

PTIDEJ: A Tool Suite

Yann-Gaël Guéhéneuc, Jean-Yves Guyomarc'h, Duc-Loc Huynh,
Olivier Kaczor, Naouel Moha, and Samah Rached

Ptidej Team

GEODES – Group of Open, Distributed
Systems, Empirical Software Engineering

Department of Informatics and Operations Research
University of Montreal – CP 6128 succ. Centre Ville
Montréal, Québec, Canada, H3C 3J7
{guehene, guyomarj, hyunhduc, kaczorol,
mohanaou, rachedsa}@iro.umontreal.ca

Abstract

Ptidej is a tool suite dedicated to the analysis and maintenance of object-oriented architectures. It includes four tools: (1) SAD, a tool for the detection and correction of software architecture defects; (2) EPI, a tool for the efficient identification of design patterns occurrences in a program; (3) DRAM, a tool for the visualisation of the static and dynamic data of programs with adjacency matrices; and (4) Aspects, a tool for modeling and computing metrics on aspect-oriented abstractions.

1 Introduction

The PTIDEJ project (Pattern Trace Identification, Detection, and Enhancement in Java) aims at developing a tool suite to evaluate and to enhance the quality of object-oriented programs, promoting the use of patterns, at language-, design-, or architectural-level [5]. More precisely, PTIDEJ is a reverse engineering tool suite to build program representations from static and dynamic models of Java programs semi-automatically [8] and to perform analyses.

Four tools have been added to PTIDEJ :

- SAD: a tool for the detection and correction of software architecture defects.
- EPI: a tool for the efficient identification of design patterns occurrences in a program.
- DRAM: a tool for the visualisation of the static and dynamic data of programs with adjacency matrices.
- ASPECTS: a tool for modeling and computing metrics on aspect-oriented abstractions.

All these tools aim at evaluating the quality of object-oriented programs and help maintainers and developers in improving their programs by highlighting the design of their programs.

2 Ptidej

2.1 Overview of Ptidej

Architecture. PTIDEJ has been developed in Java under the ECLIPSE platform. It decomposes in about 30 projects, 190 packages, and 1,150 classes for 74,000 lines of code. PTIDEJ is an open-source project, technically mature and robust. Ptidej is easy to install, to configure,

Copyright © 2005 Yann-Gaël Guéhéneuc. Permission to copy is hereby granted provided the original copyright notice is reproduced in copies made.

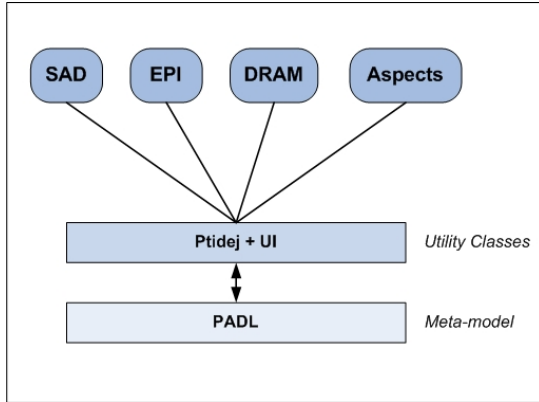


Figure 1: Design of PTIDEJ

and to extend, thanks to the the configuration files provided and its open architecture.

Design. PTIDEJ is based on the PADL metamodel (*Pattern and Abstract-level Description Language*) [1], with which to represent programs at different levels of abstraction. PADL offers constituents, such as `Model`, `Class`, `Method`, `Relationship`, with which we can build representations of programs. It offers also methods to manipulate these representations easily and to generate other representations, using the *Visitor* design pattern. Figure 1 illustrates the overall design of PTIDEJ.

Interface. Figure 2 presents the interface of the PTIDEJ tool suite with a simple document description program as example. It consists of two panels to display representations of programs (left) and to control the tool (right).

In particular, the vertical toolbar on the right lists all the design patterns that the tool can detect in object-oriented programs. Another toolbar in the second tab “Tools” presents all the tools offered by PTIDEJ, including the four tools previously presented. The next tab controls the display of the graphical representations of inheritance, aggregation, and composition relationships, and names of binary class relationships.

