concept lattice with the occurence of micro patterns as formal objects (upper node part) and class names with their tokens and tags as formal attributes (lower node part). /NN indicates a noun tag.

FCA is chosen because it fits neatly with the kind of data we are dealing with, as described in second paragraph of this section. Furthermore, we can easily evaluate the consistency and preciseness of a class, using a subset of its patterns. These concepts are namely also created during the FCA phase. If we, for example, would use a database to query classes with the same micro patters, relatively complex queries are needed, that have to be executed frequently (for each class). To this respect, FCA is a more efficient and easier applicable technique.

It is important that the class names and micro patterns of the application to be evaluated are filtered out of the recurring suffix classes (if they are in). If this is not done, the class name and micro patterns of the analyzed application can occur twice in the formal concepts, skewing frequencies of the class tokens and tags, and thus the renaming suggestions.

Name Analysis

Now that all the classes containing the same micro patterns (implementing similar concepts) are grouped together in nodes of the concept lattice, the consistency and preciseness of class names in each node can be analyzed. The analysis is done using an analysis algorithm. The algorithm is first explained by an example (Figure 2.5), followed by an explanation using an activity diagram (Figure 2.6).

The counts of all the tokens or tags at a certain position of the class names in combination with a threshold value are used to determine wether a token or tag is considered inconsistent or imprecise. The counting of tokens or tags is done the following way: We count the tokens or tags of all classes within a concept at a certain position. The tokens or tags of the first position contain all the last tokens or tags of all the classes within a concept (the last token or tag is the first under analysis).

The relations of the threshold value and the token or tag counts with respect to the consistency and preciseness of a class name are defined in the following ways:

- *Inconsistency:* A token or tag at a certain position of the class name under analysis is considered inconsistent, if: The token or tag count of the current position of the class name under evaluation, divided by the number of classes in the concept is smaller than the threshold value; *and* the count of at least one other token or tag divided by the number of classes in the concept is greater the threshold value.

- *Impreciseness:* The count of a token or tag divided by the number of classes in the concept is greater than the threshold value, at a position greater than the size of the class name under evaluation.

For the example we use the concept figure from 2.4 with the Sink and Taxonomy micro patterns (See Table 2.1 for the definitions of each micro pattern) as formal objects (the gray concept in the figure). Furthermore, we use a threshold value of 30%. This value might not be very suitable for a real-world analysis, but it simplifies our example. A high threshold value allows us to keep the number of classes in the concepts of the example low. The influence of the the threshold value will be discussed at end of this section. The first step of the analysis is, is to reverse the list of tokens and tags of the class name, so that the last tokens (heads of the compounds) are evaluated first. The analysis algorithm will take the following steps:

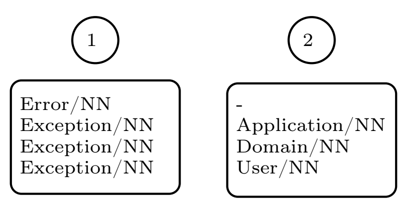


Figure 2.5: Example of the analysis order of the consistency and preciseness analysis algorithm

1. The analysis starts with analyzing tokens by class, thus starts at the first token of the first class, the "Error" token. The token "Error" occurs in a

ratio of 25%. The "Exception" token occurs in a ratio of 75%. This means that the "Error" token at this position is *inconsistent* for our threshold value.

2. The only token occurring above our threshold value is "Exception", which is added to the set of (partial) suggestions for the "Error" class (suggestions are now: { "Exception" }).

3. At the next position, position two, the "Error" class has no tokens or tags. We now directly check if there are any tokens that occur above the threshold value. These new parts of the suggestion will be added to the existing (partial) suggestions. In this case there is no such token.

4. Since there is no suggested token at this position, the algorithm starts analyzing tags. At this position there is a tag that occurs above our threshold value, namely "/NN". The "Error" class can thus be more *precisely* named by adding a noun. The noun-tag will be appended in front of each suggestion in our set of (partial) suggestions (suggestions are now: { "/NN, Exception" })

5. This was the last possible position, the analysis restarts for the next class.

Note, that this example does not discuss two enclosing loops of the discussed algorithm explicitly. The most outer loop iterates over all the concepts, since all the concepts are analyzed. A loop that is nested within this most outer loop, iterates over all the classes within a concept, since all the classes within concepts are evaluated. We left out this loops to simplify the explanation. During the rest of this chapter we will not discuss the mentioned loops.

Figure 2.6 shows an activity diagram of the algorithm for the analysis of one class in a concept. Notice that it contains two similar phases. The analysis of tokens (items shown in grey) and tags (items shown in white). Each phase contains four of the same decision points and activities.

At the main decision point (first of its color) is decided, whether the current position is:

- *Lower than the phrase length of the class:* In this case the consistency of the class name is analyzed. During this process is first evaluated wether the token or tag of the class occurs below the threshold value. Then if another token or tag occurs above the threshold value.

- *Greater than the phrase length of the class:* In this case the preciseness of the class name is analyzed. During this process is directly evaluated wether there are tokens or tags in the concept at that position that occur above the threshold value.

- *Greater than the max phrase length in the concept:* If the position is greater than the maximum phrase length of all classes of the concept, the analysis for the class stops.

At the activity points the following actions are taken:

- The current token or tag of a class is added to the list of suggestions, when its not inconsistent.

- The tokens or tags above the threshold are added to the list of suggestion, when the class name is inconsistent or imprecise.

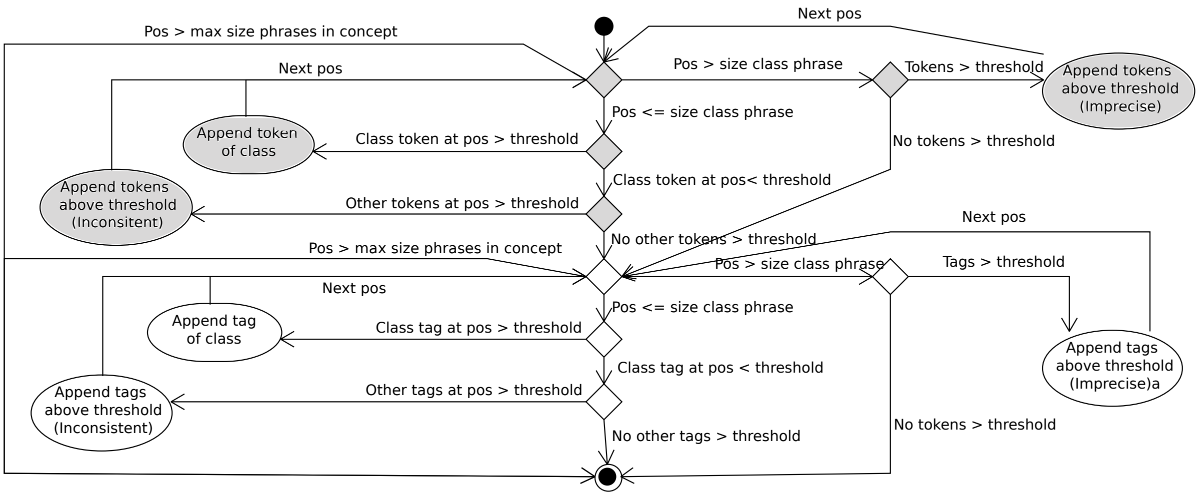


Figure 2.6: Actvity diagram of the analysis algorithm for the analysis of a class. "Pos" means position. Append... means append token / tag in front of current (partial) suggestions.

The algorithm can be refined by introducing a separate threshold value indicating when tokens or tags are used in a renaming suggestion. The two threshold values are then:

- *Inconsistency Threshold (IT):* This threshold value indicates when token or tag at a certain position is considered inconsistent. If a threshold of 1% is chosen, a token or tag is considered inconstant if it occurs 1% or less at a certain position in the concept under analysis. Expected is that a higher value yields more inconsistently evaluated classes yielding more false positives, and vice versa, yielding more false negatives.

- *Suggestion Threshold (ST):* This threshold value determines when a token or tag is used as renaming suggestion. If this threshold is 5%, a token for a certain position in the concept will only be used as suggestion when it occurs for 5% or higher. Expected is that a lower value will not only produce more suggestions (since a token or tag is only considered inconsistent when it occurs below the inconsistency threshold AND other tokens or tags occur above the suggestion threshold), but also results in suggestions that are likely to be more divers, because more (and possibly less prominent) tokens and tags will be used in renaming suggestions.

We will not try to find optimum threshold values during this research, but reasoning about them makes it possible to refine our results later.

Note, that the use of threshold values is not necessarily the most sophisticated "statistical" method that can be used to analyze the consistency and preciseness of classes. However, this is a simple method to reason about and implement. This makes it relatively simple to perform the analysis. More sophisticated, and possibly more effective statistical methods could be used in the future.

2.4 Case Study

In this section a single Java system from the Qualitas Corpus is analyzed according to the method discussed in the previous section. We chose the AspectJ project for our case study, because it is medium sized, hopefully yielding a manageable amount of renaming suggestions. AspectJ contains 4795 classes that contain at least one micro pattern, that can thus be evaluated. For the first analysis we use a inconsistency threshold of 0.5% and a suggestion threshold of 7.5%.

Our case study is explorative. We try to determine if usable renaming suggestions are produced by the proposed analysis process, and try to estimate how effective this process is. We also try to determine why good renaming suggestion are produced, or not.

We start section by discussing the algorithm performance. Then we discuss the renaming suggestions given by the analysis algorithm (results).

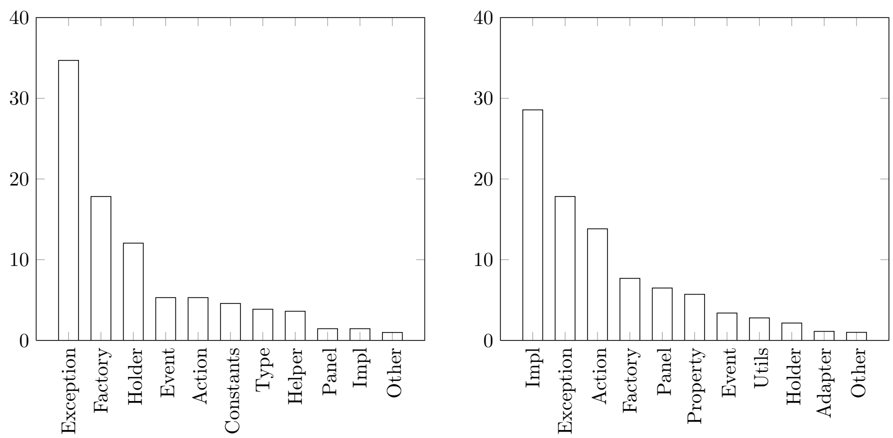
2.4.1 Algorithm Performance

Initially the formal concept analysis module of the Rascal meta-programming language was used to construct a concept lattice [9]. The performance of this algorithm seemed sufficient for the creation of concepts for the names and micro patterns of a small quantity of classes, but performed slowly on a large number class names and micro patterns. Rough estimations indicated that the runtime of the formal concept algorithm would be around 26 hours for 20k-25k classes. This kind of runtimes are not very practical. Since the lattice is not needed for this experiment, a more efficient algorithm is created, that calculates the concepts directly using map data structures. This algorithm calculates concepts for 25k classes in roughly 4 minutes. The code can be found in appendix C. The other algorithms used in the consistency and preciseness analysis process performed acceptably.

2.4.2 Initial Evaluation of the Results

Figure 2.7a shows that the variation of tokens used as the suffix of the renaming suggestions is very low. Only 10 (out of 31) suffixes account for 90% of the suffixes in the renaming suggestions. Other suffixes occur only 10% of the time. Such a low variation might indicate problems with the process or threshold values chosen. Since for example, it is not very likely that over 30% of the classes with a suggestion should be renamed to a name with the "Exception" suffix.

