Predicting Maintainability using Object Oriented System Decomposition Metrics

Nejmeddine Tagoug
Computer Science and Information Systems College
King Saud University
Riyadh Kingdom of Saudi Arabia

ABSTRACT: *The maintenance of existing software can account for 60 percent of all effort expended by the resources used in the system development life cycle. To improve the situation, analysts must design systems that are easy to use and maintain at the early stage of the development process. At the design phase, an object-oriented system (o-o system) is decomposed into subjects. Each subject is decomposed into classes. A maintainable o-o system is a system where the change affects a less number of subjects. In this work, maintainability metric is proposed. This metric measures the localization of a change in a subject at the early stages of the development process during the analysis and design phases. This metric seems to be a good indicator of the quality of a system at the early stages.*

Keywords: Object Oriented System Decomposition, Maintainability Metrics, Software Engineering

Received: 16 October 2013, Revised 24 November 2013, Accepted 30 November 2013

1. Introduction

Object-oriented system development life cycle comprises four main phases: Analysis, design, implementation and maintenance. During the design phase, an o-o system is decomposed into subjects, each subject into classes. During maintenance phase, a set of system changes are made after the system is operational. Many large companies use o-o systems as their main systems. They currently expend from 60 percent of all programming effort on maintenance [7]. Growing maintenance costs have became a major concern for developers and users of o-o systems. To improve the situation, analysts must design o-o systems that are easy to maintain. A maintainable o-o system is a system where the impact of change affects a less number of subjects. Maintainability is defined by the ISO 9126, as the effort to implement a set of changes. A consensus has emerged that the maintainability of a software system is dependent on its design [10]. At the design phase, maintainability metric is needed to assess the impact of changes and to prevent maintainers to expend a high cost at the maintenance phase.

This paper is organized as follows: Section 2 presents the related work and a very brief summary of the need for research in this area. Section 3 describes the object-oriented information system model. Section 4 defines the maintainability metrics at the system design level. An experiment of the metrics on a real information system is presented and discussed in Section 5. Conclusion and future work are presented in section 6.

2. Related Work

Although there have been a number of papers addressing object-oriented software systems maintenance problems. Few of

them presented a concrete and systematic metrics to deal with the problems at the design phase. In this section, we briefly review the existing work, and then relate this work to them.

Rajaraman and Lyu (1992) [9] discussed some difficulties that one encounters in the testing and maintenance of C++ programs. They show by arguments and by some empirical evidence that widely used complexity metrics like lines of code, cyclomatic complexity, may not be appropriate to measure the complexity of C++ programs.

Basili et al. (1996) [1] were interested in a specific part of maintenance, i.e. fault detection and fault-proneness. Experiments on eight systems developed by students showed that Chidamber and Kemerer's OO metrics appear to be useful to predict class fault-proneness during the early phases of the life-cycle.

Kiran et al. (1997) [5] considered only inheritance, aggregation and association but not invocation and friendship. Their change model is incomplete.

Chauman et al. (1999) [2] define a change impact model and apply it on an industrial C++ software system to assess its changeability. However, high level metrics at the design level and their impacts on maintenance are not considered in the method.

Rajendra et al. validate three metrics: interaction level, interface size and operation argument complexity. The three metrics interaction level, interface size and operation argument complexity. The three metrics appear to be useful in the experiment in predicting maintenance performance. However, their metrics can only be used for interactive systems to assess their user interface design.

In summary, most of the existing work addresses the metrics at the micro level or at the algorithmic level and not at the macro level or the architectural level. A conventional distinction is made between architectural or high-level design and algorithmic or low-level design, and according to Rombach, the former has more influence on maintainability than the later [10]. In this paper, a macro metric at the architectural level is defined for maintainability.

