A Novel Approach to Find Reusability using Coupling and Cohesion Metrics

Annushri Sethi1, Prof. Ritu Tandon2

1Student, Department of Computer Science, TCET, Indore, India
2Professor, TRUBA College of Engineering and Technology, Indore-Rao Bypass Road, India

Abstract:
The evaluation of the changeability of software program structures is of most important subject for customers of big structures found in rapid moving domains, which include telecommunications. One way of approaching this problem is to research the dependency between the changeability of the software program and its layout, with the aim of locating design properties that can be used as changeability signs. In the realm of object- orientated systems, experiments have been performed showing that coupling among classes is such an indicator. However, magnificence brotherly love has now not been quantitatively studied in admire to changeability. In this research, we set out to research whether brotherly love is correlated with changeability. As concord metrics, LCC and LCOM have been followed, and for measuring changeability, an alternate impact version changed into used. The facts gathered on three take a look at systems of commercial size suggest no such correlation. Guide investigation of training purported to be weakly cohesive showed that the metrics used do now not seize all of the facets of sophistication cohesion. We finish that cohesion metrics inclusive of LCC and LCOM ought to not be used as changeability indicators.

Keywords:
Cohesion, Coupling, Object Oriented Software, CBO

1. Introduction:

The object-oriented (OO) software improvement era became to begin with delivered inside the early 1990's. OO era employs classes collectively with gadgets and their interdependencies to layout and put into effect structures. OO introduced various underpinning techniques to software improvement that distinguish OO from traditional software improvement paradigm. It's miles used to encapsulate a fixed of closely associated capability in a dependent hierarchy wherein not unusual functionality is added in one elegance and more specialized capability of that magnificence is delivered in other classes.

Manuscript received February 5, 2025
Manuscript revised February 20, 2025
https://doi.org/10.22937/IJCSNS.2025.25.2.16

Item-oriented generation is turning into an increasing number of famous in industrial software improvement environments [7]. This technology facilitates within the improvement of a software product of better high-quality and lower upkeep prices. Since the traditional software metrics targets at the system-orientated software program improvement so it cannot satisfy the requirement of the object-oriented software, as an end result a hard and fast of new object oriented software metrics came into existence. Object orientated Metrics are the measurement gear adapted to the item oriented paradigm to assist control and foster best in software program improvement [7]. OO generation delivered diverse underpinning procedures like idea of training, interfaces and so on. To the software program improvement which distinguish it from traditional software improvement paradigm.

Item/instance is a run time structure with country and conduct. Object kingdom is stored in its fields (variables) and behavior as its methods (capabilities). Magnificence is static description of object [6]. Inheritance is one of the maximum widely used ideas of OO paradigm. It's far used to encapsulate a set of intently associated functionality in a established hierarchy wherein commonplace functionality is introduced in one magnificence and more specialized functionality of that class is brought in other training. The specialized training inherits the common capability from their great elegance and uploads their very own greater functionality. The primary subject of inheritance is to promote reusability in a machine.

2. Cohesion:

Cohesion may be a live that defines the degree of intra-dependability inside components of a module. The bigger the cohesion, the higher is that the program style below figure shows how to determine cohesion module.

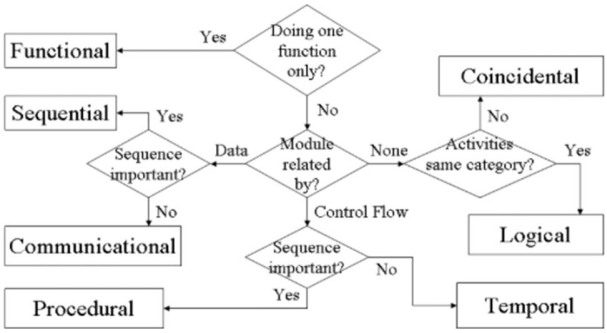

Figure 1: Determine Cohesion Modules

3. Coupling:

Coupling may be a live that defines the amount of inter-dependability among modules of a program. It tells at what level the modules interfere and act with one another. The lower the coupling, the higher the program.

