2F-4

Detection of Design Problems in Object Oriented Development

Munkhnasan Choinzon and Yoshikazu Ueda
Ibaraki University

1 Introduction

The last goal of the software development is to produce high-quality software. However, assessing the quality of a software at an early stage, not after a product is ready, is essential for saving the cost. If design defects are not fixed in the design phase, the cost for fixing them after delivery of the software is 100 times higher[1].

This paper introduces a metrics-based approach for detecting defects which have a significant impact on the OO design quality. Moreover, thresholds to help in identifying critical values are defined for each of the metrics.

2 Development of Approach

This section introduces the approach for detecting defects in OO designs.

Within the framework of this work, following four sequential steps are carried out.

- Survey of design defects into literature and define a list of design problems(defects).
- Categorize the design defects found during the survey into corresponding design properties that are affected by that design defect.
- Identify metrics to detect each design defect.
- Identify threshold for each of the metrics.

These steps are described in detail in the following subsections.

2.1 Identifying OO Design Defects

We have made a wide survey into literature to collect the OO design defects that may occur when defining the classes and objects, attributes and methods of a class, relationships between classes, and class hierarchies. The number of papers that we have studied during the our survey is thirty five.

During the survey, we found many design guidelines, rules, and flaws originating from various authors[3], [7]. Violations of these design rules may contribute a potential design defect to the software. However, it is not always true that every violation of a design rule corresponds to a design defect, since heuristics are just guidelines, and not hard and fast rules that must be followed. Depending on the application, violations of some design heuristics may be acceptable to some extent.

We collected forty seven OO design defects which best covered our interesting area for further examination. The rejected guidelines and flaws were either equivalent to an already chosen one and thus redundant, or requires too detail information of implementation and thus impossible to detect from a design.

2.2 Classifying Design Defects into OO Design Properties

In this section, design defects described in the previous subsection are categorized into OO design properties depending on what design property's quality aspect is violated. Design properties are characteristics to define the quality of an OO design from the perspective of the internal attributes.

We choose the following ten design properties as OO design internal quality characteristics: encapsulation, abstraction, cohesion, coupling, messaging, complexity, inheritance, composition, and polymorphism. We consider that these design properties are most general and most important characteristics for indicating internal quality of OO design.

2.3 Defining Metrics to Detect Design Defects

Metrics is an important tool for detecting the defects. Each of the design defects identified in this section can be detected by metrics defined on each defect. Many design metrics for OO design have been proposed by the authors[4], [6]. Twenty two well-known metrics are used in the assessment of some design properties, such as size, encapsulation, and cohesion.

For the other design defects, there are no suitable metrics, therefore, we define eighteen new metrics.

2.4 Identifying Thresholds on Metrics

Once we have a metrics value, we need to judge whether the value indicates critical situation or not. Thresholds can help us do this judgement. If a design metrics exceeds a certain threshold, the design ele-

Table 1: Definition of New Design Metrics

Metrics	Definition	Threshold of Undesirable Value
NOA	Number of(N.of) attributes in a class	7-9 little bad, 9< bad
NOM	N.of methods in a class	20-30 little bad, 31-50 bad, 50< very bad
NMSM	N.of message sends in a method	10-11 little bad, 12-14 bad, 14< very bad
NOP	N.of parameters in a method	5-6 little bad, 7-8< bad, 8< very bad
DAC	Data abstraction coupling (N.of abstract data types within a class)	3-4 little bad, 5-6 bad, 6< very bad
CBO	Coupling between object classes (N.of other classes to which class is coupled)	6-7 little bad, 8-9 bad, 9< very bad
NID	N.of inherited attributes	0 bad
NMMI	N.of messages to a method itself	0 bad
NMSSO	Ratio of the n.of messages sent to same object from same method	60∼70% little bad, 70∼80% bad, 80∼100% very bad

ment can then be considered as "critical" and must be redesigned.

We define thresholds for all design metrics defined in the previous subsection. If a metrics value is equal to threshold or in a range of undesirable values defined by threshold, it means that there exists the defect and should be reconsidered.

There is no general standard for thresholds of metrics. Different researchers proposed different threshold values for the same OO metrics. For this reason, we define several thresholds for some metrics instead of a single one. Then we divide these thresholds on one metrics into three levels, so as to get more fine-grained threshold. To make clear these three levels meaning, we use linguistics values such as "little bad", "bad", and "very bad". Due to the lack of space, we show only few of the metrics and their undesirable thresholds in Table 1.

As an example, for metrics NOM, threshold value is proposed by [2], [5], and [6] respectively 20, 50, and 60. These three of thresholds are quite different, so we define several thresholds for metrics NOM, according to our scheme as mentioned above. 3 levels of thresholds on metrics NOM are defined as follows: "little bad" ranges from 20 to 30, "bad" ranges from 31 to 50, and "very bad" is larger than 50.

There is defect "There exist methods which do not make sense in the class's responsibilities". For this defect, we define metrics NMMI "number of messages to a method itself". If there is no message to the method itself, it means that method is useless. Thus threshold is 0.

There is another defect "A Subclass uses only part of base class's interface". This implies that the subclass does not inherit its data. So we define threshold of 0 for metrics NID "number of inherited attributes", which detects this defect.

We believe that detection of design elements, which have still high probability to be "critical" even if their values below the threshold, can be improved by defining several thresholds on their metrics.

3 Conclusions and Future Work

In this paper, we have presented a metrics-based approach to identify design defects in an OO design.

By defining metrics on each design defect, these defects can be detected automatically. Consequently, many design heuristics, and flaws which are described qualitatively can be evaluated quantitatively. On the other hand, it can be seen that intended application of metrics has been clearly determined. Furthermore, we have proposed thresholds to help in judging whether a metrics value indicates problematic situation or not, on each metrics. The levels of the thresholds allow the user to judge whether metrics value is acceptable or not by intuition.

Link to external attributes of OO design and emprical validation of our work are remained as future work. An automated tool that helps collect metrics values more efficient and effective way is now under development.

References

[1] B. Boehm and V. Basili, "Software Defect Reduction Top 10 list", IEEE Computer, 34(1), 135-137, 2001.

[2] M. Akroyd, "AntiPatterns Session Notes", Object World West, 1996.

[3] W.J. Brown, R.C. Malveau, H.W. McCormick, and T.J. Mowbray, "AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis", John Wiley & Sons, 1998.

[4] S.R. Chidamber and C.F. Kemerer, "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, 21(3), 263-265, 1994.

[5] R.E. Johnson and B. Foote, "Designing reusable classes", Journal of Object-Oriented programming, 1(2), 22-35, June 1988.

[6] M. Lorenz and J. Kidd, "Object-Oriented Software Metrics: A Practical Guide", Prentice Hall, 1994.

[7] A.J. Riel, "Object Oriented Design Heuristics", Addison-Wesley, 1996.