Component-based Software System Dependency Metrics based on Component Information Flow Measurements

Majdi Abdellatief[ab], Abu Bakar Md Sultan[a], Abdul Azim Abd Ghani[a], Marzanah A.Jabar[a]
[a]Department of Information System, Faculty of Computer Science & Information Technology,
University Putra Malaysia, 43400 Serdang, Selangor, Malaysia
[b]Mehareeba Technical College, Technical Education Corporation, 2081 Khartoum, Sudan
Khwaja24@yahoo.com, {abakar, azim, marzanah}@fsktm.upm.edu.my

***Abstract*-The motivation of this paper is that the measurement based on the flow of information connecting software components can be used to evaluate component-based software system dependency. The ability to measure system dependency implies the capability to locate weakness in the system design and to determine the level of software quality. In this paper, dependency between components is considered as a major factor affecting the structural design of Component-based software System (CBSS). Two sets of metrics namely, Component Information Flow Metrics and Component Coupling Metrics are proposed based on the concept of Component Information Flow from CBSS designer's point of view. We also discuss the motivation for and possible uses of system level metrics and component level metrics. Initial results from our on-going empirical evaluation indicate that the proposed metrics are very intuitive.**

Keywords-Component-based software system; Software metric; Dependency; Information flow.

I. INTRODUCTION

In Component-based development (CBD) paradigm, Component-based software system (CBSS) are developed using a set of independent components which work together. Some of these components may be developed in-house, while others may be third-party components, without source code [1]. Nowadays, this development methodology has become one of the predominant software engineering solutions for the design of a large and a complex system [2].

Analysis of CBSS dependencies is an important part of software research for understandability [3], testability [4], maintainability [5] and reusability [6][7] of a component-based system. Thus, dependency metrics could have a real impact on the quality of the system delivered to the user. If valid dependency metrics could be identified, they could provide the information required by developers, testers and maintainers to understand the system, identify the critical components, evaluate the impact of change in one component on the other components and even to support the future evolution of the CBSS when adding, removing and modifying some components. It is difficult to perform such tasks without understanding potential component dependencies [8]. In addition, a large and complex CBSS should be evaluated early at the specification phase, to avoid faults, poor interaction among components and failure of one component which could lead to a total system failure [9][10].

Previous research conducted in CBSS metrics concentrated on one of two major areas. Many research papers [11][12][13], focused on measuring the reusability of software components, while others [2][10][14][15][16], focus on measuring the interaction complexity of integrated components. In the past, only a few papers based on graph theory addressed the evaluation of CBSS dependency [8][10][17][18][19]. However, there has been no theoretical or empirical validation conducted for the proposed metrics. In this paper, interface dependency is considered to be the main dependency affecting CBSSs. Interface dependency exists as relationships among different functionalities and parameters of software components. For example, when one interface relies on other to obtains functionalities necessary for its own tasks. However, if the components produced by component providers only include specifications of the interfaces [19][20], the interface specification does not supply adequate information for analysis of integrated CBSS dependency. Thus, in CBSSs, due to the black box nature and the separation of interface specification from its implementation, the analysis of information flows will be quite difficult using the traditional information flow techniques. Therefore, we first proposed a new method named Component Information Flow (CIF) to analyze the information flows into a component, out of a component and between components. We believe that the CIF is a more suitable and practical basis for characterizing and evaluating CBSS for several reasons. First, often the component's internal structure is not available. Second, the elements of CIF could be directly determined at design phase. Third, the availability of metric values early in the design phase allows the CBSS structure to be corrected with the least cost. Fourth, as seen in the subsections of this paper, it's based on standard Information flow [21], which is considered more sensitive than other measurements.

Based on the concept of CIF, we also propose two sets of metrics, namely, Component Information Flow Metrics and Component Coupling Metrics that represent the CBSS designer's point of view (they are also relevant to testers and maintainers). The proposed metrics depict details about the quality of a structure design at three levels, entire CBSS level, component level and interface level. For each level they concern with the way in which components or interfaces connect.

This paper is organized as follows: Section II describes research methodology. Section III illustrates component-based information flow definitions and concepts. Section IV provides the definition of the metrics and their description. Section V applies our proposed metrics in a small scale example and discusses the results. The conclusion and direction for future work are in Section VI.

II. RESEARCH METHODOLOGY

The metrics are derived in the following steps:

1. Conducting systematic mapping study on existing CBSS metrics and metrics validation techniques.
2. Defining information flow for CBSS.
3. Defining a new dependency metrics for CBSS specification.
4. Application of the proposed metrics in a small scale example.

