4-2006

Effect of changing requirements: A tracking mechanism for the analysis workflow

Subhajit DATTA
Singapore Management University, subhajitd@smu.edu.sg

Robert van Engelen
Florida State University

Effects of Changing Requirements: A Tracking Mechanism for the Analysis Workflow

Subhajit Datta
Department of Computer Science
and School of Computational Science
Florida State University
Tallahassee, FL 32306, USA
sd05@fsu.edu

Robert van Engelen
Department of Computer Science
and School of Computational Science
Florida State University
Tallahassee, FL 32306, USA
rvaneng@fsu.edu

ABSTRACT

Managing the effects of changing requirements remains one of the greatest challenges of enterprise software development. The iterative and incremental model provides an expedient framework for addressing such concerns. This paper presents a set of metrics – *Mutation Index*, *Component Set*, *Dependency Index* – and a methodology to measure the effects of requirement changes in the analysis workflow from one iteration to another. Results from a sample case study are included to highlight a usage scenario. Future directions of our work based on this mechanism are also discussed.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*complexity measures, process metrics*

General Terms

Algorithms, Management, Measurement, Design

Keywords

SDLC, Iterative and incremental model, Requirements, Analysis, Design, Metrics

1. INTRODUCTION

The iterative and incremental model is widely used for enterprise software development. The essence of this approach involves :

- recognizing that requirement changes – even while a software system is being designed and built – are inevitable,
- developing software through a controlled and coordinated sequence of steps going back and forth over the development stages of requirement specification, analysis, design, implementation and test.

Each set of such stages constitutes an iteration while the additive realization of the system's functionality results in its incremental development. Unlike the traditional *Waterfall Model* which insists on a chimerical *freezing* of requirements before any subsequent activity can start, (a condition almost never satisfied in non-trivial systems), the iterative and incremental paradigm seeks to successfully accommodate shifting stakeholder needs. Thus the focus revolves around efficient tracking of the changing requirements and understanding effects of such change on the system's design. In this paper we present a mechanism for quantitative evaluation of the effects of changing requirements. The following metrics are derived:

- *Mutation Index*
- *Component Set*
- *Dependency Index*

We also suggest a methodology for applying and interpreting the metrics. Following the introduction, Section 2 discusses the context of the metrics with emphasis on the salient features of the analysis workflow. In Section 3 related work is briefly reviewed. We next underscore the assumptions and derive the metrics. Section 6 proposes a methodology - a *process wrapper* around the metrics. Results from a case study are then presented, followed by future directions of our work and concluding remarks.

2. THE ANALYSIS WORKFLOW

The analysis workflow is a vital juncture in the software development life cycle (SDLC). This is when the description of the system in the user's language - captured in requirement specifications and illustrated through use cases - is first sought to be interpreted in the developer's idiom by introducing the essential formalism that helps clarify the internal workings of the system. "An analysis model can be viewed as a first cut at a design model (although it is a model of its own), and thus an essential input when the system is shaped in design and implementation" [1].

During requirement specification and use case modeling, the focus is principally on understanding how the user derives value from the behavior of the system; each usage scenario is deliberately viewed as being atomic and indepen-

SAC'06 April 23-27, 2006, Dijon, France

dent of others. Analysis discovers the significant perspective of *interaction*; how the components must *collaborate* to deliver not just separate slices of functionality but one cohesive system that facilitates understanding, implementation and maintenance.

The set of metrics *Mutation Index*, *Component Set* and *Dependency Index*, collectively isolate the effects of changing requirements on the collaborating system components from one iteration to another. The metrics are useful in clarifying the consequences of requirement changes to the stakeholders, adjusting project schedule and budget based on such change and discerning the *drivers* of overall architecture early in the life cycle.

