Research Report 6/99

Assessing Object-Oriented Application Framework Maturity - A Replicated Case Study

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—99/6—SE

Assessing Object-Oriented Application Framework Maturity - A Replicated Case Study

by Michael Mattsson, Jan Bosch

ISSN 1103-1581
ISRN HK/R-RES—99/6—SE

Printed by Psilander Grafiska, Karlskrona 1999

Assessing Object-Oriented Application Framework Maturity - A Replicated Case Study

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
Object-oriented application frameworks present one of the most successful approaches to developing reusable assets in industry, but developing frameworks is both difficult and expensive. Framework generally evolve to maturity through a number of iterations due to the incorporation of new requirements and better domain understanding. Since changes to frameworks have a large impact due to the effects on the applications build based on the asset, it is important to assess the maturity of a framework. Bansiya [3, 4] presents an approach to assessing framework maturity based on a set of design metrics and formulates four statements. In this paper, we present the results of a replicated case study of the framework maturity assessment approach. Our study subject consists of four successive versions of a proprietary black-box application framework. Our findings partly support the statements formulated in the original study, but differ in some places. The differences are discussed and explanations and argumentation provided.

Keywords
Object-oriented frameworks, framework maturity, software reuse

1 INTRODUCTION

During recent years, object-oriented framework technology has become common technology in object-oriented software development [1, 6, 13, 18, 19]. Frameworks allow for large-scale reuse and studies show the use of framework technology reduces development effort, e.g. [19].

An object-oriented application framework is a reusable asset which constitutes the basis for a number of similar future applications in the application domain captured by the framework. Examples of well-known frameworks are ET++ [11], Java AWT [12], Microsoft Foundation Classes [21] in the domain of graphical user interface. Examples of application frameworks in other domains are fire alarm systems [18] and measurement systems [7].

Since a framework is intended to be reused several times, for similar applications, it is of importance to make the framework both general and flexible enough. However, the task of developing object-oriented application frameworks is both difficult and expensive [20, 14]. Major challenges in framework development are capturing the core domain concepts and incorporating the variability required by application developers. It is generally hard to foresee in what ways the application developers will reuse a framework. Therefore, frameworks generally evolve through a number of iterations, leading to new versions, due to the incorporation of new or changed requirements, better domain understanding and fault corrections.

New framework versions generally have a large impact since the framework often is used in several products. Evolving frameworks tend to change quite substantially, both internally and at their interfaces, leading to high maintenance cost for the products built based on the framework. Once a framework has matured, the changes between versions are considerably lower, with associated lower maintenance cost for the involved products. Because of this, it is important for the framework user to assess the maturity of a framework. Framework maturity assessment can, among others, be used by potential users when selecting a framework from multiple frameworks for a particular domain, by management to predict required maintenance effort both for the framework and applications built using it and by the framework developers to decide when a framework can be released for regular product development.

Despite its importance, few framework maturity assessment methods exist. The most prominent is the work by Bansiya [3, 4]. Bansiya suggests that the maturity of a framework can be assessed by observing the extent and degree of changes of a framework between versions. He defines a mature version of a framework as “a version that has the capabilities and structure to effectively be used in the development of most applications within the domain”. This definition is a measure of the stability in the framework structure and functionality and is a quantification of the extent-of-change (degree-of change). Bansiya’s work is based on one case, the Microsoft Foundation Classes (MFC) framework, for which he analyzed five subsequent versions. Based on the results from the case, he formulates four statements about framework maturity.

In this paper, we present a replicated case study of assessing the maturity of an object-oriented application framework using the technique proposed by Bansiya. We use the same overall approach, collect the same metrics and use the same tool for gathering these metrics. The only difference, to the

best of our knowledge, is the studied framework: a mediation framework developed by Ericsson Software Technology. We are able to support some of Bansiya's statements and unable to support others. Where we find differences, we present explanations and argumentation.

The reminder of the paper is organized as follows. In the next section, the studied framework and the assessment procedure are described. Section 3 presents the results of assessment for the mediation framework. The comparison between the original and replicated study are discussed in section 4 and the paper is concluded in section 5.

2 CASE STUDY DESCRIPTON

The study subject

The subject for the maturity assessment case 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 a 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 has been developed during the last four years.

