

Models and Measures (Product)

I Part II: OO MEASURES

I Object-Oriented Software Measurement

- High level design
- Low level design
- Coding
- Testing

Object-Oriented Software Measurement

I Models and Metrics

I Some definitions

Coupling

Intuitively, it refers to the degree of interdependence between parts of a design

Since objects of the same class have the same properties, two classes are coupled when methods declared in one class use methods or instance variables of the other class.

Cohesion

Intuitively, it refers to the internal consistency within parts of the design

The degree of similarity of methods can be viewed as a major aspect of object class cohesiveness. If an object class has different methods performing different operations on the same set of instance variables, the class is cohesive.

Object-Oriented Software Measurement

- I Models and Metrics
- I Size (requirements description, high and low level design)
 - Lorenz and Kidd [Lorenz94]**
 - I Number of scenarios (use-cases) scripts (NSS)
 - directly correlated to the size of the application and to the number of test cases
 - I Number of key classes (domain classes) (NKC)
 - concerned with high level design and the amount of effort necessary to implement the system and the reuse necessary
 - I Number of support classes (NSC)
 - concerned with low level design and the amount of effort necessary to implement the system and the reuse necessary
 - I Average number of support classes per key class (ANSC)
 - I Number of subsystems (NSUB)

Object-Oriented Software Measurement

- I Models and Metrics
- I Size (requirements description, high and low level design)
 - Lorenz and Kidd [Lorenz94]**
 - I Class Size (CS)
 - total number of operations + number the attributes (both including inherited features)
 - I Number of Operations Overridden by a Subclass (NOO)
 - I Number of Operations Added by a subclass (NOA)
 - I Specialization Index (SI)
 - $SI = [NOO \times level] / (Total\ Class\ methods)$

Object-Oriented Software Measurement

- I Models and Metrics
- I Size (requirements description, high and low level design)

Lorenz and Kidd [Lorenz94]

Phase Metric	Requirements Description	High Level Design	Low Level Design	Coding	Testing
NSS					
NKC					
NSC					
ANSC					
NSUB					
CS					
NOO					
NOA					
SI					

Object-Oriented Software Measurement

- I Models and Metrics

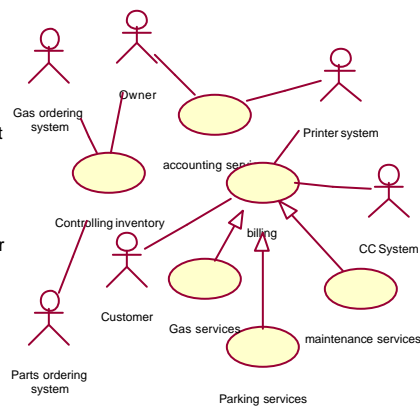
The Gas Station problem

Requirements:

1 – A customer has the option to be billed automatically at the time of purchase (gas, car maintenance or parking spots) or to be sent a monthly paper bill. Customers can pay via cash, credit card or personal check. Gas Station services have a fixed price (gas: US\$ 1.09 gallon, car maintenance: US\$ 150.00 and parking spot: US\$ 5.00 per day). The tax is 5% added to the final price of the purchase. Sometimes, the Gas Station owner can define discounts to those prices.

2 – The system has to track bills on a month-to-month basis and the services performed by the gas station on a day-to-day basis. The results of this tracking can be reported to the owner upon request.

