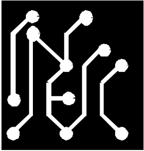


ECOOP'95
Quantitative Methods Workshop
Aarhus, August 1995

Design Metrics for Object-Oriented Software Systems

Fernando Brito e Abreu
INESC / ISEG
PORTUGAL

PRESENTATION OUTLINE

This presentation proposes:

☞ The experimental validation of a **set of metrics** suitable for evaluating the use of the mechanisms that support the main concepts of the Object-Oriented paradigm and the consequent emphasis on reuse, that are believed to be responsible for the increase in software quality and development productivity.

QUALITY: TWO SCHOOLS OF THOUGHT

Software quality can be characterized by the presence of several **external attributes. However**

- ☞ no consensus on how to evaluate those **external attributes**
- ☞ some attributes can only be assessed when the system is available (**too late**...)

THE "**OUTSIDE-IN**" APPROACH FOR QUALITY:

A defined and controlled process is a **required** condition for the production of quality software products and with fewer costs

THE "**INSIDE-OUT**" APPROACH FOR QUALITY:

The quality of internal structure is a **required** condition for ensuring that external quality and increased productivity are achieved.

"INSIDE-OUT" ***QUALITY IN OBJECT-ORIENTATION***

Internal quality in OO is due to the use of:

- encapsulation
- information hiding
- inheritance
- polymorphism
- message passing
- reuse (class, pattern or framework level)

☞ the use of mechanisms that support those concepts can be **varied**, **depending on the designer ability** - several alternatives for the same system are possible

☞ for the same specification we can expect rather different quality products to emerge, as well as different productivity gains, **depending on the corresponding design**

GOAL, STRATEGY AND TACTICS

Goal:

Improve the OO design process to achieve better maintainability and reusability, by setting design recommendations (heuristics). Those may be included in OO CASE tools.

***Strategy**:*

Identify structural quality in designs by means of quantitative evaluation of the use of the OO paradigm mechanisms that are supposed to be responsible for internal quality

Tactics:

Establish comparisons throughout the academic and practitioners community

CRITERIA FOR DESIRED METRICS

Criterion 1: metrics determination should be formally defined

☞ different people at different times or places get the same values for the same systems
☞ avoids subjective ratings (e.g. Very Low, Low, Average, High, Very High)

Criterion 2: non-size metrics should be system size independent

☞ allows comparisons across different projects (cumulative knowledge)

Criterion 3: metrics should be dimensionless or expressed in some unit system

☞ avoids subjective or "artificial" units that inevitably yield to misunderstandings. (e.g. LOC, Function Points)

Criterion 4: metrics should be obtainable early in the life-cycle

☞ allows to identify possible flaws, before too much effort is built on top of them

Criterion 5: metrics should be orthogonal

☞ thus representing different aspects of the system under measurement

Criterion 6: metrics should be easily computable

☞ eases the repetitive, tedious and expensive collection process
☞ requires that criterion 1 is met and that designs are also formally defined

Criterion 7: metrics should be language independent

☞ allows a common base of understanding for the metrics analysis process

THE MOOD METRICS SET (V1.2)
(Metrics for Object Oriented Design)

Several authors have suggested sets of metrics for the OO paradigm, but none known fulfills all the above criteria. The MOOD metrics are:

- *Attribute Hiding Factor*
- *Method Hiding Factor*
- *Method Inheritance Factor*
- *Attribute Inheritance Factor*
- *Coupling Factor*
- *Polymorphism Factor*
- *Clustering Factor*
- *Reuse Factor*

☞ These metrics can be viewed as ***probabilities*** - quantifying the use of different abstractions - ranging from **0** (total absence of use) to **1** (maximum possible use)

☞ They allow the application of statistical theory to software metrics

VALIDATION EXPERIMENT

THE SAMPLE (all in C++ source code):

- **Microsoft Foundation Classes (MFC) -** Microsoft Corporation
- **GNU glib++ (GNU) -** Free Software Foundation / Cygnus Support
- **ET++ library (ET++) -** UBILAB / Union des Banques Suisses (Switzerland)
- **NewMat library (NMAT) -** Robert B. Davies (Victoria University - New Zealand)
- **MotifApp library (MOTIF) -** Douglas A. Young [Young92]
- **Sun C++ Compiler Class library (SunC++) -** Sun Corporation
- **Centerline C++ Compiler Class Library (Centerline) -** Centerline Corporation
- **Not Invented Here Class Library (NIHCL) -** Free Software Foundation

	MFC	**GNU**	**ET++**	**NewMat**	**Motif**	**SunC++**	**Centerline**	**NIHCL**	TOTAL
Classes	135	84	262	90	35	54	36	64	760
Methods	2970	1478	4812	881	199	465	428	2094	13327
Attributes	613	151	980	119	76	119	102	181	2341
LOC	74895	32371	55022	12795	4884	36487	4636	15315	236405

