# 25 Recovering Design Elements in Large Software Systems

**Jörg Niere**
University of Siegen, Software Engineering Group
joerg.niereuni-siegen.de

**Introduction.** Reverse engineering large systems means to be able to analyse programs with more than 100K lines of code. Especially, recovering design elements means, e.g. detecting classes, relations or design-patterns, or recovering the architecture of a program. All activities need large analysis parts, because they nearly span over the whole system and manifest themselves not only in small parts.

In addition to the large analysis parts, design elements are usually described on a high abstract level. For example, design-pattern descriptions introduced by Gamma et al. contain more parts written in prose than parts described with diagrams. This makes them flexible for the application in an actual system design, but hard to detect according to of the large variety of implementation variants.

Current approaches of recovering design elements use automatic analyses. On the one hand, approaches performing fine-grained analyses fail, because of the large system size. On the other hand, approaches performing coarse-grained analyses produce too many false-positives. In general we have a trade-off between the granularity of the analysis and the size of the system to be analysed.

**Calibrating rule catalogs.** Our approach to recover design elements is to use a rule catalog, which describes the design elements in terms of graph rewrite rules [4]. The analysed source code is presented as an abstract syntax graph or may be represented as any other kind, e.g. control- or data-flow graphs.

The idea of our approach is that the rule catalog to analyse the system contains only those rules that fit to the system. In means of design patterns the rules describe only the implementation variants included in the system and no others. Therefore, our approach highly involves the reverse engineer in an anlysis process to calibrate the rule catalog to his/her needs, cf. Figure 25.1. This allows for performing fine-grained and coarse-grained analysis and for letting the catalog as small as possible enabling analysis of large systems.

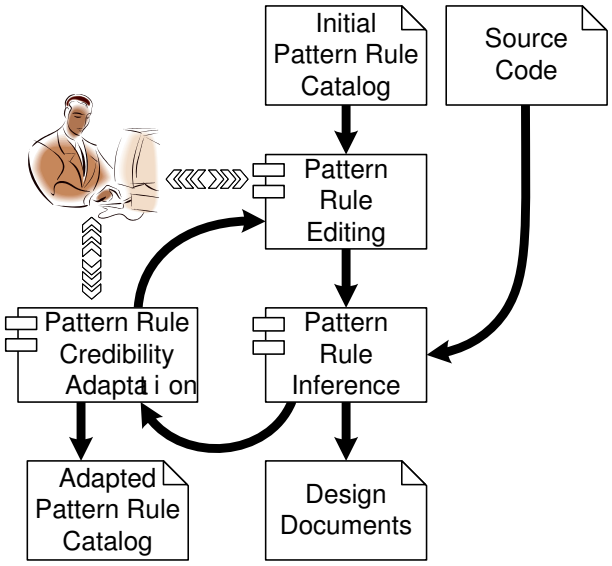


Figure 25.1: Interactive Rule Catalog Calibration

The process starts with an initial catalog of rules, which the reverse engineer may edit (*Editing*). In the *Inference* step, the rules are automatically applied by an inference algorithm, which produces meaningful results early and not only after a complete analysis of the system. The reverse engineer can interrupt the inference to inspect the results produced so far.

Our approach supports the inspection of the reverse engineer with accuracy values assigned to each result [5]. The accuracy values result from credibility values which are attached to the rules. So far, the choice of the credibility values highly depends on the personal experience of the reverse engineer. To overcome this limitation the process includes the *Credibility Adaption* step, where the credibility values are automatically adapted based on changes of accuracy values made by the reverse engineer. In addition to changing accuracy values the reverse engineer may also add hypotheses and let them validate in the inference step. For more details see [1]

**Practical experiences.** The approach has been implemented as plugins for the Fujaba Tool Suite and is available under www.fujaba.de. For each step in the process exists a separate plugin in order to test different inference or adaptation strategies.

The prototype has been used to analyse different systems. For example it has been used to detect design-patterns in Java's Abstract Windowing Toolkit (AWT) consisting of about 114 KLOC. The screenshot in Figure 25.2 shows a cut-out of the results of this analysis as UML class diagrams. The detected design-patterns are represented by dashed ovals consisting of the pattern's name and the accuracy value. We took this analysis as proof-of-concept, because the design-patterns included in the library are known.

In order to show the scalability of the approach, we analysed the Fujaba Tool Suite itself with about 750 KLOC. The goal of the analysis was also the detection of design-patterns.

We have also used the prototype to analyse the JigSaw webserver and have recovered the hot spots of the system detecting architecture patterns. The results were comparable to others produced by different reverse engineering tools. During the analysis process early produced meaningful results have been taken to proof hot spot hypotheses at certain parts of the system. Compared to other tool's results, ours have not only pointed out the hot spots but also additional information such as bridges between parts or central data structures.

The prototype was also successfully used in an industrial context on an application with more than 300 KLOC. The result of the analysis was that design decisions have been neglected by the developers implementing the system.

**Further research.** Our approach is not limited to detect design-patterns. Currently the prototype is used to detect Anti-Patterns and to rewrite 'bad' patterns by 'good' ones. Therefore the rules get property values describing positive and negative aspects.