We can try to improve the suggestions by altering the suggestion threshold value. As suggested in section 2.3.4, we could try to increase the variation in suggested tokens by lowering the ST. Lowering the suggestion threshold to 5%, increased the amount of suffixes to 94, but the amount of suffixes occurring in 90% of the suggestions remains approximately unchanged, as shown in figure 2.7b. The results from the second plot suggest that the suggestion threshold is not the problem, so a problem might reside somewhere else in the process.



(a) Suggestion threshold = 7.5%

(b) Suggestion threshold = 5.0%

Figure 2.7: Suffix occurrence (in %) for renaming suggestions.

2.5 Evaluation of the Results

The lack of variation in suffixes may be caused by problems in the analysis process. In this case a lot of the renaming suggestions with common suffixes may not be useful. To confirm this hypothesis a sample of 20 random classes is manually examined to estimate the usefulness of suggestions generated by the algorithm. We use the results of the second analysis (Suggestion threshold = 0.5%). Ten classes are evaluated that have a suggestion with a commonly occurring suffix, shown in figure 2.7b. Ten classes having suggestions with a less commonly occurring suffix are also evaluated. This way we can see if less commonly suggested suffixes lead to better renaming suggestions.

The results are summarized in table 2.3. A complete overview of the evaluation, including motivation, can be found in Appendix D. The results suggest that the renaming suggestions are generally not very useful. However, some renaming suggestions that might be useful can be found in the set of suggestions with a less commonly occurring suffix. However, this type of renaming suggestion only accounts for 10% of the generated suggestions. In the next section we try to find the cause for the large amounts of useless renaming suggestions.

	No	Maybe	Yes	Total
Common	10	0	0	10
Uncommon	7	3	0	10

Table 2.3: Usefulnes of the renaming suggestions. No = the suggestion does not make sense, and is less suitable than the original name. Maybe = the suggestion makes some sense, and can be swapped with the original name. Yes = the suggestion is better than this original name. Suggestions with the Impl suffix are ignored. See Appendix D for a motivation.

2.6 The Cause

From the previous sections we can conclude two things:

- There is a lack of token variation in renaming suggestions (or at least the suffix). A small amount of tokens occurs frequently in renaming suggestions.
- Suggestions with tokens (or at least suffixes) that occur less frequent tend to make more sense. These suggestions are however overruled by the other suggestions.

In this section we try to find the cause for the large amount of useless renaming suggestions.

2.6.1 Suffix Occurrence

Our hypothesis is that the large amount of useless renaming suggestions is caused by the fact that commonly occurring suffixes are present in a wide variety of concepts. In other words, the common suffixes are not isolated enough to a restricted amount of concepts, overruling other tokens during the analysis process. Until now we assumed that after this phase classes with common suffixes are grouped together, however confirmation is needed.

Our hypothesis is supported by figure 2.8. The plot shows the amount of times a suffix occurs in total, and in how many concepts a suffix occurs in the concepts created for the AspectJ project. Suffixes that occur many times and in many different concepts (upper right part of the plot) can be considered very influential during the analysis process, because they are very generally used (occurring in many concepts), and tend to occur above the suggestion threshold (high occurrence), resulting in many unwanted suggestions. In figure 2.8 we can see that many suffixes from figure 2.7b are indeed positioned further up and right in the plot than most other suffixes. However, there many more outliers that do not occur often as suffix in the renaming suggestions.

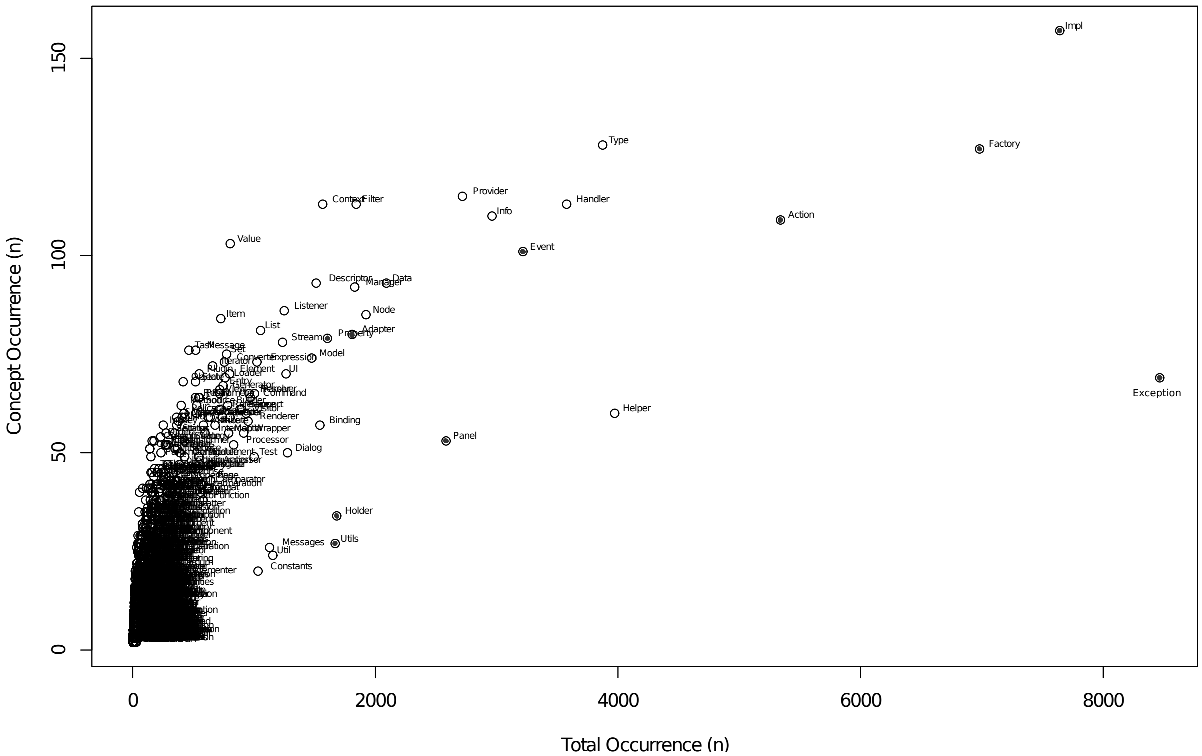


Figure 2.8: Scatter plot of suffix occurrence for the concepts used for the AspectJ analysis. The x-axis indicates the total amount of times a suffix occurs over all the concepts. The y-axis shows the total amount of concepts the suffix is present in. The dot markers of the common suffixes from figure 2.7b are filled.

2.6.2 Suffix Influence

In the previous section we did not include one important factor in our plot. This is for how many classes the suffix will be used as renaming suggestion. We cal this the influence of a suffix for short. The influence of a suffix can be defined informally as: "The sum of all classes below the inconsistency threshold in the concepts for which the suffix occurs above the ST". The suffixes with the greatest influence are shown in figure 2.9.

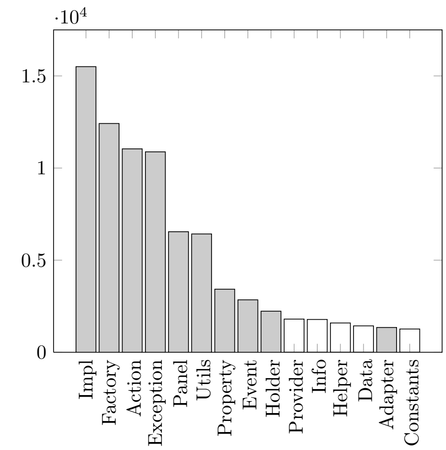


Figure 2.9: The suffix influence for an inconsistency threshold of 0.05% and a suggestion threshold of 0.5%. The gray bars mark the common sufixes from figure 2.7b.

All the common suffixes are highly ranked in the figure, which means that they are highly influential for the renaming suggestions. Notice that the adapter suffix is ranked slightly lower than expected. This is very likely the result of the fact that multiple renaming suggestions can be given to a class, depending on the consistency and preciseness of other tokens or tags in the class name under analysis. This might slightly skew the results.

2.7 Discussion

The results regarding the renaming suggestions from table 2.3 indicate that the proposed analysis process does not yet produce suggestions of a quality sufficient enough to use in practice, or even perform an extensive case study. Some renaming suggestions that make sense can be found, but these suggestions are overruled by a large amount of useless renaming suggestions with a common suffix (false positives).

We successfully found the root cause of the large amount of useless renaming suggestions proposed by the analysis algorithm. Some suffixes are not isolated well enough during the FCA phase of the analysis process, because micro patterns do not discriminate enough between the implementation semantics of classes. Therefore these suffixes can be highly influential during analysis of the consistency and preciseness of class names.

Furthermore, some suffixes are very generic, like the "Impl" suffix. This suffix is used for classes with a wide variety of implementation semantics.

Improving the renaming suggestions is not straightforward. The micro patterns proposed by Gil & Maman seem to be useful, but not sufficient enough to group classes during the FCA phase for the purpose of this research. Introducing new patterns, specific to our purpose might enhance the results. Introducing a set of micro patterns, that could lead to the production of usable results is however a lot of work, and deserves a research on its own.

However, the introduction of new patterns will probably not solve the problem of generally used names. These names could probably never be isolated. Ignoring these names or removing generic tokens from the name in advance of the analysis, might be a suitable strategy to reduce the large amount of unusable renaming suggestions caused by these names.

2.8 Summary

In this chapter we tried to answer the following research question: Can the consistency and preciseness of class names in object-oriented software be evaluated automatically? This research suggests that the proposed method yield only some useful results using micro patterns. The proposed method is therefore not yet ready to use in an extensive case study or in practice.

Chapter 3

Introducing Purpose-Specific Patterns

This chapter describes a follow-up research of the research described in the previous chapter.

3.1 Introduction

The proposed analysis process from the previous chapter does not yet produce renaming suggestions usable enough to perform a case study. We found that this problem is mainly caused by the fact that the existing micro pattern catalog does not contain micro patterns that distinguish sufficiently enough between the implementation semantics of classes for the purpose of this research.

The goal of this chapter is not to introduce a complete set of new patterns, that would produce usable renaming suggestions. This goal would deserve a research on its own. Instead, we try to determine if such a research might be useful, and therefore justifiable. The research question we try to answer in this chapter is: Can suffixes be isolated better using new patterns specific to this purpose? By answering this question, we try to answer the more abstract question presented in the introduction of this thesis: Can the renaming suggestions produced by the proposed analysis process be improved?

Notice, that we will not use the term micro-pattern here. We are indeed trying to find some traceable patterns on class level. However, we do not assess the individual value of each pattern with respect to the existing micro pattern catalog (as done with each of the existing micro patterns by Gil & Maman). Furthermore, the patterns presented here are designed specific for our purpose, whereas micro patterns are more generally applicable.

3.2 Research Method

In this section the research methods are discussed.

3.2.1 Designing Purpose-Specific Patterns

To evaluate if the introduction of new purpose-specific patterns (PSPs) will mitigate the problems described in the previous chapter, we will perform a sample experiment. In this chapter we will try to isolate two of the common suffixes, shown in figure 2.9.

The suffixes used as sample for this experiment are:

- *Exception*: This is the most commonly occurring suffix in the renaming suggestions after Impl (Figure 2.7b)[1].
- *Factory*: The most highly ranked design pattern name used as suffix in figure 2.7b. Since the factory patterns are clearly defined [5], it is expected that it is relatively easy to find commonalities in the implementation of classes with names ending with Factory.

Two PSPs are designed to specifically isolate these suffixes. The definition of the PSPs are determined by finding commonalities in the implementation of a sample of classes with one of the suffixes above. The commonalities will be formalized into new pattern definitions, and implemented in the analysis tool used for this research.

3.2.2 Case Study

A small explorative case study will be performed, to determine if the isolation of the Exception or Factory suffix is improved with respect to the isolation of this suffixes using micro patterns. We will again analyze the AspectJ project to obtain the results.

