A benchmark platform for design pattern detection

Francesca Arcelli, Marco Zanoni, Andrea Caracciolo
Dipartimento di Informatica, Sistemistica e Comunicazione
University of Milano Bicocca
Milano, Italy
Email: {*arcelli,marco.zanoni*}*@disco.unimib.it, a.caracciolo1@campus.unimib.it*

***Abstract*—Design patterns detection is a useful activity in reverse engineering to gain knowledge on the design issues of an existing system, on its software architecture and design quality, improving in this way the comprehension of the system and hence its maintainability and evolution. Several tools have been developed, but they usually provide different results analyzing the same systems. Some works have been proposed in the literature to compare these results, but a standard widely accepted benchmark is not yet available. In this work we propose our benchmark platform for design patterns detection, based on a community driven evaluation.**

***Keywords*-design pattern detection; benchmark**

I. Introduction

Design pattern detection (DPD) is a topic which received a great interest during the last years. Finding design patterns (DPs) [1] in a software system can give very useful hints on the comprehension of a software system and on what kind of problems have been addressed during the development of the system itself; their presence can be considered as an indicator of good software design. Moreover, they are very important during the re-documentation process, in particular when the documentation is very poor, incomplete or not up-to-date.

Several DPD approaches and tools have been developed both for forward and reverse engineering aims and involving different techniques for the detection such as fuzzy logic, constraints solving techniques, theorem provers, template matching methods and classification techniques (i.e. [2], [3], [4], [5], [6], [7]). In spite of the many approaches proposed, the results obtained are often quite unsatisfactory and different from one tool to the other.

Many tools find many false positive instances but other correct instances are not found. One common problem in DPD is the so called variant problem: DPs can be implemented in several ways, often very different from one another. The main variants for each pattern are described in the catalog of [1], others are applied when the context of application requires it. These variations cause the failure of most pattern instances recognition using rigid detection approaches, which are based only on canonical pattern instances.

Hence the comparison of the results provided by the different tools is very important, in order to be able to evaluate the best approach and technique. Some works that have been proposed respect to the comparison of DPD tools are described in the next section.

We analyzed and faced the problem related to the different results provided by the DPD tools, since we are developing a tool called MARPLE (Metrics and Architecture Reconstruction plug-in for Eclipse) [8] whose main aims are related to DPD and software architecture reconstruction. The Marple DPD module is characterized by the following steps:

- the detection of sub components or micro structures which give useful hints on the DP detection, with the aim of mitigating the variant problem;
- the detection of the largest possible set of DP candidates performed by a module called Joiner, whose results are characterized by very high recall values;
- the refining of the previous results through data mining techniques, in particular through a step of clustering and a step of supervised classification.

The aim of this work is to introduce and describe a benchmark platform to be used to compare DPD tools. We propose some mechanisms to obtain safer results and to make them available to the DPD community in an easy way. Our approach is characterized by:

- a general model for design pattern representation;
- a way of comparing results coming from different tools;
- the possibility to evaluate instances and discuss about their correctness.

The adoption by the DPD community of a benchmark can improve the cooperation among the researchers and the reuse of tools written by other instead of the development of new ones.

II. Related Work

As we observed before, undertaken a comparison among design pattern detection tools is certainly a difficult task. Few benchmark proposals for the evaluation of design pattern detection tools have been presented in the literature. In [9] we describe our first proposal for a benchmark, that we extend and finalize in this paper. In [10] the authors present their work in progress towards creating a benchmark, called DEEBEE (DEsign pattern Evaluation Benchmark Environment), for evaluating and comparing design pattern detection tools. Currently, the benchmark database contains the results of three tools: Columbus (C++), Maisa (C++), and Design Pattern detection Tool (Java). The tools were evaluated on reference implementations of patterns and on open source software (Mozilla, NotePad++, JHotDraw, JRefactory and JUnit).