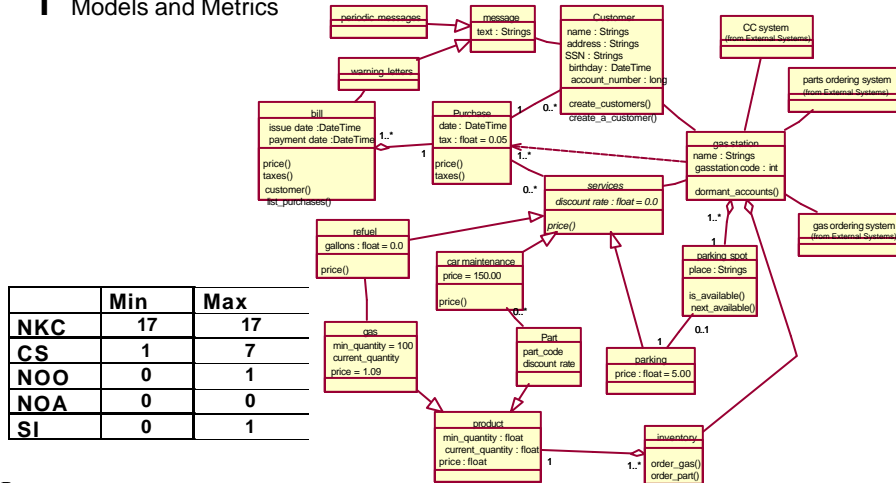
3 – The gas station owner can use the system to control inventory. The system will either warn of low inventory or automatically order new parts and gas.



NSS = 6

Object-Oriented Software Measurement

I Models and Metrics



Object-Oriented Software Measurement

I Models and Metrics

I Product (high and low level design)

Chidamber and Kemerer Metrics Suite [Chidamber94]

- I Weighted Methods per Class (WMC)
- I Depth of Inheritance (DIT)
- I Number of children (NOC)
- I Coupling Between Objects (CBO)
- I Response for a class (RFC)
- I Lack of Cohesion in Methods (LCOM)

Phase Metric	High Level Design	Low Level Design	Codina	Testina
WMC				
DIT				
NOC				
CBO				
RFC				
LCOM				

Object-Oriented Software Measurement

I Models and Metrics

I Product

I **Weighted Methods per Class (WMC)**

Consider a Class C_i , with methods M_1, \dots, M_n that are defined in the class. Let c_1, \dots, c_n be the complexity of the methods. Then:

$$WMC = \sum_{i=1}^n c_i$$

If all method complexities are considered to be unity, then $WMC = n$, the number of methods.

Viewpoints:

- 1) number of methods and the complexity of methods correlated with how much time and effort is required to develop and maintain the class
- 2) the larger the number of methods in a class the greater the potential impact on children, since children
- 3) Classes with large numbers of methods are likely to be more application specific, limiting the possibility of reuse



UFRJ/COPPE
CAPES - Brazil



UMCP - CS
ESEG

Object-Oriented Software Measurement

I Models and Metrics

I Product

I **Depth of Inheritance (DIT)**

Depth of inheritance of the class is the DIT metric for the class. In cases involving multiple inheritance, the DIT will be the maximum length from node to the root of the tree.

Viewpoints:

- 1) The deeper a class is in the hierarchy, the greater the number of methods it is likely to inherit, making it more complex to predict its behavior
- 2) Deeper trees constitute greater design complexity, since more methods and classes are involved
- 3) The deeper a particular class is in the hierarchy, the greater the potential reuse of inherited methods



UFRJ/COPPE
CAPES - Brazil



UMCP - CS
ESEG

Object-Oriented Software Measurement

I Models and Metrics

I Product

I **Number of children (NOC)**

NOC = number of immediate subclasses subordinated to a class in the class hierarchy

Viewpoints:

- 1) Greater the number of children, greater the reuse, since inheritance is a form of reuse
- 2) Greater the number of children, the greater the likelihood of improper abstraction of the parent class. If a class has a large number of children, it may be a case of misuse of subclassing
- 3) The number of children gives an idea of the potential influence a class has on the design. If a class has a large number of children, it may require more testing of the methods in that class.

