CREATIVE MATH. & INF.
**16** (2007), 143 - 150

*Dedicated to Professor Ioan A. RUS on the occasion of his $70^{th}$ anniversary*

# Metrics for component-based system development

CAMELIA ŞERBAN and ANDREEA VESCAN

ABSTRACT. Component-based development (CBD) advocates the acquisition, adaptation, and integration of reusable software components to rapidly develop and deploy complex software systems with minimum engineering effort and resource cost. 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.

Useful insight on the specificities to consider when developing metrics for CBD are presented, both concerning individual components (assessing components in isolation) and component assemblies (assembly-centric evaluation approach) are presented. Concerning the component-assembly approach we adapt metrics for object-oriented design (CBC - Coupling Between Components) and new metrics are defined (DDT - Depth Dependence Tree and BDT - Breadth Dependence Tree).

## 1. INTRODUCTION

Since the late 90's Component Based Development (CBD) is a very active area of research and development. Component Based Software Engineering (CBSE) is the emerging discipline of the development of software components and the development of systems incorporating such components. Its goals, among others, are to consistently increase return on investment and time to market, while assuring higher quality and reliability than can be achieved through current software development [4].

Development using components is focused on the identification of reusable entities and relations between them [3]. 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.

Metrics have become essential in some disciplines of software engineering. In forward engineering they are being used to measure software quality and to estimate cost and effort of software projects [1]. In the field of software evolution, metrics can be used for identifying stable or unstable parts of software systems, as well as for identifying where refactorings can be applied or have been applied [5], and for detecting increases or decreases of quality in the structure of evolving software systems.

Received: 15.09.2006. In revised form: 19.02.2007
2000 *Mathematics Subject Classification.* 68Q01, 68Q15.
Key words and phrases. *Software metrics, component-based development, assembly-centric evaluation.*

In the area of software reengineering and reverse engineering, metrics are being used for assessing the quality and complexity of software systems, as well as getting a basic understanding and providing clues about sensitive parts of software systems.

A short overview of component assembly metrics, including the proposals of Narasimhan and Hendradjaya [10] and Hoek et al. [7] is presented in Section 2.

Our view of component assemblies as a graph is presented in Section 3. Starting from the assembly graph we construct the dependences tree. Existing metrics for object-oriented design [8, 11] are adapted (see Section 3.3) for component assemblies - *Coupling Between Components* (CBC).

Metrics for the component-based system hierarchy are proposed, based on the same new approach of component interaction. The new metrics *Depth Dependence Tree* - DDT and *Breadth Dependence Tree* - BDT are presented in Section 3.4. These metrics help us to asses quality attributes of the system.

An example that illustrate the importance of the proposed metrics through the evaluation of a PDA (Personal Digital Assistant) software is described in Section 4.

## 2. Metrics overview in CBD

Some proposals aim at establishing requisites and guidelines for CBD metrics, both concerning individual components and component assemblies.

Several authors made proposals for the evaluation of component interfaces and dependencies [1], [12]. The metrics follow a component-centric view of component quality evaluation, assessing components in isolation.

Other metrics proposals follow the assembly-centric evaluation approach:

- Narasimhan and Hendradjaya [10] proposed metrics to assess component integration density (a measure of the complexity of relationships with other components);
- Hoek et al. [7] proposed metrics to assess service utilization in component assemblies.

## 3. Our approach

This section presents our view of component assemblies as a graph. Starting from the graph assembly we construct the dependence tree. Based on this approach we adapt some existing metrics from object-oriented design and define new metrics for depth and breadth components hierarchy.

### 3.1. Formal Approach of Assembly.

**Definition 3.1.** An **assembly** is a binary relation denoted by $DR = (C, D), D \subseteq C \times C$, where $C$ is a set of components and $D$ is the relation graphic that contains the dependences between components. There is a component $c_0 \in C$ with a special role, to start the system execution.

**Definition 3.2.** A **dependence** is a pair $d = (c_1, c_2) \in D$ with the meaning that the execution of $c_1$ needs some services provided from $c_2$ (in other words, $c_1$ depends of $c_2$).

We model a component-based system (an assembly of components) as a directed graph ($DR$) in which the vertices are the components (the set $C$) and the edges are the dependences (the set $D$) between components. In this way we mapped each assembly to a directed graph. In the following we use the latter view.

**Remark 3.1.** Additional conditions must be satisfied by an assembly:

(1) $\forall c \in C \Rightarrow (c,c) \notin D$;
(2) $(\forall c_1, c_2, ...c_i, c_{i+1}, ...c_n \in C, i = \overline{1, n-1},\ n \geq 3 : (c_i, c_{i+1}) \in D) \Rightarrow c_1 \neq c_n$.