In [11] a general framework for the comparison of design recovery tools (hence not only for the comparison of design pattern detection tools) is proposed, illustrating how the framework can be applied for the comparison of two different projects, Licor and Ptidej. Moreover in [12] it has been defined P-MARt, a repository of pattern-like micro-architectures, with the purpose to serve as baseline to assess the precision and recall of pattern identification tools. The repository contains the analysis of specific version of different projects as for example QuickUML, Lexi, JRefactory, JUnit and other projects. While in [13] the authors compare different design patterns detection tools and they propose a novel approach based on data fusion, build on the synergy of proven techniques, without requiring any re-implementation of what is already available.

III. DP REPRESENTATION

In order to work on design pattern instances we need a way to represent them in some kind of data structure. In [9] we presented a model for the representation of DP definitions and instances. That model now stands behind all the elaboration made by the benchmark platform.

The model specifies that a design pattern can be defined from the structural point of view using the roles it contains and the cardinality relationship between couple of roles. A design pattern is defined as a tree whose nodes are called *Levels*; each Level has to contain at least one of the roles of the pattern and it can contain other nested Levels, recursively. In Figure 1 it is possible to see the tree structure of the *LevelDef* class (representing the level definition), and the *RoleDef*s it owns; finally, *DpDef* defines that a design pattern definition is a tree having as root one *LevelDef*.

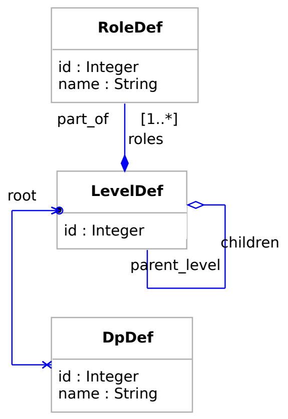


Figure 1. DP Definition UML class diagram

When two roles are contained in the same level, they are in a one-to-one relationship; instead when a role is in a nested Level it means that for each instance of the roles in the parent level, there can be many sets of roles of the child level. The most common case is when a pattern defines that a class must extend another class. In most cases we identify a single instance of that pattern as the parent class connected with all the children classes. Instances are modeled as in Figure 2; the model is simply an extension of the definition, as it models the instantiation of the concepts contained in the definition: a *RoleAssociation* is the realization of a *RoleDef*, a *LevelInstance* is the realization of a *LevelDef*, and so on. The only complex detail is the splitting of *Level* and *LevelInstance*; the explanation is that each *LevelDef* is instantiated as a *LevelInstance* when the *RoleAssociation*s are filled, but to define a child *Level* we need to specify which particular parent instance it belongs to.

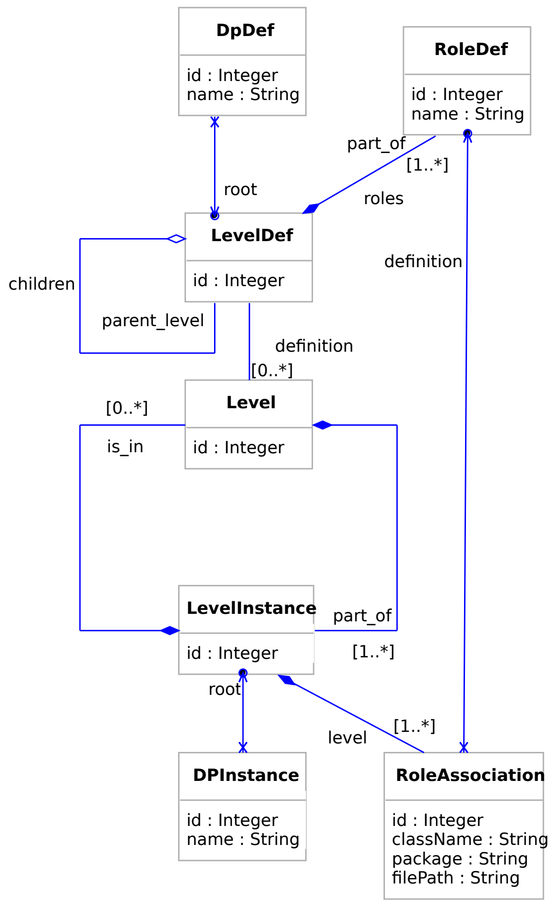


Figure 2. Model UML class diagram

A. XML format

The XML format for specifying design pattern instance is modeled according to the representation model, so it follows the same concepts. The XML Schema Definition, which the submitted XML file must comply to, is available at http://essere.disco.unimib.it:8080/DPBWeb/faces/resources/DpAnalysis.xsd; moreover, the list of the pattern definitions the tool can support is also available at page http://essere.disco.unimib.it:8080/DPBWeb/faces/Doc_DpDef.jsp.