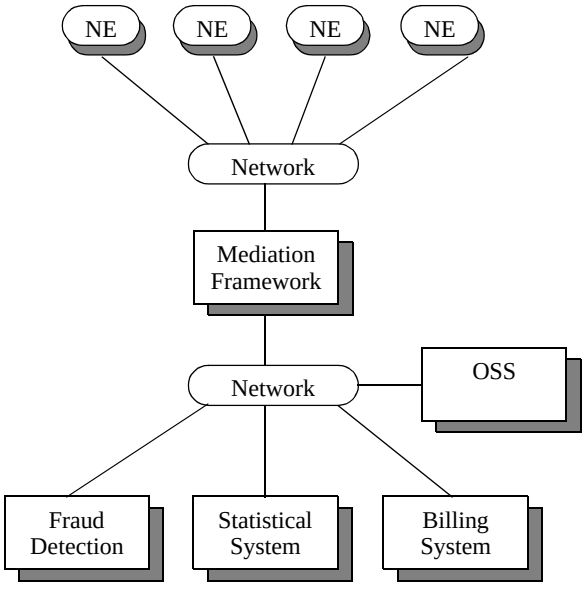

Figure 1: The Mediation Framework Context

The study subject in Bansiya's study [3] consisted of five versions of Microsoft Foundation Classes (MFC) [21], a commercial white-box application framework.

Assessment procedure description

Since our intention is to replicate the case study performed by Bansiya, we used the same assessment procedure for the framework maturity assessment. The assessment approach is straightforward and requires the class declarations of the framework as input, no additional source code is needed. This procedure consists of four steps, discussed in the sections below.

Table 1: Overview of Metrics

Framework Structure[a]	*Class (External)*	*Class (Internal)*
DSC (design size in classes)	DOI (depth of inheritance)	NOM (number of methods)
NOH (#hierarchies)	NOA (#ancestors)	CIS (class interface size)
NIC (#independent (non-inherited) classes)	NOC (#immediate children (subclasses))	NOI (#trivial (inline) methods)
NSI (#single inheritance instances)	DCC (#directly coupled classes)	NOP (# polymorphic methods)
NMI (#multiple inheritance instances)		NPT (#unique parameter types)
ADI (average depth of the class inheritance structure)		NPM (#parameters per method)
AWI (average width of the class inheritance structure)		NOD (#data attributes)
ANA (average number of ancestors per class)		NAD (#abstract data types)
		NRA (#reference attributes)
		NPA (#public attributes)
		CSB (class size in bytes)

a.The NAC (#abstract classes) metric has been excluded from the study due to tool limitations.

Identification of evolution characteristics. When assessing framework maturity, one has to select suitable indicators. Bansiya identifies three categories of evolution characteristics. First, the class structure and the relationships between the classes change during evolution. Consequently, the study of changes in inheritance relationships and the number of classes is of importance for studying framework evolution. Second, the external characteristics of framework classes evaluate the functional dependency of a class on other framework classes, i.e. relationships such as inheritance, aggregation, containment and acquaintance.

Finally, the internal characteristics of a class relate to its methods, method parameters, data declarations and member access declarations.

Selecting metrics to assess each evolution category. Several authors have proposed and studied metrics for object-oriented programming [8, 10, 5, 16] and for frameworks in particular [9]. However, since the original study [3] selects a set of metrics for each category and we replicate that study, we are forced to use the same set. The framework structure metrics are shown in the first column of table 1. A suite of four metrics is used to assess external class characteristics of the framework classes. These metrics are shown in the second column of table 1. The internal class characteristics are evaluated using a suite of 11 metrics, shown in the third column of the table.

Collect metrics data. The metrics data was collected using the same metric tool as used in the study by Bansiya, i.e., QMOOD++[4].

Computing the extent-of-change between the versions. For each of the framework versions, the framework structural, the external class and internal class metrics are calculated. Then all changes for each metric, for all classes, between two versions is calculated and summarized. The extent-of-change is then calculated by dividing the sum with the theoretical maximum of possible changes. The results of the case study are described in the next section, whereas the comparison to the original study is described in section 4.

Table 2: Structural metric values for four versions of the mediation framework

Metric	*V1*	*V2*	*V3*	*V4*
DSC	322	445	535	598
NOH	30	35	40	44
NIC	57	74	101	118
NSI	265	371	434	480
NMI	5	1	1	1
ADI	1.972	2.245	2.088	2.137
AWI	0.730	0.755	0.736	0.729
ANA	2.003	2.249	2.092	2.140

3 MEDIATION FRAMEWORK RESULTS

Framework Structure: Analyzing Changes and Results
In table 2, the structural metric data collected for the four versions of the mediation framework are shown. This data shows that the number of classes nearly doubles from 322 in version 1 to 598 in version 4. The values for number of hierarchies (NOH), number of independent classes (NIC) and the number of single inheritance (NSI) agree with the increasing value of number of classes. In addition, the values of average depth of inheritance (ADI) and average number of ancestors (ANA) are nearly the same due to the low number of multiple inheritance instances (NMI).

