# Towards an Integrated Quality-Oriented Modeling Approach for Software Evolution Control

Adeel Ahmad, Henri Basson, Laurent Deruelle, Mourad Mohamed Bouneffa

# Towards an Integrated Quality-Oriented Modeling Approach for Software Evolution Control

Adeel Ahmad, Henri Basson, Laurent Deruelle, Mourad Bouneffa
Univesité Lille Nord de France
Laboratoire d'Informatique, Signal et Image de la Côte d'Opale
BP-719 62228 CALAIS Cedex FRANCE
Email: {ahmad, basson, deruelle, bouneffa}@lisic.univ-littoral.fr

*Abstract*—**The operating environment can be significant for an applied change on a software artifact to propagate its mpact on the other components of the system. We present a modeling approach for tracing the reliable qualitative change impact flow during the software evolution. The approach is implemented as a framework including software specific information data gathering and automatic mechanism providing both structural and qualitative change impact analysis. This facilitates the decision making during the software evolution and maintenance processes.**

***Keywords-change impact analysis; software modeling; software quality; software structure; software evolution;***

## I. INTRODUCTION

To control software evolution within the constraints of time and resources without quality degradations when constant changes make software larger, more sophisticated and more complex is a significant challenge. Changes in software structure may affect the original software quality. It is observed that the software quality tends to erode with the increase in size and complexity due to the addition of new functionality and changed structure. The Software Quality Assurance (SQA) has always been a complicated task and deriving formulations to evaluate the whole software quality is more complicated without traceability information.

The software quality engineering community has been studying quality evaluation issues for a significant amount of time and made much progress in quality metrics. There have been proposed many theoretical and numerical frameworks to assess the software quality. The software quality models proposed by B. Bohem [1] and Mc. Call [2], [3] are known to be the historical models to represent, conceive, and evaluate the software quality. They have investigated the involvement of multiple quality factors to assess the whole software quality. Several variations have been formulated to the Boehm-McCall Model since its proposition [4], [5], [6], [7], [8], [9], [10], [11]. These address the analysis of software quality factors and their qualitative or quantitative metrification. The increased complexity in quality assessment models makes it more difficult to analyze basic causal flows.

A successful software change incorporation process may require to understand the change impact propagation considered from several aspects to prevent the software decay. In the context of software applications, the propagation of change impact can be considered from structural, qualitative, functional, logical, or behavioral point of views [12]. We particularly focus on qualitative change impact flow caused by a structural modification in a software artifact. This cannot be achieved without having an exploitable knowledge, describing exhaustively software artifacts and the different kind of relationships linking them. Improving the software evolution control without quality degradation issues is directly related to understand software dependencies. An exhaustive information regarding different software artifacts should help to analyze qualitative aspects of incorporated changes. In this article, we discuss analysis of structural and qualitative impact propagation of an intended change. Our approach aims at understanding the software evolution. This is achieved by means of formalization of the software artifacts and their various interactions. The objective is to provide a systematic approach for successful change incorporation in the developed software and thus to minimize risks generated by the change.

The article is organized as follows. Section 2 explains an insight to the qualitative evaluation of software systems. Section 3 further explores the qualitative model for the software evolution and its mapping with the structural one. Section 4 shows a part of the platform implementing our approach. In section 5, we give some concluding remarks and highlight the perspectives of our work.

## II. QUALITATIVE MODEL FOR SOFTWARE EVOLUTION

The number of modifications of an artifact can be an important indicator of artifact fault rate, stability, and subsequently artifact complexity. The proposed qualitative model is coupled to another model called *Structural Model of Software Evolution* (*SMSE*) [13], [14]. SMSE serves as a prerequisite for the qualitative assessments regarding the change impact traceability and the software comprehension. We present *Qualitative Model for Software Evolution* (*QMSE*) to better analyze the causal flow links in software quality attributes. It particularly refers to the traceability links in qualitative assessment of software as a result of any structural change.

### A. *A quality sub-graph of phase*

In structural aspect, software development phases connect to each other with inter-phase relationships. Generally, the Factors and the Criteria represent the quality attributes of an individual phase of software life cycle, the Metrics destined to their evaluations are represented to traverse a specific sub-graph instead of a global representation in the general graph.

The artifacts from different software development life cycle phases, which principally targeted by the quality assurance, are primarily evaluated across the individual phases during their development. Then, these are considered in their particular evaluations in correspondence with artifacts from other phases, or in a global evaluation from multiple phases.