In order to have more chances of being able to compare results coming from different tools, currently users cannot supply their own pattern definitions. When the platform will be more tested and filled with data, we will add this functionality. Meanwhile we accept and encourage suggestions coming from the users about new definitions or mistakes in the current ones.

In the following we report an XML file example that represents an instance of an Abstract Factory design pattern.

Listing 1. Example of Abstract Factory instance

```
<analysis
xmlns="http://www.essere.disco.unimib.it/DPBWeb"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance"
xsi:schemaLocation="http://www.essere.disco.unimib
    .it:8080/DPBWeb/resources/DpAnalysis.xsd">
<pattern name="Abstract_Factory">
<patternInstance>
  <role name="Abstract_Factory"
     package="org.foo.bar"
     class="AbstractFactoryClassName"
     filePath="org/foo/bar/a1.java" />
  <level>
    <levelInstance>
      <role name="Concrete_Factory"
         package="org.foo.bar"
         class="ConcreteFactoryClassName"
         filePath="org/foo/bar/b1.java" />
    </levelInstance>
  </level>
  <level>
    <levelInstance>
      <role name="Abstract_Product"
         package="org.foo.bar.baz"
         class="AbstractProductClassName"
         filePath="org/foo/bar/baz/c1.java" />
      <level>
        <levelInstance>
          <role name="Concrete_Product"
             package="org.foo.bar.baz"
             class="ConcreteProductClassName1"
             filePath="org/foo/bar/baz/d1.java" />
        </levelInstance>
        <levelInstance>
          <role name="Concrete_Product"
             package="org.foo.bar.baz"
             class="ConcreteProductClassName2"
             filePath="org/foo/bar/baz/d2.java" />
        </levelInstance>
      </level>
    </levelInstance>
  </level>
  <level>
    <levelInstance>
      <role name="Client"
         package="org/foo/bar/baz"
         class="ClientClassName"
         filePath="org/foo/bar/baz/e1.java" />
    </levelInstance>
  </level>
</patternInstance>
</pattern>
</analysis>
```

The file refers to the roles *Abstract Factory*, *Concrete Factory*, *Abstract Product*, *Concrete Product*, each associated to a class name and to a package name, and organized following the Abstract Factory definition.

IV. DPD PLATFORM

The platform is available at the url http://essere.disco.unimib.it/DPB and it is subdivided in:

- the *documentation* section contains some references and guides to use the platform; in addiction the home page briefly introduces the system functionalities and provides a step by step tutorial;
- the *search* section lets the user to find the results of a particular analysis, according to different parameters;
- the *compare* section allows to compare the instances found by different tools on the same input project;
- the *browse* section provides a tree-like view of the contents of the platform.

Through these different kinds of exploration the user can obtain the detailed view of each pattern instance, that includes two different types of graphic visualization and a simple forum for the *evaluation* of the instance by the users.