Object-Oriented Software Measurement

I Models and Metrics

I Product

I **Coupling Between Objects (CBO)**

CBO for a class is a count of the number of other classes to which is coupled

Viewpoints:

- 1) excessive coupling between object classes is detrimental to modular design and prevents reuse. The more independent a class is, the easier it is to reuse it in another application
- 2) In order to improve modularity and promote encapsulation, inter-object class couples should be kept to a minimum. The larger the number of couples, the higher the sensitivity to changes in other parts of the design, and therefore maintenance is more difficult
- 3) A measure of coupling is useful to determine how complex the testing of various parts of a design are likely to be. The higher the inter-object class coupling, the more rigorous the testing needs to be.

Object-Oriented Software Measurement

I Models and Metrics

I Product

I Response for a class (RFC)

The response set of a class is a set of methods that can potentially be executed in response to a message received by an object of that class

RFC = |RS| where RS is the response for the class.

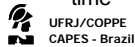
RS = {M} Uall I {Ri}

where {Ri} = set of methods called by method I and

{M} = st of all methods in the class

Viewpoints:

- 1) If a large number of methods can be invoked in response to a message, the testing and debugging of the class becomes more complicated since it requires a greater level of understanding required on the part of the tester
- 2) the larger the number of methods that can be invoked from a class, the greater the complexity of the class
- 3) A worst case value for possible responses will assist in appropriate allocation of testing time



UFRJ/COPPE
CAPES - Brazil



UMCP - CS
ESEG

Object-Oriented Software Measurement

I Models and Metrics

I Product

I Lack of Cohesion in Methods (LCOM)

Consider a class C1 with n methods M_1, M_2, \dots, M_n . Let $\{i\}$ = set of instance variables used by method M_i .

There are n such sets $\{i_1\}, \dots, \{i_n\}$. Let $P = \{(i, j) \mid i \cap j = \emptyset\}$

and $Q = \{(i, j) \mid i \cap j \neq \emptyset\}$. If all n sets $\{i_1\}, \dots, \{i_n\}$ are 0 then let $P = 0$.

$LCOM = |P| - |Q|$, if $|P| > |Q|$

$LCOM = 0$ otherwise

Viewpoints:

- 1) **Cohesiveness of methods within a class is desirable, since it promotes encapsulation**
- 2) **Lack of cohesion implies classes should be probably be split into two or more subclasses**
- 3) **Any measure of disparateness of methods helps identify flaws in the design of classes**
- 4) **Low cohesion increases complexity, thereby increasing the likelihood of errors during the development process.**



UFRJ/COPPE
CAPES - Brazil



UMCP - CS
ESEG

Object-Oriented Software Measurement

I Models and Metrics

I Product

Lie and Henry [Lie93]

- I metrics validated by studying the number of changes performed in two commercial systems implemented with an OO dialect of Ada
- I extends Chidamber&Kemerer's OO metrics with two additional metrics:
 - Message Passing Coupling (MPC): is calculated as the number of send statements defined in a class
 - Data Abstraction Coupling (DAC): is calculated as the number of abstract data types used in the measured class and defined in another class of system
- I model was adequate in predicting the size of changes in classes during the maintenance phase

Object-Oriented Software Measurement

I Models and Metrics

Basili, Briand and Melo [Basili95]

- I Based on CK suite, empirically validate the suitability of such metrics for predicting the probability of detecting faulty classes for C++ based software
- I Slightly adjusted some of the metrics to capture C++ flavors (the metrics are not language independent)

WMC: all methods have complexity 1 and "friend" operators are not count

DIT: measures the number of ancestors of a class

NOC: number of direct descendants for each class

LCOM: number of pairs of member functions without shared instance variables, minus the number of pairs of member functions with shared instance variables. It is set to 0 whenever the subtraction is negative

CBO: a class is coupled to another on if it uses its member functions and/or instance variables

RFC: number of functions directly invoked by member functions or operators of a class

Object-Oriented Software Measurement

I Models and Metrics

Basili, Briand and Melo Viewpoints:

WMC: a class with significantly more member functions than its peers is more complex, and by consequence tends to be more fault-prone