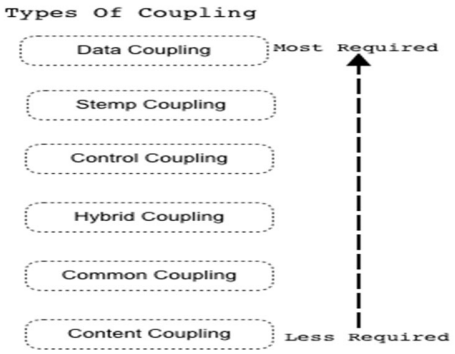

Figure 2: Type of Coupling and its importance

4. Literature Review:

Literature almost about the software evolution genuinely introduces the erosive developments inside the software architecture at the same time as meeting the changes imposed by using the software program evolution. On this thesis, we can try to become aware of such erosive tendencies with the help of class brotherly love and coupling metrics. Based totally at the literature assessment, we suppose that both magnificence cohesion and coupling need to follow deteriorating developments at the same time as evolution within the software architecture.

Table. 1 Literature survey

Author Name / Title	Journal	Strength	Weakness
N. Rajkumar1 ”Measuring Cohesion And Coupling In Object Oriented System Using Java Reflection”	ARPN Journal of Engineering and Applied Sciences	This paper proposes a set of new measures to find coupling and cohesion in a developmental system using Java reflection components to assess the usability. It will predict the fault in an object-oriented system.	Next version will calculate coupling and cohesion metrics for UML representations
Martin Hitz “Measuring Coupling and	http://www.isys.uni-klu.ac.at/PDF/1995-	This distinction refers to dynamic dependencies between objects on one hand	important aspects of software quality at

Cohesion In Object-Oriented Systems “	0043-MHBM.pdf	and static dependencies between implementations.	run-time and during the maintenance phase, respectively.
Aaron B. Binkley “A classical view of object-oriented cohesion and coupling”	http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.99.4519&rep=rep1&type=pdf	Evidence is starting to accumulate that this paradigm is indeed as effective as has been suggested	Most of the metrics used in conjunction with the object-oriented paradigmare, in fact, classical metrics.
Mr. KailashPatidar “Coupling and Cohesion Measures in Object Oriented Programming”	International Journal of Advanced Research in Computer Science and Software Engineering	A large numbers of metrics have been built and proposed for measuring properties of object-oriented software such as size, inheritance, cohesion and coupling. The coupling is an important aspect in the evaluation of reusability and maintainability of components or services.	To achieve consistent and satisfying results, empirical data obtained from reallifesoftware engineering projects
Shweta Sharma “A review of Coupling and Cohesion metrics in Object Oriented Environment”	International Journal of Computer Science & Engineering Technology (IJCSET)	This paper focuses on two very significant factors of complexity measurement of software, which are coupling and cohesion. An extensive study of approximately all types of coupling and cohesion metrics has been reported in this paper	Very little work has been done in areas of dynamic coupling and cohesion metrics and need further more investigations

5. Proposed Work:

Object oriented design is becoming greater famous in software development environment and object orientated design metrics is a vital part of software program surroundings. Metrics measure certain residences of software gadget through mapping them to numbers (or to different symbols) in keeping with well-described, objective dimension guidelines. Design Metrics are measurements of the static kingdom of the project’s design and extensively utilized for assessing the size and in a few cases the pleasant and complexity of software program. Analysis and preservation of object-oriented (OO) software is costly and difficult.

We take two C# applications one implemented with inheritance and one with interface. Then we follow concord Metrics Tight class cohesion (TCC) and unfastened magnificence cohesion (LCC) at the applications to calculate the cohesion fee and evaluate the result. On the premise of result we differentiate between complexities of inheritance and interface.

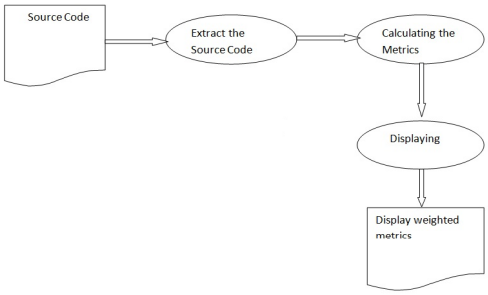

Figure 3. Proposed System Architecture

6. Result Analysis:

In this paper we take two programs as an input.We consider an inheritance program and one with maximum possible interface program in C#. Calculate number of joint and disjoint sets. Apply cohesion metrics on the calculated values. Compare the result.

