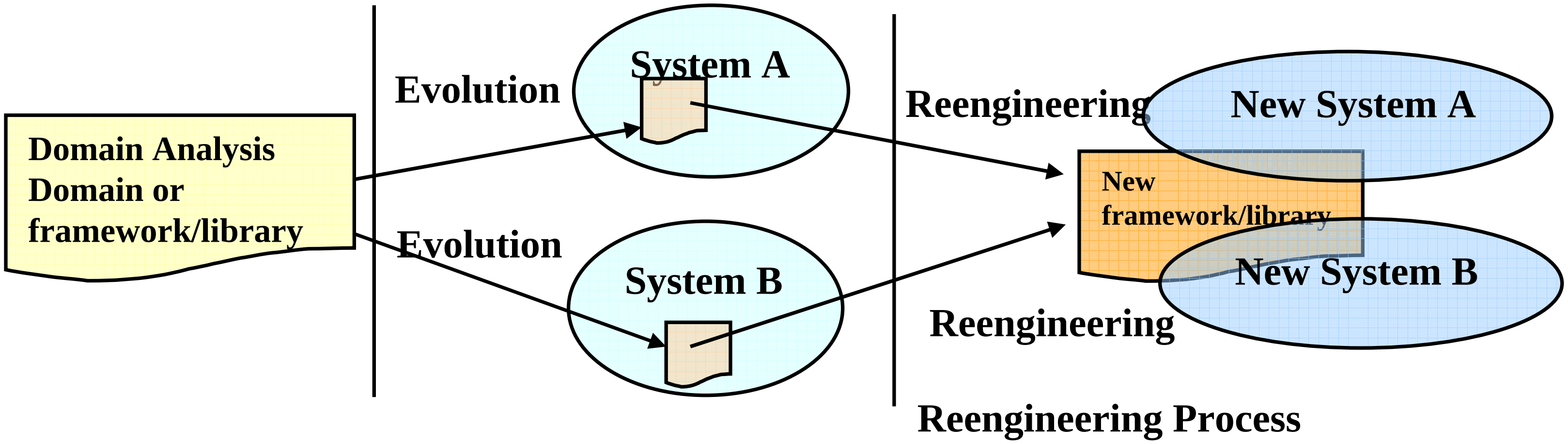
Reengineering Analysis of Object-Oriented Systems via Duplication Analysis

F. Fioravanti, G. Migliarese, P. Nesi

Department of Systems and Informatics
University of Florence
Via S. Marta 3, 50139, Firenze, Italy
tel: +39-055-4796523, fax: +39-055-4796363
nesi@ingfi1.ing.unifi.it,   nesi@dsi.unifi.it
http://www.dsi.unifi.it/~nesi

Problem Description

- **Different OO software applications** evolved on the basis of the same initial version of an OO Class Library or FW
- **Re-engineering the**
  - library (generalising, improving, etc.) and
  - applications (sharing more code with the library)
- **Via Code Analysis looking for duplications**
  - manual/assisted inspection for duplication and reasoning
  - duplication analysis with TREND tool
  - comparison of these processes



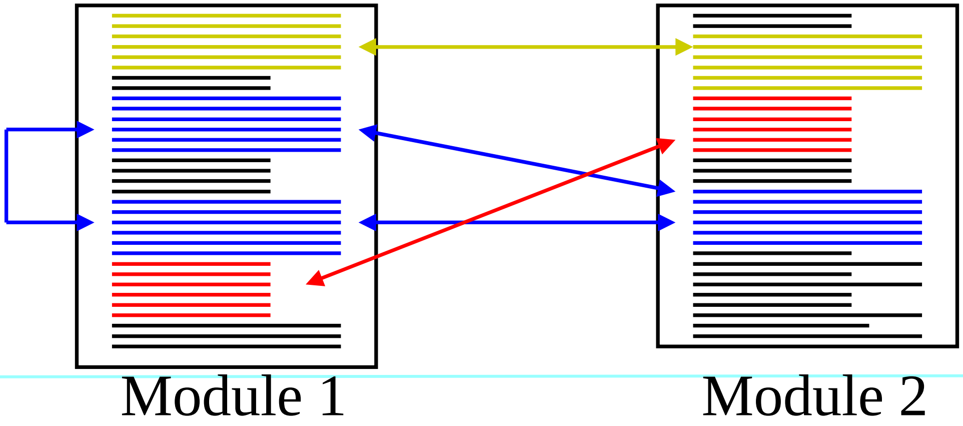
Problem Description, Why and Effects

- **Reengineering performed to**
  - ♣ extend the library/framework application domain
  - ♣ reduce costs of maintenance, of applications and of library
  - ♣ increase maintainability, ...
  - ♣ reduce the fault proneness, ….