The average depth of inheritance (ADI) reflects the level of specialization in the framework and is expected to increase in newer versions and the average width of inheritance (AWI) is expected to decrease in new versions since new classes are generally added at deeper levels in the inheritance hierarchy. In our case study, we find a minor increase in the average depth of inheritance (ADI) values and a minor decrease of the average width of inheritance (AWI) values. Although this is as expected, the differences between the versions could have been larger. Two possible explanations are that the framework was already rather mature in its first version or that this is a property of black-box frameworks.

In table 3, the changes of the number of classes between the framework versions are presented. Frameworks evolve due to new or changed requirements and improved domain understanding. An increase of the number of classes is expected since newer framework versions generally incorporate additional functionality. The number of added classes is relatively high in version 2 (about. 50% of the number of classes in version 1). The number or removed classes is relatively low compared to the number of added classes. One explanation is that often classes are not removed due to backward compatibility.

Table 3: Changes in the number of classes between framework versions

Version	*V1*	*V2*	*V3*	*V4*
New Classes Added	322	174	120	128
Classes Removed	0	51	30	65
Total classes in the version	322	443	531	598

Class Characteristics: Analyzing Changes and Results
The external and internal class metrics defined in section 2 can be computed in two ways, i.e., including or excluding inherited properties. The first approach requires analyzing a class and all its ancestors for the computation of the class metrics. Bansiya's study showed that independent of the interpretation, the results will be similar. Because of that and due to reasons of space, only the first approach, i.e., including inherited properties, will be presented in this paper.

The metric for each of the 15 class characteristics can be changed between two successive versions. If the value of a particular metric (e.g. number of methods (NOM)) changes from one version to the next, this is defined as *one Unit of Change*. Since there are 15 characteristics that can be changed for a class, a class can contribute with between 0 to 15 Units of Change. The 15 metric values of each class are compared with the metric values of the class in its predecessor version to compute the total number of units-changed for the classes. The *extent-of-change* for a framework version is the sum of units-changed in all classes

between a version and its predecessor. In table 4, the 15 class metrics obtained for the four framework versions are presented.

Table 4: Changes in class characteristics and changes in metric values considered when a class is present in both of the compared versions

Metrics	*V1->V2*	*V2->V3*	*V3->V4*	*Units Changed*	*% Change*
DOI	47	13	10	70	1.7
NOC	19	31	47	97	2.4
NOA	47	13	10	70	1.7
DCC	89	80	141	310	7.5
NOM	148	142	198	488	11.9
CIS	147	131	177	455	11.1
NOT	89	63	69	221	5.4
NOP	105	93	117	315	7.7
NPT	77	125	187	389	9.5
NPM	147	156	190	493	12.0
NOD	78	126	144	348	8.5
NAD	69	51	82	202	4.9
NRA	76	88	58	222	5.4
NPA	16	9	16	41	1.0
CSB	77	125	187	389	9.5
Totals	1 231	1 246	1 633	4 114	100.0

To assess the significance of the extent-of-change measure and what the number really means, we compare it with the maximum possible change. The maximum possible change represents the case where all the metrics would have changed values for all classes in a framework version. In table 5, the extent of change is presented relative to the maximum possible change. For example, version 1 of the mediation framework has 322 classes which gives a theoretical maximum of 322 * 15 (number of metrics that can change for a class) = 4 830 possible changes for the (previous) version. The actual number of changes between version 1 and version 2 of the mediation framework is 1 231 which represents a (1 231 / 4 830) * 100 = 25.5% change between version 1 and 2. The extent-of-change measure is then between 18-21% of the theoretical maximum for version 2 to 4. From this figure we draw the conclusion that all the versions of the mediation framework are relatively mature since the extent-of-change measure is relatively constant.

In table 4, the metrics were calculated only for those classes that are present in both compared versions of the framework. In table 6, the same metrics are calculated for all framework classes including classes removed from the old version and classes added to the new version of the framework. Each new or added class will contribute with one unit-of-change for each metric.

Table 5: Extent-of-Changes as a function of the maximum changes for the Mediation framework versions

Units Change	*V1->V2*	*V2->V3*	*V3->V4*	*Total Change*
Max. units of change	4 830	6 675	8 025	23 670
Actual units changed	1 231	1 246	1 633	4 110
Extent of change (%)	25.5%	18.7%	20.3%	21.0%

In figure 2, the distribution of the changes is presented. The figure shows that the major types of changes are in the number of methods (NOM), number of parameter types (NPT), and number of parameters per method (NPM). We do not present the data from the MFC study but also there the NOM, NPT and NPM metrics possessed the highest number of changes.