DIT: well-designed OO systems are those structured as forests of classes, rather than as one very large inheritance lattice

LCOM: a class with low cohesion among its methods suggests an inappropriate design, (i.e., the encapsulation of unrelated program objects and member functions that should not be together), which is likely to be fault-prone

NOC: classes with large number of children are difficult to modify and usually require more testing because the class potentially affects all of its children

CBO: highly coupled classes are more fault-prone than weakly coupled classes

RFC: larger the response set of a class, the higher the complexity of the class, and more fault-prone and difficulty to modify



UFRJ/COPPE
CAPES - Brazil



UMCP - CS
ESEG

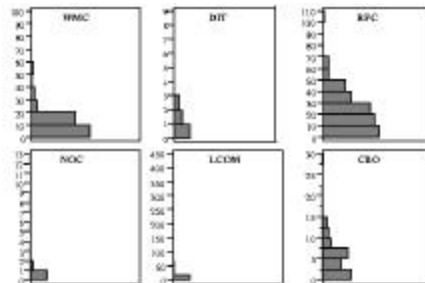
Object-Oriented Software Measurement

I Models and Metrics

Analysis of Distributions:

OO metrics based on 180 classes:

DIT indicates that inheritance hierarchies are somehow flat and **NOC** that classes have, in general, few children. Classes seemed to have high cohesion (LCOM near 0)



	WMC	DIT	RFC	NOC	LCOM	CBO
Maximum	99.00	9.00	105.00	13.00	426.00	30.00
Minimum	1.00	0.00	0.00	0.00	0.00	0.00
Median	9.50	0.00	19.50	0.00	0.00	5.00
Mean	13.40	1.32	33.91	0.23	9.70	6.80
Std Dev	14.90	1.99	33.37	1.54	63.77	7.56



UFRJ/COPPE
CAPES - Brazil



UMCP - CS
ESEG

Object-Oriented Software Measurement

I Models and Metrics

results regarding to the probability of fault detection in a class during test phases

WMC: the **larger** WMC, the **larger** the probability of fault detection

DIT: the **larger** the DIT, the **larger** the probability of defect detection

NOC: the **larger** the NOC, the **lower** the probability of defect detection

RFC: the **larger** the RFC, the **larger** the probability of defect detection

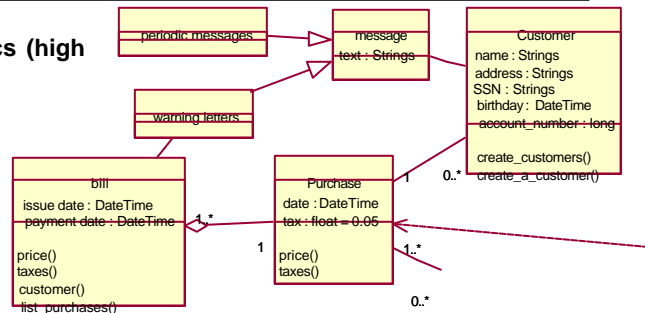
LCOM: no significant

CBO: more significant for UI classes than for DB classes (no satisfactory explanation found for differences in pattern between UI and DB classes)



Object-Oriented Software Measurement

Models and Metrics (high level)

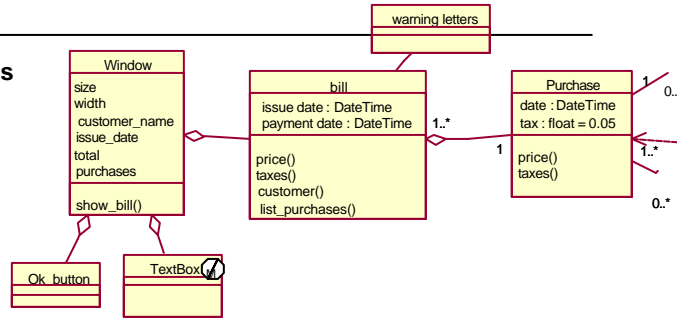


	Bill	Purchase	Warning Letters	Periodic Messages	Message	Customer
WMC	4	2	0	0	0	2
NOC	0	0	0	0	2	0
DIT	0	0	1	1	0	0
RFC	--	--	--	--	--	--
CBO	2	4	2	1	1	2
LCOM	--	--	--	--	--	--