In figure 1 the left assembly satisfies the above conditions, while the right one does not:

- $\exists c_3 \in C\ such\ that\ (c_3, c_3) \in D$ (condition (1) is violated);
- $\exists c_1, c_2 \in C\ such\ that\ (c_1, c_2), (c_2, c_1) \in D$ (condition (2) is violated).

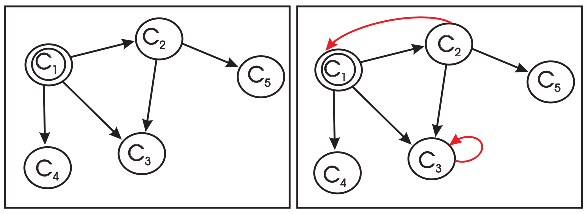

FIGURE 1. Additional assembly conditions

3.2. **Assembly Dependences Tree.** Using directed graph view of the assembly is difficult to provide the depth and breath of the dependences between involved components. A better view implies the transformation of the directed graph into a tree. The dependences tree construction is described in detail in $DTA$ and $CDTA$ algorithms.

**Algorithm 1** Dependences Tree Algorithm (DTA)

```
1: $CD = D; l = 1$;
2: Identify the root - start node from the graph;
3: repeat
4:   for each component $c$ from level $l$ do
5:     for each $d \in C : (c,d) \in D$ do
6:       if $((c,d) \in CD)$ then
7:         Add $d$ in level $(l+1)$: $c$ father of $d$;
8:         $CD = CD - \{(c,d)\}$;
9:       end if
10:    end for
11:  end for
12:  $l = l + 1$;
13: until $CD = null$
```

A short example for the Algorithm 1 can be visualized in Figure 2. The root (level one) is the second component. In the next level are added three components (third, forth and fifth) because the relations $(2,3),(2,4)$ and $(2,5)$ are in $CD$ set. In the next step of the algorithm the components for the third level will be discovered: all the components from the second level are in relations with other components in the assembly. The third component is dependent of the forth and the firth component and as a consequence in the forth level (from the third component) there are draw two links to another forth and fifth component. We have denoted with an index if more than one instance of a component appears.

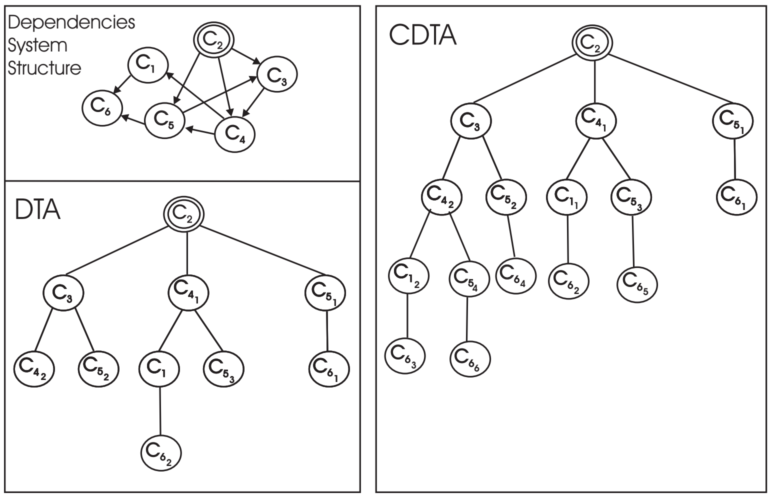

FIGURE 2. Construction of the dependences trees using DTA and CDTA algorithms

To obtained an optimal tree that contains all the paths from the directed graph representation, an additional algorithm that completes the tree is required. See Algorithm 2. For example, the $(2,3,4,1,6)$ chain is not included in the tree structure (see Figure 2 - $DTA$).

**Algorithm 2** Complete Dependences Tree Algorithm (CDTA)

```
1: for each leaf $n$ of the tree do
2:     Determine the node $m$ equal to $n$ that is situated on the lowest level.
3:     Attach to node $n$ the sub-tree of the $m$ node, if exist.
4: end for
5: for each leaf $n$ of the tree, previous added do
6:     Determine the node $m$ equal to $n$ that is situated on the lowest level.
7:     Attach to node $n$ the sub-tree of the $m$ node, if exist.
8: end for
```

Figure 2 - $CDTA$ contains the new tree obtained after applying the $CDTA$ algorithm.

3.3. **Adapted Metric.** In an object-oriented design, coupling is ”the interconnectedness between its pieces“ [2]. The declaration of an object of a remote class creates a potential collaboration between the two classes. This is measured by metric CBO [11]. If two classes are collaborators, then a value of one is added the CBO irrespective of how many messages flow between the two collaborators. We adapt this metric for a component-based system.

We consider an assembly of components, $DR = (C, D)$, where $C$ is a set of components and $D$ is the relation graphic that contains the dependences between components.