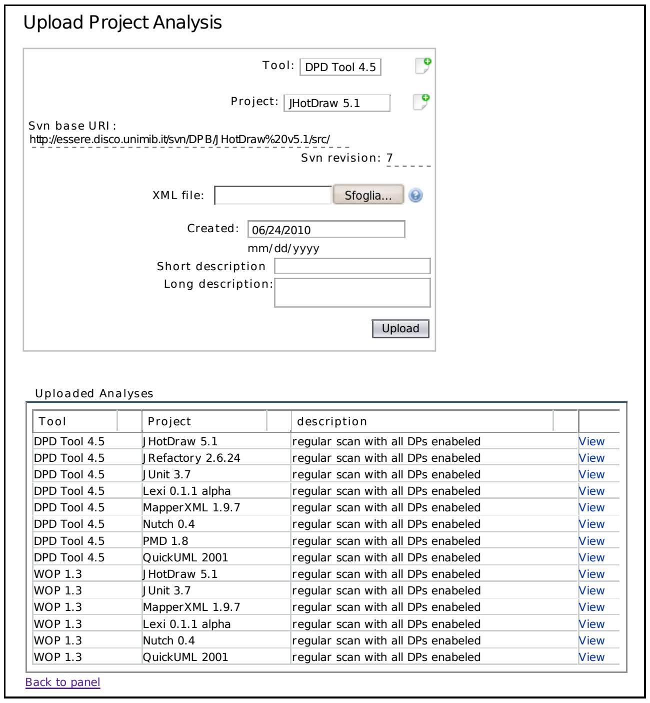
All the above functionalities are visible to all users; if a user wants to load new pattern instances into the platform he must be registered, log into the platform, and submit the XML file containing the instances and some metadata, as shown in Figure 3. In our platform an *analysis* consists of the combination of the set of the instances, its description, the choice of the DPD tool and the analyzed project.

A. Search

The user can search a pattern instance according to the chosen programming language, project, detection tool, and design pattern. An example of a search section is shown in Figure 4. In the user interface, whenever the user chooses a filter, the page shows the available further filters. In each selection list it is possible to choose more than one value, to make the search more flexible and personalized.

Figure 4 shows an example of search parameters and results, in the case the user looks for the analysis of project written in the Java language, on the “JHotDraw 5.1” and “MapperXML 1.9.7” projects, analyzed by the “DPDTool 4.5” and “WOP 1.3” tools and filtered on the “Composite” and “Template Method” design patterns.

The table (see Figure 4) shows the results with the tools and their analysis on the rows and the projects with the patterns on the columns. Each cell is the combination of an analysis of a tool and a pattern belonging to a project, and it contains the number of instances considered correct and the number of the ones considered incorrect, respectively under the column label “T” and “F”.



Upload Project Analysis

Tool: DPD Tool 4.5

Project: JHotDraw 5.1

Svn base URI : http://essere.disco.unimib.it/svn/DPB/JHotDraw%20v5.1/src/

Svn revision: 7

XML file:

Created: 06/24/2010 mm/dd/yyyy

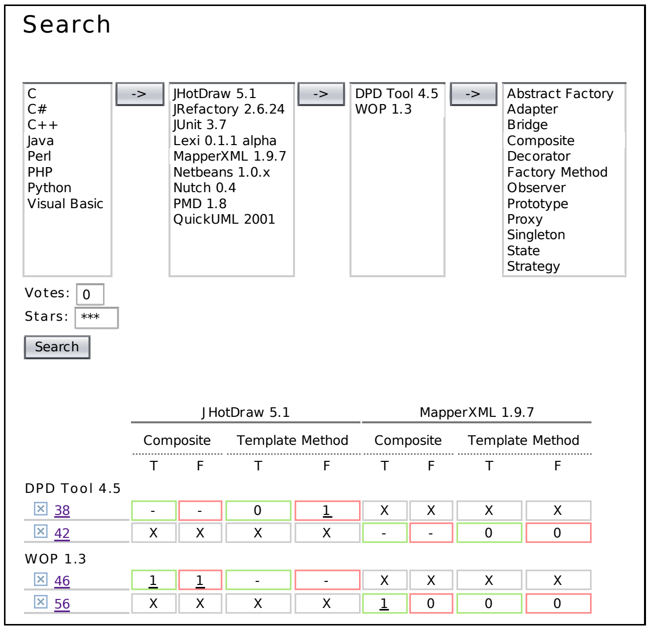
Short description

Long description:

Uploaded Analyses

Tool	Project	description	
DPD Tool 4.5	JHotDraw 5.1	regular scan with all DPs enabeled	View
DPD Tool 4.5	JRefactory 2.6.24	regular scan with all DPs enabeled	View
DPD Tool 4.5	JUnit 3.7	regular scan with all DPs enabeled	View
DPD Tool 4.5	Lexi 0.1.1 alpha	regular scan with all DPs enabeled	View
DPD Tool 4.5	MapperXML 1.9.7	regular scan with all DPs enabeled	View
DPD Tool 4.5	Nutch 0.4	regular scan with all DPs enabeled	View
DPD Tool 4.5	PMD 1.8	regular scan with all DPs enabeled	View
DPD Tool 4.5	QuickUML 2001	regular scan with all DPs enabeled	View
WOP 1.3	JHotDraw 5.1	regular scan with all DPs enabeled	View
WOP 1.3	JUnit 3.7	regular scan with all DPs enabeled	View
WOP 1.3	MapperXML 1.9.7	regular scan with all DPs enabeled	View
WOP 1.3	Lexi 0.1.1 alpha	regular scan with all DPs enabeled	View
WOP 1.3	Nutch 0.4	regular scan with all DPs enabeled	View
WOP 1.3	QuickUML 2001	regular scan with all DPs enabeled	View

Figure 3. Example of analysis loading



Search

C	JHotDraw 5.1	DPD Tool 4.5	Abstract Factory
C#	JRefactory 2.6.24	WOP 1.3	Adapter
C++	JUnit 3.7		Bridge
Java	Lexi 0.1.1 alpha		Composite
Perl	MapperXML 1.9.7		Decorator
PHP	Netbeans 1.0.x		Factory Method
Python	Nutch 0.4		Observer
Visual Basic	PMD 1.8		Prototype
	QuickUML 2001		Proxy
			Singleton
			State
			Strategy

Votes: 0

Stars: ***

	JHotDraw 5.1				MapperXML 1.9.7			
	Composite		Template Method		Composite		Template Method	
	T	F	T	F	T	F	T	F
DPD Tool 4.5								
38	-	-	0	1	X	X	X	X
42	X	X	X	X	-	-	0	0
WOP 1.3								
46	1	1	-	-	X	X	X	X
56	X	X	X	X	1	0	0	0

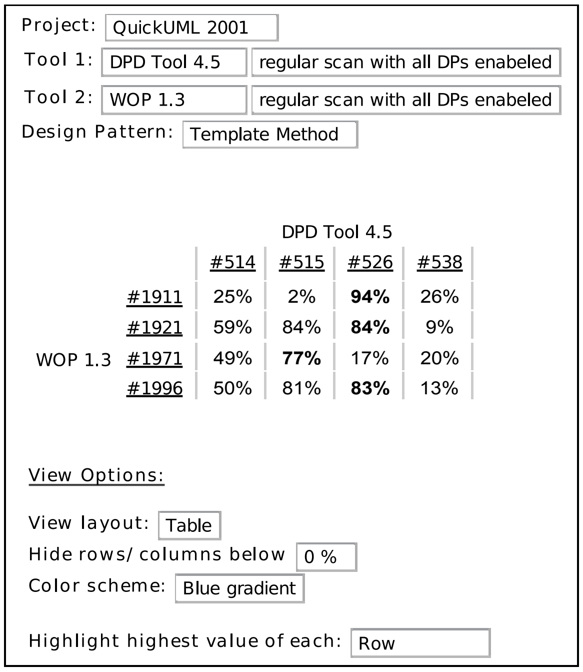
Figure 4. Example of a pattern instance search result

The bias value that allows the platform to choose if an instance is correct or not is specified by the two combo boxes named “Votes” and “Stars”. The details of this topic are explained in Section IV-D.

There is a special case regarding the cell values: for example, when an analysis does not contain instances of a particular design pattern in a particular project, the corresponding cells are rendered as “X” in light grey.