This representation not only considers the special formalisms and applied method for each phase but also it allows to separately manage the quality per phase for the medium and larger sized projects.

The *QMSE* considers the quality graph of a software built on several artifact levels in respective software phases. The root of the model, denoted as $QG(\Sigma_\Phi .Art)$, represents the general quality of the software artifacts. We denote the individual quality of the phase *j* as $Q_i(\Phi_j)$ and the number of phases of the development model as $|\Sigma_\Phi|$. The node $(QG(\Sigma\Phi.Art))$ is then defined such as the descendant set (Dsc):

$$Dsc(QG(\Sigma_\Phi .Art)) = \{Q_i(\Phi_1.Art), Q_i(\Phi_2 .Art), \ldots , Q_i(\Phi_{|\Sigma\Phi|} .Art\}$$

In the same way, descendants contain the roots of the sub-graphs of the quality of software artifacts from each phase. We consider the common phases as shared by all major development models [15] (waterfall, V, Spiral, Incremental, Evolutionary, Transformational, Agile, Rapid Prototyping etc.). The table I links each descendant of $QG(\Sigma_\Phi .Art)$ to the corresponding phase.

The sub-graph of qualitative synthesis ($Q_s(\Sigma_\Phi.Art)$) is as each node represents a qualitative view of a set of phases specific to the quality expert. This view is based on the occurrence of qualitative attributes across the software artifacts from individually designated phases. It refers to the factors, criteria and the sub-criteria of sub-graphs of quality associated to the different phases. The descendants of a node of $Qs(\Sigma\Phi .Art)$ may either refer to the node $\in Q_i(\Phi_j)$, $i = 1 \ldots |\Sigma_\Phi|$ or represent another node $\in Q_s(\Sigma_\Phi .Art)$ (Fig. 1).

TABLE I. ROOTS OF THE SUB-GRAPHS OF THE INDIVIDUAL PHASES OF DEVELOPMENT LIFE CYCLE

| Root of the sub-graph of the quality | Concerned phase of the development life cycle |
|---|---|
| $Q_i(\Phi_0.Art)$ | Feasibility Study |
| $Q_i(\Phi_1.Art)$ | Requirements Definition |
| $Q_i(\Phi_2.Art)$ | Functional Specification |
| $Q_i(\Phi_3.Art)$ | Primary Design |
| $Q_i(\Phi_4.Art)$ | Detailed Design |
| $Q_i(\Phi_5.Art)$ | Coding |
| $Q_i(\Phi_6.Art)$ | Unit Testing |
| $Q_i(\Phi_{\ldots}.Art)$ | … |
| $Q_i(\Phi_x.Art)$ | Evolution |
| $Q_s(\Sigma_\Phi.Art)$ | sub-graph of qualitative synthesis |

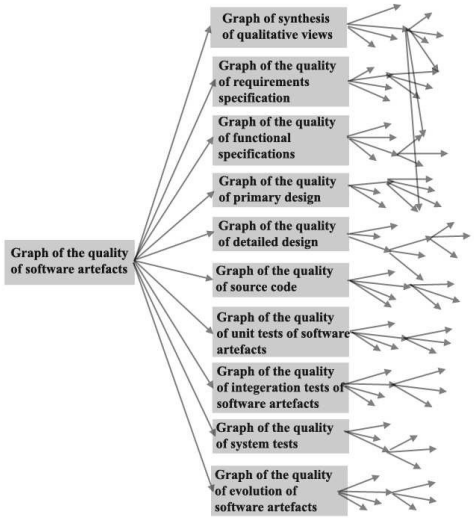


Figure 1. Graph of the quality of artifacts of individual phases.

### B. *Graph of profound variability*

The global software quality is generally assessed as a result of quality evaluation on three different levels (factor, criterion, and metric). Factor represents the quality attribute from external view. Criteria evaluate the factor from an internal or developer aspect. The only separated refinement level are the brief and global terms of the quality (factors), and their qualitative and/or quantitative evaluation by some metrics. While in practice, the only refinement of factors into criteria is an immediate transition to the recognized meaning-ful quantification, and it is often found that the criteria involves various underlying attributes. This general quantification by metrics may cause a lack of precise and certain evaluation. In consequence, applying *sufficient* number of refinements, seems necessary to allow an exhaustive representation of the software quality. This can make its evaluation preciser and more detailed.