In step 1, a systematic mapping study of the values for various metrics was carried out by the authors of this paper and the limitations of the current research were drawn from them in “unpublished” [22]. The third author suggested step 2. The planning, data collection and reporting of steps 2 and 3 were performed by the first author with respect to the context defined in Section III. Each step and its content was checked and reviewed by the rest of authors independently and carefully. In case there is ambiguity point, a negotiation took place. Particularly, step 2 was investigated many times since it is considered as the core of this study. Step 4 was conducted by all of the authors as stated in Section V.

III. COMPONENT INFORMATION FLOW CONCEPTS AND CONTEXT

To provide a context for this Section and the next Section, we need a background of software component and CBSS specification method. Components definition adopted in this study clearly fall under Szyperski’s definition [1]. The CBSS structural specification method used is that of Cheesman and Daniels [23]. Our measurement approach assumes that the proposed approach is generally applicable to developments using any of the technology standards such as Sun’s EJB, Microsoft’s COM+ and CORBA Models.

A. *Software Component Concept*

We visualize software component concepts from the perspective of component developers and CBSS designers. Figure 1 provides a simplified model of a component such that a specification defines the functionality and behaviour of a component which is composed of an interface part and a body part. The specification and interface are visible to CBSS designers, whereas the specification, interface and body are visible to component developers.

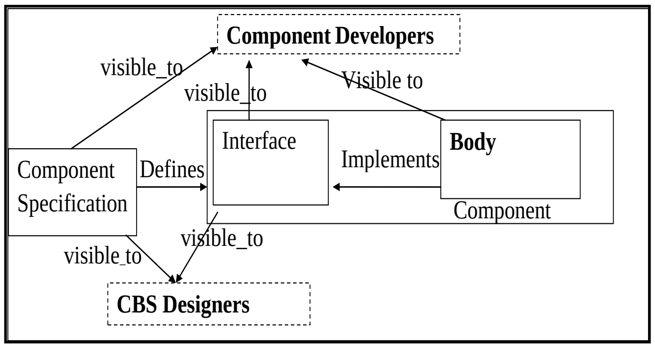

Figure 1. Simplified component model

The interface definition includes a collection of one or more operations to specify the functionality and behaviour identified in the specification. The body of the system implements the external methods and any other internal methods that are required to provide the functionality and behaviour identified in the specification. Metrics may be derived from the specification, interface or body but only metrics derived from the interface and specification can be used by CBSS designers.

B. *Definition of Component Information Flow*

This subsection describes the mechanisms for deriving the various types of component information flow based on the above assumptions.

The separation of interface from implementation is a core principle of component based development. That is, the functionality specified in the interface could be implemented in different applications by different programming languages. Therefore, it is important to view interfaces and their specifications separately from any specific component that may implement or use such interfaces. To explain this view, it suffices to consider the interface of a component to define the component’s access point [24]. These access points allow clients of a component, usually components themselves, to access the functions provided by the component. Normally, a component could have multiple access points corresponding to different functions provided in the interface [1].

In Figure 2, we depict this view from an interface perspective. This model focuses on what the interface must do to fulfill the client’s information required without considering how this will be accomplished. With respect to the proposed model in Fig 3, for any component in CBSSs, two boundaries are considered: (1) Interface boundary which separates the provider interface from a client interface. The client might be a user, a required interface or an engineering device. (2) The body boundary which separates the provider interface from its implementation.

Component Information Flow (CIF) is characterized by two types of flows, Inter-component flow and Intra-component flow. In the Inter-component flow, the provider interface communicates with client to exchange information by In-flows and Out flows. Thus, the information flows across the interface boundary. The In-flow carries information from a client to a provider interface through the

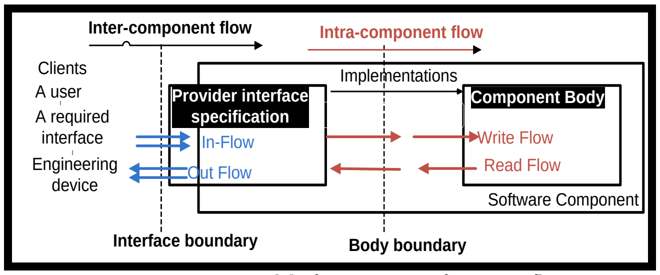

Figure 2. Generic model of component information flow

list of in-parameters. The Out flow carries information from a provider interface to a client through the list of out-parameters. In the Intra-component flow, it is assumed that the data structure is used (i.e., a component body) to store and retrieve the information needed by the provider interface, represented by Read flow and Write flow. Thus, the information flows across the body boundary. In other words, an intra-component flow takes place when an interface retrieves data from or updates a data structure.