For this experiment we define the isolation of a suffix S using two variables, namely: The count of the suffix S in a concept, and the count of other suffixes in a concept. We say that a suffix is better isolated when it occurs in a higher number in a concept, other suffixes occur in a lower number in a concept, or both.

During the analysis concepts are created by performing FCA on all the recurring suffix classes in the corpus of the previous chapter. Only this time we include classes that contain no pattern. This is done because it is more likely for the classes to have no patterns using the 2 new PSP, then using the existing 27 micro patterns. Leaving out classes that contain no pattern might skew the results.

[1] The Impl suffix is not chosen, because this suffix may be used for any kind of class implementing an interface.

3.2.3 Evaluating the Results

We will see that the use of PSPs indeed increased the isolation of the Exception and Factory suffix, when the concepts created using these patterns contain more of the classes with the Factory or Exception suffix than any other concept using micro patterns. Furthermore, the concepts will contain fewer classes with another suffix, with respect to the concepts created using micro patterns.

3.3 Designing Purpose-Specific Patterns

This section describes the commonalities between classes with the Factory or Exception suffix. Using these commonalities a definition for the new PSPs is given.

Exception

The observations made during the manual evaluation of 10 random classes with the exception suffix, are shown in table E.1.

From the table we can conclude that all of the sample classes with the exception suffix extend the java.lang.Exception class, or have super class that extends this class. Furthermore, all of the classes contain a constructor, that calls a constructor of a super class (the Exception class). This is done using the "super" keyword.

Since the use of the super keyword is not detectable in Java byte-code, we will define our new PSP for the isolation of the Exception suffix in Java classes as: "A class which (indirectly) extends the java.lang.Exception class." We call this PSP the "Exceptional" pattern.

Factory

The observations made during the manual evaluation of 15 random classes with the Factory suffix, are shown in table E.2.

From the table we can conclude that the properties of the classes with the factory suffix are more varied. This makes it hard to find similarities between the classes. Therefore we try to create a definition using the description of the "Abstract Factory" and "Factory Method" design patterns form Gamma et al. [5]. We define the PSP for the Factory suffix as follows: "A class that implements an interface or extends an abstract class. At least of the implemented or overridden methods is a Factory Method." We define a "Factory Method" as: "A method of which the return type is abstract or an interface, and which creates an object that is a subtype or an implementation of the return type."

We can see that 9 out of 15 classes from table E.2 could be determined that they contain a factory method. 5 of of those also implement an interface or extend an abstract class. So by using this definition we are expected to isolate roughly one third of the factory suffixes.

Loosening the definition (For example, not requiring a factory class to implement an interface or extend an abstract class) might group more factory suffixes in a concept, but is also likely to add more other suffixes to the concept.

3.4 Case Study

In this section we discuss the results of the case study performed on the AspectJ project. Two separate studies are performed for the Exception and Factory suffix.

3.4.1 Exception

Table 3.1 shows the occurrence information of the Exception suffix for five concepts containing the most classes with that suffix. These concepts are created using micro patterns. The concept with Sink micro pattern (combination) contains by far the most classes with Exception suffix. Therefore, we will focus on this concept during the comparison between the micro patterns en the Exceptional PSP.

Micro Patterns	Suffix in concept	Suffix not in concept	Other suffixes in concept*
Sink	2253 (81.2%)	520 (18.8%)	11743 (83.9%)
Taxonomy & Sink	974 (35.1%)	1799 (64.9%)	1483 (60.4%)
Extender	454 (16.4%)	2319 (83.6%)	10715 (95.9%)
Sink & Extender	337 (12.1%)	2436 (87.9%)	1371 (80.3%)
Overrider	154 (5.6%)	2619 (94.4%)	7777 (98.0%)

Table 3.1: Occurrence information of the Exception suffix for five concepts containing the most classes with that suffix. * percentage = (Concept size - Exception suffixes) / Concept size

Table 3.2 shows the same kind of information for the Exceptional PSP. When we compare the numbers with the Sink micro pattern concept, we can not only see that the PSP captures more Exception suffixes compared to the Sink micro pattern, but also excludes almost all other micro patterns from the concept. This is a significant improvement in isolation compared to the Sink micro pattern.

Pattern	Suffix in concept	Suffix not in concept	Other suffixes in concept*
Exceptional	2723 (98.2%)	49 (1.8%)	216 (0.1%)

Table 3.2: Occurrence information of the Exception suffix using the Exceptional PSP. * = see table 3.1.

3.4.2 Factory

Table 3.3 shows the occurrence information of the Factory suffix for five concepts containing the most classes with that suffix. These concepts are created using micro patterns. The concept with Stateless micro pattern (combination) contains by far the most classes with Factory suffix. Therefore, we will focus on this concept during the comparison between the micro patterns and the Manufacturing PSP.

Micro Patterns	Suffix in concept	Suffix not in concept	Other suffixes in concept*
Stateless	1269 (43.0%)	1685 (57.0%)	11355 (90.0%)
Extender	270 (9.1%)	2684 (90.9%)	10899 (97.6%)
Common State	229 (7.8%)	2725 (92.2%)	3536 (93.9%)
Implementor	202 (6.8%)	2752 (93.2%)	5360 (96.4%)
Function Pointer & Stateless	194 (6.7%)	2760 (93.3%)	820 (80.7%)

Table 3.3: Occurrence information of the Factory suffix for five concepts containing the most classes with that suffix. * = (Concept size - Factory suffixes) / Concept size

Table 3.4 shows the occurrence information of the same suffix regarding the Manufacturing PSP. We can see that the Stateless pattern seems to be better at capturing the Factory suffix, than the Manufacturing PSP. Note however, that the absolute amount of other suffixes is much smaller for the manufacturing pattern, relative to that amount in the concept of the stateless micro pattern. This means the influence of the Factory suffix is likely to be smaller for this concept during the analysis phase.

Pattern	Suffix in concept	Suffix not in concept	Other suffixes in concept*
Manufacturing	325 (11.0%)	2627 (89.0%)	3228 (90.8%)

Table 3.4: Occurrence information of the Factory suffix using the Manufacturing PSP. * = see table 3.3

Now that we know the isolating properties of the Stateless pattern regarding the Factory suffix, we can try to improve the isolation of it by refining the Manufacturing PSP. Since a lot of factory classes do not seem to implement an interface or extend an abstract class, we try to increase the amount of factory suffixes by dropping this requirement and therefore loosening the definition. Furthermore, we require the factory to be stateless, to tighten the definition somewhat. The results are shown in table 3.5.

Pattern	Suffix in concept	Suffix not in concept	Other suffixes in concept*
Manufacturing	370 (12.5%)	2584 (87.5%)	1294 (77.8%)

Table 3.5: Isolation of the Factory suffix using the redefined Manufacturing PSP. * = see table 3.3

Although the results are not spectacular, there is a significant improvement in isolation. The number of factory suffixes in the concept increased only slightly, but more significantly, the amount of other suffixes decreased by 13%. Further refinements of the Manufacturing PSP might further improve the results.

3.5 Discussion

In this chapter we showed that it is possible to improve the isolation of some suffixes with little effort, as with the Exception suffix. For other suffixes this task might be more complex, as for the Factory suffix. Programmers do not seem to implement a concept like the Factory pattern in a predictable manner. A large amount of variations occur (For example, a concrete Factory implementation might implement an interface or not), which make the detection of classes with such a suffix hard. This fact makes it questionable if we could ever isolate all the classes with the Factory suffix, which seems to be quite generically used.

However, we made some progress in this chapter. The Sink micro pattern might create a concept that contains a lot of classes with the Factory suffix, but also contains a lot of other classes. The (redefined) Manufacturing PSP might capture less classes with the factory suffix, but also adds less other suffixes to the concept. This might create more significant renaming suggestions during the analysis process. Further refinement of the PSP might improve the results more.

Another approach to isolate classes with a name containing a name of a design pattern, would be to apply more "heavyweight" design pattern recovery/detection techniques. These techniques are described by Antoniol for example [1].

3.6 Summary

In this chapter we addressed the following research question: Can the suffixes be isolated better using new patterns specific to this purpose? We can conclude that in some cases it is possible to define a PSP that isolates a suffix very well. In other cases, it might be hard or even impossible to come up with such a definition, making the production of (near) perfect renaming suggestions perhaps too ambitious. However, renaming suggestions might very well improve using, and become usable by the introduction of a new set of PSP.

Chapter 4

Conclusions and Future Work

In this work we have found that the analysis of the consistency and preciseness using micro patterns as proposed in chapter 2 does not yet yield results that are usable in practice. The micro patterns proposed by Gil and Maman do not seem to discriminate sufficiently enough between the implementation semantics of classes. This causes classes with certain suffixes to be present in a wide variety of class groups (formal concepts).

We have seen that some common suffixes are easy to isolate using patterns specific to the purpose of this research (PSP), instead of micro patterns. Other suffixes are harder, or even impossible to isolate well. These results imply that the renaming suggestions produced by the proposed analysis process could very well be improved by the introduction of a carefully assembled set of PSP, yielding (more) usable renaming suggestions.

Areas of future research can be summarized as follows:

- *Adequately Abstracting over Implementation Semantics:* To understand and improve software on a large scale, we need to find ways to abstract over implementation semantics of code automatically and adequately. Gil and Maman did pioneering work in this area, by the introduction of micro patterns. However, this tool did not seem to abstract adequately enough over these semantics for the purpose of this research.
  The refinement of the current micro pattern catalog is one way to more adequately abstract over the implementation semantics of classes, but these patterns might not be suitable for any type of software analysis. Therefore a set of PSPs could be introduced, for example for the purpose of this research.

- *Understanding Concepts in Software:* More research could be done in the field of concepts in software. We have seen that the concept of an Exception is implemented quite uniformly by different programmers. Other concepts, like that of a Factory, are more arbitrary implemented.
  By understanding the concepts that are generally known by programmers and how (consistently) these concepts are implemented, we could try to

asses how successful we might be at the automatic capturing of concepts in software. Furthermore, we could try to improve current theories using this information. For example, the abstraction over implementation semantics using micro patterns.

- *Effectiveness of Renaming Approaches:* Current research on consistency of class and method names (for example, Singer and Høst & Østvold), do not report the effectiveness (false positive, false negatives, etc) of the proposed methods. Some numbers are available, but very significant numbers (for example, as the result of a case study) are not available.
  The lack of these numbers makes it hard asses how relevant the results of newly proposed methods are.

Bibliography

[1] G. Antoniol, R. Fiutem, and L. Cristoforetti. Design pattern recovery in object-oriented software. In *Proceedings of the 6th International Workshop on Program Comprehension*, IWPC '98, pages 153–, Washington, DC, USA, 1998. IEEE Computer Society.

[2] S. Butler. Intt: Identifier name tokenisation tool. `http://oro.open.ac.uk/28352/`, March 2011.

[3] S. Butler, M. Wermelinger, Yijun Yu, and H. Sharp. Mining java class naming conventions. In *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, pages 93 –102, sept. 2011.

[4] F. Deissenboeck and M. Pizka. Concise and consistent naming. *Software Quality Control*, 14(3):261–282, September 2006.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addisson-Wesley, Toronto, Ontario. Canada, 1995.

[6] J. Gil and I. Maman. Micro patterns in java code. *SIGPLAN Not.*, 40(10):97–116, October 2005.

[7] R.L. Glass. *Facts and Fallacies of Software Engineering*. Addison-Wesley Professional, October 2002.

[8] E. Høst and B. Østvold. Debugging method names. In Sophia Drossopoulou, editor, *ECOOP 2009 – Object-Oriented Programming*, volume 5653 of *Lecture Notes in Computer Science*, pages 294–317. Springer Berlin / Heidelberg, 2009. 10.1007/978-3-642-03013-0-14.

[9] Centrum Wiskunde & Informatica. Rascal - webhome. `http://www.rascal-mpl.org/`. [Accesed: may, 2012].