Another research direction is to combine the presented approach with other ones. As example, the approach has been combined with a dynamic analysis. The static analysis indicates possible design-pattern instances, which will be verified by a dynamic analysis. Technically, an (intermediate) analysis result is a graph and therefore exchangeable with other tools using common exchange formats such as GXL.

A significant aspect concerning the overall duration of an analysis is the choice of the initial rule catalog. Using a catalog with coarse-grained rules means much iteration

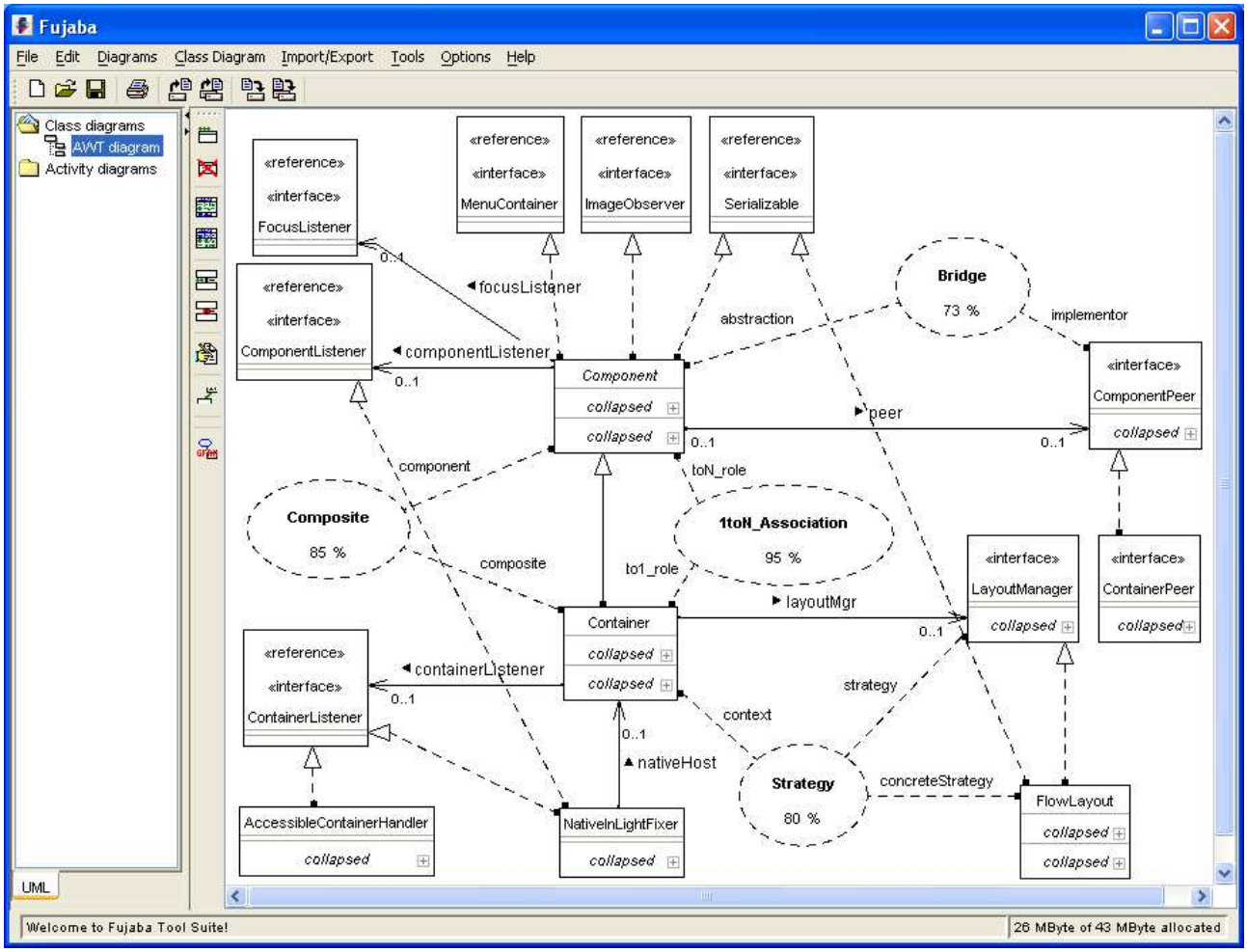


Figure 25.2: Annotated UML Class Diagram

to get a detailed result. Starting with a catalog with fine-grained rules may have the effect that the rules are not applicable, which means to get no results. Therefore we currently investigate the opportunity to use problem management systems for catalogs and their relations to analysed systems.

## Bibliography


[1] J. Niere. *Incremental Design-Pattern Recognition*. PhD thesis, University of Paderborn, Paderborn, Germany, 2004. in german (to appear).

[2] J. Niere, W. Schäfer, J. Wadsack, L. Wendehals, and J. Welsh. Towards pattern-based design recovery. In *Proc. of the* $24^{th}$ *International Conference on Software Engineering (ICSE), Orlando, Florida, USA*, pages 338–348. ACM Press, May 2002.

[3] J. Niere, J. Wadsack, and L. Wendehals. Handling large search space in pattern-based reverse engineering. In *Proc. of the* $11^{th}$ *International Workshop on Program Comprehension (IWPC), Portland, USA*, pages 274–279. IEEE Computer Society Press, May 2003.