KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES
Proceedings of the International Conference on Knowledge Engineering,
Principles and Techniques, KEPT2007
Cluj-Napoca (Romania), June 6–8, 2007, pp. 324–331

METRICS-BASED SELECTION OF A COMPONENT ASSEMBLY

CAMELIA SERBAN$^{(1)}$ AND ANDREEA VESCAN$^{(1)}$

Abstract. The work of integrating the components with each other and with the rest of the system is the most important part of the component-based development process. The interaction among components in an assembly is essential to the overall quality of the system. When integrating components into a system assembly, it would be useful to predict how the quality attributes for the whole system will be. In order to predict and to asses quality attributes, the usage of software metrics is a necessity.

Software metrics that follow the assembly-centric evaluation approach are used to select (from all obtained assemblies) the solution that best represents the system requirements.

1. Introduction

The goal of software composition is to find a good combination of components that leads to a software system that responds to client-specific requirements. Software composition promises components reuse and therefore productivity gains, because of shorter time-to-market and improved quality. Two steps need to be done when a system is to be constructed from a set of components: building all possible configurations and then analyzing each of these assemblies in order to obtain the final solution.

The long-term success of Component-Based Development (CBD) depends on the ability to predict the quality of the obtained systems. For this reason researchers and practitioners are keen on developing techniques for efficient component selection and composition [3]. Issues related to developing a composition theory include determining how to predict the properties of assemblies, how to measure properties of components, how to verify the measurements, and how to communicate the property values to component users.

The realm of software metrics includes proposals for both product and process assessment. In this paper, we are concerned with product metrics, with a focus on metrics for component selection and composition.

2000 *Mathematics Subject Classification.* 68N30, 68N19.
Key words and phrases. software metrics, component assembly, assemblies quality attributes prediction.

Problem statement. A set of specified components and the system requirements that we want to develop are given. In our previous work we had developed an algorithm [10] that obtains all the possible and correct system configurations from the given components. In order to decide which solution to choose (that best represents the system requirements) we use metrics to assess quality attributes that are of interest for assembly evaluation.

The remainder of this paper is organized as follow. After this introduction, section 2 presents our previous and current view on component and component composition. Next, section 3 presents a collection of assembly metrics that are relevant for measuring the quality attributes which we are interested in. An example and result analysis are given in Section 4. The paper finishes by drawing some conclusions and outlining further research activities.

2. Component Specification Elements and Composition

There are many similar but not identical definitions of components, although the basic idea seems to be the same. All definitions highlight the basic characteristics of a component: it is an independent software module; provides a functionality, but is not a complete system; can be accessed only through its interface and can be incorporated in a software system without regard to how it is implemented.

2.1. **Component Specification Elements.**

In this section our previous approach on component specification elements are presented. There are two kind of components: simple and compound [4]. A simple component can have many input data and many output data that represent the parameters of the functionality (only one) being implemented. A compound component is a group of connected components, in which the output of a component is used as input by another component from this group. Two particular simple components are the source [1] and the destination [2] component.

Another previous approach of component specification involved the following characteristics: component *id* and *interface*. The interface of the component should describe all the exported features of the component. The services provided by the component are seen as functions, so the interface specifies a list of function signatures. In this paper we separate the interface of a component in provided interface and required interface.

2.2. **Components Composition Reasoning.**

The goal of software composition is to find a good combination of components that leads to a software system that responds to client-specific requirements [1]. The components assembly process consists of building a set of all possible configurations with the given candidate

[1]A source component, i.e. a component without inports, is a component that generates data provided as outports in order to be processed by other components.

[2]A destination component, i.e. a component without outports, is a component that receives data from the system as its inports and usually displays it, but it does not produce any output.

components. A configuration is constructed by adding based on the data dependencies (provided and required services) a candidate component [10].

3. Some relevant suite of metrics

In order to assess, in an quantitative manner, some quality attributes that are considered important for our system, we need to define a set of metrics that measure these attributes. Before we define metrics, we need to know the type of information that is available about the entities we plan to asses (the software components and the assembly). The fact that components are black box and binary units of composition, whose internals cannot be viewed or accessed, only leaves us with externally observable elements of the component that allow to assess its quality. The quality attributes that we decided to measure are: reusability, functionality, understandability, maintainability and testability.

In this section we present a collection of software component metrics that are relevant in measuring the quality attributes stated above. All these metrics are context-dependent: a given component will have different metrics values depending on the particular architectural configuration in which it is placed.