Table 1 - Some indicators of sample size

Information Hiding

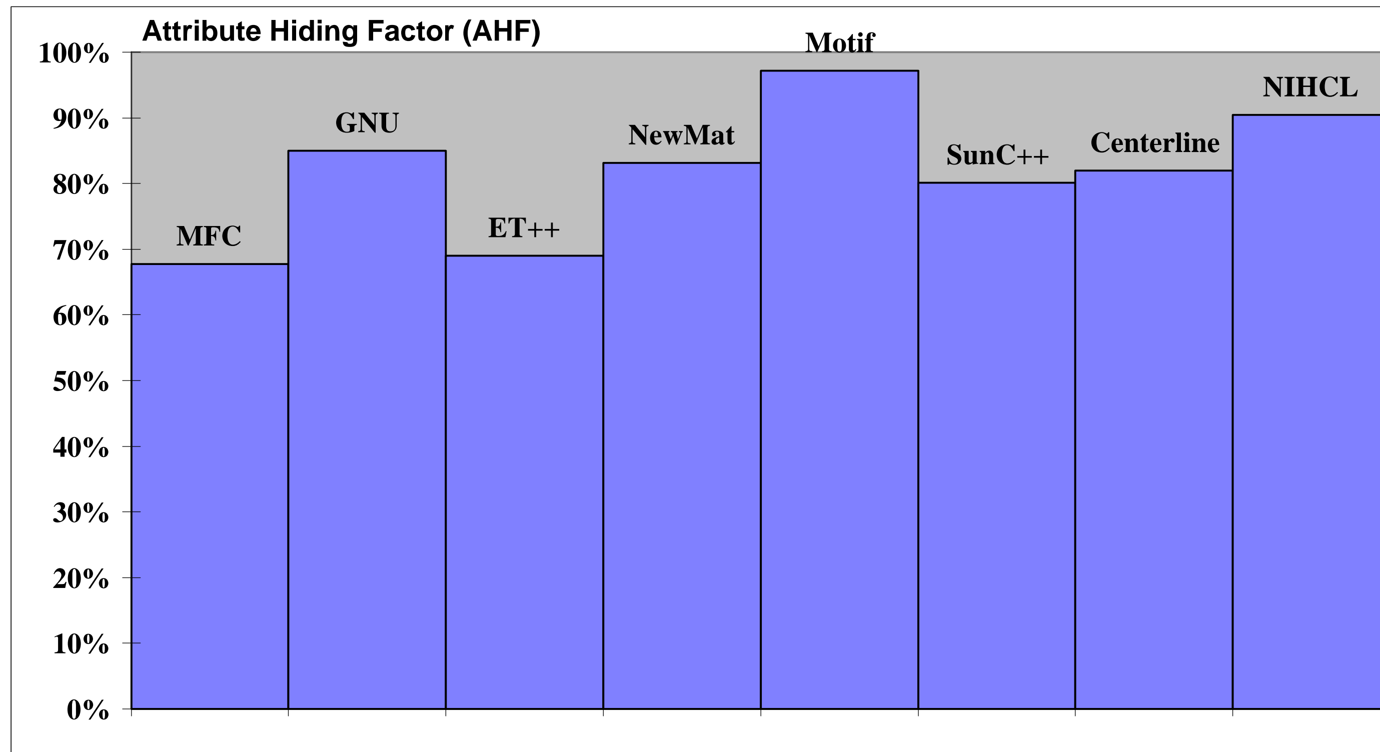
Attributes (or ***variables***, ***fields, data members, ...***) may be visible or hidden

$$A_d(C_i) = A_v(C_i) + A_h(C_i)$$

Attribute Hiding Factor

$$AHF = \frac{\sum_{i=1}^{TC} A_h(C_i)}{\sum_{i=1}^{TC} A_d(C_i)} = 1 - \frac{\sum_{i=1}^{TC} A_v(C_i)}{\sum_{i=1}^{TC} A_d(C_i)}$$

SAMPLE VALUES FOR AHF

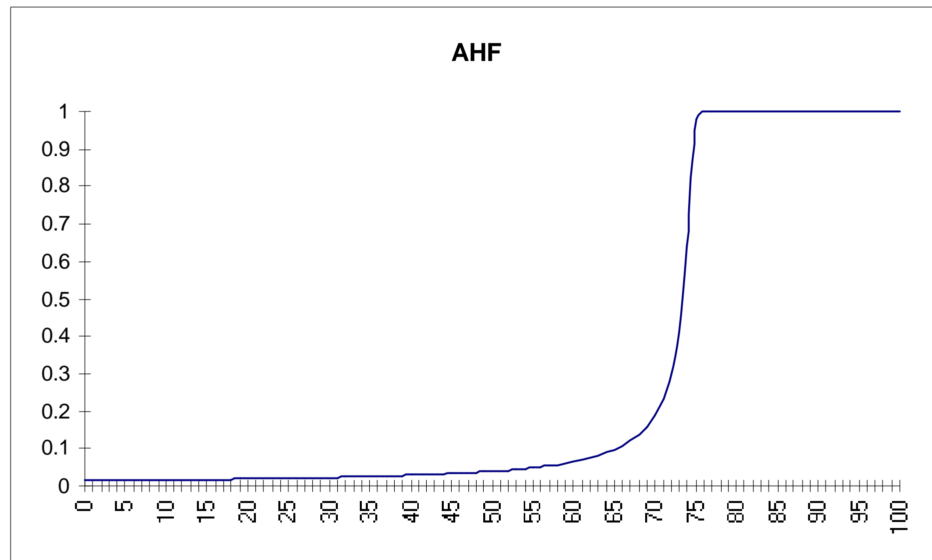


	AHF
MFC	67.7%
GNU	85.0%
ET++	69.0%
NewMat	83.2%
Motif	97.1%
SunC++	80.2%
Centerline	81.9%
NIHCL	90.4%

HEURISTIC FOR AHF

AHF and *MHF* are a measure of the use of the *information hiding* concept that is supported by the *encapsulation* mechanism. Information hiding should be used to:

- cope with complexity by looking at complex components such as black boxes
- reduce "side-effects" provoked by implementation refinement
- support a top-down approach
- test and integrate systems incrementally.



