Metrics About Fault-Proneness In Object-Oriented Systems

Sabrina Bhm
Universitt Ulm
Ulm, Germany
sabrina.boehm@uni-ulm.de

***Abstract*—Software quality is becoming increasingly important in modern times. Faulty or insufficient software can have severe consequences. For this reason, aspects such as qualitiy, security, reliability and maintainability must be considered at an early stage of the development process. Early detection of bugs or problems in the software prevents enormously high costs at the later times. In safety-critical areas, for example, the property of fault-proneness must be carefully considered. In order to include this aspect early in the development process, there are a number of metrics and methods that can estimate the fault prediction or reduce the fault-proneness of the software. By following certain rules and procedures, high costs and time can be saved for both developers and consumers. The goal of this work is to illustrate some of the widely used methods and design metrics that consider fault-proneness in object-oriented systems.**

I. Introduction

In the field of software development there are some keywords like security, consistency or reliability that are indispensable today. Everyone wants the best and most intelligent software, but with increasing complexity the possibility of fault-proneness in the software increases. In the following, the aforementioned topic is examined under the programming language model of object orientation and metrics that can be considered, which is an important topic in research trends in software technology nowadays. One might think that testing the software and the resulting errors is one of the last steps in the software development process, but this is a false assumption. The earlier the system is examined and tested for critical points, the more work will be saved in later, more cost-intensive development steps.

Fault-proneness is an important external software quality attribute of interest to software developers and practitioners. The fault-proneness of an object-oriented class indicates the extent to which the class, given the metrics for that class, is fault-prone. Since it is difficult to measure the fault-proneness of software that is not yet in use, predictive models are applied to estimate the fault-proneness of software classes.

Several studies (hier zitieren) have been carried out to determine which metrics are useful in capturing important quality attributes such as fault-proneness and fault prediction, which are summarized in the following sections. Furthermore the main research objective is the consolidation of some metrics and methods related to fault-proneness in the software development process.

This paper is organized as follows: Section III presents research methodology and background as well as related work. In Section II the content background is described. Section V contains the analysis of design metrics and the results of some empirical studies. After the investigation of the effects on the fault-proneness metrics, a discussion follows, including the classification of the relevance in the software development process and the parties involved. The paper is concluded and gives a future outlook in Section VII.

II. Content Background

In the following, we will consider fault-proneness as already mentioned, but from a restricted point of view, in an object-oriented context. Many software systems in use are based on object-oriented design. This means that data and program code are encapsulated in reusable objects. Everything is based on the communication of objects. For this purpose, classes, interfaces and methods, as well as attributes are declared and thus serve to represent states. This structure alone protects against fault-proneness, since the code is reusable and thus the programming effort is reduced. Therefore, object orientation in itself offers advantages for maintainability and reusability. Thus, fewer errors occur [1], [2].

In the object oriented context there are a few constructs like class, coupling, cohesion, inheritance, information hiding and polymorphism, which also influence the fault-proneness. Examples of languages that program object-oriented are C#, C++ or Java. One of the most common aspects incorporated into metrics is that of coupling, which refers to the degree of direct knowledge that one element has of another. The degree of interconnectedness of the whole system is a key element in the software development, for example it is important how big the effect of changing one attribute of one class has on all others and especially how many. The coupling of the classes in the system plays a central role in the effects of software faults. To give a short introduction in object orientation and the relation between its properties, it is shown in figure 1, that invoking methods and accessing attributes is the basic principle of communication of these software systems, which means that faults can occur here depending on the frequency of use. In how far the interaction of the individual components are connected with the fault-proneness metrics, is described in the metrics chapter V more near.

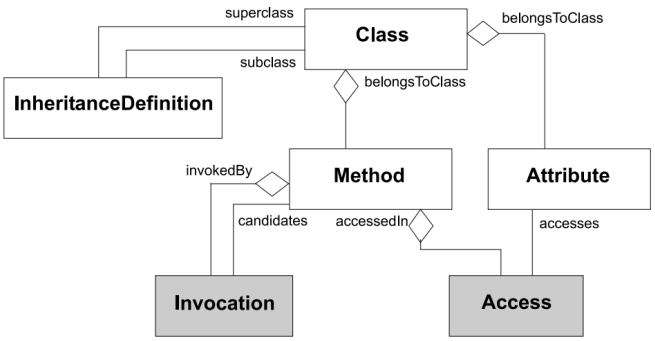

Fig. 1. A meta model for main object-oriented concepts [2].

