Characterizing Stability in Evolving Frameworks

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—99/8—SE

Characterizing Stability in Evolving Frameworks

by Michael Mattsson, Jan Bosch

ISSN 1103-1581

ISRN HK/R-RES—99/8—SE

Copyright © 1999 by Michael Mattsson, Jan Bosch

All rights reserved

Printed by Psilander Grafiska, Karlskrona 1999

Characterizing Stability in Evolving Frameworks

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. Frameworks generally evolve through a number of iterations due to the incorporation of new requirements and better domain understanding. Since changes to frameworks have a large impact on the applications built based on the asset, it is important to assess the stability of a framework. Recently, an approach for assessing framework stability has been proposed [3]. We have extended and applied the assessment approach on one proprietary telecommunication framework and two commercial GUI application frameworks. Based on our findings we formulate a set of hypotheses, which characterize the stability of an object-oriented application framework. We believe these hypotheses to be the most promising ones for further studies of framework stability.

Keywords: Object-oriented frameworks, framework stability, framework evolution, metrics

1 INTRODUCTION

During recent years, object-oriented framework technology has become common technology in object-oriented software development [1, 6, 15, 21, 22]. Frameworks allow for large-scale reuse and studies show the use of framework technology reduces development effort, e.g. [22].

An object-oriented application framework is a reusable asset that 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++ [12], Java AWT [14] and Microsoft Foundation Classes [24] in the domain of graphical user interface. Examples of application frameworks in other domains are fire alarm systems [21] 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 [23, 16]. 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 tend to 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 versions of a framework have considerable impact on the products that include it, especially if its interface is changed or when the framework has been adapted for the product. 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 achieved stability, 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 stability of a framework. Framework assessment can, among others, be used by potential users when selecting a framework from multiple available 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.

To assess the suitability of a framework for use in application development, one is interested in a general *maturity* of the framework, in terms of structural and behavioral stability, available documentation, support from the framework vendor, etc. Our experience is that the stability of the framework is a good indicator of general framework maturity. Since this stability can be assessed using metrics on the framework code, it leads to a more objective evaluation than the other aspects of maturity.

Despite the fact that the use of object-oriented frameworks is relatively widespread in software industry, assessment methods for framework maturity are few. One example is the work by Bansiya [3, 4]. Bansiya suggests that the

stability of a framework can be assessed by observing the extent and degree of changes of a framework between versions. He defines a structural stable 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). In this paper, we take Bansiya’s approach as a starting point for defining a framework stability assessment method. Using this method, we present data collected from three object-oriented frameworks, i.e. four versions of the billing gateway framework developed by Ericsson Software Technology, five versions of the Microsoft Foundation Classes and four versions of Borland’s Object Windows Library. The data for the latter two versions is taken from [3], whereas the data for the former is collected by us. For all data collection, an identical approach is used, including the same metrics and the same tool for gathering these metrics. The only difference, to the best of our knowledge, is that the billing gateway framework is a proprietary framework used for the development of applications within Ericsson Software Technology, whereas the other two frameworks are commercial GUI frameworks.

Based on the data collected from these framework versions, we propose six framework stability hypotheses¹ that are, either completely or partially, supported by the collected data. These hypotheses, once sufficiently validated, provide objective and quantitative means to assess framework stability and, indirectly, framework maturity. However, we also show that careful analysis of the data in its context is required before one can make well-founded statements about framework stability.

The main contribution in this paper is a set of six framework stability hypothesis that we are able to either completely or partially validate based on the available data. The hypotheses has been formulated by applying a proposed method for assessing framework stability. So, a minor contribution is the validation of the assessment method’s applicability.

The reminder of the paper is organized as follows. In the next section, the studied framework and the assessment procedure are described. Section 3 and 4 present the architectural and class level data and the hypotheses. In section 5 reflections about the assessment methodology and the analyzed frameworks are presented. Related work is found in section 6 and the paper is concluded in section 7.

2 CASE STUDY DESCRIPTON

2.1 The study subjects

The data for the framework stability study presented in this paper consists of three object-oriented frameworks of considerable size. For each framework, at least four versions have been analyzed and data is collected. The first framework is the billing gateway framework (BGW) developed and used by Ericsson Software Technology. The framework is proprietary and is not sold to others. The framework has lived through five major versions, of which the latter four have been analyzed. The second framework is a graphical user interface framework developed by Microsoft, i.e. Microsoft Foundation Classes (MFC). Five versions of this framework have been analyzed. Finally, the third framework is the Object Windows Library (OWL) developed by Borland. Metrics data for four versions of this framework is available. The latter two frameworks, i.e. MFC and OWL, are GUI frameworks that are commercially sold on a mass-market. Since the BGW framework is not well known we describe in the following section to provide the reader with appropriate knowledge of the framework and the captured domain. The MFC and OWL frameworks are not described since they capture the well known graphical user interface domain.

