Identifying Impacts of Database Schema Changes on Applications

Amela Karahasanovic
amela@ifi.uio.no
Industrial Systems Development Group,
Department of Informatics, University of Oslo
P.O. Box 1080 Blindern, N-0316 Oslo, Norway
Tel: +47 22 84 00 60 Fax: +47 22 84 00 72

April 16, 2001

Abstract

Research in schema evolution has been driven by the need for more effective software development and maintenance. Finding impacts of schema changes on the applications and presenting them in an appropriate way are particularly challenging. We have developed a tool that finds impacts of schema changes on applications in object-oriented systems. This tool displays components (packages, classes, interfaces, methods and fields) of a database application system as a graph. Components potentially affected by a change are indicated by changing the shape of the boxes representing those components. We have evaluated the tool by own judgement on a real-life application and by a controlled student experiment. Our results indicate that identifying impacts at the level of fields and methods can reduce the time needed to conduct schema changes and reduce the number of errors compared with identifying impacts at the level of classes. Moreover, the subjects of the experiment appreciated the idea of visualizing the impacts of schema changes.

1 Introduction

Managing change is a fundamental aspect of information and database systems. Changes to a database schema after a system has become operational may be quite frequent and will typically affect other parts of the schema, the data in the database and the applications [1]. When a schema is changed, consistency between the database schema, the database and the applications should be preserved. Due to the complexity of dependencies between them, supporting a gradual process of schema change, schema evolution, is considered expensive and troublesome. Finding impacts of schema changes on the applications and presenting them in an appropriate way are particularly challenging.

Dependencies between database schemata and applications are even more complex in object-oriented database application systems than in conventional systems. Transitivity of inheritance and of aggregation structures makes it difficult to understand dependencies between classes, methods, and fields of a system [2].

Research conducted in the area of schema evolution can be classified into three main approaches. These are direct schema evolution, schema versioning and schema evolution by views. *Direct schema evolution* [3-8] is supported when schema changes are allowed without loss of existing data. The old schema and the old state of the schema extension are inaccessible after the change took place. *Schema versioning* [9-17] is supported when schema changes produce new state of the schema (new schema version) and the extension associated with it. Accessing different schema versions and corresponding extensions is allowed both prospectively and retrospectively. Schema evolution can also be supported *by views* [18, 19]. When a schema change request is received, a new schema view is computed with the desired semantic. The majority of schema evolution research prototypes [3-19] provide support for maintaining consistency between the database schema and data. Different strategies for maintaining consistency between the database schema and the applications have been developed. Models based on versioning and views maintain consistency between the schema and applications by keeping the old schema versions. A schema change produces a new schema version or view. Old application programs can continue to be used together with the old schema version or the old view version. These approaches cannot handle situations when a new version of a schema is meant to be used in the old applications. *A priori* change impact analysis [21] identifies impacts of schema changes on applications.

Proposed schema evolution technologies have typically been evaluated according to performance [5, 8, 22, 23] and provided functionality [19, 20, 24, 25]. Even if the primary goal of the research in schema evolution has been to support schema and application developers in software development and maintenance, little attention has been paid to the usability evaluation of the proposed schema evolution technologies.

1.1 Research Questions

The discussion above motivates our research. The goal of this research is to provide a technology for maintaining consistency between *database schemata* and their *corresponding applications* (application compatibility) in an object-oriented context, and evaluate it from the perspective of the schema and application developers. Does *a priori* impact analysis help schema and application developers in maintaining these consistencies? To answer this question, several questions that are narrower in their scope have to be addressed. These are:

- RQ1 How do the existing schema evolution tools support application compatibility?
- RQ2 To what extent can former work within pure *software change impact analysis in programming languages* contribute to identifying impacts of schema changes on the applications?

- RQ3 What is a trade-off between a textual and visual presentation of the impacts of schema changes?
- RQ4 How to evaluate (including conducting experiments) schema evolution technologies regarding their support of application compatibility?

The remainder of this paper is organised as follows. Section 2 describes research methods used to address research questions. A technology we propose is briefly presented in Section 3. An evaluation of the technology is given in Section 4. Section 5 concludes and proposes future work.

2 Research Methods

Various research methods can be used for evaluating software engineering methods and tools [26, 27]:

- **Qualitative screening** – a feature analysis based on existing literature describing tools. A major weakness of this method is its subjectivity in selecting both features and their importance. On the other hand, because of its flexibility and low costs, this method is used very often.
- **Assertion** – an experiment where a developer uses a tool. As the developer is both experimenter and subject of study, this method is considered potentially biased. However, if the experiment is done in the context of a large industrial project, it can be classified as a case study. The major weakness of the case study is that it is very context biased (each development project is unique). This method is often used for preliminary evaluation.
- **Controlled experiment** – several subjects perform a task in multiple ways. The researcher determines factors that can influence the results, e.g., duration and staff experience. Therefore, it is possible to provide a greater level of statistical validity than in case studies. However, the high cost of the experiment limits the possibility of replication.