But why should a programmer actually pay attention to fault-proneness at all? The accurate prediction of where bugs are more likely to occur in the code can help manage efforts in testing, reduce costs, and improve the quality of the software. For different programming paradigms and programming constructs different rules apply which must be considered in relation to fault-proneness. The restriction on object orientation is to facilitate the understanding. In addition many principles are contained, which are taken up in other programming concepts again. Thus some conclusions which are drawn here are also differently realizable and applicable to other software concepts.

In terms of fault-proneness, other quality characteristics that must not be forgotten also play a role like reliability, correctness, completeness, maintainability, as some are dependent on each other in terms of time. For example, software may not be reliable or correct if faults occur frequently that cause the software to become unusable.

When software errors are made, it is often not only tedious to find the programming error, but also expensive. In large systems that are used by many people every day, a small error can cost millions. When software errors are made, it is often not only tedious to find the programming error, but also expensive. In large systems that are used by many people every day, a small error can cost millions. A common misconception is that fault-proneness should only be considered at the end of the software development process. Especially at the beginning of the development you should build the software architecture in a way that it is less fault-prone. Changes in the architecture are always more expensive later. In addition, even before the programming itself begins, attention should be paid in the planning to various concepts and metrics, which are shown below.

- to what extent can you do analysis now and what should you consider when developing object-oriented

- research background: summary of interpretation of previous research and what this paper is about depending on the research

III. Research Methodology

First, this paper deals with research on the keywords ”fault-proneness”, ”fault-proneness in object-oriented systems” and ”fault prediction metrics”. To get more literature on this topic, the keywords linked in the papers were used as further search. Among them are ”object-orientation”, ”object-oriented design metrics” and ”faulty classes”. As first an overview was created in such a way, in order to be able to classify the term ”fault-proneness” into the software development process. The top listed papers that were suggested in google scholar were either sorted out or read more closely by reading the abstract. If the abstract sounded interesting for the topic, then the introduction was read and the conclusion. In some papers, such as the research on metrics, the exact procedure and the analysis section were also read in detail. Two of the metrics were then chosen as a showcase model to illustrate fault-proneness in software systems. Further literature was researched for the content background to summarize the basic knowledge about object orientation and a simple understanding of objects, classes and methods.

IV. Related Work

- related work, in welchen papers machen die hnliche sachen wie ich jetzt anschaue etc

- andere die Listen ber metriken in sw untersuchen, wie grenze ich mich von denen ab!

- Auswirkungen von fehlern

- kritische auseinandersetzung, fr welche flle sind welche metriken gut und wieso

- Wann setz ich welche Metriken ein wann im SW Prozess!! und was kann da eintreten passieren

- further metrics/studies; cite more related work [4], [5], [6], [7], [8]

In order to provide guidance on how to proceed in the software process, some metrics will be explained.

- from the large overall background to the more detailed topic of fault proneness $\rightarrow$ transition to metrics

V. Summary of Metrics

[2] zuerst gibts auch metriken die entweder class metirken oder methoden metirken oder attribute metriken sind. kann man so klassifzieren

- evaluation possibilities

- The chapter in which the metrics are enumerated in general and then (maybe) two are picked out and discussed in detail

- introduction to some metrics, properties, studies that deal with fault-proneness and investigate the issue of FP

- Some of these metrics are based simply on counting the number of interactions

- add path connectivity class cohesion (PCCC) to table

- metrics table:

TABLE I
METRICS.

Abbrevation	Definition	Sources
CBO	Coupling between Objects for a class is .. Coupling between Objects for a class is ..Coupling between Objects for a class is ..Coupling between Objects for a class is ..Coupling between Objects for a class is ..	[7]
LOC	Lack of Cohesion counts ..	[8]
DOI	Depth of Inheritance ..	[8]
...	...	...

A. A Precise Method-Method Interaction-Based Cohesion Metric for Object-Oriented Classes

- explain as an example the Precise Method-Method Interaction-Based Cohesion Metric for Object-Oriented Classes from paper [9].

- explain method: the proposed class cohesion metric is based on counting the number of possible paths in graph that represents the connectivity pattern of the class members

- relevance of methods, attributes in classes and their connectivity, figure 2, figure 3, figure 4.

