XIX IMEKO World Congress
Fundamental and Applied Metrology
September 6–11, 2009, Lisbon, Portugal

FLEXIBILITY EXPERIMENTAL TEST OF THE SOFTWARE FRAMEWORK FOR MAGNETIC MEASUREMENTS AT CERN

Pasquale Arpaia[1,2], *Marco Buzio*[1], *Vitaliano Inglese*[1,3]

[1] CERN, Dept. TE (Technology), Group MSC, Geneva, Switzerland
Marco.Buzio@cern.ch - Vitaliano.Inglese@cern.ch
[2] Department of Engineering, University of Sannio, Benevento, Italy
arpaia@unisannio.it
[3] Department of Electrical Engineering, University of Naples Federico II, Napoli, Italy

Abstract – The paper deals with the flexibility test of software frameworks for measurement applications, and, in particular, of the Flexible Framework for Magnetic Measurements (FFMM), developed at the European Organization for Nuclear Research (CERN) in order to satisfy the new magnetic measurement requirements and to provide a uniform platform to handle all magnetic measurement applications. FFMM is designed to be flexible, reusable, maintainable, and portable. As part of the characterization of the framework from the point of view of both software quality and performance, this paper presents a metric suitable for its flexibility characterization. Experimental results are also provided for typical application scenarios of FFMM.

Keywords: Accelerator measurement systems; Software development; Software flexibility.

1. INTRODUCTION

Flexibility, the modification easiness of a system or component for use in applications or environments different from the design [1], is definitely one of the most desirable properties of any system to face changes in operational environment during its life. This is particularly true for software systems, both because they are often subject to extremely rapid technological development, and because some of them are specifically conceived to be employed in environment spanning a wide range of functional requirements, not fully predictable at the design stage. Unfortunately, despite their importance, flexibility and more in general software quality, are often neglected in software design and development.

This paper presents the work related to software flexibility characterization carried out at the European Organization for Nuclear Research (CERN). In the past years, the test of the LHC (Large Hadron Collider) superconducting magnets brought to incremental development of software for magnetic measurements with very strict requirements, without focusing on its quality, namely flexibility and reusability. The end of the series test on the LHC magnets marked a change in the requirements: the need for more specialized measurement applications to be performed on small-medium magnet batches arose. As a consequence, a new platform was required to span all magnetic measurement applications, increasing the flexibility of the measurement stations, and facilitating changes in the hardware configuration and measurement conditions.

Conceptual work in this direction started in cooperation with the University of Sannio, by analyzing the state of the art [2]-[6], and subsequently by developing the Flexible software Framework for Magnetic Measurement (FFMM). The project was presented in [7] as an object-oriented framework, and further improved in [8] by exploiting an aspect-oriented approach, more widely discussed in specific papers [9]-[11]. Finally, a design of the framework kernel was presented [12]. However, now after prototyping and experimental applications, a comprehensive characterization of software performance, in terms of code quality and use flexibility is needed.

In this paper, a metric suitable for FFMM flexibility characterization, as well as experimental results of tests are illustrated for some typical application scenarios of FFMM.

2. PROBLEM

The FFMM is a software framework for magnetic measurement applications based on Object Oriented Programming (OOP), and Aspect-Oriented Programming (AOP) [13]. Its basic ideas and architecture are discussed in [7], [11]. In particular, FFMM aims at supporting the user in developing software for automatic measurement systems by maximizing quality in terms of flexibility, reusability, maintainability, and portability, without neglecting efficiency, vital in actual test applications. Moreover, the requirements for a wide range of magnetic measurement applications, such as needed for the test of superconductive magnets for particle accelerators, have to be satisfied.

In Fig.1, the FFMM architecture is illustrated. A test engineer (end user) produces a description of the measurement application, *User Script*, whose semantic

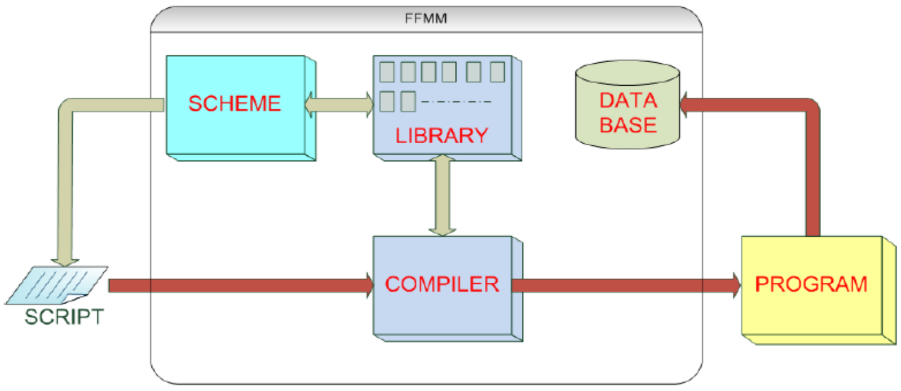

Fig. 1. The FFMM architecture.

