Fault-Proneness

Sabrina Böhm
Universität 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. The conditional probability that a class in software systems contains a fault, given the metrics for that class, is called fault-proneness. In the object oriented context there are a few constructs like class, coupling, cohesion, inheritance, information hiding and polymorphism, which influence the fault-proneness. 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 networking of the classes in the system plays a central role in the effects of software faults. Several studies have been carried out to determine which metrics are useful in capturing important quality attributes such as fault-proneness, effort or productivity, which are summarized in the following sections.

This paper is organized as follows: Section II presents research methodology and background as well as related work. In Section III the content background is described. Section IV 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 VI.

II. Research Methodology

- research method/background/strategy and related work, research objectives

- the way I came across literature: google scholar and there recursively linked further literature and papers on terms like ”fault-proneness”, ”fault-proneness in object-oriented systems”, ”fault prediction metrics”, so on

- discuss findings in a chronological manner

- [13] Empirical validation of object-oriented metrics for predicting fault proneness models

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

- the main research objective is the consolidation of some metrics and methods related to fault-proneness in the software development process

III. Content Background

- fault content, fault proneness, fault prediction background

- relation to other aspects like reliability, correctness, completeness, maintainability etc.

- classification in software development - why is it important to pay attention to FP at all?

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

- name important aspects as classes, methods, attributes

- from the large overall background to the more detailed topic of fault proneness → transition to metrics

IV. Analysis of Metrics

- 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

- further metrics/studies; cite more related work [10], [11], [12], [14], [15]

- 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 ..	[14]
LOC	Lack of Cohesion counts ..	[15]
DOI	Depth of Inheritance ..	[15]
..	..	..

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 [8].
- 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 1, figure 2, figure 3.
- search for other graphical representation ($\rightarrow$ java code snippet)
- benefits and limitations of the method (maybe to results?)

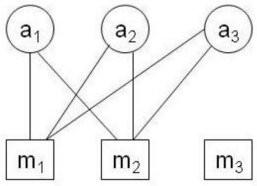

Fig. 1. Sample representative graph for a hypothetical class [3].

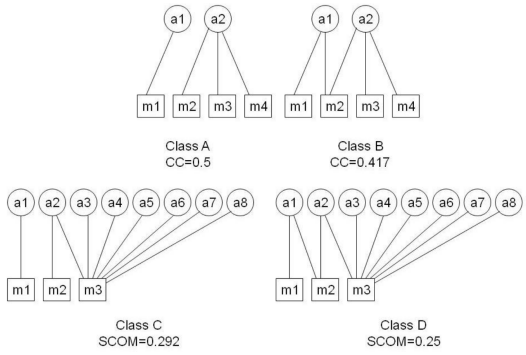

Fig. 2. Classes with different method-method connectivity patterns [3].

B. A Bayesian network

- is a concise representation of a joint probability distribution on a set of statistical variables
- encoded as an acyclic graph of nodes and directed edges [9].

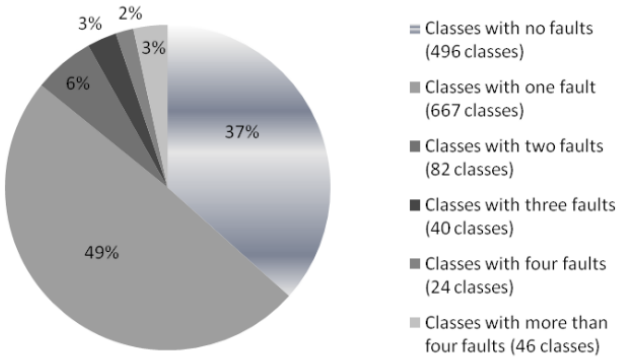

Fig. 3. Sample representative graph for a hypothetical class [3].

- 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

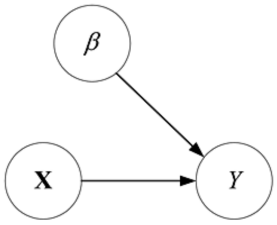

Fig. 4. BN representation of a general linear model.

- model parameters from [9]: (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 [9].)
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 [9].)