Object-Oriented Software Measurement

Models and Metrics (low level)



	Bill	Purchase	Warning Letters	Window	Ok button	Textbox
WMC	4	2	0	1	0	0
NOC	0	0	0	0	0	0
DIT	0	0	1	1	0	0
RFC	--	--	--	--	--	--
CBO	3	4	2	3	1	1
LCOM	--	--	--	--	--	--

Object-Oriented Software Measurement

I Models and Metrics

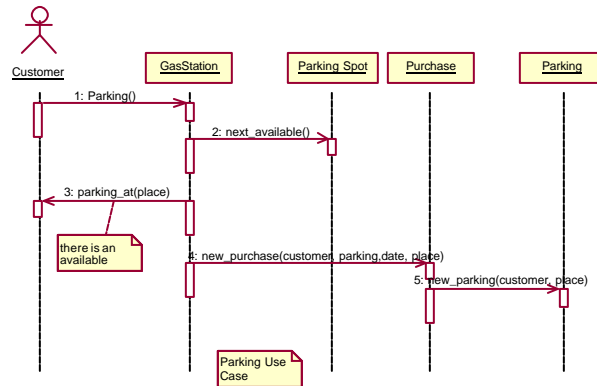
Some other possible metrics

- I Average operation Size (OS_{avg})
 - the number of messages sent by the operation
- I Average number of parameters per operation (NP_{avg})
- I Operation Complexity (OC)
 - can be computed using any of the complexity metrics proposed for conventional software
- I Percent public and protected (PAP)
- I Public access to data members (PAD)
- I Number of root classes (NOR)
 - shows the number of class hierarchies that exist in the system. Large values should be avoid
- I Fan in (FIN)
 - $FIN > 1$ indicates multiple inheritance

Object-Oriented Software Measurement

I Models and Metrics

	Min	Max
OS	5	5
NP	0	4



Object-Oriented Software Measurement

Class name: refuel

Category: Service

External Documents:

Export Control: Public

Cardinality: n

Hierarchy:

Superclasses: services

Associations:

<no rolename> : gas in association <unnamed>

Operation name: price

Public member of: refuel

Documentation:

// Calculates the final price of the fuel

Preconditions:

gallons > 0

Object diagram: (Unspecified)

Semantics:

final_price = gallons * price

Object diagram: (Unspecified)

Concurrency: Sequential

	Value
CS	4
PAD	0
PAP	0

Public Interface:

Operations:

price

Private Interface:

Attributes:

gallons

price = 1.09

Implementation:

Attributes:

gallons

price = 1.09

State machine: No

Concurrency: Sequential

Persistence: Transient

Object-Oriented Software Measurement

Model x Metrics:
where to measure/visualize OO metrics

Model Metric	Use Cases	Class Diagrams	Interaction Diagrams	Class Descriptions	State Diagrams	Package Diagrams	Deployment Diagrams
NSS							
NKC							
NSC							
ANSC							
NSUB							
CS							
NOO							
NOA							
SI							

Object-Oriented Software Measurement

Model x Metrics:
where to measure/visualize OO metrics

Model Metric	Use Cases	Class Diagrams	Interaction Diagrams	Class Descriptions	State Diagrams	Package Diagrams	Deployment Diagrams
WMC							
DIT							
NOC							
CBO							
RFC							
LCOM							
OS _{avg}							
NP _{avg}							
OC							
PAP							
PAD							
NOR							
FIN							