B. Comparison

The user can compare the results produced by two different analysis, obtained by two different tools, on the same project and for the same single chosen pattern definition. The comparison results (see Figure 5) are shown in a table where the two instance sets, found by the two analysis, are shown one on the rows and the other on the columns. The cells of the table contain values indicating the similarity of the corresponding couple of instances. The similarity is currently evaluated through a very simple algorithm described below. The background color of each cell is proportional to its percentage value, according to the selected color scheme (red:0% to green:100% or white:0% to blue:100%).



Project: QuickUML 2001

Tool 1: DPD Tool 4.5 regular scan with all DPs enabeled

Tool 2: WOP 1.3 regular scan with all DPs enabeled

Design Pattern: Template Method

		DPD Tool 4.5			
		#514	#515	#526	#538
WOP 1.3	#1911	25%	2%	**94%**	26%
	#1921	59%	84%	**84%**	9%
	#1971	49%	**77%**	17%	20%
	#1996	50%	81%	**83%**	13%

View Options:

View layout: Table

Hide rows/columns below 0 %

Color scheme: Blue gradient

Highlight highest value of each: Row

Figure 5. Example of a result comparison

Figure 5 shows an example of comparison: the user selects to compare the instances of the Composite pattern found on project “QuickUML 2001” by:

- the “DPD Tool 4.5” tool in the analysis named “regular scan with all DPs enabled”;
- the “WOP 1.3” tool in the analysis named “regular scan with all DPs enabled”.

The table shows that each tool found four instances, and that for example the instances #1911 and #526 are very similar, with a score of 94%, and instances #1911 and #515 are very different, having a score of only 2%. The numbers in bold are the highest value of the rows: in fact the last option selected is to highlight the highest value of each row; it is also possible to do the same on the columns. This option simplifies the task of finding the instances that are more similar in order to understand if they are really the same.

The view can be also filtered removing all the rows or columns having all the values less or equal to the one specified, in order to simplify the table and to include only meaningful results.

1) Comparison algorithm: In the current version of the benchmark platform we implemented a comparison algorithm we developed as a proof of concept. The algorithm tries to express the similarity of two instances giving more weight to the parent roles and less to the children roles, and produces a number between 0 (total difference) and 1 (total equivalence). The actual version of the algorithm is very simple and any suggestions and contribution coming from the DPD community, in order to populate the platform with other comparison algorithms, is welcome.

Our algorithm gives a descending score to each level depth in the pattern definition: for example in a definition with only a parent level and a child level, the parent level depth takes a score of 2 and the child level depth takes a score of 1; with three level depths the first one (the root) would take weight 3, and so on. Then each level in the definition takes the score of the depth it belongs to, and an overall score is calculated as the sum of the score of all levels.

Finally a weight is assigned to each level dividing the level score by the overall score. In this way the sum of all the weights is 1.

When the weights are set, the algorithm compares a couple of instances starting from the root and recursively distributing the weight of each level on its level instances; we consider that two level instances are equal if their sets of role association are completely equal.

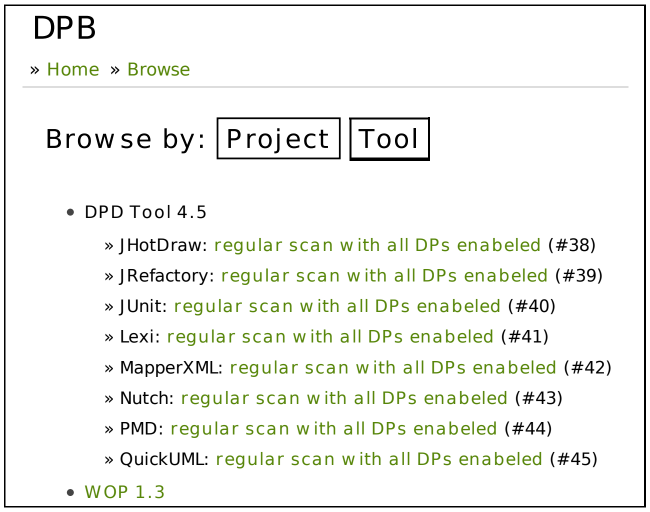
For example: let us say a level weights $1/2$ and the two levels to compare contain the first one instance and the second two instances; in addition the instance in the first level is equal to one of the two in the second one level. Their direct comparison value is $1/2$, because we have only one match out of two, that is the maximum of the number of the instances in each level. Then the overall score of the comparison is $1/2 * 1/2$, because we have to weight the local score with the global weight.