Summary of the Mediation Case Study

A framework which mediates information between network elements and billing systems has been assessed with respect to framework maturity. The original size of the framework is 322 classes in version 1 and have evolved to 598 classes in version 4. During the evolution the values of the average depth of inheritance (ADI) and average width of inheritance (AWI) of the class hierarchies has been above 2.0 (ADI) and below 0.8 (AWI). The major changes in metric values between the versions has been for, number of methods (NOM), number of parameter types (NPT), and number of parameters per method (NPM). The percentage of changes for the framework versions calculated as the actual number of changes divided by the theoretical maximum of changes has been relatively constant, starting at 25% and than decreased to about 20% for the subsequent versions. Based on this constant change rate we consider the mediation framework to be mature (at least from version 2).

4 THE REPLICATED STUDY VERSUS THE ORIGINAL STUDY

Since we replicate a case study, it is relevant to discuss the similarities and differences between the results of our study and the results of the study by Bansiya [3, 4]. We will discuss the difference in case study subject and then discuss our results in relation to the Bansiya's statements about framework maturity. The following four statements of Bansiya were stated about framework maturity:

Statement 1: It takes the development and release of several versions (generally between three to five) to produce a

mature framework solution.

Statement 2: The percentage of changes in evolving frameworks as a function of the overall change tends to range between 50% to 30% of the maximum possible change. This extent of change drops to below 10% for mature versions.

Statement 3: The average number of changes in mature framework versions is about 50% less than the average number of changes in evolving frameworks.

Statement 4: Mature frameworks tend to have narrow and deeply inherited class hierarchy structures, characterized by high values for the average depth of inheritance (above 2.0) of classes and low values for the average width of inheritance (below 0.8) hierarchies.

Table 6: Changes in class characteristics and changes in metric values considered when a class is present in only one of the compared versions

Metrics	*V1->V2*	*V2->V3*	*V3->V4*	*Units Changed*	*% Change*
DOI	272	163	203	638	5.1
NOC	244	181	240	665	5.3
NOA	272	163	203	638	5.1
DCC	314	230	334	878	7.0
NOM	373	292	391	1 056	8.4
CIS	372	281	370	1 023	8.1
NOT	314	213	262	789	6.2
NOP	330	243	310	883	7.0
NPT	302	275	380	957	7.6
NPM	372	306	383	1 061	8.4
NOD	303	276	337	916	7.3
NAD	294	201	275	770	6.1
NRA	301	238	251	790	6.3
NPA	241	159	209	609	4.8
CSB	302	275	380	957	7.6
Totals	4 606	3 496	458	12 630	100.0

The case study subject

We have studied a proprietary black-box object-oriented framework, the *Billing Gateway Framework* by Ericsson Software Technology (EST), which is a mediation framework in the domain of collecting, translating and distributing billing information generated by mobile phone switches. There exist four versions of the framework. The size of the framework has increased from 322 to 598 classes during the development of the four versions. Prior to the framework development, EST had developed one application within the same domain, so the software engineers had domain experience preceding the development of the first framework.

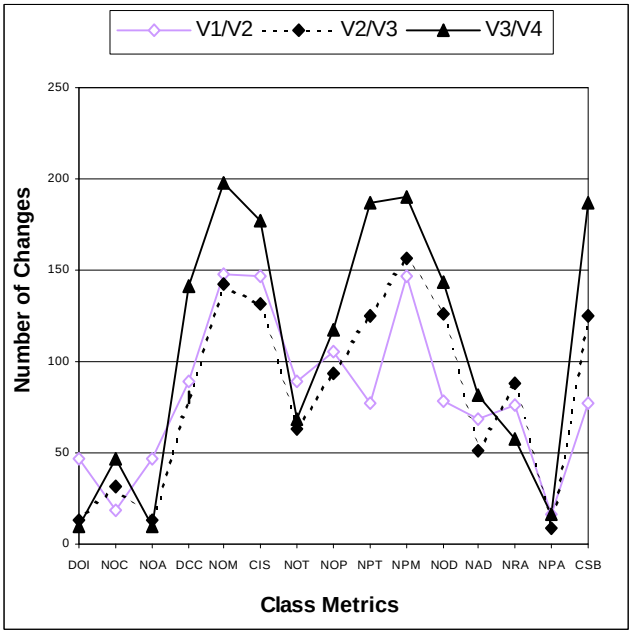

Figure 2: Distribution of changes per version

The subject in Bansiya's study is a commercial available white-box object-oriented framework, Microsoft Foundation Classes (MFC), which is a framework in the domain of graphical user interfaces for MS Windows applications. There exist five versions of the framework. The size of the framework has increased from 72 classes in the first version to 233 classes in version 5.

