ISSN No. 0976-5697

Volume 8, No. 5, May-June 2017

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Metric Models for Object Oriented Frameworks

Manjari Gupta
Department of Computer Science
Institute of Science
Banaras Hindu University
,

Abstract: Software Reuse by frameworks attempts to capture, document and specify the architectural design experiences in form of a semi-code. It seems that high level reuse by design patterns and frameworks are more promising than other reuse techniques. We define a possible framework for metric models for object-oriented frameworks that can be used by an organization easily and further newer metric models can be added herein easily. The whole thing is divided into Metric models for a domain, Metric models for a framework for a domain, Metric models for applications (wherein this framework's reuse is expected) and Metric models to evaluate the success of framework-based development.

Keywords: Metric, object oriented framework, domain reuse

1. INTRODUCTION

The idea of Formal Software Reuse, as first introduced by Mcllroy, was proposed at the NATO Software Engineering Conference in 1968 [1]. In the earlier days of software engineering (at the time of structured programming), programmers began developing standard block of code to perform operations like printing, and then copied and pasted that code into every application they wrote. While this reduced the application time for new applications, it was difficult if a change was needed in that block of code, because the developer had to make change everywhere that code had been copied. Object-oriented programming helps to reuse within application by avoiding repeating same code more than one [2]. Later the need to formally reuse across applications, with not much effort, is realized. Object-oriented programming does not provide a straight forward method to reuse across applications. This paradigm is "put together a bunch of objects, and then just focus on specific application". To collect and arrange these objects is another time-taking activity. Object-oriented frameworks were introduced to solve it. An object-oriented framework is the whole architecture that can be reused in many similar applications.

Metrics play a central role in any software development. It is essential to develop and use metrics to predict, evaluate and hence to improve the software development process so that it may result in acceptable quality products. Chidamber et. al. [6] proposed metrics to quantify the characteristics of object-oriented design. Several reuse metrics [7-11] are also presented in the literature but unfortunately it is difficult to find metrics for framework reuse. To make framework development, as well as its reuse process clear and understandable, it is necessary to quantify these processes and the products resulted by them.

It is essential to understand various characteristics and qualities of products, processes, project and people, in software engineering. Metrics play a central and vital role in software engineering. Frameworks must be highly reusable so as to be able to obtain the benefits promised by reuse. To make framework-based software development successful, it is necessary to quantify its characteristics. In this paper, we propose some metric models relevant to framework reuse technology.

In the next section we discuss briefly metric models for Framework-Based Software Development. Section 3 describes in detail metric models for a domain. Metric models for frameworks to be used in a domain and for applications to be developed using frameworks are explained respectively in section 4 and 5. To evaluate the success of framework-based development we proposed metrics in section 6. Finally we conclude in section 7.

2. METRIC MODELS FOR FRAMEWORK-BASED SOFTWARE DEVELOPMENT

In spite of the importance of frameworks, a widely accepted set of measures to quantify its characteristics has not been established. To make framework-based software development successful, it is necessary to do quantitative analysis of cost/benefit of frameworks. We have tried to address the question of metric development for software frameworks. Obviously, the reusability of a framework is an important issue**.** One should be able to quantify the reusability of, and the benefits promised by, a framework being developed. It is essential to consider various factors that affect these characteristics of interest.

In this paper, some metric models for frameworks are presented that can help organizations develop a business case to support the early development and easy reuse of frameworks. We, here, consider metric models that may be useful, early in the software life-cycle for estimating framework-based development benefits and after that for calculating the benefits that actually resulted from using this framework. Metrics for a framework must be different from other proposed reuse metrics because these metrics assume Reused Source Instruction (RSI) to count as reuse while, framework is not only the code, and thus the same cannot be used in this context.

Several metrics have been defined for reuse by different researchers. Many of them are having the same meaning but proposed the metrics in different ways. Thus, we can say the metrics for reuse proposed in the literature have redundancy. Because of this redundancy and gap it is very difficult for an organization to follow a set of metrics with confidence to measure their reuse effort and their benefits. Thus, first of all, we define a possible framework for metric models for an object-oriented framework that can be used by an organization easily and further newer metric models can be added herein easily. The whole thing can be divided into following categories:

Metric models for a domain: these metric models help in deciding for which applications in a domain, development of frameworks would be useful. That is, this sub-area contains metric models that help in identifying the reusability potential in a domain.