Billing Gateway (BGW)

The BGW framework is a major part of 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. Five versions of the mediation framework have been developed during the last four years, all in C++. However, due to practical reasons, we were unable to include the first version in the metrics collection.

2.2 Assessment procedure description

Recently Bansiya presented a methodology for assessing stability of an object-oriented framework [3]. In our work we have used the proposed method and added an additional metric, *Relative-Extent-of-Change*. The Relative-Extent-of-Change metric is described in detail in section 4.1. The assessment approach analyzes the class declarations of the framework and no additional source code is required. The procedure consists of four steps, discussed in the sections below.

1. Two of the hypotheses are variations of statements in an earlier version of [3].

Identification of evolution characteristics. When assessing framework stability, one has to select suitable indicators. Bansiya identifies two categories of evolution characteristics, *architectural* and *individual class* characteristics. First, the changes of architectural characteristics are related to the *interaction* (collaborations) among classes and the *structuring* of classes in inheritance hierarchies.

Figure 1: The Mediation Framework Context

Second, the characteristics of individual classes are related to the assessment and change of the *structure*, *functionality* and *collaboration* (relationships) of individual classes between versions. The structure of objects is described and detailed by the data declarations in the class declarations. The functional characteristics are related to the object's methods. The parameter and data declarations that use user-defined objects define the collaboration characteristics of a class with other classes.

Selecting metrics to assess each evolution category. Several authors have proposed and studied metrics for object-oriented programming [8, 10, 5, 18] and for frameworks in particular [9]. In table 1 the architectural framework structure and interaction metrics, which are a selection from [5, 8], are described.

Table 1: Architectural Metrics

<i>Architectural Metrics (Structure)</i>	<i>Architectural Metrics (Interaction)</i>
DSC, design size in classes	ACIS, average number of public methods in all classes
NOH, number of hierarchies	ANA, average number of distinct (parent) classes from which a class inherits information
NSI, number of single inheritance instances	ADCC, average number of distinct classes that a class may collaborate with
NMI, number of multiple inheritance instances	
ADI, average depth of the class inheritance structure	
AWI, average width of the class inheritance structure	

The individual class metrics comprise in total of 11 metrics divided into three separate suits. The structural suite consists of the following four metrics

- NOD, the number of attributes in a class
- NAD, the number of user defined objects (ADTs) used as attributes in a class
- NRA, the number of pointers and references used as attributes in a class

- CSB, the size of the objects in bytes that will be created from a class declaration. Size is computed by summing the size of all attributes declared (class size in bytes).

The functional suite has the following four metrics

- NOM, count of all the methods defined in a class
- NOP, the number of public methods in a class
- NPT, the number of different parameter types used in the methods of a class
- NPM, average of the number of parameters per method in a class.

The final three metrics comprise the relational suite

- NOA, the number of distinct classes (parents) which a class inherits
- NOC, the number of immediate children (sub classes) of a class
- DCC, count of the different number of classes that a class is directly related to. Includes classes that are directly related by attribute declarations and message passing (parameters) in methods.

Collect metrics data. The metrics data was collected using the QMOOD++ tool [4]. The data has been collected for each version of the three frameworks with the same tool, thereby reducing the probability for different interpretations of the metric definitions.

Analysing the changed architectural characteristics and computing the extent-of-change in the architecture between framework versions.

For each of the frameworks and their versions, the collected architectural data are analyzed and the Normalized - Extent-of-Change metric are calculated. In addition a calculation of the relative distribution of the total number of changes for all framework versions is done.

3 ARCHITECTURAL LEVEL DATA AND ANALYSIS

3.1 Analyzing Architectural Metric Values and Results

BGW Observations

In table 2, the architectural metric data collected for the four versions of the BGW framework 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) 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. One possible explanation may be that this is a property of black-box frameworks.

The average number of public methods (ACIS), 15-19 methods, and the average of directly coupled classes (ADCC), 3,6- 4,0 classes, are fairly constant throughout the versions.