- search for other graphical representation ($\rightarrow$ java code snippet)

- benefits and limitations of the method (maybe to results?)

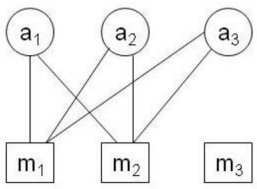

Fig. 2. Sample representative graph for a hypothetical class [10].

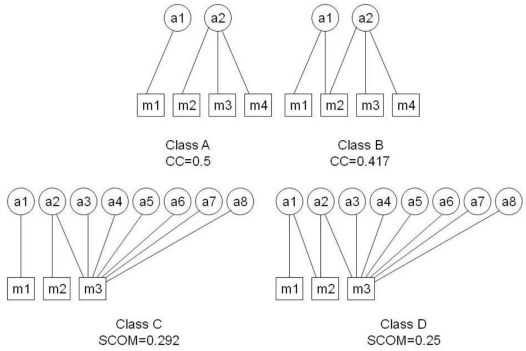

Fig. 3. Classes with different method-method connectivity patterns [10].

B. A Bayesian network

- is a concise representation of a joint probability distribution on a set of statistical variables

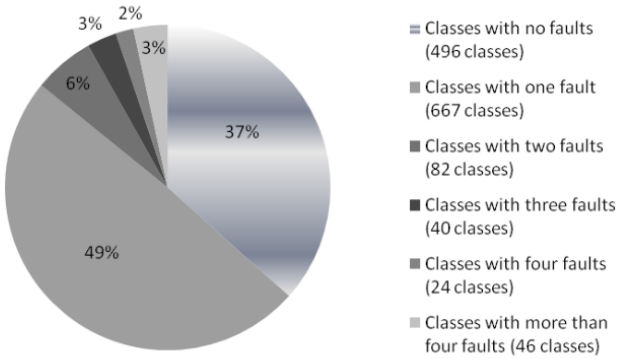

Fig. 4. Sample representative graph for a hypothetical class [10].

- encoded as an acyclic graph of nodes and directed edges [11].

- how Bayesian methods can be used for assessing software fault content and fault proneness

- dependent variables, which serve as surrogate metrics of software quality: fault content and fault proneness.

- a BN model whose underlying representation is the generalized linear model (infos to the model)

- (maybe) definition probabilistic network (acyclic graph $G = (V, E)$; A set S, of (prior) conditional probability distributions)

- (maybe) given a BN structure, the joint probability distribution over X is encoded as
$p(X) = \prod_{i=1}^{n} p(X_i|P_{ax_i})$

- some maths: short summary of the mathematical aspects!?

- explain the variables and the fault-pron context

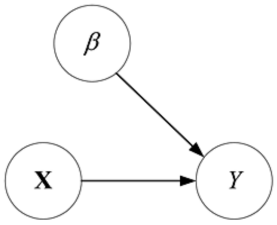

Fig. 5. BN representation of a general linear model.

- model parameters from [11]: (somehow merge with table!?; until now almost copied out from paper)

1. Weighted methods per class (WMC): The number of methods implemented in a given class.
2. Depth of inheritance tree (see table)
3. Response for class (RFC): Number of methods implemented within a class plus the number of methods accessible to an object class due to inheritance. (Traditionally, it represents the number of methods that an object of a given class can execute in response to a received message [11].)
4. Number of children (NOC): The number of classes that directly inherit from a given class.

6. Coupling between object classes (CBO): The number of distinct non-inheritance related classes to which a given class is coupled. (i.e., when a given class uses the methods or attributes of the coupled class [11].)
7. Lack of cohesion in methods (LCOM): A measure of the degree to which a class represents single or multiple abstractions. There are varying definitions for LCOM. (i.e., by computing the average percentage of methods in a given class using each attribute of that class, and then subtracting that percentage from 100 percent [11].)
8. Source lines of code (SLOC): This is measured as the total lines of source code in the class and serves as a measure of class size.

