Observations on the Evolution of an Industrial OO Framework

by

Michael Mattson, Jan Bosch

Department of
Software Engineering and Computer Science
University of Karlskrona/Ronneby
S-372 25 Ronneby
Sweden

ISSN 1103-1581
ISRN HK/R-RES—99/7—SE

Observations on the Evolution of an Industrial OO Framework

by Michael Mattsson, Jan Bosch

ISSN 1103-1581

ISRN HK/R-RES—99/7—SE

Copyright © 1999 by Michael Mattsson, Jan Bosch

All rights reserved

Printed by Psilander Grafiska, Karlskrona 1999

Observations on the Evolution of an Industrial OO Framework

Michael Mattsson & Jan Bosch

University of Karlskrona/Ronneby

Department of Software Engineering and Computer Science

S-372 25 Ronneby, Sweden

[Michael.Mattsson | Jan.Bosch]@ipd.hk-r.se

<http://www.ipd.hk-r.se/~michaelm/~bosch>

ABSTRACT

Recently an approach for identifying potential modules for restructuring in large software systems using product release history was presented[4]. In this study we have adapted the original approach to better suit object-oriented frameworks and applied it to an industrial black-box framework product in the telecommunication domain.

Our study shows that using historical information as a way of identifying structural shortcomings in an object-oriented system is viable and useful. The study thereby strengthens the suggested approach in [4] and demonstrates that the approach is adaptable and useful for object-oriented systems. The usefulness of the original approach has been validated through this study too.

Keywords

Object-oriented frameworks, Software evolution, Object-oriented framework evolution, Experience report

1 INTRODUCTION

In recent years, object-oriented framework technology has become common technology in object-oriented software development [1, 2, 5, 9, 10]. Frameworks allow for large-scale reuse and studies show that the use of framework technology reduces development effort, e.g. [10].

It is argued that designing a good framework requires several iterations [8, 6, 11]. Of special importance in the development of a framework is the identification of the stable and variable parts of the design. The stable part of the design represents a reusable core of application domain functionality, which is reused for all applications. The stable part provides hooks that are intentionally designed to provide for customising or tailoring of the framework (i.e. the variable parts).

As with all software, frameworks evolve to meet new demands from customers/framework users. To alleviate or remedy the problem of complex software and the difficulties of evolving it, it is important to identify modules that are likely candidates for restructuring. This is of importance in framework evolution since there are large amounts of effort invested in the design of the framework and information about where likely changes occur may reduce the cost of redesign the framework to meet the new demands. Studies show that software engineers can only identify a subset (<50

percent) of future changes and they can not provide the complete picture of change [7]. Thus, a more objective method for identifying structural shortcomings will provide more reliable input for maintenance effort estimations and thereby more accurate estimates.

Recently an approach for identifying potential modules for restructuring in large software systems (about 10 millions lines of code) has been proposed by Gall et al. using product release history [4]. We have adapted the approach to better suit object-oriented frameworks. Instead of programs we are investigating classes as the observed entities.

In this paper, we describe the evolution of an industrial black-box framework product, the Mediation framework, in the telecommunication domain. The investigation covers four consecutive versions of the framework that were developed over a period of six years. Over 30 instantiations of the framework have been delivered to customers.

The new versions of the framework were forced by product improvements, e.g. better performance, and new customer demands. Our analysis is based on information about modules and classes (the class declarations only) of the framework. The constant development of the framework has increased both the size and the complexity. The size of the Mediation framework has increased from 322 classes to 598 classes.

The objective of our study is to evaluate the approach to identifying potential structural shortcomings of the framework using the aforementioned framework as a case. The approach uses historical data, as in the case by Gall et al. [4]. Once the eventual shortcomings have been identified, the relevant subsystems or modules can be subject to restructuring. As an effect of our adapted approach we are also validating the usefulness of the original approach by Gall et al.

The paper is organized as follows: In section 2, the Mediation framework is described in sufficient detail to get an understanding of what kind of software system is the subject of the case study. The subsystems and modules have been renamed to make the discussion clearer and avoid bothering the reader with unnecessary details. Section 3 describes the evolution observations on the complete Mediation framework. In section 4, the evolution

observations of the subsystems and their modules systems is described. Section 5 discusses related work and the paper is concluded in section 6.