MFC Observations

We see that MFC has had an increase of the number of classes more than three times, from 72 classes in version 1 to 233 classes in version 5. Regarding inheritance relationships MFC do not make use of multiple inheritance and in version 4 the number of inheritance hierarchies was increased to five from only one in the previous versions. The values of average depth of inheritance (ADI) and average number of ancestors (ANA) is the same due to the lack of multiple inheritance in MFC. The average number of public methods has increased from 75,6 public methods up to 110-115 in the last two versions. The average number of directly coupled (ADCC) classes are relatively constant around 9,0 with an exception for the early version where the values was lower.

OWL Observations

The metric data for the OWL framework are obtained from [5] but in our study we will in only make use of the data for the two major versions of OWL. This since we would like to provide correct input to the assessment procedure. A practical effect of using minor versions would be that they, per definition, exhibit a low degree of change between versions. The minor OWL version are marked with grey table cells.

The OWL framework has a major increase of number of classes from 82 classes in version 4 to 357 classes in version 5. The values for number of hierarchies (NOH) and the number of single inheritance (NS) agree with the increasing

value of number of classes. The OWL framework is the only framework of the three that makes extensive use of multiple inheritance (NMI), i.e. 9 cases in version 4 and 30 cases in version 5. The average depth of inheritance (ADI), average width of inheritance and average number of ancestors (ANA) are all lower than for the other frameworks. This is probably due to the high number of multiple inheritance instances. The values for the average number of public methods (ACIS) and directly coupled classes (ADCC) show a considerable increase between the two versions.

Table 2: Architectural metric values

<i>Metric</i>	<i>BG W1</i>	<i>BG W2</i>	<i>BG W3</i>	<i>BG W4</i>	<i>MF C1</i>	<i>MF C2</i>	<i>MF C3</i>	<i>MF C4</i>	<i>MF C5</i>	<i>OW L4</i>	<i>OW L4. 5</i>	<i>OW L5</i>	<i>OW L5. 2</i>
DSC	322	445	535	598	72	92	132	206	233	82	142	357	356
NOH	30	35	40	44	1	1	1	5	6	12	16	29	27
NSI	265	371	434	480	66	84	117	174	194	34	56	202	198
NMI	5	1	1	1	0	0	0	0	0	9	13	30	30
ADI	1.97	2.24	2.09	2.14	1,68	2,11	2,45	2,33	2,30	0,78	0,81	1,42	1,41
AWI	0.73	0.75	0.74	0.73	0,92	0,91	0,89	0,85	0,83	0,52	0,49	0,65	0,64
ANA	2.00	2.25	2.09	2.14	1,68	2,11	2,45	2,33	2,30	1,00	0,97	1,56	1,55
ACIS	15,8 4	17,5 8	18,9 0	18,6 8	44,6	75,6	95	114	109	27,7	24	37,2	36,8
ADCC	3,61	4,07	3,99	3,98	6,03	7,59	9,02	9,33	8,94	1,87	2,26	4,73	4,71

3.2 Hypotheses

Frameworks evolve due to new or changed requirements and improved domain understanding. An increase of the number of classes is expected, as we have seen in table 2 for all three frameworks, since newer framework versions generally incorporate additional functionality. From analysis of the values in table 2 we formulate the following two hypotheses about framework stability:

Hypothesis 1: Stable frameworks tend to have narrow and deeply inherited class hierarchy structures, characterized by high values for the average depth of inheritance (above 2.1) of classes and low values for the average width of inheritance hierarchies (below 0,85).

Through observations of the values for the MFC and BGW frameworks we see that the initial ADI values are increasing from values below 2.0 and then are successively increasing to values above 2.1. During the evolution of the frameworks, some versions have peaking values for the ADI metric. For later versions, the ADI metric decreases to a level between 2.1 and 2.3. The OWL framework exhibits deviating values for the ADI metric compared with the other two frameworks.

The AWI metric for the BGW framework is around 0,75 and decreases to 0.83 for the MFC framework. The OWL framework exhibits deviating values for the AWI metric as well. A possible explanation why the OWL has deviating values for both the ADI and AWI metrics could be its use of multiple inheritance which make the hierarchies more shallow and narrow.

Based on the observation above we formulate the hypothesis, “Stable frameworks tend to have narrow and deeply inherited class hierarchy structures, characterized by high values for the average depth of inheritance (above 2.1) of classes and low values for the average width of inheritance hierarchies (below 0,85)”.

Hypothesis 2: A stable framework has an NSI/DSC ratio just above 0.8 if multiple inheritance is seldom used in the framework. I.e. the number of subclasses in a stable framework is just above 80 percent.