7. Evaluation Parameters:

Software functionality very well, and also how can we use the software functionality in new

environment thus we can find our purpose with few fault and few pace. And it also increases the ratio since we utilized software functionality effectively to receive the desire purpose of the project. Understandability components are calculated by using of the following metrics and the descriptions metrics are:

1) Number of Association per class metric (NASSocC)
The Number of Association per Class metric is defined as the total number of associations a class has with other classes or with itself. When the number of associations is less the coupling between objects are reduced [29]. Brian introduced this metric.

2) Number of Dependencies In metric (NDepIn)
The quantity of Dependencies In metric is defined because the range of instructions that depend upon a given elegance [29]. When the dependencies are reduced the elegance can characteristic extra independently.

3) Number of Dependencies Out metric (NDepOut)
This metrics carried out for measuring the dimensions of this system through thinking about the no of lines in software. strains of Code (LOC) counts all traces like as supply line and the number of statements, the number of comment lines and the quantity of clean traces [39].

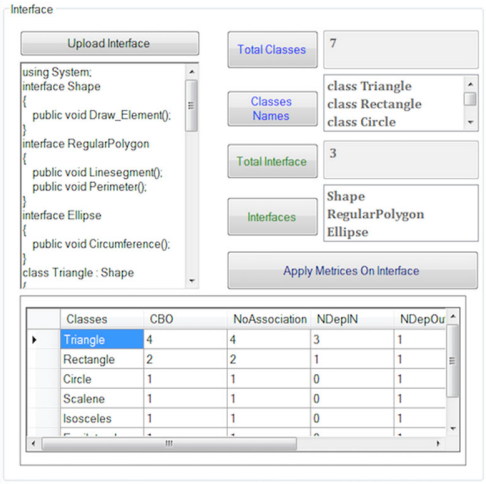

Figure5.3: Calculate CBO, No of Association, Number of Dependencies In metric and Number of Dependencies out metric for Interface Program

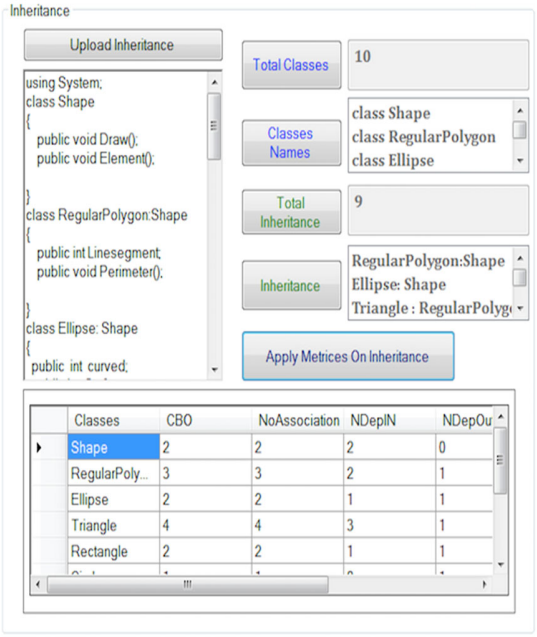

Figure 4: Calculate CBO, No of Association, Number of Dependencies In metric and Number of Dependencies out metric for Inheritance Program

4) Lines of Code (LOC):
This metrics applied for measuring the size of the program by considering the no of lines in program. Lines of Code (LOC) counts all lines like as source line and the number of statements, the number of comment lines and the number of blank lines [28].

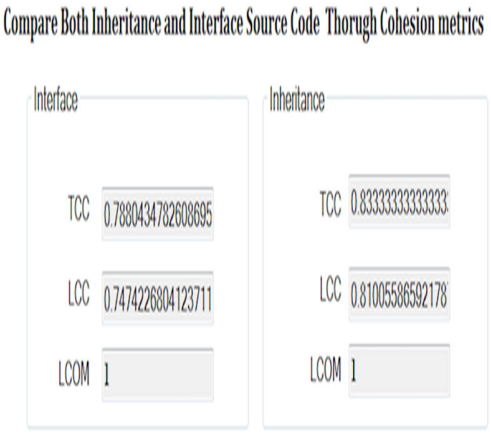

Figure 5: Calculate TCC ,LCC and LCOM metric for Inheritance and Interface Program