2 THE CASE STUDY

The method

The method used in our study is an adaptation of the approach proposed by Gall et al. [4]. The focus for the study by Gall et al. was on macro-level software evolution by tracking the release history of a system. Their system comprises about 10 million LOC and was structurally divided into a number of subsystems where each subsystem was further divided into modules and each module consisted of a number of programs. Based on historical information about the programs, subsystems etc. they investigated the change rate, growth rate and size of the software (and its parts) using the *version numbering* of the programs as the units for 20 releases of the software systems. The method steps in the approach are

1. calculate, for all releases, the change and growth rate for the whole system,
2. calculate, for all releases, the change and growth rate for each of the subsystems,
3. for those subsystems that exhibit high growth and change rates calculate, for all releases, the change and growth rates for the modules,
4. those modules that exhibit high change and growth rates are identified as likely candidates for restructuring.

In our study we have adapted the approach to better suit for smaller systems (about 600 classes). We use the same decomposition of the system into subsystems and modules. Regarding the modules these are in our study composed of classes (not programs). So our calculations of change and growth rates are made in terms of classes as the units. Changed classes are identified by a *change in the number of public operations* in the class interface. This since a change in the public interface represents an addition or better understanding of the functionality that the class has to offer the rest of the system. Changes to the private part of the class interface are mostly related to perfective changes such as performance improvements and as such less interesting in this study. Regarding the method steps we use the same steps as the approach by Gall et al.

Essentially, the adaptation is the change of programs to classes as the observed unit to make the approach useful for object-oriented software systems.

The software system

The subject for the study consists of four successive versions of a proprietary black-box application framework. This framework encompasses the Billing Gateway product developed by Ericsson Software Technology. The framework provides functionality for mediation between network elements, i.e., telecommunication switches, and billing systems (or other administrative post-processing systems) for mobile telecommunication. Essentially, the mediation framework collects call information from mobile switches (NEs), processes the call information and

distributes it to among others, billing processing systems. Figure 1 presents the context of the mediation framework graphically. The driving quality requirements in this domain are reliability, availability, portability, efficiency (in the call information processing flow) and maintainability. The framework is black-box, i.e., the framework provides a number of pre-defined objects for application configuration. The four versions of the mediation framework have been developed during the last six years.

Figure 1: The Mediation Framework Context

The structure of the Mediation framework system is a traditional tree structure with four levels; the system, subsystem, module and class level. Each level consists of one or more elements and each element on a specific level is connected to one element on a higher level. The logical structure of the Mediation framework has been defined during the evolution of the four versions of the framework. The time spans between the versions are approximately 18 months each.

3 FRAMEWORK EVOLUTION OBSERVATIONS

Based on the structural information derived from the four framework versions we are able to make statements about the Mediation framework's evolution. For example, the size of the framework and its subsystems and the number of classes changed in a particular module from one version to the next.

Using the extracted information we try to identify potential shortcomings in the current structure of the Mediation framework.

We have extracted information, about the following subset of system properties, which is an adaptation of the descriptions found in [4]:

- The *size* of each system, subsystem or module is defined as the number of classes it contains. We use the notion of classes as the size counting entity instead of measuring

the size of the source code. This since it is easier to collect necessary information and measure on the detailed design level compared to the source code level.

- The *change rate* is the percentage of classes in a particular system, subsystem or module that changed from one version to the next. To compute the change rate, two versions are required for comparison. The relative number of the changed classes represents the change rate.
- The *growth rate* is defined as the percentage of classes in a particular system, subsystem or module, which have been added (or deleted) from one version to the next. To compute the growth rate, two versions are compared and the numbers of the added and deleted classes are computed. The relative number of the new classes (i.e. the difference between added and removed classes) represent the growth rate.

Using the information gathered from the Mediation framework we can make statements about framework evolution observations related to: 1) the whole Mediation framework system, 2) the subsystems of the Mediation framework; and 3) the modules of particular subsystems that we chose for further study.

System Observations

Diagram 1 shows the historical development of the size of the Mediation framework. The framework has a high growth rate; it consisted of 322 classes initially and 598 classes at version 4.0. Thus, the total size of the Mediation framework increased by over 85 percent over the four versions. From the diagram we also see that the size of the Mediation framework is growing in a way similar to linear.

Diagram 1:
Evolution of the size of the Mediation framework

Diagram 2 shows the number of classes that were added in each version (i.e. added minus removed classes). For example, the diagram shows that at version 2.0 more than 120 classes were added to the system. A general observation is that the *number of added classes is decreasing* with newer versions of the framework. This can be taken as an indication that the framework is becoming more mature, i.e. achieving a structural stability.

