International Journal of Research

International Journal of Research (IJR)

e-ISSN: 2348-6848, p- ISSN: 2348-795X Volume 2, Issue 05, May 2015

Available at http://internationaljournalofresearch.org

Review paper on Coupling and Cohesion

Neelima Saini & Sunita Mandal

Dronacharya college of Engineering, Gurgaon

neelimasaini1994@gmail.com, coolsunitamandal@gmail.com

1. Introduction

software engineering (SE) has attempted to use software measures and models to reduce complexity, and thereby achieve other goals, such as greater productivity. However, complexity cannot always be reduced Coupling is usually contrasted with cohesion. Low coupling often correlates with high cohesion, and vice versa. Low coupling is often a sign of a well-structured computer system and a good design, and when combined with high cohesion, supports the general goals of high readability and maintainability.Cohesion is an ordinal type of measurement and is usually described as “high cohesion” or “low cohesion”. Modules with high cohesion tend to be preferable because high cohesion is associated with several desirable traits of software including robustness, reliability, reusability, and understandability whereas low cohesion is associated with undesirable traits such as being difficult to maintain, test, reuse, and even understand. The objective of this paper is to understand how software design decisions affect the structural complexity of software. This is important because variations in the structural complexity of software can cause changes in managerial factors of interest, such as effort and quality.

This paper makes a number of research and practical contributions. The critical role of the concepts of coupling and cohesion in structuring software is theoretically established. This assists in moving from a general notion of software structure to an understanding of specific factors of structural complexity. Complexity analysis typically proceeds by considering coupling and cohesion independently. Based on theoretical and empirical evidence, this research argues that they must be considered together when designing software in order to effectively control its structural complexity.By studying software at higher levels the effect of design decisions across the entire life cycle can be more easily recognized and rectified.

2 Abstract

*This research examines the structural complexity of software, and specifically the potential interaction of the two most important structural complexities: coupling and cohesion .Coupling and Cohesion are the two terms which very frequently occur together. The coupling is an important aspect in the evaluation of reusability and maintainability of components or services. The coupling metrics find complexity between inheritance and interface programming. in software engineering, **coupling** is the manner and degree of interdependence between software modules; a measure of how closely connected two routines or modules are;[1] the strength of the relationships between modules. In computer programming, **cohesion** refers to the degree to which the elements of a module belong together. Thus, it is a measure of how strongly related each piece of functionality expressed by the source code of a software module is. The theory-driven approach taken in this research considers both the task complexity model and cognition and lends significant support to the developed model for software complexity. Furthermore, examination of the task complexity model steers this paper towards considering complexity in the holistic sense of an entire program, rather than of a single program unit, as is conventionally done*

Keywords-

Software complexity; software structure; task complexity; coupling; cohesion

3 Conceptual Background

It is widely believed that software complexity cannot be described using a single dimension. The search for a single, all encompassing dimension has been likened to the "search for the Holy Grail". To find such a dimension would be like trying to gauge the volume of a box by its length, rather than a combination of length, breadth, and height. Early attempts to determine the key dimensions of software complexity have included identifying factors in single or small numbers based on observing programmers in the field or adapting, refining, and/or improving existing factors.

3.1 Performance using coupling

Whether loosely or tightly coupled, a system's performance is often reduced by message and parameter creation, transmission, translation (e.g. marshaling) and message interpretation (which might be a reference to a string, array or data structure), which require less overhead than creating a complicated message such as a SOAP message. Longer messages require more CPU and memory to produce. To optimize runtime performance, message length must be minimized and message meaning must be maximized.

3.2 Message Transmission Overhead and Performance

Since a message must be transmitted in full to retain its complete meaning, message transmission must be optimized. Longer messages require more CPU and memory to transmit and receive. Also, when necessary, receivers must reassemble a message into its original state to completely receive it. Hence, to optimize runtime performance, message length must be minimized and message meaning must be maximized.

Message Translation Overhead and Performance

Message protocols and messages themselves often contain extra information (i.e., packet, structure, definition and language information). Hence, the receiver often needs to translate a message into a more refined form by removing extra characters and structure information and/or by converting values from one type to another. Any sort of translation increases CPU and/or memory overhead. To optimize runtime performance, message form and content must be reduced and refined to maximize its meaning and reduce translation.