An important characteristic of the CIF described above is that the knowledge essential to build the complete flow structure can be established from a simple analysis of a UML requirements specification. The UML modeling technique describes the component specification, the component interaction diagram and the interface specification to design the intended CBSS. Component specifications name the interfaces that a component adhering to the specification must implement. An interface specification consists of a set of operation specifications. An interface specification has to specify how the inputs, outputs, and component object state are related, and what the effect of calling the operation has on that relationship [23]. An operation specifies an individual action that an interface will perform for the client. Each action shows one or more types of information flow (i.e., In-flow, Out flow, Read flow or Write flow), where each type of information flow shows one possible execution flow. Thus, for each interface we can identify all potential flows from the interface specification. To facilitate the mapping of CIF to a complete flows structure, we describe a template for CIF analysis and data collection as shown in Table 1.

To understand the relationship between components and make the concept of CIF clear, consider an example presented in Figure 3, which shows three components, A, B, C and their relationship to each other. This example is purely from the specification perspective. It is assumed that some functionality required by component "A" is implemented by "B" and "C. We depict the information flow among components as a result of methods calling and events firing as Inter-component flow, and the information flow inside the components to update or retrieve from component store as Intra-component flow. The information flow from component "A" to "B" or "A" to "C" can be represented by a set of direct inter-flows plus a set of intra-flows, whereas the information flow from "B" to "C" can be represented by a set of indirect inter-flows plus a set of intra-flows.

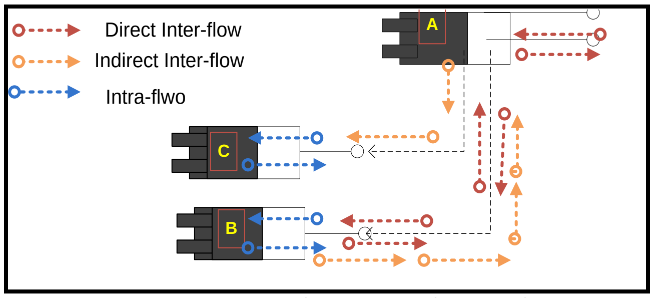

Figure 3. An example of component information flow

The following definitions describe precisely the terms and the four types of information flow presented informally above. These four types of flow identify the logical flow of information between components. The reader should refer to Figure 3 to understand definitions 1, 2 and 4, and Figure 3 to understand definition 4, 5, 6 and 7.

Definition 1: Information flow is the set of messages streaming across the boundaries which define a particular communication between two components based on the logical representation of the interface specification.

Definition 2: There is an Intra-component flow of information from component "B" to component "A" if a component "B" implements some functionality of component "A".

Definition 3: There is an Inter-component flow of information from component "A" to component "B" if one or more of the following conditions hold:

1) If a component "A" invokes a component "B" and passes information to it; or component "B" returns a result to a component "A" (termed direct inter-component flow).

2) If a component "A" invokes both a component "B" and a component "C" passing output values from "B" to "C" (termed indirect inter-component flow).

Definition 4: In-flow is an inter-component flow type and carries information provided or passed from a client entity to a provider interface.

Definition 5: Out flow is an inter-component flow type and carries information returned from a provider interface to a client entity.

TABLE 1. TEMPLATE FOR COMPONENT INFORMATION FLOW ANALYSIS AND DATA COLLECTION

Interfaces	**operations**	**Operation Description**	**Information Flow Types**	**Source of Information Flow**	**Destination of Information Flow**
Each component can consists of one or more interfaces	Each interface can consists of one or more operations	Each operation could be described as a set of messages with respect to the definitions information flow (i, e., definitions 1, 4, 5, 6, and 7)	In-flow	Client interface	Provider Interface
			Out flow	Provider interfaces	Client interface
			Read flow	Provider Interface	Component store
			Write Flow	Provider interface	Component store

Definition 6: Read flow is an intra-component flow type and carries information retrieved from a component store to a provider interface.

Definition 7: Write flow is an intra-component flow type and carries information from a provider interface to update component store.