Diagram 3 compares the change and growth rates for each version of the Mediation framework. In general, the change

Diagram 2: Number of added classes in each version

rate is higher than the growth rate. The growth rate is decreasing with successive versions of the framework, from about 40 percent at version 2.0 down to about 15 percent at version 4.0. This *decrease in growth rate* is another indication of the framework's maturity.

Diagram 3:
Comparison of the change and growth rates

The change rate is increasing at version 4.0 and is not following the trend for version 2.0 and 3.0. One reason for this may be an *architectural change*, i.e. a change affecting entities throughout the whole system. In the Mediation framework the architectural change was the introduction of hot billing.

The change rates are around 40-50 percent in between all versions. In version 2.0 and 3.0 this may be a sign of better domain understanding but in version 4.0 the major reason is the architectural change introduced by the hot billing requirement.

A summary of the findings at the system level is:

- The size of the Mediation framework is increasing linearly.
- The development of the whole Mediation framework becomes stable with respect to the growth rate that is decreasing over the versions.
- The change rate is about 40-50 percent for all versions. The change rate shows a decrease to version 3.0 but

increases again in version 4.0. The increase of the change rate in version 4.0 is an indication that some unexpected evolution has taken place, i.e. evolution with architectural impact.

The latter two observations are signs of structural stability of the Mediation framework eventually with a small exception for the increase of the change rate in version 4.0.

The observations on the system levels indicate that the framework evolves in a satisfactory fashion. This since the growth rate is decreasing and the change rate is at an acceptable level (<50 percent). However, the observations are only valid on system level and it may be the case that the subsystems do not behave in the same way. In the next section we will characterize the evolution of the four subsystems for the Mediation framework.

4 THE EVOLUTION OF SUBSYSTEMS

Up to now we have achieved an impression of how the overall system has evolved over time. In this section, we will examine the subsystems further to see if the subsystems evolve the same way as the whole system.

Table 1: Change and growth rates of subsystems

<i>Subsystem</i>	<i>Change rate</i>	<i>Growth rate</i>	<i>Relative Size</i>
A	51%	191%	34%
B	64%	153%	24%
C	92%	191%	37%
D	75%	700%	5%

In table 1, we see the change and growth rates for the four subsystems in the Mediation framework. Subsystem D exhibits the highest growth rate, 700 percent, but since subsystem D only comprises 5 percent of the total system size in version 4.0 the subsystem is of minor interest. Subsystems A and C have a growth rate on 191 percent each. Their relative sizes of the whole system are 34 percent for subsystem A and 37 percent for subsystem C, respectively. Thus, both are candidates for further study but since Subsystem A represent the GUI part of the system and comprises, to a very high degree, of automatic generated UI classes it is not a representative subsystem.

If we consider the change rate it is obvious that subsystem C has undergone major changes between version 1.0 to version 4.0 since it has a change rate of 92 percent. It may also be of interest to investigate subsystem B since its change rate, 64 percent, is higher than the change rates for the whole system (diagram 3). Subsystem C represents a major part of the system, 37 percent of the classes, and we will investigate it further on the module level in the following subsection.

Concluding, in the following section we will further investigate the evolution characteristics for Subsystem C due to its high growth and change rates and Subsystem B due to its relatively high change rate.

Change and growth rates of subsystems

Diagram 4:
Change and growth rates of subsystems

The evolution of subsystem C

Subsystem C exhibits the highest growth and change rates among the subsystems. These facts make the subsystem a likely candidate for restructuring and/or reengineering activities. We will investigate how the modules in subsystem C evolves and whether they possess the same evolution characteristics as the whole subsystem.

Subsystem C consists of 5 modules named module C1 to C5. Their relative size of Subsystem C is presented in diagram 5 .We see that all five modules comprise between 10-30 percent of the subsystem. Modules C3 and C4 has a relative size larger than 25 percent whereas as modules C1, C2 and C5 all have a relative size below 15 percent. In fact, module C3 and C4 are two of the three largest modules of the Mediation framework.

Relative Module Sizes in Subsystem C

Diagram 5:
The relative size of modules in Subsystem C

Diagram 6 , which shows the number of classes for each module in Subsystem C confirms that module C3 and C4 are large modules and that they both have had a large increase of the number of added classes. In addition modules C1, C2 and C5 has been relatively stable in size but a minor increase can be observed for module C1.

