Unified Software Method: Towards a Method of Measurement of the Necessary Changes to Software in Maintenance

Stéphane Mercier, Alain Abran, Michel Lavoie, Roger Champagne

École de technologie supérieure (ÉTS)
University of Québec
1100 Notre-Dame Street West
Montréal, Québec H3C 1K3, Canada

stephane.mercier.5@ens.etsmtl.ca, [aabran, mlavoie, rchampagne]@ele.etsmtl.ca

Abstract:

Within the context of use of the Unified Software Method (USM), traceability links are identified between all data elements of a software project that have a relation with another data element. Knowledge of these links provides complete traceability, which in turn means that synchronization of the information in a software project can be maintained. In this article, we propose a method of measurement based on the USM, which is aimed at quantifying the amount of information related to a maintenance action planned on an existing element in a software project. This measurement method makes it possible to quantify the ratio of project information to be maintained to total project information, at the same time as the amount of information involved in the maintenance project being considered.

Keywords

Maintenance, measurement method, traceability, USM, synchronization

Zusammenfassung:

Innerhalb des Kontextes des Gebrauches von der vereinheitlichten Software-Methode (USM), werden traceability Verbindungen zwischen allen Datenelementen eines Software-Projektes gekennzeichnet, die eine Relation mit einem anderen Datenelement haben. Wissen dieser Verbindungen liefert komplettes traceability, das der Reihe nach Mittel, daß Synchrounisierung der Informationen in einem Software-Projekt beibehalten werden kann. In diesem Artikel schlagen wir eine Methode des Maßes basiert auf dem USM vor, das die Menge von Informationen quantitativ bestimmend enthalten in einem Software-Projekt angestrebt wird, das mit einer Wartung Tätigkeit zusammenhängt, die auf einem vorhandenen Element in diesem Projekt geplant wird. Diese Maßmethode macht es möglich, das Verhältnis, um Projektinformationen zusammenzuzählen der beibehalten zu werden Projektinformationen quantitativ zu bestimmen, zur gleichen Zeit wie die Menge von Informationen mit einbezogen in das Wartung Projekt, das betrachtet wird.

Schlüsselbegriffe

Wartung, Methode des Maßes, traceability, USM, Synchronisierung

1 Introduction

The Guide to the SWEBOK [1] classifies maintenance in four categories: corrective, adaptive, perfective and preventive. Within the software engineering discipline, the majority of corrective maintenance actions are currently carried out at code level, and hence several of the measurement parameters identified with maintenance are related to code (application type, programming languages, software age, etc.). However, the initial source of defects should also be tracked, not only to its code instantiation, but to the design level as well. Moreover, since past maintenance associated with a requirement can significantly burden existing code, then, if the age of the software is substantial, the question arises as to what would be less expensive, to start again from scratch or to modify an existing application.

As a result, one of the objectives of software maintenance management is to establish whether, at a certain stage, it is preferable to keep maintaining the software or to rewrite it. The difficulty lies in achieving backward traceability or back annotation from the code to the software specifications and the failure to do so is a major cause of premature software aging. As stated in [6]: "*Software aging is caused by two kinds of sources. The first ones are natural and can be identified as technical obsolescence, incompatibility with new technologies, or supplantation by a new generation product. This aging is natural and can be predicted and anticipated. The other sources, artificial and undesired, are induced by humans. This occurs when modifications are directly made in the code without being reflected back in the design document and/or requirements. If synchronization between code and documentation is lost, it becomes harder to make changes in the code because we don't know exactly which parts are still needed and which parts are not. Progressive degradation of synchronization, caused by successive modifications without appropriate updating of documentation, will artificially create premature aging of the software. By losing traceability between code, design documents and requirements information, it becomes difficult to obtain clean code.*"

In the context of our research project on the development of a Unified Software Method (USM) [6], a key objective is to implement full traceability of the information to make it possible, among other things, to maintain the synchronization of information in a software project. Ideally, if the software information is always synchronized, the maintenance activities should not lead to unused code in the software, which, among other things, contributes to making software maintenance more challenging, time-consuming and expensive.

