DOI: http://dx.doi.org/10.26483/ijarcs.v8i8.4617

ISSN No. 0976-5697

Volume 8, No. 8, September-October 2017

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

CONVENTIONAL TO COMPONENT-BASED SOFTWARE: A CRITICAL SURVEY ON INTERACTION AND INTEGRATION COMPLEXITIES

Umesh Kumar Tiwari
Department of Computer Science and Engineering
Graphic Era University, Dehradun
Dehradun, India

Abstract: There are various standard paradigms of software development including conventional, modular, object-oriented and component-based software engineering (CBSE). Interaction and integration complexities of various piece of code play a vital role in the overall behavior of software. As the code count increases the interaction level of software also increases as per the requirements of the software. In this paper we perform a critical literature survey on the works of eminent researchers and practitioners. In this work we analyze three paradigms of development, namely, conventional software, object-oriented software and component-based software (CBS). In this survey, we have considered three parameters of comparison: measures and metrics used, key findings, and factors affecting the interaction and integration behavior of software.

Keywords: object-oriented; component-based software; interaction; integration; complexity

I. INTRODUCTION

In general, complexity is termed as the assessment of hardware and software resources needed by software. In software development, complexity is treated as an indirect measurement unlike the direct measurements like lines-of-code or cost-estimation [1]. Internal as well external interactions contribute a major role in software complexity. In the context of software development, interaction behaviour of various parts of program is used to measure the complexity. These parts may be single line code, a group of line of codes (functions), a group of functions (modules) or ultimately components. As the size of parts of s software increases, the count of interactions will also increase, as well as the complexity.

Software Engineering principles are applicable on the applications developed through either development paradigm. Component-based software development (CBSD) emphasizes "development with reuse" as well as "development for reuse". Development with reuse focuses on the identification, selection and composition of reusable components. The property of reusability is not applied only to develop the whole system but also to develop the individual components. The development for reuse is concerned with the development of such components that may be used and then reused in many applications, in similar and heterogeneous contexts.

After discussing the introduction of work in section 1, we have summarized the interaction and integration issues in section 2. In section 3, we have performed the survey on the literature available. Finally section 4 concludes this work.

II. INTEGRATION AND INTERACTION ISSUES

Software applications are composed of dependent or independently deployable components. Assembling of these components has a common intension to contribute their functionalities to the system. Technically this assembling is referred to as integration of and interaction among components. We have sufficient number of measures and metrics to assess the complexity of stand alone programs as well as small-sized conventional software, suggested and practiced by numerous practitioners [2, 3, 4, 5, 6, 7, 8]. In literature, complexity of programs and software is treated as a "multidimensional construct" [3, 9].

III. LITERATURE SURVEY

Thomas J. McCabe [10] developed a method to assess the Cyclomatic complexity of a program. He used control-flow graph of code to compute the complexity. McCabe used graph theoretic notations to draw the control-flow graph where a graph denoted as 'G' having 'n' number of nodes, 'e' number of connecting edges and 'p' number of components. Cyclomatic complexity V(G) calculated as, $V(G) = e - n + 2p$, where 2 is the "result of adding an extra edge from the exit node to the entry node of each component module graph" [2]. In control-flow graph, a sequential block of code or a single statement is represented as a node, and control flows among these nodes are represented as edges. Cyclomatic complexity metric is easy to compute and maintenance, gives relative complexity of various designs.

Finally, Halstead's [5] identified a complete set of metrics to measure the complexity of a program considering various factors. These metrics include the program vocabulary, length, volume, potential volume, and program level. Halstead proposed methods to compute the total time and effort to develop the software. These metrics are based on the lines of codes of the program. He defined program vocabulary as the count of distinct operators and distinct operands used in the program. The count of total operators and operands used in a program is proposed as the Program length. The Program volume has been defined as the storage volume required representing the Program, and the representation of program in the shortest way without repeating operators and operands is known as potential volume. Halstead has also defined the relationship between these factors and metrics of programs.