Source Code and Documentation. The PTIDEJ source code, a documentation for its installation, and configuration files are all open

source¹. A demo is available on the website dedicated to the tool and the tool includes a repository of examples.

3 Tool Suite

PTIDEJ has been seamlessly extended with four tools. We present these tools briefly, discuss their functionalities, and summarise their contributions.

3.1 SAD

The SAD (Software Architectural Defects) tool aims at detecting automatically software architectural defects and as correcting these by applying refactorings [6]. We include antipatterns and design defects in SAD [15]. Antipatterns are bad solutions to recurring design problems. For example, the antipattern “Blob” corresponds to one single complex controller class that monopolises the processing and is surrounded by simple data classes [3]. Design defects are occurring errors in the design of the software that come from the absence or the bad use of design patterns [7].

3.2 EPI

The EPI (Efficient Pattern Identification) tool aims at identifying efficiently occurrences of design patterns [7] in a program. It can help maintainers understand a program by recovering specific parts of its design or by identifying possible defects in the program architecture. It uses a constraints-based algorithm, which is an adaptation of bio-informatics string matching algorithms [2]. The algorithm finds exact and approximate occurrences of design patterns using bit-wise operations on a finite set of bit-vectors representing a program. The inherent parallelism of bit-wise operations combined with an adequate use of constraints makes this approach reliable and very efficient compared to other popular approaches such as logic programming [13, 16] or fuzzy reasoning nets [11].

¹All files are available at:
ptidej.iro.umontreal.ca/downloads/toolsuite

3.3 DRAM

The DRAM (Dynamic Relational Adjacency Matrix) tool allows visualising static and dynamic relationships between classes of large programs to help maintainers in the comprehension of their programs. We use dynamic analysis to generate program traces, we enrich this data with static analysis, and then we develop different data conversion algorithms to adjacency matrices: grouping by classes, and search of program functionalities.

3.4 Aspects

The ASPECTS tool aims at modeling and compute metrics on aspect-oriented abstractions. Focusing on the issues related to the language ASPECTJ [12] (the aspect-oriented extension to Java), this tool helps in understanding the impact of aspect-oriented programming on object-oriented programs wrt. quality [9].

4 Related Work

There exist several tools that provide similar functionalities as PTIDEJ in the research community or in the industry. We present some of them briefly and discuss their limitations.

Research Tools. Several tools has been built along the years in the software engineering community to apply new research ideas and as proofs of concept. Among all these tools, we can cite the FAMOOS project at Bern². This project consisted in defined reengineering techniques and tools for object-oriented programs. We can also cite the FUJABA project at Paderborn³, which consists in a UML-like CASE tool for Java. These are but examples of research projects related to improving the quality of programs through reengineering. Finally, we can cite the SABLE project at McGill⁴ to provide a variety of tools that will lead to better understanding and faster execution of Java programs.

²See www.iam.unibe.ch/scg/Archive/famoos/.

³See wwwcs.uni-paderborn.de/cs/fujaba/.

⁴See www.sable.mcgill.ca/.

Industrial Tools. McCabe, Eclipse, and the Refactoring Browser are among the industrial tools most similar to PTIDEJ. Based on software metrics, McCabe analyses the complexity of source code, to identify the areas of a program that are at the highest risk of failure [14]. However, this tool is only based on metrics and so analyses only the static structure of programs contrary to PTIDEJ, which analyses the dynamic behaviour of programs. Eclipse also provides a plugin that allows to perform metrics on the programs [4], but as McCabe, it does not perform dynamic analyses. The Refactoring Browser is a tool which provides several refactorings that allow programmers to change rapidly the source code without affecting its behaviour [10]. However, these refactorings are at the code level while the SAD tool of PTIDEJ aims at proposing refactorings at the design level.

5 Conclusion

We presented PTIDEJ, a tool suite for reengineering object-oriented programs in class diagrams. Four tools are provided by PTIDEJ: SAD, EPI, DRAM, and ASPECTS.

