Practical Object-Oriented Measurement

Khaled El Emam

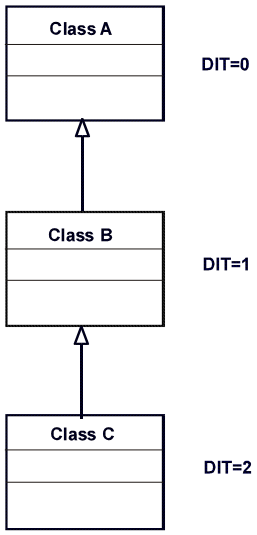
Agenda

- Definition of Object-Oriented Measures
- Utility of Object-Oriented Measures
  - Preventative action
  - Quality prediction
  - Design guidelines
- Methodological Considerations
  - Confounding effects
  - Optimal class size & faults
  - ROI calculations
- Conclusions

Definition

- Object-oriented measures quantitatively characterize the structure of a software system
- We are only concerned with static measures that can be collected from design documents or source code
- We are only concerned with class-level measures (as opposed to, for example, method-level measures)
- Many OO measures have been developed:
  - size (measure how big a class is)
  - inheritance (characterize inheritance hierarchy)
  - coupling (characterize relations amongst classes)
  - cohesion (characterize relations within classes)
  - complexity (aggregates of method measures)

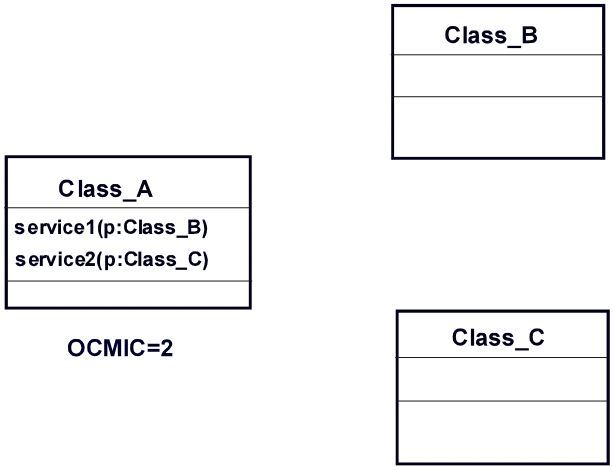
Example of Inheritance Measure: DIT



Example of a Coupling Measure: Class-Method Import Coupling

Contents

- Definitions
- Utility
- Confounding
- Optimal Size
- ROI
- Closing

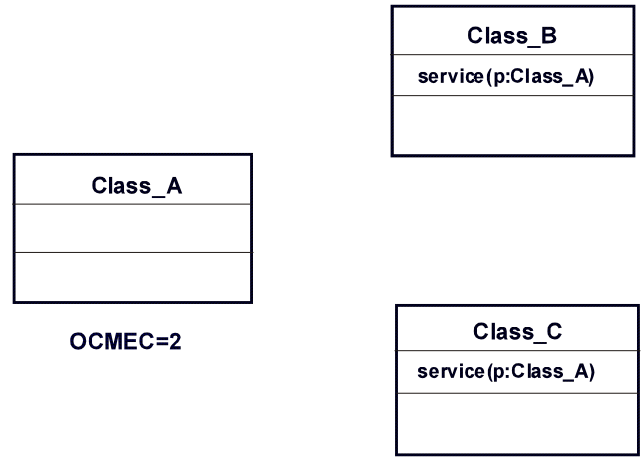


OCMIC=2

Example of a Coupling Measure: Class-Method Export Coupling

Contents

- Definitions
- Utility
- Confounding
- Optimal Size
- ROI
- Closing

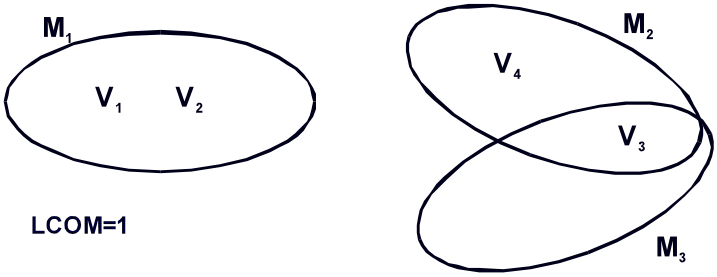


OCMEC=2

Example of a Cohesion Measure: Lack of Cohesion

Contents

- Definitions
- Utility
- Confounding
- Optimal Size
- ROI
- Closing



Most Useful OO Measures

Contents

- Definitions
- Utility
- Confounding
- Optimal Size
- ROI
- Closing

Name	Description
Size	This can be the number of methods in a class or Lines of Code, depending on whether a design or source code are being evaluated.
DIT	Depth of Inheritance Tree indicates how deep in the inheritance hierarchy a class is.
OCMEC	Class-method export coupling to other classes indicates how many other non-descendant classes use a class type in their method signature.
OCMIC	Class-method import coupling to other classes indicates how many other class types are used in a class' method signatures.
ACMIC	Class-method import coupling to ancestor classes indicates how many of a class' method signatures use ancestor types.
DCMEC	Class-method export coupling to descendant classes indicates how many descendents use a class' type in their method signature.