and syntactic correctness is verified by the *Script Checker*. Then, from the *User Script*, the *Builder* assembles the *Measurement Program*, according to the architecture of the *Scheme* by picking up suitable modules from the *Software Module Library*. If some modules are not available in the library, a template is provided to the user (administrator user) in order to implement them according to a suitable predisposed structure. Once debugged and tested, the *Measurement Program* will be stored in the *Database* in order to be reused.

Even though the framework was designed to be flexible, reusable, maintainable, and portable, so far a comprehensive approach to the statement of the fulfilment of these project goals is still missing.

As part of a wider work aimed at the characterization of the framework from the point of view of both software quality and performance, the purpose of this paper is to quantify, through the introduction of suitable metrics, the degree of flexibility achieved by the current release 3.0 of FFMM.

3. PROPOSAL

Classic and contemporary literature in software design recognize the central role of flexibility in software design and implementation. Structured design, modular design, object-oriented design, software architecture, design patterns, and component-based software engineering, among others, seek to maximize flexibility.

During its life cycle, a software system is forced to face variable requirements. As a consequence, in the process of maintenance and improvement, often the implementation has to be adapted to provide a solution to problems in new application domains. An *evolution step* is defined as the unit of evolution with relation to a particular change in the implementation.

It has been observed that predicting the class of changes is the key to understanding software flexibility. During the phases of design and development of the software, initially the changes that are likely to occur over the lifetime of the product are characterized . Since it is impossible to predict the actual changes, the predictions will be about classes of changes [14].

The notion of evolution step can be used for estimating software flexibility [15]: *a* is more flexible than *b* towards a particular evolution step if the number of changes required for *a* is smaller than the number of changes required for *b*. Thus, the complexity of an evolution step measures how inflexible the implementation is towards a particular class of changes: the less changes are required, the more flexible it is.

It is therefore useful to organize the software so that the items that are most likely to change are confined to a small amount of code, so that if those things do change, only a small amount of code would be affected [14]. In other words, flexibility (measured in terms of the cost of the evolution process) is directly linked to the amount of code that is affected. Thus, a first approximation to measuring the cost of executing an evolution step ε is given by the evolution cost metric which counts the number of modules that are affected by ε. Under the assumption that the costs of adding, removing, or changing each modular unit commensurate, the *evolution cost metric* can be obtained by calculating the number of modules that were added, removed, or adjusted as a result of the evolution. This number is obtained by calculating the symmetric set difference between the sets of classes in the old (i_{old}) vs. the adjusted ($i_{adjusted}$) implementations. Formally [15]:

$$C_{Classes}(\varepsilon) = \left| \left(Classes(i_{old}) - Classes(i_{adjusted})\right) \cup \left(Classes(i_{adjusted}) - Classes(i_{old})\right) \right| \tag{1}$$

This evolution cost metric is inadequate in some situations: when the evolution of different modules do not commensurate, when the modules are not implemented yet, and when the programming language does not support classes at all or adds other programming units (as in the case of AOP). It is therefore necessary to accommodate the metric for varying degrees of modular granularity, as well as for varying degrees of information on each module. This leads to the definition of the *generalized evolution cost metric* [15]:

$$C^{\mu}_{Module}(\varepsilon) = \sum\nolimits_{m \in \Delta Modules(i_{old}, i_{adjusted})} \mu(m) \qquad (2)$$

where $\Delta Modules(i_{old}, i_{adjusted})$ is the symmetric set difference between the set of modules in i_{old} and the set of modules in $i_{adjusted}$. The generalized metric is parameterised by the variables *Modules* and μ: *Modules* represents any notion of module that is appropriate for the circumstances, such as class, procedure, method, aspect, and package; μ represents any software complexity metric that is meaningful with relation to a particular module m. Finally, it is to be pointed out that evolution complexity is a *measure of growth*, not an absolute value, and therefore it does not measure the actual cost of the evolution process but how it grows.