For attributes (AHF) we want this mechanism to be used *as much as possible*. Ideally all attributes would be hidden (being only accessed by the corresponding class methods)

Very low values for AHF should trigger the designers' attention.

Information Hiding

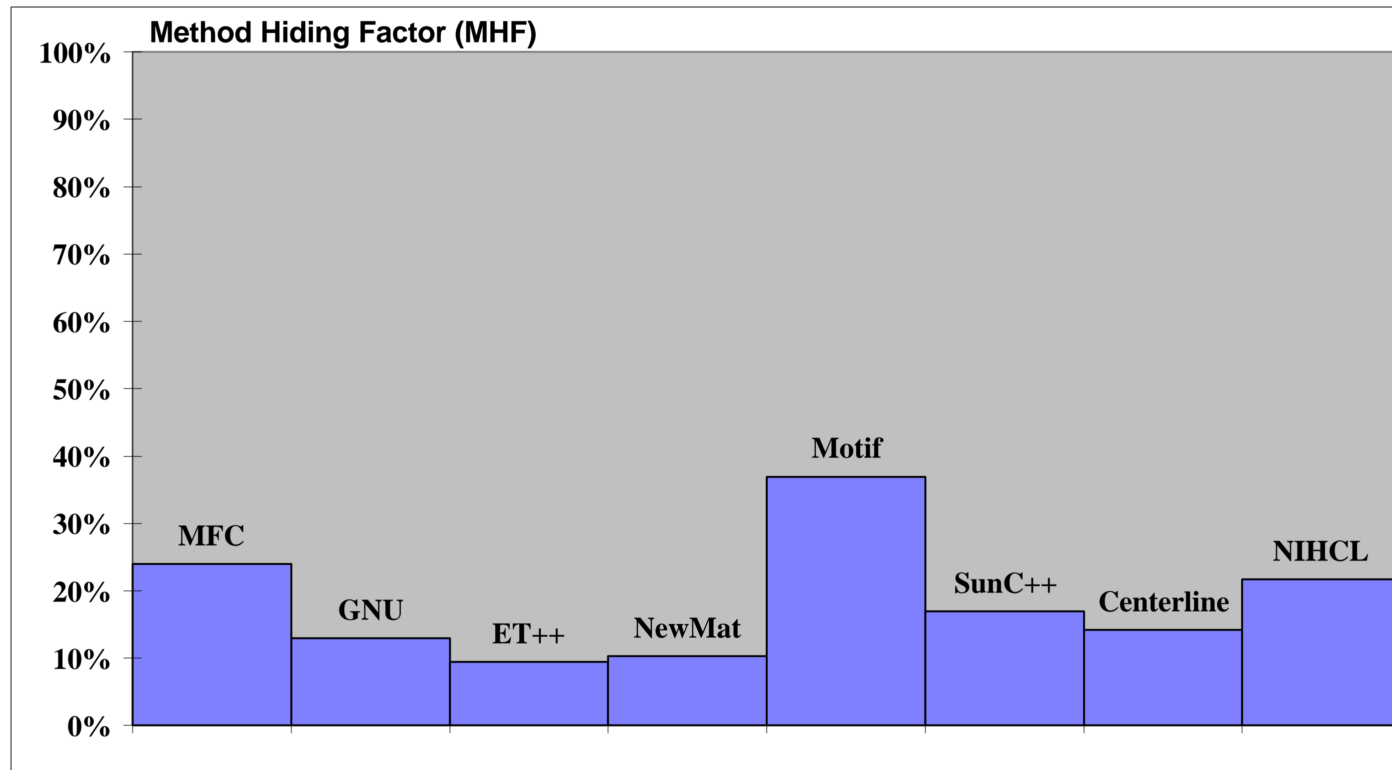
Methods (or ***operations, function members, tasks, routines, ...***) may be visible or hidden

$$M_d(C_i) = M_v(C_i) + M_h(C_i)$$

Method Hiding Factor

$$MHF = \frac{\sum_{i=1}^{TC} M_h(C_i)}{\sum_{i=1}^{TC} M_d(C_i)} = 1 - \frac{\sum_{i=1}^{TC} M_v(C_i)}{\sum_{i=1}^{TC} M_d(C_i)}$$

SAMPLE VALUES FOR MHF

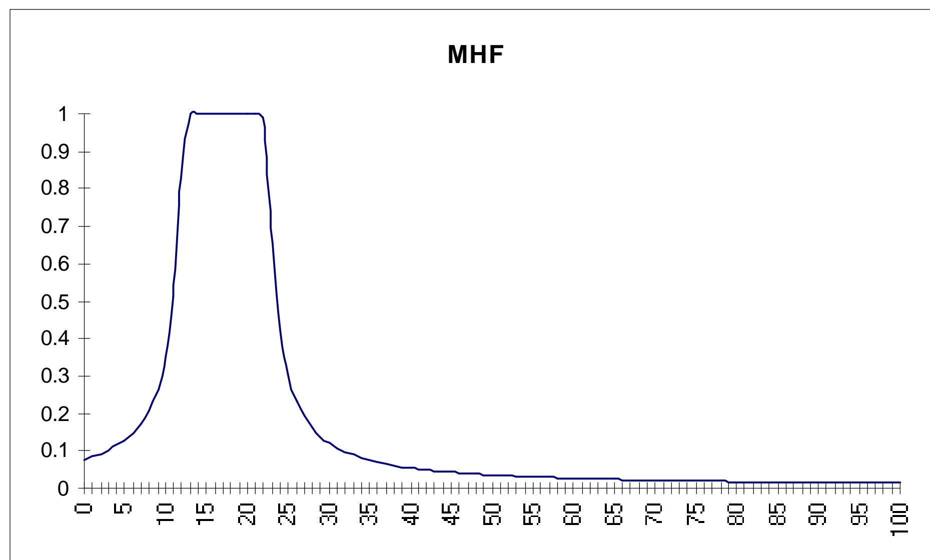


	MHF
MFC	24.0%
GNU	12.9%
ET++	9.5%
NewMat	10.3%
Motif	36.9%
SunC++	16.9%
Centerline	14.3%
NIHCL	21.7%