Object-Oriented Software Measurement

- I How Developers use OO Metrics:
 - I evaluating software quality
 - I understanding design process
 - I identifying product problems
 - I improving solutions
 - I acquiring design knowledge

- I For instance
 - I using metrics to evaluate OO structural complexity
 - approach applied to two different application domains:
 - Telecomm software (C++)
 - Software Engineering Environments (Eiffel)

Object-Oriented Software Measurement

- I Using metrics to evaluate OO structural complexity [Travassos99]
- I Developers can combine:
 - Heuristics applied to reduce software complexity during the design activities (*Guidelines to reduce the O-O Design Complexity*)
 - Knowledge about development experiences (*O-O Design Principles*)
 - Metrics to characterize how the solutions were designed.

G1	Classes Restructuring Guideline (set of 10 procedures)
G2	Adapter Class Guideline
G3	Bridge Class Guideline
G4	Facade Class Guideline
G5	Multiple Inheritance Elimination Guideline
G6	Large Hierarchies Redefinition Guideline

Heuristics

M1	DIT
M2	NOC
M4	Coupling
M5	Class Size
M6	Number of Multiple Inheritance
M11	Number of Abstract Classes

Metrics

Object-Oriented Software Measurement

I Using metrics to evaluate OO structural complexity [Travassos99]

Principles

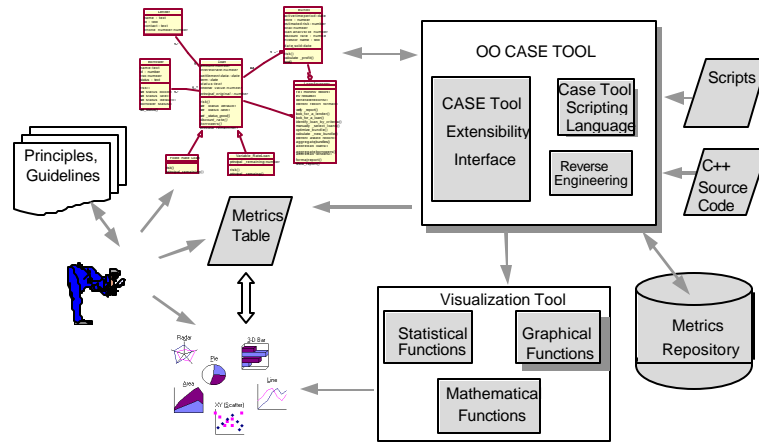
- P1 Few Interfaces Principle : "All module (class) should communicate with as few others as possible".
- P2 Small Interfaces Principle: "If any two modules (classes) communicate at all, they should exchange as little information as possible".
- P3 Explicit Interfaces Principle: "Whenever two modules (classes) *A* and *B* communicate, this must be obvious from the text of *A* or *B* or both".
- P4 Information Hiding Principle: "All information about a module (class) should be private to the module (class) unless it is specifically declared public".
- P5 The Open-Closed Principle: "Software entities (classes, modules, etc.) should be open for extension but closed for modification".
- P6 The Liskov Substitution Principle: "It is only when derived types are completely substitutable for their base types that functions with use those base types can be reused with impunity, and the derived types can be changed with impunity".
- P7 The Dependency Inversion Principle: "Details should depend on abstractions. Abstractions should not depend on details. High level policies should not depend on low-level implementations; rather both should depend on abstractions".
- P8 The Interface Segregation Principle: "Clients should not be forced to depend on interfaces that they do not use".