C. CIF supports:

- a variety of software architectures from simple stand-alone application to large distributed software based on OSI 7 layers or J2EE n-tiers. Therefore, almost any kind of CBSS structure can be analyzed and evaluated.
- all stages of the software life cycle. Analysis can be carried out as early in the requirement specifications or as late in the life cycle as necessary.
- a defined measurement unit. An elementary unit of CIF defined by us is a base flow type (i.e., in-flow, out flow, read flow and write flow)

IV. Definition of Dependency Metrics

We use measurement based on the flow of information to evaluate and mange dependencies between components in the CBSS. Particularly, we use the following metrics to characterize the effect of dependency on the structure design of CBSS.

A. Component Coupling Metrics

In our literature survey, we found inconsistencies in the definition of coupling in the literature [6][7][25][26][27][28]. There were several different definitions of coupling, depending on the measurement goal and entity being measured (i.e., inheritance coupling, messages passing coupling or data abstraction coupling) [29]. Thus, the coupling attribute has been defined, measured and interpreted in various ways. Xia [27] studied this ambiguity of coupling concept and redefined it based on its essence. We adopted his definition here. "*Component coupling of m is the impact-dependence of components to m*". The impact-dependence of X2 to X1 means that when X1 is modified, there will be an impact on X2. For example, when changing component X1 in Figure 4, we only need to consider how component X2 will be affected. Component X2 returns F1 and F2 to component X1. F1 and F2 are out-flows of component X2 and in-flows of component X1 which will influence component X1 when component X2 is changed. But when X1 is modified, F1 and F2 have no impact on X2. Therefore, the right definition should consider only the out flow of X1 for its coupling. Another important source which could influence the change in X1 is the number of distinct components receiving the out flows [30]. For example, an impact on a component that depends on one component is not the equivalent to a component that depends on three components, even if both components receive the same number of out flows.

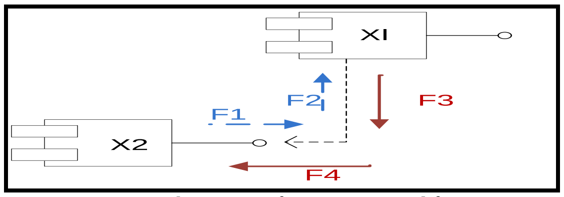

Figure 4. The impact of component modification

Assumption 1: The more the spread of inter-flow from a component, the larger the context of its interface operations and the more the external information required to test and maintain the components.

Accordingly, we defined coupling metrics as

$$\text{Interface Coupling (IC)} = n \times \sum_{i=1}^{p} OF_i$$

where

p = number of operations in an interface

OF_i = number of out flows in each operation (i)

n = the number of other component to which an interface is coupled

$$\text{Component Coupling (CC)} = \sum_{i=1}^{p} IC_i$$

where

IC_i = interface coupling

p = the number of interfaces in a component.

$$\text{CBSS coupling} = \sum_{i=1}^{n} CC_i$$

where

CC_i = component coupling

n = the number of components in the system.

This definition consistent with the study by Kitchenham and Likman [31], which indicated that all the information flow metrics studied, except for informational fan-in, appear to act as indicators of future problems.

B. Component Information Flow Metrics

We adopted the definition of information flow proposed by Ince and Shepperd [32] which is considered to be a more sophisticated metric than the original information flow proposed by Henry and Kafura [21]. The aim of this metric is to predict a critical components. A critical component is one that is more likely to contain errors during testing, faults during operation and is more likely to be costly after faults are found [33]. If a critical component is identified early, then a CBSS designer can take appropriate action to reduce the potential problem, such as redesigning critical components or allocating additional test resources.

Fan-in and fan-out are defined with respect to individual interface as follows:

Definition 8: Fan-in of an interface "I" is the sum of inter-flows into an interface "I" plus the number of intra-flows which an interface "I" retrieves.

Definition 9: Fan-out of an interface “I” is the sum of inter-flows from an interface “I” plus the number of intra-flows which an interface “I” updates.