- **Unfortunately, *high costs for code reengineering***
  - → ***understanding***
  - → ***navigation***
  - → ***processing:***
    - → ***method generalisation, moving attributes, changing class hierarchy, etc.***

Duplications and Similarities in OO systems

- **Functional Duplications (cut and past !!)**
  - Self duplication of code segments (methods and/or classes)
  - duplication of methods instead of parameterisation
  - similar methods in different classes, different types
- **Structural Similarities**
  - similar classes in terms of data structure
  - classes with different names, modelling “similar” objects
  - class attributes of the same type with different names for “modelling the same aspects”



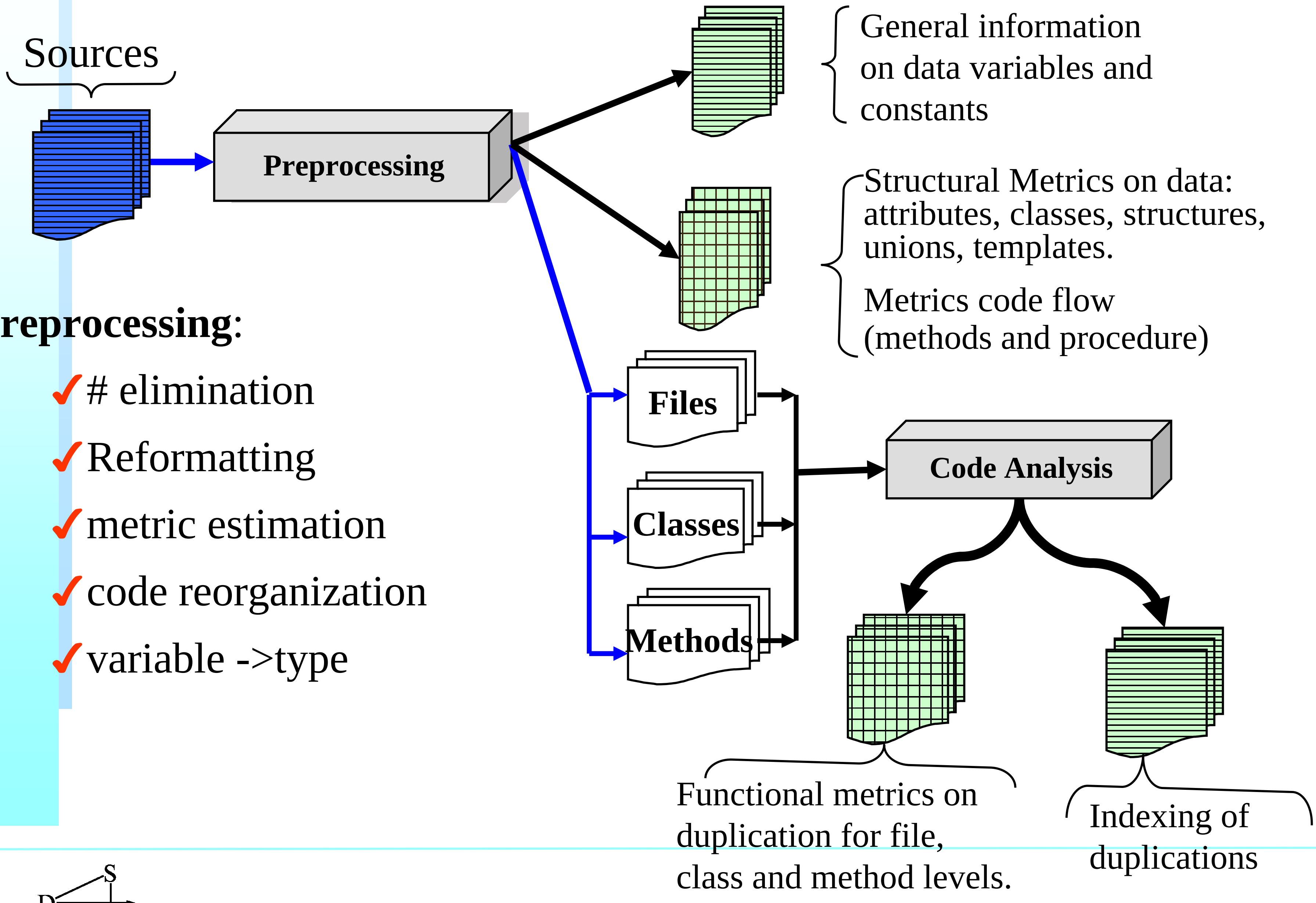
Duplication Analysis and Metrics for OO

- **Duplication analysis can be influenced by**
  - #… solving
  - format dependency
    - blank lines, comments, etc.
  - single line vs small sequences of duplicated lines
  - variable and their types
  - computationally intensive
    - **polishing code, standardizing formatting**
- **To reduce these problems: pre-processing and specific metrics and a method for OO systems**
  - detecting duplications at levels of
    - files, classes and methods
  - class similarity on class structure and hierarchy

Research Tool: TREND

- **support for the Reengineering Process**
  - ♣ **Code Analysis for object oriented code**
- **Several different *algorithms and metrics* for**
  - ♣ pre-processing
  - ♣ duplication analysis
  - ♣ similarity detection
- **Reasoning on results, Method proposed**
  - ♣ support of visualisation methods
- **Reorganisation of classes/methods in modules**
  - ♣ classes and methods assignment to modules, packages
- **code indexing for**
  - ♣ fast manual precise reengineering

General TREND Process Architecture



Sources

Preprocessing

General information on data variables and constants

Structural Metrics on data: attributes, classes, structures, unions, templates.

Metrics code flow (methods and procedure)

Files

Classes

Methods

Code Analysis

Functional metrics on duplication for file, class and method levels.

Indexing of duplications

Preprocessing:

- ✓# elimination
- ✓Reformatting
- ✓metric estimation
- ✓code reorganization
- ✓variable ->type

Pre-processing algorithms

- **SPA,** Source Processing Algorithm