If we consider the ratio between number of subclasses and the total number of classes for a framework version we see that the ratio is going towards a limit value of 0.8 for the BGW and MFC frameworks, table 3. In the case of the OWL framework the limit value is below 0.6 which probably are due to the extensive use of multiple inheritance. Thus, we can formulate the following hypothesis for stable frameworks that do not rely on multiple inheritance, “A stable

framework has an NSI/DSC ratio just above 0.8 if multiple inheritance is seldom used in the framework. I.e. the number of subclasses in a stable framework is just above 80 percent”

Table 3: The NSI/DSC Ratio

<i>Metric</i>	<i>BGW 1</i>	<i>BGW 2</i>	<i>BGW 3</i>	<i>BGW 4</i>	<i>MFC 1</i>	<i>MFC 2</i>	<i>MFC 3</i>	<i>MFC 4</i>	<i>MFC 5</i>	<i>OWL 4</i>	<i>OWL 4.2</i>	<i>OWL 5</i>	<i>OWL 5.2</i>
NSI	265	371	434	480	66	84	117	174	194	34	56	202	198
DSC	322	445	535	598	72	92	132	206	233	82	142	357	356
<i>NSI/ DSC</i>	<i>0,82</i>	<i>0,83</i>	<i>0,81</i>	<i>0,80</i>	<i>0,92</i>	<i>0,91</i>	<i>0,89</i>	<i>0,84</i>	<i>0,83</i>	<i>0,41</i>	<i>0,39</i>	<i>0,57</i>	<i>0,56</i>

3.3 Analyzing Normalized Architectural Metric Values and Results

In table 4 the normalized architectural metrics and the normalized-extent-of-change metric is presented. In the table the architectural metric values are normalized with respect to the metric's values in the previous version of the framework. The metric values for the first version of each framework is used as a base for normalization. The normalized metric values are computed by dividing the actual metric values of a version with the metric's value in the previous version for the actual framework. The normalized-extent-of change metric is computed by summarizing the normalized metrics for a framework version. The aggregated metric has for the first version of the framework the same values as the number of architectural metrics (nine). Second, the normalized extent-of-change metric is then computed by taking the difference of the aggregate metrics for the framework version, V_i , and the first framework version V_1 . For the first version of a framework the normalized extent-of-change metric is set to 0. For example, the normalized extent-of-change metric for version 3 of the BGW framework is computed as 9,41 (aggregate metric values for version 3) minus 9 (aggregate metric value for version 1) which is 0,41. Thus, the normalized-extent-of-change metric has the value 0,41 for version 3 of the BGW framework.

Table 4: Normalized Architecture metrics values

<i>Metric</i>	<i>BG W1</i>	<i>BG W2</i>	<i>BG W3</i>	<i>BG W4</i>	<i>MFC1</i>	<i>MFC 2</i>	<i>MFC 3</i>	<i>MFC 4</i>	<i>MF C5</i>	<i>OW L4</i>	<i>OWL 5</i>
DSC	1	1,38	1,20	1,12	1	1,28	1,43	1,56	1,13	1	4,35
NOH	1	1,17	1,14	1,10	1	1	1	5	1,2	1	2,42
NSI	1	1,40	1,17	1,11	1	1,27	1,39	1,49	1,11	1	5,94
NMI	1	0,2	1	1	1	1	1	1	1	1	3,33
ADI	1	1,14	0,93	1,02	1	1,26	1,16	0,95	0,99	1	1,82
AWI	1	1,03	0,97	0,99	1	0,99	0,98	0,96	0,98	1	1,25
ANA	1	1,12	0,93	1,02	1	1,26	1,16	0,95	0,99	1	1,56
ACIS	1	1,11	1,08	0,99	1	1,70	1,26	1,20	0,96	1	1,34
ADCC	1	1,13	0,98	1	1	1,26	1,19	1,03	0,96	1	2,53
Aggregate Change (Vi)	9	9,68	9,41	9,35	9	11,01	10,57	14,14	9,31	9	24,55
Normalized Extent-Of- Change	0	0,68	0,41	0,35	0	2,01	1,57	5,14	0,31	0	15,55

The normalized extent-of-change metric is a relative indicator of the framework's architectural stability. A high value of the metric indicate relative instability of the framework structure and a low value of the normalized-extent-of-change metric indicate greater stability.

BGW Observations

For the BGW framework we see that the normalized values for number of hierarchies (NOH) is decreasing from 1,17 to 1,10. This indicates that the addition of new inheritance hierarchies is slowly decreasing with newer versions of the framework. Another observation is that the average number of directly coupled classes (ADCC) is fluctuating just over and below 1.0. This indicates that the “relationship complexity” is not increasing with newer versions, i.e. the number of relationships for a class is not increasing significantly in a new version of the framework.