4. EXPERIMENTAL RESULTS

The proposed approach to flexibility assessment is applied at CERN in the context of the Flexible Framework for Magnetic Measurement [7]-[12]. The platform was designed in order to satisfy the requirements for a wide range of magnetic measurement applications, thus the most probable scenarios it will have to face are the different techniques currently used for the test of magnets for accelerator, besides those that could be developed in the future. As said before, the framework is based on OOP and AOP, therefore the modules involved in these scenarios are *methods*, *classes* and *aspects*.

In a preliminary analysis phase, the classes of changes due to the different measurement techniques were classified as: (i) adding/modifying software modules implementing the devices, (ii) changing the strategies for handling the services provided by the framework (fault detection, logging, synchronization), (iii) implementing new measurement algorithms. The abovementioned classes of changes involve different users of the framework, namely (i) and (ii) the developer and (iii) the test engineer.

In the following, some preliminary results of the flexibility assessment are illustrated. The tests were carried out at CERN on the release 3.0 of FFMM for different measurement methods. The experimental results are summarized in Tab. 1. The *generalized evolution cost metric* is obtained by fixing $\mu = CC$ (Cyclomatic Complexity [16], a measure of the number of linearly independent paths through a program's source code and therefore of its logical complexity), thus yielding the metric $C^{CC}_{Modules}$. This metric is used to compare the degree of flexibility of the different classes of changes, and not as an absolute measure of flexibility. A high cyclomatic complexity (>10 [17]) denotes a complex procedure that is hard to understand, test and maintain. Therefore, the lower the cyclomatic complexity (and consequently $C^{CC}_{Modules}$), the higher the flexibility.

4.1. Adding/modifying a device

When new devices are required by the measurement application, it is not possible to completely avoid the effort for their implementation. In this case the flexibility is therefore intrinsically limited. Nevertheless FFMM is fairly flexible towards this class of changes, since it helps the user effectively develop the new components. Namely, it provides services, such as event handling and fault detection [12], [18], whose infrastructure is easily accessible and whose implementation is customizable with limited effort.

The possible changes at device level can be classified as (i) adding the device into the framework from scratch, and (ii) modifying it to satisfy new requirement when it already exists, by adapting its interface or some method implementation.

The cost of adding a new device strongly depends on its size. Formally, rather than through the lines of code (LOC), this cost can be expressed as the sum of the cyclomatic complexity of all its methods, including additional code devoted to events and faults handling, and computed as the average cyclomatic complexity of a software unit (method) multiplied by the number of units implemented. The generalized evolution cost metric results therefore proportional to the number of member functions, events and faults of the new class (Tab. 1). The member functions of the class are likely to be more complex than the methods handling events and faults, thus the generalized evolution cost metric usually

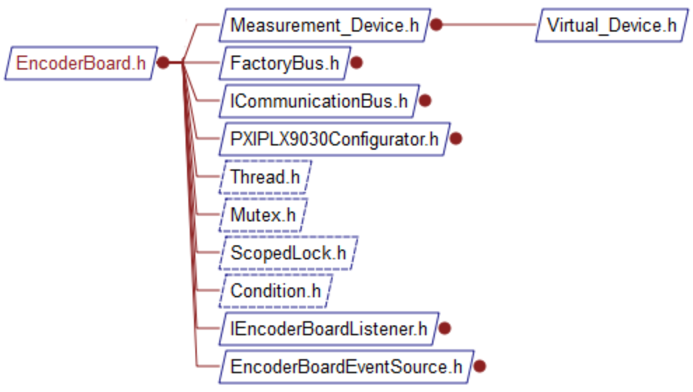

Fig. 2. Encoder Board class hierarchy.

Table 1. Generalized evolution cost metric for different classes of changes in FFMM.

Class of change	User involved	$C^{CC}_{Modules}$	Increasing flexibility
Add device	Developer	$\propto$ # methods,events,faults	↓
Change device interface	Developer	$\propto$ depth inheritance hierarchy	↓
Change fault detection strategy	Developer	$\propto$ # faults involved	↓
Change measurement procedure	Test engineer	0	↓

depends more on the former set of functions.

If a class interface has to be modified, for example by adding/removing a method, the change will involve many modules since typically a device is part of a hierarchy of classes in a generalization relationship [7]. The effort to add/remove a method is fixed and determined by its own complexity, thus the growth of the evolution cost metric depends only on the depth of the inheritance hierarchy (Tab. 1). In the design phase of FFMM the maximum depth was kept to a reasonable value (4), so this class of changes requires a limited effort.