One can identify a number of relevant differences between the two frameworks. First, the mediation framework is a proprietary framework, whereas the MFC framework is a commercially sold framework. We consider a proprietary owned framework to be a more typical study subject since it represents the more common situation in software industry and because the framework developing organization in EST has to deal with explicit customers and customer requirements rather than distributing the framework to a mass market.

Second, the mediation framework is a black-box framework, whereas the MFC framework is white-box. Our experience is that black-box frameworks generally contain a larger number of classes and inheritance hierarchies than white-box frameworks. The rationale for this is that the normal extension mechanism in white-box frameworks is subclassing through inheritance, whereas black-box frameworks use parametrization. To support parametrization, black-box frameworks generally include multiple "options"-hierarchies containing concrete subclasses that can be used in framework instantiations.

A third difference is the size, measured in number of classes, of the two frameworks. The original mediation framework (322 classes) was originally 4.5 times larger than first MFC framework (72 classes). Version 4 of the mediation framework (598 classes) is still 2.9 times larger than the fifth version of the MFC framework (233).

Finally, the domain covered by the frameworks is considerable different. The MFC framework implements graphical user interface (GUI) functionality. Compared to most domains, the GUI domain is quite stable in the behavior it is supposed to provide. The mediation domain is considerable less stable and the developers of the framework have had to incorporated impressive amounts of requirement changes and additions during its life-time.

Statement 1: *It takes the development and releases of several versions (generally between three to five) to produce a mature framework solution.*

This statement from Bansiya is drawn from the fact that the number of added classes dropped from 76 for version 4 to 28 for version 5, table 7. In our study, we identify no drop of added classes but rather a very low increase of added classes between version 3 and version 4 (128 classes added, see table 3). This figure is, however, considerably less than for the first two frameworks (322 and 174 classes added, respectively). Concluding, we are not able to support the statement based on the argument that there has to exist a drop of the number of classes. This since the mediation framework does not experience a similar large drop of the number of added classes.

Table 7: Changes in the number of classes between MFC framework versions

Version	*V1*	*V2*	*V3*	*V4*	*V5*
New classes added	73	22	42	76	28
Classes removed	0	3	2	2	1
Total #of classes	73	92	132	206	233

Our explanation is that one should also incorporate the changes to the requirements for the framework and rather than an absolute figure, evaluate the added classes metric with respect to the changes and additions to the requirements. Thus, a drop or very low increase of the number of added classes to a version may be an indication of maturity, but one has to analyze the context of the framework development.

Statement 2: *The percentage of changes in evolving frameworks as a function of the overall change tends to range between 50% to 30% of the maximum possible change. This extent of change drops to below 10% for mature versions.*

Our study does not support this statement since we have a constant changing rate around 20% for the later two version transitions compared to the observed drop in the MFC study. In the mediation framework study there is a drop of about 7% from the first to the second version transition, but this is considerably less than the almost 30% drop in the MFC framework. See table 8 for the change percentages and table 9 for the actual change figures for the MFC framework.

Table 8: Percentage of changes as a function of the maximum changes for the Mediation and MFC frameworks

Units Change	*V1->V2*	*V2->V3*	*V3->V4*	*V4->V5*
Mediation	25.5%	18.7%	20.3%	N/A
MFC	47.5%	37.4%	35.8%	7.5%

Table 9: Change figures for the MFC framework

Units Change	*V1->V2*	*V2–>V3*	*V3->V4*
Max. units of change	1080	1380	1980
Actual units changed	513	516	709
Extent of change (%)	47.5	37.4	35.8

We agree with Bansiya that it is intuitively likely that there may exists this kind of a drop for a framework evolving towards maturity. However, our position is that Bansiya's statement was not validated in the original study since it is only based on one version (version 5) having a change rate of less 10%. If a change rate below 10% had been observed for 2-4 successive version for the same framework, the statement would have been considerably better founded.

Our interpretation of the differences between the mediation and the MFC framework change rates is that the mediation framework had a high degree of maturity from, at least, the second version. Indications of that is that there are 234 classes that exist throughout all the four versions. That is 72.7% of the classes in version 1 that exist also in version 2. In addition between version 1 to 3 there are 262 classes that exist in all three version and for version 2 to 4 there are 362 classes that exist. Unfortunately, we have not this kind of information for the MFC study.

In the mediation framework study we have a stabilized degree of changes, around 20% for all but the first version, which, we believe, is a realistic level for mature frameworks. Since this change rate has been observed for three consecutive framework versions and there is a stable set of classes throughout all versions, we have an indication of framework maturity. To make a stronger statement about an appropriate level for the degree of change for a mature framework, data on a larger variety of frameworks needs to be available.

