Software Change Complexity: A New Dimension for Analyzing Requested Change

Aprna Tripathi, Dharmender Singh Kushwaha, Arun Kumar Misra
Department of Computer Science and Engineering
MNNIT Allahabad
Allahabad, India

ABSTRACT
It has been well accepted by the software professionals as well as researchers that software systems have to evolve themselves to survive successfully. Software evolution is a crucial activity for software organizations. Software complexity has existed as an important issue ever since the software programs came into existence. Thus, it becomes necessary to visualize and analyze the complexity of requested change before implementation. The goal of this paper is to identify the software complexity after change. The complexity will be used in taking decision about approval or rejection of the requested change, estimating effort for implementing change, estimating effort required in regression testing predicting number of possible faults. We have applied our proposed approach on four case studies. These case studies show some evidence that our approach is reasonably efficient and precise as well as being practical for software change management.

Index Terms
Software change management, cohesion, coupling, and software change complexity.

1. INTRODUCTION
Even today, software change management is a very challenging area for the researchers. While about three decades, researchers are putting their continued effort in this direction and share their contemplation. Change seems to be very simple when someone demands it, but the complexity of the task appears when it moves towards implementation phase. Software maintenance and evolution is an expensive phase in the software development life cycle. The quality and demand of software can judge by its maintainability. The Software maintenance is by far the costliest and most difficult phase in the software life cycle. The motivation behind this work is to improve the maintainability of software systems, so that maintenance effort is reduced. Reduction in effort can achieve by analyzing the software complexity.

As a change is requested, it is not only the issue where to make the change. However, how the change will be process, is also an important consideration in order to keep the quality parameter of the software in terms of reliability, understandability, reusability and maintainability?

Software complexity plays a very significant role in understanding the degree of difficulty associated with the development of proposed change, in estimating the effort required in implementation as well as in the regression testing.

The most challenging activity during maintenance is to assess the impact of the requested change on the existing system. The ignorance of impact analysis of proposed change could decrease the maintainability of the system. Software complexity may have a direct impact upon maintenance costs as well as the costs incurred through the presence of software errors.

This paper proposes a method for analyzing change with four case studies. Section II discusses related work. Section III details proposed approach. Section IV and V presents the case studies and results. Section VI, the last section of the paper, outlines conclusions and future work.

2. STATE - OF- THE- ART
The Change is a continuous process. Change may be requested by different sources and may have different types. Buckley et al. [1] propose taxonomy of software change. As a change is requested, it is analyzed and then after the approval of the decision committee it is further processed. An impact is the effect of one object on another. Software Change Impact analysis (SCIA) is used to determine the scope of change requests as a basis for resource planning, effort estimation and scheduling. Angelis et al. [2] discusses the importance of the change impact analysis issues during software change implementation process. Arnold and Bohner [3] define a three-part conceptual framework to compare different impact analysis approaches and assess the strengths and weaknesses of individual approach. Gethers et al. [4] propose a framework for impact analysis based on the degree of automation and developer augmentation information, which is obtained from system maintenance scenario. The Pfleeger and Atlee [5] focus on the risks associated with the change and state, "Impact Analysis (IA) is the evaluation of many risks associated with the change, including estimates of the effects on resources, effort, and schedule". However, affect on non-functional properties of the system such as maintainability, readability, reusability etc are also important to analyze before implementing the change. Complexity of the system has the ability to measure the non-functional parameter. Banker et al. [6] examine the relationships between software complexity and software maintainability in commercial software environments. Author proposes a framework to enable researchers (and managers) to assess such products and techniques more quickly by introducing software complexity as a factor linking software development tools and techniques and software maintenance costs. For visualizing the impact of a requested change on the non-functional parameter, it is required to analyze impact on the complexity of the software. There are various metrics proposed by various authors for measuring the software complexity. Hassan et al. [7] proposes complexity metrics that are based on the code change process instead of on the code. We conjecture that a complex code change process negatively affects its product, i.e., the software system. Mc Cabe [8] uses a fundamental assumption that the software complexity is related to the number of control paths generated by the code. Reddy and A. Ananda Rao [9] proposes three metrics: dependency oriented complexity metric for structure (DOCMS(R)), dependency oriented complexity metric

for an artifact causing ripples (DOCMA(CR)), and dependency oriented complexity metric for an artifact affected by ripples (DOCMA(AR)). DOCMS(R) metric value indicated the higher complexity for the structure due to presence of design defects.