The overall score of the comparison is the sum of all the local scores.

C. Browse

The Browse section offers a tree view of the content of the platform. Basically it allows the user to find all the analysis made by a tool on a project: the user can choose as entry point both the available projects or tools. Whenever the users clicks on one of them, the page shows the analysis available in combination with the other.

In Figure 6 it is shown that for the "DPD Tool 4.5" tool different analysis are available on projects like "JHotDraw" and "JRefactory". Clicking on the name of the analysis (in


DPB

» Home » Browse

Browse by: Project Tool

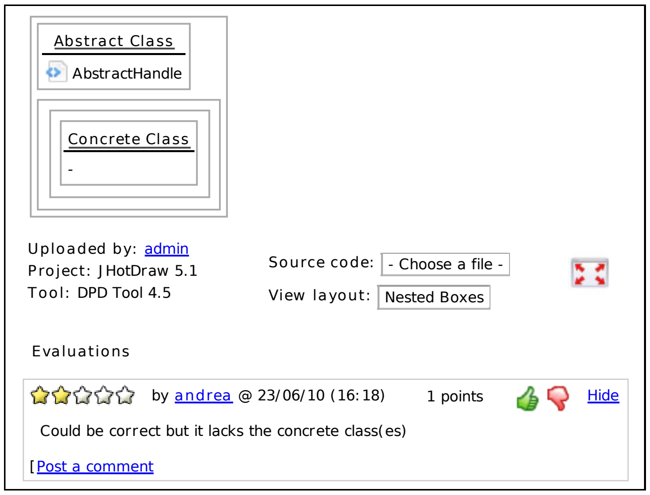
- DPD Tool 4.5
  - » JHotDraw: regular scan with all DPs enabeled (#38)
  - » JRefactory: regular scan with all DPs enabeled (#39)
  - » JUnit: regular scan with all DPs enabeled (#40)
  - » Lexi: regular scan with all DPs enabeled (#41)
  - » MapperXML: regular scan with all DPs enabeled (#42)
  - » Nutch: regular scan with all DPs enabeled (#43)
  - » PMD: regular scan with all DPs enabeled (#44)
  - » QuickUML: regular scan with all DPs enabeled (#45)
- WOP 1.3

Figure 6. Example of a browse page

the example all the analysis are named "regular scan with all DPs enabled"), the platform shows the analysis details page containing the list of the found DP instances, grouped by design pattern.

D. Evaluation

The evaluation phase allows the user to analyze the reported instance. The instance can be graphically viewed in two ways: in the first way using a graph representation of the tree representing the instance (it requires java working in the browser), and in the second way using nested boxes (see Figure 7) to represent the nested structure of the tree representing the pattern instance.


Abstract Class

AbstractHandle

Concrete Class

-

Uploaded by: admin
Project: JHotDraw 5.1
Tool: DPD Tool 4.5

Source code: - Choose a file -
View layout: Nested Boxes

Evaluations

by andrea @ 23/06/10 (16:18) 1 points Hide

Could be correct but it lacks the concrete class(es)

[Post a comment

Figure 7. Example of the visualization of an instance

The rest of the evaluation phase is concerned with the discussion forum about the found instance. This forum is dedicated to the evaluation of the correctness of the found instance:

- it allows to express a score (the "Stars") in the range 1-5, where 1 means the instance is fully incorrect, and 5 means it is correct; it is also possible to insert a comment to argument the evaluation;
- it lets other users to express an agreement or disagreement (the "Votes") with previous evaluations.

The overall scoring is used as a parameter in the search page (see Figure 4) to have an immediate idea about the overall correctness of a found instance.