2.1 A Tool for Automatic Data Collection

To conduct experiments on schema evolution technologies, we have developed a tool that collects data about the subjects of an experiment and their usage of the technology under study [28]. In the beginning of the experiment, the Personal Information Questionnaire screen appears. It requests information about experience, cognitive style and attitude to learning new tools. At the end of the experiment, the User Evaluation Questionnaire screen appears, which is based on the IBM Post-Study System Usability Satisfaction Questionnaire [29]. Behavioural data is collected in the *think-aloud screen*. The experiment participants are asked to write short comments on what they are thinking during the experiment in this screen. To remind them, the screen changes colour every 10 minutes and the text *'What are you thinking now?'* appears. These comments are saved in a text file, and the time used

Table 1: **Summary of research questions and methods**

Presented in	**Research questions**	**Research methods**	**Status**
Section 1	RQ1	Qualitative screening, controlled experiment	Partly addressed, an experiment to be conducted
Section 3, Papers [30,31]	RQ2, RQ3	Implementation	Partly addressed, implementation of some algorithms remained
Section 4	RQ2, RQ3	Assertion, controlled experiments	Partly addressed, more experiments to be conducted
Section 2.1, Section 4, Paper [28]	RQ4	Proposal of a framework, controlled experiments	Addressed

for writing them is recorded. The think-aloud screen allows us to conduct experiments with groups of students working in the same room and makes data processing easier. Usage of window operations and *unix* commands is logged together with execution timestamps. All keystrokes and operations on the mouse buttons within an adapted *emacs* window are also recorded.

2.2 Research Questions and Methods

Our research can be described with respect to the research methods used to address the research questions raised in Section 1.2. The relation between research questions and research methods is summarised in Table 1.

3 Proposed Technology – SEMT

Software change impact analysis is a research area considering consequences and implementation of a software change on software artefacts like design documents, documentation, source code and test material. The transitive closure algorithm is considered to be one of the fundamental impact-analysis techniques. This algorithm takes as input a directed acyclic graph with components as nodes and relationships between them as edges. It then finds all components reachable from a given component.

We apply this algorithm to find the effect of schema changes on applications in the following way. Schema and application components, i.e., packages, classes, interfaces, fields and methods of a database application system are represented as nodes and the relationships between them are represented as edges. The kinds of relationship between components are inheritance, encapsulation, aggregation and usage. When a schema change request appears, there are two possibilities:

- this change will delete/modify a component/relationship in the search space

- this change will insert a new component/relationship in the search space.

In the first case, the transitive closure algorithm starts with a given component, while in the second case it starts with a component semantically enclosing a given component. For example, if a field is to be added to a class, then this class encloses the new field. Real-life database application systems typically consist of numerous components. The original algorithm [32] with execution time $\mathcal{O}(n^3)$, where *n* is the number of components will thus often give unsatisfactory performance. Hence, several improved variants have been developed [33, 34]. Their execution times are evaluated in [33]. We have chosen the Delta-wavefront algorithm because it is more efficient than the original algorithm and independent of low-level memory management. For several schema changes we have adapted the algorithm to achieve greater precision of identified impacts.

Based on the above explained ideas, we have developed a tool called SEMT (Schema Evolution Management Tool) in the context of object oriented systems. Applications access persistent classes definitions through the ODMG bindings. SEMT takes as input Java source files of a schema and applications. Components are extracted from the source files and the relationships between them are identified. Components are presented as nodes, and relationships between them as edges in a graph. Impacts of a schema change on the rest of the schema and the applications are identified and presented. SEMT offers the following functionality:

- visualising a schema and applications
- creating, modifying or deleting a schema
- visualizing potential effects of a proposed schema change
- canceling a change
- confirming a schema change – rollback is possible
- committing a schema change – rollback is not possible

The following schema changes are currently supported by SEMT:

- Changes to the properties of a class
  - add/delete/rename a field
  - add/delete/rename a method
- Changes to the inheritance graph
  - add a class to the superclass list of a class
  - delete a class from the superclass list of a class
- Changes to classes

– add/delete a class

We try to make the visual syntax of SEMT as simple and intuitive as possible. To achieve *efficient information assimilation* [35], we propose usage of graphs with limited number of schematic figure types (rectangles, ellipses, lines, and arrows) and colours. To achieve *consistency of data display* [35], we propose use of the same colours and shapes for the same type of components and relationships.