3. RELATED WORK

Although it is common to use the terms *measure*, *measurement* and *metrics* in place of one another, some authors have underscored subtle distinctions [2], [3], [4]. For our discussion, we take *metrics* to mean "a set of specific measurements taken on a particular item or process" [5]. Halstead's seminal work [6] introduces metrics for source code. Metrics for analysis include the closely reviewed function point based approaches [7] and the Bang metric [8]. Card and Glass [9] have proposed software design complexity in terms of *structural complexity*, *data complexity* and *system complexity*. [10] identifies some important uses of complexity metrics. Fenton underscores the challenges of trying to formulate general software complexity measures [4]. Measurements of *Coupling* and *Cohesion* have been the focus of several studies [11], [12].

Chidamber and Kemerer present a widely referenced set of object oriented software metrics in [13], [14]. Harrison, Counsell and Nithi have evaluated a group of metrics for calibrating object-oriented design [15].

Karlsson et al. [16] use the Analytical Hierarchy Process to model a cost value approach for prioritizing requirements. An event based traceability approach is used by Cleland-Huang et al. [17] to manage evolutionary change of development artifacts. Lam and Loomes [18] have suggested an EVE (EVolution Engineering) framework for dealing with requirement evolution. Robinson et al. [19] propose a set of activities codified as Requirements Interaction Management (RIM), directed toward the discovery, management, and disposition of critical relationships among sets of requirements.

While these studies illuminate important aspects of software engineering in general and understanding requirements in particular, *it is necessary to connect the effects of changing requirements with the analysis artifacts in a clear, quantified strategy.* The measurement scheme derived in the following sections aims at capturing the effects of requirements changes in terms of the essential *continuity* of a development process. Our mechanism also provides a framework for automating the tracking of requirement changes and their consequences.

4. ASSUMPTIONS

During the analysis workflow, each requirement is scrutinized to ascertain the broad layers of the software system that will be required for its fulfillment. (*Fulfillment* is intuitively understood to be satisfying any user defined criteria to judge whether a requirement has been implemented to her satisfaction.) At this level the software system may be segregated into the layers of *Display*, *Processing* and *Storage*. Analysis reveals how these three categories can combine in a feasible design to relate to a particular requirement.

We make the following assumptions:

- The context of our discussion is *functional* requirements. We recognize *non-functional* requirements may warrant a different approach [20].

- By reviewing a particular requirement, an experienced analyst is able to recognize whether it concerns the *Display*, *Processing* or *Storage* aspects of the system. *Display* subsumes all features of the user interface and interaction facilities between the user and the system. *Processing* is any non-trivial modification of information performed by the system. *Storage* includes all activities associated with persisting information and accessing such information.

- When a requirement changes, the change can affect $Display(D)$, $Processing(P)$ or $Storage(S)$; singly or collectively. Thus, between iterations, each changing requirement, R_n is attributed a $Mutation\ Value$ $MV(n)$ of D, P or S; or any of their combination.

- The *Display*, *Processing* and *Storage* aspects may be associated with the three basic *stereotypes* of analysis classes; *Boundary*, *Control* and *Entity* in object oriented analysis [1]. The following derivation is based on this mapping; for non object oriented systems, corresponding components/modules may be substituted. The derived metrics are independent of Object Oriented Analysis and Design (OOAD) principles.

- The metrics address requirement changes between iterations; the identification of current, previous and next iterations is implicit in the discussion.

- *System* refers to the software under development along with its interfaces. *Component* refers to logical/physical entities whose interaction is necessary for the working of the system.

5. DEFINING THE METRICS

The metrics we now derive are meant to be *heuristics* for understanding and resolving the issues related to the evolution of a software system driven by customer needs. Analysis and design involve frequent decision making - choosing one option over another based on a gamut of (often conflicting) inputs. These metrics introduce an element of *objectivity* into the process such that the rationale behind the decisions can be clearly justified, communicated and documented. When automated, the mechanism can also significantly expedite the analysis of complex systems with closely related requirements.

For a system, let every requirement identified in the requirement workflow be uniquely named as $R_1, R_2, R_3, ..., R_n$. Between iterations I_{n-1} and I_n each requirement is annotated with its $Mutation\ Value$; a combination of the symbols D, P and S. The symbols stand for:
$D \equiv Display(1)$
$P \equiv Processing(3)$
$S \equiv Storage(2)$

The parenthesized numbers denote the *Weights* attached to each symbol. The combination of more than one symbol signifies the addition of their respective *Weights*, thus:
$DP \equiv 1 + 3 = 4$
$DS \equiv 1 + 2 = 3$
$PS \equiv 3 + 2 = 5$
$DPS \equiv 1 + 3 + 2 = 6$

Evidently, the order of the combination is irrelevant here, i.e. $DP \equiv PD, DPS \equiv SPD$ etc.

