A NOVEL APPROACH TO SUPPORT CHANGE IMPACT ANALYSIS IN THE MAINTENANCE OF SOFTWARE SYSTEMS

Guenter Pirklbauer
Software Competence Center Hagenberg, Softwarepark 21, A-4232 Hagenberg, Austria

Michael Rappl
Oberoesterreichische Gebietskrankenkasse, Gruberstrae 77, A-4020 Linz, Austria

Keywords: Change Coupling Analysis, Change Impact Analysis, Data Warehouse, Dynamic Dependency Analysis, Software Maintenance.

Abstract: The costs for enhancing and maintaining software systems are up to 75% of the total development costs. It is therefore important to provide appropriate methods, techniques and tools for supporting the maintenance phase of the software life cycle. One major maintenance task is the analysis and validation of change impacts. Existing approaches address change impact analysis, but using them in practice raises specific problems. Tools for change impact analysis must be able to deal with analysis- and design-models which are not compliant with the released software system. These models are not a good basis to perform change impact analysis. The proposed approach combines methods of dynamic dependency analysis and change coupling analysis to detect physical and logical dependencies between software components. The goal is to detect low-level artefacts and dependencies based on only up-to-date and system-conform data, including logfiles, the service repository, the versioning system database and the change management system database. The implementation of the approach supports both the management and developers.

1 INTRODUCTION

The necessity to support the maintenance team with change impact analysis approaches rises, if the software system gets larger, gets connected to web-services via interfaces and is subject to permanent changes - thus: the software system gets more and more complex. To keep complexity within manageable scope, the team can observe the evolution of the software system (Gall and Lanza, 2006) and can continuously check architecture guidelines to countersteer architectural erosion (Lehman et al., 1997). Nevertheless, service support centers that maintain complex software systems need a powerful environment consisting of methods, techniques and tools for change impact analysis. Many approaches and tools try to support change impact analysis in a more or less automatic manner, but the results are not as satisfying as expected (INCOSE, 2005). Most of the approaches require well defined artefacts, established dependencies and at best all stored in a database. This is hardly achievable in real life projects. In practice, the need to perform change impact analysis arises after the software system was made available to customers and the analysis- and design models are not compliant with the released software system anymore. So, implementing change impact approaches or finding and introducing standard tools is often very complex, if not unfeasible. An ideal approach for change impact analysis is required to be fully automatic, cheap to implement and/or introduce, provides a safe and exact range of dependencies and ideally, supplements dependencies with additional metrics to calculate the "strength" or "probability" of relations between artefacts.

2 RELATED WORK

Dynamic Dependency Analysis aims at detection dependencies based on runtime data (Goradia, 1993; Zhao, 1998). The quality level of dynamically detected dependencies is better, because the amount of dependencies are less and as a consequence more pre-

Pirklbauer G. and Rappl M. (2008).
A NOVEL APPROACH TO SUPPORT CHANGE IMPACT ANALYSIS IN THE MAINTENANCE OF SOFTWARE SYSTEMS.
In *Proceedings of the Tenth International Conference on Enterprise Information Systems - DISI*, pages 453-456
DOI: 10.5220/0001692004530456

cise than in static analysis. Static analysis techniques determine all possible dependencies based on static data, e.g. source code. As a result, the range of statically determined dependencies therefore can be enormous and cope little with dependencies which are relevant for change impact analysis. But the major advantage is the fact, that implementing dynamic analysis techniques is cheap due to low software instrumentation effort. A disadvantage is that a lot of runtime data needs to be stored and managed.

Moe et al. developed a method to improve the understanding and development of distributed systems (Moe and Sandahl, 2002). The method is based on operational data and includes three steps: 1. Collecting remote procedure-call during operation, 2. Extracting trace data for statistics and reconstructing call graphs, 3. Visualizing the data.

Law presents a dynamic slicing technique named *"Program Path Profiling"* to identify dependencies between program units (Law, 2005). Here, the software needs to be instrumented to store calls on procedure level.

These approaches are what we interpret as "cheap", because dependencies will be examined by instrumenting the software system. Nevertheless these approaches only concentrate on the determination of dependencies without considering their strength. Moe et al. and Goradia address this issue (Moe and Sandahl, 2002; Goradia, 1993).

Change Coupling Analysis is a subdiscipline in the field of MSR (Mining Software Repositories) and aims at identifying logical couplings between modules, classes and methods. In the research field of software evolution, these logical change couplings are used to identify shortcomings in the architecture of the software system (Gall et al., 2003). In the context of software change prediction and impact analysis, logical change couplings can be used to supplement physical dependencies (Kagdi and Maletic, 2007).

The QCR-approach (Quantitative Analysis, Change Sequence Analysis, Relation Analysis) has been already applied in various case studies (Gall et al., 2003; Gall and Lanza, 2006). The approach was used to learn about the evolution of a software system based on its (change) history.