The most favorable parameters for measuring the software complexity found in literature are the coupling and cohesion. Chidamber et al. [10, 11] say that classes are coupled if methods or instance variables in one class are used by the other. Coupling Between Object-classes (CBO) for a class is number of other classes coupled with it. For measuring the cohesiveness author also proposes the metric Lack of Cohesion in Methods (LCOM) Number of non-similar method pairs in a class of pairs. Li et al. [12] proposes Data abstraction coupling (DAC) for a class is the number of attributes having other classes as their types. There are a large number of methods for computing the software complexity but the change involves some additional parameters thus the available methods are not sufficient for computing the software change complexity. To compute the software change complexity is still challenging. The main objective of this paper is to propose and implement a novel approach for analyzing the change and design a formula for computing the software change complexity.

3. PROPOSED APPROACH

Complexity is an important issue for software development as it affects the cost, effort and quality of the product. Change complexity could be helpful in analyzing the impact of change. In addition, effort required in implementing change, effort required in testing of the change and effort required for complete system testing after the change has been implemented can be estimated. Cohesion and coupling are very useful property for estimating the quality and complexity of any software. S. Pfleeger [5] defines coupling as the degree of dependence among components and cohesion as the degree to which all elements of components, directed towards a single task and all elements directed towards that task are contained in a single component. High coupling could makes modification process difficult while high cohesion is preferable for software. In the proposed work, coupling and cohesion are considered as the basis for estimating the complexity of requested change. Higher complexity of the change indicates the lesser the possibility of change implementation in the existing software. Complex change may require a huge amount of effort and may harm the system.

Before implementation of the change, determining the change complexity could give a strong criterion for making decision about acceptance or rejection of the requested change. The complexity of the change includes coupling and cohesion as the basic parameters. There are different levels of coupling as well as cohesion. The decrement in the cohesion level and / or increment in coupling level make the system more complex, and maintainability of system may also negatively affected. For computing the complexity of change, it is required to measure the coupling and cohesion. The coupling and cohesion have various levels. For computing complexity, first, we find the coupling and cohesion level along with corresponding weight for existing system class diagram and then same for the new class diagram that includes the proposed changes. Finally, the deviation in the total weight for coupling and cohesion is computed. Table 1 and Table 2 summarize the level of coupling and cohesion with its corresponding weight. As the level of coupling increases, corresponding weight increases and as the level of cohesion decreases, weight of cohesion level increases. Increment in level of cohesion and decrement in level of coupling is desirable to maintain or enhance the software quality.

TABLE I. Coupling Levels with their Weight

Level of Coupling	Weight *(Wcoupling)*	High ↑ Low
Content	6	
Common	5	
Control	4	
Stamp	3	
Data	2	
Uncoupled	1	

TABLE II. Cohesion Levels with their Weight

Level of Cohesion	Weight ($W_{cohesion}$)	Low ↑ High
Coincidental	8	
Logical	7	
Temporal	6	
Procedural	5	
Communicational	4	
Sequential	3	
Informational	2	
Functional	1	

Let we assume that N1 and N2 are the number of external links in between classes in the existing and changed system respectively. C1 and C2 is the number of classes in the existing and changed system respectively. A class may have NK1 and NK2 are number of internal links in existing classes and changed classes respectively, where, N1, N2, C1, C2, NK1 and NK2 are positive integer. Each link carries a weight according to the table 1 and table 2.Thus,

Coupling weight of existing system

$$\text{CupWexisting} = \sum_{j=1}^{N1} W_{Ecoupling(j)} \quad (1)$$

Coupling weight of changed system

$$\text{CupWchanged} = \sum_{k=1}^{N2} W_{Ccoupling(k)} \quad (2)$$

Cohesion weight of existing system

$$\text{CohWexisting} = \sum_{i=1}^{C1} \sum_{m=1}^{NK1} W_{Ecohesion(i,m)} \quad (3)$$

Cohesion weight of changed system

$$\text{CohWchanged} = \sum_{i=1}^{C2} \sum_{m=1}^{NK2} W_{Ccohesion(i,m)} \quad (4)$$