[10] K Karlsen and B.M. Østvold. lancelot-eclipse - a method name analyzer for eclipse - google project hosting. `http://code.google.com/p/lancelot-eclipse/`. [Accesed: apr, 2012].

[11] D. Lawrie, C.F. Morrell, and D. Binkley. What's in a name? a study of identifiers. In *In 14th International Conference on Program Comprehension*, pages 3–12. IEEE Computer Society, 2006.

[12] R.C. Martin. *Clean Code: A handbook of agile software craftsmanship*. Prentice Hall, 2009.

[13] OW2. ASM Home Page. `http://asm.ow2.org/`. [Accesed: apr, 2012].

[14] V. Rajlich and N. Wilde. The role of concepts in program comprehension. In *In IWPC '02*, pages 271–278, 2002.

[15] J. Singer, G. Brown, M. Luján, A. Pocock, and P. Yiapanis. Fundamental nano-patterns to characterize and classify java methods. *Electronic Notes in Theoretical Computer Science*, 253(7):191 – 204, 2010.

[16] J. Singer and C. Kirkham. Exploiting the correspondence between micro patterns and class names. *Source Code Analysis and Manipulation, IEEE International Workshop on*, 0:67–76, 2008.

[17] Sun. *Java Code Conventions*. Sun, 1997.

[18] E. Tempero, C. Anslow, J. Dietrich, T. Han, Jing Li, M. Lumpe, H. Melton, and J. 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.

Appendix A

Tokenizer and Tagger Accuracies

In this appendix the accuracies of two tokenizers and two taggers from related research are determined. This is done by manually verification of their output for random class names from the corpus described in section 2.3.3. It is not necessary for this research to strive for perfect tokenization and tagging, but more accurate results will probably result in more usable data. The most accurate tokenizer and tagger will be used during this research.

A.1 Tokenizer Accuracies

The accuracy of the following tokenizers from related research are evaluated:

- *Lancelot algorithm:* The tokenization algorithm extracted from the Lancelot Eclipse plug-in. This plug-in is based on the work by Høst [10].
- *Intt:* Identifier name tokenization tool by Butler. A Java library that is claimed have an overall accuracy of 96.5% for class names. Uses an "oracle" to more accurately tokenize terms and abbreviations like "J2SE" [2].

Manual verification of output of the two tokenizers indicate that the Intt tokenizer has an accuracy of 96.4 - 98.0%, while the lancelot is 94.8 - 96.8% accurate (n = 251, see table A.1 for the output). The accuracy is given in ranges, because I found it hard to judge the correctness of some class names without knowing the purpose of the class, and therefore the correctness of the tokenization output. Results suggest that the Intt tokenizer is slightly more accurate. Therefore this tokenizer is used for the experiment.
Remarkable is that the Intt tokenizer successfully tokenizes "J2EEResourceBase" into "J2EE-Resource-Base", but incorrectly tokenizes "Guid" into "G-uid".

This is probably because it prefers to tokenize "uid", because it is contained in the oracle.

Input	Intt	Incorrect (Intt)	Lancelot	Incorrect (Lancelot)	Remarks
CommonBehaviorFactory	Common Behavior Factory		common behavior factory		
XMLTestCase	XML Test Case		XML test case		
TestGraphBaseToString	Test Graph Base To String		test graph base to string		
CopyInfoParser	Copy Info Parser		copy info parser		
ColumnText	Column Text		column text		
WrongDocumentErr	Wrong Document Err		wrong document err		
JmsConnectionMetaData	Jms Connection Meta Data		jms connection meta data		
CompletionPane	Completion Pane		completion pane		
TableViewerAction	Table Viewer Action		table viewer action		
SpringRepeat	Spring Repeat		spring repeat		
PostExeNode	Post Exe Node		post exe node		
MemoryGraphPanel	Memory Graph Panel		memory graph panel		
LocalTrackerPlugin	Local Tracker Plugin		local tracker plugin		
listNetworksListener	list Networks Listener		list networks listener		
VersionColumns	Version Columns		version columns		
UShortToUInt	U Short To UI nt	1	u short to u int		
AntElement	Ant Element		ant element		
FragReceiver	Frag Receiver		frag receiver		
BorderLeftStyle	Border Left Style		border left style		
LicenseContentProvider	License Content Provider		license content provider		
NetBeansOrgEntry	Net Beans Org Entry	1	net beans org entry	1	
DomainServerSocket	Domain Server Socket		domain server socket		
PrintStarter	Print Starter		print starter		
Jdbc3PoolingDataSource	Jdbc3 Pooling Data Source		jdbc 3 pooling data source		Both are ok
NavigatorContentService	Navigator Content Service		navigator content service		
Unsigned16	Unsigned 16		unsigned 16		
FileTableContentProvider	File Table Content Provider		file table content provider		
ComponentMapper	Component Mapper		component mapper		
InterpolationMethodTypeBinding	Interpolation Method Type Binding		interpolation method type binding		
StyledEditorKit	Styled Editor Kit		styled editor kit		
Recolor	Recolor		recolor		
IRepositoryQuery	I Repository Query		i repository query		
InputTransferSelectDirective	Input Transfer Select Directive		input transfer select directive		
ConfigSexpression	Config Sexpression		config sexpression		
PEMReader	PEM Reader		PEM reader		

JRRtfExporterContext	JR Rtf Exporter Context	0	JR rtf exporter context	0	
BindYellow	Bind Yellow		bind yellow		
DynamicIdentityPolicy	Dynamic Identity Policy		dynamic identity policy		
UninstallFeatureAction	Uninstall Feature Action		uninstall feature action		
DisabledFacet	Disabled Facet		disabled facet		
ExternalToolsBuilderTab	External Tools Builder Tab		external tools builder tab		
NullPointerException	Null Pointer Exception		null pointer exception		
ASTProject	AST Project		AST project		
T4CInputStream	T 4 C Input Stream	1	t 4 c input stream	1	
JDKProvider	JDK Provider		JDK provider		
FailureDetector	Failure Detector		failure detector		
DefinitionKindHolder	Definition Kind Holder		definition kind holder		
DelegatingIoHandler	Delegating Io Handler		delegating io handler		
SettingsTabJava	Settings Tab Java		settings tab java		
QuadTo	Quad To		quad to		
OutputMultiplexor	Output Multiplexor		output multiplexor		
HTMLTableComponent	HTML Table Component		HTML table component		
SybasePlatform	Sybase Platform		sybase platform		
HttpSessionBindingListener	Http Session Binding Listener		http session binding listener		
FooWorkManager	Foo Work Manager		foo work manager		
FilteredSourcePackage	Filtered Source Package		filtered source package		
MenuDetectListener	Menu Detect Listener		menu detect listener		
AntObject	Ant Object		ant object		
FileStatsCacheItem	File Stats Cache Item		file stats cache item		
MruCacheStorage	Mru Cache Storage		mru cache storage		
DelegatingTilesRequestProcessor	Delegating Tiles Request Processor		delegating tiles request processor		
FolderNode	Folder Node		folder node		
IWorkerStatusChangeListener	I Worker Status Change Listener		i worker status change listener		
AgentHandler	Agent Handler		agent handler		
StringTypeDescriptor	String Type Descriptor		string type descriptor		
RtfMapper	Rtf Mapper		rtf mapper		
KateBadPacketException	Kate Bad Packet Exception		kate bad packet exception		
SubjectKeyIDRequest	Subject Key ID Request		subject key ID request		
HTMLIndentEngineBeanInfo	HTML Indent Engine Bean Info		HTML indent engine bean info		
BaseSVGNumberList	Base SVG Number List		base SVG number list		
AnnotationMark	Annotation Mark		annotation mark		

ActionAddClassifierRoleBase	Action Add Classifier Role Base		action add classifier role base		
InitClassDiagram	Init Class Diagram		init class diagram		
FreeCol	Free Col		free col		
JavaSourceFilePrintWriter	Java Source File Print Writer		java source file print writer		
MemberOptimizationInfoSetter	Member Optimization Info Setter		member optimization info setter		
DefaultScheduledExecutorFactory	Default Scheduled Executor Factory		default scheduled executor factory		
je	je		je		
DataFlavorComparator	Data Flavor Comparator		data flavor comparator		
WhiteSharkWeapon	White Shark Weapon		white shark weapon		
MoreTypes	More Types		more types		
ContentHandlerAdaptor	Content Handler Adaptor		content handler adaptor		
UpdateUnitProviderPanel	Update Unit Provider Panel		update unit provider panel		
OriginatorIdentifierOrKey	Originator Identifier Or Key		originator identifier or key		
ClassRenamer	Class Renamer		class renamer		
NetworkRegistryMBean	Network Registry M Bean		network registry m bean		
AFProxy	AF Proxy		AF proxy		
SystemFlavorMap	System Flavor Map		system flavor map		
UseCasesFactory	Use Cases Factory		use cases factory		
QosPolicyCountHelper	Qos Policy Count Helper		qos policy count helper		
MemoryViewSynchronizationService	Memory View Synchronization Service		memory view synchronization service		
UnorderableException	Unorderable Exception		unorderable exception		
SvnHookFactoryImpl	Svn Hook Factory Impl		svn hook factory impl		
DatabaseServiceImpl	Database Service Impl		database service impl		
EnableCommand	Enable Command		enable command		
SingleStream	Single Stream		single stream		
TaskConfigurationChecker	Task Configuration Checker		task configuration checker		
IElementReference	I Element Reference		i element reference		
WatchpointTypeChange	Watchpoint Type Change		watchpoint type change		
CatchClause	Catch Clause		catch clause		
DatatypeRef	Datatype Ref		datatype ref		
LL1Analyzer	LL1 Analyzer		LL 1 analyzer	1	
PropertySetter	Property Setter		property setter		
SACParserCSSmobileOKBasic1Constants	SAC Parser CSS mobile OK Basic1 Constants	0	SAC parser CS smobile OK basic 1 constants	0	
DividerPainter	Divider Painter		divider painter		
DOMInputImpl	DOM Input Impl		DOM input impl		

SimpleSequence	Simple Sequence		simple sequence		
CollapsedBorderSide	Collapsed Border Side		collapsed border side		
ToolBarButtonTag	Tool Bar Button Tag		tool bar button tag		
PlayerExploredTile	Player Explored Tile		player explored tile		
ValueDifferenceImpl	Value Difference Impl		value difference impl		
JavaModuleGlobals	Java Module Globals		java module globals		
Guid	G uid	1	guid	0	
SurfaceInterpolation	Surface Interpolation		surface interpolation		
RepositoryContentMetadata	Repository Content Metadata		repository content metadata		
EStringToStringMapEntryImpl	E String To String Map Entry Impl	0	e string to string map entry impl	0	
KeySortedCollectionHelper	Key Sorted Collection Helper		key sorted collection helper		
CompatibleExecutor	Compatible Executor		compatible executor		
X_T_ReportStatement	X T Report Statement		NameSplitter encountered unexpected character: _	1	
KLNFOptionPane	KLNF Option Pane		KLNF option pane		
DefineFunction	Define Function		define function		
NativeMachine	Native Machine		native machine		
SkipIndexWriter	Skip Index Writer		skip index writer		
CalendarData_mk	Calendar Data mk		NameSplitter encountered unexpected character: _	1	
RolloverMouseListener	Rollover Mouse Listener		rollover mouse listener		
JDBCEvent	JDBC Event		JDBC event		
BiffHeaderInput	Biff Header Input		biff header input		
ServiceProviderTypeValidator	Service Provider Type Validator		service provider type validator		
ReleaseListener	Release Listener		release listener		
JRAbstractCompiler	JR Abstract Compiler		JR abstract compiler		
SearchFilterReference	Search Filter Reference		search filter reference		
IIOWriteProgressListener	IIO Write Progress Listener		IIO write progress listener		
FinalStateClass	Final State Class		final state class		
ModifierKeyword	Modifier Keyword		modifier keyword		
AbstractPreferenceInitializer	Abstract Preference Initializer		abstract preference initializer		
AttributePanelListener	Attribute Panel Listener		attribute panel listener		
NoneLockManager	None Lock Manager		none lock manager		