Object-Oriented Software Measurement

	P1	P2	P3	P4	P5	P6	P7	P8
G1	M3, M7, M9, M10, M12	M8	M14	M14	M1, M2, M3, M6, M6, M11	M1, M2,	M1, M2, M11	M2, M4, M5
G2		M13						M5
G3					M11			
G4	M3				M3			
G5	M3				M3, M6			
G6	M3				M1, M2, M3	M1, M2,	M1, M2,	M4

Guidelines, Principles and Metrics linkage

Object-Oriented Software Measurement



A set of tools exploring the approach

Object-Oriented Software Measurement

** Legend: A: Attribute, B: Behavior,
 ** IA: Inherited Attribute, IB: Inherited Behavior,
 ** TA: Total Attributes, TB: Total Behavior,
 ** DIT: Hierarchical Level, NOC: Number of Children

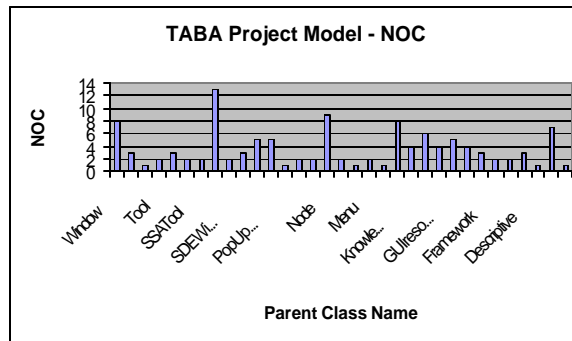
** Normal NOC for this model.....: 3
 ** Number of leaf classes.....: 130
 ** Number of Non-Hierarchical Classes...: 11
 ** Number of Abstract Classes.....: 33
 ** Number of Classes.....: 163

A metric table produced with the case tools

Parent Class Name	A	B	IA	IB	AT	BT	DIT	NOC
Shapes	1	2	0	0	1	2	0	13
Node	2	0	3	4	5	4	1	9
Window	7	0	1	0	8	0	1	8
KnowledgeWindow	0	2	8	0	8	2	2	8
Arcs	2	1	3	4	5	5	1	7
Knowledge	6	4	0	0	6	4	0	6
PullDownMenu	8	9	4	0	12	9	1	5
GUIresoruces	1	0	0	0	1	0	0	5
SDEFramework	0	1	1	0	1	1	2	5
InternalTool	1	1	0	0	1	1	0	4
KnowledgeREpresenta tion	0	0	8	0	8	0	2	4
GeneralWindows	6	2	15	10	21	12	1	4
Editor	1	0	0	0	1	0	0	3
ToolWindow	0	0	1	1	1	1	3	3
GeneralMenus	0	0	8	0	8	0	2	3
SDEWindow	0	0	21	12	21	12	2	3

Object-Oriented Software Measurement

Visualizing the Metrics



Object-Oriented Design

Bibliography

- Fowler97. Martin Fowler and Kendall Scott, UML Distilled: Applying the standard object modeling language, Addison-Wesley, 1997
- Booch94. Grady Booch, Object-Oriented Analysis and Design with Applications, The Benjamin/Cummings Publishing Company, Inc, second edition, 1994
- Lorenz94. Lorenz, M., and J. Kidd, Object-Oriented Software Metrics, Prentice-Hall, 1994
- Chidamber94. Chidamber, S.R. and Kemerer, C. F.; A Metrics Suite for Object Oriented Design, IEEE Transactions on Software Engineering, vol.20, no6, June 1994.
- Basili95. Basili, Victor R.; Briand, Lionel and Melo, Walcelio; A validation of Object-Oriented design metrics, Technical Report, Univ. of Maryland, Dep. Of Computer Science, CS-TR-3443, May, 1995
- Lie93. W. Lie and S. Henry (1993). Object-oriented metrics that predict maintainability, Journal of Systems and Software. 23(2):111-122.
- Travassos99. G. H. Travassos, R. S. Andrade, Combining Metrics, Principles and Guidelines for Object Oriented Design, Workshop on Quantitative Approaches on Object Oriented Software Engineering, ECOOP'99, Lisbon, Portugal, 1999
- Pressman97. R. S. Pressman, Software Engineering: A Practitioner's Approach, 4th edition, McGraw Hill, 1997