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 [9].)
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

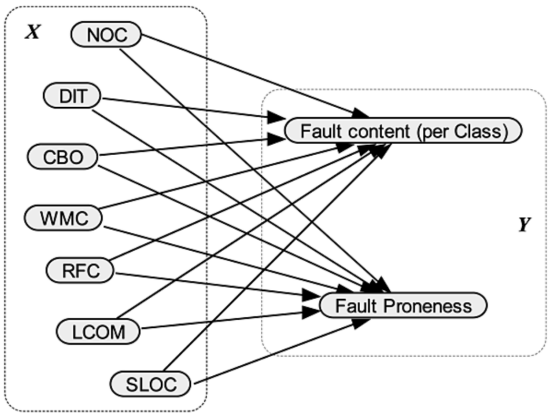

Fig. 5. BN model for defect content and defect proneness analysis.

- summary of study/paper; note important aspects

C. Results

- maybe also delete this section and add this content to the study/metrics below
- 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

V. Discussion

- discuss the metrics depending on the benefits/limitations and which metrics scored well in comparison
- 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 (→ 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. [8])
- 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

VI. 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] Aggarwal, K. K., et al. "Investigating effect of Design Metrics on Fault Proneness in Object-Oriented Systems." J. Object Technol. 6.10 (2007): 127-141.

[2] Aggarwal, K. K., et al. "Empirical analysis for investigating the effect of object-oriented metrics on fault proneness: a replicated case study." Software process: Improvement and practice 14.1 (2009): 39-62.

[3] Al Dallal, Jehad. "Fault prediction and the discriminative powers of connectivity-based object-oriented class cohesion metrics." Information and Software Technology 54.4 (2012): 396-416.

[4] Al Dallal, Jehad. "The impact of accounting for special methods in the measurement of object-oriented class cohesion on refactoring and fault prediction activities." Journal of Systems and Software 85.5 (2012): 1042-1057.

[5] Bellini, Pierfrancesco, et al. "Comparing fault-proneness estimation models." 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05). IEEE, 2005.

[6] Aggarwal, K. K., et al. "Investigating effect of Design Metrics on Fault Proneness in Object-Oriented Systems." J. Object Technol. 6.10 (2007): 127-141.

[7] Radjenovic, Danijel et al. “Software fault prediction metrics: A systematic literature review.” Inf. Softw. Technol. 55 (2013): 1397-1418.
[8] Dallal, Jehad Al and L. Briand. “A Precise Method-Method Interaction-Based Cohesion Metric for Object-Oriented Classes.” ACM Trans. Softw. Eng. Methodol. 21 (2012): 8:1-8:34.
[9] G. J. Pai and J. Bechta Dugan, ”Empirical Analysis of Software Fault Content and Fault Proneness Using Bayesian Methods,” in IEEE Transactions on Software Engineering, vol. 33, no. 10, pp. 675-686, Oct. 2007, doi: 10.1109/TSE.2007.70722.
[10] El Emam, Khaled, Walcelio Melo, and Javam C. Machado. ”The prediction of faulty classes using object-oriented design metrics.” Journal of systems and software 56.1 (2001): 63-75.
[11] Basili, Victor R., Lionel C. Briand, and Walcélio L. Melo. ”A validation of object-oriented design metrics as quality indicators.” IEEE Transactions on software engineering 22.10 (1996): 751-761.
[12] Briand, Lionel C., Walcelio L. Melo, and Jurgen Wust. ”Assessing the applicability of fault-proneness models across object-oriented software projects.” IEEE transactions on Software Engineering 28.7 (2002): 706-720.
[13] Singh, Yogesh, Arvinder Kaur, and Ruchika Malhotra. ”Empirical validation of object-oriented metrics for predicting fault proneness models.” Software quality journal 18.1 (2010): 3.
[14] Chidamber, Shyam R., and Chris F. Kemerer. ”A metrics suite for object oriented design.” IEEE Transactions on software engineering 20.6 (1994): 476-493.
[15] Chidamber, Shyam R., and Chris F. Kemerer. ”Towards a metrics suite for object oriented design.” Conference proceedings on Object-oriented programming systems, languages, and applications. 1991.