The *Weight* assigned to each category of components – *Display*, *Processing* and *Storage* – is a relative measure of their complexities. *Processing* components usually embody application logic and are most design and implementation intensive. *Storage* components encapsulate the access and updating of application data stores; their level of complexity is usually lower than that of the *Processing* components but higher than *Display* ones. Accordingly, *Display*, *Processing* and *Storage* have been assigned the *Weights* 1, 3 and 2 respectively. Exact values of *Weights* may be varied from one project to another; the essential idea is to introduce a quantitative differentiation between the types of components.

5.1 Mutation Index

The Mutation Index $MI(n)$ for a requirement R_n is a relative measure of the extent to which the requirement has changed in terms of the components needed to fulfill it.

Expressed as a ratio, the $MI(n)$ for requirement R_n :

$$MI(n) = \frac{The\ Mutation\ Value\ for\ R_n}{The\ maximum\ Mutation\ Value} \quad (1)$$

Thus, if at iteration I_n, Requirement R_m has been assigned a *Mutation Value* $MV(m) = DS$ with reference to iteration I_{n-1}, $MI(m)$ is calculated as :

$$MI(m) = \frac{DP}{DPS}$$

$$MI(m) = \frac{3}{6}$$

$$MI(m) = 0.5$$

Intuitively, if change in R_m can only affect the *Display* aspects of the system, the corresponding $MI(m) = D/DPS = 1/6 = 0.17$, which is less significant than the changes affecting only *Processing*, i.e. $MI(m) = P/DPS = 3/6 = 0.5$ or only *Storage*, i.e. $MI(m) = S/DPS = 2/6 = 0.33$.

At the boundary conditions, if a requirement has not changed from one iteration to another, the *Mutation Value* is 0 and $MI(m) = 0/6 = 0$. And, if all of *Display*, *Processing* and *Storage* aspects will be affected by changes in the requirement, the $MI(m) = 6/6 = 1$. $MI(m)$ for a requirement R_m can vary between these extreme values.

5.2 Component Set

The Component Set $CS(n)$ for a requirement R_n is the set of components required to fulfill the requirement.

During analysis, only the software components at the highest level are identified; they typically undergo several cycles of refinement over subsequent workflows. The *Component Set* is determined for components at their level of granularity at the analysis stage. Let the following components combine to fulfill requirement R_n :

$C_B \equiv Set\ of\ Boundary\ classes$
$C_C \equiv Set\ of\ Control\ classes$
$C_E \equiv Set\ of\ Entity\ classes$
$C_X \equiv Set\ of\ helper,\ utility\ and\ other\ classes.$

Then, the *Component Set* $CS(n)$ for R_n is defined as,

$$CS(n) = C_B \cup C_C \cup C_E \cup C_X \quad (2)$$

5.3 Dependency Index

The Dependency Index $DI(m)$ for a requirement R_m is a relative measure of the level of dependency between the components fulfilling R_m and those fulfilling other requirements of the same system.

For a set of requirements R_1,R_2,...,R_m,...,R_{n-1},R_n, let us define,
$Y = CS(1) \cup CS(2) \cup ... \cup CS(n-1) \cup CS(n)$

For a requirement R_m $(1 \leq m \leq n)$, let us define,
$Z(m) = (CS(1) \cap CS(m)) \cup ... \cup ((CS(m-1) \cap CS(m)) \cup ((CS(m) \cap (CS(m+1)) \cup ... \cup ((CS(m) \cap (CS(n))$

We can say, *semantically*, $Z(m)$ is the set of components that are not only relevant to R_m but also to some other requirement, say R_j.