Alan Albrecht [6] proposed Function-point analysis technique to measure the size of a system in terms of functionalities provided by the system. FPA categorizes all

the functionalities provided by the software in five specific functional units: External inputs provided to the software, External outputs provided by the software, External inquiries of the system under consideration, Internal logical files presents data and content residing in the system, and External interface files are the data and contents residing with other systems and can be called to system under consideration. Three complexity weights High, Low and Medium are associated with these functional units using a set of pre-defined values. In function-point analysis, 14 complexity factors have been defined, which have a rating from 0 to 5. On the basis of these factors, Alan calculated the values of unadjusted function-point, complexity adjustment factors, and finally the value of function points [2].

Henry and Kafura [11] proposed a set of complexity computation method for software modules. Author's suggested a "Software Structure Metrics Based on Information Flow that measures complexity as a function of fan-in and fan-out" [12]. Authors proposed the complexity as "the procedure length multiplied by the square of fan-in multiplied by fan-out." This method is used to calculate the count of "local information flows" coming to (fan-in) and going from (fan-out) the module. Henry and Kafura defined a length of the module as the procedure length which calculated with the help of LOC or McCabe's complexity metric. This metric can be computed comparatively early stage of the development.

Kenneth Morris [13] proposed some object-oriented metrics to assess complexity and productivity metrics. Author's identified some complexity factors like Maintainability, Reusability, Extensibility, Testability, Comprehensibility, Reliability and Authorability, that they called "productivity impact variables". Morris proposed a complete set of nine eligible metrics for Methods, Class, Inheritance, Coupling and Cohesion.

Boehm [7] developed the 'object-point' metric through level of complexity of the amount of screenshots, reports and components. The level of complexities is categorized as simple, medium or difficult.

Chidamber and Kemerer's [14] proposed a metric suite for object-oriented software called as CK Metrics-suite. This metric suite is one of the most detailed and popular research works for object-oriented applications. Authors defined metric suite for complexity, coupling cohesion, depth of inheritance, and response set. These metric set are used to asses the complexity of an individual class as well as the complexity of the entire software system. In their metrics, Chidamber and Kemerer used Cyclomatic method for the complexity computation of individual classes.

Abreu and Rogerio Carapuca [15, 16, 17] proposed a metric set named 'Metrics for Object-Oriented Design'. In this metric suite, two fundamental properties of object-oriented programming are used, attributes and methods. Authors proposed metrics for the basic structural system of object-oriented idea as encapsulation, inheritance, polymorphism, and message passing. This suit consists of metrics for methods and attributes as assessment method for encapsulation.

Cho et al. [18] developed some measure for the quality and complexity of components for CBSE. They used mathematical equations and expressions in their metrics. In their work, authors identified three categories of complexity, quality of component, customizability and reusability. They used size, costs, efforts, and reuse level as the complexity factors.

Narasimhan et al. [19] suggested couple of metrics to assess the complexity of Component-Based Software. The packing density metric maps the count of integrated components, and the interaction density metric is used to analyse the interactions among components. They identified some constituents of the component in their work; these constituents include line of code, operations, classes, and modules. Authors also suggested a set of criticality criteria for component integration and interaction.

Vitharana et al. [20] developed a method for fabrication of components. Authors suggested some managerial factors like cost-efficiency; assembling easiness, customization, reusability, and maintainability. These are used to estimate technical metrics as coupling-cohesion, count, volume and complexity of components. They developed 'Business Strategy-based Component Design' model.

Rashmi Jain et al. [21] assesses the association and mappings of cause-and-effect among the requirements of the system, structural design of the system and the complexity of the procedure of the systems integration. They argued the requirement of fast integration of components so that the complexity impact of integration on architectural design of components can be controlled. Authors identified 5 major factors to analyse the integration complexity of software system. Further these factors are divided into 18 sub-factors including commonality in hardware and software subsystems, percentage of familiar technology, physical modularity, level of reliability, interface openness, orthogonality, testability and so on.

