Design Patterns in Performance Prediction

A. Inkeri Verkamo, Juha Gustafsson, Lilli Nenonen, and Jukka Paakki
Department of Computer Science
University of Helsinki
Finland
{verkamo, gustafss, lnenonen, paakki} @cs.helsinki.fi

ABSTRACT

The use of design patterns in object-oriented software development is recommended to improve both the software structure and the development process. On the other hand, their effect on the performance of the final product may greatly depend on the particular application and the projected workload. In addition to intentional use, the design may also contain patterns (and anti-patterns) that have not been used on purpose. We propose an approach where all patterns and anti-patterns embedded in the design are discovered by a mining tool and their performance effects are evaluated using previous experience stored in a pattern library.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*performance measures*; D.2.11 [**Software Engineering**]: Software Architectures—*patterns*

1. INTRODUCTION

Inadequate performance is among the leading reasons why fully implemented and functional systems are sometimes discarded by the users. The users of interactive systems have little tolerance for slow response, and for a hard real-time system a delayed response is considered no better than system failure. Hence it should be natural to consider performance aspects early in the development life-cycle.

However, the traditional solution is to postpone all performance considerations until the product, or at least the performance critical part, has been implemented [6]. Unfortunately, not all performance problems can be solved by tuning the software or upgrading the hardware, and this approach does not address the actual reason for deficient performance. It may even turn out to be extremely expensive by resulting in massive rework of the implementation, and in delays of product delivery.

Software Performance Engineering (SPE) emphasizes taking performance into consideration early in the software process by careful definition of the product performance requirements and designing for responsiveness [6]. Using this strategy, a performance model of the software is provided in the architectural design phase, and refined models covering the performance critical parts are developed at a pace with more detailed design. Possible performance problems will be found so early that improving the design does not require discarding already implemented parts of the software.

SPE asks for a careful and timely study of performance aspects, which may seem too laborious when developing products with no hard real-time performance requirements. Since performance modeling tools are seldom embedded in CASE toolsets, an average software developer may not have the necessary expertise. Moreover, the architectural design of the product may be either too coarse or too incomplete to serve as the basis of orderly performance prediction. There is demand for tools that do not require stating the performance needs early in the development cycle, but can still give, at a later phase, some level of confidence on the performance of the product.

For the developer of a product with soft performance requirements, the most natural phase of performance prediction is in the detailed design phase. When the architecture of the software has been decided, support may be needed in selecting components to be used as building blocks of the software. In particular, when choosing between the reuse of an old approved solution and some suggested improvement, the developer will want to know the performance effects of each choice.

This approach is particularly attractive in object-oriented software development, where the modularity of the software promotes reusing previously developed design ideas. The prevalent use of a common representation language for the designs, the *Unified Modeling Language (UML)*, has enabled active collection of reusable design solutions. High quality design solutions to frequently occurring software problems, generally known as *design patterns*, have been described and stored in pattern libraries [3]. Correspondingly, bad solutions, or *anti-patterns* have been detected and can be stored together with a description of their disadvantages [2].

In this paper we address the question of performance prediction in the design phase of object-oriented software development, and the possibilities that the existence of design patterns offers for performance modeling. The work is part of an ongoing research project where various quality issues, including performance, are evaluated and predicted based on the patterns recovered in the software architecture.

2. PATTERNS AND PERFORMANCE

Using design patterns is mainly advocated because of their positive effect on the software structure and their potential to speed up the design process and improve its quality. But the advantages of the patterns can also be related to various quality factors of the product, and such information could also be recorded in the pattern library. Moreover, some of the recurring patterns are not necessarily good or bad per se, as their quality effects may depend heavily on the situation where they are used. Therefore the library should be augmented by recording user experience on previous successful or unsuccessful use of the patterns, to improve the common understanding of their applicability.

