Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Technology 4 (2012) 754 – 760

Procedia Technology

C3IT-2012

# Development of Encapsulated Class Complexity Metric

A. Yadav[a], R. A. Khan[a]

[a]*D. I.T,Babasaheb Bhimrao Ambedkar University, Lucknow,226025, India*

**Abstract**

Software based system is increasing dynamically over the past decades. Safe and reliable software operation is a significant need for many types of systems. Higher complexity in the software decreases reliability of software. The paper aims to find out the role of Encapsulation in improving reliability of an object oriented design. It is found that higher encapsulation in design increases reliability and decreases complexity of design. Node-Host (N-H) approach is proposed to increase encapsulation of the design. An Encapsulation Class Complexity Metric ($EC_2M$) is proposed to measure complexity of a class design. The same has been implemented for the two versions of designs, one is Design 1 and other is modified design of Design 1 named as ReDesign-1 to evaluate the value of proposed metric.



**Keywords**: Complexity, Encapsulation, Design, Reliability

## 1. Introduction

Object oriented languages are now established and well known to the programming society. In earlier development of software, faults that are likely to cause failure and errors are most likely to be detected during testing phase of software development life cycle. It is best to maximize the probability of faults being detected during design phase while minimizing the probability of faults causing failures during testing phase and implementation phase of software development life cycle. Errors, faults and failures increase complexity of software by making the software more error prone, less reliable, hard to understand, hard to modify and hard to test. There are hundreds of software complexity measures such as Cyclomatic Complexity, Weighted Method per Class, McCabe Complexity, Halstead Metric etc[1][10]. Complexity is a common source of error in software. Complexity is a certain point where software's ability decreases to perform accurate and error free results. Complexity is closely related with faults and errors. The common saying is that the more complex a module is, the more likely it is to persist errors [9]. The certain point where error occurred, it starts increasing sharply and makes the system more complex.


doi:10.1016/j.protcy.2012.05.123

After knowing the facts many organizations have started to limit the complexity of software to increase overall reliability.

Reliability theory is based on the concept of failures [2]. Reliability is defined as the ability of a system or component to perform its required functions under stated conditions for a specified period of time. Reliability is a key factor to quality and software reliability depends on the application of quality attributes at each phase of the software development life cycle. Reliability of software increases if complexity decreases. Complexity has a negative impact on reliability. Several approaches are existing to maximize reliability of software [3]. One approach to increase reliability is through object oriented design constructs. Object oriented design constructs support encapsulation, inheritance, coupling and cohesion. Encapsulation increases reliability thereby decreasing complexity [3]. So it can be concluded that reliability of software may be maximized by maintaining complexity of class in support of maximum encapsulation.

Role of OO design constructs on complexity and reliability is discussed in section 2. Section 3 describes the background of the proposed work. The proposed n-h approach is introduced in section 4. . Description of proposed Encapsulated Class Complexity Metric ($EC_2M$) is included in section 5. Implementation is discussed in section 6. Section 7 includes interpretation. Findings are incorporated in section 8. At last paper is concluded in section 9.

## 2. Role of OO design constructs on complexity & reliability

Object oriented language is a successful language among other languages in the era of $20^{th}$ centaury. Object oriented languages provide facility of inheritance, abstraction, encapsulation, coupling, polymorphism and cohesion thereby improving the ability of software to be reused, tested, maintained and extended. Encapsulation among other OO constructs becomes so popular if encapsulation is maximized during the design phase of software development life cycle. Inheritance and coupling decreases the influence of encapsulation on software in software development life cycle. On the basis of literature survey on software engineering [11-12] suggests that early removal of faults is vital to the success of a reliable software system. Software development life cycle and reliability engineering together allow for the capacity of reliability estimation to designers to help them clarify, examine and assert the reliability requirement in the early phase of the life cycle [6].

Table 1 represents the influence of design constructs on complexity and reliability. Upper mark arrow points high impact and down arrow indicates less impact. If inheritance and coupling of class deign is high, complexity will definitely high and less reliable the system. On the other hand, if there is tight cohesion and encapsulation, complexity will be low and software will highly reliable. To make the system reliability of a class design can be maintained by maximizing encapsulation of a class design.