  - Comment elimination, preprocessing phase of compiler
- **SFA,** Source Formatting Algorithm
  - one instruction per line ({ and } on different lines)
  - *structural metrics estimation*
  - *code reorganization, one Hxx + one Cxx per class*
  - *flow charts of methods and procedures*
- **VSA**, Variable Substitution Algorithm
  - substitution of Variable names with their type/class name

Code analysis, Duplication analysis

- **Algorithms for Duplication Estimation**
  - line duplication at level of files, class and methods
  - sequence of duplicated lines (minimum number of duplicated lines to consider a duplication)
  - estimation of duplication metrics

line sequences

- **Extraction of structural and functional aspects**
  - structural similarities of classes
  - metric values about similarities

- **Results analysis**
  - →*some guidelines for reasoning on tables*
  - →*visualisation mechanisms*
  - →*code indexing for code navigation*

General and Structural Metrics

- ***TNAL*** Total Number of Locally defined Attributes
- ***TLOC*** Total number of Lines Of Code
- ***NCL*** Number of system classes
- ***TNML*** Total Number of Locally defined Methods
- ***Nbyte*** Total Number of bytes of the system source files
- ***NFile*** Total Number of system source Files

- ***NAL***$_i$ Number of local attributes of the i-th class
- ***NALST***$_{ij}$ *(Number of Attributes Locally defined with the Same Type) number of identical in type attributes between two classes, independently of the access qualifier (i.e., private, protected and public)*

Metrics for Duplication Analysis

- ***NL_i*** Number of Lines of entity X_i
- ***$NLID_{ij}$*** (Number of LInes Duplicated) number of code lines of X_i which are also present in X_j
- ***$NLIDS_{ij}$*** (Number of Lines In Duplicated Sequences) length of the sequence of lines of X_i that are also present in X_j,
  In the next tables the number of consecutive lines was set to 3
- ***$IID_{ij} = 100\ NLD_{ij}/Nl_i$*** (Identity Index of Duplication)
  Duplication index of X_i with respect to X_j
- ***$IID\_S_{ij} = 100\ NLIDS_{ij}/Nl_i$*** (Identity Index of Duplication in Sequences)
  Duplication index of X_i with respect to X_j, by considering only the sequences of duplicated lines
- ***IID_{ij}*** typically different than ***IID_{ji}*** , etc.