A final observation concerning change rates is that framework changes are caused both by increased domain understanding and requirement changes. The MFC framework covers the GUI domain, which is a rather stable domain. For that framework, the change rate is mainly caused by the evolution of the framework. As stated earlier, we consider the mediation framework to be stable and believe that the change rate is primarily caused by the relatively rapidly evolving domain and the associated requirement changes. For instance, in the case of the mediation framework, there have been a large number of new and changed requirements that had to be incorporated into the framework due to requirements from individual customers.

Statement 3: *The average number of changes in mature framework versions is about 50% less than the average number of changes in evolving frameworks.*

This statement by Bansiya is somewhat ambiguous, but we decided to interpret the statement as the average number of changes for a class in a mature framework is about 50% less than the average number of changes for a class in an evolving framework.

In table 10, the average number of changes per class is shown for each of the two frameworks. The average number of changes per class drops in the MFC framework from 8-9 down to just above 3 for version 5. In the mediation framework, the average number of changes per class is rather constant around 3 changes per class for the later two version transitions.

Table 10: Average number of units changed per class when a class is present in both versions

	V1->V2	*V2->V3*	*V3->V4*	*V4->V5*
Mediation Framework	3.82	2.81	3.07	N/A
MFC Framework[a]	9.00	8.80	8.74	3.17

a.The MFC values are calculated the way we interpret the statement and differs from the values in [4].

The fact that the mediation framework has a low, relatively constant average number of changes per class, we believe, can be explained by the maturity of the framework. Although the framework has been expanded dramatically since the first version, there are relatively few changes in existing classes. One explanation for the framework achieving maturity very quick may be the competence of the involved software engineers. The main framework designers, but also most of the other staff involved in the framework, are highly skilled, experienced software engineers with a solid understanding of object-oriented concepts. In addition, as mentioned earlier in the paper, the organization had developed one application in the domain before designing the object-oriented framework.

Based on our case study, we do not believe that a drop in the average number of changes per class is a necessity for indicating framework maturity. Instead, we feel that a low average changes per class, e.g. around 3, is an indication of framework maturity. Thus, the absolute figure may be a better indicator than a drop in the change figure over versions, since an absolute low figure shows that the concepts represented by the classes are stable and need not be modified extensively when requirements are changed or added.

Statement 4: *Mature frameworks tend to have narrow and deeply inherited class hierarchy structures, characterized by high values for the average depth of inheritance (above 2.0) of classes and low values for the average width of inheritance (below 0.8) hierarchies.*

The limits stated by Bansiya, ADI > 2.0 and AWI < 0.8, are also valid for the evolution of the mediation framework. In table 11, the metric values for the frameworks are presented. For the MFC framework, the ADI and AWI metrics are based on one or a few class hierarchies, whereas, for the mediation framework, the metrics are calculated based on 30-40 class hierarchies present in the system. The large number of class hierarchies is, most likely, a consequence of the fact that the mediation framework is a black-box framework. It is interesting to note that the ADI and AWI metrics are within the ranges specified in statement 4 for such diverse frameworks. The statement seems to have a rather general validity.

Table 11: The ADI and AWI metric values for the Mediation and MFC frameworks

Metric	*V1*	*V2*	*V3*	*V4*	*V5*
ADI (Mediation)	1.972	2.245	2.088	2.137	N/A
ADI (MFC)	1.68	2.11	2.45	2.33	2.3
AWI (Mediation)	0.730	0.755	0.736	0.729	N/A
AWI (MFC)	0.917	0.913	0.886	0.845	0.833
NOH (Mediation)	30	35	40	44	N/A
NOH (MFC)	1	1	1	5	6

Summary

We have discussed the similarities and differences between the results from the two framework maturity assessments as well as the difference between the two studied frameworks. Here, we briefly summarize the results of the study.

The mediation framework study does not support statement 1, i.e. that it takes 3-5 releases of a framework before an evolving framework becomes a mature framework. This statement is based on a drop of the number of added new classes between two consecutive versions of a framework. This drop was not observed in the mediation framework case. However, a very low difference between the number of added classes in the last two versions was observed, indicating that the level has stabilized. In addition, we believe that it is necessary to incorporate added and changed requirements on the framework and relate them to the number of added classes to get an indication of a framework's maturity.

With respect to statement 2: The percentage of changes did not drop for the mediation framework from 30-50% down to below 10% as stated in the original study. Instead the mediation framework has a relatively constant percentage of changes (around 20%) throughout all versions, except the first. The second part of statement 2, i.e. that for mature frameworks overall change drops to 10%, we consider not to be backed by the original study since it is based on only one version of the MFC framework. We believe that the mediation framework is mature since it shows a constant degree of changes (20%), contains a stable set of classes that have existed in all four versions and shows a low average number of changes per class throughout the versions. The difference in change rate between the frameworks can be explained by the domain covered by the framework. In the MFC case, the GUI domain is stable and the framework evolves. The mediation framework is stable from, at least version 2, but the domain is changing due to new and changed requirements from explicit customers. An additional explanation is that the mediation framework was developed by highly skilled, experienced software engineers.