MFC Observations

The normalized value for the number of hierarchies (NOH) metric makes a jump up to 5 in version 4 due to the introduction of five separate hierarchies in the framework. The normalized values for the number of multiple inheritance hierarchies (NMI) is constantly equal 1, which is not surprising, since there are no occurrences of multiple inheritance in the framework. In the MFC framework case one can make the same observation as in the BGW framework regarding the average number of directly coupled classes (ADCC) metric. It is close to 1, indicating that the “relationship complexity” is not increasing with newer versions of the framework.

OWL Observations

For the OWL version the values for only the major versions 4.0 and 5.0 are presented in table 4. For all metrics except the average of public methods (ACIS) metric the normalized values are significantly higher than the values for the other frameworks. The values for ACIS have an increase to 1,34, which can be compared to the peaks found in version 2 of the other two frameworks. In general it is difficult to identify any kind of pattern due to the low number of framework versions.

3.4 Hypotheses

Hypothesis 3: The normalized ADCC metrics is going towards 1.0 or just below for stable frameworks.

Through analysis of the BGW and MFC framework we see that the ADCC metric have a little peak in version 2 and then is decreasing slowly down to 1 or below in later versions. Since the OWL framework only has two major versions it does not contradict this observation. The fact that the ADCC is close to 1 means that the “relationship complexity” is not increasing with newer versions nor is it decreasing, i.e. the number of relationships for a class is relatively stable and constant throughout all versions of the framework in average. Thus, we are able to formulate the following hypothesis “the normalized ADCC metrics is going towards 1.0 or just below for stable frameworks”.

Hypothesis 4: The Normalized-Extent-of-Change metric is below 0.4 for a stable framework.

The MFC framework shows a Normalized-Extent-of-Change value beginning at 2,01 and then successively decreasing down to 0,31 for version 5. For version 4 the Normalized-Extent-of-Change value is 5,14 which is an atypical value for the MFC framework. The reason for this the introduction of four new hierarchies in version 4 of MFC, otherwise the values would probably be around 1,5 or lower. For the OWL framework the Normalized-Extent-Of-Change metrics is 15,55 for version 5, which is extremely high. The reason for the high value can not be found from just one single normalized architectural metric but is the result of the total increase for all normalized architectural metrics. In the case of the BGW framework the normalized extent-of-change metric is decreasing from 0,68 to 0,41 and ending with 0,35 for version 4. All version having a values less than 0,7. Based on discussion above we formulate the following hypothesis “The Normalized-Extent-of-Change metric is below 0.4 for a stable framework”.

4 CLASS LEVEL DATA AND ANALYSIS

4.1 Analyzing Changes and Results

The individual class metrics defined in section 2.2 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. Typically, changes made to the internal parts of parent classes in a framework ripple the effect of changes to all descendant of the parent class. So all metrics values presented here are counted including the inherited properties.

The metric for each of the 11 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 11 characteristics that can be changed for a class, a class can contribute with between 0 to 11 Units of Change. The 11 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 *total-extent-of-change* for a framework version is the sum of units-changed in all classes between a version and its predecessor. In table 5, the 11 class metrics obtained for the framework versions are presented. To assess the significance of the total-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 6, the total-extent-of-change is presented relative to the maximum possible change, we name the metric *relative extent-of-change*. Note that the data from the OWL framework was obtained from [5], which presented data for all OWL

versions including the minor versions 4.5 and 5.2. Thus, we are not able to present the changes in class characteristics data since we are only considering major versions. This because the change class characteristics data is computed between two successive major versions in our study.

Table 5: Changes in class characteristics and changes in metric values considered when a class is present in both of the compared versions