To maintain the synchronization of information in the USM, it is necessary to be able to identify the information to be checked (and updated if necessary) whenever a modification to the software is investigated. Without a complete and up-to-date traceability map of the software, it is not possible to establish exactly what the impacts of maintenance activities on it are. In this context, maintenance

activities rely on improvization and blind searching. With the USM, this software traceability map becomes available and appears to offer new possibilities for maintenance measurement. This paper presents the design of a new measure to quantify, for a project with its information structured according to the USM, the size of the potential impact associated with a software modification under consideration.

This paper is organized as follows. In section 2, the concept of traceability in the USM is presented. Section 3 presents some measures suggested in the maintenance literature. Section 4 describes the measurement model proposed in the context of this new measurement, and, in section 5, an analysis of the application of this measure is performed. Finally, section 6 presents some concluding observations and suggestions for future work.

2 Traceability in the USM

The USM is a method proposed for organizing the information of a software development project in the form of a graph in order to offer complete traceability of the software project information without being constrained by a methodology or a particular process. To construct such a graph using the USM tools, elements of information must be assigned to the nodes of the graph using a system of identification which indicates the address of the information and relates that to links of different types in such a way as to make traceability possible (see further information about these link types in section 5). The resulting graph constitutes the traceability scheme of the software project. It is to be noted that, even though this method can be used for both new and existing software projects, the result always depends on the availability of the software project documentation. If the documentation is incomplete, the graph will be incomplete; the expression “garbage in/garbage out” clearly applies in this case.

An information structure designed in conformity with the USM must also (as a self-imposed design constraint) be adapted to humans, as must the USM navigation tool offered as the main search engine in the context of the bidirectional traceability of information. Project information structured in this way offers both forward and backward traceability: each piece of information is related to both its predecessors and its successors. This makes it possible, for example, to identify which code implements a particular software specification. With the USM, project information can be represented in the form of graphs where the nodes correspond to that information and the links to a relation between two pieces of information. Thus, traceability indicates not only information associated with a particular requirement, but also with all the interrelationships between various kinds of information. For example, the same code can be associated with more than one requirement. Such links can thus be used to estimate the practical impacts of a modification.

To stay within human cognitive limits, one of the characteristics common to every node of a graph is that the level of complexity of the information must, by design in the USM, be limited. The following USM design constraints permit standardization of the amount of information complexity included in the various nodes:

- the information associated with a node should not exceed approximately 7 +/- 2 elements [7],
- the information associated with a node must contain at most 4 different variables (relational complexity) [4],
- the information associated with a node must provide a maximum of information in a minimum amount of time, using minimal space and with a minimum number of printed elements [8].

These constraints should be considered as guidelines in achieving the goal of creating information with manageable complexity. Moreover, since the synchronization of information is mandatory in the USM, whenever a modification is made to an information node, all the nodes influenced directly or indirectly must be updated.

3 Related work on software maintenance measures

Several measures associated with software maintenance have been proposed. Binkley and Schach [2] evaluated six types of measures in four case studies and proposed the Coupling Dependency Metric (CDM) to predict run-time faults, as well as effort, for corrective maintenance. The CDM combines three different types of measurements of dependency: referential ("*a measure of the extent to which a program relies on its declarations remaining unchanged*"), structural ("*a measure of the extent to which a program relies on its internal organization remaining unchanged*") and data integrity dependency ("*a measure of the vulnerability of data elements in one module to be changed by other modules*").

VanDoren [10] proposed a number of measures for maintenance, including one to determine whether or not it is preferable to maintain a software application or rewrite it. Their proposed Maintainability Index (MI) measure is defined as:

$$MI = 171 - 5.2\ln(aveV) - 0.23aveV(g') - 16.2\ln(aveLOC) + 50\sin(\sqrt{2.4\,perCM}) \quad (3.1)$$

where:

- aveLOC is the average number of lines of code per module;
- aveV (g') is the average extended cyclomatic complexity per module;
- aveV is the average Halstead Volume V per module [9]:

$$V = N * (\log_2 n) \quad (3.2)$$

where:

- N represents the program length (summation of all operators and operands);
- n represents the program vocabulary (summation of distinct operators and operands).
- perCM (optional) is the average percentage of comments per module.