Table 1: Impact of OO design constructs on complexity and reliability

| | Complexity | Reliability |
|---|---|---|
| Inheritance ↑ | ↑ | ↓ |
| Coupling ↑ | ↑ | ↓ |
| Cohesion ↑ | ↓ | ↑ |
| Encapsulation ↑ | ↓ | ↑ |

## 3. Background

Reliability is a probability of failure free operation in a specified time within a specified environment. Reliability is a function of complexity. When complexity increases reliability of the system decreases.

$$\text{Reliability} = f(\text{complexity}) \quad (1)$$

As the complexity in the software increases, reliability decreases, therefore reliability is inversely proportional to complexity.

$$\text{Reliability} \propto 1/\text{complexity} \quad (2)$$

Increasing demand of anything makes the system more error prone, unreliable, and difficult to understand. More error prone system decreases the reliability of the system by increasing the complexity of the system. The approach proposed is a part of the work which minimizes the complexity of the system by concentrating on object oriented design constructs such as inheritance, coupling, cohesion and encapsulation [4]. Positive and negative impact of object oriented constructs has been examined over the complexity of the system as shown in table 1. In the previous work, it has been proved that inheritance, coupling increases complexity and cohesion decreases complexity [13-15]. The object oriented design construct, encapsulation has already been proven to improve quality of the object oriented software. Complexity increases when design construct encapsulation decreases. Hence complexity is inversely proportional to encapsulation.

$$\text{Complexity} \propto 1/\text{Encapsulation} \quad (3)$$

Hence from (2) and (3)

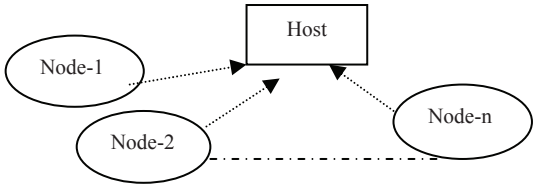
$$\text{Reliability} \propto \text{Encapsulation}. \quad (4)$$

Reliability increases when encapsulation of design is maximum. As reliability is an attribute of quality, and has a close relation with object oriented design constructs. On the basis of forgoing discussion a hypothesis can be proposed: As encapsulation increases, complexity decreases.

## 4. N-H approach

N-H approach is similar to client server approach. Lets us first understand the meaning of client server model. Client server model of computing is a distributed application that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients [5]. Server provides service and client only request services. N-H is node-host approach, similar to client server approach. Fig 1 shows the architecture of N-H model. Researcher considered design classes either node or host. All nodes depend on host class. Host class fulfills the services demanded by node classes. N-H approach will work on action and information. Here action is related with nodes and information is related with host class. What action will be happen is referred to as action taken by nodes and what information will be shared is referred to as information given by host.

Fig 1: N-H (Node-Host) model




Host class is the main class from where service to other classes has to be given. Node class will only request the required service. The class which contain sensitive data means the class which is more error prone and data is more frequently accessible by other classes is host class. Host class should be encapsulated and it should lock away from the node. A class may accessible by three kinds of nodes: external class, subclass and class itself. In OO languages subclasses inherits the behavior and also sometimes the structure of the super class that violates encapsulation by increasing complexity of the software. Self client is a class that is viewed as a client of themselves. What will happen by this is that the structure of the class will be encapsulated in the minimum number of methods or the classes which are

directly depended on the structure of the class should be minimized to some extent. This will provide minimization of an object on its own structure.

## 5. Encapsulated Class Complexity Metric ($EC_2M$)

Object oriented design supports encapsulation. The object oriented design is benefited only when encapsulation of design is maximum. Encapsulation is hiding of data from unauthorized access. Encapsulation is a technique for minimizing interdependencies among separately written modules by defining strict external interfaces. [7]. Complexity of a class can be controlled by maximizing encapsulation of class design. Encapsulated class complexity metric measure two aspects, one is encapsulation aspect and other one is complexity aspect. Here, $EC_2M$ is calculated at method and attribute level. Following metric is used to calculate $EC_2M$:

$$EC_2M = \{\boldsymbol{\Sigma}\ E\ (m\ \&\ a) + \boldsymbol{\Sigma}\ DA\ (m\ \&\ a)\}/\ \boldsymbol{\Sigma}\ T\ (m\ \&\ a)$$

