Design Pattern Smell Input Files Specification

Bruno. L. Sousa[1], **Mariza A. S. Bigonha**[1], **Kecia A. M. Ferreira**[2]

[1]Computer Science Department – Federal University of Minas Gerais (UFMG)
Belo Horizonte – MG – Brazil

[2]Department of Computing – Federal Center for Technological Education of
Minas Gerais (CEFET-MG) – Belo Horizonte – MG – Brazil

{bruno.luan.sousa,mariza}@dcc.ufmg.br, kecia@decom.cefetmg.br

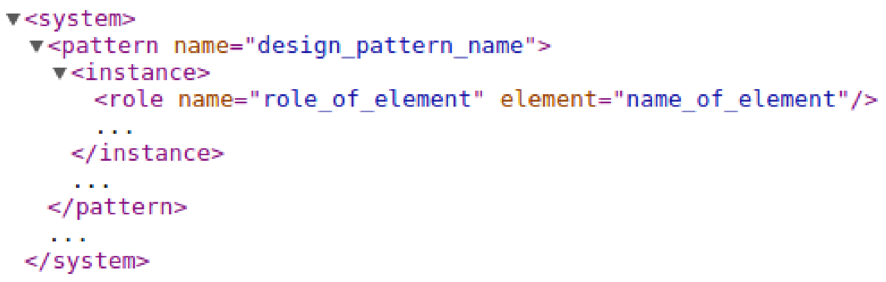
1. Introduction

This document presents a specification of the XML and CSV files provided as input to the Design Pattern Smell running. In addition to presenting a template for describing these files, this document describes the function of each tag and attribute used to create the XML file and each row and column used for create of CSV file.

The remainder of this document is organized as follows. Section 2 describes the XML file used to represent of design pattern instances of a system. Section 3 describes the CSV file used to indicate the classes or methods of a system that have the presence of bad smell. Section 4 concludes this document.

2. XML Input Pattern

This section describes the XML input file format that should be provided for Desig Pattern Smell. This file should contain information about design pattern instances, collected from a particular software system. Figure 1 shows the template for creating the XML file, as well as the main tags and attributes that should be used to describe the data.



```
▼<system>
  ▼<pattern name="design_pattern_name">
    ▼<instance>
       <role name="role_of_element" element="name_of_element"/>
       ...
     </instance>
     ...
   </pattern>
   ...
 </system>
```

Figure 1. Template for creating the input XML for Design Pattern Smell.

The format of this file was inspired by the output pattern used by a collection tool of design pattern instances, called *Design Pattern Detection*[1] [Tsantalis et al. 2006]. The tags and attributes used in this document are described below.

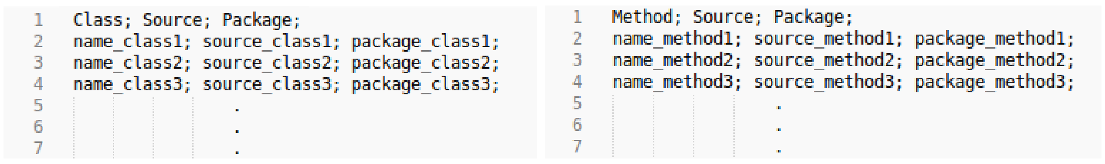
- **system:** this tag should be used in the first line of the document. It informs to Design Pattern Smell that the information within its scope refers to the data on design patterns from a particular software.

[1]https://users.encs.concordia.ca/~nikolaos/pattern_detection.html

- **pattern:** indicates that informations about instances of certain patterns (Factory Method, Observer, Template Method, among other) will be informed within its scope. The pattern tag should be accompanied by the following attributes:
  - **name:** specifies to the Design Pattern Smell the name of design pattern, whose instance informations is referenced.
- **instance:** this tag is used within the scope of the tag pattern. It indicates information about the instances identifies for each design pattern, displaying the classes, methods and attributes present in its composition. In addition to displaying information about the components of a particular instance, it indicates the role that each component exerts on within the specific instance.
- **role:** indicats the role played by a component (class, method or attribute) within a design pattern instance. This tag has the following attributes: name and element.
  - **name:** this attribute indicates the role name of a component of its instance of the pattern. The possible roles that may be used as value for this attribute are indicated in the Table 1.
  - **element:** indicates the name of the component (classe, method or attribute) that makes up a particular instance of a design pattern and performs the role defined by the name tag within that instance.

3. CSV Input Pattern

This section describes the format of the CSV input file that must be provided for Design Pattern Smell. This file should contain information about artifacts (classes or methods) with the presence of bad smell collected of a particar software system. Figure 2 shows the template for creating of CSV file.



```
Class; Source; Package;
name_class1; source_class1; package_class1;
name_class2; source_class2; package_class2;
name_class3; source_class3; package_class3;
                    .
                    .
                    .
```

```
Method; Source; Package;
name_method1; source_method1; package_method1;
name_method2; source_method2; package_method2;
name_method3; source_method3; package_method3;
                    .
                    .
                    .
```

Figure 2. (i) Template for creating the CSV file for classes with bad smell; (ii) Template for creating CSV file for methos with bad semll.

