A Validation of Object-Oriented Design Metrics as Quality Indicators

Victor R.Basili, Lionel C.Briand and Walcélio L.Melo

Presentation: Fischer Jônatas

Motivation

- Software metrics are, thus, necessary to identify where the resources are needed; they are a crucial source of information for decision making.
- It needs to be able to identify fault-prone modules so that testing / verification effort can be concentrated on these modules

Issues

- Some studies have concluded that ”traditional” product metrics are not sufficient for characterizing, assessing, and predicting the quality of OO software systems

Main goal

- The purpose evaluates wheter or not Chidamber and Kemerer's metrics are useful for predicting the probability of detecting faulty classes.

Metrics used at the Empirical Study

- **WMC** - Weighted Methods per Class
- **DIT** - Depth of Inheritance Tree of a class
- **NOC** - Number Of Children of a class
- **CBO** - Coupling Between Object classes
- **RFC** - Response For a Class
- **LCOM** - Lack of Cohesion on Methods

Hypotheses(hypothesis)

- **H-WMC**: A class with significantly more member functions than its peers is more complex and, by consequence, tends to be more fault-prone;
- **H-DIT:** A class located deeper in a class inheritance lattice is supposed to be more fault-prone because the class inherits a large number of definitions from its ancestors

Hypotheses

- **H-NOC**: Classes with a large number of children are difficult to modify and usually require more testing because the class potentially affects all of its children.
- **H-CBO:** Highly coupled classes are more fault-prone than weakly coupled classes because they depend more heavily on methods and objects defined in other classes.

Hypotheses

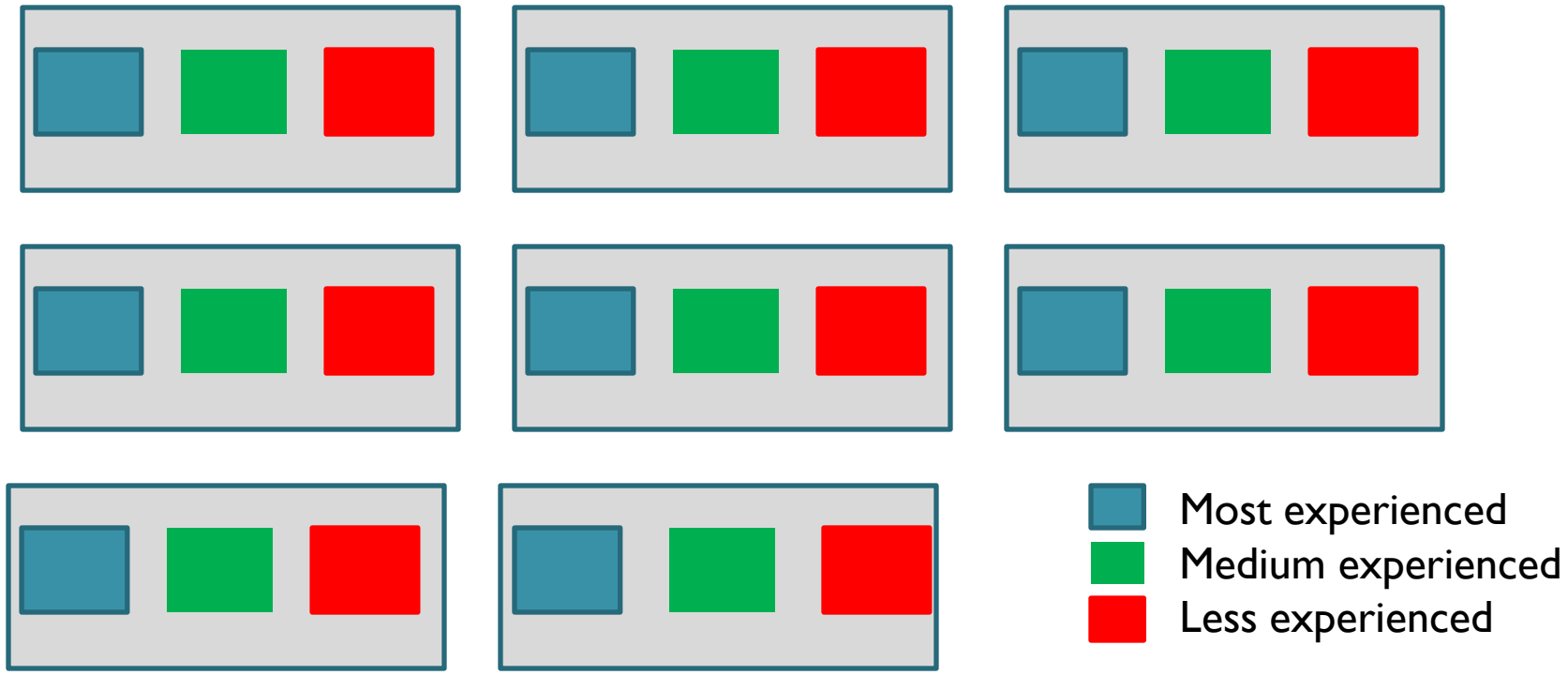
- **H-RFC**: Classes with a larger response sets implement more complex functionalities and are, therefore, more fault-prone.
- **H-LCOM**: Classes with low cohesion among its methods suggests an inappropriate design which is likely to be more fault-prone.

Empirical Study

- A Empirical study was performed for more than four months;
- Participants were the students (undergraduated/graduated) of University of Maryland.
- All students had some experience with C or C++ programming.
- The students were not required to have previous experience or training in the application domain or OO methods.

Study participants

- Questionnaires and performed interviews
- Students were assigned to a different group.