In fact, during the search phase, the user can specify the minimum number of stars a pattern instance must have in order to be considered a correct instance. In addition, the user can specify the absolute number of agreements (or disagreements) each evaluation must have to be safely included in the overall stars computation. The overall stars of an instance is computed as the average star number weighted with the agreement balance (the difference between the number of agreements and disagreements), considering only the evaluations having the minimum number of absolute agreements. Currently, the platform allows the users to choose the number of stars in the range 1-5, and the number of agreements (called "Votes" in the platform) from 0 to 20.

We believe that this community driven system of classifying found instances provides good common datasets, for the test of new tools and the enhancement of the existing ones.

Moreover, in the evaluation phase it is possible to inspect directly the source code, selecting the file to inspect from the combo box under the graphical representation of the pattern.

V. Conclusion

In this paper we presented a platform helping the design pattern detection community having a way to compare the results produced by the tools and techniques that have been proposed in the literature.

Our final intent is not only the tool "competition" but also the creation of a container for design pattern instances that, through the users' voting, will allow us to build a large and "community validated" dataset for tool testing and benchmarking.

For all these reasons we are convinced that this kind of platform can be really valuable in our research area because it allows the real sharing of information and knowledge among all research groups interested in design patterns for both reverse and forward engineering.

In future works we are interested in integrating different comparison algorithms, maybe suggested and discussed with the DPD community, and to refine and tune the platform settings. We are also investigating the possibility to expose some kind of web services in order to let registered users to make their tool able to automate the loading of their analysis into the platform.

References

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[2] N. Shi and R. A. Olsson, "Reverse engineering of design patterns from java source code," in *ASE '06: Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 123–134.

[3] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. T. Halkidis, "Design pattern detection using similarity scoring," *IEEE Transactions on Software Engineering*, vol. 32, no. 11, pp. 896–909, 2006.

[4] Y.-G. Guéhéneuc, "Ptidej: Promoting patterns with patterns," in *Proceedings of the 1st ECOOP workshop on Building a System using Patterns*. Springer Verlag, 2005.

[5] J. Niere, W. Schäfer, J. P. Wadsack, L. Wendehals, and J. Welsh, "Towards pattern-based design recovery," in *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*. New York, NY, USA: ACM, 2002, pp. 338–348.

[6] J. Dietrich and C. Elgar, "Towards a web of patterns," *Web Semant*, vol. 5, no. 2, pp. 108–116, 2007.

[7] Y.-G. Gueheneuc and G. Antoniol, "DeMIMA: A multi-layered approach for design pattern identification," *IEEE Transactions on Software Engineering*, vol. 34, pp. 667–684, 2008.

[8] F. Arcelli, C. Tosi, M. Zanoni, and S. Maggioni, "The marple project - a tool for design pattern detection and software architecture reconstruction," in *Proceedings of the International Workshop on Advanced Software Development Tools and Techniques (WASDeTT 2008)*, Paphos, Cyprus, July 2008.

[9] F. Arcelli, C. Tosi, and M. Zanoni, "A benchmark proposal for design pattern detection," in *FAMOOSr 2008: Proceedings of 2nd Workshop on FAMIX and Moose in Reengineering*, 2008.

[10] L. Fulop, R. Ferenc, and T. Gyimothy, "Towards a benchmark for evaluating design pattern miner tools," in *Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on*, 1-4 2008, pp. 143 –152.

[11] Y.-G. Gueheneuc, K. Mens, and R. Wuyts, "A comparative framework for design recovery tools," *Software Maintenance and Reengineering, European Conference on*, vol. 0, pp. 123–134, 2006.

[12] Y.-G. Guhneuc, "Pmart: Pattern-like micro architecture repository," in *Proceedings of the 1st EuroPLoP Focus Group on Pattern Repositories*, M. Weiss, A. Birukou, and P. Giorgini, Eds., July 2007.

[13] G. Kniesel and A. Binun, "Standing on the shoulders of giants - a data fusion approach to design pattern detection," in *ICPC*. IEEE Computer Society, 2009, pp. 208–217.