Utility of Object-Oriented Measures

- Object-oriented measures can potentially be used for:
  - Taking preventative action
  - Quality (and cost) estimation
  - Developing design guidelines
- You can get different results depending on which criterion you are talking about:
  - pre-release faults
  - post-release faults
  - development & maintenance effort

Preventative Action - I

- There exists evidence showing that most faults in a software system are detected in a small percentage of its components
- To improve defect detection effectiveness and efficiency it is desirable to identify these components early on
- Then one can take mitigating actions, such as more inspections, targeted testing, or even a redesign

Preventative Action - II

Quality Models

- Quantitative models are constructed to predict which components are likely to have a fault

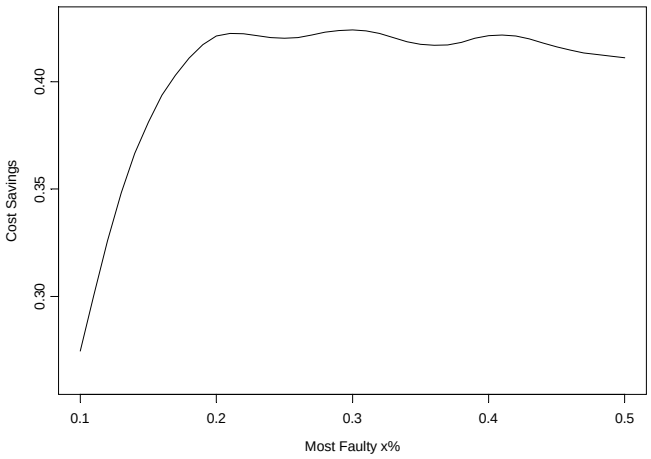
Preventative Action - Example (1)

- A commercial Java application
- A quality model was constructed to predict the probability of a post-release fault
- Quality model was used to identify the classes that should be inspected
- Evaluated the cost savings from using the quality model (we had data on the cost of inspections and dealing with post-release faults)

Preventative Action - Example (2)

Contents

- Definitions
- Utility
- Confounding
- Optimal Size
- ROI
- Closing



Quality Prediction

Contents

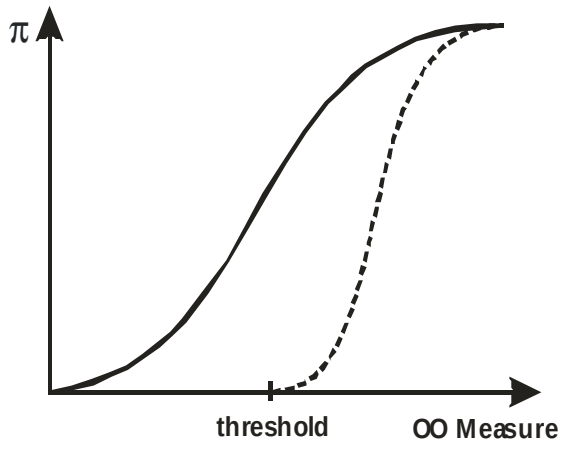
- Definitions
- Utility
- Confounding
- Optimal Size
- ROI
- Closing

- Developed a quality model on one release of a commercial Java application
- Used that model to predict the proportion of classes in the next release that will have a fault in them
- The prediction has approx. 9% error (i.e., underestimated the proportion by 9%)
- This can be considered a good prediction accuracy

Design Guidelines - I

- It is also possible to empirically identify which structural properties of the software cause problems (e.g., which types of coupling)
- This information can then be used to construct proscriptive design guidelines to minimize problems in future development efforts
- Guidelines can be enforced automatically or through inspections
- Design guidelines expressed in the form of thresholds on the measures
- If rule is not satisfied then there is a greater risk of a fault

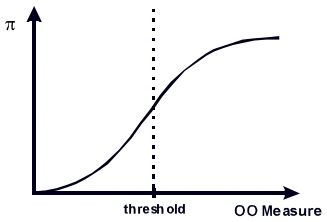
Design Guidelines - II

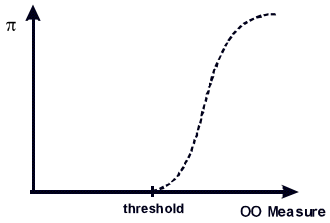


Design Guidelines - III

- Thus far, no evidence that thresholds exist
- We could not identify a specific value on any of the object-oriented measures where the probability of a fault suddenly increases
- Does this mean that thresholds are useless ?