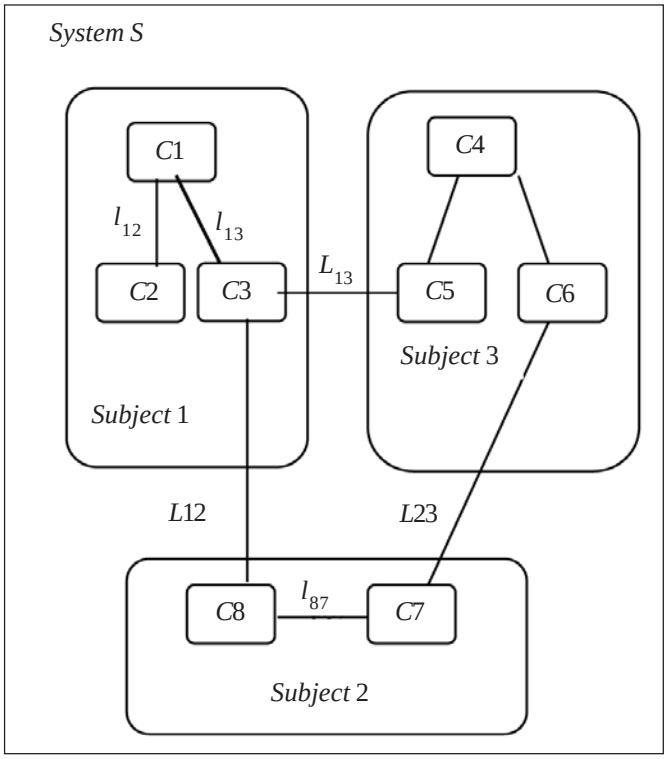

Figure 1. Example of an object-oriented system decomposition

3. Object-Oriented System Design Model

The model is inspired from the Coad and Yourdon object-oriented analysis and design model [3] and is influenced by Courtois' quasi-decomposable systems model [4]. The components are defined below and illustrated with the graphical model in Figure 1.

3.1 Object System Model

Object: An instantiation of some class which is able to save state and which offers a number of methods to examine or affect this state.

Class: A set of objects that share a common structure and a common behavior manifested by a set of methods.

Attribute: Defines the structural properties of classes.

Method: An operation upon object, defined as part of the declaration of a class.

Instance connection: The process of creating an instance of the object and binding or adding the specific data that one class needs with other classes.

Message connection: A request that a class makes to another class to perform a method.

Inheritance: A relationship among classes, wherein an object in a class acquires characteristics from one or more other classes.

Link: A link between two classes. There are two kinds of links: physical and conceptual.

A physical link represents a message connection or instance connection or inheritance (generalization specialization structure) or a whole part structure between two classes.

There is a conceptual link between two classes if they share some property. For example if two classes are instantiated in the same slice time, they are linked by a conceptual time link.

System: A set of classes, linked together and contributing to a common goal.

Subject: A system linked to other subjects in a larger system.

Object-oriented system decomposition: A set of subjects such that each class is part of one subject only, and each internal link is either internal to a subject or between two classes of different subjects.

4. Maintainability Model

4.1 Introduction

There are few maintenance effort models, and those available are usually based on a number of lines of codes changed by class [6] or [1]. Their effort models are simple and not applicable to decomposition at a system level. A maintenance effort model based on design information is defined.

4.2 Maintainability Metric

Maintenance comprises usually three activities:

Activity 1: Analysis, isolation of the components affected by the change.

Activity 2: For each affected module, all activities related to its change.

Activity 3: Integration and regression tests.

Most effort models do not involve activity 1. It was assumed that the corresponding effort was independent of the decomposition and therefore not considered. The proposed model involves activities 2 and 3 and is based on the assumption that decomposition is better, if a change affects fewer subjects. The general model is given in the following formula (Formula 1):

$$Effort_{change} = Effort_{classes} + Effort_{integration} \quad (1)$$

$Effort_{change}$ is the total effort to implement a change, $Effort_{classes}$ is the sum of the efforts required to change each affected class,