3.1. **Metrics Overview in CBD.** In this section we present metrics already introduced by other researchers. In [6] the metrics described above were formalized in OCL and a comparison was made.

In [7] Hoek et al. proposed metrics to asses service utilization in component assemblies. The metrics follow the assembly-centric evaluation approach.

Definition 1. ***Component service utilization metrics.*** [7] *The Provided Services Utilization (PSU) represents the ratio of services provided by the component which are actually used (Equation 1 - left side). The Required Services Utilization (RSU) is similar, but for required services (Equation 1 - right side).*

$$PSU_X = \frac{P_{actual}}{P_{total}} \qquad RSU_X = \frac{R_{actual}}{R_{total}} \tag{1}$$

where: P_{actual} = number of services provided by component X that are actually used by other components and P_{total} = number of services provided by component X; R_{actual} = number of services required by component X that are actually provided by the assembly and R_{total} = number of services required by component X.

Definition 2. ***Compound Service Utilization metrics.*** [7] *The Compound Provided Service Utilization (CPSU) represents the ratio of services provided by the components in the assembly which are actually used (Equation 2 -left side). The Compound Required Service Utilization (CRSU) is similar, but for services required by the components (Equation 2 - right side).*

$$CPSU_X = \frac{\sum_{i=1}^{n} P^i_{actual}}{\sum_{i=1}^{n} P^i_{total}} \qquad CRSU_X = \frac{\sum_{i=1}^{n} R^i_{actual}}{\sum_{i=1}^{n} R^i_{total}} \tag{2}$$

where P^i_{actual}= number of services provided by component i that are actually used by other components and P^i_{total} = number of services provided by component i; R^i_{actual} = number of services required by component i that are actually provided by the assembly and R^i_{total}= number of services required by component i.

In [8] Narasimhan and Hendradjaya proposed metrics to asses component interaction density (a measure of the complexity of relationships with other components).

Definition 3. ***Interaction density of a component.*** [8] *The Interaction Density of a Component (IDC) is defined as a ratio of actual interactions and potential ones (Equation 3). The Incoming and Outgoing Interaction Density of a Component (IIDC and OIDC, respectively) are similar, but considering only incoming interactions (Equation 4 - left side) or outgoing ones (Equation 4 - right side).*

$$IDC = \frac{\#I}{\#I_{max}} \tag{3}$$

where $\#I$ = Actual Interactions and $\#I_{max}$ = Maximum available interactions.

$$IIDC = \frac{\#I_{IN}}{\#I_{maxIN}} \qquad OIDC = \frac{\#I_{OUT}}{\#I_{maxOUT}} \tag{4}$$

where $\#I_{IN}$ = Actual incoming interactions and $\#I_{maxIN}$ = Maximum available incoming interactions and $\#I_{OUT}$ = Actual outgoing interactions and $\#I_{maxOUT}$ = Maximum available outgoing interactions.

Definition 4. ***Average Interaction Density of Software Components.*** [8] *The Average Interaction Density of Software Components (AIDC) represents the sum of IDC for each component divided by the number of components.*

$$AIDC = \frac{IDC_1 + IDC_2 + IDC_n}{\#components} \tag{5}$$

where IDC_i = IDC of component i and $\#components$ = number of components in the system.

In our previous work [9] a component assembly was view as a graph (transformed in a dependences tree). This approach enabled us to define new metrics for depth and breadth components hierarchy (measuring dependences calls between components).

3.2. **Proposed Metrics.** We propose the following two metrics for measuring coupling between components.

Definition 5. ***Component Coupling Grade.*** *The Component Coupling Grade (CCG) of a component X witch is dependent by a component Y, represents the number of services provided by Y that X uses. In what follows we will denote this value with CCG(X,Y).*

Definition 6. ***Component Coupling Total Grade.*** *The Component Coupling Total Grade (CCTG) of a component X which is dependent by a set of components* C_1,C_2,...,C_n, *represents the number of services provided by all these components that X uses.*

$$CCTG = CCG(X, C_1) + CCG(X, C_2) + ... + CCG(X, C_n). \tag{6}$$

3.3. **The influence of metrics values on quality attributes.** We stated before that our aim is to define metrics that are relevant in measuring the quality attributes which we are interested in. We need these informations for choosing the solution that best represents the system requirements. Table 1 presents the influence of metrics values on the quality attributes witch we consider important for the assembly evaluation. We use the following notations: m for metric low value, M for hight value of the metric, $+$ for positive influence and $-$ for negative influence. For example a low value of IDC influences positively the reusability of the component.