DiffCupW is difference in coupling weight of existing system and changed system

$$\text{DiffCupW} = \text{CupWchanged} - \text{CupWexisting} \quad (5)$$

DiffCohW is difference in cohesion weight of existing system and changed system

$$\text{DiffCohW} = \text{CohWchanged} - \text{CohWexisting} \quad (6)$$

The effects of coupling and cohesion on complexity can summarize in the following tables III:

TABLE III. Coupling Levels with their Weight

Case Id	Case Description	Effect on System Complexity
CA1	If $DiffC_{up}W$ and $DiffC_{oh}W$ both zero	No change in complexity
CA2	If $DiffC_{up}W$ and $DiffC_{oh}W$ both positive	Complexity decreases
CA3	If $DiffC_{up}W$ and $DiffC_{oh}W$ both negative	Complexity increases
CA4	If $DiffC_{up}W$ is positive, $DiffC_{oh}W$ is negative and $\|DiffC_{up}W\| > \|DiffC_{oh}W\|$	Complexity decreases
CA5	If $DiffC_{up}W$ is negative, $DiffC_{oh}W$ is positive and $\|DiffC_{up}W\| < \|DiffC_{oh}W\|$	Complexity decreases
CA6	If $DiffC_{up}W$ is zero and $DiffC_{oh}W$ is positive	Complexity decreases
CA7	If $DiffC_{up}W$ is zero and $DiffC_{oh}W$ is negative	Complexity increases
CA8	If $DiffC_{oh}W$ is Zero and $DiffC_{up}W$ is positive	Complexity decreases
CA9	If $DiffC_{oh}W$ is Zero and $DiffC_{up}W$ is negative	Complexity decreases

4. CASE STUDIES

For validating the proposed approach, we considered the four different case studies. Here we assume that unit difference in cohesion weight decreases one unit complexity and one unit increment in coupling weight increases two units of system complexity. Since addition of one class, include at least addition of one function in the added class. We assume that, the existing system complexity is 50 units for all the cases.

A. A Leap Year Identification System (LYIS) identifies a year as valid or invalid leap year for a given year.

1) Existing System: There is a LYIS version 1.0. A user inputs a valid integer of four digits to the system. System will display the message "The given year is a leap year" if given year is leap year otherwise, system will display "The given year is not a leap year".

2) *Proposed Change:* User input a valid string of four letters for providing the year to the system for identifying the given year is a leap year or not. For example, 1978 will be input as "1978" while "19AB" will be the invalid input.

With the reference of case study taken, figure 1 and 2 are showing the cohesiveness of the existing and changed system respectively.

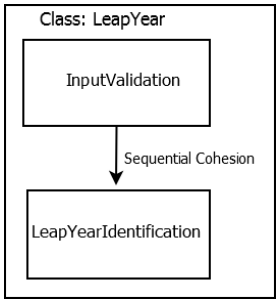

Fig. 1. **Cohesion in Existing LYIS**

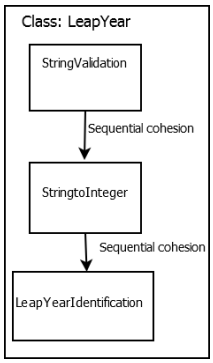

Fig. 2. **Cohesion in Changed LYIS**

In this case, there is no change in coupling since there is only one class. Thus, from Eq. 1, DiffCohW = 0. For computing the value of DiffCohW, we consider Eq. 3, 4 and 6. Here C1 = C2 = 1, NK1 = 1 and NK2 = 2. Referring to the Table I and Table II, for Existing system WEcohesion(11) = 3 and for the changed system WCcohesion(11) = 3, WCcohesion(12) = 3

So, by using equation (3) and (4), CohWexisting = 3 and CohWchanged = 6. On putting these values in equation 6,

DiffCohW = 6 - 3 = 3

Finally, we have DiffCupW = 0 and DiffCohW is positive.

This is indented case 6 from Table III i.e. the CA6. The above results show that there is no change in the coupling, the cohesiveness of the system is increased, and the complexity decreases for the proposed change.

B. A Prime Number Identification System (PNIS) identifies a number as a prime or non-prime for a given number.

1) *Existing System:* There is a There is a PNIS version 1.0. A user inputs a valid integer to the system. System will display the message "The given number is a prime number" if given number is prime number otherwise, system will display "The given number is non-prime".