Matrices

NxM

N+M entities

- $$HM = \frac{1}{mean\left\{\frac{1}{a_1},\ldots,\frac{1}{a_n}\right\}}$$ ***HM*** Harmonic Mean of the above ***IID, IID_S*** metrics

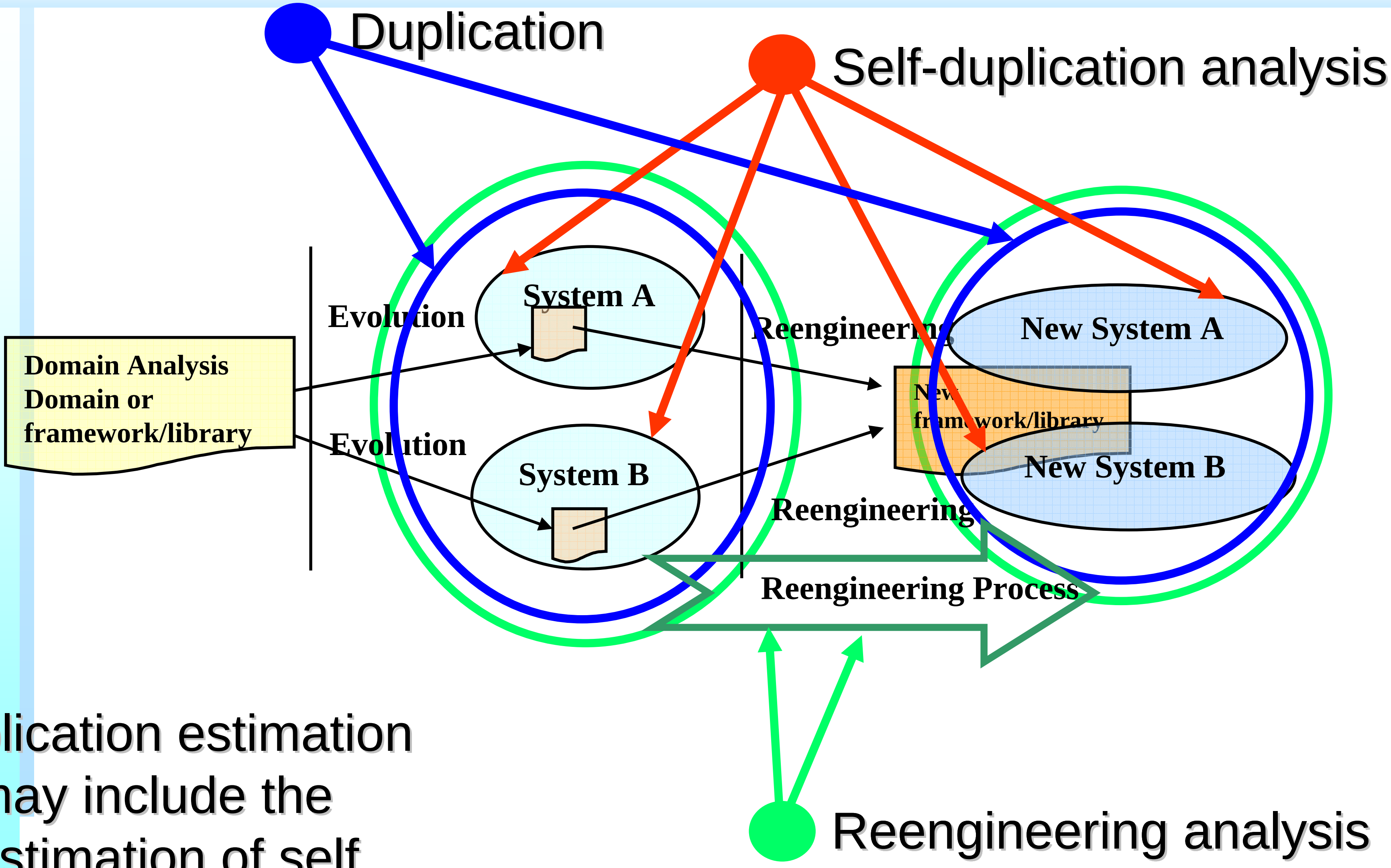
- $\forall i,j \in S_1 \cup S_2$: ***SI = mean{$IID_{ij}$}*** System Identity on NxM, (N+M)x(N+M) values

Reengineering Process: Case Study

- **two applications: MWB and VM**
  - based on the same class library
  - developed by different teams
  - based on the same application domain
- **Reengineering to obtain a New Library and New Applications**
  - increment of maintainability, less code, larger library
  - investment for new applications
- **Manual reengineering**
  - Adoption of a simple tool for duplication detection, no pre-processing and no metrics.
  - skilled people