Table 1. The influence of metrics values on quality attributes

	Reusability	Functionality	Understandability	Maintainability	Testability
PSU	m/+	m/-	m/+	m/+	m/+
RSU	m/+	-	m/+	m/+	m/+
CPSU	m/+	m/-	m/+	m/+	m/+
CRSU	m/+	-	m/+	m/+	m/+
IDC	m/+	m/-	m/+	m/+	m/+
IIDC	m/+	-	m/+	m/+	m/+
OIDC	m/+	m/-	m/+	m/+	m/+
AIDC	m/+	-	m/+	m/+	m/+
CCG	M/-	M/+	M/-	M/-	M/-
CCTG	M/-	M/+	M/-	M/-	M/-

A threshold is a limit (high or low) placed on a specific metric. All the above metric values scale between 1 and 0, except the CCTG and CCG. We set the value of the threshold at 0.5. In our future work we will apply precise methods in choosing the threshold value.

4. Example and Result Analysis

In this section we present an example to illustrate the above metrics and our approach for the best solution selection based on metrics.

The system designer, during the requirements analysis phase, grouped the input and output data of the system into three required interfaces and two provided interfaces. The first step of how configurations can be built consists of selecting from a repository the set of components that may potentially participate in the final system. In this example, nine components have been found as candidates. We add two more components to complete the final system: a *Read* (R) and a $Write$ (W) component. The algorithm [5, 10] provided several solutions. For the purpose of this paper we only present two of them and discuss the different metrics values for each system-solution and their influences on the quality attributes.

The first solution is represented in figure 1 - right side. From the set of selected candidate-components this solution contains only six of them (without taking into consideration the R and W components).

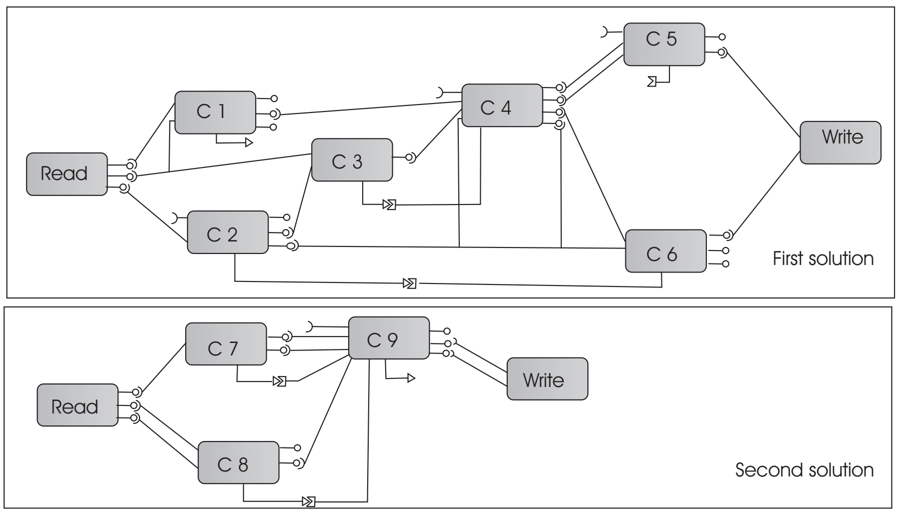

Figure 1. System first and second solutions

The values of the metrics for each component in the final system are presented in table 2 and the assembly value metrics in table 3. The tables show that the solution described above has the values for the metrics around the medium value, for all quality attributes. For example, the majority of the components have a very high functionality in the system (PSU, CPSU, OIDC and IDC values are very close to 1) and at the same time they can offer new functionalities for the future improvement by adding new provided services (influences the maintainability attribute).