SEMT displays a search space as a graph. All component types are displayed as rectangles with the name of the component written inside. The size of the rectangles carries no additional information. Colours are used for denoting different component types and coloured directed arrows for denoting different relationships between the components.

The techniques explained above will produce very large and incomprehensible graphs when applied to real-life database application systems. Therefore, we decided to present only components and dependencies at the coarse granularity (packages, classes and interfaces). If the user wants to explore the graph more closely, he may expand classes and interfaces. The user may choose between presentation of fields or/and methods. Corresponding components and dependencies at the granularity of fields and methods are then presented. Our solution is a variation of hierarchical graphs limited by the implementation technology we have used [36].

4 Evaluation of the Proposed Technology

Schema evolution technologies can be compared and evaluated from different viewpoints. Managers may be interested in costs and utility of a schema evolution technology while schema and application developers may be interested in usability of the same technology. We focus on the usability of schema evolution technologies. ISO 924-11 defines *usability* as 'the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use'.

4.1 A Real-Life Application

A version of SEMT developed in the context of C++ and the object-oriented Pse/Pro was evaluated regarding performances and usability. A Norwegian company developing a CASE tool provided us with the source code of the tool. The tool is implemented using C++ and ObjectStore Pse/Pro. SEMT was tested on about 300 header and source files stored in 6 directories (about 135000 lines of source code). It took about 30 minutes to parse the files and store the components in the repository. As parsing of all the files represents a seldom-performed operation, this should not be considered as a problem. However, even if tested by the authors themselves, this version of user interface was not experienced as user friendly. With a large number of the components presented in the same graph, several screens were needed to present the components potentially affected by the proposed schema change.

4.2 A Controlled Experiment

We have conducted a controlled student experiment to evaluate usability of SEMT. The subjects of this experiment were 13 students from the Computer Science Department, University of Oslo, who conducted schema changes on a library application. All of them were unfamiliar with the functionality of SEMT. They were provided short training (15 minutes). They had access to the SEMT documentation during the experiment.

The independent variables of the experiment were the granularity level, complexity of the database application system, complexity of the schema change task, and level of user expertise. The dependent variables were time needed to conduct impact analysis, correctness of the answers, and subjective user satisfaction.

In the main experiment, the 13 students were divided into two groups with respectively 7 students and 6 students. The assignments to the respective groups were done at random. Both groups were asked to conduct two database schema changes (one subtractive and one additive) on the library database application system borrowed from [37]. This application consists of 25 files stored in 5 directories (about 3600 lines of source code). The first group used the SEMT version operating at the fine granularity, while the second group used the SEMT version that identifies affected parts at the coarse granularity.

Data about the subjects of the experiment and their usage of the technology under study were collected by our logging tool [28]. The main focus of our analysis was to identify trends and significant results on time, correctness and user satisfaction. The data comparisons were conducted using the non-parametric 'distribution free' Mann-Whitney rank test (our data is not normally distributed).

4.2.1 Time

The students were asked to conduct two change tasks in 90 minutes. Since they did not finish the second task on time, we compare time only for the first task, but correctness for both tasks. The median time used for solving the task was 6 minutes shorter (13% of the total mean time) for the fine granularity version of SEMT the for the coarse granularity version ($p = 0.11$).

4.2.2 Correctness

The first change task was to delete a field and identify all impacts of this change. This change had impacts on 27 places in 4 different files. We counted all errors that could lead to compilation or execution problems. The students using the fine granularity version of SEMT made fewer errors than those using the coarse granularity version ($p = 0.04$).

The second change task was to add a functionality to the library system. This change had impacts on 20 places in 4 different files. We counted all errors that could lead to compilation or execution problems. The students using the fine granularity version of SEMT made fewer errors than those using the coarse granularity version did ($p = 0.02$). However, these results should be treated cautiously since most students did not finish this task on time.

4.2.3 User Satisfaction

The students were asked to express their agreement with statements addressing adequacy of the SEMT technology by selecting a number on a seven-point scale. Score 1 was used when the students fully agreed with the positive statements on the usability of SEMT; score 7 when they fully disagreed. The general user satisfaction was relatively high (median 2 for the fine granularity; 3 for the coarse granularity). This is similar with the results of the study [38] where visual representation was preferred for solving information retrieval tasks. There was no statistically significant difference in user satisfaction for the two versions of SEMT ($p = 0.5$).