**Definition 3.3.** A component $c_1$ **is coupled** with component $c_2$ if $(c_1, c_2) \in D$.

**Definition 3.4. Coupling Between Components (CBC)** metric measures the number of components with witch a given component is coupled.

We are interested in coupling from the perspective of quality evaluation because an excessive coupling plays a negative role on many external quality attributes:

- The *reusability* of components is low when the coupling between these is high, because an entity is strong dependent on the context where it is used. Normally a module (subsystem) should have a low coupling with the rest of the modules.
- A high coupling between the different parts (modules) of a system has a negative impact on the *modularity* (responsibilities of each part are not clearly defined).
- The *understandability* and *testability* are also affected by high coupling between components.

3.4. **New Defined Metrics.** We consider an assembly of components, $DR = (C, D)$, where $C$ is a set of components and $D$ is the relation graphic that contains the dependences between components, and the assembly dependences tree described in Section 3.2.

In the following we will define two metrics related with this approach.

**Definition 3.5. Depth Dependence Tree (DDT) metric.** Let us consider a component $c_n \in C$ and the corresponding elementary chain $c_0, c_1, ..., c_n$, where $c_0$ is the root node. The metric value is $DDT = n$. In other words, DDT measures the length chain dependences from a given component to the root.

**Definition 3.6. Breadth Dependence Tree (BDT)** metric represents the number of chains dependences from the root to all the leafs.

We evaluate these metrics taking into account the impact on quality attributes. From this point of view, a high value of metric DDT makes the component hard to *reuse* in a different context. In addition, *understandability*, *maintainability* and *testability* are also affected. The understanding of an entity requires a recursive understanding of all the components that it depends. Moreover any change in a component requires changes in all components that depend of this. A high value of the metric affects *maintainability* and *understandability*. The system tends to become increasingly complex.

## 4. Example: PDA - Personal Digital Assistant

In order to highlight the importance of our approach (regarding computation and interpretation of the metrics values) we proposed the evaluation of PDA (Personal Digital Assistant) software. Possible improvements are made and a comparative study between the first system and the one obtained after the improvements is presented.

The initial system from Figure 3 has six main components. The *Person* component is used to record data about a person (name, address, phone, birthday, etc.). The *Activity* component is used to record data about an activity (subject, description, list of persons involved in the activity).

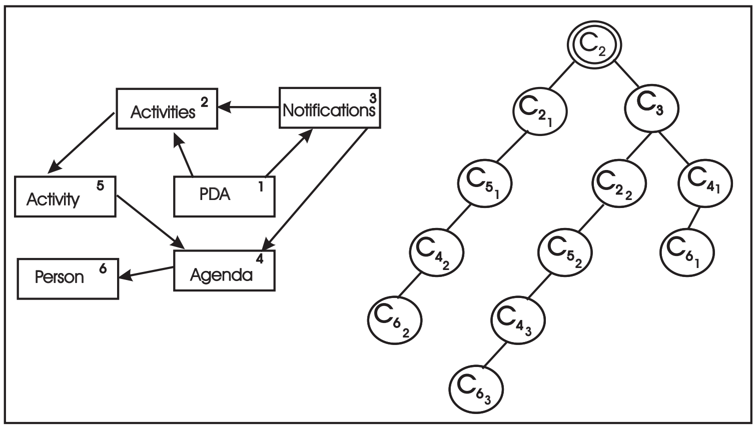

Figure 3. Personal Digital Assistant and the associated dependences tree

The *Agenda* component contains a list of activities for each person, and the *Activities* component manages the list of all activities and has operations concerning activities. All the notifications about birthdays that are coming, deadline of an activity and other important information are provided by the *Notification* component. The *PDA* component is the start component of the system.

The CBC and DDT metrics values, computed for the initial PDA system are presented in Table 1 and Table 2. The value of the BDT metric is 3.

Table 1. Values for the CBC metric

| $Component$ | $C_1$ | $C_2$ | $C_3$ | $C_4$ | $C_5$ | $C_6$ |
|---|---|---|---|---|---|---|
| CBC metric value | 2 | 1 | 2 | 1 | 1 | 0 |

**Measurements interpretation.** After analyzing the components hierarchy obtained by applying the assembly dependences tree construction algorithm and computing the metrics values, we have concluded that the components hierarchy is slightly branched (three branches) and very deep (six levels). This implies a high coupling at the system level as we can see from the DTT metric values.

TABLE 2. Values for the DDT metric

| *Component* | $C_1$ | $C_{2_1}$ | $C_{2_2}$ | $C_3$ | $C_{4_1}$ | $C_{4_2}$ | $C_{4_3}$ | $C_{5_1}$ | $C_{5_2}$ | $C_{6_1}$ | $C_{6_2}$ | $C_{6_3}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DDT metric value | 0 | 1 | 2 | 1 | 2 | 3 | 4 | 2 | 3 | 3 | 4 | 5 |

Taking into account the above conclusions, the system has been redesigned where the *Activities* component encapsulates the *Activity* component and the *Agenda* component contains the *Person* component (see Figure 4).