Expressed as a ratio, the $DI(m)$ for requirement R_m :

$$DI(m) = \frac{|Z(m)|}{|Y|} \quad (3)$$

(For a set S, $|S|$ is taken to denote the number of elements of S.) We next present a process for applying and interpreting the metrics defined in this section.

6. A METHODOLOGY

The *Mutation Index*, *Component Set* and *Dependency Index* together with a methodology for applying them comprise a mechanism to evaluate of the effects of requirement changes. The following sequence of steps is a suggested methodology for the metrics :

`During the analysis workflow,`

1. Analyze each requirement for changes from the previous iteration.
2. Annotate each requirement with its *Mutation Value*.
3. Compute the *Mutation Index* for each requirement.
4. Identify the high level components (C_B, C_C, C_E, C_X etc.) required to fulfill each requirement.
5. Compute the *Component Set* for each requirement.
6. Compute the *Dependency Index* for each requirement.
7. From the *Mutation Index* and *Dependency Index* values, analyze the potential impacts of change for each requirement.
8. Refine the components to minimize the extent of such impact. If the change of a particular requirement may have serious budget/schedule impacts, intimate the project management/customer on the possibilities before proceeding further with development. (This and the immediately preceding step are best performed by experienced analysts and designers.)

`Perform the above steps for each iteration.`

7. CASE STUDY AND DISCUSSION

The following case study illustrates a typical scenario for applying the metrics and interpreting the results. We track the effects of changing requirements across several iterations of a sample application. Due to space constraints details of calculations are not shown.

7.1 The Scenario

A book reseller, in this paper referred to as *Books International Inc. (BII)* sells new and used books through their retail outlets across the nation. BII also offers a premium product, *autographed books* – first edition books signed by a select group of authors.

As a part of its expansion plan, BII decided to *webify* their business, launching an internet store and fulfilling electronic orders. An online presence for BII is expected to result in increased revenue from its standard as well as niche market of autographed books.

A software development organization, which we will call *Next Gen Tech (NGT)* has been contracted to develop the online BII store. NGT decided to use an iterative and incremental model for developing the system.

7.2 Results

We give the requirements for each iteration (I_n) and calculate the corresponding values for $MI(n)$, $CS(n)$ and $DI(n)$ as per the relations derived in section 5. Section 7.3 interprets the results. It is implied that the requirements are being forwarded by the stakeholders from BII; the analysts and designers from NGT evaluate them using the metrics to gauge their effects.

Requirements for I_1 :

R_1 - *The system will provide an online home page of BII, including a masthead with BII's logo, a static welcome message, and a hyperlink to a catalog page.*

R_2 - *The catalog page will contain alphabetical listing of BII's books. Initially BII will provide a list of up to 2000 different titles on its online store. The system will allow a user to select one/more listing(s) for purchase.*

R_3- *The system will record the Name, Mailing Address, Credit Card Number, Expiration Date and Credit Card Billing Address of the user wishing to purchase book(s).*

R_4- *The system will verify the credit card information and provide confirmation to the user along with total cost of the purchase (base price plus shipping and handling charges - BII will only offer standard shipping).*

In the very first iteration the requirements have not had a chance to mutate, hence the $MV(n) = 0$ and $MI(n) = 0$ for all R_n.

A list of components identified at this time with a brief description of their functionality is given below. These are at a very high level of abstraction, and will likely undergo refinements in subsequent workflows. The respective component type as defined earlier is noted in parenthesis beside the component's name. (Here, we do not seek to justify analysis and design decisions; focus is on the metrics values.)

List of components for I_1 :

`Page generator`(C_B) : To generate web pages with dynamic content.

`User input verifier`(C_B) : To validate form inputs from user.

`User info recorder`(C_C) : To process user information before persisting.

`Credit card verifier`(C_C) : To verify credit card details.

`Catalog store`(C_E) : To persist book catalog information.

`User info store`(C_E) : To persist user information.

`Total price calculator`(C_X) : To calculate total price of purchase.