Diagram 6:
Sizes of modules in Subsystem C.

Diagram 7 shows the growth and change rates for the modules in subsystem C from version 1.0 to version 4.0. Module C5 has a growth rate of 460 percent that is extremely high. Also modules C1 and C4 has high growth rates, 233 and 306 percent respectively.

Module C5 has the highest growth rate but is the smallest module in the subsystem. Also Module C4 has a high growth rate and it represents 25 percent of the subsystem. Through discussions with the software engineers it was clarified that the reason for the high growth rate is that the module is responsible for coding and decoding the Call Data Records which exist in a number of formats. Thus, the high growth rate is the result of an increased number of formats supported by the Mediation framework, e.g. module C4 handles a number of different Call Data Record formats which results in a set of classes whose objects are used as parameters in the framework. I.e. a typical black-box framework evolution observation.

All modules except Module C5 exhibit high change rates. A reason for the relative low change rate for Module C5 in the subsystem is that it captures the encapsulation of some other software such as database systems.

Diagram 7:
Change and growth rates of Subsystem C's modules

Regarding the change rates for the modules in Subsystem C we see in diagram 8 that they behave as the system in whole,

i.e. the change rate is increasing from version 3.0 to version 4.0. Modules C3 and C4 exhibit the highest change rates throughout all the versions.

Diagram 8:
Change rates of modules in subsystem C

Concerning the high change rate, all modules in subsystem C have high change rates and the main reason for this is two-fold;

1. a design concept called Node has been divided into two concepts, Childnode and ParentNode, this to achieve higher maintainability and lower complexity,
2. another design concept called Point has changed from being file-oriented to be transaction-oriented due to the need of providing decreased latency for the Call Data Record (the billing information input) processing. This to fulfil the requirement of hot billing.

To summarize the findings of Subsystem C we can foresee changes in all modules due to the changes of the two design concepts. These changes have presumably not achieved full effect on the system design. Regarding Module C4 and its growth rate; this is viewed as an anticipated kind of evolution due to the nature of black-box frameworks.

The evolution of Subsystem B

We present a few observations of the Subsystem B evolution characteristics since the subsystem showed a relatively high change rate.

In diagram 9 , we see the change and growth rates for the modules in Subsystem B. We see that module B4 has the highest growth rate, 242 percent, and that module B2 has the highest change rate, 73 percent. Module B2 is in fact the largest module in Subsystem B representing 68 percent of the classes in the subsystem.

The high growth and change rates of Module B2 are explained by the development from a simple arithmetic language to a mini-language for handling data units (a design concept based on ASN.1 descriptions).

Module B4 has the highest growth rate but it is relatively small, 30 classes, and the increase can be explained by an increased number of supported operating system thread representations (i.e. a typical black-box framework evolution observation similar to the one described earlier).

Diagram 9: Change and growth rates of Subsystem B's modules

To summarize subsystem B, we have again an anticipated kind of evolution for Module B4. We may foresee changes in Module B2 or modules that are using Module B2 depending on the future evolution of the mini-language.

Through the examination of the four subsystems, a different picture evolves since the subsystems do not exhibit the same behaviour as the framework as a whole. Especially subsystem C and B are likely candidates for forthcoming restructuring and/or re-engineering activities.

5 RELATED WORK

Related work includes, of course, the work by Gall et al. [4] where historical information about modules and programs is used as a means to identify structural shortcomings of a large telecommunication switching system. Our work uses a similar approach but on a smaller system, 300-600 classes, and we use historical information about modules and classes, which are entities of smaller granularity than in the work Gall et al. Recent work by Lindvall and Sandahl [7] shows that software developers are not so good at predicting from a requirements specification how many and which classes that will be changed. Their empirical study shows that only between 30-40 percent of the actual changed classes were predicted to be changed by the developers. This can be seen as an argument for a more objective approach for identifying change-prone and evolution-prone parts of a software system.

Other research related to framework evolution is the work by Roberts and Johnson [11] and by Erni and Lewerentz [3]. Roberts and Johnson present a pattern language that describes typical steps for the evolution of an object-oriented framework. Common steps in the evolution are development of a white-box framework (extensive use of the inheritance mechanism), the transition to a black-box framework (extensive use of composition), development of a library of pluggable objects etc. These steps give a coarse-grained description of a framework's evolution. However, they do not explicitly describe where and how the framework evolves when it has reached a certain state.