For performance considerations this is particularly prominent. Williams and Smith give an example where applying a well chosen design pattern caused only minor performance improvement, since under the given conditions the improved part of the design was not a performance bottleneck [7]. More generally, a standard solution may have positive or negative performance effects depending on the characteristics of the application area and the intended workload. The difference between a "good" pattern and a "bad" one may be in the parameters of the system, and storing this type of information in the pattern library can aid the designer in predicting the performance of the software.

Not all pattern usage is preplanned [4]. In fact, many of the good practices and patterns of software design have been created long ago and they may frequently recur in software products without the developers being aware that their design solution is an identified pattern at all. Even more probably, the occurrence of an anti-pattern may be unknown to the developer. It is therefore useful to detect all the recurring patterns in the design regardless of their origin, and to apply the information in the pattern library to predict the performance effects of the detected patterns.

3. THE MAISA APPROACH

Maisa (Metrics for Analysis and Improvement of Software Architectures) is an ongoing research project for developing methods and tools to automatically analyze the software architecture and predict some central quality factors [5]. Among the quality factors that are examined, size, complexity and performance are the most important. The strategies used for quality prediction rely on two basic ideas: pattern discovery and hierarchical measures.

The architecture analysis in Maisa is based on discovering patterns from the UML diagrams describing the software architecture. Instead of relying on the developer to provide information of the patterns in the architecture, we aim to discover patterns and anti-patterns directly from the software design. The pattern mining method used in the Maisa project is described in [5]. To evaluate the performance effects of the embedded patterns, we use additional information recorded in the pattern library. This includes the necessary data for predicting performance effects of each pattern, and historical information about their usage and relevance.

Typically only part of a UML diagram is included in the pattern, while performance computations must of course cover the entire system. Appplying the ideas of SPE [6], the performance measures of the entire system are computed hierarchically from the measures of its subsystems using information about the looping and branching structure. In a multiprocess situation also contention, communication delays, etc. must be taken into account. The input of the Maisa tool is the presentation of the software as a collection of diagrams representing different views of the software. Hence performance evaluation may require combining data from several diagrams of different kinds. Instead of single measures, performance characterization typically involves presenting a set of numbers. Typically the developer wants to predict at least the worst case and best case behavior of the system, and also evaluate one or several typical cases. The Maisa tool will allow interactive use, where the user gives parameter values for each of the interesting cases, and the tool computes the required predictions.

To model performance with UML we may need to use several types of diagrams to represent the different views of the software with different granularity. In their standard form, none of the UML diagrams contain the information required for performance predictions, but by structured use of annotations, such as stereotypes, tagged values and constraints [1], the formalisms may be enhanced for this purpose. In the Maisa project, we use activity diagrams as the basis of our performance model, and base our performance calculations directly on the extensions included in the diagrams.

At present, the Maisa tool produces only simple metrics for software size and complexity. Our next task is to implement the pattern discovery algorithm and the pattern library. The first version of the library will contain previously published patterns [3], but later the library can be augmented with specific patterns of the user community and their own experience of pattern usage. We are confident that more such patterns will emerge as the users gather experience of the gains of including their specific patterns in the library.

4. REFERENCES

[1] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1998.

[2] W.J. Brown, R.C. Malveau, W.H. Brown, H.W. III McCormick, and T.J. Mowbray. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley and Sons, 1998.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[4] R. Martin. Discovering patterns in existing applications. In J.O. Coplien and D.C. Schmidt, editors, *Pattern Languages of Program Design*, pages 365–393. Addison-Wesley, 1995.

[5] J. Paakki, A. Karhinen, J. Gustafsson, L. Nenonen, and A.I. Verkamo. Software metrics by architectural pattern mining. In *Proc. of the Int. Conf. on Software: Theory and Practice (16th IFIP WCC)*, August 2000.

[6] C.U. Smith. *Performance Engineering of Software Systems*. Addison-Wesley, 1990.

[7] L.G. Williams and C.U. Smith. Performance evaluation of software architectures. In *Proc. of the First Int. Workshop on Software and Performance (WOSP'98)*, pages 164–177, October 1998.