Design Guidelines - IV





Methodological Considerations

Contents

- Confounding effects
- Optimal class size
- ROI calculations

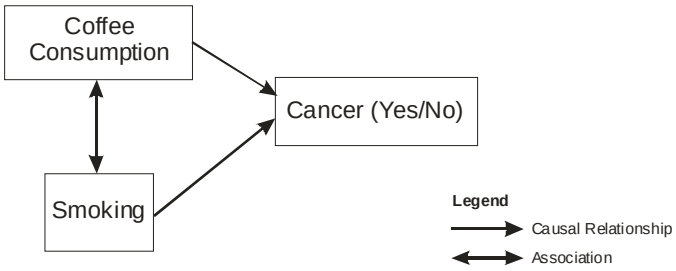
Common Validation Method

Contents

- You collect data on the object-oriented measure and on whether a fault occurs in a class
- If there is a positive relationship between the measure and fault-proneness, then the measure is said to be validated
- At least 12 “successful” validation studies

An Epidemiological Example

Contents
- Definitions
- Utility
- Confounding
- Optimal Size
- ROI
- Closing



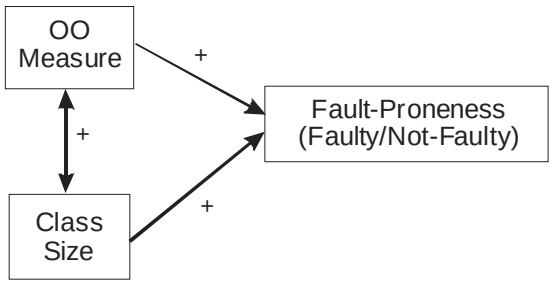
- If you look only at the coffee consumption → cancer relationship, you can get very misleading results
- Smoking is a confounder

Confounding Effects

Contents
- Definitions
- Utility
- Confounding
- Optimal Size
- ROI
- Closing

- It is known that smoking and coffee consumption are associated
- It is known that smoking causes cancer
- If smoking is not controlled during the study, then it is likely that an inflated coffee effect will be found
- This is a classic confounding effect
- Without proper controls, the study results can show a relationship where none really exists

Confounding with Object-Oriented Measures



- In this case, size is the confounder
- There is substantial evidence that most object-oriented measures are associated with class size

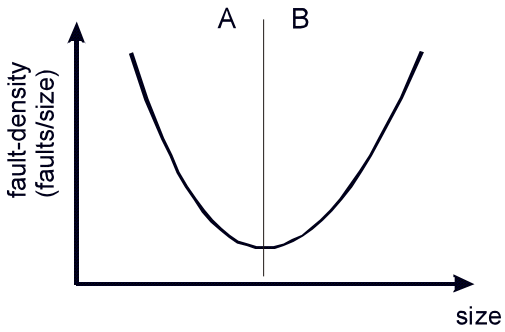
Example Relationship with Size

OO Metric	*LOC*	
	Rho	**p-value**
WMC	0.88	<0.0001
DIT	0.098	0.19
CBO	0.46	<0.0001
RFC	0.88	<0.0001
LCOM	0.24	0.0011
NMA	0.86	<0.0001
NPAVG	0.27	0.000256

The Confounding Effect of Size

- Just looking at the relationship between an object-oriented measure and faults does not make sense
- It is necessary to at least control for size

Optimal Class Size & Faults - I

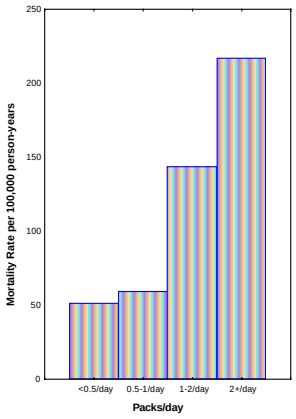


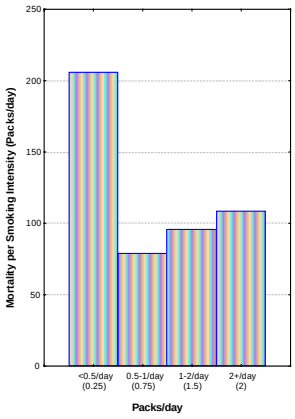
- Smaller classes (or modules) are more faulty than larger ones (program decomposition is bad)
- There is an optimal class size

Optimal Class Size & Faults - II

- The first conclusion is due to a mathematical artifact of plotting Y/X vs. X
- You will always get a negative association, even if there is no association between the raw variables
- It is easy to obtain absurd conclusions (turn a positive association to a negative one) this way
- Some examples ...