Interface Information Flow (IIF) = $(\text{Fan-in} * \text{Fan-out})^2$

The following is a step by step guide to derive the information flow metrics values for a CBSS:

1. For each interface in a component, calculate the Interface Information Flow (IIF) value of that interface using the formula below:

   Interface Information Flow (IIF) = $(\text{Fan-in} * \text{Fan-out})^2$

2. For each component in a CBSS, sum the Interface Information Flow (IIF) values for all interfaces in that component. We will term this the Component Information Flow (CIF).

   $$\text{Component Information Flow (CIF)} = \sum_{i=1}^{p}(\text{IIF}_i)$$

   where

   p = the number of interfaces in a component

3. Sum the Component Information flow (CIF) values for all components in a CBSS. We will term this the (CBSIF).

   $$\text{CBSS Information Flow (CBSIF)} = \sum_{i=1}^{n}(\text{CIF}_i)$$

   where

   n = the number of components in a CBSS

Kitchenham [31], Shepperd [34] and Lanza [35] have shown that the multidimensional metrics are a more effective approach in understanding, assessing and identifying problem components than any method based on a single metric. Therefore, we grouped the set of metrics to characterize and evaluate different levels of design as follows:

1. Dependency Structures of Interface (DSI)

To characterize and evaluate the dependency behavior of the interfaces we can rank the interfaces according to the Interface Coupling metrics (IC) and Interface Information Flow metrics (IIF) in a scatter plot

2. Dependency Structures of Component (DSC)

To characterize and evaluate the dependency behavior of the components we can rank components according to the Component Coupling metric (CC) and the Component Interface Information Flow metric (CIF) in a scatter plot.

3. Dependency Structures of CBSS (DS-CBSS)

To characterize and evaluate the dependency behavior of the CBSSs we can rank the CBSSs according to the CBSIF and CBSS coupling in a scatter plot.

DSI and DSC represent component level metrics while DS-CBSS represents CBSS level metrics. For CBSS level metrics, CBSS designers should compare different compositions of the same system with respect to testing and maintenance. For component level metrics, CBSS designers should compare different component of the same system with respect to reusability of component.

V. INCORPORATING THE METRICS INTO WEB-BASED CBSS APPLICATION

To study the usefulness of our metrics, we applied them to assess the structure design of Hotel Management System (HMS) which is used in [23] as well as in [36]. Other researchers such as Mahmood and Lai [14] use a similar approach. The choice of HMS was even better since it developed according to [23], which is a good example of Szyperski’s CBSS specification methodology. Figure 5 shows HMS architecture used in the study. The HMS is a web based application that allows a user to search, reserve a hotel room and checks the availability of rooms and prices or cancels his reservation at any time. (Full details of the application can be found at [37]).

In the context of HMS the goals of the application were:

- To explain and demonstrate the capabilities of our proposed metrics and to help software engineering community gain a deep understanding of their definition and application context.
- To investigate whether the metrics results yielded intuitive information to characterize and evaluate the CBSS dependency.

A. Data Collection

Data collection was done by manual inspection of the HMS specification (i.e., components specification, interfaces specification and interaction diagrams). The CIF analysis was performed for each component in the HMS using template defines in Table 1. The following quantitative data was collected:

- The number of inter-component flows.
- The number of intra-component flows.
- The number of components.
- The number of interfaces in each component.
- The number of operations in each interface.

This information was tabulated and analyzed using Excel program. We discarded billing component from the study because we did not find enough information about it is specification. The Data were primarily collected by the first author and checked by the second and third authors independently to help avoiding bias and error. In the event of a disagreement, a negotiation took place. The results were reviewed and discussed in a formal meeting by the authors of this paper.

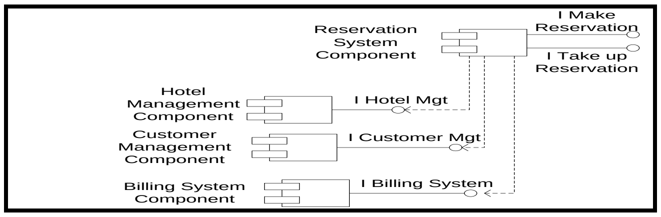

Figure 5. HMS architecture

B. Data analysis

Given the goal of producing components which have a better dependency and with respect to the concept of coupling and information flow complexity, we should interpret the coupling metric and the information flow metric in isolation to verify their functionality, since they reflect the behavior of components based on different concepts, goals and definitions. This claim should, as we understand it, not be interpreted outside the context of metrics hypothesis. Obviously, the coupling metrics reflect the behavior of components in terms of a one directional relationship (i.e., the number of inter-flows out of the component), which in turn assesses the component's impact on the overall system. Whereas, the information flow complexity metrics reflects the behavior of components in terms of bi-directional relationship (i.e., fan-in and fan-out), which assesses the amount of information flowing to and from other components of the system.