It must be noted that on the righthand side of equation (3.1) the various quantities used in the summation are not of the same type: each added element refers to a distinct entity type and therefore has distinct units of measurement: number of lines of code, cyclomatic complexity number, percentage of comments, the 171 constant parameter and the log and sine values of some of these elements. Normally, in a valid mathematical equation, the combination of the various parameters must be coherent. Otherwise, it is the equivalent of adding apples and oranges, without a known and meaningful definition of either. Not only is this mathematically unacceptable, it is impossible to determine the corresponding units on either side of the equation. Thus, for equation (3.1), the MI unit is unknown and is certainly neither a value of the effort (in person-hours, for example) nor a measure of profitability, since these units are not used on the righthand side of the equation. None of these proposed measures is adequate for the USM context because none can be applied to the USM traceability map.

4 Design of the measurement method

4.1 Overview

In this section, we present the main elements of the design of the proposed measure. The methodology used is illustrated in Figure 1 and corresponds to Step 1 of the measurement process proposed in Jacquet and Abran [5]:

- Step 1: Design of the measurement method;
- Step 2: Application of the rules of the measurement method;
- Step 3: Analysis of the measurement results;
- Step 4: Exploitation of the measurement results.

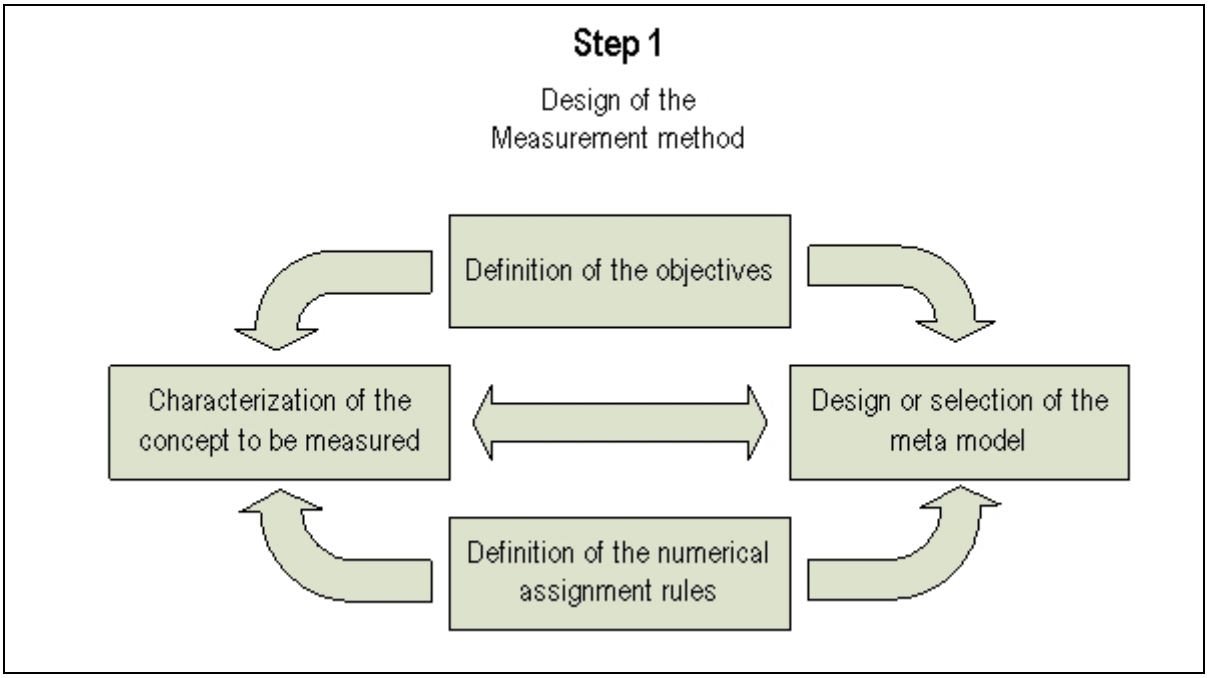

Figure 1: Step 1 of the measurement process: [5]

4.2 Definition of objectives

In the context of maintenance, three principal types of activities with respect to information are possible: modification, removal and addition. Removal and modification activities can have a direct or indirect impact on existing software information, and it is the anticipated impact of the necessary modification that we wish to measure (addition is not considered in our measurement model because it means creating new elements of information, and its impact is not included within the scope of the measure proposed below).