Metric models for a framework for a domain: these metric models would help in finding the quality related attributes of a framework.

Metric models for applications (wherein this framework's reuse is expected): these metric models help in finding the quality of applications that have been developed using a framework.

Metric models to evaluate the success of framework-based development: metric models in this last section help in comparing traditional versus framework-based software development to get the economic benefits of framework-based software development over traditional development.

These metric models are described in detail in the following sections.

3. METRIC MODELS FOR A DOMAIN

As mentioned earlier, these metric models would help framework developers to decide whether developing a framework for similar applications in a domain would be profitable or not. An application domain is modeled by analyzing the common and variant aspects of the family of applications in the domain. To decide it, one should analyze scope of various reusability types in a domain. Further, it is needed to decide about the structure of a framework, to be developed, by considering its size and complexity.

3.1 SCOPE OF VARIOUS REUSABILITY TYPES IN A DOMAIN

As said above, a domain may have common as well as variant aspects applicable in various applications for some purpose. Variant aspects in a family of applications may be:

1. Different functional requirements of similar applications in a domain.
2. Different behavioral requirements of similar applications in a domain.
3. Different languages in which similar applications, in a domain, are required to be developed.
4. Different environment in which similar applications, in a domain, are required to be used.

Many other variant aspects may also be there. We can define reusability index by concentrating on the following terms regarding some possible application or software system in a domain:

Total number of reusable aspects (N_{Tra}),
Number of variable aspects (N_{Va})
Total number of aspects (N_{Ta}) = $N_{Tra} + N_{Va}$
Number of common reusable aspects (N_{Cra}),
Number of variant reusable aspects (N_{Vra}),
Total number of reusable aspects (N_{Tra}) = $N_{Cra} + N_{Vra}$
Reusability of a domain (R_d)

$$R_d \propto N_{Cra}$$

$$R_d \propto \frac{1}{N_{Vra}}$$

Reusability (for reusable aspects)

$$= k \times \frac{N_{Cra}}{N_{Tra}}$$

$$\text{Reusability Index} = k \times \frac{N_{Cra}}{N_{Tra}} \times \frac{N_{Tra}}{N_{Ta}} \times \frac{1}{N_{Vra}}$$

$$= k \times \frac{N_{Cra}}{N_{Ta}} \times \frac{1}{N_{Vra}}$$

3.2 Monolithic Frameworks versus Multi-Framework Arrangements

Here, in this section, we address the question of heaviness of a framework because of multiple activities, in similar applications of a domain, being performed. In such complex skeleton structures there may be sub structures that merit to be considered as separate candidates for development of a framework for the activity that it carries out. This consideration would help us in considering the over all skeleton structure as a framework that itself may contain some other frameworks. The overall skeleton structure, now, becomes a system and the sub-structures may be designed and implemented as subsystems. The traditional method of a rigid framework development, with no frameworks for its substructures, would be known as a monolithic framework. Such a framework would be rigid as the substructures would also be fixed up to substantive extent. In a non-monolithic framework arrangement, multiple frameworks may be designed and implemented for the internal different activities whenever they can be represented by separate control abstractions. Any change in these structures can be separately managed rather newer frameworks developed for these purposes can always be taken up without disturbing the overall framework.

To be able to decide about inclusion of some activity as part of the framework definition, one should consider whether it can be defined once and reused many times. If the activity has to be performed in many different ways for various applications, then there is no point in accommodating it in the framework. Thus,

$$\text{Consideration of an activity in a main framework} \propto \frac{N_S}{N}$$

where, N_S is the number of similar ways in which this activity can be performed and

N is the total number of ways this activity can be performed.

4. METRIC MODELS FOR A FRAMEWORK TO BE USED IN A DOMAIN

In this area, we define one of the most important metrics for judging a framework; reusability of a framework metric. Other metrics in this section are understandability, complexity, customizability of a framework etc.