Based on the above, the following is derived :

Component Sets $CS(n)$ for I_1 :

$CS(1)$ - {`Page generator`}

$CS(2)$ - {`Page generator,User input verifier,Catalog store`}

$CS(3)$ - {`Page generator,User input verifier,User info recorder,User info store`}

$CS(4)$ - {`Page generator,User input verifier,Total price calculator,Catalog store,Credit card verifier`}

The $DI(n)$ values in Table 1 signifies, at a later stage, changes to R_1 will have the least impact on the overall current design while changing R_2 and R_4 will affect the system most.

Table 1: *Mutation Value*, *Mutation Index* **and** *Dependency Index* **for requirements of I_1**

R_n	$MV(n)$	$MI(n)$	$DI(n)$
R_1	0	0	0.143
R_2	0	0	0.429
R_3	0	0	0.286
R_4	0	0	0.429

For the next iteration I_2, the following versions of the requirements were addressed.

Requirements for I_2 :

R_1 - *In addition to earlier requirement, the home page will present a list of authors whose autographed editions are currently available through BII. The list will be updated by BII's management once a month.*

R_2 - *In addition to earlier requirement, the catalog will provide a facility to search all of BII's books by author's name and/or book title.*

R_3- *In addition to earlier requirement, the system will present a disclaimer that none of the personal information of the user recorded by BII will be shared with any third party other than credit card agencies.*

R_4- *Remains unchanged*

As a marketing drive, all pages will have a list of five new arrivals at the top, under the heading,"BookWorm Recommends".

In the light of the changes, Table 2 shows the *Mutation Value* assigned to each requirement and the corresponding *Mutation Index*. It is worth noting, even as R_4 is declared to remain unchanged, it has a non zero *Mutation Value* since all pages now need to display a list of new books.

It is apparent R_2 has changed most and R_3 least. The corresponding $DI(n)$ values from I_1 suggests, absorbing the effect of R_3's change will be relatively easier than that of R_2's . With this insight, the revised set of components required to fulfill the current requirements are listed below.

List of components for I_2 :

`Page generator`(C_B) : To generate web pages with dynamic content.

`User input verifier`(C_B) : To validate form inputs from user.

`User info recorder`(C_C) : To process user information be-

Table 2: *Mutation Value*, *Mutation Index* **and** *Dependency Index* **for requirements of** I_2

R_n	$MV(n)$	$MI(n)$	$DI(n)$
R_1	D,P	0.67	0.22
R_2	D,P,S	1	0.33
R_3	D	0.17	0.44
R_4	D,S	0.5	0.44

fore persisting.
`Credit card verifier`(C_C) : To verify credit card details.
`Catalog searcher`(C_C) : To provide dynamic search facility of the catalog.
`New arrival identifier`(C_C) : To identify new additions to the catalog.
`Catalog store`(C_E) : To persist book catalog information.
`User info store`(C_E) : To persist user information.
`Total price calculator`(C_X) : To calculate total price of purchase.

The italicized components are the ones that have been added to address the requirement changes.

Guided by each requirement's *Mutation Value* the components were reassigned as:

Component Sets $CS(n)$ **for** I_2 **:**
$CS(1)$ - {`Page generator, New arrival identifier`}
$CS(2)$ - {`Page generator,User input verifier,Catalog searcher, New arrival identifier`}
$CS(3)$ - {`Page generator, User input verifier, User information recorder, User information store, New arrival identifier`}
$CS(4)$ - {`Page generator, User input verifier, Total price calculator, Catalog store, Credit card verifier, New arrival identifier, User information store`}

The redesign necessitated by the mutating requirements lead to a situation where R_3 and R_4 are now the most dependent followed respectively by R_2 and R_1. (As indicated by the *Dependency Index* values for this iteration in Table 2.)

As each iteration lead to an incremental release of the product, BII was satisfied with the project's progress. Accordingly, another round of requirements were put forward.

Requirements for I_3 **:**
R_1 - *Remains unchanged.*
R_2 - *Remains unchanged.*
R_3- *Remains unchanged.*
R_4- *In addition to earlier requirement, BII will also accept checks or BookWorm Coupons as payment for purchases. BookWorm Coupons are valid only for specific purchases. Items that can be bought with coupons will be marked in the online listing.*

This was put forward as a "minor addition" by BII's management; after all, R_4 is the only one requirement that changes.