4.2.4 Threats to Validity

The following possible threats to validity have been identified:

- The time for learning SEMT and the time for conducting the change tasks were relatively short for the practical reasons.
- A learning effect is caused by the students learning SEMT during the experiment. The threat to this study was that the students were more familiar with the functionality of SEMT during the second change task than during the first change task.
- The database application system and schema changes may not be representative in their size and complexity.
- The students who participated in this experiment were not professionals.

To help reduce the first two threats we provided the students with the SEMT documentation and the assistance with use of SEMT. The last two threats are more difficult to handle. Although there may be difference between computer science students and professionals, it has been argued that they are close enough to provide useful results in software engineering experiments [39]. Nevertheless, the hypotheses supported in this experiment should be further tested in more realistic industrial settings

5 Conclusions and Future Work

Ensuring consistency between a database schema and applications during schema evolution is a non-trivial task. Applying knowledge from the software change impact analysis and software visualisation in the area of database schema evolution might be useful. We have developed a model and presented a tool called SEMT for identifying impacts of schema changes on applications in an object-oriented context. SEMT applies the modified Delta-viewfront algorithm on the components of the schema and of the applications, and visualises impacts of database schema changes. Usability of SEMT was evaluated by own judgement on a large real-life application and by a controlled student experiment. The general user satisfaction was relatively high. The students made fewer errors and spent somewhat shorter time to solve the task with the fine granularity version of SEMT compared with the coarse granularity version.

We intend to continue the development of SEMT. Our plans are to provide support for a wider spectrum of schema changes. We also plan to explore patterns of use of SEMT commands. Further, we plan a controlled student experiment (and hopefully with some professionals) to compare SEMT with a commercial tool that present impacts of schema changes as a text. Achieving generality in usability studies is difficult. Evaluations based on experiments and case studies have their limitations. However, we believe that empirical results presented here may increase our understanding of the use of schema evolution technology and may be useful for its further development.

References

[1] D. I. K. Sjøberg, Quantifying Schema Evolution, Information and Software Technology, Vol. 35, No.1, pp. 35-44, 1993.

[2] D. Kung, J. Gao, P. Hsia, F. Wen, Y. Toyoshima, and C. Chen, Change impact identification in object oriented software maintenance, In International Conference on Software Maintenance., Los Alamitos, CA, USA, 1994.

[3] J. Banerjee, W. Kim, H. J. Kim, and H. F. Korth, Semantics and Implementation of Schema Evolution in Object-Oriented Databases, In ACM SIGMOD Conference on the Management of Data, San Francisco, CA, 1987.

[4] G. Barbedette, Schema modifications in the LISPO2 persistent object-oriented language, In ECOOP '91. European Conference on Object Oriented Programming. Proceedings. Springer Verlag, Berlin, Germany, 1991.

[5] R. Zicari and F. Ferrandina, Schema and Database Evolution in Object Database Systems, In Advanced Database Systems, J. Pabst, Ed. San Francisco, Morgan Kaufmann Publishers, inc., 1997.

[6] R. Zikari, A Framework for Schema Updates in an Object-Oriented Database System, In 7th IEEE International Conference on Data Engineering, Kobe, Japan, 1991.

[7] M. Dmitriev and M. Atkinson, Evolutionary Data Conversion in the PJama Persistent Language, In 1st ECOOP Workshop on Object-Oriented Databases, Lisbon, Portugal, 1999.

[8] M. P. Atkinson, M. Dmitriev, C. Hamilton, and T. Printezis, Scalable and Recoverable Implementation of Object Evolution for the PJama Platform, In POS9, 2000.

[9] S. E. Lautemann, An introduction to schema versioning in OODBMS, In Seventh International Workshop on Database and Expert Systems Applications. IEEE Comput. Soc. Press, Los Alamitos, CA, USA, 1996.

[10] S. E. Lautemann, P. Eigner, and C. Wohrle, The COAST project: design and implementation, In Deductive and Object Oriented Databases. 5th International Conference, DOOD '97., 1997.

[11] S. E. Lautemann, A propagation mechanism for populated schema versions, In 13th International Conference on Data Engineering IEEE Comput. Soc. Press, Los Alamitos, CA, 1997.

[12] S. Monk, R. Sommerville, A Model for Versioning of Classes in Object-Oriented Database, In 10th British National Conference on Databases, Aberdeen, Scotland, 1992.

[13] S. Monk, A Model for Schema Evolution in Object-Oriented Database Systems, Ph.D. Thesis,University of Lancaster, 1993.

[14] A. H. Skarra and S. B. Zdonik, Type Evolution in an Object-Oriented Database, In Research Directions in Object-Oriented Programming, MITP, Cambridge, MA, Computer Systems, 1987, pp. 393-415.