- dependencies between metrics and impact

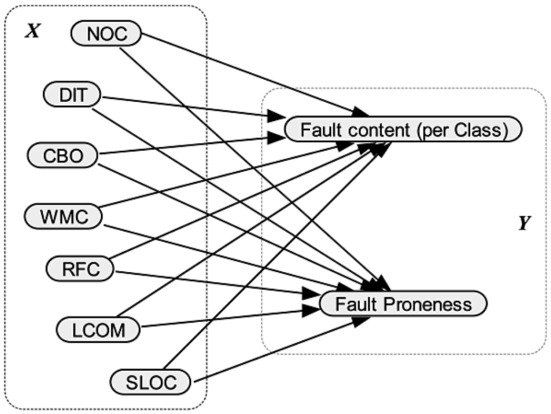

Fig. 6. BN model for defect content and defect proneness analysis.

- summary of study/paper; note important aspects
- RESULTS
- BN: the results of performing multiple regression, the metrics WMC, CBO, RFC, and SLOC are very significant for assessing both fault content and fault proneness
- this specific set of predictors is very significant for assessing fault content and fault proneness in large software systems
- it was observed that import coupling (that count the number of other classes called by a class) metrics are strongly associated with fault pron and predict faulty classes with high accuracy
- Associate refactoring to the last sections

VI. Discussion

- discuss the metrics depending on the benefits/limitations and which metrics scored well in comparison (die genauer betrachteten oder alle)
- discussion with the classification according to relevance in the development process and the relevance for stakeholder
- what you should always pay attention to, no matter how large the project is; what depends on the size of the project; other projectsprops ($\rightarrow$ limitations)

A. Classification of relevance

- in which parts of the development is it important to pay attention to fault proneness
- the aspect of fault-proneness must be considered early on, during planning and development
- in addition always pay attention to all aspects early to avoid costs etc.
- who is most influenced, developers or customers
- for the developer important how to build and construct his object-oriented classes, with a metric as orientation to avoid bugs eg

B. Limitations

- Refactoring classes requires not only accounting for cohesion, but also accounting for other quality attributes, such as coupling (ref. [9])
- all errors that are feasible can never be covered
- many studies have only been tested with small software projects and classes, which is not very meaningful over large ones

C. Benefits

- advantages if fault-proneness is included in early software steps; pay attention to metrics
- note time and money

VII. Conclusion

- summarize discussion and results; name important metrics/properties that should stay in mind after reading this
- what can reduce the fault-pron in software development
- outlook
- This paper discussed the concept of fault proneness using the example of object-oriented programming and design metrics, which means that for the most part the results can only be transferred to the object-oriented context.
- maybe future work
- final catch and the link to the superordinate context (research trends in software technology); link to introduction and motivation

References

[1] R. G. Fichman and C. F. Kemerer, "Adoption of software engineering process innovations: The case of object orientation," *Sloan management review*, vol. 34, pp. 7–7, 1993.

[2] M. Lanza, S. Ducasse *et al.*, "Beyond language independent object-oriented metrics: Model independent metrics," in *Proceedings of 6th International Workshop on Quantitative Approaches in Object-Oriented Software Engineering*. Citeseer, 2002.

[3] Y. Singh, A. Kaur, and R. Malhotra, "Empirical validation of object-oriented metrics for predicting fault proneness models," *Software quality journal*, vol. 18, no. 1, p. 3, 2010.

[4] K. El Emam, W. Melo, and J. C. Machado, "The prediction of faulty classes using object-oriented design metrics," *Journal of systems and software*, vol. 56, no. 1, pp. 63–75, 2001.

[5] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Transactions on software engineering*, vol. 22, no. 10, pp. 751–761, 1996.

[6] L. C. Briand, W. L. Melo, and J. Wust, "Assessing the applicability of fault-proneness models across object-oriented software projects," *IEEE transactions on Software Engineering*, vol. 28, no. 7, pp. 706–720, 2002.

[7] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on software engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[8] ——, "Towards a metrics suite for object oriented design," in *Conference proceedings on Object-oriented programming systems, languages, and applications*, 1991, pp. 197–211.

[9] J. Al Dallal and L. C. Briand, "A precise method-method interaction-based cohesion metric for object-oriented classes," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 21, no. 2, pp. 1–34, 2012.

[10] J. Al Dallal, "Fault prediction and the discriminative powers of connectivity-based object-oriented class cohesion metrics," *Information and Software Technology*, vol. 54, no. 4, pp. 396–416, 2012.

[11] G. J. Pai and J. B. Dugan, "Empirical analysis of software fault content and fault proneness using bayesian methods," *IEEE Transactions on software Engineering*, vol. 33, no. 10, pp. 675–686, 2007.