JAXWSDeployerHookEJB3	JAXWS Deployer Hook EJB 3		JAXWS deployer hook EJB 3		
ListViewerAdapter	List Viewer Adapter		list viewer adapter		
ExpressionFactoryImpl	Expression Factory Impl		expression factory impl		
ZipEntryStorageEditorInput	Zip Entry Storage Editor Input		zip entry storage editor input		
TraceConfiguration	Trace Configuration		trace configuration		
LabelUI	Label UI		label UI		
OperationsCompartmentContainer	Operations Compartment Container		operations compartment container		
DebugManagerAboutAction	Debug Manager About Action		debug manager about action		
RelationOrJoin	Relation Or Join		relation or join		
CSSParseException	CSS Parse Exception		CSS parse exception		
RemovedCallbackFacetAbstract	Removed Callback Facet Abstract		removed callback facet abstract		
ForwardingAnnotatedAnnotation	Forwarding Annotated Annotation		forwarding annotated annotation		
JhlLogMessageChangePath	Jhl Log Message Change Path		jhl log message change path		
rdfparse	rdf parse		rdfparse	1	
XHTMLTagSerializer	XHTML Tag Serializer		XHTML tag serializer		
SortCalc	Sort Calc		sort calc		
RBCollationTables	RB Collation Tables		RB collation tables		
RenderException	Render Exception		render exception		
BusinessList	Business List		business list		
ValueTask	Value Task		value task		
HtmlEscape	Html Escape		html escape		
SchemaCopy	Schema Copy		schema copy		
ManagedPropertyDelegate	Managed Property Delegate		managed property delegate		
ImageUsingCacheProperty	Image Using Cache Property		image using cache property		
SortingJob	Sorting Job		sorting job		
TableRowSWTPaintListener	Table Row SWT Paint Listener		table row SWT paint listener		
MarkMapping	Mark Mapping		mark mapping		
IsCollectionContaining	Is Collection Containing		is collection containing		
ObjectFilter	Object Filter		object filter		
JmsSecurityException	Jms Security Exception		jms security exception		
QueueRendererData	Queue Renderer Data		queue renderer data		
CompactorDictBlock	Compactor Dict Block		compactor dict block		
CapturingELResolver	Capturing EL Resolver		capturing EL resolver		
t2	t 2	0	t 2	0	
MessagingItem	Messaging Item		messaging item		
JFacePreferences	J Face Preferences		j face preferences		
NameReference	Name Reference		name reference		

EnumDefIRHelper	Enum Def IR Helper		enum def IR helper		
RefState	Ref State		ref state		
StaticMethodName	Static Method Name		static method name		
TextPaneView	Text Pane View		text pane view		
JasperReportErrorHandler	Jasper Report Error Handler		jasper report error handler		
CompilationUnitVisitor	Compilation Unit Visitor		compilation unit visitor		
JDBCResourceMBean	JDBC Resource M Bean		JDBC resource m bean		
GeneratedOrderByLexer	Generated Order By Lexer		generated order by lexer		
TomcatResolver	Tomcat Resolver		tomcat resolver		
PdfCollectionItem	Pdf Collection Item		pdf collection item		
PluginValidationStatusHandler	Plugin Validation Status Handler		plugin validation status handler		
1TypeList	1 Type List		1 type list		
IIntroConstants	I Intro Constants		i intro constants		
ConfigViewPlatform	Config View Platform		config view platform		
NotificationResultDeserFactory	Notification Result Deser Factory		notification result deser factory		
DocFrame	Doc Frame		doc frame		
J2SELibraryTypeProvider	J2SE Library Type Provider		j 2 SE library type provider	1	
RemoveResultAction	Remove Result Action		remove result action		
ExpressionView	Expression View		expression view		
SpellingSuggestionRequest	Spelling Suggestion Request		spelling suggestion request		
IconRule	Icon Rule		icon rule		
JMIHyperlinkAction	JMI Hyperlink Action		JMI hyperlink action		
GoStateToIncomingTrans	Go State To Incoming Trans		go state to incoming trans		
JRXmlDataSource	JR Xml Data Source		JR xml data source		
JasperOpenCookie	Jasper Open Cookie		jasper open cookie		
MicrosoftSqlServerDialect	Microsoft Sql Server Dialect		microsoft sql server dialect		
NbJarURLConnection	Nb Jar URL Connection		nb jar URL connection		
RootWalker	Root Walker		root walker		
SynchronizedCounter	Synchronized Counter		synchronized counter		
NullLogWriter	Null Log Writer		null log writer		
ExtendedSelector	Extended Selector		extended selector		
LocalClientRequestImpl	Local Client Request Impl		local client request impl		
EntityManagerEditor	Entity Manager Editor		entity manager editor		
StandardHostMapper	Standard Host Mapper		standard host mapper		

NameMatchMethodPointcutAdvisor	Name Match Method Pointcut Advisor		name match method pointcut advisor		
ConvTable921	Conv Table 921		conv table 921		
ButtonBorder	Button Border		button border		
XSTypeImpl	XS Type Impl		XS type impl		
PMDException	PMD Exception		PMD exception		
StringDef	String Def		string def		
StyledLayerDescriptorImpl	Styled Layer Descriptor Impl		styled layer descriptor impl		
NodesL	Nodes L		nodes l		
CompoundSelectorIterator	Compound Selector Iterator		compound selector iterator		
JmsWrapperFactoryContainer	Jms Wrapper Factory Container		jms wrapper factory container		
ContentModuleImpl	Content Module Impl		content module impl		
DocFile	Doc File		doc file		
style	style		style		
HtmlDocument	Html Document		html document		
OrderedConfiguration	Ordered Configuration		ordered configuration		
CacheFileManagerStatsImpl	Cache File Manager Stats Impl		cache file manager stats impl		
FilterToCQL	Filter To CQL		filter to CQL		
DistributedDatabase	Distributed Database		distributed database		
GoStimulusToAction	Go Stimulus To Action		go stimulus to action		
ServiceExceptionReportHandler	Service Exception Report Handler		service exception report handler		
EditorActionBuilder	Editor Action Builder		editor action builder		
JsBracesMatcherFactory	Js Braces Matcher Factory		js braces matcher factory		
J2EEResourceBase	J2EE Resource Base		j 2 EE resource base	1	
ElementBindings	Element Bindings		element bindings		
MailFileSystemView	Mail File System View		mail file system view		
LineStripArrayState	Line Strip Array State		line strip array state		
EnableWatchExpressionAction	Enable Watch Expression Action		enable watch expression action		
RequestPartitioningComponentImpl	Request Partitioning Component Impl		request partitioning component impl		
CacheConfigurationMBean	Cache Configuration M Bean		cache configuration m bean		
AsciiCharacterTranslator	Ascii Character Translator		ascii character translator		
TestGanttRolloverButton	Test Gantt Rollover Button		test gantt rollover button		
TreeBasedTask	Tree Based Task		tree based task		
OnlyOneReturnRule	Only One Return Rule		only one return rule		
DataWriterHolder	Data Writer Holder		data writer holder		
StatefulSessionInterceptor	Stateful Session Interceptor		stateful session interceptor		

NativeTextHandler	Native Text Handler		native text handler		
UMLOperation	UML Operation		UML operation		
SPIAccessorImpl	SPI Accessor Impl		SPI accessor impl		
NDupFunction	ND up Function	1	n dup function		
GridDataFactory	Grid Data Factory		grid data factory		
ConstructorResultItem	Constructor Result Item		constructor result item		
ASTProperty	AST Property		AST property		
ReorgMessages	Reorg Messages		reorg messages		
Faulty		5		8	
Doubt		4		5	
Faulty and Doubt		9		13	

Table A.1: Results of the tokenization of random classes. A 0 is a doubt and a 1 is unknown.

A.2 Tagger Accuracies

The accuracy of the following taggers from related research are evaluated:

- *Lancelot algorithm:* Tagger used by the lancelot tool. This tagger is based on the WordNet library. In their work Høst and Østvold claim an accuracy of approximately 97%, although it is not clear if this is for individual words or full method names.
- *Tagger by Butler:* Stanford tagger trained by Butler for Java class names. Has a reported accuracy of 87% for whole class names [3].

Manual verification of the output of the two taggers indicates that the lancelot tagger is 90.5 - 94.5% accurate, and Butlers tagger 84.9 - 90.5% (n = 199, correctly tokenized class names shown in A.2). However, the lancelot tagger tags a lot of words as unknown. If we consider the unknown tags as faulty, the tagger is 64.8-68.8% accurate. Since class names that contain unknown tags are not very useful, it seems more sensible to use Butlers tagger. It should be noted that Butlers tagger seems to tag words as noun if it does not know the word type. However, since most class names consist of nouns, this strategy seems to work (although it is not a very sophisticated approach).
Note that by combining the Intt tokenizer and Butlers tagger we accomplish an accuracy of 81.8 - 88.6% in the process of class name analysis.

Input	Output (Butler)	Correct (Butler)	Output (Lancelot)	Correct (Lancelot)
Common Behavior Factory	Common/NN Behavior/NN Factory/NN	1	noun noun noun	1
XML Test Case	XML/NN Test/NN Case/NN		noun noun noun	

Test Graph Base To String	Test/NN Graph/NN Base/NN To/NN String/NN	0	noun noun noun unknown noun	2
Copy Info Parser	Copy/NN Info/NN Parser/NN		noun noun noun	
Column Text	Column/NN Text/NN		noun noun	
Wrong Document Err	Wrong/NN Document/NN Err/NN	1	noun noun verb	1
Jms Connection Meta Data	Jms/NN Connection/NN Meta/NN Data/NN	0	noun noun unknown noun	2
Completion Pane	Completion/NN Pane/NN		noun noun	
Table Viewer Action	Table/JJ Viewer/NN Action/NN	1	noun noun noun	
Spring Repeat	Spring/NN Repeat/NN		noun noun	
Post Exe Node	Post/NN Exe/NN Node/NN		noun unknown noun	2
Memory Graph Panel	Memory/NN Graph/NN Panel/NN		noun noun noun	
Local Tracker Plugin	Local/JJ Tracker/NN Plugin/NN		noun noun unknown	2
list Networks Listener	list/NN Networks/NNS Listener/NN		noun noun noun	
Version Columns	Version/NN Columns/NNS		noun noun	
Ant Element	Ant/NN Element/NN		noun noun	
Frag Receiver	Frag/NN Receiver/NN		unknown noun	2
Border Left Style	Border/NN Left/NN Style/NN		noun noun noun	
License Content Provider	License/NN Content/NN Provider/NN		noun noun noun	
Domain Server Socket	Domain/NN Server/NN Socket/NN		noun noun noun	
Print Starter	Print/NN Starter/NN		noun noun	
Jdbc3 Pooling Data Source	Jdbc3/NN Pooling/NN Data/NN Source/NN	0	unknown adjective noun noun	2
Navigator Content Service	Navigator/NN Content/NN Service/NN		noun noun noun	
Unsigned 16	Unsigned/JJ 16/CD		adjective number	
File Table Content Provider	File/NN Table/JJ Content/NN Provider/NN	1	noun noun noun noun	