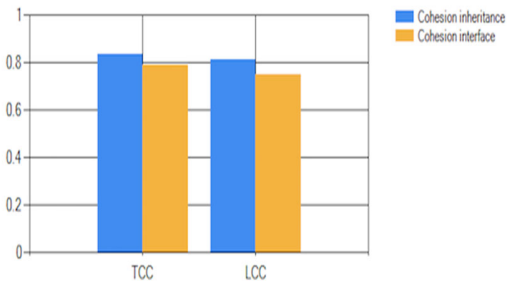

Figure 6: Graph show TCC ,LCC and LCOM metric for Inheritance and Interface Program

5) Comment Percentage (CP):

CP is computed by number of comment line separated along Line of Code. High evaluate of the CP increases the maintainability and understandability [39].

CP = Comment Line / LOC;

6) Weighted Method per Class (WMC):

This metrics is applied towards calculating the structure complexity of the programs. Method complexity is measured by using Cyclomatic Complexity and WMC is sum of complexity of the all methods, which is applied in class.Suppose class is getting the methods (m1, m2, and m3…mn) and complexity of the methods are (c1, c2, and c3…cn) then

WMC = c1+c2+c3+…. +cn;

Cyclomatic Complexity causes foundation of the graph theory and is computed in one of the 3 directions. Number of regions in flow graph.Cyclomatic Complexity determined in flow graph as follow

C (G) = E – N +2;

Where N is the no of the nodes in graph and E is the no off the edge in the graph.Cyclomatic Complexity defined in flow graph as follow

C (G) = P+1;

Where 'P' is number of predicate nodes in the graph.Statement where we are taking some decision are called predicate node [39].

7) Depth of Inheritance Tree (DIT):

This metric is applied for measuring the inheritance complexity for the programs, when programmer usages the inheritance in his program then this Metric can be utilized. DIT is the Maximum depth from the root node of tree to special node. Here class is represented as a node. Deeper node in the tree accepts more no of the methods because they inherit and the more classes in the tree and it make the class more complex [23]. DIT metric is the length of the maximum path from the node to the root of the tree. So this metric calculates how far down a class is declared in the inheritance hierarchy. The following figure shows the value of DIT for a simple class hierarchy. DIT represents the complexity of the behavior of a class, the complexity of design of a class and potential reuse.

8) Flexibility:

It is defined as "the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed" [43]. Flexibility is considered as a factor affecting the reusability of a component. Flexibility =1 - [(0.5 X Coupling) + (0.5 X Cohesion)], Coupling = CBO, Cohesion = LCOM.'

9) Understandability:

It is defined as "the ease with which a system can be comprehended at both the system-organizational and detailed statement levels" [43].Understandability is considered a factor of reusability. Understandability = 1 - [(0.25 X Coupling) + (0.25 X Cohesion) + (0.25 X Comments) + (0.25 X Size)].

10) Independence:

The term "independence" is introduced to reflect the property of the system concerning the ability of a class to perform its responsibilities on its own. Independence is measured by DIT. Other classes inherit the classes lower in the hierarchy; these classes depend on their ancestors to perform their functionalities [43]. Portability = Independence = 1 - adjusted DIT.

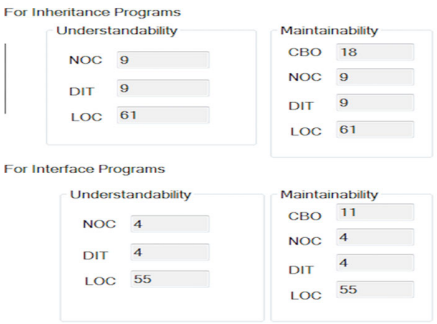

Figure 7: Calculate NOC, DIT and LOC metric for Inheritance and Interface Program

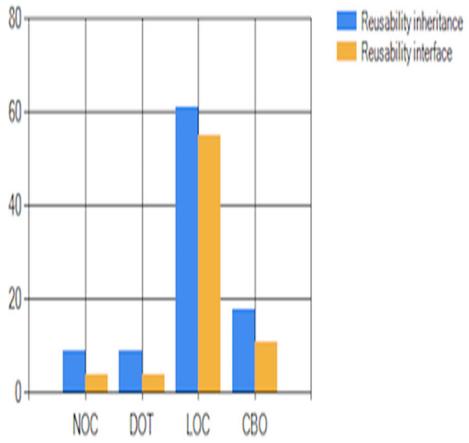

Figure 8: Graph shows NOC, DIT and LOC metric for Inheritance and Interface Program