Currently, it is almost impossible to identify all the secondary defects inserted as a consequence of a maintenance modification. Moreover, as traceability is a fundamental concept in the USM, it is possible to identify the relations between the various data elements in software at any time. Consequently, since all the links are known, it is possible to identify the impacts on the whole of the software of a modification to be carried out at a specific point in a program. In USM maintenance, , it becomes possible, thanks to the connection between elements of information, to evaluate in advance the impact of a modification or of the suppression of information on the whole of the software in advance.

In summary, the objective of the proposed measurement is to arrive at the number of information nodes requiring an update as a result of a maintenance activity. With this quantitative information, we can gain a better understanding of the required maintenance and are in a better position to determine the resources needed to complete the maintenance activity.

4.3 Selection of the meta model

Figure 2 illustrates the meta model selected for the design of our proposed measure. The circles (nodes) in the graph correspond to the information contained in a USM project. The connections between the nodes make it possible to navigate in the graph to identify the nodes associated with a maintenance request. In the current version of the meta model, three types of connections can be identified. A simple arrow identifies the vertical decomposition connections. The multiple-view connections are identified by a double-arrow and the related-information connections with no arrow. The purpose of the various types of connections is to correctly lay down the rules of selection of the nodes associated with maintenance, and with each type of connection, there is an associated type of information.

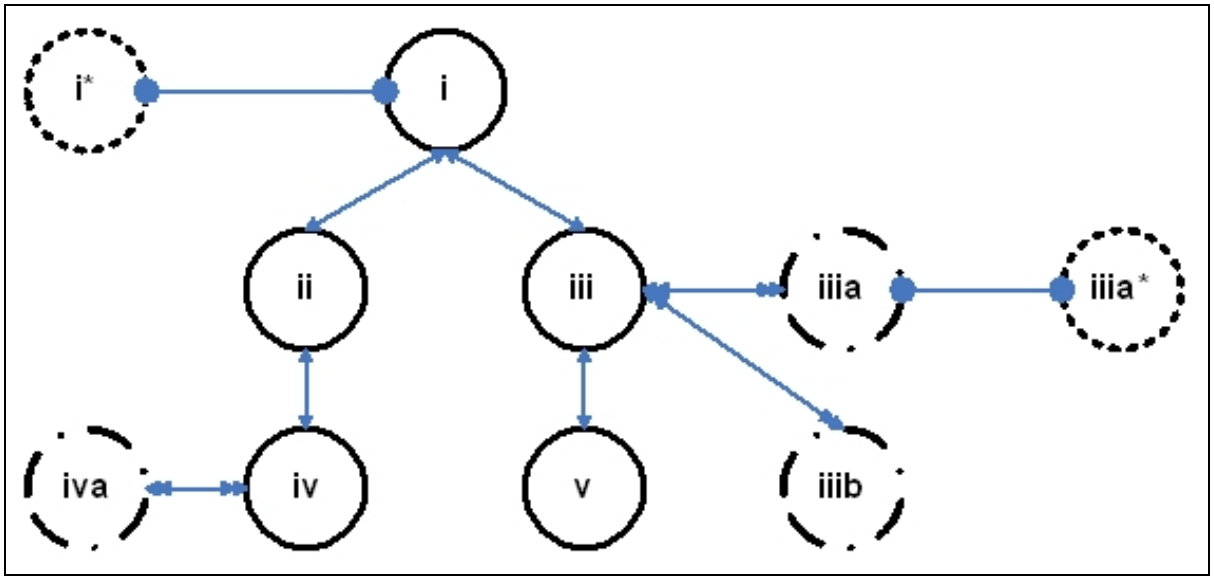

Figure 2: Meta model representation

The nodes (thick-lined circles) are associated with the vertical decomposition connections that contain information associated with the software life cycle (requirements, architecture, design and programming). The nodes represented as dashed circles are associated with the related information connections, which contain information not essential to the life cycle (a personal note, a rejected solution, tests, etc.) and the other nodes represented as dashed-dotted circles are associated with the multiple views containing information which is complementary to that of the vertical decomposition (static view, dynamic view, etc.). For a request to modify a software element, all types of information node must be considered. Following a maintenance activity, all information associated with a USM software project must remain synchronized.