NGT's analysts were quick to detect, R_4's change also affects R_3 and has the potential to impact the existing design as newer business rules (*What qualifies a product for purchase against coupons ? Can coupon purchases be combined with card/check purchases ? etc.*) need to be addressed and interfaces (*What other user information need to be recorded for check payments ? Which agency would verify such payments ? etc.*) built. Table 3 quantifies their qualms.

The $MI(n)$ values, coupled with corresponding $DI(n)$ values from the previous iteration indicate that far reaching changes have to be introduced to fulfill the latest requirements. The Technical Lead of NGT's development team reviewed the situation with her Project Manager, recommending the customer be notified of these implications.

We take leave of the case study now, when NGT is persuading BII for cost and timeline revisions before further development can proceed.

Table 3: *Mutation Value*, *Mutation Index* **for requirements of** I_3

R_n	$MV(n)$	$MI(n)$
R_1	0	0
R_2	D,S	0.5
R_3	P,S	0.83
R_4	D,P,S	1

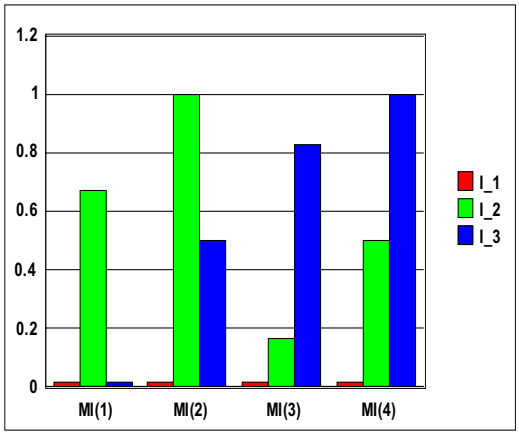

Figure 1: Variation of $MI(n)$ **across iterations for the case study's requirements**

7.3 Interpreting the Metrics

We highlight some key themes of the approach and summarize the results of the case study.

The metrics are meaningful *collectively*, they together give a view of the process continuum. *Mutation Index* is calculated with reference to the previous iteration, *Component Set* summarizes the design for the *current* iteration and *Dependency Index* reflects the potential effects in the *next* iteration.

The metrics are essentially indicators, they are meant to facilitate better understanding and judgment in the inherently subjective exercises of analysis and design.

Overall, the mechanism presented complements existing canons of software engineering. For example, calculation of *Dependency Index* is based upon *Component Set*, which is populated by component choices backed by common design considerations of cohesion, coupling etc.

Ideally, a requirement's low $MI(n)$ value reflects it has not undergone significant change; an unusually high value may indicate a need to spawn a new requirement or segregate the original requirement into two or more parts. Similarly, low $DI(n)$ values suggest low interdependencies; in the limiting case we may have independent components with zero

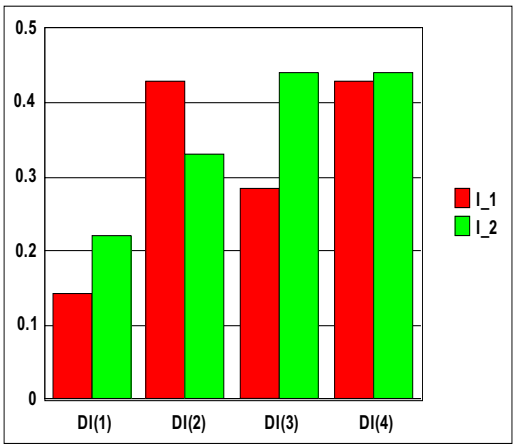

Figure 2: Variation of $DI(n)$ across iterations for the case study's requirements

interaction, an undesirable situation for interactive systems.

On the other hand, a high $MI(n)$ for a requirement may not necessarily be alarming if the corresponding $DI(n)$ is moderate. Likewise, a high $DI(n)$ for a requirement with a low $MI(n)$ does not necessitate involved redesign.

The metrics thus signify general directions in the architecture as a system is iteratively understood, built and refined.