2) *Proposed Change:* User input a range for finding the prime numbers between the given ranges.

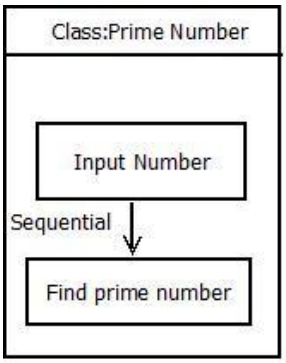

Fig. 3. **Cohesion in Existing PNIS**

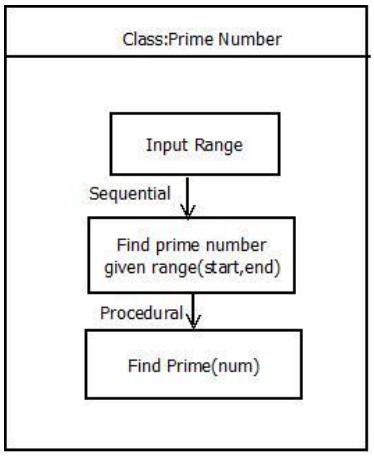

Fig. 4. **Cohesion in Changed PNIS**

With the reference of case study taken, figure 3 and 4 are showing the cohesiveness of the existing and changed system respectively. In this case, there is no change in coupling since there is only one class. Thus, from Eq. 1, $DiffC_{oh}W = 0$. For computing the value of $DiffC_{oh}W$, we consider Eq. 3, 4 and 6.

Here C1 = C2 = 1, NK1 = 1 and NK2 = 2. Referring to Table I and Table II, for Existing system $W_{Ecohesion(11)} = 3$ and for the changed system $W_{Ccohesion(11)} = 3$, $W_{Ccohesion(12)} = 5$

So, by using equation (3) and (4), $C_{oh}W_{existing} = 3$ and $C_{oh}W_{changed} = 8$. On putting these values in equation 6, $DiffC_{up}W = 8 - 3 = 5$. Finally, we have $DiffC_{up}W = 0$ and $DiffC_{oh}W$ is positive, This can be observed in case 6 from Table III i.e. the CA6. The above results show that there is no change in the coupling, the cohesiveness of the system is increased and the complexity decreases for the proposed change.

C. An Age Calculator System (ACS) identifies the employee current age.

1) *Existing System:* There is an ACS version 1.0. The date of birth of the employee of the organization is given as to the ACS 1.0 It calculates the current age of the employee in terms of day month and year

2) *Proposed Change:* System also sends an e-card to the employee email on his birthday or marriage anniversary or on his retirement day.

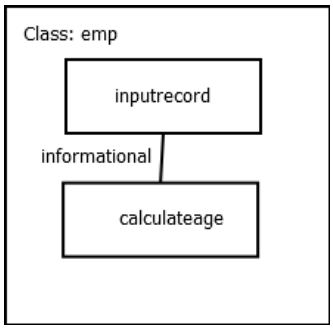

Fig. 5. **Cohesion in Existing ACS**

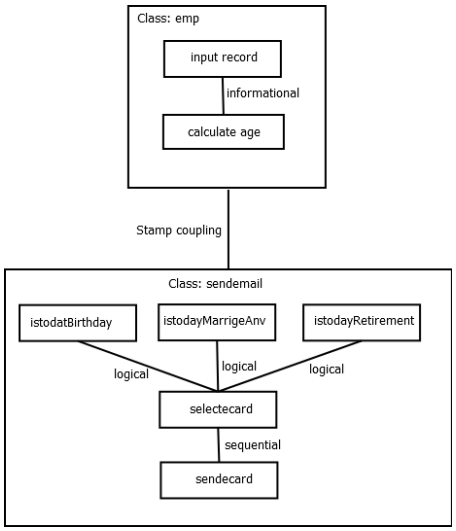

Fig. 6. **Coupling and cohesion in Changed ACS**

With the reference of case study taken, figure 5 and 6 are showing the cohesiveness of the existing and changed system respectively. In this case study, there is a change in both coupling as well as in cohesion. For computing the value of DiffCWup, Eq. 1, 2 and 5 are considered. In the existing system there is only on class while after change there are two classes. Here C1 = 1 , C2 = 2, N1 = 0 and N2 = 1. Referring to Table 2 and Table 3, for Existing system $W_{coupling(1)} = 0$ and for the changed system $W_{coupling(1)} = 2$ So, by using equation (1) and (2), we have $C_{up}W_{existing} = 0$ and $C_{up}W_{changed} = 2$. On putting these values in equation 6, $DiffC_{up}W = 2\text{-}0 = 2$. For computing the value of $DiffC_{oh}W$, Eq. 3, 4 and 6 are considered.