The circles (nodes) illustrated in Figure 2 represent software information. In the USM graph, they are also nodes that can represent contextual information and personal notes.

4.4 Characterization of the concept to be measured

Since the synchronization of information is mandatory in the USM, all nodes undergoing modification which are either directly or indirectly influenced must be updated by the person who carries out the modification. In the context of this measurement, by design and by convention in this measurement proposal, only the nodes need to be considered in the equation. Specifically, measurement is characterized by three types of information which, taken together, constitute the result:

- identification of the nodes to be updated in the maintenance context considered;
- the exact number of nodes potentially concerned with the maintenance request being considered;
- the ratio of the nodes of the project which is potentially concerned with the maintenance request.

The connections illustrated in Figure 2 are essential to the measurement mechanism. Even though they do not appear in the measurement results, they make it possible to obtain those results. Without these connections, measurement would not be feasible.

4.5 Definition of the rules of numerical assignment

In addition to adequately identifying the nodes in question, two numerical assignment rules are defined: the number of information nodes to be updated and the ratio of information nodes to be updated relative to the total number of nodes (%).

$$ni = \sum i_m \tag{4.1}$$

According to the first rule, the total number of information nodes of the software, *ni*, associated with the node in maintenance is calculated and then the number of information nodes affected by this maintenance activity, i_m, is added to that total - see equation (4.1). With this value, it is possible to identify the maximum number of information nodes associated directly or indirectly with a maintenance request. By knowing the identity of all these nodes, it is then possible to carry out a maintenance action while preserving the synchronization of the information of the software project through checking and updating the information of the nodes identified by this measure.

$$\%ni = \frac{\sum i_m}{\sum i} * 100 \tag{4.2}$$

According to the second rule, the percentage of information nodes of the project *%ni* associated with the node undergoing maintenance by calculating the sum of the information nodes affected by this maintenance i_m and comparing it to the total

number of information nodes of the project *i* - see equation (4.2). With this value, the maximum proportion of information nodes of the project associated directly or indirectly with the maintenance being considered can be identified.

The percentage alone is an insufficient basis for a decision: without knowledge of the context, this information means little. For example, a statement that maintenance *A* will affect 10.7% of module *X*, and that maintenance *B* will affect 0.1% of module *Y* does not give the same information as a statement that maintenance *A* will affect 107 nodes of module *X*, and that maintenance *B* will affect 4900 nodes of module *Y*. The ideal would to be able to indicate that maintenance *A* will affect 107*ni*¦10.7% of module *X*, and that maintenance *B* will affect 4900*ni*¦0.1% of module *Y*. With this type of information, one can conclude that maintenance *A* involves a larger proportion of the information nodes of its project than maintenance *B*, but that *B* is smaller in scope than *A* in terms of the quantity of nodes.

5 Introduction to the application of the measurement method

5.1 Overview

This section is an introduction to the application of Step 2 of the measurement process. The main difficulty in applying this method is in the correct identification of the information nodes that could be influenced indirectly by the maintenance carried out on any node of information in a USM project. In theory, if a USM graph is traversed, it is possible to pass by all the information nodes. How, then, would it be possible to identify the nodes that are really potentially associated with the maintenance of only one node? To correctly identify all the nodes associated with maintenance, the USM graph must be analyzed according to three different types of connections: vertical decomposition and multiple-view, and connections with associated information. By adequately identifying these connections, it becomes possible to identify the nodes of the graph that are correctly linked, either directly or indirectly, to the information subject to the maintenance considered. It is important to note here that the measurement method is not defined according to the type of connection, but according to the information nodes, which are all considered to be comparable, that is to say, associated with information in textual form, diagrams, code, etc.