<i>Metrics</i>	<i>BGW 1/2</i>	<i>BGW 2/3</i>	<i>BGW 3/4</i>	<i>BGW Total Units Changed</i>	<i>BGW % Change</i>	<i>MFC 1/2</i>	<i>MFC 2/3</i>	<i>MFC 3/4</i>	<i>MFC 4/5</i>	<i>MFC Total Units Changed</i>	<i>MFC % Change</i>
NOC	19	31	47	97	2.4	7	10	12	12	41	1.8
NOA	47	13	10	70	1.7	18	11	5	0	34	1.5
DCC	89	80	141	310	7.5	29	46	77	23	175	7.7
NOM	148	142	198	488	11.9	68	84	126	120	398	17.5
NOP	105	93	117	315	7.7	64	81	119	12	276	12.1
NPT	77	125	187	389	9.5	68	87	124	93	372	16.3
NPM	147	156	190	493	12.0	68	84	124	117	393	17.2
NOD	78	126	144	348	8.5	49	47	83	25	204	8.9
NAD	69	51	82	202	4.9	6	10	21	5	42	1.8
NRA	76	88	58	222	5.4	14	38	66	15	133	5.8
CSB	77	125	187	389	9.5	48	48	90	26	212	9.3
Totals	1 231	1 246	1 633	4 114	100.0	439	546	847	448	2280	100.0

MFC Observations

In table 5, we see that the NOA (number-of-ancestor) metric changed values for 18 classes between version 1 and 2 of the MFC framework. The reason for this large change in the metric values was the restructuring of the MFC class hierarchy which introduced several new classes in the middle of the hierarchy [3, 24]. We can also see that the majority of the changes are related to methods and method parameters i.e. the functionality suit of metrics. For example, the number-of-methods (NOM) 17,5%, number-of-polymorphic-methods (NOP), 12,1% and average number-of-parameters per method (NPM), 17,2%.

BGW Observations

For the BGW framework we observe the same pattern, the majority of changes belong to the functional suit of metrics. For example, the number-of-methods (NOM) 11,9%, the average number-of-parameter types (NPT) 9,5% and average number-of-parameters per method (NPM), 12,0%. An exception is the number-of-polymorphic-methods (NOP) which only represent 7,5% of the total changes. A possible explanation is that the BGW framework is intentionally designed as a black-box framework.

4.2 Hypotheses

The Relative Extent of Change metric

The relative-extent-of-change metric is in one way similar to the normalized-extent-of-change metrics since it intends to capture the stability of the framework. On the other side, the relative extent-of-change metrics is on another abstraction level since it is composed of a set of finer grained class metrics and is not directly measured on the architectural level of the framework. As mentioned earlier, the relative extent of change metric is computed by dividing the sum of actual number of changes for all classes between two successive framework versions with the theoretical maximum of possible changes. In table 6, the Relative-Extent-of-Change metric values are presented. For example, version 1 of the BGW framework has 322 classes which gives a theoretical maximum of $322 * 11$ (number of metrics that can change for a class) = 3 542 possible changes for the (previous) version. The actual number of changes between version 1 and version 2 of the mediation framework is 932 which represents a $(932 / 3 542) * 100 = 26,3\%$ change between version 1 and 2. Thus, the Relative-Extent-of-Change metric is 26,3% for version 2 of the

BGW framework. Once again, note that the data from the OWL framework is not presented since we are only considering major versions of frameworks.

Table 6: Relative-Extent-of-Change Metric values

	<i>BGW1</i> /2	<i>BGW2</i> /3	<i>BGW3</i> /4	<i>Total</i> <i>BGW</i>	<i>MFC1</i> /2	<i>MFC2</i> /3	<i>MFC3</i> /4	<i>MFC4</i> /5	<i>Total</i> <i>MFC</i>
Actual Units Changed	932	1030	1361	3323	439	546	847	448	2280
Max. units of Change	3542	4895	5885	14322	792	1012	1452	2266	5522
<i>Relative Extent-of-Change</i>	26,3%	21,0	23,1%	23,2%	55,4%	54,0%	58,3%	19,8%	41,3%

Hypothesis 5: A stable framework exhibits a relative-extent-of-change value of less than 25%.

We see that the BGW framework despite its size exhibits a relative-extent-of-change value just above 25% for the first transition, version 1 to version 2. For the successive transitions the values are below 25%. In the case of the MFC framework it begins at a level above 55% for the first transition and continues to be on a level above 50% until version 5 when a large drop occurs down to 19,8%. Based on the observations of the frameworks' behavior we formulate the following hypothesis "A stable framework exhibits a relative-extent-of-change value of less than 25%".

Hypothesis 6: It takes the development and release of three to five versions to produce a stable framework.

This hypothesis can be formulated by studying for which versions of the three frameworks the other five hypotheses are true. In table 7, the five hypotheses and the versions for the three frameworks are listed. An 'X' indicates that the hypothesis is fulfilled and a '-' that the hypothesis is not fulfilled. Note that the OWL framework is not fulfilling any of the five hypotheses. The OWL framework has an NSI/DSC ratio below 0,8 but since it is using multiple inheritance heavily it is not fulfilling hypothesis 2.