Kagdi et al. combine single-version and evolutionary dependencies for estimating software changes (Kagdi and Maletic, 2007). This approach is particular interesting, because we also want to combine dependency analysis and MSR-analysis. They hypothesize, that combining dependencies out of classical impact analysis approaches (e.g. dependency analysis) and out of mining software repositories will improve the support of software change prediction.

3 APPROACH

3.1 Approach Overview

The approach assumes, that the combination of both dynamic dependency analysis and change coupling analysis methods result in an overall improvement of change impact analysis. Kagdi et al. investigated a combined approach to support software change prediction (Kagdi and Maletic, 2007). Change prediction is one of the tasks you can do with the results of impact analysis (others will be estimating timetables, estimating trends of failures, etc). But the proposed approach and the approach from Kagdi et al. differs in one main point: Kagdi pursues the paradigm, that fine-grained analysis on the level of source code (i.e. analysing source at syntactic level) is necessary to support change prediction. We strive at increasing the basic set of dependencies (physical and evolutionary) and the precision of these dependencies to avoid analysing software artefacts on source code level.

Figure 1 on page 3 describes the principal approach on a coarse level. As a starting point various data sources have to be examined using dynamic dependency analysis and change coupling analysis. Based on a data warehouse, the framework provides information on two abstraction levels to support both groups of users of change impact analysis - developers and managers.

Based on the text-formulated change requests, *developers* have to identify primary affected artefacts (initial set). This artefacts are components and classes. Then the framework proposes artefacts, which have dependencies to artefacts of the initial set. These artefacts are the impact set. To consider the importance and relevance (impact) of dependencies, the framework calculates the strength of them based on metrics. These metrics are described in subsection 3.2. Then *developers* confirm or reject artefacts of the proposed impact set. This process of proposing and confirming or rejecting will take place in several iterations.

Project managers have to be supported in creating timetables, allocating human resources and estimating the risk exposure. Based on the risk exposure, the framework can support *quality managers* in allocating quality assurance activities, e.g. on which components to concentrate testing resources. The framework has to determine the quality level of the next

release of the software system, or - at least - be able to recognize trends.

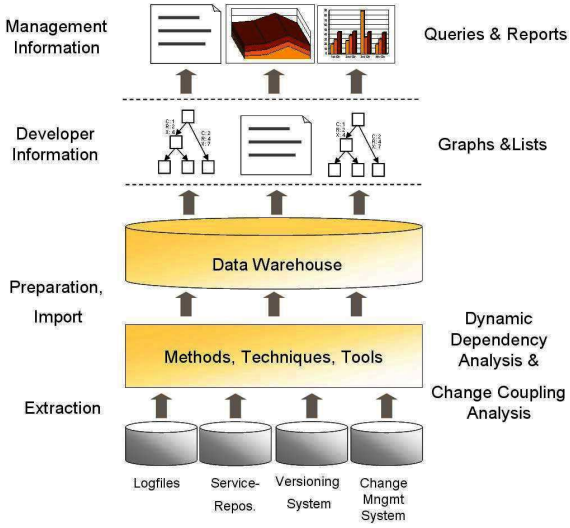

Figure 1: Framework Architecture.

3.2 Data Sources

The determination of artefacts and dependencies has to be based on up-to-date and system-conform data. Following data sources have to be exploited:

- Logfiles: Footprints in logfiles enable the documentation of calls on the level of components, classes and methods. This footprint can be used to determine physical dependencies between artefacts just mentioned. Data to exploit: Artefacts, physical dependencies. Metrics: Number of calls on different levels, number of warnings and errors at runtime in the live-system, duration of calls in milliseconds.
- Service Repository: Often, service repositories are in use to administrate services including related definitions of request-, response-messages and data-records. These kind of data structures are important center points between services and indicators for dependencies. Data to exploit: Artefacts, physical dependencies - especially data-based dependencies. Metrics: Lines-of-Code (LOC)
- Versioning System: Entries in the versioning system are a good basis to determine logical couplings. With the utilization of various association-rules, the framework will be able to find dependencies between files which are not based on the source code. These rules comprise association by transaction-matching (all files checked-in by a developer at once), bug-ID-matching, comment-matching and special time-frame-based algorithm. Additionally, the number of changed files due to these association rules can be determined to calculate the strength of dependencies. Further, the error- and change-rate can be calculated on the basis of classes, but also can be aggregated to package or subsystem level. Data to exploit: Logical dependencies, Metrics: Strength of dependencies, error- and change-rate of artefacts.
- Change Management System: Change requests in the change management system should be correlated to entries in the versioning system. At best, the developer who is about to do the check-in, will be caused to insert the unique number of the change request. If not, techniques are necessary to correlate these entries automatically, e.g. using techniques as proposed by (Canfora and Cerulo, 2005; Zimmermann and Weigerber, 2004). Data to exploit: All relevant data from change requests (priority, severity, efforts, detailed description, etc.). Metrics: Error- and change-rate.