The evolution cost estimation strongly depends on the device considered. In order to provide a quantitative example, in the following the driver for an Encoder Board, developed at CERN and employed in different scenarios typical of the magnetic measurements [12], is taken into account. The device is part of the hierarchy of classes shown in Fig. 2. Adding the device requires a considerable programming effort, anyway FFMM provides support in the following ways: it provides libraries implementing communication features on different buses, so that all the required functionalities are already available and accessible through a suitable interface. Furthermore, FFMM already implements and makes available infrastructures for event handling and fault detection. The tasks of exploiting events and improving system fault tolerance are therefore extremely simplified for the user. He just needs to add few small modules to extend the event structure and the fault detection logic. The *generalized evolution cost metric*, computed as the sum of the total cyclomatic complexity of the modules to be added, has a value of 301 in the particular case considered.

4.2. Changing service strategies

FFMM provides many services to help the user employ the framework and enlarge its application domain. The choice of OOP reduces the number of modules affected by possible changes, thus assuring a good level of flexibility. Moreover, some services, for example the fault detection [18], were implemented by means of AOP. By this solution, the classes of FFMM are oblivious of triggering the execution of specific code in the related aspects providing the services. Classes and aspects are therefore completely decoupled, further increasing software flexibility. Namely, a change of the fault detection strategy typically involves only one module, without affecting in any way the corresponding device. The complexity of such a change can be estimated as the average complexity of a fault handling method multiplied by the number of methods to be modified, and is therefore proportional to the number of faults involved in the change (Tab. 1).

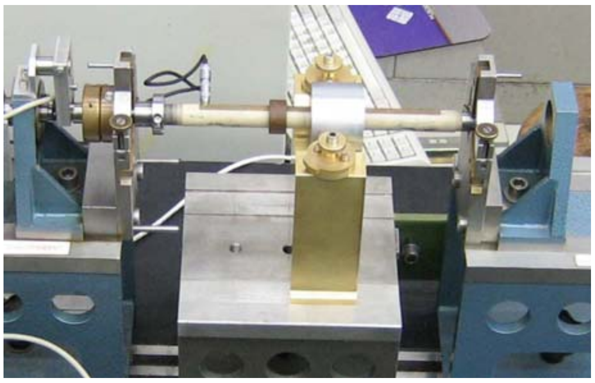
Fig. 3. Test bench for the quadrupoles of Linac4.

To provide a quantitative estimation, for the fault detection code specific of the Encoder Board one gets $C^{CC}_{Modules} = 4$, while for fault detection code common to other devices one gets $C^{CC}_{Modules} = 25$, for a total generalized evolution cost of 29.

4.3. Implementing new measurement algorithms

Several measurement techniques are currently employed for the test of accelerator magnets, among which fixed and rotating coils, stretched wire. As an example, the rotating-coil-based measurement station employed at CERN to test the quadrupoles of the new linear accelerator Linac4 [19] is shown in Fig. 3.

FFMM was designed to reduce drastically the amount of code affected by modification to the measurement procedure. As said before, the test engineer interacts with the framework mainly through the User Script, a formal description of the measure he wants to be executed. All the changes required by a new measurement algorithm are focused in the User Script, without affecting any other modules. In this case the framework is therefore provides the highest degree of flexibility, with $C^{CC}_{Modules} = 0$. This result was proven experimentally by developing from scratch an application for permeability measurements [20] by means of devices already developed and previously employed for rotating coil benches.

5. CONCLUSIONS

In this paper, an approach for the software flexibility assessment of measurement frameworks is proposed. In particular, this approach is meant to be applied in the context of the Flexible Framework for Magnetic Measurement (FFMM), developed at CERN in cooperation with the University of Sannio.

FFMM was designed to be flexible, reusable, maintainable, and portable. Now, a complete release of FFMM is available and already proved its effectiveness on the field, thus the evaluation of its degree of flexibility starts a new comprehensive phase of software quality and performance characterization, aimed at stating the fulfillment of challenging project goals.

The flexibility of the system cannot be stated in absolute terms, but only with respect to specified classes of changes, involving different users. The results highlight that the framework achieves increasing degrees of flexibility moving from the programming level to the user script level, and at the same time from the point of view of the developer to that of the test engineer. The highest flexibility is attained for the changes involving the measurement procedure, namely at the level where flexibility was mainly required.

ACKNOWLEDGEMENTS

This work is supported by CERN trough the agreement LHC/AT/K1464 with the University of Sannio, whose support authors gratefully acknowledge. Authors thank Giuseppe La Commara for his useful suggestions.

REFERENCES

[1] IEEE. Standard Glossary of Software Engineering Terminology 610.12-1990, Vol. 1. Los Alamitos: IEEE Press, 1999.