The Development Process

- Each team was asked to develop a medium-sized management information system that supports the rental / return process of a hypothetical **video rental business**, and maintains customer and video databases.
- The development process was performed according to a sequential software engineering life-cycle model
- Test: spending eight hours testing each system.

The Development Process

- Libraries provided to the students:
  - *MotvApp.*
  - *GNU library.*
  - C++ *database library.*
  - OSF / Motif
- A hundred small programs exemplifying how to use OSF/Motif widgets were also provided

Data Collection Procedures and Measurement Instruments

1. the source code of the C++ programs delivered at the end of the implementation phase.
2. data about these programs,
3. data about errors found during the testing phase and fixes during the repair phase
4. the repaired source code of the C++ programs delivered at the end of the life cycle

Extract metrics (GEN++) extract

Data Collection Procedures and Measurement Instruments

Data Collection Form:

- Total 180 classes across the eight systems

A fault report form was used to gather data about

1. the faults found during the testing phase,
2. classes changed to correct such faults, and
3. *the effort in correcting them*

Data Analysis

UNIVARIATE ANALYSIS–SUMMARY OF RESULTS

Metrics	Coefficient	Δψ	p-value	R^2	Classes
WMC (1)	0.022	2%	0.0607	0.007	ALL
WMC (2)	0.086	9%	0.0003	0.024	New-Ext
WMC (3)	0.027	3%	0.0656	0.015	DB
WMC (4)	0.094	10%	0.0019	0.047	UI
DIT (1)	0.485	62%	0.0000	0.065	ALL
DIT (2)	0.868	138%	0.0000	0.131	New-Ext
DIT (3)	0.475	60%	0.043	0.019	DB
DIT (4)	0.29	34%	0.024	0.017	UI
RFC (1)	0.085	9%	0.0000	0.065	ALL
RFC (2)	0.087	8%	0.0000	0.248	New-Ext
RFC (3)	0.077	8%	0.0000	0.188	DB
RFC (4)	0.108	11%	0.0000	0.362	UI
NOC (1)	–3.3848	–96%	0.0000	0.143	ALL
NOC (2)	–3.62	–97%	0.0011	0.362	New-Ext
NOC (3)	–2.05	–77%	0.0000	0.083	DB
CBO (1)	0.142	15%	0.0000	0.068	ALL
CBO (2)	0.079	8%	0.017	0.020	New-Ext
CBO (3)	0.086	9%	0.006	0.034	DB
CBO (4)	0.284	33%	0.0000	0.170	UI

Analysis Methodology

$$\pi(X_1, X_2, \ldots, X_n) = \frac{e^{(C_0 + C_1 \bullet X_{i1} + \ldots + C_n \bullet X_{in}) \bullet Y_i}}{1 + e^{(C_0 + C_1 \bullet X_{i1} + \ldots + C_n \bullet X_{in})}}$$

- π : probability that a fault was found in a class during the validation phase.
- X: are the design metrics included as explanatory variables in the mode.
- C: will be estimated through the maximization
- of a likelihood function, built in the usual fashion

Analysis Methodology

$$R^2 = \frac{LL_S - LL}{LL_S}$$

$$LL_S = m_0 \ln\left(\frac{m_0}{m_0 + m_1}\right) + m_1 \ln\left(\frac{m_1}{m_0 + m_1}\right)$$

- **LL**: is the log likelihood obtained by Maximum Likelihood
- **LLs**: is the log likelihood obtained by Maximum Likelihood Estimation of a model **without any variables**

Analysis Methodology

$$\Delta\psi = \frac{\psi(X+1)}{\psi(X)}$$

- **ψΔ:** represents the ratio between the probability of having a fault and the probability of not having a fault when the value of the metric is X

Univariate Analysis

- **H-WMC**: hypothesis is supported (The larger the WMC, the larger the probability of fault detection).
  - Overall coefficient = 0.06.
  - New-Ext: 0.0003
  - UI: 0.0019

Univariate Analysis

- **DIT**: hypothesis is supported. The larger the DIT, the larger the probability of fault detection, the strength of the relationship increases (R^2 goes from 0.06 to 0.13)

Univariate Analysis

- **RFC**: hypothesis is supported. The larger the RFC, the larger the probability of fault detection. Again, R2 improved significantly for new and extensively modified classes and UI classes.
  - ALL : 0.06
  - New-Ext: 0.24
  - UI: 0.0036

Univariate Analysis

- **NOC**: hypothesis is not supported. The larger the NOC, the lower the probability of fault detection.
  - ALL : 0.06
  - New-Ext: 0.24
  - UI: 0.0036

Univariate Analysis

- **LCOM**: was shown to be insignificant in all cases.

Univariate Analysis

- **CBO**: No satisfactory explanation could be found for differences in pattern between UI and DB classes

Multivariate Analysis

MULTIVARIATE ANALYSIS WITH OO DESIGN METRICS

	Coefficient	p-value
Intercept	3.13	0.0000
DIT	0.50	0.0004
RFC	0.11	0.0000
NOC	–2.01	0.0178
CBO	0.13	0.0072
Class Origin	1.84	0.0000

Multivariate Analysis

CLASSIFICATION RESULTS WITH OO DESIGN METRICS

Actual	Predicted	
	No Fault	Fault
No Fault	90	32
Fault	10 (18)	48 (250)

- When $\pi > 0.5$, *the class is classified as faulty and, otherwise, as* no faulty

Conclusions

- Five out of the six Chidamber and Kemerer's OO metrics appear to be useful to predict class fault-proneness during the high- and low-level design phases of the life-cycle.