Trevor Parsons et al. [22] proposed some specific dynamic methods for attaining and utilising interactions among the components in component-based development. They also proposed component-level interactions that achieve and record communications between components at runtime and at design time. For their work, authors used Java components.

Lalit and Rajinder [23] proposed a set of integration and interaction complexity metrics to analyse the complexity of Component-Based Software. They argue that complexity of interaction have two implicit features, first within the component, and second interaction from the other components. Their complexity metrics include percentage of component interactions, interaction percentage metrics for component integration, actual interactions, and total interactions performed, complete interactions in a Component-Based Software.

Some complexity assessment techniques for CBSE are on the basis of complexity properties including communication among components, pairing, structure, and interface. The interaction and integration complexity measures available in the literature are explored considering the development paradigms like: Convention Software and Programs, Objet-Oriented Software, and Component-Based Software and summarized in Table 1.

Table I. Summary of Interaction and Integration Complexities

Paradigm	*Measures and Metrics Used*	*Key Findings*	*Factors affecting Interaction and Integration Complexity*	*Author(s)/ References*
Conventional Software and Programs	• Line of Code, • Interaction among Statements, • Nodes and Interactions	• Author used control flow graph of a program to compute the Cyclomatic complexity. • McCabe used graph theoretic notations to draw the control flow graph where a graph *G* with *n* vertices, *e* edges and *p* connected components.	• Conditional Statements, • Loop Statements • Switch cases	Thomas J. McCabe [10]
Conventional Software and Programs	• Line of Code, • Count of operators, • Count of Dissimilar operands, • Total count of Dissimilar operators, • Total count of Dissimilar operands.	• Proposed a complete set of metrics to measure the complexity of a program considering various factors, like Program vocabulary, Program length, Program volume, • Potential volume, and others.	• Program Vocabulary, • Program Length, • Program Volume, • Effort, • Time	M. H. Halstead [5]
Modular Programming	• External inputs • External outputs • External Enquiries • Internal logical files • External Interface files	• Proposed Function-point analysis technique to measure the size of a system in terms of functionalities provided by the system.	• 5 functional units, • 14 Complexity factors, • Complexity adjustment factors, • Degree of influence.	Alan Albrecht and J. E. Gaffney [6]
Modular Programming	• Fan-in information, • Fan-out information, • Complexity of the module • Line of Code • McCabe Cyclomatic Complexity	• Author's suggested a "Software Structure Metrics Based on Information Flow that measures complexity as a function of fan-in and fan-out".	• Number of calls to the module, • Number of calls from the module, • Length of the module.	S. Henry and D. Kafura [11]
Object-Oriented Software	• Methods, • Inheritance, • Coupling, • Cohesion, • Object Library Effectiveness, • Factoring Effectiveness, • Method Complexity • Application Granularity.	• Proposed some object-oriented metrics to assess complexity and productivity metrics, including Average number of methods per object class, Inheritance tree depth, Average number of uses dependencies per object, Arcs, and Degree of cohesion of objects.	• Maintainability, • Reusability, • Extensibility, • Testability, • Comprehensibility, • Reliability and • Authorability	Kenneth Morris [13]
Object-Oriented Software	• Lines of code to count the size, • Number of Screenshots, • Number of reports.	• Authors suggested the object-point metric that is computed using counts of the number of screenshots, reports and components based on their complexity levels. • Complexity levels are classified as simple, medium or difficult.	• Line of Code, • Complexity Levels.	B. Boehm [7]
Object-Oriented Software	• Cyclomatic method, • Class complexity, • Methods, • Object-oriented properties.	• Proposed one of the most detailed and popular research works in the context of object oriented software, including Weighted Method per Class, Depth of Inheritance Tree, Number Of Children.	• Complexity, • Coupling, • Cohesion, • Inheritance, • Number of children, and	S. Chidamber and C. Kemerer [14]