4.1 Reusability of Frameworks

As for any general software product, the reusability of a framework would be based on four things:

$$R_{Fr} \propto f(C, E, C_R, P)$$

where, C is the commonality among applications belonging to a domain, for which this framework is developed,
E is the efficiency of the framework,
C_R is the rarely changed code in the framework and
P is the portability of the framework.
Three types of things are reusable in a framework:
1. Reuse of code (R_C)
2. Reuse of abstract code (R_{AC}) and
3. Reuse of non code portion (R_{NCP})
First type of reuse is better than the second type of reuse which is better than the third type of reuse. But, first type of reuse makes a framework rigid while second and third type of reuse leads to more flexible framework. Thus, there should be balance among these types of reuse in a framework.
Further, a framework defines the whole architecture of several similar applications in a domain. As described earlier, different applications may have some variability in their architecture. By generalizing these different architectures the framework is developed. But some times it is hard to develop generalized architecture without some pre-assumptions. These pre-assumptions make it difficult to reuse a framework in many applications. Hence, a framework's reusability can also be defined as

$$R_{Fr} \propto \frac{1}{\mathrm{N_A}}$$

where, N_A is the number of architectural assumptions in a framework.
If certain assumption(s) is (are) not valid/ true for some application, then it would be difficult to deploy this framework in that case.
Another way, as for any general reusable asset, of specifying reusability of a framework is constrained by the extent of new code that needs to be written by the application developers at the time of its instantiation and the customization required in these applications. That is,

$$\mathrm{R_{Fr}} \propto \frac{1}{\mathrm{NCQ}} \times \frac{1}{\mathrm{CCQ}} \times \frac{1}{\mathrm{E_M}}$$

where, NCQ is the new code quantity, CCQ is the code quantity that needs to be customized and E_M is the extent of (and type of) these modifications required.

4.2 Customizability of Frameworks
The most common way to instantiate a framework is to inherit from some abstract classes defined in the framework hierarchy and write the code that is called by the framework itself [3]. Thus, customizability of a framework (F_C) would be good if it is easy to identify which code and where this code should be written. One of the factors that hinder in identifying this information is the complexity of a framework's class hierarchy. Thus, we can say

$$F_C \propto \frac{1}{\text{Complexity of a Framework}}$$

A requirement, that is similar in most of the applications, would be a good candidate to be included in a framework as it would increase the customizability. More the number of such requirements more would be the customizability of such a framework. That is, if a requirement, similar in more number of applications, is addressed in a framework, it would be easy to customize it in those many instantiations. By considering each similar requirement addressed in a framework, we can say that

$$F_C \propto \sum_{i=1}^{n} N_{A_i}$$

where, for a requirement addressed in a framework, N_A is the number of applications having this requirement in slightly different forms and
n is the number of similar requirements addressed in a framework

4.3 Usability of Frameworks
In the case of frameworks, usability is always reusability. It is the ease in identification, understanding and deployment of a framework. As in general software products, users may not (possibly do not) require to understand its design and design decisions. While, in case of frameworks, its users must not only understand the internal design but also understand the philosophy and reasoning of the design decisions taken. As the general software product that needs only to be compiled and run, it needs to be understood and translated into a specific software architecture implementation.
Such a metric model should address the question of framework requirements to be identified by the user and consequent selection of the same for deployment. If it is difficult to understand the scope and functionality of the framework and it demands a good deal of modification in the framework and the software being developed, then the (re)usability of the framework is poor. Thus,
Usability = f (Ease in identification, Ease in understanding, Ease in deployment)
Difficulty in identification can be defined as follows,

$$\text{Difficulty in identification} = \frac{N_M}{N}$$

where, N_M is the number of mismatches between a framework specification and architectural requirements of the application and,
N is the total number of architectural requirements of an application.
Difficulty in deployment can be defined as follows,

$$\text{Difficulty in deployment} = \sum_{n} \frac{C_i}{S_i}$$

where, C_i is the customizability of i^{th} element,
S_i is the size of i^{th} element,
And n is the total number of such elements in a framework.
Understandability of a framework is described later in this section.