The component dependency might be characterized as better, if the component has relatively low values of both coupling metric and information flow metric, which in turn indicates lower CBSS maintenance time and cost.

C. Result and discussion

When changing the reservation system component, we need to consider how both the hotel management component and customer management component will be affected. Whereas, when modifying either the hotel management component or customer management component, we only need to consider how the reservation system component will be influenced. According to the component coupling metric results shown in Figure 6, the coupling of reservation system component is quite high compared with hotel management and customer management components. This means that the reservation system component depends strongly on the customer management component and hotel management components. Usually, high Coupling refers to a more elusive problem [38][39]. Any changes made to a highly coupled component would probably require changes to many other components in the design. Consequently, in the future, understandability, maintainability and reusability of the reservation system component is likely to be quite difficult. The customer management component has the lowest coupling degree which means it's the easiest to modify and reuse.

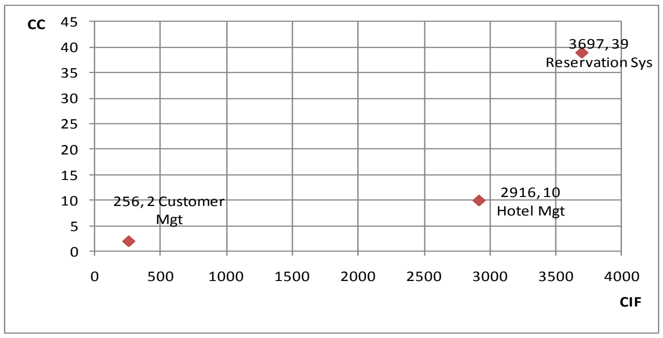

Figure 6. Dependency Structures of Components

In addition, it is interesting to note that the CIF metric values are consistent with the component coupling metric values. Empirical studies in the literature confirm that a high value of information flow measure can reveal three potential problem areas: component which possibly lack functionality, component with stress point (which means a change to it could affect other component in its environment) and/or an inadequate refinement [21].

As shown in Figure 7, in the case of IC metric, the "I make reservation" (IMR) and "I take reservation" (ITR) interfaces indicate highly coupled interfaces. Therefore, it is recommended to investigate IMR and ITR interfaces in terms of the number of other component to which each interface is coupled. The underlying theory of this metrics is that an interface should have a low coupling with other interfaces in a system. The high values of IC metric might mean that the responsibilities of their operations are not clearly defined, which in turn means that the understandability and testability of those interfaces in isolation is very hard, significantly lowering design quality. In contrast, the "I Hotel" and "I Customer" interfaces show lower coupling degree which means they can be easily tested and maintained.

The IIF metric shows interesting results when looking at the total level of information flow. The results show that "I Hotel" interface and IMR interface have relatively high values. The high value of "I Hotel" interface is due to large number of operations exposed by the "I Hotel" interface. This implies that the "I Hotel interface" and IMR interface should be redesigned or investigated by an expert.

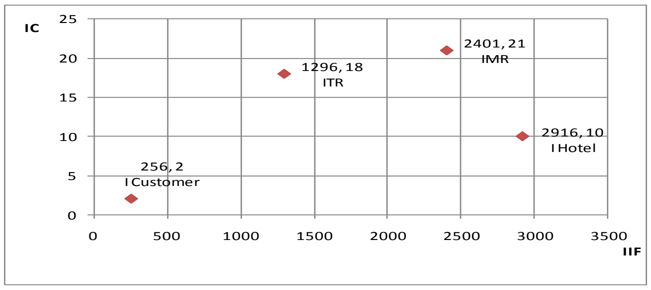

Figure 7. Dependency Structures of Interfaces

VI. CONCLUSION AND DIRECTIONS FOR FUTURE RESEARCH

In this paper, first, we proposed a method named CIF for analyzing information flow in CBSSs. We believe that the CIF is very useful, much easier to collect earlier in the lifecycle, and is a practical basis for evaluating CBSS.

Second, we proposed two sets of metrics which characterize and evaluates the dependency between components, so that CBSS designers can identify critical components in terms of error-proneness and evaluate the impact of the change on the whole CBSS in terms of the difficulty of making a corrective change, which in turn allows designers to target components that need to be revised to improve the quality of the design.