			• Response set	
Object-Oriented Software	• Method Hiding Factor, • Attribute Hiding Factor, • Method Inheritance Factor, • Attribute Inheritance Factor for Inheritance, • Polymorphism factors, • Coupling factors	• Authors identified two fundamental properties of object-oriented programming are used, attributes and methods. • The Method Hiding Factor and (MHF) and Attribute Hiding Factor (AHF) are proposed together as measure of encapsulation.	• Encapsulation, • Inheritance, • Polymorphism, and • Message passing	Fernando Brito and Rogerio Carpuca [15]
Component-Based Software	• Levels of complexity, • Quality of components, • Customizability.	• Proposed metrics to measure the quality and complexity of components. • They used mathematical equations and expressions in their metrics.	• Size, • Costs, • Efforts, and • Reuse level	E.S. Cho, M.S. Kim, and S.D. Kim [18]
Component-Based Software	• Indicates these values as high or low, • Establishes a relationship among these proposed metrics.	• Proposed metrics through a hierarchical model consisting of three layers, quality, criteria and metrics.	• Understandability, • Adaptability, and • Portability	H. Washizaki, Y. Hirokazu and F. Yoshiaki [19]
Component-Based Software	• Line of code, Operations, • Classes, and • Modules, • Number of components	• Suggested two sets of metrics to assess the complexity of Component-Based Software. • Two complexity metric suites Component Packing Density metrics and Component Interaction Density.	• Risk associated with components, • Constituents, • Interactions among components,	Narasimhan et. al. [20]
Component-Based Software	• Coupling, • Cohesion, • Number of Components, • Component Size, • Complexity.	• Proposed a methodology for fabrication of components.	• Syntax and • Semantics	Padmal Vitharana, Hemant Jain, and Fatemeh “Mariam” Zahedi [21]
Component-Based Software	• Prioritization of Requirements, • Functional Modularity, • Feasibility, • Interface, • Testability	• Assesses the association and mappings of cause-and-effect among the requirements of the system, architecture of the system and the complexity of the procedure of the systems integration. • Identified 5 major factors to analyse the integration complexity of software system. • Further these factors are divided into 18 sub-factors	• Commonality in hardware and software subsystems, • Percentage of familiar technology, • Physical modularity, • Level of reliability, • Interface openness, • Orthogonality, testability	Rashmi Jain, Anithashree Chandrasekaran, George Elias, and Robert Cloutier [23]
Component-Based Software	• Static Interaction complexity, • Dynamic Interaction complexity, •	• Proposed some specific dynamic methods for attaining and utilising interactions among the components in component-based development. • Component-level interactions that achieve and record communications between components at runtime and at design time.	• Call traces, • Call graphs, • Runtime paths • Calling context trees	Trevor Parsons, Adrian Mos, Mircea Trofin, Thomas Gschwind, and John Murphy [24]
Component-Based Software	• Interface, • Implementation,	• Proposed a set of integration and interaction complexity	• Maintainability, • Reusability,	Latika Kharb, Rajender Singh

	• Deployment, • Incoming and • Outgoing interactions	metrics to analyse the complexity of Component-Based Software, including Percentage of component Interactions, Interaction percentage metrics for component integration.	and • Reliability	[25]

IV. CONCLUSION

After Methods and metrics proposed so far in the literature are defined on the basis of interactions among instructions, operations, procedures, and functions of individual and standalone programs and codes. These metrics are appropriate for small-sized codes. Some measures are also defined for object-oriented software, but for CBSE applications these methods are not inadequate. In the CBSE, components have connections and communications with each other to exchange services and functionalities. Interaction edges are used to denote the connections among components. So there is an edge for each requesting communication and similarly an edge for each responding communication. But practitioners and researchers have not included both edges in their complexity computations. They have used single edge theory in their graph representations and in all their assessments, which is not true for CBSE.

V. REFERENCES

[1] B. W. Boehm, M. Pendo, A. Pyster, E. D. Stuckle, and R. D. William, “An Environment for Improving Software Productivity”, IEEE Computer, June 1984.

[2] S. Pressman Roger, “Software Engineering A practitioners Approach. Sixth Edition”, TMH International edition, 2005.