4.4 Portability of Frameworks (P)
As for general software products, this metric attempts to capture the requirement of a framework being language and/or software architecture independent. The frameworks that are written for a fixed language or environment may not be portable at all, whereas a framework becomes highly portable if it can be deployed in diverse situations in various applications. Thus,

P $\propto$ The number of direct ("As-is") deployments / the total number of applications wherein such a framework has potential of being deployed.

Or we can say,

$$P \propto \frac{NS_{LP}}{NR_{LP}} \quad \Rightarrow$$

$$P = k\,\frac{NS_{LP}}{NR_{LP}}$$

where, NS_{LP} is the number of languages and platforms on to which the framework can be ported,
NR_{LP} is the number of languages and platforms that such a framework may be required to be ported, and
k would be constant that will specify the constraints imposed by status of the technology.

4.5 Complexity
As for any general software product, the complexity of a framework can be defined as follows:

Complexity = f (structure, content, size)

Since, frameworks are built to be more flexible so that they can accommodate changing requirements, their complexity cannot be measured. Apart from domain classes we always have in a system control flow (application logic) objects/classes. These application logic objects hide the complexity of the control flow into these application logic classes. That is why an object-oriented framework will be easy to understand as the domain objects, application objects, interface objects, utility objects and their interaction would be easily understandable.

Another way of defining the complexity of a framework (C_{Fr}) may be by considering the complexity of domain, control logic, utility and interface classes along with the complexity of their interaction. That is,

$$C_{Fr} \propto f(C_{DC}, C_{CLC}, C_{UC}, C_{IC}, C_{FA})$$

where, C_{DC} is the complexity of domain classes of the framework,
C_{CLC} is the complexity of control logic classes,
C_{UC} is the complexity of utility classes,
C_{IC} is the complexity of interface classes, and
C_{FA} is the complexity of framework architecture that is the complexity of the interactions among different parts of a framework.

4.6 Understandability (U_n)
As described earlier, frameworks support reuse by having aspects that can be reused "as-is", aspects that need customization and some material that guide how to add application specific aspects consistently so that reuse of it can be done as intended by a framework developer. Thus, to reuse a framework it is required to understand the above said. As a framework contains both, design and semi-code, its understandability will depend on the understandability of both. Thus,

$U_n \propto$ design clarity,
$U_n \propto$ document clarity.

Further, as for any general software product, the understandability of a framework would be inversely proportional to its complexity. That is,

$$U_n \propto \frac{1}{\text{Complexity}},$$

The complexity of a framework is defined earlier in this paper. And hence,

$$U_n \propto \frac{\text{design clarity} \times \text{document clarity}}{\text{complexity}}$$

Further, clarity = $\frac{1}{\text{ambiguity}}$

The total clarity (of design and document) can be found by reducing ambiguity.
If we define the ambiguity factor as follows:

Ambiguity factor = $\frac{N_{AM}}{N}$

where, N_{AM} is the number of specified items that can have more than one interpretation,
N is the total number of specified items in the framework. Hence,

$$\text{Design clarity} \propto \frac{1}{\left(\frac{N_{AM_{Design}}}{N_{DesignI}}\right)} \quad \Rightarrow$$

$$\text{Design clarity} \propto \frac{N_{DesignI}}{N_{AM_{Design}}}$$

where, $N_{AM_{Design}}$ is the number of items in a framework's design that can have more than one interpretation and
$N_{DesignI}$ is the total number of specified items in a framework's design.
Similarly,

$$\text{document clarity} \propto \frac{1}{\left(\frac{N_{AM_{Document}}}{N_{DocumentI}}\right)} \Rightarrow$$

$$\text{Document clarity} \propto \frac{N_{DocumentI}}{N_{AM_{Document}}}$$

where, $N_{AM_{Document}}$ is the number of items in a framework's document that can have more than one interpretation and
$N_{DocumentI}$ is the total number of specified items in a framework's document.

Maintainability, testability and reliability etc. are other interested quality attributes that may be defined in the context of framework reuse.