Overall, we believe that our propose metrics can become a very useful tool in help monitoring, managing and controlling test cost estimation, quality estimation and complexity analysis. The component level metrics can be used to identify complex components and/or critical components. Complex and/or critical components assembly would potentially take longer time to develop and test than a simple one. Therefore, developers, tester and maintainers with better experience and more money should be used to integrate and test critical components. For a software tester, complex components require substantial testing effort [2]. The metrics could be used as the basis of a coverage measure of testing for each component (i.e. testers should as a minimum cover all input and output flows). There are also coverage measures that can be based on combinatorial testing of the inputs. Components produced by component providers only include specifications of the interfaces. This imposes difficulties on sufficient testing of an integrated CBSS [40]. For testing such components, we need techniques that do not require the source code and instead relay mainly on the specification of system [20][41]. We believe that the CIF analysis is very useful for this purpose.

The system level metrics might be suitable for effort estimation. In particular, the CBSS metrics should be related to testing costs (since testing requires activating the information flows to confirm the functional and non-function requirements have been met). They might be used to estimate minimal set of test cases that must be run when one component is modified.

This paper represents only the beginning of the research that should be undertaken to explore this approach. So we invite researchers to comment on whether the new approach we proposed captures the real essence of component information flow or if there are areas that are left out.

ACKNOWLEDGMENT

We would like to thank Barbara Kitchenham for her ideas, comments, suggestions and support as we prepare this paper.

REFERENCES

[1] C. Szyperski, Component Software: Beyond Object Oriented Programming,Second Editioned, Addison Wesley, New York, 2002,

[2] L. Narasimhan and B. Hendradjaya, "Some theoretical considerations for a suite of metrics for the integration of software components," Information Sciences, vol.177, 2007, pp. 844-64.

[3] A. De Lucia, A.R. Fasolino and M. Munro, "Understanding function behaviors through program slicing," wpc, 1996, pp. 9.

[4] S. Bates and S. Horwitz, " Incremental program testing using program dependence graphs," Proc. Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, ACM, 1993, pp.384-396

[5] K.B. Gallagher and J.R. Lyle, "Using program slicing in software maintenance," Software Engineering, IEEE Transactions on, vol.17, 1991, pp. 751-61.

[6] G. Gui and P.D. Scott, "Measuring Software Component Reusability by Coupling and Cohesion Metrics," Journal of Computers, vol.4, 2009, pp. 797-805.

[7] G. Gui and P. Scott, "Ranking reusability of software components using coupling metrics," Journal of Systems and Softwar, Journal of Systems and Software, vol.80, 2007, pp. 1450-9.

[8] B. Li, " Managing dependencies in component-based systems based on matrix model," Proc. Proceedings Of Net. Object. Days, Citeseer, 2003, pp.22-25

[9] J. Gorman, "OO Design Principles & Metrics," Online verfügbar unter http://www.parlezuml.com/metrics/OO% 20Design% 20Principles% 20&% 20Metrics.pdf, zuletzt geprüft am, vol.15, 2006, pp. 2009.

[10] N.S. Gill and Balkishan, "Dependency and interaction oriented complexity metrics of component-based systems," SIGSOFT Softw. Eng. Notes, vol.33, 2008, pp. 1-5., http://doi.acm.org/10.1145/1350802.1350810.

[11] M.A.S. Boxall and S. Araban, " Interface Metrics for Reusability Analysis of Components," Proc. Proceedings of the 2004 Australian Software Engineering Conference, IEEE Computer Society, 2004, pp.40

[12] H. Washizaki, H. Yamamoto and Y. Fukazawa, " A Metrics Suite for Measuring Reusability of Software Components," Proc. Proceedings of the 9th International Symposium on Software Metrics, IEEE Computer Society, 2003, pp.211

[13] O.P. Rotaru and M. Dobre, " Reusability metrics for software components," Proc. Proceedings of the ACS/IEEE 2005 International Conference on Computer Systems and Applications, IEEE Computer Society, 2005, pp.24-I

[14] S. Mahmood and R. Lai, "A complexity measure for UML component-based system specification," Software: Practice and Experience, vol.38, 2008, pp. 117-34.

[15] N. Salman, "Complexity Metrics AS Predictors of Maintainability and Integrability of Software components," Journal of Arts and Sciences, 2006,

[16] L. Kharb and R. Singh, "Complexity metrics for component-oriented software systems," SIGSOFT Softw. Eng. Notes, vol.33, 2008, pp. 1-3., http://doi.acm.org/10.1145/1350802.1350811.