HEURISTIC FOR MHF

The number of visible methods is a measure of the class functionality. Increasing the overall functionality will *reduce MHF*.

For implementing that functionality we must adopt a top-down approach, where the abstract interface (visible methods) should only be the tip of the iceberg. This suggests a *MHF increase*.



Very low MHF indicate an insufficiently abstracted implementation.

Very high MHF indicate very little functionality

Inheritance

Methods available in a class instance (response for a class) can be the defined ones or inherited ones (not overridden):

$$M_a(C_i) = M_d(C_i) + M_i(C_i)$$

Methods defined in a class can be new (locally defined) or a redefined version (overriding) of inherited ones:

$$M_d(C_i) = M_n(C_i) + M_o(C_i)$$

Method Inheritance Factor

$$MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)} = 1 - \frac{\sum_{i=1}^{TC} M_d(C_i)}{\sum_{i=1}^{TC} M_a(C_i)}$$

Inheritance

Attributes available in a class can be the defined ones or inherited ones (not overridden):

$$A_a(C_i) = A_d(C_i) + A_i(C_i)$$

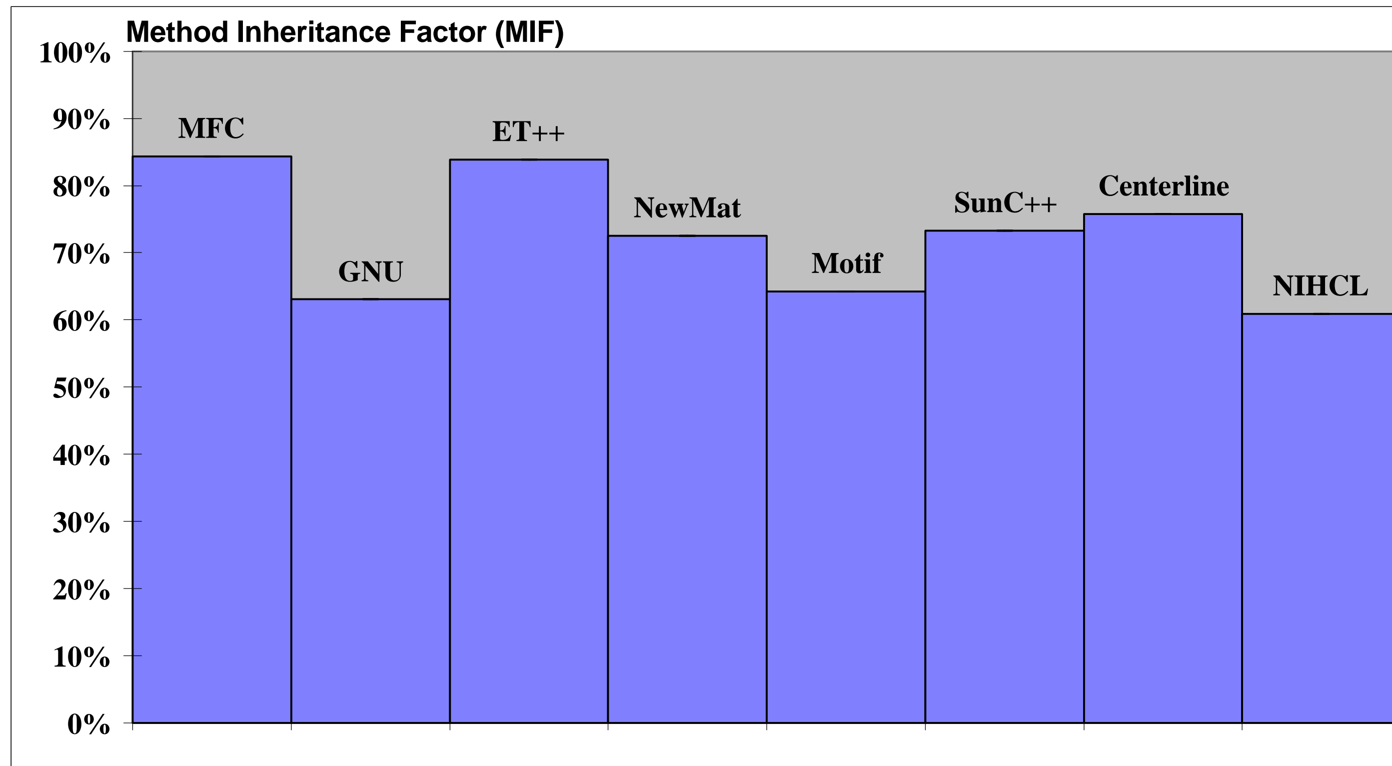
Attributes defined in a class can be new (locally defined) or a redefined version (overriding) of inherited ones:

$$A_d(C_i) = A_n(C_i) + A_o(C_i)$$

Attribute Inheritance Factor

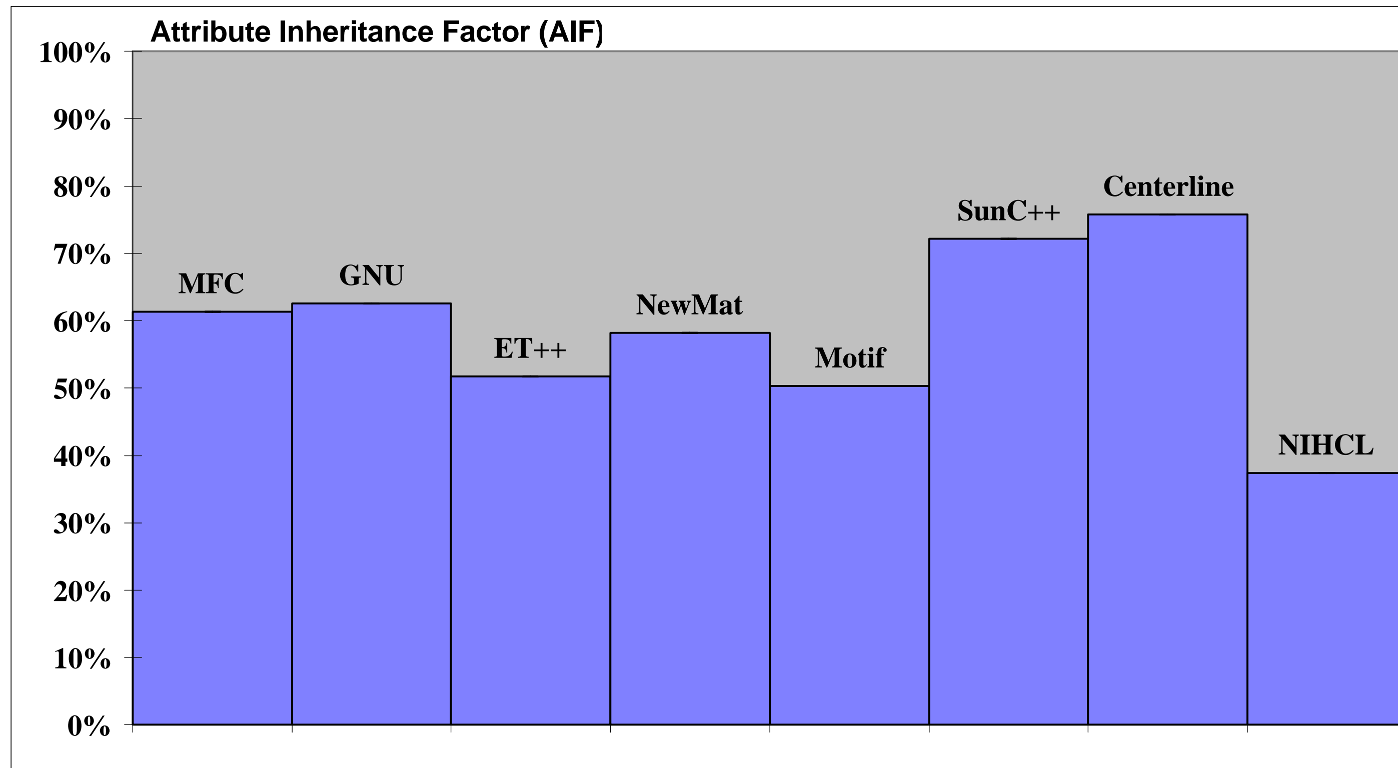
$$AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)} = 1 - \frac{\sum_{i=1}^{TC} A_d(C_i)}{\sum_{i=1}^{TC} A_a(C_i)}$$

SAMPLE VALUES FOR MIF



	MIF
MFC	84.4%
GNU	63.1%
ET++	83.9%
NewMat	72.5%
Motif	64.3%
SunC++	73.3%
Centerline	75.8%
NIHCL	60.9%

SAMPLE VALUES FOR AIF



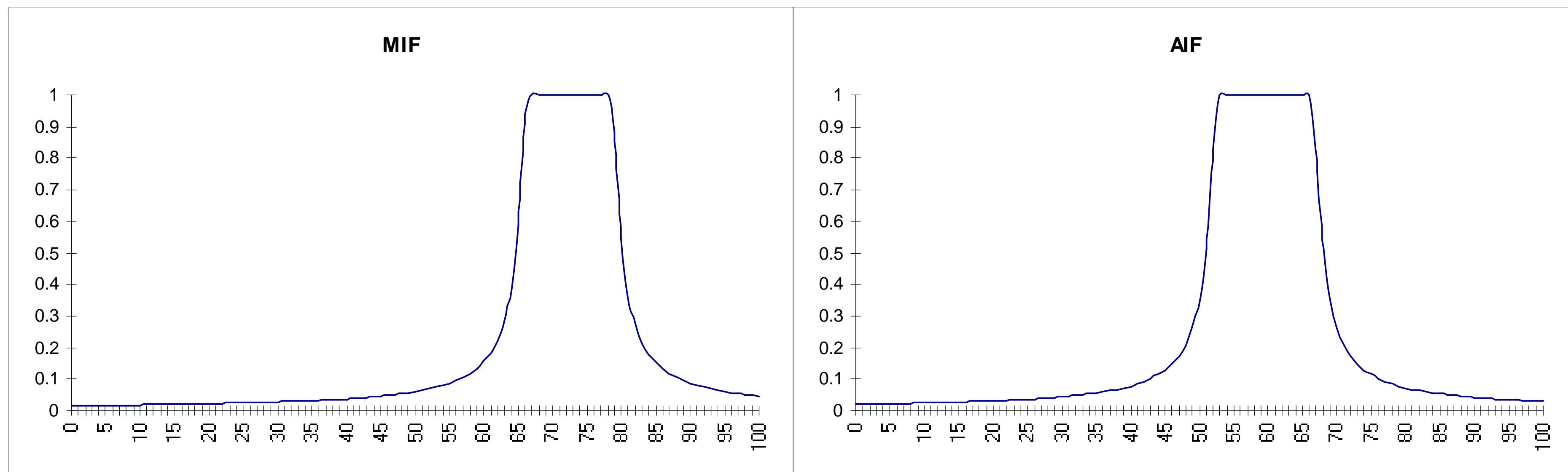
	AIF
MFC	61.3%
GNU	62.6%
ET++	51.8%
NewMat	58.1%
Motif	50.3%
SunC++	72.2%
Centerline	75.7%
NIHCL	37.4%