E **(m & a)** (Encapsulated methods and attributes):- It contains only those methods and attributes which are not used by any other class.
**DA (m & a)** (Direct accessible methods and attributes):- It incorporates only those methods and attributes which are used by other classes in a given design.
**T (m & a)** (Total numbers of methods and attributes):- It considers all methods and attributes in a class whether it is encapsulated method and attributes or accessible method and attributes.
**$EC_2M$** (Encapsulated class complexity metric):- Encapsulated class complexity metric comprised of encapsulated method and attributes and accessible methods and attributes. Having more the number of classes, not mean to have more complexity. Complexity of a class design is depended on the number of attributes and methods in classes. A class having more number of attributes and methods is more complex and will provide difficulty in understanding. Researcher tried to control the complexity of a design through the increase of encapsulation in a class design. A design has been taken to calculate $EC_2M$. The taken design is again redesign for the same calculation of metric. Both the designs are than compared for the assessment of proposed metric $EC_2M$.

## 6. Implementation

To exhibit the use of the $EC_2M$ metric a sample system is examined. The proposed metric is implemented for the design version of General Ledger Account System. The accounts are stored in a linked list. The design is taken from the research paper of David P. Tegarden and Steven D. Sheetz [8]. For the sake of simplicity and understanding, the design version is denoted as Design-1 contains 7 classes and ReDesign 1 contains 8 classes are as follows:
**Design-1:** *AccountList, LinkNode, Asset, Liability, OwenersEquity, Revenue, Expense*
**ReDesign-1**: *AccountList, LinkNode, Asset, Liability, OwenersEquity, Revenue, Expense, Encap_Senstive*

**Calculation of $EC_2M$ for design-1**: The class LinkNode and Revenue are two more, Revenue, and Expense. The GetKey method is accessed by LinkNode classerror prone and sensitive classes in the design. The presence of attribute DataPart in class LinkNode and attribute Balance in Revenue makes the class design more error prone and complex to understand. The $EC_2M$ for each class is measured and presented in table-3. The attributes Number, Name and Balance is directly accessible to classes Asset, Liability, OwenersEquity, Revenue, Expense. Methods GetKey, GetDataNode, GetBalance, Update, Print are separately accessible by classes Asset, Liability, OwenersEquity. Here, similar methods are repeating five times and some of them are repeating six time. Repetition of methods and attributes in a class creates

unnecessary complexity and makes the classes more error prone and more sensitive. Table 2 shows the results of computation of $EC_2M$ for design 1.

Table 2: Result of computation of $EC_2M$ for design-1

| Classes of Design-1 | Encapsulation Calculation | | | Complexity Calculation | | |
|---|---|---|---|---|---|---|
| | E(m) | E(a) | $\Sigma${E(m &a)} | DA(m) | DA(a) | $\Sigma${DA(m &a)} |
| AccountList | 5 | 2 | 7 | 2 | 0 | 2 |
| LinkNode | 3 | 3 | 6 | 1 | 0 | 1 |
| Asset | 1 | 0 | 1 | 5 | 3 | 8 |
| Liability | 1 | 0 | 1 | 5 | 3 | 8 |
| Oweners Equity | 1 | 0 | 1 | 5 | 3 | 8 |
| Revenue | 1 | 0 | 1 | 5 | 3 | 8 |
| Expense | 1 | 0 | 1 | 5 | 3 | 8 |
| | E1 | | 18 | C1 | | 43 |
| T(m &a) | | | | | 26 | |
| Encapsulated class complexity metric ($EC_2M$) | | | | | 2.92 | |

**Calculation of $EC_2M$ for ReDesign-1:** ReDesign 1 is a modified version of design 1. ReDesign 1 contains 8 classes. According to N-H approach the classes which are more error prone are considered under a new class encap_senstive. This class contains all those methods and attributes which are used very frequently. Those methods and attributes which are repeating five and six time in a class design are now placed in Encap_Senstive class. By this, repetition of methods and attributes will stop and value of complexity will decrease. When any other class have a need of those methods and attributes, can directly access the particular class. Encap_Sensitive class contains three attributes and five methods: Number, Name, Balance and GetKey, GetDataNode, GetBalance,Update,Print respectively. The attributes and methods of Encap_Senstive class are the repeated methods and attributes of classes Asset, Liability, OwenersEquity, Revenue, and Expense of design 1. In ReDesign 1 these methods and attributes are considers in new class for the purpose of avoiding unnecessary complexity in the design. Table 3 shows the results of computation of $EC_2M$ ReDesign 1.