[17] A. Sharma, P.S. Grover and R. Kumar, "Dependency analysis for component-based software systems," SIGSOFT Softw. Eng. Notes, vol.34, 2009, pp. 1-6., http://doi.acm.org/10.1145/1543405.1543424.

[18] S.M. Alhazbi, " Measuring the complexity of component-based system architecture," Proc. Information and Communication Technologies: From Theory to Applications, 2004. Proceedings. 2004 International Conference on, 2004, pp.593-594

[19] M.E.R.V.M.S. Dias and D.J. Richardson, " Describing Dependencies in Component Access Points," Proc. Proceedings of the 4th Workshop on Component Based Software Engineering, 23rd International Conference on Software Engineering, 2001,

[20] S.D. Cesare, M. Lycett and R.D. Macredie, Development of Component-based Information System, Prentice Hall of India, New Delhi, 2006,

[21] S. Henry and D. Kafura, "Software Structure Metrics Based on Information Flow," IEEE Trans. Softw. Eng., vol.7, 1981, pp. 510-8., http://dx.doi.org/10.1109/TSE.1981.231113.

[22] M. Abdellatief, A.b.M. Sultan, A.A. Abdul Ghani and M. Jabar, "A mapping Study to Investigate Component-based System Metrics,"

[23] J. Cheesman and J. Daniels, UML Components: A Simple process for Specifying Compoent Based Software, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2001,

[24] I. Crnkovic, B. Hnich, T. Jonsson and Z. Kiziltan, "Specification, implementation, and deployment of components," Commun ACM, vol.45, 2002, pp. 35-40.

[25] M.M. Pickard and B.D. Carter, "A field study of the relationship of information flow and maintainability of COBOL programs," Information and Software Technology, vol.37, 1995, pp. 195-202.

[26] E.B. Allen, T.M. Khoshgoftaar and Y. Chen, " Measuring coupling and cohesion of software modules: an information-theory approach," Proc. metrics, Published by the IEEE Computer Society, 2001, pp.124

[27] F. Xia, "On the concept of coupling, its modeling and measurement," Journal of Systems and Software, vol. 50 pp. 75-84. 2000.

[28] W. Khlif, N. Zaaboub and H. Ben-Abdallah, "Coupling metrics for business process modeling," WSEAS Transactions on Computers, vol.9, 2010, pp. 31-41.

[29] L. Sallie, "Object-oriented metrics that predict maintainability," J.Syst.Software, vol.23, 1993, pp. 111-22.

[30] L.C. Briand, S. Morasca and V.R. Basili, " Measuring and assessing maintainability at the end of high level design," Proc. Software Maintenance, 1993. CSM-93, Proceedings., Conference on, IEEE, 1993, pp.88-87

[31] B.A. Kitchenham and S.J. Linkman, "Design metrics in practice," Information and Software Technology, vol.32, 1990, pp. 304-10.

[32] D. Ince C. and M. Shepperd J., " An empirical and theoretical analysis of infromation flow-based system design metrics," Proc. 2nd European Software Engineering Conf, Springer Verlag, 1989,

[33] K. El-Emam, "A methodology for validating software product metrics," 2010,

[34] M. Shepperd, "Measurement of structure and size of software designs," Information and Software Technology, vol.34, 1992, pp. 756-62.

[35] M. Lanza and R. Marinescu, Object-Oriented Metrics in Practics: Using softqware Metrics to Characterize, Evaluate, and improve the Design of Object-Oriented Systems, Springer, Berlin Heidelberg - Germany, 2006,

[36] M. Heisel and J. SouquiÃ¨res, "Adding Features to Component-Based Systems," Objects, Agents, and Features, vol. 2975 pp. 25-36. 2004.

[37] "http:www.umlcomponents.com," August/8/ 2011.

[38] L. Briand, S. Morasca and V.R. Basili, "Defining and validating high-level design metrics," pp. 31. 1994.

[39] S.R. Chidamber and C.F. Kemerer, "A Metrics Suite for Object Oriented Design," IEEE Trans. Softw. Eng., vol.20, 1994, pp. 476-93., http://dx.doi.org/10.1109/32.295895.

[40] Y. Wu, M.H. Chen and J. Offutt, "UML-based integration testing for component-based software," COTS-Based Software Systems, 2003, pp. 251-60.

[41] E.J. Weyuker, "Testing component-based software: A cautionary tale," Software, IEEE, vol.15, 1998, pp. 54-9.