3.3 Dependency Types

The framework must be able to detect and manage low-level-artefacts and -dependencies, e.g. components, classes and methods. Therefore, two types of dependencies have to be considered: (1) *Physical dependencies* which result from relations based on the source code. The implementation of dependency analysis and detection techniques should be based on dynamic dependency analysis techniques with an emphasis on logfile analysis. (2) *Logical dependencies* which result from transitive relations. If dependencies are not based on physical relations via the source code, two possible reasons have to be considered: Transitive dependencies across business process objects, and transitive dependencies across database objects. Here, dependencies have to be examined with heuristics (association-rules) and data mining techniques.

4 RESEARCH DESIGN

The principal idea of this work is to use a combination of dynamic dependency analysis and change coupling analysis methods in order to improve change impact analysis. The research hypothesis underlying this project is, that the combination of methods will improve the support of change impact analysis. To verify the research hypothesis, research questions

Table 1: Project Phases and Research Methods

Project Phase	Research Method
Identification and Analysis of State-of-the-Art	Literature Analysis
Definition of Approach	Conceptual Construction
Analysis of Change Impact Analysis Process	Empirical Case Study
Definition and Implementation of Methods, Techniques and Tools	Construction, Prototyping, Empirical Evaluation (Experiment, Test)
Validation of Framework	Empirical Validation (Experiment, Test, Survey)

have to be established and answered. To validate results in practice, empirical research methods will be used in the research project. But also constructive and conceptual methods will take place. To see which methods will be used in which project phase, see Table 1 on page 4.

Especially in order to verify the research hypothesis, well established quality parameters *precision* and *recall* will be used. The central point is the combination of methods, this means we have to compare and analyse dependencies determined via dynamic dependency analysis or change coupling analysis techniques. We have to conduct several empirical experiments to show, how the combination of techniques can help project managers, quality managers and developers to perform change impact analysis.

5 CONCLUSIONS AND OUTLOOK

In this paper we have outlined some facts of our approach to support change impact analysis. Currently we are implementing first prototypes to examine logfiles and detect logical change couplings in the versioning system. Especially we investigate causes and types of logical couplings. In order to explore the change impact analysis process in detail, empirical case studies within our industrial partners will be carried out.

REFERENCES

Canfora, G. and Cerulo, L. (2005). Impact analysis by mining software and change request repositories. In *METRICS '05: Proceedings of the 11th IEEE International Software Metrics Symposium (METRICS'05)*, page 29, Washington, DC, USA. IEEE Computer Society.

Gall, H. C., Jazayeri, M., and Krajewski, J. (2003). CVS release history data for detecting logical couplings. In *Proceedings of the International Workshop on Principles of Software Evolution*, pages 13–23, Helsinki, Finland. IEEE Computer Society Press.

Gall, H. C. and Lanza, M. (2006). Software Evolution: Analysis and Visualization. In *ICSE '06: Proceeding of the 28th international conference on Software engineering*, pages 1055–1056, New York, NY, USA. ACM Press.

Goradia, T. (1993). Dynamic impact analysis: A cost-effective technique to enforce error-propagation. In *ISSTA '93: Proceedings of the 1993 ACM SIGSOFT international symposium on Software testing and analysis*, pages 171–181, New York, NY, USA. ACM Press.

INCOSE (2005). Incose requirements management tool survey, http://www.paper-review.com/tools/rms/read.php.

Kagdi, H. and Maletic, J. I. (2007). Combining single-version and evolutionary dependencies for software-change prediction. In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 17, Washington, DC, USA. IEEE Computer Society.

Law, J. (2005). *Path-Based Dynamic Impact Analysis*. Phd-thesis, Oregon State University - School of Electrical Engineering and Computer Science.

Lehman, M. M., Ramil, J. F., Wernick, P. D., Perry, D. E., and Turski, W. M. (1997). Metrics and laws of software evolution - the nineties view. In *METRICS '97: Proceedings of the 4th International Symposium on Software Metrics*, page 20, Washington, DC, USA. IEEE Computer Society.

Moe, J. and Sandahl, K. (2002). Using execution trace data to improve distributed systems. In *ICSM '02: International Conference on Software Maintenance*, pages 640–648.

Zhao, J. (1998). Dynamic slicing of object-oriented programs. Technical report.

Zimmermann, T. and Weigerber, P. (2004). Preprocessing cvs data for fine-grained analysis. In *Proceedings of the First International Workshop on Mining Software Repositories*, pages 2–6.