5. METRIC MODELS FOR APPLICATIONS

Under this section metric models that quantify the quality of applications, that are developed using frameworks, are kept. The enhancement in each element of the quality (Q_{Enh}) of applications developed using frameworks can be expressed by:

$$Q_{Enh} \propto \frac{Q_{WF}}{Q_{WOF}}$$

where,
Q_{WF} is the quality factor of an application developed with framework,
and Q_{WOF} is the quality factor of an application developed without using a framework.
Quality of an application, i.e. of any software, has been rigidly described by McCabe. This model has been extensively referenced for non-reuse based software development. Whether it needs to be redefined or some newer ones are required is the basic question that needs to be addressed.

5.1 Reliability of applications

The reuse of frameworks will enhance the reliability of an application due to the obvious fact that a software skeleton structure will work properly, if it has already worked for someone else. The reason is that framework-based applications would have many parts that have been rigorously exercised and verified in the previous development of applications developed using a framework.
As in general,
Software Reliability = f (number of bugs in the application, profile of execution)

The possibility of introduction of new bugs increases during customization and deployment of a framework because of need of writing newer code. Thus, the reliability of an application (A_{RL}), developed using framework, would be proportional to number of 'as-is' reused aspects (N_{ARA}) and it would be inversely proportional to number of aspects that need customization (N_{CA}) and newly developed aspects (N_{NDA}) because new code writing may introduce bugs. Thus,

$$A_{RL} \propto N_{ARA} \times \frac{1}{N_{CA}} \times \frac{1}{N_{NDA}}$$

It only shows that software reliability models will have to consider this situation.

5.2 Complexity of Applications

An application, based on a framework, not only reuses the frameworks source code, but also its architectural design. This amounts to a standardization of the application structure, and allows a significant reduction in the size and complexity of the source code that has to be written by developers who instantiate a framework [4]. By modularization of the architecture, a framework manages the complexity of a solution architecture and consequently the complexity of the applications also gets reduced. Since a framework is the main architecture, of such applications, that calls application specific code which is generally in less quantity; one can describe the complexity of these applications (A_C) in terms of complexity of frameworks (F_C) that they use. The complexity of a framework interface may complicate the interaction among newly developed code and the framework. Thus, we can say

$$A_C \propto F_C$$

5.3 Testability of Applications

Testability of an application, developed using a framework, due to use of standard architecture, would be high. Testability is inversely proportional to testing effort. Testing of an application requires testing of an application specific parts, along with the testing of their integration with framework. To test this integration, much of the test suits developed to test the framework used, would be reused and some test cases need to be extended. That is, unit testing of application specific aspects and integration testing of these aspects with frameworks (because of dependency of these aspects on prewritten aspects of the framework) and system testing to test overall requirements of the application are needed. Thus, testability of such applications (T) can be estimated by calculating the size of "as-is" reusable part of framework (F_{AS}) and the total size of the application (A_{TS}).

$$T \propto \frac{F_{AS}}{A_{TS}}$$

Other metric models like understandability, maintainability, portability etc. for applications, wherein this framework reuse is expected, may also be developed on the similar line.

6. METRIC MODELS TO EVALUATE THE SUCCESS OF FRAMEWORK-BASED DEVELOPMENT

These metric models are basically used by managers. Managers primarily need to know if reuse results in improved productivity and in reduction of program risks such as cost, quality and schedule overrun etc. They may want to check whether the quality of software development process using frameworks reuse has enhanced or not, what are the economic effects of this reuse etc. We require to have Cost/Benefit Estimation models and Investment analysis models. The following observations (drawn in this section) are possible in the context of cost/benefit analysis.

Managers are always interested to know the productivity of developers, whether it is enhanced by framework reuse or not, whether the software development process has become difficult or easy because of framework reuse etc. In this sub section we consider the requirements for these three metric models in context of framework reuse.

6.1 Productivity enhancement of developers

Productivity, most often refers to the relationship between inputs and outputs, is typically a ratio of outputs to a single input such as lines of code per person/day [4]. In FBSD, we need to concentrate on the productivity of framework developers and application developers (who develop it using a framework). Productivity of framework developers would be improved as long as they would get confidence in a domain. One of the keys to improve productivity is to improve quality [5]. Since applications based on existing frameworks are of better quality than non-reuse based applications thus productivity would certainly improve. In any general reuse, there is confusion that whether the reusable code part should be considered in a developer's productivity or not. In framework reuse, generally, a developer needs to learn the framework functionality; and thus the productivity of application developers should include the size of framework also.