Figure 9: Calculate Size, Flexibility, Portability and Indecency metric for Inheritance and Interface Program

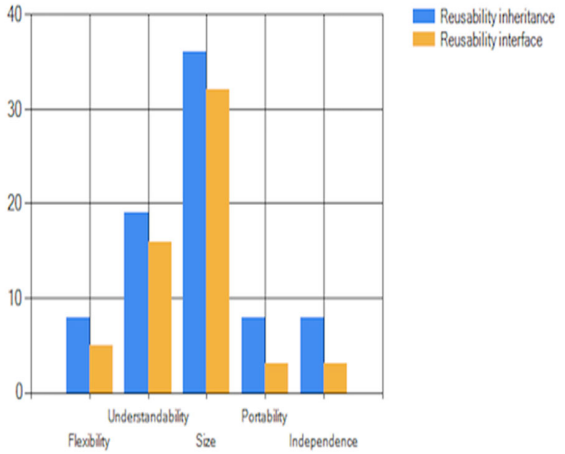

Figure 10: Graph shows Size, Flexibility, Portability and Indecency metric for Inheritance and Interface Program

8. Conclusion:

The reason of this thesis is to locating the approach and way to perceive complexity between inheritance and interface programming via concord metrics in item orientated packages. Metrics measure certain homes of software program device via mapping them to numbers (or to other symbols) according to properly defined, goal measurement guidelines.

Code Metrics are measurements of the static kingdom of the project's Code and extensively utilized for assessing the dimensions and in some cases the first-rate and complexity of software. Analysis and upkeep of object-orientated (OO) software program is highly priced and hard. As a consequence, measuring the relationships has turn out to be a prerequisite to broaden efficient strategies for analysis and protection. Diverse concord metrics had been proposed and used in past empirical investigations; however none of these have taken the run-time houses of software into account. "To improve modularity and encapsulation the inter magnificence brotherly love measures need to be large. By using greater interfaces compared to inheritance the coupling measures are reduced. True abstractions normally show off high cohesion. In evaluation of concord in among inheritance and interface for the modules, capabilities, attributes, classes in oops thru concord metrics is carried out, and interface is calculated as greater reusable code

than inheritance. The extra unbiased a category it's miles easier to be reused with the aid of any other software."

9. Future work:

Having delivered a framework for a complete metric for brotherly love in item-orientated structures on class levels, we are capable of discover a fundamental assessment of brotherly love and concluded the reusability of code by way of differencing among inheritance and interface in order that the proposed problem can be resolved theoretically but it is able to be enforce almost, to be able to make available the decreased price and complexity for development of in practical international. The similarly advanced metrics are given that also can be implement in realistic behavior in order that a green manner can be recognized to optimize our approach for improvement of IT merchandise.

References:

[1] V. Krishnapriya, K. Ramar, "Exploring the Difference Between Object Oriented Class Inheritance and Interfaces Using Coupling Measures," ace, pp.207-211, 2010 International Conference on Advances in Computer Engineering, 2010

[2] K.K.Aggarwal, Yogesh Singh, ArvinderKaur, RuchikaMalhotra. "Empirical Study of Object-Oriented Metrics",2006

[3] Martin Hitz, BehzadMontazeri."Measuring Coupling and Cohesion.In Object-Oriented Systems" in Angewandte Informatik (1995)

[4] James M. Bieman andByung-kyookang."Cohesion and Reuse in Object Oriented System" Department of Computer Science, Colorado State University Fort Collins,Colorado,1995

[5] Shyam R. Chidamberand Chris F. Kemerer" A Metrics Suite For object Oriented Design" IEEE Transactions on software Engineering, Vol. 20, No. 6, June 1994

[6] KrishnaprasadThirunarayan." Inheritance in Programming Languages" Department of Computer Science and Engineering ,Wright State University ,Dayton, OH-45435

[7] ArtiChhikara Maharaja Agrasen College, Delhi, India. R.S.Chhillar "Applying Object Oriented Metrics to C#(C Sharp) Programs"Deptt. Of Computer Sc.And Applications, Rohtak, India.SujataKhatriDeenDyalUpadhyaya College, Delhi, India(2011)

[8] Christopher L. Brooks, Chrislopher G. Buell, "A Tool for Automatically Gathering Object-Oriented Metrics", IEEE, 1994

[9] Friedrich Stiemann, Philip Mayer and Andreas Meibner, "DecouplingClasses with Inferred Interfaces", Proceedings of the 2006 ACMSymposium on Applied Computing, P.No:1404 – 1408.