Component Mapper	Component/NN Mapper/NN		noun noun	
Interpolation Method Type Binding	Interpolation/NN Method/NN Type/NN Binding/NN		noun noun noun noun	
Styled Editor Kit	Styled/JJ Editor/NN Kit/NN		adjective noun noun	
Recolor	Recolor/NN	1	unknown	2
I Repository Query	I/NN Repository/NN Query/NN		noun noun noun	
Input Transfer Select Directive	Input/NN Transfer/NN Select/NN Directive/NN		noun noun adjective noun	0
Config Sexpression	Config/NN Sexpression/NN		unknown unknown	2
PEM Reader	PEM/NN Reader/NN		noun noun	
Bind Yellow	Bind/NN Yellow/NN	0	noun noun	0
Dynamic Identity Policy	Dynamic/NN Identity/NN Policy/NN	1	noun noun noun	1
Uninstall Feature Action	Uninstall/NN Feature/NN Action/NN		unknown noun noun	2
Disabled Facet	Disabled/JJ Facet/NN		noun noun	1
External Tools Builder Tab	External/NN Tools/NNS Builder/NN Tab/NN	1	noun noun noun noun	1
Null Pointer Exception	Null/NN Pointer/NN Exception/NN		noun noun noun	
AST Project	AST/NN Project/NN		noun noun	
JDK Provider	JDK/NN Provider/NN		noun noun	
Failure Detector	Failure/NN Detector/NN		noun noun	
Definition Kind Holder	Definition/NN Kind/NN Holder/NN		noun noun noun	
Delegating Io Handler	Delegating/NN Io/NN Handler/NN	0	noun noun noun	0
Settings Tab Java	Settings/NNS Tab/NN Java/NN		noun noun noun	
Quad To	Quad/NN To/NN	0	noun unknown	2
Output Multiplexor	Output/NN Multiplexor/NN		noun unknown	2
HTML Table Component	HTML/NN Table/JJ Component/NN	1	noun noun noun	
Sybase Platform	Sybase/NN Platform/NN		unknown noun	2
Http Session Binding Listener	Http/NN Session/NN Binding/NN Listener/NN		noun noun noun noun	

Foo Work Manager	Foo/NN Work/NN Manager/NN		unknown noun noun	2
Filtered Source Package	Filtered/JJ Source/NN Package/NN		adjective noun noun	
Menu Detect Listener	Menu/NN Detect/NN Listener/NN	0	noun verb noun	1
Ant Object	Ant/NN Object/NN		noun noun	
File Stats Cache Item	File/NN Stats/NNS Cache/NN Item/NN		noun unknown noun noun	2
Mru Cache Storage	Mru/NN Cache/NN Storage/NN		unknown noun noun	2
Delegating Tiles Request Processor	Delegating/NN Tiles/NNS Request/NN Processor/NN	1	noun noun noun noun	1
Folder Node	Folder/NN Node/NN		noun noun	
I Worker Status Change Listener	I/NN Worker/NN Status/NNS Change/NN Listener/NN		noun noun noun noun noun	
Agent Handler	Agent/NN Handler/NN		noun noun	
String Type Descriptor	String/NN Type/NN Descriptor/NN		noun noun noun	
Rtf Mapper	Rtf/NN Mapper/NN		noun noun	
Kate Bad Packet Exception	Kate/NN Bad/NN Packet/NN Exception/NN		unknown noun noun noun	2
Subject Key ID Request	Subject/NN Key/NN ID/NN Request/NN		noun noun noun noun	
HTML Indent Engine Bean Info	HTML/NN Indent/NN Engine/NN Bean/NN Info/NN		noun noun noun noun noun	
Base SVG Number List	Base/NN SVG/NN Number/NN List/NN		noun noun noun noun	
Annotation Mark	Annotation/NN Mark/NN		noun noun	
Action Add Classifier Role Base	Action/NN Add/NN Classifier/NN Role/NN Base/NN		noun noun noun noun noun	
Init Class Diagram	Init/NN Class/NN Diagram/NN	1	unknown noun noun	2
Free Col	Free/NN Col/NN		noun noun	

Java Source File Print Writer	Java/NN Source/NN File/NN Print/NN Writer/NN		noun noun noun noun noun	
Member Optimization Info Setter	Member/NN Optimiza-tion/NN Info/NN Set-ter/NN		noun noun noun noun	
Default Scheduled Executor Factory	Default/NN Scheduled/JJ Executor/NN Factory/NN		noun adjective noun noun	
je	je/NN		unknown	2
Data Flavor Comparator	Data/NN Fla-vor/NN Com-parator/NN		noun noun un-known	2
White Shark Weapon	White/NN Shark/NN Weapon/NN		noun noun noun	
More Types	More/NN Types/NNS		noun noun	
Content Handler Adaptor	Content/NN Handler/NN Adaptor/NN		noun noun noun	
Update Unit Provider Panel	Update/NN Unit/NN Provider/NN Panel/NN		noun noun noun noun	
Originator Identifier Or Key	Originator/NN Identifier/NN Or/NN Key/NN	0	noun noun noun noun	0
Class Renamer	Class/NN Re-namer/NN		noun unknown	2
Network Registry M Bean	Network/NN Registry/NN M/NN Bean/NN		noun noun noun noun	
AF Proxy	AF/NN Proxy/NN		noun noun	
System Flavor Map	System/NN Flavor/NN Map/NN		noun noun noun	
Use Cases Factory	Use/NN Cases/NNS Factory/NN		noun noun noun	
Qos Policy Count Helper	Qos/NN Policy/NN Count/NN Helper/NN		unknown noun noun noun	2
Memory View Synchroniza-tion Service	Memory/NN View/NN Synchroniza-tion/NN Ser-vice/NN		noun noun noun noun	
Unorderable Exception	Unorderable/JJ Exception/NN		adjective noun	
Svn Hook Factory Impl	Svn/NN Hook/NN Factory/NN Impl/NN		noun noun noun unknown	2
Database Service Impl	Database/NN Service/NN Impl/NN		noun noun un-known	2

Enable Command	Enable/JJ Command/NN		verb noun	1
Single Stream	Single/NN Stream/NN		noun noun	
Task Configuration Checker	Task/NN Configuration/NN Checker/NN		noun noun noun	
I Element Reference	I/NN Element/NN Reference/NN		noun noun noun	
Watchpoint Type Change	Watchpoint/NN Type/NN Change/NN		unknown noun noun	2
Catch Clause	Catch/NN Clause/NN		noun noun	
Datatype Ref	Datatype/NN Ref/NN		unknown noun	2
Property Setter	Property/NN Setter/NN		noun noun	
Divider Painter	Divider/NN Painter/NN		noun noun	
DOM Input Impl	DOM/NN Input/NN Impl/NN		noun noun unknown	2
Simple Sequence	Simple/NN Sequence/NN	1	noun noun	1
Collapsed Border Side	Collapsed/JJ Border/NN Side/NN		adjective noun noun	
Tool Bar Button Tag	Tool/NN Bar/NN Button/NN Tag/NN		noun noun noun noun	
Player Explored Tile	Player/NN Explored/JJ Tile/NN		noun adjective noun	
Value Difference Impl	Value/NN Difference/NN Impl/NN		noun noun unknown	2
Java Module Globals	Java/NN Module/NN Globals/NNS		noun noun unknown	2
Surface Interpolation	Surface/NN Interpolation/NN		noun noun	
Repository Content Metadata	Repository/NN Content/NN Metadata/NN		noun noun noun	
Key Sorted Collection Helper	Key/NN Sorted/JJ Collection/NN Helper/NN		noun adjective noun noun	
Compatible Executor	Compatible/JJ Executor/NN		adjective noun	
KLNF Option Pane	KLNF/NN Option/NN Pane/NN		noun noun noun	
Define Function	Define/NN Function/NN	1	verb noun	0
Native Machine	Native/JJ Machine/NN		noun noun	
Skip Index Writer	Skip/NN Index/NN Writer/NN		noun noun noun	
Rollover Mouse Listener	Rollover/NN Mouse/NN Listener/NN		noun noun noun	

JDBC Event	JDBC/NN Event/NN		noun noun	
Biff Header Input	Biff/NN Header/NN Input/NN		noun noun noun	
Service Provider Type Validator	Service/NN Provider/NN Type/NN Validator/NN		noun noun noun unknown	2
Release Listener	Release/NN Listener/NN		noun noun	
JR Abstract Compiler	JR/NN Abstract/NN Compiler/NN		noun noun noun	
Search Filter Reference	Search/NN Filter/NN Reference/NN		noun noun noun	
IIO Write Progress Listener	IIO/NN Write/NN Progress/NN Listener/NN	0	noun verb noun noun	0
Final State Class	Final/JJ State/NN Class/NN		noun noun noun	
Modifier Keyword	Modifier/NN Keyword/NN		noun unknown	2
Abstract Preference Initializer	Abstract/NN Preference/NN Initializer/NN		noun noun unknown	2
Attribute Panel Listener	Attribute/NN Panel/NN Listener/NN		noun noun noun	
None Lock Manager	None/NN Lock/NN Manager/NN		noun noun noun	
JAXWS Deployer Hook EJB 3	JAXWS/NN Deployer/NN Hook/NN EJB/NN 3/CD		noun unknown noun noun number	2
List Viewer Adapter	List/NN Viewer/NN Adapter/NN		noun noun noun	
Expression Factory Impl	Expression/NN Factory/NN Impl/NN		noun noun unknown	2
Zip Entry Storage Editor Input	Zip/NN Entry/NN Storage/NN Editor/NN Input/NN		noun noun noun noun noun	
Trace Configuration	Trace/NN Configuration/NN		noun noun	
Label UI	Label/NN UI/NN		noun noun	
Operations Compartment Container	Operations/NNS Compartment/NN Container/NN		noun noun noun	
Debug Manager About Action	Debug/NN Manager/NN About/NN Action/NN	1	verb noun adjective noun	1
Relation Or Join	Relation/NN Or/NN Join/NN		noun noun noun	

CSS Parse Exception	CSS/NN Parse/NN Exception/NN	0	noun verb noun	0
Removed Callback Facet Abstract	Removed/JJ Callback/NN Facet/NN Abstract/NN		adjective noun noun noun	
Forwarding Annotated Annotation	Forwarding/JJ Annotated/JJ Annotation/NN		noun adjective noun	
Jhl Log Message Change Path	Jhl/NN Log/NN Message/NN Change/NN Path/NN		unknown noun noun noun noun	2
XHTML Tag Serializer	XHTML/NN Tag/NN Serializer/NN		noun noun unknown	2
Sort Calc	Sort/NN Calc/NN		noun unknown	2
RB Collation Tables	RB/NN Collation/NN Tables/NNS		noun noun noun	
Render Exception	Render/NN Exception/NN		noun noun	
Business List	Business/NN List/NN		noun noun	
Value Task	Value/NN Task/NN		noun noun	
Html Escape	Html/NN Escape/NN		noun noun	
Schema Copy	Schema/NN Copy/NN		noun noun	
Managed Property Delegate	Managed/JJ Property/NN Delegate/NN		adjective noun noun	
Image Using Cache Property	Image/NN Using/NN Cache/NN Property/NN	1	noun noun noun noun	1
Sorting Job	Sorting/NN Job/NN		noun noun	
Table Row SWT Paint Listener	Table/JJ Row/NN SWT/NN Paint/NN Listener/NN	1	noun noun noun noun noun	
Mark Mapping	Mark/NN Mapping/NN		noun noun	
Is Collection Containing	Is/NN Collection/NN Containing/NN	1	noun noun adjective	0
Object Filter	Object/NN Filter/NN		noun noun	
Jms Security Exception	Jms/NN Security/NN Exception/NN		noun noun noun	
Queue Renderer Data	Queue/NN Renderer/NN Data/NN		noun unknown noun	2
Compactor Dict Block	Compactor/NN Dict/NN Block/NN		unknown unknown noun	2
Capturing EL Resolver	Capturing/NN EL/NN Resolver/NN	1	adjective noun unknown	2