Let G = (X, U) the Factor Criterion Metric (FCM) graph of the quality established for an application: The set X of nodes is the union of subsets of nodes representing the various elements of the FCM graph as:

$X = X_0 \cup F \cup C \cup M$

- $X_0$ is the root of the graph, It represents the most abstract element of the quality, or the general quality of the software, and corresponds to the level 0 of the refinement;
- $F = \{f_1, f_2, \ldots , f_m\}$ is the set of factors constructing the primary refinement of the general quality or the level 1 of the refinement;
- $C = \{c_1, c_2, \ldots c_p\}$ represents the set of criteria which refine the factors. It compounds the level 2 of the refinement;
- $M = \{m_1, m_2, \ldots , m_n\}$ represents the set of metrics permitting the quantitative and/or qualitative evaluation of criteria.

We adopt a profound variability, in which the numbers of refinement levels (of a factor or criteria) vary in function of the complexity of the concerned factor or criteria. The refinement of a criterion in sub-criteria and then recursively their sub- criteria into more detailed versions is an expert activity of the quality modeling. The quality engineer drives this process recursively until (s)he accomplishes the precise attributes that can be evaluated in consideration of relevance and therefore interpretable, with one or more metrics. In the same way, the Metrics destined to their evaluations, traverse a specific sub-graph recursively until the assessment of a global quality of the software system. We also intend to define the related change impact propagations between quality factors, criteria and the sub-criteria of the same level of refinement. The representation of these horizontal relations allows to consider the contradictions which can exist between different quality attributes. It then permits to define an evolutionary construction of the quality.

It will be useful to underline that there is no conceptual difference between criteria and descending sub-criteria. Both represent the technical attributes linked by a composition relation, implying that all the sub-criteria help to directly evaluate their ascendants (and indirectly their ancestors).

Practically, in all software applications the related levels of factors and criteria are identical, these describe the general properties, contributing each, in the certain measure to the assurance of the global quality. This refinement of criteria into further sub-criteria allows to pass from a generality shared by a family of languages, formalisms, methods to the particularity of each individual artifact expressed from its specific use in the application. The same refinement reflects the knowledge of the *quality expert* to define the relations between criteria and sub-criteria. It also reflects the desired importance of each factor, as a function of comparative priorities of other factors. This expertise is therefore in relevance to design "how to detail the quality?" by which it shows the general description of qualitative objectives to the expression of sub-objectives in condition to their accomplishments. Between criteria and the metrics, multiple levels Si of sub- criteria can exist (Fig. 2). The set X (of nodes) of the FCM graph is then be defined as:

X = X0 ∪ F ∪ S ∪ M where:

$S = \{s_1, s_2, \ldots, s_z\}$ represents the set of different levels of sub-criteria, and

$s_i = \{c_{i,1}, c_{i,2}, \ldots, c_{i,n}\}$ denotes the set of sub-criteria of the $i^{th}$ refinement.

### C. *Classification of metrics in layers*

For changing evaluations, a layered model has been adopted based on the *QMSE* and it is comprised of a core and two layers. The core constitutes a set of metrics of BMS (Basic Metric Set) base, such that each metric of this set may neither calculate nor reduce from multiple metrics. Moreover, like all metrics, a base metric is destined to test the presence of a quality attribute in one or more artifacts, to qualify or quantify this attribute. The choice of attributes is pragmatic and it is based on the experience in qualitative evaluation of application.

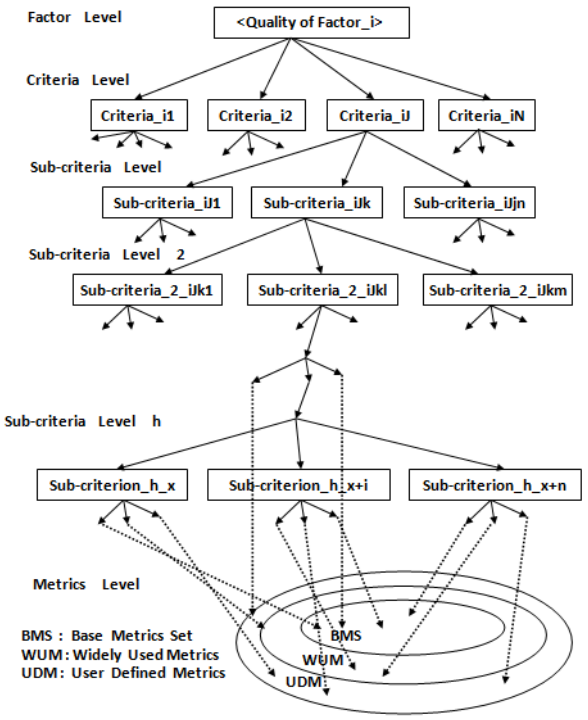


Figure 2. Tree like quality criteria, sub-criteria, and metrics

The base metrics are insufficient for the divers evaluations of quality attributes, two layers of metrics have been defined above the core:

WUM denotes the set of Widely-Used Metrics. These are proposed metrics in the literature comprising the evaluation of quality attributes. These metrics are comparatively largely applied, in the industry. These metrics are formulated by the functions like parameters of base metrics and making call to certain number of operators.

UDM denotes the set of User-Defined Metrics. The difference between these metrics and the set of precedent is to justify by the fact that an expert can, through experience, define a certain metric, although it does not belong to the set of basic metrics or Widely-Used Metrics. It can be pertinence to the measurement of a specific aspect of the project or to put in evidence certain particulars found in the distinct context of development. The UDM then respond to the specific needs in observation and in evaluation putting the specific data to the project from an expert point of view. The definition of these metrics is based globally on the two layer metrics. These metrics needs the evident that *quality expert*, defining these metrics provides the conditions of their usage, then it define and refresh the interpretation of numerical results issued from the application.

Therefore, an element of WUM is a function parameter by:

1) one or more artifacts targeted for the evaluation,
2) one or more metrics belonging to BMS.

Each element of UDM is a function parameter by:

1) one or more artifacts targeted for evaluation,
2) one or more metrics belonging to BMS ∪ WUM.

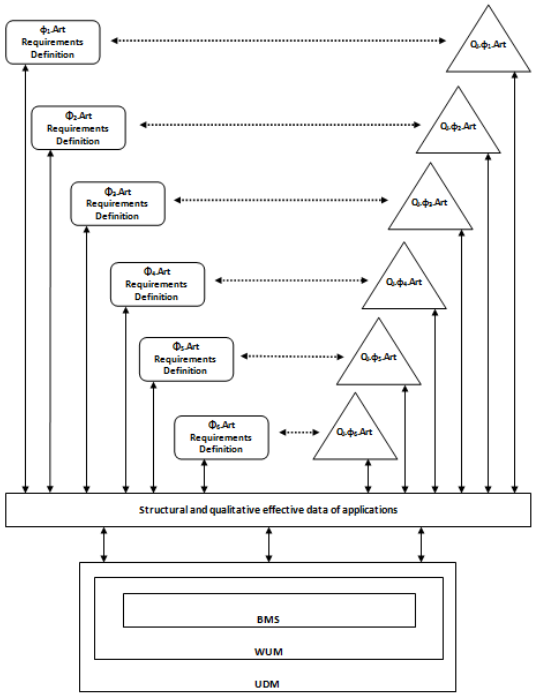


Figure 3. Structural graphs of artifacts and associated qualitative graphs

## III. MAPPING BETWEEN SMSE AND QMSE

Each set $\Phi_i$ .Art of artifacts of a software development life cycle phase corresponds to a multi-graph constructed according to the instantiated structural model (Fig. 3). The qualitative set $\Phi_i$.Art corresponds to a quality graph denoted by $Q_i(\Phi_i.Art)$ which is elaborated by the quality expert and which allows to evaluate the artifacts of $\Phi_i$ of multiple qualitative points of view represented as the quality factors and criteria.

Any structural change($\Delta S$ ) invoked on an artifact may cause a relative qualitative change ($\Delta Q$) in the corresponding quality criteria. The impact of $\Delta Q$ propagated progressively to the precedent criteria and then to factors can be traced up to the global quality of software.

We can determine the qualitative change $\Delta Q$ affected by structurally changed artifact $C_x$ . So, the notation $\Delta S \Rightarrow \Delta Q$ means that the structural change $\Delta S$ implies a qualitative change $\Delta Q$ . Let us consider some qualitative metrics set M = $\{m_1, m_2, m_3, \ldots m_n\}$ associated to the artifact $C_x$. The values associated to these metrics may be represented by the set V = $\{v_1, v_2, v_3, \ldots v_n\}$. A structural change $\Delta S$ applied to the artifact $C_x$ may result in a new set of the metric values represented by V = $\{v_1, v_2, v_3 , \ldots v_n\}$. This is shown as below:

$m_1, m_2, m_3, \ldots m_n (C_x) = v_1, v_2 , v_3, \ldots v_n$

$\Rightarrow M (C_x) = V$

After the change, $\Delta Q(C_x') = \Delta Q(C_x) + \Delta Q(\text{change})$, then we get the set V' as:

$m_1, m_2, m_3, \ldots m_n(C_x') = v'_1, v'_2, v'_3 , \ldots v'_n$

$\Rightarrow M (C_x' ) = V'$

The difference in metrics values ($\Delta M = V' - V$) denotes a variation of the corresponding quality criterion. It could be measured by applying specific metric mi. The corresponding difference of values from concerned metrics $\Delta m_i$, $\Delta m_{i+1}$, $\Delta m_p$ represents the change in quality criterion as $\Delta C$ . The change in a criterion can affect the value of other related criteria ($\Delta c_j$, $\Delta c_{j+1}$, … $\Delta c_q$ ) on the same level. This identified difference with other related criteria yields the obvious change in their describing quality factor. In the same way, the combinations of changes in overall quality factors represent quality variation of the whole software product.

## IV. PROTOTYPE OF IMPLEMENTATION

For an operational validation of the proposed models of software evolution, we have implemented a prototype allowing friendly experimentation of quality graphs associated to a software development project. The prototype is built as a set of Eclipse IDE plug-ins. This prototype contains a graph editor which provides in a very simple way the nodes and arcs of the structural graph. We have used the Java Universal Network/Graph (JUNG) Framework. This is a software library that can be re-used for the modeling, analysis, and visualization of data as a graph or network. This library allows to define the structure of data Graph and also to use certain graph primitives for the construction of user interfaces associated with the graph manipulation tools. We used it in interaction with the built-in capabilities of Java API, as well as those of other existing third party Java libraries i.e Drools. We have specialized the class Graph available in the JUNG library in a class that we called ArchitectGraph. The large quality information can be stored and manipulated through a semi-automated knowledge-based system. The Drools is a business logic platform which provides an integrated unified platform for Rules, Workflow and Event Processing. We make use of it in writing quality expert rules as well as rules of propagation to improve the interactivity of quality graph.

The developed prototype is able to parse the information from distributed applications built on Java 2 Platform, Enterprise Edition (also including XML mappings and database schemas) to demonstrate the structural graph based on the underlying interactions of the artifacts. In case of an invoked change on a software artifact, the affected artifacts are identified by an automotive analysis of the change impact

```
11  public Collection findEdge(Node pNode){
12      //Collection edges = new ArrayList();
13      Collection edges = new Vector();
14      Enumeration resources = edgeResource.keys();
15      while(resources.hasMoreElements()){
16          String aResource = (String)resources.nextElement();
17          if (aResource != pNode.getResource()){
18              Vector v = (Vector)edgeResource.get(aResource);
19              for (int i =0;i<v.size();i++) {
20                  Edge anEdge = (Edge)v.get(i);
21                  Node aNodeSource = anEdge.getNodeSrc();
22                  if (aNodeSource!=null && aNodeSource==pNode){
23                      System.out.println("Edge found :"+ pNode);
24                      edges.add(anEdge);
25                  }
26              }
27          }
28      }
29      return edges;
30  }
```

Figure 4. The Java Code Snip

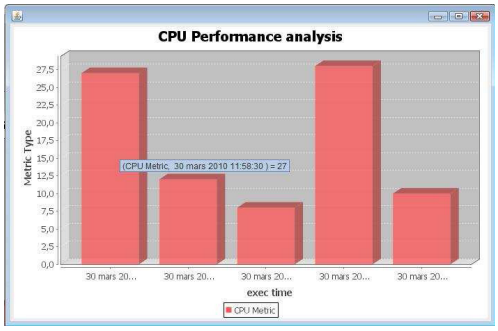


Figure 5. The performance graph of example code (before structural change)