[10] Pradeep Kumar Bhatia, Rajbeer Mann, " An Approach to Measure Software Reusability of OO Design", Proceedings of 2nd International Conference on Challenges & Opportunities in InformationTechnology(COIT-2008),RIMT-IET,MandiGobissndgarh, March 29, 2008.

[11] Fried Stiemann, Wolf Siberski and Thomas Kuhne, " Towards the Systematic Use of Interfaces in Java Programming", 2nd Int. Conf. on the Principles and practice of Programming in Java PPJ 2003, P.No:13-17.

[12] Girba, T.; Lanza, M.; Ducasse, S. (2005) Characterizing the Evolution of Class Hierarchies. Proceedings of the 9th European International Conference on Software Maintenance and Reengineering.Manchester, UK, pp.2-11.

[13] Gilb, T. (1976) Software Metrics. Chartwell-Bratt, Cambridge, MA.

[14] Hall, T., Rainer, A., Jagielska, D. (2005) Using software development progress data to understand threats to project outcomes. Proceedings of the 11th IEEE International Software Metrics Symposium (METRICS 2005). Como, Italy, 10 pages.

[15] Harrison R., Counsell S. and Nithi R.: "Experimental Assessment of the Effect of Inheritance on the Maintainability of Object-Oriented Systems", the Journal of Systems and Software, vol. 52, pp. 173-179, 2000.

[16] Henry, S.M., Kafura, D.G. (1981) Software structure metrics based on information flow. IEEE Transactions on Software Engineering, 7(5):510-518.

[17] Hudli, R., Hoskins, C., Hudli, A., "Software Metrics for Object-oriented Designs", IEEE, 1994.

[18] Judith Barnard," A new reusability metric for object-oriented software Journal software quality control volume 7 issue 1, 1998.

[19] Kemerer, C.F. and Slaughter, S. (1999) An Empirical Approach to Studying Software Evolution. IEEE Transactions on Software Engineering, 25(4):493-509.

[20] Ken Pugh," Interface Oriented Design", Chapter 5, 2005.

[21] Lee, Y., Liang, B., Wang, F., "Some Complexity Metrics for Object-Oriented Programs Based on Information Flow", Proceedings: CompEuro, March, 1993, pp. 302-310.

[22] L.C., Briand, Daly, J., Wust, J. (1999b) A unified framework for coupling measurement in object-oriented systems. IEEE Transactions on Software Engineering, 25(1):91-121.
[23] Lehman, M. M., Programs, Cities, Students, Limits to Growth?, Inaugural Lecture, in Imperial College of Science and Technology Inaugural Lecture Series, Vol. 9, 211-229 (1970, 1974). Also in Programming Methodology, (D. Gries. ed.), Springer Verlag, 42-62 (1978). Reprinted in Lehman and Belady, 1985.
[24] Lorenz, Mark and Kidd, Jeff, Object-Oriented Software Metrics, Prentice Hall Publishing, 1994.
[25] Lorenz, M., Kidd, I. (1994) Object-Oriented Software Engineering Metrics, Prentics-Hall Englwood Cliff, NJ.
[26] Marcela Genero, Mario Piattini and Coral Calero,“ A Survey of Metrics for UML Class Diagrams”, in Journal of Object Technology,Vol. 4, No. 9, Nov-Dec 2005.
[27] McCabe, T. (1976) A software complexity measure. IEEE Transactions on Software Engineering, 2(4):308-320.
[28] Mohsen D. Ghassemi and Ronald R. Mourant,”Evaluation of Coupling in the Context of Java Interfaces”, Proceedings OOPSLA 2000, P. No: 47-48, Copyright ACM 2000, 1-58113-307-3/00/10.
[29] Mathew Cochran, “Coding Better: Using Classes Vs Interfaces”, January 18th, 2009.