Here C1 = C2 = 1, NK1 = 1 and NK2 = 2. Referring to Table I and Table II, for Existing system $W_{Ecohesion(11)} = 2$ and for the changed system $W_{Ccohesion(11)} = 2$, $W_{Ccohesion(21)} = 7$ $W_{Ccohesion(22)} = 7$ $W_{Ccohesion(23)} = 7$ $W_{Ccohesion(24)} = 2$

So, by using equation (3) and (4), $C_{oh}W_{existing} = 2$ and $C_{oh}W_{changed} = 26$. On putting these values in equation 6, we have, $DiffC_{up}W = 26 - 2 = 24$ Finally, we have $DiffC_{up}W$ = positive and $DiffC_{oh}W$ is positive, This can be observed in case 2 from Table III i.e. the CA2. The above results show that there is no change in the coupling, the cohesiveness of the system is increased and the complexity decreases for the proposed change.

D. Student Information System (SIS) maintains the updated record of the student of acollege.

1) *Existing System:* There is an SIS version 1.0. At the end of the year, it prepares the list of top five students for displaying on the college notice board.

2) *Proposed Change:* Now the college management wants to display the name of the top five students along with few details like father's name and marks secured in the final exam.

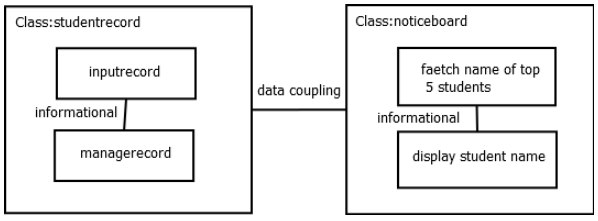

Fig. 7. Coupling and Cohesion in Existing SIS

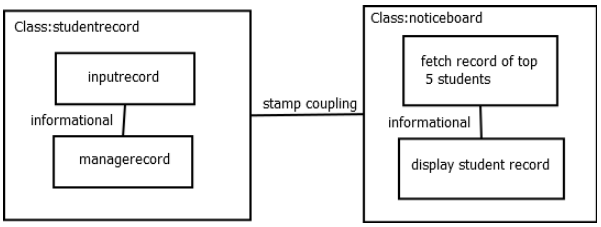

Fig. 8. **Coupling and Cohesion in Changed SIS**

With the reference of case study taken, figure7 and 8 are showing the cohesiveness of the existing and changed system respectively. In this case, there is a change in coupling. For computing the value of DiffCWup, Eq. 1, 2 and 5 are considered. In the existing system there is only on class while after change there are two classes. Here C1 = 2, C2 = 2, N1 = 1 and N2 = 1. Referring to Table I and Table II, For Existing system $W_{coupling(1)} = 2$ and for the changed system $W_{coupling(1)} = 3$

So, by using equation (1) and (2), $C_{up}W_{existing} = 2$ and $C_{up}W_{changed} = 3$ On putting these values in equation 6, $DiffC_{up}W = 3 - 2 = 1$ There is no change in cohesion. Thus, $DiffC_{oh}W$ is zero This can be observed in case 9 from Table III i.e. the CA9. The above results show that the coupling of the system reaches at higher level after the change, the cohesiveness of the system is not affected and the complexity increased for the proposed change.

5. RESULTS

The results indicate that the complexity of the system is negatively affected by adding new classes while the complexity of the system decreases by adding new functions in the existing system. More effort is required in making change, if it includes the addition of new classes as compared to making modifications in the existing classes. It is concluded that the maintainability of the system and the complexity of the system is proportionally related with each other. The complexity of the system is affected by the change in coupling and cohesion. This information could be used in making decision about the acceptance or rejection of the change, the effort estimation in implementing change, effort required in regression testing.