Message Interpretation Overhead and Performance

All messages must be interpreted by the receiver. Simple messages such as integers might not require additional processing to be interpreted. However, complex messages such asSOAP messages require a parser and a string transformer for them to exhibit intended meanings. To optimize runtime performance, messages must be refined and reduced to minimize interpretation overhead.

The degree of coupling and its relation to efficiency of energy conversion in multiple-flow systems

The description of systems of two coupled flows in terms of their "degree of coupling" (Kedem & Caplan, 1965) is extended to systems in which more than two coupled flows occur. The degree of coupling between any pair of flows is defined, and related to a generalized overall degree of coupling between sets of flows. As in two-flow systems, it is uniquely related to the maximum efficiency of energy conversion; this is by virtue of the fact that the overall degree of coupling, although not in general independent of the forces or the flows, reaches a maximum value defined by the phenomenological matrix in a particular series of stationary states. These "maximum coupling states" are states of minimal entropy production under conditions of energy conversion, and include as limiting cases the states previously described as static head and level flow

3.3 Coupling And Cohesion

Given two lines of code, A and B, they are **coupled** when B must change behavior only because A changed.

They are **cohesive** when a change to A allows B to change so that both add new value.

The difference between CouplingAndCohesion is a distinction on a process of change, not a static analysis of code's quality

- Does "must change behavior" refer to source code changes, or run-time results?
- *It refers to explicit modification of behavior - i.e. the source code. Automatically adapting run-time results in a value-added way more closely matches the concept of **cohesion**, below. 'Coupling' is more readily identified by the way things break. Consider: the only reason your B "must" change behavior as cause of A changing is that the change to A broke the behavior of B.*

It's my summary judgement that "change" is too open-ended to make this a rigorous concept, a discussed below. There's an effectively infinite way any given module can "change". One has to first "tame" and classify "change" as a prerequisite to rigor-tizing C-and-C if it's tied to the term "change". --top

The rubric is chosen to give the term the most value in an AgileSoftwareDevelopment context; hardening code against bugs by pushing C1 down & C2 up.

These are some of the better-defined qualities that separate good software from bad software. Although they were formalized during the invention of StructuredProgramming, they apply exactly as well toObjectOrientedProgramming as to any other kind.

Cohesion of a single module/component is the degree to which its responsibilities form a meaningful unit; higher cohesion is better.

- Someone had vague reference to decomposability here. Clarification?
- How about: 'Cohesion is inversely proportional to the number of responsibilities a module/component has.'

Coupling between modules/components is their degree of mutual interdependence; lower coupling is better.

- size: number of connections between routines
- intimacy: the directness of the connection between routines
- visibility: the prominence of the connection between routines
- flexibility: the ease of changing the connections between routines

A first-order principle of software architecture is to increase cohesion and reduce coupling.

4 Conclusion

Low Coupling and High Cohesion are as you may see very related to each other. Both leads to a better and less fragile systems where the maintainability, testing and good reuse are favoured. Separation of Concerns is a principle or mechanism that would help us achieving this goal.

5 References

[1] Shikha Gautam et al, Int.J.Computer Technology & Applications,Vol 4 (1),155-161

[2] ***Journal of Software Engineering and Applications*, 2012, 5, 671-676** http://dx.doi.org/10.4236/jsea.2012.59079 Published Online September 2012 (http://www.SciRP.org/journal/jsea)

[3] I. Vanderfeesten, H. A. Reijers and W. M. P. van der Aalst, “Evaluating Workflow Process Designs Using Cohesion and Coupling Metrics,”

Computers in Industry, Vol. 59, No. 5, 2008, pp. 420-437. doi:10.1016/j.compind.2007.12.007

[4] T. M. Meyers and D. Binkley, "An Empirical Study of Slice-Based Cohesion and Coupling Metrics," ACM Transactions on Software Engineering and Methodology, Vol. 17, No. 1, 2007, Article No. 2.

[5]ISO/IEC/IEEE 24765:2010 Systems and software engineering — Vocabulary

[6] ISO/IEC TR 19759:2005, Software Engineering — Guide to the Software Engineering Body of Knowledge (SWEBOK)

[7] F. Beck, S. Diehl. On the Congruence of Modularity and Code Coupling. In Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (SIGSOFT/FSE '11), Szeged, Hungary, September 2011. doi:10.1145/2025113.2025162

[8] Pressman, Roger S. Ph.D. (1982). Software Engineering - A Practitioner's Approach - Fourth Edition. ISBN 0-07-052182-4