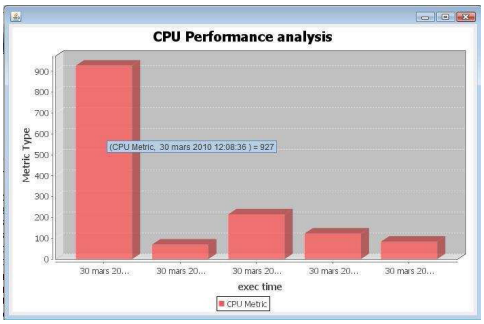


Figure 6. The performance graph of example code (after structural change)

conductivity of the relationship (edge), which connects the different artifacts (nodes) in the graph. This impact propagation is also followed in software code, simultaneously with the help of integrated textual messages and notifications of warnings, tasks, or errors inside the Eclipse IDE. It helps in tracing the impact of change on a particular software artifact and supports decision making during incorporation of changes. To illustrate the impact of a structural change to qualitative model, we present the code snip as in Fig. 4. For the purpose of explaining the qualitative impact propagation, we measured and observed the performance of the method findEdge. We invoked a change on the line # 12 of this code snip from ArrayList data to Vector in line #13. We observed the performance measure of the method findEdge by executing it 5 times each before and after the structural change. We found a great difference of execution time of the method (qualitative impact) whereas there wasn't any structural change impact propagation. The Fig. 5 shows the bar graph of the performance with ArrayList whereas the Fig. 6 shows the bar graph with Vector datatype.

## V. CONCLUSION

We presented an integrated software modeling approach intended to deal with the change impact analysis from different point of views including both structural and qualitative ones. The approach allows enhanced traceability of the change impact flow.

We particularly address the representation of software quality attributes related to artifacts belonging to the different software life cycle phases. This result in a graph based representation of the software quality that is mapped to the structural representation of the software artifacts also represented by a graph. This makes it possible to implement automatic tools providing the artifact change impact analysis from a qualitative view.

We are continuing this approach, also to consider the possibility of a *quality expert* to define and apply the metrics for an assessment of the qualitative changes experienced by different features in an evolving software application.

## REFERENCES

[1] B. W. Boehm, J. R. Brown, and M. Lipow, "Quantitative evaluation of software quality," in *ICSE '76: Proceedings of the 2nd international conference on Software engineering*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1976, pp. 592–605. [Online]. Available: http://portal.acm.org/citation.cfm?id=800253.807736

[2] J. A. Mccall, J. P. Cavano, and G. Walters, "Factors in software quality," vol. 1, 2, and 3, November 1977.

[3] J. P. Cavano and J. A. McCall, "A framework for the measurement of software quality," *SIGSOFT Softw. Eng. Notes*, vol. 3, no. 5, pp. 133–139, 1978.

[4] K. Ishikawa, What Is Total Quality Control?: The *Japanese Way*. Prentice Hall, March 1985. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike0720&path=ASIN/0139524339

[5] Y. Akao, "Qfd: Past, present, and future," in Transactions of the Third International Symposium on Quality Function Deployment. vol. 1. Plenary Session (downloadable from the QFD Institutes website: http://www.qfdi.org), October 12, 1997.

[6] A. Yoji, "History of quality function deployment in japan," in The Best on Quality, IAQ Book Series Vol. 3. International Academy for Quality, 1990, p. 183196.

[7] B. Kitchenham and L. Pickard, "Towards a constructive quality model part 2: Statistical techniques for modeling software quality in the esprit request project," Softw. Eng. J., july 1987.

[8] P. Petersen and B. Kitchenham, "The development of a software quality model," in Proceedings of 1st European Conference on Software Quality, Brussels, April 1988.

[9] S. Chulani and B. Boehm, "Modeling software defect introduction and removal:coqualmo (constructive quality model)," 1999.

[10] V. C. University, "Cocomo ii model definition manual," 1999.

[11] V. Basili, G. Caldiera, and D. Rombach, "Goal/question/metric paradigm," Encyclopedia of Software Engineering, vol. 1, pp. 528–532, 1994.

[12] A. Ahmad, H. Basson, and M. Bouneffa, "Software evolution control: Towards a better identification of change impact propagation," in ICET'08: Proceedings of the 4th IEEE International Conference on Emerging Technologies. IEEE Computer Society, October 2008, pp. 286–291.

[13] A. Ahmad, H. Basson, L. Deruelle, and M. Bouneffa, "A knowledge-based framework for software evolution control," in INFORSID'09: Actes du XXVIIeme Congr`es Informatique des organisation et systmes d'information et de dcision. Toulouse, France: IRIT Press (www.irit.fr), May 2009, pp. 111–126.

[14] A. Ahmad, H. Basson, and M. Bouneffa, "Rule-based approach for software evolution management," in IEEE APSSC 2009: IEEE Asia-Pacific Services Computing Conference, December 2009.

[15] R. S. Pressman, Software Engineering: A Practitioner's Approach, 7th edition. McGraw Hill Higher Education, January 2009