PTIDEJ is a tool suite fully implemented, tested and used by our research team and other researchers in the community. So far, there is no industrial partner but we believe that PTIDEJ could stir the industry's interest because of its capabilities.

We will take advantage of the exhibition to get feedback from the industry and researchers to evaluate and to improve our tool suite.

References

- [1] Hervé Albin-Amiot and Yann-Gaël Guéhéneuc. Meta-modeling design patterns: Application to pattern detection and code synthesis. In Bedir Tekinerdogan, Pim Van Den Broek, Motoshi Saeki, Pavel Hruby, and Gerson Sunyé, editors, *proceedings of the 1st ECOOP workshop on Automating Object-Oriented Software Development Methods*. Centre for Telematics and Information Technology, University of Twente, October 2001. TR-CTIT-01-35.

- [2] Anne Bergeron and Sylvie Hamel. Vector algorithms for approximate string matching. *International Journal of Foundation of Computer Science*, 13(1):53–66, February 2002.
- [3] William J. Brown, Raphael C. Malveau, William H. Brown, Hays W. McCormick III, and Thomas J. Mowbray. *Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley and Sons, 1st edition, March 1998.
- [4] Eclipse. McCabe, August 2005. Provide metrics calculation and dependency analyzer plugin for the Eclipse platform. Measure various metrics with average and standard deviation and detect cycles in package and type dependencies and graph them.
- [5] Amnon H. Eden and Rick Kazman. Architecture, design, implementation. In Laurie Dillon and Walter Tichy, editors, *proceedings of the 25th International Conference on Software Engineering*, pages 149–159. ACM Press, May 2003.
- [6] Martin Fowler. *Refactoring – Improving the Design of Existing Code*. Addison-Wesley, 1st edition, June 1999.
- [7] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1st edition, 1994.
- [8] Yann-Gaël Guéhéneuc. A reverse engineering tool for precise class diagrams. In Janice Singer and Hanan Lutfiyya, editors, *proceedings of the 14th IBM Centers for Advanced Studies Conference*. ACM Press, October 2004.
- [9] Jean-Yves Guyomarc’h and Yann-Gaël Guéhéneuc. On the impact of aspect-oriented programming on object-oriented metrics. In Fernando Brito e Abreu, Coral Calero, Michele Lanza, Geert Poels, and Houari A. Sahraoui, editors, *proceedings of the 9th ECOOP workshop on Quantitative Approaches in Object-Oriented Software Engineering*, pages 42–47. Springer-Verlag, July 2005.
- [10] The Refactory Inc. Refactoring browser, October 1999. In addition to the features present in the standard browsers, the Refactoring Browser provides several refactorings that allow programmers to rapidly change their code without affecting its behavior. This allows programmers to rapidly make design changes to existing code without having to worry about introducing errors into the code. In addition to the refactorings, the Refactoring Browser provides several productivity enhancements that programmers have requested to make the browser a better programming tool.
- [11] Jens H. Jahnke, Wilhelm Schäfer, and Albert Zündorf. Generic fuzzy reasoning nets as a basis for reverse engineering relational database applications. In Mehdi Jazayeri, editor, *proceedings of the 6th European Software Engineering Conference*, pages 193–210. ACM Press, September 1997.
- [12] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of AspectJ. *Lecture Notes in Computer Science*, 2072:327–355, 2001.
- [13] Christian Krämer and Lutz Prechelt. Design recovery by automated search for structural design patterns in object-oriented software. In Linda M. Wills and Ira Baxter, editors, *proceedings of the 3rd Working Conference on Reverse Engineering*, pages 208–215. IEEE Computer Society Press, November 1996.
- [14] McCabe. McCabe, August 2005.
- [15] Naouel Moha and Yann-Gaël Guéhéneuc. On the automatic detection and correction of software architectural defects in object-oriented designs. In *Proceedings of the 4th ECOOP Workshop on Object-Oriented Reengineering*, July 2005.
- [16] Roel Wuyts. Declarative reasoning about the structure of object-oriented systems. In Joseph Gil, editor, *proceedings of the 26th conference on the Technology of Object-Oriented Languages and Systems*, pages 112–124. IEEE Computer Society Press, August 1998.

