Using Learning Towards Automatic Reengineering
(Position Paper)

Jörg Niere
Software Engineering Group, Department of Mathematics and Computer Science
University of Paderborn
Warburger Str. 100, D-33098 Paderborn
Germany
nierej@uni-paderborn.de

Keywords
reengineering, soft computing, user interaction, learning, fuzzy logic, fuzzy reasoning

1 INTRODUCTION

Reengineering is a big challenge in computer science [MJS+00]. The Y2K problem was one of the major reengineering projects preponderant successfully resolved and the Euro conversion problem is still running until the end of this year. Automatic reengineering and analysis approaches have been developed and have resolved approx. 80%. Many of the rest has been done by hand, but there are still some percentages left unresolved. Typically, the developed analysis solutions are rather problem specific and can not be used for other problems.

Although the original application to reengineer is in a consistent state, inconsistency must be handled by non-monotonic reasoning during the reengineering process. Typically large software systems must be analysed partly by different tools which includes imperfect knowledge. Furthermore, imperfect knowledge results from non-available documentation and design of a system.

However, two major problems have to be solved: (1) handlin of the number of different implementation variants to ensure reliable results and (2) the applicability of the approach for large software systems. Handling the number of different implementation variants is well understood and solved by more or less complex algorithms. Usually, many approaches fail for large systems, e.g. [Wil96] and [KSRP99]. Thus there is a trade-off between both problems.

One possibility to raise the software system's size to analyse is to involve the reengineer particularly in those cases where the automatic analysis can not compute further results. Such an approach to analyse relational databases to (re)construct an object-oriented model is presented in [Jah99]. The approach allows to analyse a database, e.g. schema, applications, and dat step by step to handle the complexity. User interaction during the reengineering process fixes fuzzy intermediate results. Inconsistencies during the analysis process can be discarded by the reengineer or retained, expecting that further analysis discards the inconsistency. However, the approach is semi-automatic and results in many user input (interaction) when applying the approach to other domains, e.g. [JNW00, NWZ01].

The idea presented in this position paper is to combine a pattern matching approach to handle problem (1), with semi-automatic analysis to solve problem (2), and some kind of easy learning strategies to raise the automation of a reengineering process.

2 INCONSISTENCY AND USER INTERACTION

Jahnke and Walenstein classify in [JW00] reverse engineering as media for imperfect knowledge. They state, handling inconsistency and imperfect knowledge combined with user interaction is one solution to obtain better results concerning the applicability of a reengineering approach for large software systems.

The presented database reverse engineering process iterates an automatic analysis part followed by user interaction until there are no further information to extract from the underlying system and the result is consistent. Inconsistencies have to be resolved by the user.

Formal basis are Generic Fuzzy Reasoning Nets (GFRN) which allow a graphical notation of the analysis including uncertainty and inconsistency. Starting points for the analysis are so called clichés handled as axioms in the inference process. For database reverse engineering purposes there exists a number of well-known common clichés to analyse SQL-l fragments which occur in several database systems.

During the process, the user is able to fix intermediate results and to control and navigate the automatic analysis. Thereby, the user operates on incomplete knowledge and this may lead to wrong user decisions, i.e. to inconsistencies in further analysis. These 'new' inconsistencies have to be resolved again by the user.

Experiences have shown that the user involvement decreases within the reverse engineering progress. This results mainly from the fact that the analysis process is conducted by the user in a certain direction and this reduces the opportunities for further analysis. However, the user has to fix all fuzzy results in order to get a consistent analysis result. Also the reuse of common clichés produces good results applying to other database reverse engineering projects.

Applying the approach to other software reengineering projects, e.g. for recovering design patterns from Java source code have shown that common clichés can also be identified but on a lower level of abstraction, see [JNW00, NWZ01]. Employing the approaches to some example projects has shown that the user interaction increases dramatically in comparison to experiences in the database reverse engineering process. The reason for this phenomenon lies in the high number of implementation variants for one cliché. Typically there exist many syntactically different implementations with the same semantics in comparison to clichés in database reverse engineering. Using abstract representations of clichés known from classical compiler techniques, e.g. abstract syntax graphs, or code normalization solve the problem, only partly.

3 USING LEARNING TO INCREASE AUTOMATION

Our solution to improve the approaches in order to decrease user interaction is to learn from the interaction and adapt the GFRN to raise the automation degree of the process.

The idea is to take advantage of preferences and affectations of software developers, e.g. some developer prefer 'for'-loops other 'while'-loops. So, in a software system there can be found many comparable pattern like implementation variants of the same developer.

During the original reengineering process, the reengineer has to fix all fuzzy results by hand. Learning from user interaction in this case means that the underlying GFRN will be adapted when the reengineer fixes several times the same implementation variant. Next time a comparable implementation variant is analysed, the process fixes the situation automatically.