TABLE IV. Comparison among Change in Cohesion, Coupling and Complexity

Case study	Change in Cohesion Weight	Change in Coupling Weight	Change in System Complexity
LYIS	3	0	47
PNIS	5	0	45
ACS	24	2	30
SIS	0	1	52

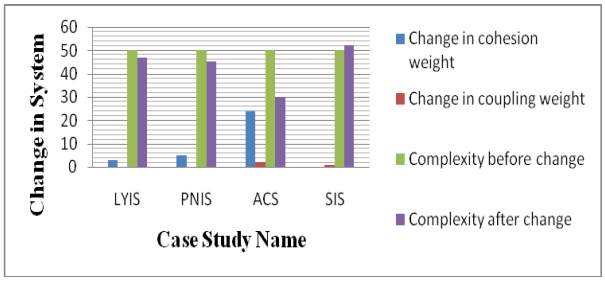

Fig. 9. **Effect of change in coupling and cohesion on system complexity**

Table IV and Fig. 9. shows the changes in coupling and cohesion weights after implementing the changes and the affected system complexity. It validates the proposed approach.

6. CONCLUSION AND FUTURE WORK

In conclusion, this paper proposes a fundamentally new approach that seeks a systematic solution to accept or reject the requested change. In this work, the software change complexity is computed before processing the change. The complexity computation gives a new dimension to the analysts for analyzing the requested change. In addition, with the inclusion of computation of change complexity the criteria of acceptance or rejection of requested change became strengthen. This research is expected to offer realistic solution for analyzing the software change impact of a proposed change on the existing system. The future work is assumed to consider the results in form that are more detailed. We are planning to quantify the complexity so that one can compute the deviation in system complexity because of requested change. We also work in the direction of estimating the effort required in implementing change through software complexity.

7. REFERENCES

[1] Jim Buckley, Tom Mens, Matthias Zenger, Awais Rashid, and Gunter Kniesel. 2005. Towards a taxonomy of software change: Research Articles. J. Softw. Maint. Evol. 17, 5 (September 2005), 309-332.

[2] Lefteris Angelis, and Claes Wohlin. 2008. An Empirical Study on Views of Importance of Change Impact Analysis Issues. IEEE Trans. Softw. Eng. 34, 4 (July 2008), 516-530.

[3] R. S. Arnold and S. A. Bohner, "Impact Analysis - Towards A Framework for Comparison," Proceedings of the Conference on Software Maintenance, Los Alamitos, CA, September 1993, pp. 292-301

[4] Malcom Gethers, Huzefa Kagdi, Bogdan Dit, and Denys Poshyvanyk., "An adaptive approach to impact analysis from change requests to source code", In Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE '11), IEEE Computer Society, Washington, DC, USA, pp. 540-543, 2011

[5] Pfleeger, S.L. and J.M. Atlee (2006)." Software Engineering Theory and Practice Upper Saddle River", New Jersey, USA, Prentice Hall

[6] R. D. Banker, S. M. Datar, and D. Zweig. 1989. Software complexity and maintainability. In Proceedings of the tenth international conference on Information Systems (ICIS '89), Janice I. DeGross, John C. Henderson, and Benn R. Konsynski (Eds.). ACM, New York, NY, USA, 247-255.

[7] Ahmed E. Hassan. 2009. Predicting faults using the complexity of code changes. In Proceedings of the 31st International Conference on Software Engineering (ICSE '09). IEEE Computer Society, Washington, DC, USA, 78-88

[8] Thomas J Mc Cabe, A Complexity Measure, IEEE Transactions on Software Engineering, Vol., SE-2, No. 4, December 1976

[9] K. Reddy Reddy and A. Ananda Rao. 2009. Dependency oriented complexity metrics to detect rippling related design defects. SIGSOFT Softw. Eng. Notes 34, 4 (July 2009), 1-7.

[10] Chidamber, S. R. and Kemerer, C. K. Towards a Metrics Suite for Object Oriented Design. Proceedings of 6th ACM

Conference on Object Oriented Programming, Systems, Languages and Applications (OOPSLA'91), (Phoenix, Arizona, 1991), 197-211.

[11] Chidamber, S. R. and Kemerer, C. K. A Metrics Suite for Object Oriented Design. IEEE Transactions on Software Engineering, Vol. 20 (June 1994), pp.476-493.

[12] Li, W. and Henry, S. Object-Oriented metrics that predict maintainability. Journal of Systems and Software. 23(2) 1993 111- 122.