HEURISTICS FOR MIF AND AIF

MIF and AIF are measures of *inheritance.* This allows:

- expressing similarity among classes
- the portrayal of generalization and specialization relations
- simplification of the definition of inheriting classes, by means of reuse

However, the composition of several inheritance relations builds a tree, whose depth and width make understandability and testability quickly fade away.



Polymorphism

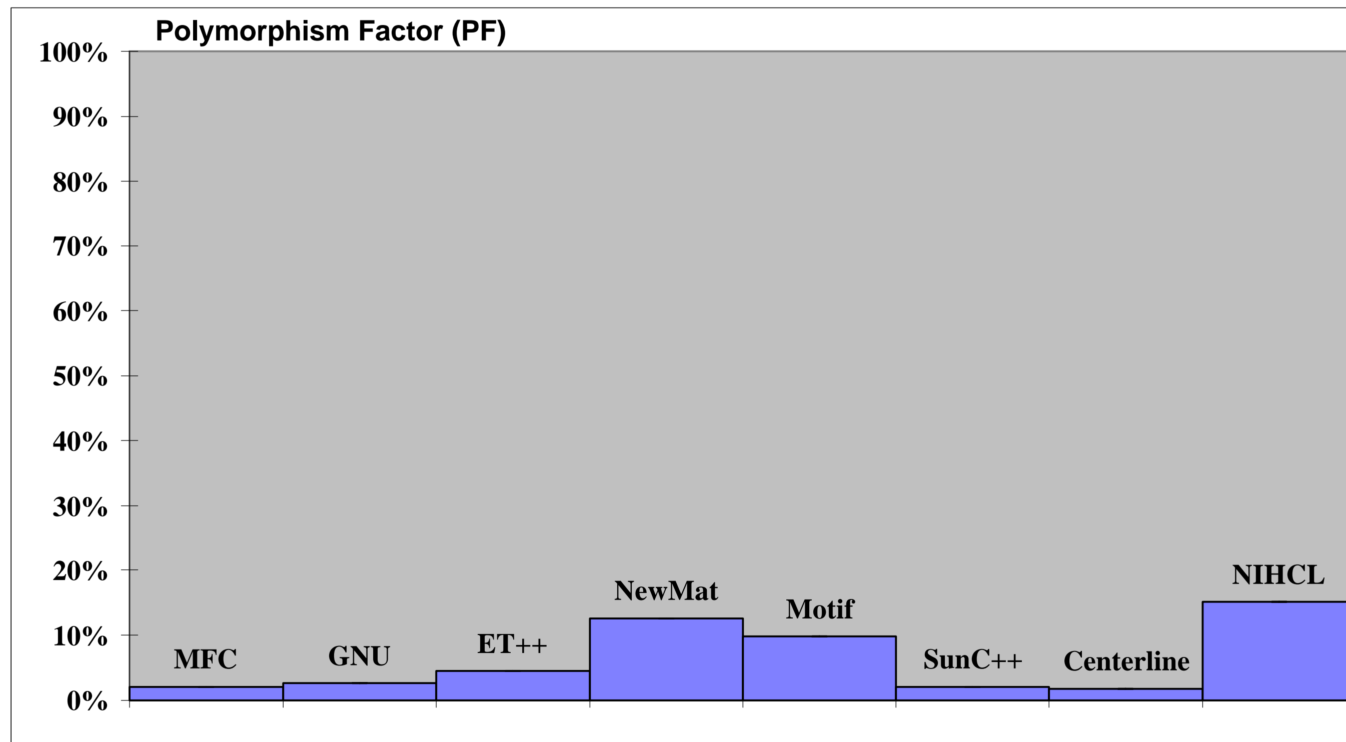
Polymorphism Factor

$$PF = \frac{\sum_{i=1}^{TC} M_o(C_i)}{\sum_{i=1}^{TC} \left[M_n(C_i) \times DC(C_i) \right]}$$

Numerator: *possible different polymorphic situations* (system under consideration)

Denominator: *maximum number of possible different polymorphic situations* (extreme situation where all methods were overridden in all classes, except the base ones)

SAMPLE VALUES FOR PF:

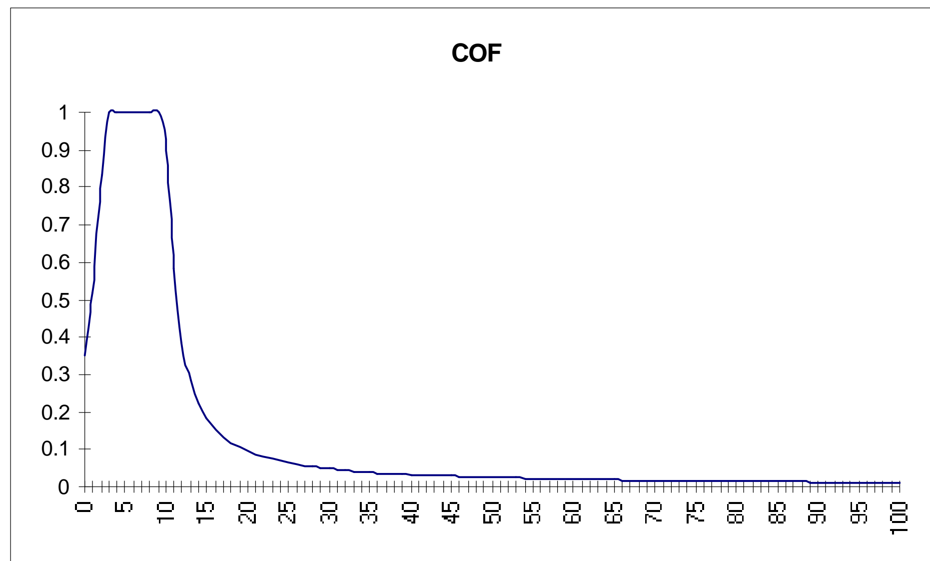


	PF
MFC	2.0%
GNU	2.6%
ET++	4.5%
NewMat	12.7%
Motif	9.8%
SunC++	2.1%
Centerline	1.7%
NIHCL	15.1%

HEURISTIC FOR PF:

By allowing to bind a common message call to one of several class instances, polymorphism allows:

- to build flexible systems
- refinement of the taxonomy without side-effects



However if we need to debug such a taxonomy, by tracing the control flow, this same polymorphism will make the job harder.

This is particularly true if we compare this situation with the procedural counterpart where, for a similar functionality, we usually have a series of decision statements for triggering the required operation.

Coupling

A ***client*** class is the one that contains at least one reference to a feature (or the whole) of another class (the **supplier**) . References can be of several kinds**:**

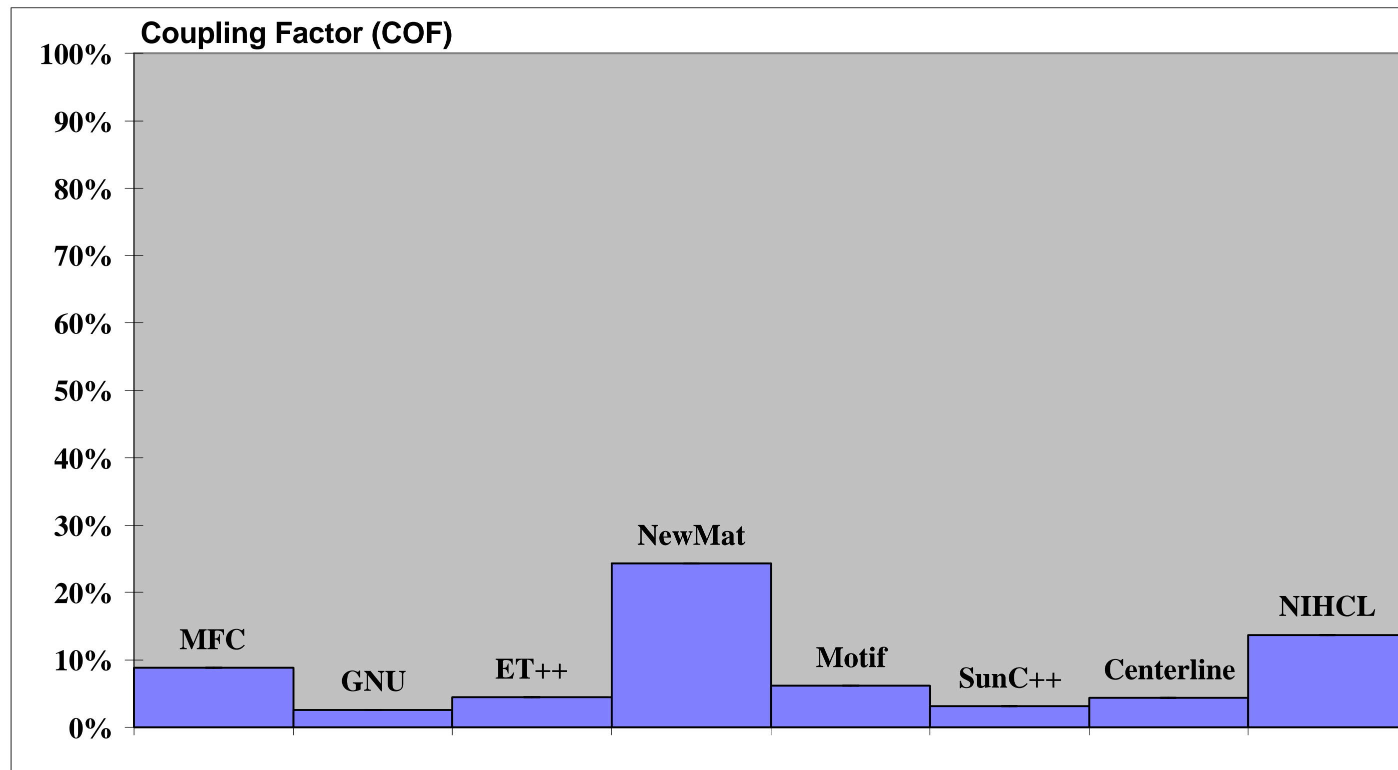
- ☞ ***messages (or event, stimulus*, ...**) to another class instance (dynamic coupling)
- ☞ references due to semantic associations (static coupling) / object containment