Erni and Lewerentz describe a metrics-based approach that supports incremental development of a framework. By measuring a set of metrics (which requires full source code access) an assessment of the design quality of the framework

can be performed. If the metric data is within acceptable limits, the framework is considered to be good enough from a metrics perspective otherwise another design iteration has to be done. The approach is intended to be used during the design iterations, before a framework is released and does not consider long term evolution of a framework.

6 CONCLUSIONS

We have studied the structure of four consecutive versions of an object-oriented black-box framework in the telecommunication domain. Based on historical information about the classes and their public interfaces we have applied an adapted version of the approach in [4] in order to identify structural shortcomings of the framework's structure by measuring the growth and change rates of the system.

Our study exhibits results similar to the study by Gall et al., i.e. there is a difference in the behaviour of the whole system versus its subsystems. A relatively stable evolution over time was observed at the system level, but detailed studies of the subsystems showed that subsystem C and B exhibit different characteristics regarding the change and growth rates when compared to the whole system. These major differences would not been identified only by an analysis of the whole system. Two modules were identified as potential subjects for redesign, i.e. module C4 and B2. In addition, to a lesser extent, module C3 is a likely candidate for redesign.

Both our study and the one by Gall et al. show that using historical information as a way of identifying structural shortcomings in a software system is viable and useful. In addition, the approach suggested in [4] is possible to use for object-oriented systems on design level using class interfaces and changes to these interfaces as the observed units. In other words, the approach by Gall et al. is possible to scale down to object-oriented systems.

The validity of the approach of using historical is strengthened by the following. First, our results are similar to the ones by Gall et al., i.e. there exists a difference in the behaviour of the whole system versus its subsystems. Second, we presented the results of this study to the software development organization owning the studied framework. They agreed to our observations, found the results interesting and consider using the approach in the future.

ACKNOWLEDGEMENTS

We would like to thank Ericsson Software Technology for providing us access to the Mediation framework, in particular Mikael Roos and Drzen Milicic.

REFERENCES

1. G. Andert, 'Object Frameworks in the Taligent OS,' *Proceedings of Compcon 94*, IEEE CS Press, Los Alamitos, California, 1994.
2. J. Bosch, P. Molin, M. Mattsson, PO Bengtsson. 'Object-Oriented Frameworks - Problems & Experiences', In *Object-Oriented Application Frameworks*, M. E. Fayad, D. C. Schmidt, R. E. Johnson (eds.), Wiley & Sons, 1998. (forthcoming)
3. K. Erni, C. Lewerentz, 'Applying design metrics to

- object-oriented frameworks,' *Proceedings of the Metrics Symposium*, 1996.
4. H. Gall, M. Jazayeri, R. G. Klösch, G. Trausmuth, 'Software Evolution Observations Based on Product Release History', *Proceedings of the Conference on Software Maintenance - 1997*, 1997, pp. 160-166.
 5. H. Huni, R. Johnson, R. Engel, 'A Framework for Network Protocol Software,' *Proceedings of the 10th Conference on Object-Oriented Programming Systems, Languages and Applications*, Austin, USA, 1995.
 6. R.E. Johnson, V.F. Russo, 'Reusing Object-Oriented Design,' *Technical Report UIUCDCS 91-1696*, University of Illinois, 1991.
 7. Lindvall, M. and Sandahl, K. (1998). How Well do Experienced Software Developers Predict Software Change? *Journal of Systems and Software*, 43(1):19 – 27.
 8. M. Mattsson, 'Object-Oriented Frameworks - A survey of methodological issues,' *Licentiate Thesis*, LU-CS-TR: 96-167, Department of Computer Science, Lund University, 1996.
 9. Peter Molin and Lennart Ohlsson, 'Points & Deviations - A pattern language for fire alarm systems,' *Proceedings of the 3rd International Conference on Pattern Languages for Programming*, Paper 6.6, Monticello IL, USA, September 1996.
 10. S. Moser, O. Nierstrasz, 'The Effect of Object-Oriented Frameworks on Developer Productivity,' *IEEE Computer*, pp. 45-51, September 1996.
 11. D. Roberts, R. Johnson, 'Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks,' *Proceedings of the Third Conference on Pattern Languages and Programming*, Montecillio, Illinois, 1996.