6.2 Difficulty in Development

To analyze difficulty in FBSD, we need to concentrate on the difficulty in framework development and further difficulty in developing an application by using frameworks. Extra difficulty in framework development comes because the scope for estimated vertical requirements of different applications (which may be developed by reusing it) need be left for developers to write. Further, the need of documentation is there to explain how to use a framework for development of applications (so that it may not be used wrongly) also makes the process of framework development more difficult. Developers may feel certain limitations in adjusting their requirement with that of architectural specifications of the framework to be deployed for this purpose. Framework designers may propose a framework considering this difficulty.

6.3 Development Time Reduction

Development time (T_{AD}) of applications using framework would always be proportional to number of reusable components in an application.

Once the architectural design of a framework has been developed, the time to develop an application based on that framework could be estimated very early.

$$T_{AD} = m \times F_C + n \times F_S + c,$$

where, T_{AD} is the estimated time for completion of an application wherein this framework reuse is expected,

F_C is the number of aspects in the framework that need to be customized,

F_S is the number of aspects specific to the application,

m, n and c are constants that can be estimated by analyzing other applications' development time that have been developed using the same framework.

7. CONCLUSION

We have tried to address the question of metric development for software frameworks. Some metric models have been presented, for frameworks, which can help organizations develop a business case to support the early development and easy reuse of frameworks. Several metrics have been defined for reuse by different researchers. Many of them are having the same meaning but have proposed the metrics in different ways. Thus, we can say the metrics for reuse proposed in the literature have redundancy. Because of this redundancy and gap it is very difficult for an organization to follow such a set of metrics with confidence to measure the reuse effort and benefits. Thus, first of all, we defined a possible framework for metric models for an object-oriented framework that can be used by an organization easily and further newer metric models can be added herein easily.

REFERENCES

1. Smolarova M. and Navrat P., Software Reuse: Principles, Patterns, Prospects, Journal of Computing and Information Technology, Vol. 5, No.1, 1997, Page(s) 33-49.
2. Boggs W. and Boggs M., Mastering UML with Rational Rose 2002, Sybex Inc., ISBN: 0-7821-4017-3.
3. Fontoura M., Braga C., Moura L. and Lucena C., Using Domain Specific Languages to Instantiate Object-Oriented Frameworks, IEE proc-Softw., Vol 147, No.4, August 2000, Page(s) 109-116.
4. Arthur L.J., Measuring Programmer Productivity and Software Quality, John Wiley & Sons, ISBN 0-471-88713-7, 1985.
5. Bieman J.M., Zhao J.X., Reuse Through Inheritance: A Quantitative Study of C++ Software, ACM SIGSOFT Software Engineering Notes**,** Volume 20, 1995, Page(s) 47-52.
6. Chidamber S and Kemerer C., Towards a Metrics Suite for Object Oriented Design, In Proc. OOPSLA, 1991, Page(s) 197-211.
7. Devanbu P., Karstu S., Melo W. and Thomas W., Analytical and Empirical Evaluation of Software Reuse Metrics, Proc. of the 18 Int. Conf. on Software Engineering, 1995, Page(s) 189 - 199.
8. Ferri R.N., Pratiwadi R.N., Rivera L.M., Shakir M., Snyder J.J. and Thomas D.W., Chen Y.F., Fowler G.S., Krishnamurthy B. and Vo K.P., Software Reuse Metrics for an Industrial Project, 0-8186-8093-8/97 IEEE, 1997, Page(s) 165-173.
9. Frakes W. and Terry C., Reuse Level Metrics, IEEE, 1994, Page(s) 139-148.
10. Frakes W. and Terry C., Software Reuse: Metrics and Models, ACM Computing Surveys, Vol. 28, No. 2, 1996, Page(s) 415-435.
11. Frakes W. and Terry C., Software Reuse and Reusability Metrics and Models, ACM Computing Surveys**,** Volume 28, Issue 2, 1996, Page(s) 415 – 435.