A benchmark proposal for design pattern detection

Francesca Arcelli  Marco Zanoni  Christian Tosi
arcelli@disco.unimib.it marco.zanoni@essere.disco.unimib.it christian.tosi@essere.disco.unimib.it

Università degli Studi di Milano Bicocca
Dipartimento di Informatica Sistemistica e Comunicazione
Viale Sarca, 336 20126 Milano, Italy

1 Introduction

Design pattern detection is a topic which received a great interest during the last years. Finding design patterns (DP) 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 design pattern detection 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. [6], [7], [4], [5], [2]). In spite of the many approaches proposed, the results obtained are 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 the design pattern detection is the so called variant problem: design patterns can be implemented in several ways, often very different from one another. The main variants for each pattern are described in the catalog of [3], 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.

Moreover no real benchmark has not yet been proposed to compare design patterns detection tools. If one tries to compare design pattern detection tools on the same system, usually retrieves very different results and often it is not possible also to replicate and obtain the results described by the authors of the tool. In spite of the validity of the results obtained by one tool respect to the other, one relevant problem is the lack of a real benchmark to be easily used to compare the results in a sound way.

We face these problems since we are developing a tool called MARPLE (Metrics and Architecture Reconstruction plug-in for Eclipse) [1] whose main aims are related to design pattern detection (DPD) and software architecture reconstruction. For what concerns to DPD Marple is characterized by the following steps:

- the detection of sub components or micro architectures which give useful hints on the DP detection and, which aim to mitigate the variant problem
- the detection of all the possible DP candidates performed by a module called Joiner whose results are characterized by 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 (in particular through naive bayes and support vector machines classifiers); in this way we are able to reduce the output size and to sort the results by their relevance through the Classifier module

In this work we would like to propose a benchmark and an approach to be used to compare DPD tools. In this way we aim to find some mechanisms to obtain safe results and for making them and the DPD tools available to the community in an easier way. The adoption by the DPD community of a benchmark could improve the cooperation among the researchers and the reuse of tools written by other instead of the development of new ones.

Our benchmark proposal is based on the definition of a standard for the representation of the results of DPD tools. Having a common standard will permit to write applications that are able to compare the results coming from different tools.
2 What we have to know about a design pattern to represent it?

A design pattern is an organized set of classes working together. These classes respect the pattern’s design rules, and each one has a specific role assigned. So a representation of a design pattern instance must contain the classes belonging to it and the role assigned to each one.

Another issue is that in a design pattern some roles can be played by more than one class. So we need a way to specify these types of situation. We propose a tree organization for the classes of design patterns. In this representation it is possible to specify one-to-one or one-to-many relationships between the roles. Each pattern has a root that can be composed of one or more roles.

3 A benchmark proposal for DP Detection

Each design pattern must follow a definition, a schema. As introduced we model DPs as trees, so the schema is the one represented in Figure 1.

The DP is essentially defined by a name and a tree of level definitions. Each level definition is a container of roles that belong to that level and of child level definitions. The defined classes are:

- **RoleDef**: it represents a role belonging to the design pattern we are defining; it is characterized only by its identification name.

- **LevelDef**: it is a container of roles that will have to be in a one-to-one relationship when associated, and can have child level definitions, implementing the tree.

- **DPDef**: it contains the name of the defined design pattern and is associated to the root of the level definition tree.

The instance of a design pattern must follow the definition and is a more complicated structure linked to the definition in all its levels. In Figure 2 we see the UML class diagram of the instance model.
the same way it is possible to compare an instance to a validated set of known instances. If a common role naming will be adopted it will be possible to compare also instances coming from slightly different DP definitions (obviously if they define the same pattern).

In Figure 3 we show an example of definition for the abstract factory pattern.

![UML Object Diagram](image)

**Figure 3. UML Object diagram for the definition of Abstract Factory DP**

In Figure 4 we show an example of an instance of the abstract factory pattern that follows the definition depicted in Figure 3. The models and the examples are represented using UML but it’s simple to define them through example XML.

### 4 Conclusion and Future Developments

The model proposed in this paper is only a draft and could be improved and changed in the next future. This represents essentially the way we think about a design pattern’s structure. We will need for an XML schema for the definition of patterns and pattern instances, in order to have easy exchangeable data.

We hope that this proposal will allow us and the community to have a standard for the representation of the results of a design pattern detection tool and a way to compare them. We would like to realize a public service which will permit users to submit their results, compare them with the results of other tools and discuss about the experimentations. This aspect will require an extension of the model in order to be able to keep track of the user who submitted the instances, to be able to tag if each instance is a good one or not (or partially), and so on.

Our final intent does not only regards the tool competition but also the creation of a container for design patterns that, through the users’ voting, will permit us to build a large and “community validated” dataset for tool testing and benchmarking.

All of these reasons convinced us that this proposal is essential for this 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.

### References


Figure 4. UML for the example of the Abstract Factory DP instance model.