In general, it can not be excluded that the learning process will adapt the net in a wrong way in case of wrong 'training' material, e.g. the reengineer makes wrong decisions on not representative information. Therefore the reengineer has to revert some of his/her prior decisions and train the net again. The opportunity to collect user interaction and adapt the net at a certain time by the reengineer should result into better solutions.

Such an approach allows to reuse an adapted net to analyse software developed by persons with comparable preferences and to benefit from previous results. Applying the approach to other reengineering projects analysing software from different developers means resetting the learning values. This should allow a high degree of reuse and may help to avoid starting from the scratch each time.

There exist many papers and other literature about learning procedures and algorithms, e.g. fuzzy logic, reasoning nets and neural networks. [JS99] presents a possibility to integrate a learning algorithm based on neural nets into the reverse engineering process. In practice, this learning algorithm fails because the data overhead is too large and slows the process. To avoid such an overhead we want to use learning based on statistics evaluation of the user interaction. Collecting statistic information does not produce acceptable overhead and should adapt the weights of the underlying GFRN.

Another opportunity to learn from the user's interaction is to extend the GFRN during the reengineering process. The reengineering process can start with low basic knowledge and is flexible enough to extend this knowledge on the fly. This tackles the problem to provide a common cliché library either for different kinds of software systems as well as for software developed by different persons.

4 CURRENT AND FUTURE WORK

We want to integrate our reengineering process based on the presented ideas into our Fujaba environment (From UML to Java And Back Again). Fujaba is a case tool with automatic code generation from an UML specification. The tool supports currently a round-trip engineering process, which allows to modify generated code and to recover the specification diagrams on basis of the code, only. Therefore Fujaba uses some heuristics and naming conventions. The process is fully automatic and fails in recovering specification diagrams if the code does not hold some implementation style guide conventions. We are currently exchanging the hard-coded reengineering process with a semi-automatic user involved process to improve the applicability to larger software systems. We use GFRN as the basis for the process and focus on the development and integration of the learning facilities described above. Since pattern matching is used, we have to develop database support also with main focus on exchanging certain pattern definitions to ensure a high reusability degree of patterns.
For more details see `http://www.fujaba.de/`

ACKNOWLEDGMENTS

The following people contributed substantially to this work with fruitful discussions, careful proof readings, and a lot of suggestions: Jens H. Jahnke, Holger Giese, Albert Zündorf, and Jörg P. Wadsack. Thank you very much.

REFERENCES

[Jah99] J.H. Jahnke. *Management of Uncertainty and Inconsistency in Database Reengineering Processes*. PhD thesis, University of Paderborn, Paderborn, Germany, September 1999.

[JNW00] J.H. Jahnke, J. Niere, and J.P. Wadsack. *Automated Quality Analysis of Component Software for Embedded Systems*. In Proc. of the 8th Int. Workshop on Program Comprehension (IWPC), Limerick, Irland, pages 18–26. IEEE Computer Society Press, 2000.

[JS99] J.H. Jahnke and C. Strebin. *Adaptive Tool Support for Database Reverse Engineering*. In Proc. of 1999 Conference of the North American Fuzzy Information Processing Society, New York, USA, June 1999.

[JW00] J.H. Jahnke and A. Walenstein. *Reverse Engineering Tools as Media for Imperfect Knowledge*. In Proc. of the 7th Working Conference on Reverse Engineering (WCRE), Brisbane, Australia. IEEE Computer Society Press, 2000.

[KSRP99] R.K. Keller, R. Schauer, S. Robitaille, and P. Page. *Pattern-Based Reverse-Engineering of Design Components*. In Proc. of the 21th Int. Conf. on Software Engineering, Los Angeles, USA, pages 226–235. IEEE Computer Society Press, May 1999.

[MJS+00] H.A. Müller, J.H. Jahnke, D.B. Smith, M.A. Storey, and K. Wong. *Reverse Engineering: a roadmap*. In A. Finkelstein, editor, Future of Software Engineering. Int. Conf. on Software Engineering (ICSE), Limerick, Irland. ACM Press, June 2000.

[NWZ01] J. Niere, J.P. Wadsack, and A. Zündorf. *Recovering UML Diagrams from Java Code using Patterns*. In Proc. of 2nd Workshop on Soft Computing Applied to Software Engineering, Enschede, The Netherlands, Lecture Notes in Computer Science (LNCS). Springer Verlag, 2001.

[Wil96] L.M. Wills. *Using Attributed Flow Graph Parsing to Recognize Programs*. In Proc. of Int. Workshop on Graph Grammars and Their Application to Computer Science, LNCS 1073, Williamsburg, Virginia, 1994, November 1996. Springer Verlag.