5.2 Vertical decomposition connection

The vertical decomposition connection derives directly from the USM constraint of adaptability to humans, and the data elements associated with these nodes directly from the design process, e.g. the elements from the definition of needs, the architectural design elements, the modeling elements, the programming elements or the verification and validation elements (test cases, for example). In a sequence

of nodes using this type of connection, all the lower nodes are concerned with maintenance, as are the nodes immediately above them.

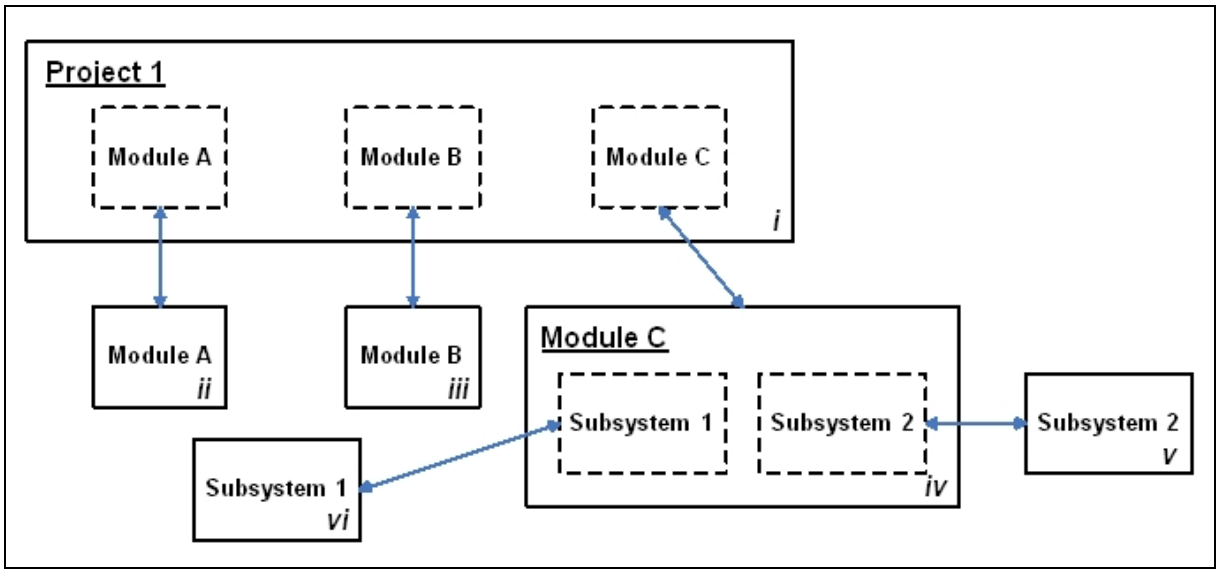

Figure 3: Vertical decomposition connection

According to the subset of nodes illustrated in Figure 3, if a maintenance action were to be carried out with the node identified as Module C, then the node immediately above it is the one identified as Project 1, which contains Module A, Module B and Module C. Everything below the nodes is considered as Subsystem 2, and Subsystem 1 could potentially have to be updated, depending on the elements that will be modified in Module C. The maximum numerical result of this measurement would be four (*i*, *iv*, *v* and *vi*). In practice, if, in Figure 3, only the Subsystem 2 element of Module C is affected by the modification, then only the branch associated with Subsystem 2 will be affected. Therefore, the maximum numerical result of this measurement would then be three (*i*, *iv* and *v*). Indeed, arrival at a lower node requiring no modification should mean that it is no longer necessary to go into more depth, starting from this node in the graph.

5.3 Multiple-view connection

The connection based on multiple views is the one that makes it possible to obtain a representation of information that is different. This type of connection corresponds to a necessary redundancy of information with a view to gaining a better understanding of it. For example, in Figure 4, a vertical decomposition connection is observed in addition to a multiple-view connection.