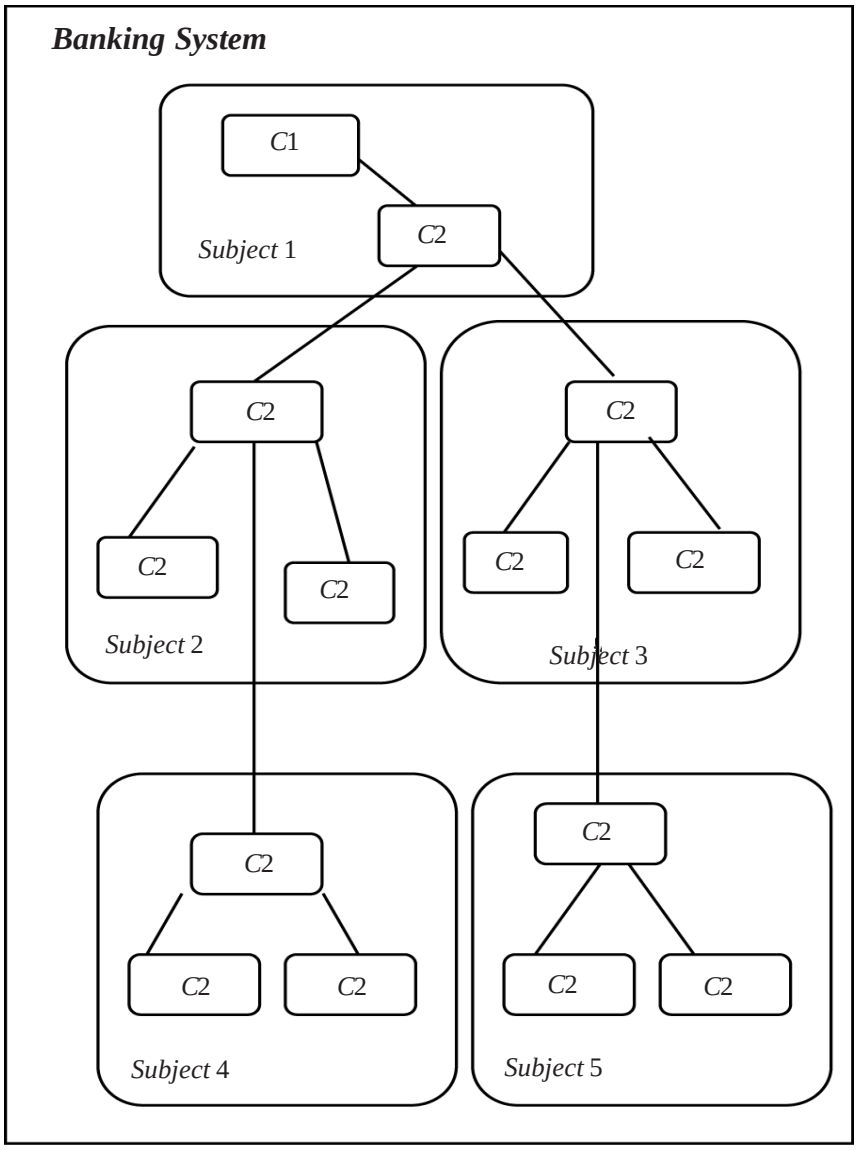

Figure 2. Example of an object-oriented banking system decomposition

and $Effort_{integration}$ is the effort required for integration testing and regression testing.

The integration effort is assumed to be proportional to the size (number of classes) of the affected subjects. The concept of ideal decomposition is introduced, based on the idea that for a given change, there is a minimal effort to implement it. The corresponding decomposition is such that all affected classes are in one subject. Also, the concept of worst decomposition corresponds to the case where each subject contains at least one affected class.

The maintainability of decomposition for a change *ch* ($maintainability_{ch}$) is defined in Formula 2:

$$maintainability_{ch} = 1 - \frac{Effort_{ch} - IdEffort_{ch}}{WstEffort_{ch} - IdEffort_{ch}} \tag{2}$$

Where $Effort_{ch}$ is the effort to implement the change *ch* on the given decomposition, $IdEffort_{ch}$ is the effort to implement the same change on the corresponding ideal decomposition, and $WstEffort_{ch}$ is the effort required to implement the same change on the corresponding worst decomposition for that change. Because of the definitions of the ideal decomposition and of the worst decomposition for a change, formula 2 turns out in formula 3:

$$maintainability_{ch} = 1 - \frac{\sum_{SS\ j \in affectedsubjects} |SB_j| - r}{|S| - r} \quad (3)$$

Where $|SS_j|$ is the size of subject *j*, i.e., the number of classes in subject *j*, $|S|$ is the number of classes in the system, and *r* is the number of classes affected by the change. Complete details on the derivation of formula 3 can be found in [11].

The range of this maintainability metric is 0 –1. A maintainability of 1 will be achieved if all the *r* affected classes are in only one subject. Conversely, if all subjects are affected, the maintainability will be 0.

For a batch of *n* changes, the maintainability is the mean of the maintain abilities of the individual changes (see Formula 4):

5. Experiment

5.1 Object-oriented Banking System Description

To illustrate the application metrics defined in this paper, a real banking information system is selected and presented below.

The bank offers money services for customers holding accounts and contracting loans in their succursales. The account type can be checking or savings. The loan type can be commercial or personal. Two types of transactions: Account transactions (Deposit and Withdraw) and Loan transactions (Release and Refund).

In this example, the banking system is decomposed into 5 subjects: Subject1, Subject2, Subject3, Subject4 and Subject5. (See Figure 2 below).

Subject1 is the customer subject, contains the following classes: C1: Succursale, C2: Customer.