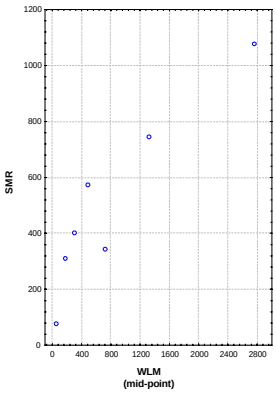
Smoking and Mortality Rate

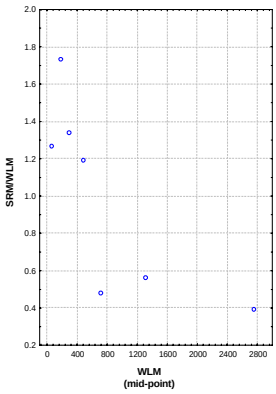




Optimal smoking: 0.75 packs/day

Uranium Exposure & Mortality





Quickly, move closer to a uranium mine

Calculating Return on Investment (ROI) - I

Principles

- Defect detection and correction in later phases of a project are much more expensive (some quoted numbers are 100:1 or even 200:1 for post-release defects)
- Effort spent now to detect and correct defects will lead to savings later on (so you invest now to save money later on)
- We can estimate how much is saved by finding and correcting defects now
- ROI uses that estimate as well as the cost of early defect detection and correction

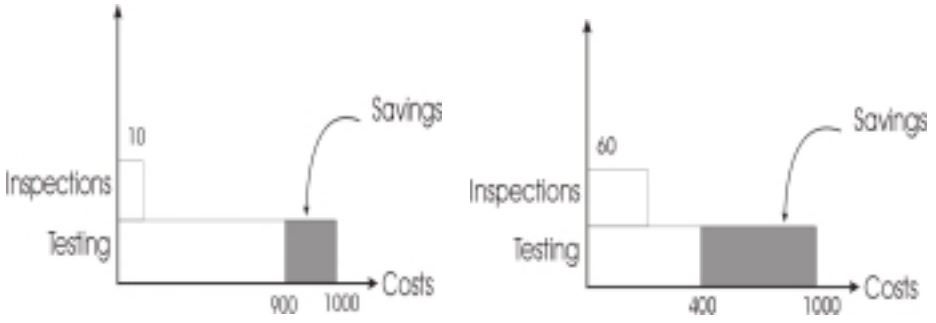
Calculating Return on Investment (ROI) - II

Basic Calculation

$$ROI = \frac{\text{cost saved}}{\text{cost consumed}}$$

Calculating Return on Investment (ROI) - III

Problems with Basic ROI

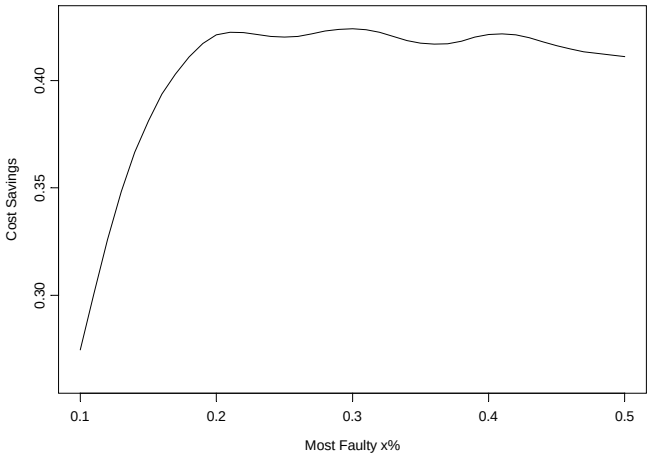


Calculating Return on Investment (ROI) - IV

Improved Calculation

$$ROI = \frac{\text{cost saved}}{\text{virtual testing cost}}$$

Calculating Return on Investment (ROI) - V



Conclusions

- Object-oriented measures are a powerful tool for detecting high risk classes early on in projects
- We have applied them in real projects and the estimated cost savings (when action was taken) were dramatic
- You will find a tool to collect product measures for C++ and Java on my web site:
  - www.seg.iit.nrc.ca/~elemam
- Improvements in quality management using product measurement can be attained trough:
  - further development of cognitive theories for the OO measures
  - additional design measures
  - linking the static measures with dynamic ones