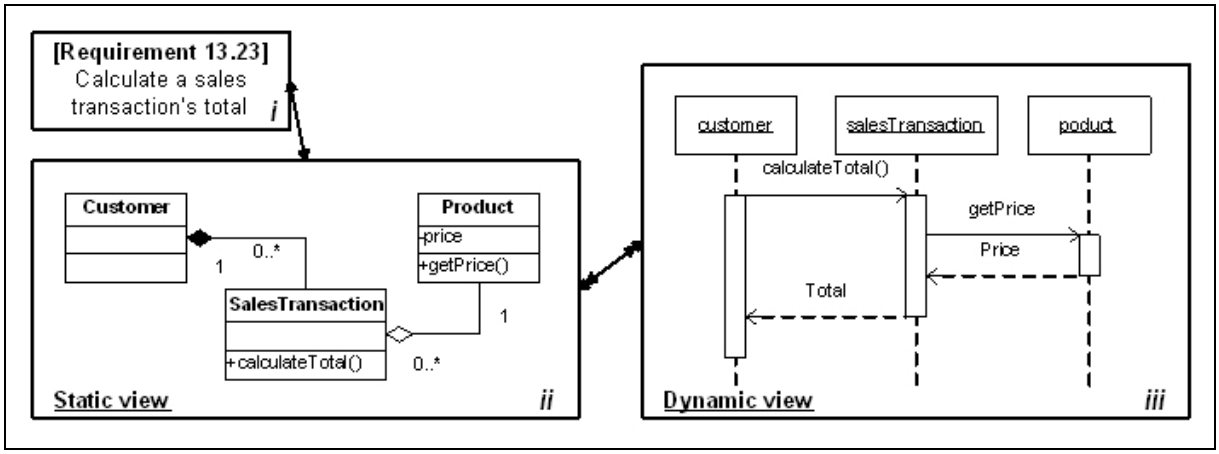

Figure 4: Connection based on multiple views

If the static-view information node is identified as a node in maintenance, then the dynamic-view node is automatically considered an extension of the static-view node. In practice, if node *ii* is in maintenance, then the numerical result associated with Figure 4 will be 3, because *i* and *ii* are associated connections of vertical decomposition and *iii* is associated with a multiple-view connection. It should be noted that a node of a certain type of information can be associated with several nodes of another type of view for the same information.

5.4 Associated information connection

A characteristic of the connection node of associated information is that it has information that is external to the design process. Like the node associated with the multiple-view connection, an associated information node is situated in parallel with another node in the hierarchy of the USM graph; however, unlike the multiple-view node, the associated information node will not necessarily require modification, but will nevertheless have to be checked. Indeed, this type of node can contain various types of data, such as additional explanatory personal notes, information on the solutions rejected, information on possible alternative solutions, various results, remarks on future evolutions, etc. In the example in Figure 5, if node *i* is in maintenance, the numerical result obtained is four; that is to say, *i*, *ii*, *iii* and *iv*, where *ii* and *iii* have an information connection associated with *I*, and *iv* has an information connection associated with *iii*, which is already related to *i*.

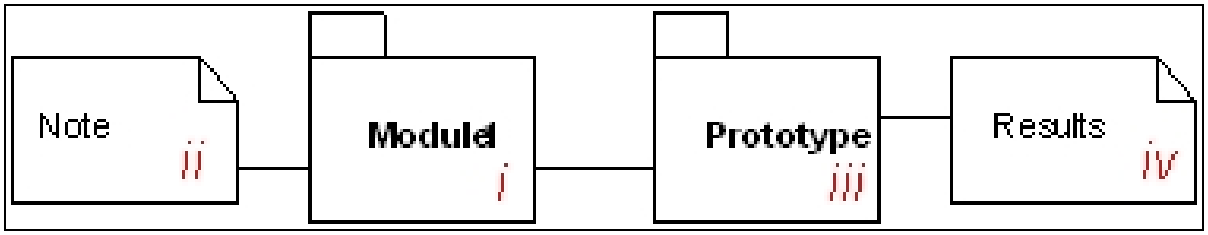

Figure 5: Information connection

5.5 Interpretation of the measurement result

As described previously, the results obtained with this measurement method in a USM context give three elements of information (via the nodes of the graph):

- the piece of software project information that is related to the intended maintenance;
- how many elements of information are related to the intended maintenance; and
- how much (what proportion) of the software project is related to the intended maintenance.