[3] S. Wake and S. Henry, “A Model Based on Software Quality Factors which Predict Maintainability”, In Proc. Conference on Sofmare Maintenance, 1988, pp. 382-387.

[4] V. R. Basili and D. H. Hutchens, “An Empirical Study of a Syntactic Complexity Family”, IEEE Transactions on Software Engineering, vol. 9, no. 6, pp. 664-672, November 1983.

[5] M. H. Halstead, “Elements of Software Science”, New York: Elsevier North Holland, 1977.

[6] Alan Albrecht and J. E. Gaffney, Software Function Source Line of code and Development Effort Prediction: A Software Science Validation, IEEE Trans. Software Engineering, SE-9, 639-648, 1983.

[7] B. Boehm, “Anchoring the Software Process”, IEEE Software, vol. 13, no. 4, pp. 73-82, 1996.

[8] M. M. Lehmam, and L. A. Belady, “Program Evolution - Processes of Software Change”, 1985.

[9] Usha Kumari and S. Bhasin, “A composite complexity measure for component-based systems”, ACM SIGSOFT Software Engineering Notes, vol. 36, no. 6, Nov 2011.

[10] T. McCabe, “A complexity measure”, IEEE Transactions on Software Engineering, vol. 2, no. 8, pp. 308–320, 1976.

[11] S. Henry and D. Kafura, "Software Structure Metrics Based on Information Flow", IEEE Transactions on Software Engineering, vol. 7, pp. 510-518, 1981.

[12] http://en.wikipedia.org/wiki/complexity.

[13] K. Morris, "Metrics for Object Oriented Software Development", Masters thesis, M.I.T., Sloan school of management, Cambridge, MA, 1989.

[14] S. Chidamber and C. Kemerer, "A Metrics Suite for Object - Oriented Design", IEEE Transactions on Software Engineering, vol. 20, issue 6, pp. 476-493, 1994.

[15] F. B. Abreu and Rogerio Carapuca, "Object-Oriented Software Engineering: Measuring and Controlling the Development Process", 4th International Conference on Software Quality, McLean, VA, USA, 1994, pp. 3-5.

[16] F. B. Abreu 1995, "Design Metrics for Object-Oriented Software System", In Proc. Workshop on Quantitative Methods £COOP, 1995, pp. l-30.

[17] F. B. Abreu and W. Melo, "Evaluating the Impact of Object-Oriented Design on Software Quality", 3rd International Software Metrics Symposium, Berlin, Germany, 1996.

[18] E.S. Cho, M.S. Kim, and S.D. Kim, Component Metrics to Measure Component Quality, Proceedings of the Eighth Asia-Pacific on Software Engineering Conference (APSEC '01), IEEE Computer Society, Washington, DC, USA, 2001, pp.419-426.

[19] V. L. Narasimhan and B. Hendradjaya, "Theoretical Considerations for Software Component Metrics", Transactions on Engineering, Computing and Technology, vol.10, pp. 169-174, 2005.

[20] Padmal Vitharana, Hemant Jain, and Fatemeh “Mariam”, Zahedi, “Strategy-Based Design of Reusable Business Components”, IEEE Transactions on Systems, Man, and Cybernetics—PART C: Applications and Reviews, vol. 34, no. 4, Nov 2004.

[21] Rashmi Jain, Anithashree Chandrasekaran, George Elias, and Robert Cloutier, “Exploring the Impact of Systems Architecture and Systems Requirements on Systems Integration Complexity”, IEEE Systems Journal, vol. 2, no. 2, June 2008.

[22] Trevor Parsons, Adrian Mos, Mircea Trofin, Thomas Gschwind, and John Murphy, “Extracting Interactions in Component-Based Systems”, IEEE Transactions on Software Engineering, vol. 34, no. 6, Nov/Dec 2008.

[23] Latika Kharb, Rajender Singh, “Complexity Metrics for Component-Oriented Software Systems”, ACM SIGSOFT Software Engineering Notes, vol. 33 no. 2, March 2008.