Coupling Factor

$$COF = \frac{\sum_{i=1}^{TC}\left[\sum_{j=1}^{TC} is_client(C_i, C_j)\right]}{TC^2 - TC - 2 \times \sum_{i=1}^{TC} DC(C_i)}$$

$$is_client(C_c, C_s) = \begin{cases} 1 \quad iff & C_c \Rightarrow C_s \wedge C_c \neq C_s \\ & \wedge \neg (C_c \rightarrow C_s) \\ 0 \quad otherwise & \end{cases}$$

SAMPLE VALUES FOR COF

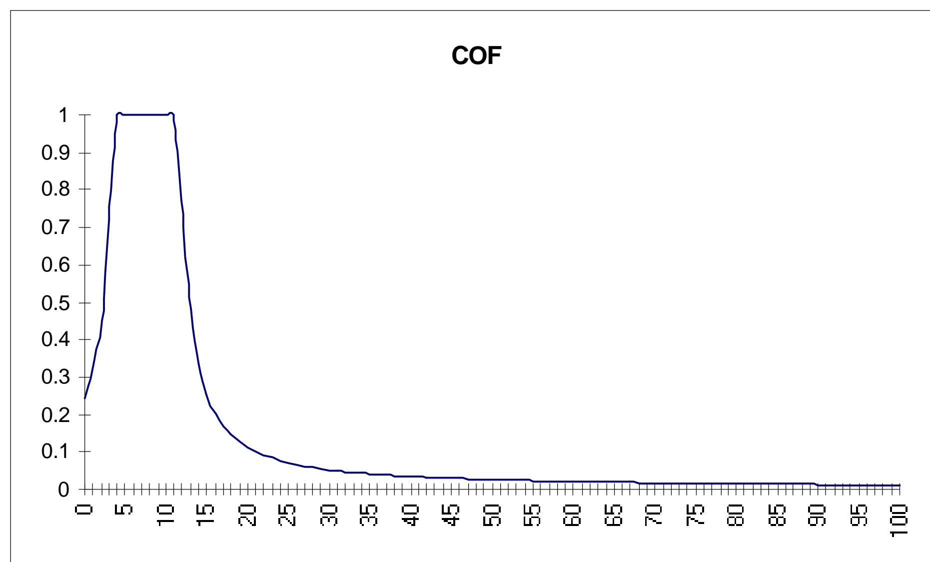


	COF
MFC	8.8%
GNU	2.6%
ET++	4.4%
NewMat	24.3%
Motif	6.3%
SunC++	3.1%
Centerline	4.3%
NIHCL	13.8%

HEURISTIC FOR COF

It has been noted [Meyer88] that it is desirable that classes communicate with as few others as possible and even then, that they exchange as little information as possible.

Coupling relations increase complexity, reduce encapsulation and potential reuse, and limit understandability and maintainability.



Thus, it seems that we should avoid it as much as possible. Very high values of COF should be avoided by designers.

However, for a given application, classes must cooperate somehow to deliver some kind of functionality. Therefore, COF is expected to be lower bounded.

HEURISTICS IN NUMBERS ...

	Minimum	Mean	Maximum	Shape
MHF	**12.7%**	17.3%	**21.8%**	BP
AHF	**75.2%**	81.9%		HP
MIF	**66.4%**	72.4%	**78.5%**	BP
AIF	**52.7%**	59.5%	**66.3%**	BP
COF	**4.0%**	7.6%	**11.2%**	BP
PF	**2.7%**	6.1%	**9.6%**	BP

Table 2 - 95% Confidence interval for the sample

- A few values below the 10% percentile and above the 90% percentile were set to the corresponding percentile limit values (outlier removal)

SIZE INDEPENDENCE

	Classes	Methods	Attrib.	LOC
MHF	-0.41	-0.23	-0.21	-0.02
AHF	-0.75	-0.71	-0.80	-0.84
MIF	0.67	0.57	0.75	0.68
AIF	-0.31	-0.42	-0.28	0.04
COF	-0.08	-0.01	-0.15	-0.22
PF	-0.18	-0.13	-0.28	-0.52

Table 3 - Correlation of MOOD metrics with some size metrics

MOOD METRICS ORTHOGONALITY

	MHF	AHF	MIF	AIF	COF	PF
MHF		0.40	-0.26	-0.30	0.04	0.17
AHF			-0.91	-0.28	0.17	0.58
MIF				0.28	-0.14	-0.50
AIF					-0.47	-0.73
COF						0.82
PF						

Table 4 - Correlation among MOOD metrics

EVOLUTION OF OUR RESEARCH

i) MOOD metrics set proposal [Abreu94]

ii) practical validation of the underlying rationale of the proposed set

iii) construction and public distribution of a tool for automatic collection of MOOD metrics

iv) support for industrial and academic wide experimentation and statistical validation

v) theoretical validation of the MOOD metrics (using Measurement Theory)

vi) MOOD set refinement based on iii) and iv) results (MOOD V2 proposal)

vii) embedding of MOOD V2 and corresponding design heuristics on a OO CASE tool

viii) assessment of correlation between MOOD and maintainability sub-characteristics