Because a USM goal is to maintain synchronization in software project documentation, the method should provide its own tools to do this for a maintenance activity. Identification of what information is related to the intended maintenance gives the engineer the knowledge to maintain the needed synchronization. In contrast, the information related to *how many* and *how much* can be used to determine what resources are needed to proceed with the maintenance activity.

Some maintenance measurement methods are intended to help the engineer to decide whether the code should be rewritten or maintained. Most of the time, the need for this decision arises when the condition of the code is very poor due to software aging, as described previously. For new USM software projects, software aging should be minimal, in which case this question will not arise, although it will if the code in an existing project is already badly damaged by artificial aging. However, at the current stage of the USM, we are not yet ready to answer to this question in the case of an existing software project adapted to a USM structure.

6 Observations and future work

In the context of the USM, the traceability map of the software that is available allows us to identify what information, and to quantify how much of it, is potentially related to a planned software maintenance activity.

Even if we could obtain exact results using the new measurement method we are proposing, the main operational difficulty will be to correctly identify the nodes of the graph that could potentially be associated with a maintenance request. Identification of the various types of connection enables us to clearly lay down the rules of identification of these nodes. With the percentage of nodes, we can measure the maximum proportion of information in the software project that may be affected, as well as calculate the number of information nodes to be modified and/or evaluated in the context of the maintenance activity considered.

Although the USM is still at the development stage, its concepts are sufficiently well defined that measurement methods related to them can also be defined. Through future experiments with various maintenance projects, it could become

possible to measure intervals by adding an evaluation of the lower limit that corresponds to the well-known upper limit. It would then be possible to parameterize equations 4.1 and 4.2 by taking into account the nature of the bonds associated with the nodes. Equations 6.1 and 6.2 illustrate this possibility, where *a*, *b* and *c* are values of the percentage of probability and *m1*, *m2*, *m3* represent the various types of connection.

$$ni = a\sum i_{m1} + b\sum i_{m2} + c\sum i_{m3} \tag{6.1}$$

$$\%ni = \frac{a\sum i_{m1} + b\sum i_{m2} + c\sum i_{m3}}{\sum i} * 100 \tag{6.2}$$

However, knowing exactly which elements are influenced directly or indirectly by the maintenance activity under investigation may enable this measurement to extend the life expectancy of the software by allowing the software engineer to do clean maintenance that does not leave behind unused code, one of the main causes of artificial software aging.

References

1. Abran, A., Moore, J., Bourque, P., Dupuis, R. and Tripp, L. *Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society, 2005.
2. Binkley, A.B. and Schach, S.R. Metric for predicting run-time failures and maintenance effort: four case studies. *CrossTalk - The Journal of Defense Software Engineering, 1998* (8). 21-23.
3. Ebert, C., Dumke, R., Bundschuh, M. and Schmietendorf, A. *Best practices in software measurement*. Springer, 2005.
4. Halford, G.S., Baker, R., McCredden, J.E. and Bain, J.D. How many variables an humans process? *Psychological Science, 16* (1). 70-76.
5. Jacquet, J.-P. and Abran, A., From software metrics to software measurement methods : A process model. in *Third International Symposium and Forum on Software Engineering Standards, ISESS '97*, (Walnut Creek (CA), 1997).
6. Mercier, S., Lavoie, M. and Champagne, R., Unified Software Method: an Engineering Approach to Software Engineering. in *30th Annual IEEE/NASA Software Engineering Workshop*, (Columbia, 2006).
7. Miller, G.A. The magical Number Seven, Plus or Minus Two - Some Limits on Our Capacity for Processing Information. *Psychological Review, 101* (2). 343-352.
8. Tufte, E. *The visual display of quantitative information*. Graphics Press,

Cheshire, Connecticut, 1983.

9. VanDoren, E. Halstead Complexity Measures *Software Technology Roadmap*, Software Engineering Institute, 1997.

10. VanDoren, E. Maintainability index technique for measuring program maintainability, Carnegie Mellon Software Engineering Institute, 2005.