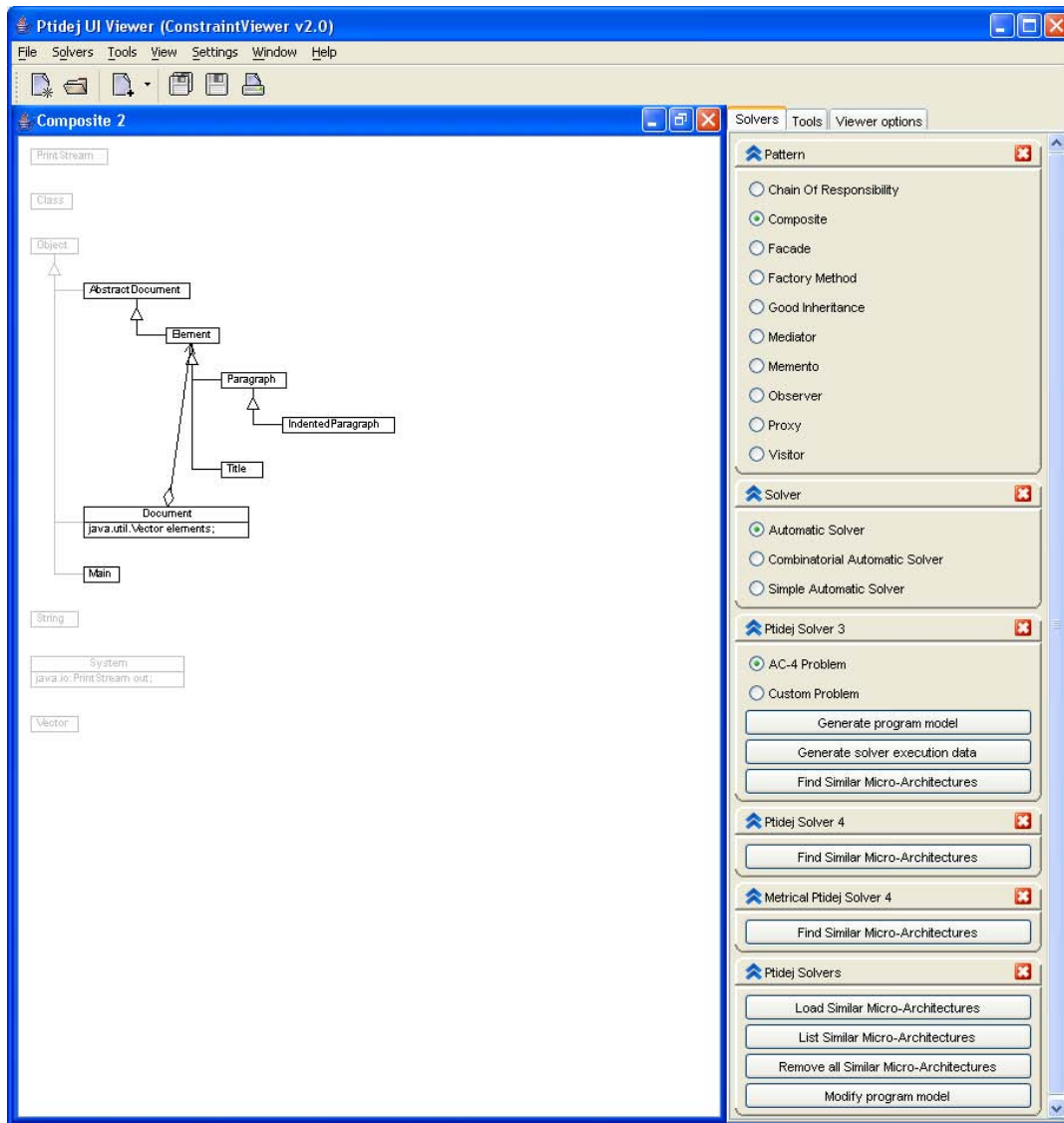


Figure 2: Interface of PTIDEJ