Table 3: Result of computation of $EC_2M$ for ReDdesign-1

| Classes of Re Design-1 | Encapsulation Calculation | | | Complexity Calculation | | |
|---|---|---|---|---|---|---|
| | E(m) | E(a) | $\Sigma${E(m &a)} | DA(m) | DA(a) | $\Sigma${DA(m &a)} |
| AccountList | 5 | 2 | 7 | 2 | 0 | 2 |
| LinkNode | 3 | 3 | 6 | 1 | 0 | 1 |
| Asset | 1 | 0 | 1 | 0 | 0 | 0 |
| Liability | 1 | 0 | 1 | 0 | 0 | 0 |
| Oweners Equity | 1 | 0 | 1 | 0 | 0 | 0 |
| Revenue | 1 | 0 | 1 | 0 | 0 | 0 |
| Expense | 1 | 0 | 1 | 0 | 0 | 0 |
| Encap_sensetive | 5 | 3 | 8 | 5 | 3 | 8 |
| | E2 | | 26 | C2 | | 11 |
| T(m&a) | | | | | 34 | |
| Encapsulated class complexity metric($EC_2M$) | | | | | 1.08 | |

## 7. Interpretation

For the sake of simplicity encapsulation and complexity calculation for Design 1 is taken as E1 and C1 for ReDesign 1 it is taken as E2 and C2. ReDesign 1 is the modified version of design 1 according to proposed N-H approach. Tabel 4 predicts the value of E1 and C1 is 18 and 43 for Design 1 and for ReDesign 1 e2 and c2 is 26 and 11 respectively. After redesigning of case study Design 1 according to N-H approach it is found that complexity value is less than the earlier calculated value of complexity of Design 1. And also found that value of encapsulation E1 is less than the value of encapsulation E2 for ReDesign1.

Means:

E1<E2 than C2<C1

Hence,

$EC_2M$ (Design 1)> $EC_2M$ (ReDesign 1)

It interpret that when encapsulation increased from E1=18 to E2=26, complexity decreases from C1=43 TO C2=11. Again, it is found that the value of $EC_2M$ of Design 1 is decreased from 2.92 to 1.08 for ReDesign 1 It concludes that high encapsulation in a design decreases the complexity of class design.

Table 4: Summary of Design-1 and ReDesign-1

| | Design 1 | Re Design 1 |
|---|---|---|
| Classes | 7 | 8 |
| Encapsulation | E1=18 | E2=26 |
| Complexity | C1=43 | C2=11 |
| Total Methods and Attributes | 26 | 34 |
| $EC_2M$ | 2.92 | 1.08 |

## 8. Findings

The contribution of the work is summarized as follows:

- Role of design constructs on complexity and reliability has been identified. It is found that encapsulation has a negative impact on complexity and positive impact on reliability. When encapsulation in a design is high, the complexity of the particular design decreases and reliability increases to some extent.
- Reliability is a function of complexity and it is inversely proportional to complexity and directly proportional encapsulation. Hypothesis is taken as encapsulation increases, complexity decreases.
- N-H is Node-Host approach, similar to client server approach. Researcher considered design classes either node or host. All nodes depend on host class. Host class fulfills the services demanded by node classes. N-H approach will work on action and information. Here action is related with nodes and information is related with host class
- Encapsulated Class Complexity Metric (EC2M) is proposed to measure complexity of design hierarchy on the bases of N-H approach.$EC_2M$ IS used to calculate complexity of an object oriented design.
- General ledger account system is considered as a sample for the measurement of proposed metric and for simplicity it is named as Design 1. ReDesign 1 is a modified version of design 1. According to both the designs it is identified that high encapsulation decreases the complexity of class design and hence concluded that Complexity of object oriented software can be maintained by increasing design construct encapsulation.

## 9. Conclusion

Reliability is a function of complexity and it is inversely proportional to complexity and directly proportional encapsulation. N-H approach has been proposed to estimate complexity of class design. N-H is node-host approach, similar to client server approach. Researcher considered design classes either node or host. All nodes depend on host class. Host class fulfills the services demanded by node classes. N-H approach will work on action and information. Here action is related with nodes and information is related with host class. Encapsulated class complexity metric ($EC_2M$) is proposed to measure complexity of class design. General ledger account system is considered as a sample for the measurement of proposed metric and for simplicity it is named as Design 1. ReDesign 1 is a modified version of design 1. According to both the designs it is identified that high encapsulation decreases the complexity of class design and hence concluded that Complexity of object oriented software can be maintained by increasing design construct encapsulation.