Table 7: Hypotheses 1 to 5 fulfillments

	H1	H2	H3	H4	H5		H1	H2	H3	H4	H5
BGW1	-	X	-	-	-	MFC1	-	-	-	-	-
BGW2	X	X	-	-	-	MFC2	-	-	-	-	-
BGW3	-	X	X	X ¹	X	MFC3	-	-	-	-	-
BGW4	X	X	X	X	X	MFC4	X	X	X	-	-
						MFC5	X	X	X	X	X
OWL4	-	-	-	-	-						
OWL5	-	-	-	-	-						

1) The normalized extent of change values was 0,41 for the version

In table 7 we see that the BGW framework fulfills hypotheses 1 and 2 for version 2 and all but hypothesis 1 for version 3. Version 4 of the BGW framework fulfills all the hypotheses. For the MFC framework the situation is that it fulfills hypotheses 1, 2 and 3 for version 4. For version 5 of the MFC framework the additional hypotheses 4 and 5 are fulfilled too.

To summarize, the BGW framework starts fulfilling hypotheses to a large extent from version 3, the MFC framework from version 4 and the OWL framework does not fulfill any of the hypotheses. Thus, we can formulate the following more general hypothesis "It takes the development and release of three to five versions to produce a stable framework".

5 REFLECTION

In this section we discuss some reflections related to the stability assessment method used in this paper and how it differs from the original approach proposed in [3], the applicability of the Normalized-Extent-of-Change metric and what kind of differences in the frameworks the assessment methodology can handle.

We have applied a framework stability assessment methodology on three different framework. Differences from the original assessment procedure are:

- The outcome from three different frameworks have been analyzed in group and not in one single isolated case
- We have been able to formulate a set of hypotheses since we have analyzed the outcome from three different frameworks
- We have added an additional metric, the Relative-Extent-of-Change metric based on individual class level metrics, which is on another abstraction level than the original normalized-Extent-of-Change metric.

We have validated the original approach by applying the method three times and the method does not show any particular weakness. In our study we have used the original proposed set of metrics. There exist a possibility to change the metric set in the assessment procedure but in our study we have chosen to use the original set.

The introduction of the Relative-Extent-Of-Change metric gives us the possibility to compare the Normalized-Extent-of-Change metric to other extent of change metrics. In table 7, we see that the hypothesis based on the Relative-Extent-of-Change metric, hypothesis 4, indicates framework stability for version 3 and 4 for the BGW framework and for version 5 for the MFC framework. The architectural metric, Normalized-Extent-of-Change, used in hypothesis 5 indicates framework stability for version 3 and 4 for the BGW framework and for version 5 for the MFC framework. Thus, we see that the Normalized-Extent-of-Change metric indicates stability in the same version as the Relative-Extent-of-Change metric indicates. The conclusion is that the Normalized-Extent-of-Change metric is a good indicator of framework stability.

Using the six formulated hypothesis we found that both the BGW and MFC frameworks have reached stability. This shows one strength of the method since it seems to cope with framework dissimilarities. We describe a four major differences between the MFC and BGW framework that are invariant to the stability assessment method.

The MFC framework is a commercial available white-box object-oriented framework with an “anonymous” market whereas the BGW framework is a proprietary 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 has to deal with explicit customers and customer requirements rather than distributing the framework to a mass market.

Second, the BGW 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 domains covered by the frameworks are quite 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 incorporate impressive amounts of requirement changes and additions during its life-time.

6 RELATED WORK

Recent work by Lindvall and Sandahl [19] 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% 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 which can be seen as the opposite of identifying stability.

Recently, Gall et al. [11] proposed an approach for identifying potential modules for restructuring in large software systems using product release history. The aim of their method is to identify evolution-prone modules using historical version numbering of the modules. In our study we are working with smaller entities, classes, than modules but eventually their approach can be adapted to work on that class abstraction level.

Other research related to framework evolution is the work by Roberts and Johnson [23] and by Erni and Lewerentz [9]. 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 and which parts that become stable.

Erni and Lewerentz describe a metrics-based approach which 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.

7 CONCLUSIONS AND FURTHER WORK

Object-oriented frameworks provide an effective means for achieving reuse of software. A framework provides a reusable asset, which constitutes the basis for a number of similar future applications in the application domain captured by the framework. Especially flexibility and generality of the framework are important to achieve since it is intended to be used for several applications. However, it has proven to be both difficult and expensive to achieve these requirements [23, 16]. Consequently, it requires the development of a number of versions of a framework before it achieves a satisfactory maturity.