Messaging Item	Messaging/NN Item/NN		noun noun	
J Face Preferences	J/NN Face/NN Preferences/NNS		noun noun noun	
Name Reference	Name/NN Reference/NN		noun noun	
Enum Def IR Helper	Enum/NN Def/NN IR/NN Helper/NN		unknown unknown noun noun	2
Ref State	Ref/NN State/NN		noun noun	
Static Method Name	Static/JJ Method/NN Name/NN		noun noun noun	
Text Pane View	Text/NN Pane/NN View/NN		noun noun noun	
Jasper Report Error Handler	Jasper/NN Report/NN Error/NN Handler/NN		noun noun noun noun	
Compilation Unit Visitor	Compilation/NN Unit/NN Visitor/NN		noun noun noun	
JDBC Resource M Bean	JDBC/NN Resource/NN M/NN Bean/NN		noun noun noun noun	
Generated Order By Lexer	Generated/JJ Order/NN By/NN Lexer/NN	1	adjective noun adverb unknown	2
Tomcat Resolver	Tomcat/NN Resolver/NN		noun unknown	2
Pdf Collection Item	Pdf/NN Collection/NN Item/NN		noun noun noun	
Plugin Validation Status Handler	Plugin/NN Validation/NN Status/NNS Handler/NN		unknown noun noun noun	2
1 Type List	1/CD Type/NN List/NN		number noun noun	
I Intro Constants	I/NN Intro/NN Constants/NNS		noun noun noun	
Config View Platform	Config/NN View/NN Platform/NN		unknown noun noun	2
Notification Result Deser Factory	Notification/NN Result/NN Deser/NN Factory/NN		noun noun unknown noun	2
Doc Frame	Doc/NN Frame/NN		noun noun	
Remove Result Action	Remove/NN Result/NN Action/NN		noun noun noun	
Expression View	Expression/NN View/NN		noun noun	
Spelling Suggestion Request	Spelling/NN Suggestion/NN Request/NN		noun noun noun	
Icon Rule	Icon/NN Rule/NN		noun noun	

JMI Hyperlink Action	JMI/NN Hyperlink/NN Action/NN		noun noun noun	
Go State To Incoming Trans	Go/NN State/NN To/NN Incoming/NN Trans/NNS	0	noun noun unknown noun unknown	2
JR Xml Data Source	JR/NN Xml/NN Data/NN Source/NN		noun noun noun noun	
Jasper Open Cookie	Jasper/NN Open/NN Cookie/NN		noun noun noun	
Microsoft Sql Server Dialect	Microsoft/NN Sql/NN Server/NN Dialect/NN		unknown noun noun noun	2
Nb Jar URL Connection	Nb/NN Jar/NN URL/NN Connection/NN		noun noun noun noun	
Root Walker	Root/NN Walker/NN		noun noun	
Synchronized Counter	Synchronized/JJ Counter/NN		adjective noun	
Null Log Writer	Null/NN Log/NN Writer/NN		noun noun noun	
Extended Selector	Extended/JJ Selector/NN		adjective noun	
Local Client Request Impl	Local/JJ Client/NN Request/NN Impl/NN		noun noun noun unknown	2
Entity Manager Editor	Entity/NN Manager/NN Editor/NN		noun noun noun	
Standard Host Mapper	Standard/NN Host/NN Mapper/NN		noun noun noun	
Name Match Method Pointcut Advisor	Name/NN Match/NN Method/NN Pointcut/NN Advisor/NN		noun noun noun unknown noun	2
Conv Table 921	Conv/NN Table/JJ 921/NN	1	unknown noun number	2
Button Border	Button/NN Border/NN		noun noun	
XS Type Impl	XS/NN Type/NN Impl/NN		noun noun unknown	2
Fault		19		11
Doubt		11		8
Fault and doubt		30		19
Contains unknown				51

Table A.2: Results of the tagging of random classes. A 0 is a doubt, a 1 is fault and a 2 is contains unkown.

Appendix B

Observations and Assumptions Regarding the Micro Pattern Tool

The observations made by comparing the output of the micro pattern tool of Maman and my tool are the following (per micro pattern):

- *Box:* The tool of Maman evaluates the FunctionObject class as box, while its function does not mutate its instance variable. The InnerClassTest$InnerClass file is also evaluated as box, while it has no fields.
- *Canopy:* The tool of Maman evaluates the Taxonomy and TaxonomyBase classes as Canopy, while there is never a variable assigned. My tool does evaluate InnerClassTest$InnerClass as inner class. This must be corrected, or inner classes must be skipped.
- *Common State:* The tool of Maman evaluates the Main class as Common State, while it has no static field. Furthermore the tool does not evaluate the CommonState class as Common State, while it has only a static field. And the Pool, Stateless and AugmentedType classes are not evaluated as common state, maybe because their static fields are also final. The PseudoClass class is not evaluated as common state, maybe because it is abstract. My tool evaluated the Trait class as common state. This is corrected.
- *Designator:* The tool of Maman does not evaluate the Designator class (no parents other than object), which is empty as Designator. The Joiner class is also empty, but also not evaluated as Designator. It seems that classes are never evaluated as a designator. This possibility is described in the paper, though. My tool does evaluate the Joiner interface as Designator. The tool of Maman not. It seems that the Designator is required to extend just one interface.

- *Extender:* Ok, after correcting my tool. Apparently only extending Object is not evaluated as Extender by the tool of Maman.
- *Function Object:* The tool of Maman does evaluate the CobolLike class as Function Object, while it has only a static method and a static field. An instance method and field are required for Cobol Like. Common state is also evaluated as function object, while it has only static fields. Apparently, super classes must also be evaluated.
- *Function Pointer:* The tool of Maman evaluates the Stateless class as Function Pointer while it has a static final field. No fields are allowed for Function Pointer pattern. Apparently, super classes must also be evaluated.
- *Immutable:* The Immutable class not evaluated as Immutable by the tool of Maman, while it conforms to the description of an Immutable class from the paper.
- *Pool:* The Pool class not evaluated as Pool the tool of Maman. Maybe, because the final field is also private.
- *Pseudo Class:* The tool of Maman does not evaluate the Augemented-Type class as pseudo class. Maybe, because the fields of this class are also final. The PseudoClass class is also not evaluated, while it conforms to the description from the paper. The CobolLike class is detected by my tool as pseudo class, but ok after correcting my tool. Furthermore, The StateMachine interface is evaluated by my tool as pseudo class, but interface can be no pseudo class.
- *Pure type:* The tool of Manan evaluates classes with no methods also as Pure Type.
- *Record:* Ok after correcting my tool. Apparently field cannot be static. This is not mentioned in the paper.
- *Sink:*The Main class is not evaluated as Sink by the tool of Maman, while it has only an empty main method. The Sink class is not evaluated as Sink, probably because the class calls its own method. This is not allowed by the description in the micro patterns paper, but it is allowed according to the description in table 1 of that paper. My tool elevated Box as Sink, because it skipped constructor calls, this is corrected. And the tool evaluated interfaces as sink. This is also corrected. Outline was evaluated as Sink, but calls super methods, which is not allowed for the sink micro pattern.
- *Stateless:* The tool of Maman does not evaluate the Main class as stateless while it has no fields or super classes that contain fields.
- *State Machine:* Ok, after correction. Only interfaces can be a state machine.

- *Trait:* The Trait class and other abstract classes with at least one abstract method and no state are not evaluated as trait by the tool of Maman. These classes do conform to the description from the paper.

Assumptions made during the implementation of the micro pattern tool are (per micro pattern):

- *Pool:* Assumed is that constraints must also apply on super classes, only default constructors are allowed (<init>) and the class must have at least one field. Test suggest that the tool of Maman adds the requirement that fields must be public. I did not add this requirement, because it is not described in the paper.

- *Function Pointer:* Assumed is that constraints must apply to the super classes, because this seems in line with the intend of the pattern. The test results of the tool of Maman also suggest that super classes are evaluated. I assume that interfaces could also be function pointers.

- *Function Object:* Assumed is that constraints must apply to super classes.

- *Stateless:* Assumed is that interfaces can not be stateless. However, it could be useful to make a distinction between interfaces that have static fields or not. Super classes are included in the analysis.

- *Common State:* Super classes also evaluated. Although it is not stated in the paper, results from Mamans tool suggest that fields cannot be final for common state classes. I followed this assumption, because stateless classes could otherwise be common state (only static final fields). I assume that methods are not required, because this is not stated in the paper. However, this would be more inline with the intend of the pattern. Assumed is that the class itself (besides it parents) must have static fields. Maybe a new pattern could be introduced, for classes that only contain static final fields (For example, fixed Common State). The Stateless pattern could be narrowed to classes that contain no fields at all.

- *Immutable:* Assumed is that the "field is assigned once during instance construction" requirement applies to all the constructors of the class.

- *Canopy:* Assumed is that the "one instance field that can only be changed by the constructors of this class" applies to all constructors.

- *Record:* Assumed is that fields cannot be static, this is more in line with the intend of the record pattern and test results suggest that the tool of Maman does the same. Assumed is that super classes must also be evaluated. Furthermore is assumed that the class must have a field.

- *Data Manager:* The definition of getters and setters is not given in the paper. Therefore I define getters as: methods that do not return void, has no arguments, have a name that starts with "get" or "is" and contain the “GETFIELD” opcode. I define setters as: methods that return void, have one argument, have a name that starts with “set” and contain the “PUTFIELD” opcode.

- *Sink:* The definition from the table and description in the paper are not the same. The description from the table used. Interfaces are always Sink, so they are excluded.
- *State Machine:* Assumed is that parents do not have to be evaluated.
- *Pure Type:* Assumed is that a minimum of one abstract method is required.
- *Pseudo Class:* The paper states that static methods are permitted and that a pseudo classes can be mechanically rewritten as an interface. These statements seem incompatible. I choose not to allow static methods.
- *Extender:* Assumed is that a class must extend an other object than the Java Object class.

No observations are done or assumptions are made for micro patterns that are not listed above. The work of Gil and Maman contains the definitions of each micro pattern [6].

Appendix C

Formal Concept Algorithm

The algorithm used to create formal concepts during this research is displayed here. The Rascal meta-programming language is used to construct the algorithm.

```
//Creates formal concepts from formal context. Faster than fca if no lattice is needed.
//Will also return a single concept if found, instead of empty lattice as fca().
public set[Concept[&Object, &Attribute]] fca2(FormalContext[&Object, &Attribute] context) =
    closeFca(unclosedFca(context));

//Creates formal concepts, but unclosed concepts may occur, like: {<{1},{1,2}>, <{1,2},{1,2}>, <{2},{1,2}>}
public set[Concept[&Object, &Attribute]] unclosedFca(FormalContext[&Object, &Attribute] context) {

    map[set[&Object] objects, set[&Attribute] attributes] openConcepts = ();
    map[&Attribute, set[&Object]] invertedContextMap = toMap(invert(context));

    for(&Attribute attribute <- range(context)) {

        set[&Object] objects = invertedContextMap[attribute];
        set[set[&Object]] powObjects = power(objects);

        for(set[&Object] powObject <- powObjects) {

            try {
                openConcepts[powObject] = openConcepts[powObject] + attribute;
            }
            catch NoSuchKey : {
                openConcepts += (powObject : { attribute });
            }
        }
    }

    return toRel(openConcepts);
}

//Closes the set of concepts, for example -> {<{1},{1,2}>, <{1,2},{1,2}>, <{2},{1,2}>} -> {<{1,2},{1,2}>}
public set[Concept[&Object, &Attribute]] closeFca(set[Concept[&Object, &Attribute]] openConcepts) {

    set[Concept[&Object, &Attribute]] closedConcepts = {};
    set[&Object] occuringObjects = {};

    map[set[&Attribute], set[set[&Object]]] invertedContextMap = toMap(invert(openConcepts));

    for(set[&Attribute] attributeCombination <- range(openConcepts)) {
```

```
        set[set[&Object]] objectsForCombination = invertedContextMap[attributeCombination];
        set[&Object] greatestObjectSet = getGreatestSet(objectsForCombination);

        closedConcepts += <greatestObjectSet, attributeCombination>;
        occuringObjects += greatestObjectSet;
    }

    set[Concept[&Object, &Attribute]] filtered = domainR(openConcepts,  { occuringObjects });
    set[set[&Attribute]] attributesForCombination = range(filtered);

    closedConcepts += <occuringObjects, getGreatestSet(attributesForCombination)>;

    return closedConcepts;
}

//Gets the greatest set of a set of sub-sets
private set[&t] getGreatestSet(set[set[&t]] sets) {

    set[&t] superSet = {};

    for(set[&t] sett <- sets) {
        if(sett >= superSet) {
            superSet = sett;
        }
    }

    return superSet;
}
```

Appendix D

Manual Evaluation of Renaming Suggestions

In this appendix a small amount of renaming suggestions for AspectJ is examined to determine how useful these suggestions are. 10 suggestions are reviewed with a suffix that occurred commonly in the generated renaming suggestions (table D.1), and 10 for suffixes that where less common (D.2).