## References


1. Arthur H. Watson and Thomas J. McCabe, "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric", NIST Special Publication 500-235, Sept 1996, http://www.mccabe.com/pdf/mccabe-nist235r.pdf , p: 1-124.
2. Alan P. Wood, "Tandem Business Unit Compaq Computer Corporation Cupertino, "Reliability-Metric Varieties and Their Relationships", 2001 Proceedings annual reliability and maintainability symposium, IEEE, p: 110-115.
3. Satwinder Singh, K.S.Kahlon and Parvinder S. Sandhu, "Re-engineering To Analyze and Measure Object Oriented Paradigms", Information Management and Engineering (ICIME), 2010 IEEE, p: 472 – 478, DOI: 10.1109/ICIME.2010.5478013.
4. Yong Cao Qingxin Zhu, "Improved Metrics for Encapsulation Based on Information Hiding", DOI: 10.1109/ICYCS.2008.76, The 9th International Conference for Young Computer Scientists, IEEE computer society, 2008, p: 742-724.
5. Scott L. Bain, "Encapsulation as a First Principle of Object-Oriented Design", http://www.netobjectives.com/resources/articles/first-principle-object-oriented-design,p: 1-15.
6. Yusen Lin and Sourav Bhattacharya, "System Engineering Encapsulation of Reliability Techniques", IEEE, 1999, pp: 138-146.
7. Nathanael Sch aril, Andrew P. Black,and St´ephane Ducasse, " Object oriented Encapsulation for Dynamically Typed Languages", OOPSLA'04 Vancouver, BC, Canada, p. 130–139, Oct. 2428, 2004, ACM 1581138318/04/0010.
8. David P. Tegarden and Steven D. Sheetz, "Effectiveness of Traditional Software Metrics for Object oriented systems", System Sciences, 1992. Proceedings of the Twenty-Fifth Hawaii International Conference, IEEE Explorer, DOI: 10.1109/HICSS.1992.183365 , ISBN: 0-8186-2420-5, 06 August 2002, vol.4, p: 359 - 368.
9. P.L.M. Kelani Bandara, G.N. Wikramanayake and J.S.Goonethillake, "Software Reliability Estimation Based on Cubic Splines", Proceedings of the World Congress on Engineering 2009 Vol I WCE 2009, ISBN: 978-988-17012-5-1, http://www.iaeng.org/publication/WCE2009/WCE2009,p169-173.
10. N.I. Enescu, D. Mancas, E. I. Manole, and S. Udristoiu, "Increasing Level of Correctness in Correlation with McCabe Complexity", International Journal of Computers, 2009, VOL. 3 (1), p: 63-74.
11. S.Singh, K.S. Kahlon, P.S. Sandhu, "Re-engineering to analyze and measure object oriented paradigms ", The 2nd IEEE International Conference on Information Management and Engineering , **ISBN:** 978-1-4244-5263-7,16-18 April 2010,p: 472 – 478,DOI: 10.1109/ICIME.2010.5478013 .
12. N.Sharygina, J. C. Browne, R. P. Kurshan, "A Formal Object-Oriented Analysis for Software Reliability: Design for Verification", Proceedings of the 4th International Conference on Fundamental Approaches to Software Engineering ISBN:3-540-41863-6, 2011, p: 1-15.
13. A.Yadav & R.A Khan, "Measuring design complexity: an inherited method perspective**",** ACM SIGSOFT Software Engineering Notes, ISSN: 0163-5948, July 2009, Vol. 34(4), p: 1-5, Available at: http://dl.acm.org/citation.cfm?doid=1543405.1543427, DOI: 0.1145/1543405.1543427
14. A.Yadav & R.A Khan, "Coupling Complexity Normalization Metric-An Object Oriented Perspective", International Journal of Information Technology & Knowledge Management, Serials Publications, ISSN: 0973-4414**2011**, Vol 4(2), p:501-509**,** http://www.csjournals.com/IJITKM/PDF%204-/Article_33.pdf,
15. A.Yadav & R.A Khan, "Class Cohesion Complexity Metric (C3M)", IEEE International Conference on Computer & Communication Technology IEEE Explorer, ISBN: 978-1-4577-1385-9, 15-17 Sept 2011, p: 363-366, DOI: 10.1109/ICCCT.2011.6075152.