Statement 3, i.e. that the average number of changes for a class is decreased with about 50% (from 8-9 down to 3 changes) for a class in a mature framework was not confirmed since the average number of changes per class for the mediation framework was, from the first version, on a low level (around 3). Instead, we believe that a low absolute number is a better indicator than the aforementioned drop.

Finally, statement 4 is supported by our study. The mediation framework study validates that the average depth of inheritance is above 2.0 and that the average width of inheritance is below 0.8. This independent from the fact that the mediation framework comprised of 30-40 class hierarchies.

Reflections

In this section we make some reflections about the four statement and the assessment approach. The assessment approach by Bansiya is based on four indicators of framework maturity:

- A drop of added classes for the framework
- A drop of the percentage of changes metric
- A drop of the average number of changes per class
- The AWI metrics is < 0.8 and the ADI metric > 2.0

The major indicator of framework maturity is the percentage of changes measure. This measure seems to be more appropriate to base statement 1 on, than the statement that the number of added classes to a version has dropped. A low number of added classes in a version can be the result of a mature framework, but it could be that the changes and additions to the requirements of the framework primarily affect the existing classes. So, although the statement has been formulated by other authors as well, e.g. [14], it should be supported by different arguments than the one used by Bansiya. We believe that an appropriate argument should make use of the percentage of change measure. A better statement is that a framework is considered mature if the percentage of changes measure is relatively constant, around 20% or below, for three or more versions. This since it may be the case that early versions (1 or 2) of a framework could have been very well-designed to meet future requirements.

Regarding statement 2 it is hard to define an interval for the extent of changes metric for an evolving framework based on one framework observation. To find a scientifically validated interval will require studies of a set of frameworks. The drop of the percentage of changes metric, if it occurs, could be an indication. But, the low degree of changes should be observed for two to three consecutive versions of the framework to be confident about the maturity. However, as argued previously, it is not always the case that a drop occur since the first versions of a framework could be well-designed and thereby mature. In addition, the value of the metric is influenced by the changes and additions to the framework requirements as well.

Since statement 3 is a 'localized' variant of statement 2 the same critique remains for that statement. In addition, based on our case study, we consider a low absolute value on the average changes per class to be a better indicator of framework maturity than a relative drop. The statement about the values for the average depth and width of inheritance metrics, statement 4, seems to accurately capture the degree of specialization in a framework.

The assessment approach is, in our experience, a straightforward procedure which is simple to apply in a software development organization, but we believe that an incorporation of the requirements aspects will increase the accuracy of the assessment procedure.

5 CONCLUSION

Assessment of the maturity of an object-oriented application framework is important since evolving frameworks cause high maintenance cost for the applications build based on the framework. Few framework maturity assessment methods exist, but one of the most prominent is the approach proposed by Bansiya [3, 4]. Based on his study of five versions of the Microsoft Foundation Classes framework, Bansiya formulates four statements: (1) framework maturity is achieved generally in the third to fifth version, (2) evolving frameworks change between 30% and 50% of the maximum number of changes between version, whereas this figure is below 10% for mature versions, (3) the average number of changes per class in mature frameworks is about 50% less than in evolving frameworks and (4) mature

frameworks have an average depth of inheritance above 2.0 and an average width of inheritance below 0.8.

In this paper, we replicated the original study by Bansiya for four versions of a proprietary, black-box framework in the mediation domain. The mediation framework has developed from 322 classes in the first version to 598 classes in version four. Our, replicated, study is identical to the original study in that it uses the same overall approach, collects the same metrics and uses the same tool for gathering these metrics. The only difference, to the best of our knowledge, is the studied framework.

One can identify four major differences between the MFC framework and the mediation framework. First, the mediation framework is a proprietary framework used for product construction by the company owning the framework, whereas the MFC framework is sold to a mass market. Second, the mediation framework is a black-box framework, whereas the MFC framework is white-box. Third, the mediation framework is considerably larger than the MFC framework; between 4.5 and 2.9 times depending on the version. Finally, the MFC framework covers the, rather stable, GUI domain, whereas the mediation domain is considerably less stable.