Assessment of the maturity of an object-oriented application framework is important since evolving frameworks cause high maintenance cost for the applications built based on the framework. Especially when the interface of the framework is changed in the new version or when the framework was adapted for the application incorporating it, the cost of incorporating the new framework may be substantial. Although framework maturity covers many aspects, our experience is that structural and behavioral framework stability provides a good, objective indicator of maturity. Few framework stability assessment methods exist, but one example is the approach proposed by Bansiya [3, 4].

Using an assessment method adapted from Bansiya's, we propose six hypotheses based on the data several versions of three object-oriented frameworks, i.e. four versions of Ericsson Software Technology's billing-gateway (BGW) framework, five versions of the Microsoft Foundation Classes framework and four versions of Borland's Object Window Library framework. We collected the data for the BGW framework, whereas the data for the other frameworks was taken from [4]. The framework stability hypotheses are:

1. Stable frameworks tend to have narrow and deeply inherited class hierarchy structures, characterized by high values for the average depth of inheritance (above 2.1) of classes and low values for the average width of inheritance hierarchies (below 0,85).
2. A stable framework has an NSI/DSC ratio just above 0.8 if multiple inheritance is seldom used in the framework. I.e. the number of subclasses in a stable framework is just above 80 percent.
3. The normalized ADCC metric is going towards 1.0 or just below for stable frameworks.
4. The Normalized-Extent-of-Change metric is below 0.4 for a stable framework.
5. A stable framework exhibits a relative extent of change value less than 25%.
6. It takes the development and release of three to five versions of a framework to produce a stable framework.

These hypotheses are supported by the data from the framework versions that we had access to. However, qualitative analysis is necessary to complement the metrics-based stability assessment. For instance, although four versions were available of the OWL framework, it turned out that two of these were minor updates. Incorporating these minor updates in the data analysis leads to incorrect conclusions. Nevertheless, we believe that the framework stability assessment method and the framework stability hypotheses discussed in this paper provide a useful and industrial applicable approach to assessing the stability of object-oriented frameworks.

As part of future work, we intend to collect data from more object-oriented frameworks in order to validate our hypotheses. In addition, we encourage the research community to do this as well.

ACKNOWLEDGMENTS

We would like to thank Ericsson Software Technology for providing us access to the BGW 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, 'Evaluating Application Framework Architecture Structural and Functional Stability' 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>, February 1998
4. J. Bansiya, 'Assessment of Application Framework Maturity Using Design Metrics,' Accepted for publication in *ACM Computing Surveys - Symposia on Object-Oriented Application Frameworks*, February 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 Transactions 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. 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.
12. E. Gamma, *Object-Oriented Software Development based on ET++: Design Patterns, Class Library, Tools* (in German), Springer-Verlag, 1992
13. E. Gamma et al., *Design Patterns: Reusable Elements of Object-Oriented Design*, Addison-Wesley, 1995
14. D. Geary, *Graphic Java 1.1 : Mastering the AWT*, Prentice Hall, 1997.
15. 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.
16. R.E. Johnson, V.F. Russo, 'Reusing Object-Oriented Design,' *Technical Report UIUCDCS 91-1696*, University of Illinois, 1991.
17. 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 Association Canadienne Francaise pour l'Avancement des Sciences*, Montreal, Canada, May 1994.
18. W. Li, S. Henry, 'Object-Oriented Metrics that Predict Maintainability,' *Journal of Systems and Software*, Vol. 23, pp. 111-122, 1993.
19. Lindvall, M. and Sandahl, K. (1998). How Well do Experienced Software Developers Predict Software Change? *Journal of Systems and Software*, 43(1):19 – 27.
20. M. Mattsson, 'Object-Oriented Frameworks - A survey of methodological issues,' *Licentiate Thesis*, LU-CS-TR: 96-167, Department of Computer Science, Lund University, 1996.
21. 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.
22. S. Moser, O. Nierstrasz, 'The Effect of Object-Oriented Frameworks on Developer Productivity,' *IEEE Computer*, pp. 45-51, September 1996.
23. 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.
24. G. Shepherd, S. Wingo, *MFC Internals: Inside the Microsoft Foundation Class Architecture*, Addison-Wesley, 1994.
25. K.J. Schmucker, *Object-Oriented Programming for the Macintosh*, Hayden Book Company, 1986.
26. S. Sparks, K. Benner, C. Faris, 'Managing Object-Oriented Framework Reuse,' *IEEE Computer*, pp. 53-61, September 1996.