	PSU	RSU	IDC	IIDC	OIDC	CCTG
C_1	0.33	1	0.5	1	0.25	2
C_2	0.66	0.50	0.66	0.5	0.75	1
C_3	1	1	1	1	1	2
C_4	1	0.75	0.88	0.80	1	3
C_5	0.50	0.66	0.50	0.50	0.50	2
C_6	0.33	1	0.71	1	0.33	3
C_R	1	–	1	–	1	–
C_W	–	1	1	1	–	2

TABLE 2. System first solution metrics values

CPSU	CRSU	AIDC
0.68	0.82	0.77

TABLE 3. Assembly metrics values

Regarding the coupling metrics we can remark that there is a maximum limit that is not very high and we can say that the maintainability and reusability are not strongly influenced. The assembly values metrics suggest that the solution is not considered to be the "best" for every quality attribute, but a medium "best" solution for the overall system. The value of the $AIDC$ metric is close to 1 but we must take also into consideration the $CCTG$ metric to decide which solution best represents our future needs (if we would like to improve and add new functionality or if we just want to have a good functionality for the system).

The second solution chosen to be represented is depicted in figure 1 - left side. The solution contains only three internal components form the set of candidate-components. The values of the metrics for each component in the final system are presented in table 4 and the assembly value metrics in table 5

	PSU	RSU	IDC	IIDC	OIDC	CCTG
C_7	1	1	1	1	1	1
C_8	0.50	1	0.80	1	0.66	2
C_9	0.66	0.75	0.70	0.83	0.50	3
C_R	1	–	1	–	1	–
C_W	–	1	1	1	–	2

TABLE 4. System second solution metrics values

CPSU	CRSU	AIDC
0.80	0.88	0.90

TABLE 5. Assembly metrics values

The metrics values that influence the functionality attribute are close to 1 revealing a good functionality of each component inside the system, but the other metrics values influence negatively the other quality attributes. The 0.50 chosen threshold is exceeded for all the computed metrics. In table 1 we can see that a high value influences negatively almost all the quality attributes discussed. The values of $CCTG$ metric are relatively high considering that there are few components in the solution. The $CCTG$ value for the ninth component is considered to be high yielding a very hard understandability, testability and maintainability.

5. Conclusions and Future work

Software metrics provide a quantitative means to control the quality of software. After building all possible configurations (component assemblies) from a given set of specified components, the designer has to decide which solution to use further. We discussed and proposed in this paper some quality attributes to consider when analyze the quality of an assembly. Software metrics are used to select the solution, among all obtained configurations, that best represents the system requirements.

We set the value of the metrics threshold at 0.5. In our future work we will apply precise methods in choosing the threshold value. There are different methods like statistic-based and even genetic-based algorithms.

References

[1] Crnkovic, I., *Component-based software engineering - new challenges in software development*, Software Focus, John Wiley & Sons, 2001

[2] Crnkovic, I., Larsson, M., *Building Reliable Component-Based Software Systems*, Artech House publisher, 2002

[3] Crnkovic, I., Schmidt, H., Stafford, J. A., Wallnau, K., *The* 6^{th} *ICSE Workshop on Component-Based Software Engineering: Automated Reasoning and Prediction*, ACM SIGSOFT Software Engineering Notes, vol 29, nr 3, pp.1-7, 2004

[4] Fanea, A., Motogna, S., *A Formal Model For Component Composition*, Proceedings of the Symposium Cluj-Napoca Academic Days, pp. 160 - 167, 2004

[5] Fanea, A., Motogna, S., Dioşan, L., *Automata-Based Component Composition Analysis*, Studia Universitas "Babes-Bolyai", Seria Informatica, vol. L (1), pp. 13 - 20, 2006

[6] Goulo,M. A., Abreu, F. B., *Composition Assessment Metrics for CBSE*, 31st Euromicro Conference, Component-Based Software Engineering Track, Porto, Portugal, IEEE Computer Society, pp. 96 - 103, 2005

[7] Hoek, A. v. d., Dincel, E., and Medvidovic, N., *Using Service Utilization Metrics to Assess and Improve Product Line Architectures*, 9th IEEE International Software Metrics Symposium (Metrics'2003), Sydney, Australia, 2003

[8] Narasimhan, V. L. and Hendradjaya, B., *A New Suite of Metrics for the Integration of Software Components*, The First International Workshop on Object Systems and Software Architectures (WOSSA'2004), Australia, 2004

[9] Serban, C., Vescan, A., *Metrics for Component-Based System Development*, Creative Mathematics and Informatics, Vol. 16,pp. 143-150, 2007

[10] Vescan, A., Motogna, S., *Syntactic automata-based component composition*, The 32nd EUROMICRO Software Engineering and Advanced Applications (SEAA), Work in Progress, 2006

(1) Computer Science Department, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania

E-mail address: {camelia, avescan}@cs.ubbcluj.ro