Figures 1 and 2 show the MI(n) and DI(n) variations across the iterations of the case study.

8. FUTURE WORK

The mechanism presented above gives quantitative insight on changing requirement's outcome on system components. Work is currently in progress to incorporate this intelligence in an automated requirement tracking and analysis tool. Given sets of requirements and analysis artifacts, the utility will gauge the levels of interaction in the system, the extent to which the overall design may be affected by changes to the requirements, and suggest workarounds to mitigate those effects. We are looking to leverage the extension mechanisms of Unified Modeling Language version 2.0 and the Eclipse Modeling Framework (EMF) of the Eclipse platform towards this automation. A case study on a larger scale is also planned.

9. CONCLUSION

In this paper, we cited the iterative and incremental model as an expedient approach for tackling requirement changes in enterprise software development. On the foundations of this model, we have presented a set of a metrics - *Mutation Index*, *Component Set* and *Dependency Index* – and a sample process that together guide the development team on how requirement changes may be negotiated during the analysis workflow. We also discuss results from a case study and the future directions of our work.

10. REFERENCES

[1] I. Jacobson,G. Booch,J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.

[2] R.S. Pressman. *Software Engineering : A Practitioner's Approach, Fifth Edition*. McGraw-Hill, 2001.

[3] A.L. Baker J.M. Bieman, N. Fenton and D.A. Gustafson. A Philosophy for Software Measurement. *The Journal of Systems and Software*, April 1990.

[4] N. Fenton. Software Measurement : A Necessary Scientific Basis. *IEEE Transactions on Software Engineering*, vol. 20, no. 3, March 1994.

[5] E.V. Berard. Metrics for Object-Oriented Software Engineering. http://www.toa.com/pub/moose.htm, 1998.

[6] M. Halstead. *Elements of Software Science*, North-Holland, 1977.

[7] A.J. Albrecht. Measuring Application Development Productivity.*Proc. IBM Application Development Symposium*, Monterey CA, October 1979.

[8] T. DeMacro. *Controlling Software Projects*. Yourdon Press, 1982.

[9] D.N. Card and R.L. Glass. *Measuring Software Design Quality*. Prentice-Hall, 1990.

[10] T.J. McCabe and A.H. Watson. Software Complexity. *Crosstalk*, vol. 7, no. 12, December 1994.

[11] J.M. Bieman and L.M. Ott. Measuring Functional Cohesion. *IEEE Transactions on Software Engineering*, vol. 20, no. 8, August 1994.

[12] H. Dhama. Quantitative Models of Cohesion and Coupling in Software, *The Journal of Systems and Software*, vol. 29, no. 4, April 1995.

[13] S.R. Chidamber and C.F. Kemerer. Towards a metrics suite for object oriented design. *OOPSLA '91*, 1991.

[14] S.R. Chidamber and C.F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, vol. 20, no. 6, June 1994.

[15] R. Harrison, S.J. Counsell and R.V. Nithi. An Evaluation of the MOOD Set of Object-Oriented Metrics. *IEEE Transactions on Software Engineering*, vol. 24, no. 6, June 1998.

[16] J. Karlsson and K. Ryan. A Cost-Value Approach for Prioritizing Requirements. *IEEE Software*, September/October 1997.

[17] J. Cleland-Huang and C. Chang. Event-Based Traceability for Managing Evolutionary Change. *IEEE Transactions on Software Engineering*, vol. 29, no. 9, September 2003.

[18] W. Lam and M. Loomes. Requirements Evolution in the Midst of Environmental Change : A Managed Approach.*2nd Euromicro Conference on Software Maintenance and Reengineering(CSMR'98)*, 1998.

[19] W. Robinson, S. Pawloski and V. Volkov. Requirements Interaction Management. *ACM Computing Surveys*, vol. 35, no. 2, June 2003.

[20] S. Datta. Integrating the FURPS+ Model with Use Cases - A Metrics Driven Approach. Accepted to be presented at *The 16th IEEE Symposium on Software Reliability Engineering (ISSRE 2005)*, November 2005.