Subject2 is the Account subject, contains the following classes: C3: Account, C4: Checking Account, C5: Saving Account.

Subject3 is the Loan subject, contains the following classes: C6: Loan, C7: Commercial Loan, C8: Personal Loan.

Subjec4 is the Account Transactions subject, contains the following classes: C9: Account Transaction, C10: Deposit, C11: Withdraw

Subject5 is the Loan Transactions subject, contains the following classes: C12: Loan Transaction, C13: Release, C14: Refund.

5.2 Batch of changes

We define a batch of changes of the banking system as one to any of the three components:

A component refers to either an attribute or a method or a class.

Examples of changes are:

- Add or delete or update of an attribute
- Add or delete or update a method
- Add or delete or update a class

If one change affects a class, it affects also the subject of this class. The effort is calculated according to the number of classes affected in the same subject (Formula 3).

5.3 Experiment and Results

Different types of changes lead to different sets of affected classes. For example the change of the attribute has an impact in all the classes referencing this attribute. For example, a change in the loan rate attribute has an impact in two classes: C13: Refund class in subject 5 and C6: Loan class in subject 3.

Applying Formula 3:

$$maintainability_{ch} = 1 - \frac{\sum_{SS\ j \in affectedsubjects} |SB_j| - r}{|S| - r}$$

Where $|SB_j|$ the number of classes in subject *j*:

In Subject3 is equal 3.

In Subject5 is equal 3.

r is the number of classes affected by the change = 2

$|S|$ is the number of classes in the system = 14

$$maintainability_{ch} = 1 - \frac{3+3-2}{14-2} = 0.67$$

If the batch of changes contains 13 changes, we should apply Formula 3 for each change and than apply Formula 4 to calculate the maintainability of the whole system.

6. Conclusion

In this paper, a maintainability model is defined and illustrated to a real banking system. A controlled experiment was conducted to investigate the effect of the system decomposition on the maintainability. Given a batch of changes, the metric is applicable in the early phases of the life cycle development process, in the design phase. The metric was found to be useful in the experiment in predicting the maintainability of o-o systems. The metric can help also to reduce the maintenance costs of o-o systems. The designer can reduce the number of affected subjects and increase the quality of the system.

Short term plans include applying the metric to three real o-o systems. This will help to give weights to links defined elsewhere [12].

References

[1] Basili, V R., Briand, L C., Melo, W L. (1996). A validation of object-oriented design metrics as quality indicators, *IEEE Transactions on Software Engineering*, October, p.751-761.

[2] Chaumun, M. A., Kabaili H., et al. (1999). A change impact model for changeability assessment in object-oriented software systems, *In*: Proc. Third European Conference on Software Maintenance and Reengineeing. *IEEE Comput. Soc*. Los Alamitos, CA. USA.

[3] Coad, P., Yourdon, E. (1991). Object-Oriented Analysis, *Prentice Hall*.

[4] Courtois, P. J. (1985). On Time and Space Decomposition of Complex Structures, Com. of the ACM, 28, 6, June, p. 590-604.

[5] Kiran, G. A., Haripriya, Jalote, P. (1998). Effect of Object Orientation on Maintainability of Software, *In*: ICSM97, Bari, Italy, October, p. 114-121.

[6] Li, W., Henry, S. (1993). Object Oriented Metrics that Predict Maintainability, *The Journal of Systems and Software*, 23 (2) 111-122.

[7] Pressman, R. S. (1997). Software Engineering: A Practitionner's Approach, 4th Edition McGraw-Hill.

[8] Rajendra, K. Bandi., Vijay, K.Vaishnavi., Daniel E. Turk. (2003). Predicting Maintenance Performance Using Object-Oriented Design Complexity Metrics, *IEEE Transactions on Software Engineering*, January, 29 (1) 77-87.

[9] Rajaraman, C., Lyu, M R. (1992). Reliability and Maintainability Related Software Coupling Metrics in C++ Programs, *In*: Proc 3rd Intl. Symposium on Software Reliability Engineering. IEEE Computer Press.

[10] Rombach, D. (1990). Design Measurement: Some Lessons Learned, IEEE Software, March, p. 17-25.

[11] Tagoug, N. (2002). Information Systems Decomposition: Exploratory Study of Influencing Factors, PhD Dissertation, University of Montreal, Montreal, Canada.

[12] Tagoug, N. (2002). Object Oriented System Decomposition Quality, *In*: Proc. Of the HASE, Tokyo, Japan, October.