The first line, shown in Figure 2, is the default for any file with bad smell information for use in Design Pattern Smell. It defines the header of this file, and tells the tool the type of content displayed in each of the columns of that file.

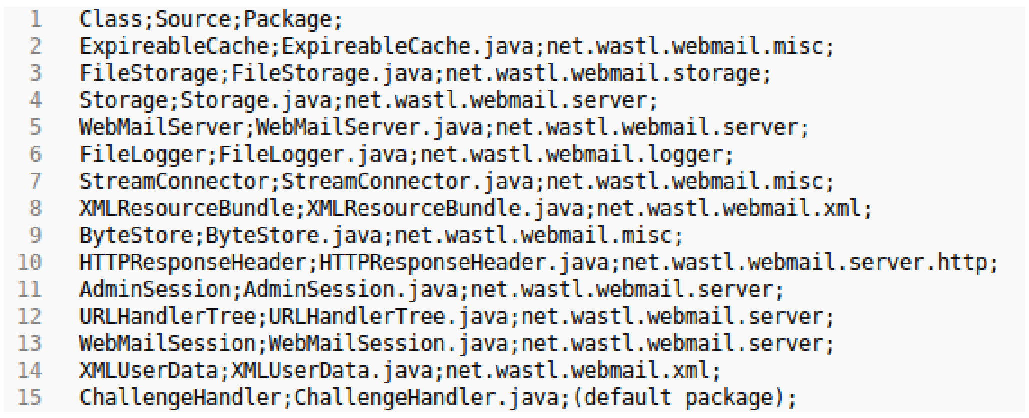
The first column of the header is intended to specify the name of the artifact that has the presence of a bad smell. As it is possible to observe in Figure 2, the name of this column varies according to the granularity of the bad smell chosen for analysis.

The second column of the header indicates the Java format file in which the artifact is stored. This column is default for both classes or methods. The third column indicates the package that the specified artifact is contained. Finally, the other lines of this file indicate the artifacts with bad smell.

Figure 3 shows an example CSV file with classes that have the presence of Data Class bad smell, built for the Webmail system release 0.7.10.

Design Pattern	Role	Type of Component
Adapter-Command	Adapter/ConcreteCommand	Class
	Adaptee/Receiver	Class
	adaptee/receiver	Attribute
	Request()/Execute()	Method
Bridge	Implementor	Class
	Abstraction	Class
	implementor	Attribute
	Operation()	Method
Composite	Component	Class
	Composite	Class
	Operation()	Method
Decorator	Component	Class
	Decorator	Class
	component	Attribute
	Operation()	Method
Factory Method	Creator	Class
	FactoryMethod()	Method
Observer	Observer	Class
	Subject	Class
	Notify()	Method
Prototype	Client	Class
	Prototype	Class
	prototype	Attribute
	Operation()	Method
Proxy	Proxy	Class
	RealSubject	Class
	Request()	Method
State-Strategy	Context	Class
	State/Strategy	Class
	state/strategy	Attribute
	Request()	Method
Singleton	Singleton	Class
	uniqueInstance	Attribute
Template Method	AbstractClass	Class
	TemplateMethod()	Method
Visitor	ConcreteElement	Class
	Visitor	Class
	Accept()	Method

Table 1. Design patterns supported by Design Pattern Smell and the possible roles exercised within each design pattern.



```
Class;Source;Package;
ExpireableCache;ExpireableCache.java;net.wastl.webmail.misc;
FileStorage;FileStorage.java;net.wastl.webmail.storage;
Storage;Storage.java;net.wastl.webmail.server;
WebMailServer;WebMailServer.java;net.wastl.webmail.server;
FileLogger;FileLogger.java;net.wastl.webmail.logger;
StreamConnector;StreamConnector.java;net.wastl.webmail.misc;
XMLResourceBundle;XMLResourceBundle.java;net.wastl.webmail.xml;
ByteStore;ByteStore.java;net.wastl.webmail.misc;
HTTPResponseHeader;HTTPResponseHeader.java;net.wastl.webmail.server.http;
AdminSession;AdminSession.java;net.wastl.webmail.server;
URLHandlerTree;URLHandlerTree.java;net.wastl.webmail.server;
WebMailSession;WebMailSession.java;net.wastl.webmail.server;
XMLUserData;XMLUserData.java;net.wastl.webmail.xml;
ChallengeHandler;ChallengeHandler.java;(default package);
```

Figure 3. CSV file with classes that have the Data Class bad smell, built for the Webmail 0.7.10 system.

4. Conclusion

This document presented a description about the default XML and CSV input files, which should be provided to Design Pattern Smell, as a requirement for their operation. It is expected that the user will use this document as a guide for the construction of these files, and will be able to take full advantage of Design Pattern Smell's features in reserach and analysis of co-occurrence between design patterns and bad smell in software systems.

References

[Fowler and Beck 1999] Fowler, M. and Beck, K. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley.

[Tsantalis et al. 2006] Tsantalis, N., Chatzigeorgiou, A., Stephanides, G., and Halkidis, S. T. (2006). Design pattern detection using similarity scoring. *Software Engineering, IEEE Transactions on*, 32(11):896–909.