[15] R. P. Brazile and S. Dongil, A unifying version model for object-oriented engineering database, Computers in Engineering, 1995.

[16] J. Andany, Leonard, M and Palisser, C, Management of schema evolution in databases, In 17th Int. Conf. on Very Large Databases, Barcelona, Spain, 1991.

[17] M. R. Fornari, L. G. Golendziner, and F. R. Wagner, Schema evolution in the STAR framework, Electronic Design Automation Frameworks, Vol. 4, No. 10,1995.

[18] Z. Bellahsene, View Mechanism for Schema Evolution in Object-Oriented DBMS, In 14th British National Conference on Databases, BNCOD 14, Edinburg, United Kingdom, 1996.

[19] R. Young-Gook and E. A. Rundensteiner, A transparent schema evolution system based on object-oriented view technology, IEEE Transactions on Knowledge and Data Engineering, Vol. 9, No. 4, pp. 600-624, 1997.

[20] J. F. Roddick, A Survey of Schema Versioning Issues for Database-systems, Information and Software Technology, Vol. 37, No.7,pp. 383-393, 1995.

[21] L. Deruelle, M. Bouneffa, G. Goncalves, and J. C. Nicolas, Local and Federated Database Schemas Evolution An Impact Propagation Model, In 10th International Conference DEXA'99 (LNCS Vol.1677), 1999.

[22] F. Ferrandina, T. Meyer, and R. Zicari, Schema Evolution in Object Databases: Measuring the Performance of Immediate and Deferred Updates, In 20th Int. Conf. on Very Large Data Bases, Santiago, Chilie, 1994.

[23] L. Al-Jadir and M. Leonard, Transposed Storage of an Object Database to Reduce the Cost of Schema Changes, In Advances in conceptual modeling/ER'99 (LNCS 1727), Paris, France, 1999.

[24] A. Rashid and P. Sawyer, Evaluation for Evolution: How Well Commercial Systems Do, In 1st ECOOP Workshop on Object-Oriented Databases, Lisbon, Portugal, 1999.

[25] X. Li, A survey of schema evolution in object-oriented databases, In Technology of Object-Oriented Languages and Systems, IEEE Comput., Soc, Los Alamitos, CA, USA, 1999.

[26] M. V. Zelkowitz and D. R. Wallace, Experimental models for validating technology, IEEE Computer, Vol. 31, No. 5, pp. 22-31, May 1998.

[27] B. Kitchenham, S. Linkman, and D. Law, DESMET: a methodology for evaluating software engineering methods and tools, Computing & Control Engineering Journal., Vol. 8, No. 3, pp. 120-126, 1997.

[28] A. Karahasanovic, D. Sjøberg, and M. Jørgensen, Data Collection in Software Engineering Experiments, In Information Resources Management Association International Conference, Toronto, 2001 (to appear).

[29] J. R. Lewis, IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use, International Journal of Human-Computer Interaction, Vol. 7, No. 1, pp. 57- 78, 1995.

[30] A. Karahasanovic and D. Sjøberg, Supporting Database Schema Evolution by Impact Analysis, In Norwegian Conference in Informatics, Trondheim, pp 303-314, 1999.

[31] A. Karahasanovic, SEMT - A Tool for Finding Impacts of Schema Changes, In NWPER'2000 Nordic Workshop on Programming Environment Research. Lillehammer, 2000, pp. 60-75.

[32] S. Warchall, A Theorem on Boolean Matrices, Journal of ACM, Vol. 9, No. 1, pp. 11-12, 1962.

[33] G. Z. Qadah, L. J. Henschen, and J. J. Kim, Efficient algorithms for the instantiated transitive closure queries, IEEE Transactions on Software Engineering., vol. 17, pp. 296-309, 1991.

[34] P. J. Purdom, A Transitive Closure Algorithm, BIT, Vol. 10, pp. 76-94, 1970.

[35] S. L. Smith and J. L. Mosier, Guidelines for Designing User Interface Software, Electronic Systems Division, MITRE Corporation, Bedford, MA Report ESD-TR-86-278, 1986.

[36] M. Werner, daVinci V2.1.x Online Documentation, Universitat Bremen, 1998.

[37] H. Eriksonn, E. and M. Penker, Case Study, In UML Toolkit., John Wiley & Sons, Inc., 1998.

[38] E. Morse and M. Lewis, Evaluating visualisations: using a taxonomy guide, Int. J. Human- Computer Studies, Vol. 53, pp. 637-662, 2000.

[39] W. F. Tichi, Hints for Reviewing Empirical Work in Software Engineering, Empirical Software Engineering, Vol. 5, No. 4, pp. 309-312, 2000.