D.1 Common Suffixes

Class	Suggestions	Suitable	Motivation
`org/aspectj/ajde/internal/StructureUtilities`	NN-NN-NN-Adapter	No. Empty Class.	All the fields and methods are commented out.
	NN-NN-NN-Constants NN-NN-NN-NNS-Exception NN-NN-NN-JJ-Exception NN-NN-NN-NN-Exception		
`org/aspectj/apache/bcel/classfile/ConstantClass`	NN-NN-JJ-NN-Impl NN-NN-NN-NN-Impl NN-NN-JJ-JJ-Impl NN-NN-NN-JJ-Impl NN-NN-NN-NN-Impl NN-NN-NN-NN-Property NN-NN-JJ-NN-Property NN-NN-JJ-NN-Impl NN-NN-JJ-NN-Impl NN-NN-NN-NN-Impl NN-NN-JJ-JJ-Impl NN-NN-NN-JJ-Impl	Maybe, <noun>-Impl	Class overrides abstract methods from super class. However Imp is very general.
`org/aspectj/org/eclipse/jdt/core/dom/ASTVisitor`	NN-NN-NN-NNS-Exception	No	This is a visitor class for abstract syntax trees.

	NN-NN-NN-JJ-Exception NN-NN-NN-NN-Exception		
`org/aspectj/org/eclipse/jdt/core/BindingKey`	NN-NN-NN-NN-Impl NN-NN-NN-NN-Property NN-NN-JJ-NN-Property NN-NN-JJ-NN-Impl	No	This class is no implementation or property.
`org/aspectj/org/eclipse/jdt/internal/compiler/env/ClassSignature`	NN-NN-JJ-Class-Impl NN-NN-JJ-Class-Property NN-NN-NN-Class-Property NN-NN-NN-Class-Impl	No	Represents a class reference in the .class file. <noun>-class property makes some sense, but the current class name seems to be more suitable.
`org/aspectj/org/eclipse/jdt/internal/core/dom/rewrite/TokenScanner`	NN-NN-NN-NN-Impl	No	Does not implement any interfaces or abstract classes.
`org/aspectj/org/eclipse/jdt/internal/core/search/PathCollector`	NN-NN-NN-NN-Impl NN-NN-NN-NN-Property NN-NN-JJ-NN-Property NN-NN-JJ-NN-Impl	Maybe, <Noun>-Impl	Implements method of abstract super class.
`org/aspectj/org/eclipse/jdt/internal/core/util/SimpleDocument`	NN-NN-NN-NN-Impl NN-NN-NN-NN-Property NN-NN-JJ-NN-Property NN-NN-JJ-NN-Impl	Maybe, <noun>-Impl	Implements an interface.
`org/aspectj/weaver/bcel/asm/StackMapAdder`	NN-NN-Factory NN-NN-Utils	No No factory.	May be a utility, but this is too general.
`org/aspectj/weaver/tools/PointcutPrimitive`	NN-NN-NN-NN-NN-Manager NN-NN-NN-NN-NN-Factory NN-NN-NN-NN-NN-Type NN-NN-NN-NNS-Exception NN-NN-NN-JJ-Exception NN-NN-NN-NN-Exception	No	A class containing constants for “point cut primitives”

Table D.1: Manual evaluation of the renaming suggestions for suggestions with a common suffix.

The results from this small sample experiment suggest that mainly renaming suggestions with the "Impl" suffix make any sense. However, this suffix can be used very generally, since it makes sense as suffix for any class implementing an interface or abstract class. Furthermore, wether the use of Impl as suffix of classes is good style is debatable. So, the table suggests that there are hardly any usable renaming suggestions to find in the suggestions with a commonly occurring suffix.

D.2 Uncommon Suffixes

Class	Suggestions	Suitable	Motivation
`org/aspectj/org/eclipse/jdt/core/util/OpcodeStringValues`	NN-NN-NN-String-Factory NN-NN-NN-String-Utils NN-NN-Adapter NN-NN-Constants NN-NN-Exception NN-NNS-Exception NN-JJ-Exception	Maybe, NN-NN-Constants or NN-NN-NN-String-Utils	Contains an array with the description of each Java opcode mnemonics. OpcodeStringConstants or OpcodeStringUtilities could be suitable.
`org/aspectj/weaver/tools/FuzzyBoolean`	NN-NN-NN-NN-NN-Type	Maybe, Fyzzy-Boolean-Type	Fuzzy Boolean class. A refined boolean type.
`org/aspectj/weaver/ast/Test`	NN-JJ-JJ-Provider NN-NN-JJ-Provider NN-NN-NN-Provider NN-JJ-NN-Provider	No	This class is a kind of abstract syntax tree node.
`org/aspectj/weaver/Position`	NN-NN-NN-Info	Maybe, WeaverPositionInfo	Contains fields for the start and end position of the weaver.
`org/aspectj/apache/bcel/classfile/Modifiers`	NN-NN-NN-Data	No	Super class for objects that have access modifiers.
`org/aspectj/weaver/patterns/WithinPointcut`	NN-NN-List	No	No list properties.
`org/aspectj/org/eclipse/jdt/internal/core/search/matching/VariablePattern`	NN-NN-NN-Variable-View NN-JJ-JJ-Variable-View NN-NN-NNS-Variable-View NN-JJ-NN-Variable-View NN-NN-JJ-Variable-View NN-JJ-NNS-Variable-View	No	No view.
`org/aspectj/org/eclipse/jdt/core/ClasspathVariableInitializer`	NN-Abstract-Variable-Provider NN-NN-Variable-Provider	No	Provides nothing. Only void methods.
`org/aspectj/weaver/bcel/asm/AsmDetector`	NN-NN-NN-Factory-IR-Format NN-NN-NN-Policy-IR-Format	No	Determines if a version of ASM is present.

	NN-NN-JJ-Factory-IR-Format NN-NN-JJ-Policy-IR-Format		
`org/aspectj/weaver/ast/Literal`	NN-NN-Type-Manager NN-NN-NN-NN-NN-Manager	?	Hard to determine intend of the class, but probably no.

Table D.2: Manual evaluation of the renaming suggestions for suggestions with a uncommon suffix.

This table shows three renaming suggestions with a suffix that is less common, namely: "Constants", "Type" and "Info". Whilst still not ideal, the results suggest that renaming suggestions with less common suffixes tend to be more specific and better suitable than those with common suffixes from the previous table.

Appendix E

Purpose-Specific Pattern Observations

This appendix contains the observations made during the manual examination of a set of sample classes with the Exception and Factory suffix. The observations for the classes with the Exception suffix are shown in table E.1, and for classes with the Factory suffix in table E.2.

Application	- Class	Description
Eclipse SDK	`org/eclipse/equinox/security/storage/StorageException`	- Extends Exception. - Contains two constructors calling the super constructor. - Contains error code constants. - Contains private error code field. - One getter for the error code. - Contains serialVersionUID.
Eclipse SDK	`org/eclipse/core/commands/NotHandledException`	- Super class extends exception. - Contains a constructor calling the super constructor. - Contains serialVersionUID.
Jrat	`org/shiftone/jrat/core/ParseException`	- Super class extends exception. - Contains two constructors calling the super constructor. - Contains constant for logging.
Azureus	`org/gudy/azureus2/pluginsimpl/remote/rpexceptions/RPObjectNoLongerExistsException`	- Super class extends exception. - Contains a constructor calling the super constructor. - Contains a getter method (not strictly a getter).
Spring Framework	`org/springframework/web/client/HttpStatusCodeException`	- Super class extends exception. - Contains multiple constructors, of which one calls the super constructor. - Contains getter methods. - Contains constant.
Jena	`com/hp/hpl/jena/shared/UnknownPropertyException`	- Super class extends exception. - Contains a constructor calling the super constructor.
Jre	`java/awt/color/CMMException`	- Super class extends exception.

		- Contains a constructor calling the super constructor.
Roller	`org/apache/roller/weblogger/business/themes/ThemeParsingException`	- Super class extends exception. - Contains constructors calling the super constructor.
Gt2	`org/geotools/filter/MalformedFilterException`	- Extends exception. - Contains constructors calling the super constructor.

Table E.1: Observations made for 10 random classes with the Exception suffix.

Application	Class	Description
fitlibrary for fitnesse	`fitlibrary/table/TableFactory`	- No super classes or interfaces. - Multiple static fields. - Multiple overloaded methods with the same return type, creating objects. - Not all methods create new objects. - Multiple factory methods
Jedit	`org/gjt/sp/jedit/gui/statusbar/MultiSelectWidgetFactory`	- Implements interface - No fields. - Single method creating an object. - Inner class. - Single factory method.
Gt2	`org/geotools/geometry/iso/operation/overlay/OverlayNodeFactory`	- Implements interface - No fields. - Single method creating an object. - No factory method. Return type has parent, but not abstract.
Openjms	`org/exolab/jms/net/connector/AbstractConnectionFactory`	- Abstract class implementing an interface. - Multiple methods, of which none directly creates objects. - Multiple fields. - No factory method. Abstract factory.
Castor	`org/exolab/castor/xml/XercesXMLSerializerFactory`	- Implements an interface. - No fields. - Multiple methods directly creating objects. - Multiple factory methods.
Tapestry	`org/apache/tapestry5/internal/bindings/ContextBindingFactory`	- Implements an interface. - No fields. - Single method creating an object. - Single factory method.
Eclipse SDK	`org/eclipse/pde/internal/core/product/ProductModelFactory`	- Implements an interface. - Contains one field. - Multiple methods directly creating new objects. - Multiple factory methods.
Openjms	`org/exolab/jms/selector/RegexpFactory`	- No super classes or interfaces. - No fields. - A single method (long) creating a new - object or throwing an exception. - Factory? N.a.
Megamek	`megamek/common/net/marshall/PacketMarshallerFactory`	- Singleton class. - No super classes or interfaces. - Get instance method and a method creating an object depending on a variable. - Single factory method.
Findbugs	`edu/umd/cs/findbugs/sourceViewer/NumberedViewFactory`	- Implements an interface.

		- Has a single field. - Has multiple methods, of which one creates objects depending of the type of the argument. - Single factory method.
Jena	`com/hp/hpl/jena/query/ResultSetFactory`	- Interface containing multiple methods. - Factory? N.a.
JreFactory	`org/acm/seguin/refactor/method/` `MethodRefactoringFactory`	- No super classes or interfaces. - Has no fields. - Has multiple methods, that create directly create new objects. - No factory method. Return type has parent, but not abstract.
Xalan	`org/apache/xpath/objects/XObjectFactory`	- No super classes or interfaces. - Has no fields. - Has two methods that create new objects depending on the argument type. - No factory method. Return type has parent, but not abstract.
Castor	`org/exolab/castor/xml/handlers/` `DefaultFieldHandlerFactory`	- Extends super class - Has one field. - Has multiple methods, of which one returns an new object depending on the argument type. - Single factory method.
Openjms	`org/exolab/jms/selector/` `DefaultExpressionFactory`	- Implements an interface. - Has no fields. - Has multiple methods creating an objects (of which the type depends on a parameter). - Multiple factory methods.

Table E.2: Observations made for 15 random classes with the Factory suffix.