The results of our replicated case study can be summarized as follows. The mediation framework did not experiences a drop in the number of added classes in the later versions as in the case of the MFC framework. Consequently, we are not able to support statement 1, but add that a more accurate evaluation would investigate the number of added classes relative to the change in framework requirements. We are unable to support statement 2, since we have a constant extent of change of around 20% for the later two versions. Our interpretation is that the mediation framework has been mature at least since version 2 but that, instead, the domain and requirements on the framework have been unstable. For the MFC framework, the GUI domain is rather stable and it is the MFC framework itself that has matured. We are also unable to support statement 3 in its current form. However, instead of a drop in the average number of changes per class, our interpretation is that an absolute low figure for this metric is an indicator of framework maturity. Finally, our case study supports statement 4. All but the first version of the mediation framework have an average depth of inheritance over 2.0 and all versions have an average width of inheritance below 0.8.

As part of our future work, we intend to apply the framework maturity assessment approach to other industrial frameworks as well to increase the base of maturity data. In addition, since we were unable to support some of Bansiya's statements, we intend to propose alternative hypothesis with respect to framework maturity and evaluate these with the available data. Finally, as part of defining alternative hypothesis, we intend to investigate how one can incorporate changes and additions to the framework requirements and relate this to the framework change figures.

ACKNOWLEDGMENTS

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

REFERENCES

1. G. Andert, 'Object Frameworks in the Taligent OS,' *Proceedings of Compcon 94*, IEEE CS Press, Los Alamitos, California, 1994.
2. Apple Computer Inc., *MacAppII Programmer's Guide*, 1989.
3. J. Bansiya, 'Assessing Maturity of Object-Oriented Application Frameworks,' Accepted for publication in *Object-Oriented Application Frameworks: Problems & Perspectives*, M. E. Fayad, D. C. Schmidt, R. E. Johnson (eds.), Wiley & Sons (Forthcoming), http://indus.cs.uah.edu/research-papers.htm.
4. J. Bansiya, 'Assessment of Application Framework Maturity Using Design Metrics,' Accepted for publication in *ACM Computing Surveys - Symposia on Object-Oriented Application Frameworks*, 1998, http://indus.cs.uah.edu/research-papers.htm
5. J. Bansiya, 'A Hierarchical Model for Quality Assessment of Object-Oriented Design,' *Ph.D Dissertation*, University of Alabama, 1997
6. 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)
7. J. Bosch, 'Design of an Object-Oriented Framework for Measurement Systems,' in *Object-Oriented Application Frameworks*, M. Fayad, D. Schmidt, R. Johnson (eds.), John Wiley, 1998. (forthcoming)
8. S. R. Chidamber, C. F. Kemerer, 'A Metrics Suite for Object-Oriented Design', *IEEE Transactons on Software Engineering*, Vol. 20, No, 6, pp. 476-493, June 1994.
9. K. Erni, C. Lewerentz, 'Applying design metrics to object-oriented frameworks,' *Proceedings of the Metrics Symposium*, 1996.
10. N.E. Fenton, *Software Metrics - A Rigorous Approach*, Chapman & Hall, 1991.
11. E. Gamma, *Object-Oriented Software Development based on ET++: Design Patterns, Class Library, Tools* (in German), Springer-Verlag, 1992
12. D. Geary, *Graphic Java 1.1 : Mastering the AWT*, Prentice Hall, 1997.
13. 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.
14. R.E. Johnson, V.F. Russo, 'Reusing Object-Oriented Design,' *Technical Report UIUCDCS 91-1696*, University of Illinois, 1991.
15. R. Lajoie, R.K. Keller, 'Design and Reuse in Object-oriented Frameworks: Patterns, Contracts and Motifs in Concert,' *Proceedings of the 62nd Congress of the Asso-*

ciation Canadienne Francaise pour l'Avancement des Sciences, Montreal, Canada, May 1994.

16. W. Li, S. Henry, 'Object-Oriented Metrics that Predict Maintainability,' *Journal of Systems and Software*, Vol. 23, pp. 111-122, 1993.
17. M. Mattsson, 'Object-Oriented Frameworks - A survey of methodological issues,' *Licentiate Thesis,* LU-CS-TR: 96-167, Department of Computer Science, Lund University, 1996.
18. 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.
19. S. Moser, O. Nierstrasz, 'The Effect of Object-Oriented Frameworks on Developer Productivity,' *IEEE Computer,* pp. 45-51, September 1996.
20. 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.
21. G. Shepherd, S. Wingo, *MFC Internals: Inside the Microsoft Foundation Class Architecture,* Addison-Wesley, 1994.
22. K.J. Schmucker, *Object-Oriented Programming for the Macintosh*, Hayden Book Company, 1986.
23. S. Sparks, K. Benner, C. Faris, 'Managing Object-Oriented Framework Reuse,' *IEEE Computer,* pp. 53-61, September 1996.