[2] http://zone.ni.com/devzone/cda/tut/p/id/3238#toc0 Designing Next-Generation Test Systems Developers Guide.

[3] J. M. Nogiec, J. Di Marco, S. Kotelnikov, K. Trombly-Freytag, D. Walbridge, M. Tartaglia, "Configurable component-based software system for magnetic field measurements", *IEEE Trans. On Applied Superconductivity*, Vol. 16, N. 2, Jun. 2006, pp. 1382-1385.

[4] http://www.tango-controls.org/

[5] P.C. Ferreira, H. Reymond, "Sequence of tests and settings to start a magnetic measurement on MMP 6.5.0", *Internal note EDMS no. 399822,* CERN 2003.

[6] A. Guerrero, J-J Gras, J-L Nougaret, M. Ludwig, M. Arruat, S. Jackson, "CERN front-end software architecture for accelerator controls", *Proceedings of ICALEPCS2003*, Gyeongiu, Korea, 2003.

[7] P. Arpaia, L. Bottura, M. Buzio, D. Della Ratta, L. Deniau, V. Inglese, G. Spiezia, S. Tiso, L. Walckiers, "A software framework for magnetic measurement at CERN", *Proc. of IEEE Instrumentation and Measurement Technology Conference*, Warsaw, Poland, May 1-3 2007.IMTC 07.

[8] P. Arpaia, L. Bottura, V. Inglese, G. Spiezia, "A flexible framework for magnetic measurements at CERN: a prototype for the new generation rotating coils", *Proc. of 15th IMEKO TC4 Symposium*, Iasi, Romania, Sept. 19-21 2007.

[9] P. Arpaia, M. L. Bernardi, G. Di Lucca, V. Inglese, G. Spiezia, "Fault Self-Detection of Automatic Testing Systems by means of Aspect-Oriented Programming", *Proc. of 15th IMEKO TC4 Symposium*, Iasi, Romania, Sept. 19-21 2007.

[10] P. Arpaia, L. Bottura, V. Inglese, G. Spiezia, "The new flexible platform for magnetic measurements at CERN", (In Italian), GMEE 07, Torino, Italy, Sept. 5-8 2007.

[11] P. Arpaia, L. Bottura, V. Inglese, G. Spiezia, "On-field validation of the new platform for magnetic measurements at CERN", *Measurement*, Volume 42, Issue 1, January 2009, Pages 97-106, ISSN 0263-2241, DOI: 10.1016/j.measurement.2008.04.006.

[12] P. Arpaia, M. L. Bernardi, L. Bottura, M. Buzio, L. Deniau, G. Di Lucca, V. Inglese, J. Garcia Perez, G. Spiezia, L. Walckiers, "Kernel Design of a Flexible Software Framework for Magnetic Measurements at CERN", *Proc. of the Instrumentation and Measurement Technology Conference*, *IMTC 2008*, 12-15 May 2008 Page(s):607 – 611.

[13] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Videira Lopes, J.M. Loingtier, J. Irwin, "Aspect-Oriented Programming", in *Proc. of the 11th European Conf. on Object-Oriented Programming (ECOOP),* Vol. 1241, pp. 220-242, Springer-Verlag, 1997.

[14] D.L. Parnas. "Software Aging." *Proc. Int'l Conf. Software Engineering—ICSE* (May 1994), pp. 279–287. Los Alamitos: IEEE Computer Society Press.

[15] A. H. Eden, T. Mens, "Measuring software flexibility", *IEE Proc.Softw.*, Vol. 153, No. 3, June 2006, pp. 113-125.

[16] McCabe. A Complexity Measure. IEEE Transactions On Software Engineering, Vol. Se-2, No. 4, December 1976, pp. 308-320.

[17] International Standard ISO/IEC 9126, "Information technology - Software product evaluation - Quality characteristics and guidelines for their use", International Organization for Standardization, International Electrotechnical Commission, Geneva, 1991.

[18] P. Arpaia, M. L. Bernardi, G. Di Lucca, V. Inglese, G. Spiezia, "An Aspect Oriented Programming-based approach to software development for measurement system fault detection", in press on *Computer Standards & Interfaces*.

[19] The Linac4 Project, "Technical Description for the Permanent Magnet Quadrupoles for Linac4", *Internal note EDMS no. 950248,* CERN 2008.

[20] K. N. Henrichsen, "Permeameter", Proc. 2nd Int. Conf. On Magnet Technology, Oxford, 1967.