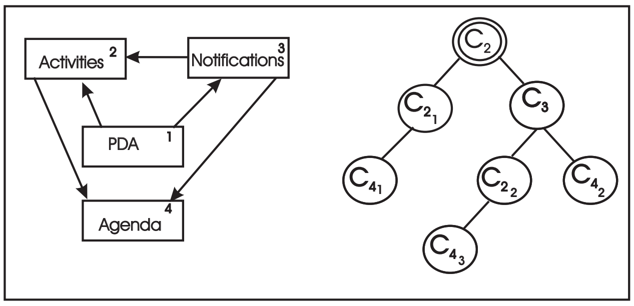

FIGURE 4. Personal Digital Assistant and the associated dependences tree for the second system

TABLE 3. Values for the CBC metric for the redesigned system

| *Component* | $C_1$ | $C_2$ | $C_3$ | $C_4$ |
|---|---|---|---|---|
| CBC metric value | 2 | 1 | 2 | 0 |

TABLE 4. Values for the DDT metric for the redesigned system

| *Component* | $C_1$ | $C_{2_1}$ | $C_{2_2}$ | $C_3$ | $C_{4_1}$ | $C_{4_2}$ | $C_{4_3}$ |
|---|---|---|---|---|---|---|---|
| DDT metric value | 0 | 1 | 2 | 1 | 2 | 2 | 3 |

The CBC and DDT metrics values, computed for the redesigned PDA system are presented in Table 3 and Table 4. The value of the BDT metric is 3.

**Conclusions.** By decreasing the degree of coupling between components, in the redesigned system we have obtained an increased reusability - the components *Activities* and *Agenda* can be reusable in a different context. A better understandability and testability are also obtained - it is much easier to follow the logic of the system when the coupling between its constituents components is lower.

## 5. Future work

Software metrics provide a quantitative means to control the quality of software. A formal approach for defining metrics that quantifies quality attributes of CBD was proposed. Our approach of component assemblies as a graph (transformed in dependences tree) enabled us to define new metrics. In this context, the main goal for defining metrics is to quantify different aspects that lead to system quality improvements, by bridging the gap between qualitative and quantitative statements.

Starting from this approach, we will focus our future work on developing new methods and techniques to provide a mechanism for interpreting measurement results and identifying those parts of the system that need improvements.

## References

[1] Boxall M. A. S. and Araban S., *Interface Metrics for Reusability Analysis of Components*, Australian Software Engineering Conference, Melbourne, Australia, 2004

[2] Coad P. and Yourdon E., *Object-Oriented Design*, Prentice Hall, London, $2^{nd}$ edition, 1991

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

[4] Crnkovic I., *Component-based Software Engineering - New Challenges in Software Development*, Software Focus, Ed. John Wiley & Sons, 2001

[5] Demeyer S., Ducasse S. and Nierstrasz O., *Finding refactorings via change metrics*, In Proceedings of OOPSLA, ACM SIGPLAN Notices, pp. 166-178, 2000

[6] Fenton N. and Pfleeger L.S., *Software Metrics: A Rigorous and Practical Approach*, International Thomson Computer Press, London, UK, $2^{nd}$ edition, 1996

[7] Hoek A. v. D., Dincel E. and Medvidovic N., *Using Service Utilization Metrics to Assess and Improve Product Line Architectures*, $9^{th}$ IEEE International Software Metrics Symposium, Sydney, Australia, 2003

[8] Lorenz M. and Kidd J., *Object-Oriented Software Metrics*, Prentice-Hall Object-Oriented Series, Englewood Cliffs, NY, 1994

[9] McCabe T. J., Watson A. H., *Software Complexity*, Crosstalk, Journal of Defense Software Engineering, pp. 5 - 9, 1994

[10] 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, Australia, 2004

[11] Chidamber S. R. and Kemerer C. F., *A Metrics suite for Object Oriented design*, IEEE Transactions On Software Engineering, Vol. 20, No. 6, pp. 476 - 493, 1994

[12] Washizaki H., Yamamoto H. and Fukazawa Y., *A Metrics Suite for Measuring Reusability of Software Components*, $9^{th}$ IEEE International Software Metrics Symposium, Sydney, Australia, 2003

Babeş-Bolyai University
Computer Science Department
M. Kogălniceanu 1
400084 Cluj-Napoca, Romania
*E-mail address*: camelia@ubbcluj.ro
*E-mail address*: avescan@cs.ubbcluj.ro