→ ***Results compared with those obtained by using the proposed metrics and tool, TREND***

Performed Analyses



Duplication

Self-duplication analysis

Domain Analysis
Domain or
framework/library

Evolution

Evolution

System A

System B

Reengineering

Reengineering

New framework/library

New System A

New System B

Reengineering Process

Reengineering analysis to assess the work performed by the team

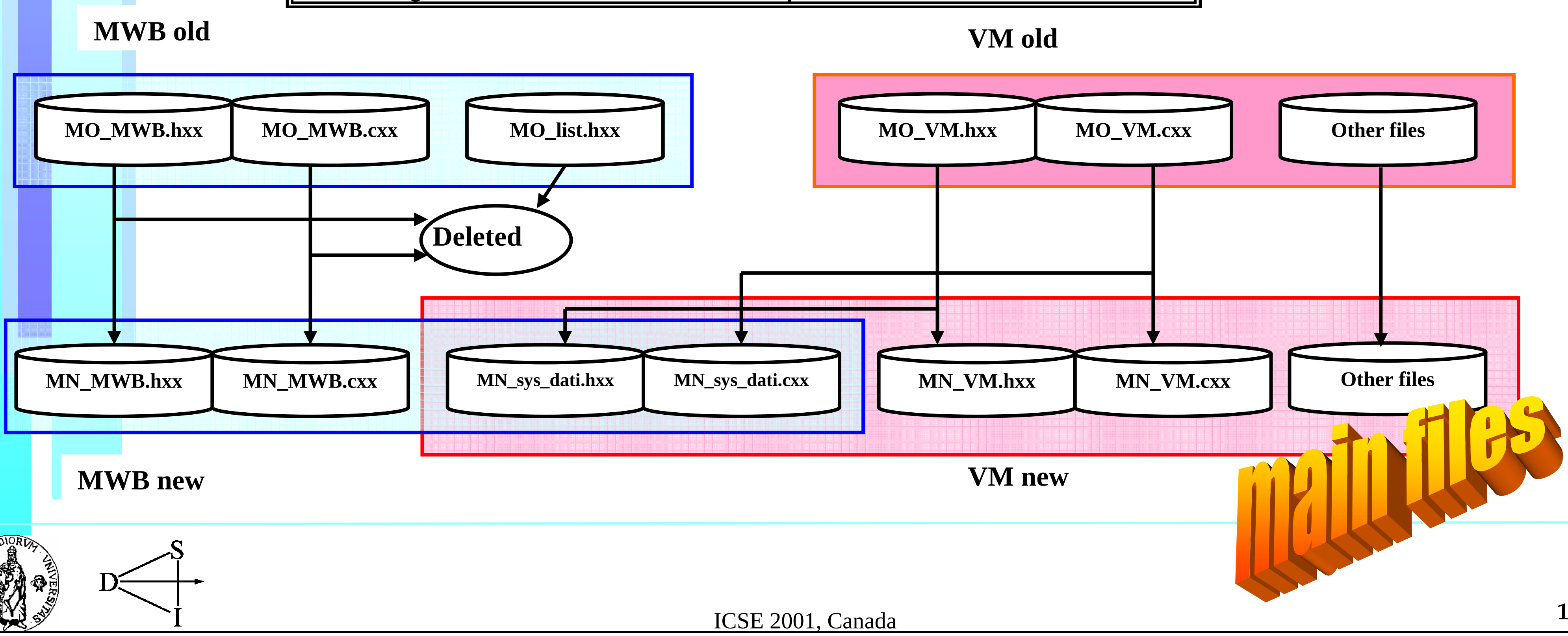
Duplication estimation may include the estimation of self duplications

Manual Re-engineering Results

System	NFile	NCL	TNML	TNAL	TLOC	NBYTE
MWB/VM old	7	76	589	674	12.016	389.921
MWB/VM new	10	62	530	640	11.708	415.659

- 20 classes have been changed and some lines added

OPERATION PERFORMED	NUMBER OF CLASSES
Deleted	14
Modified and moved in the library	13
Modified	7
Moved in the library	10
Unchanged	32



Problems Remained from Manual Reengineering.

- **Some duplications were still present in the new applications:**
  - for the lack of detection of structural similarities, the class hierarchy was not modified in deep
  - several files with more than 20% of duplication, among these: 3 present more than the 30% of duplication.
- **Similarities were hard to be identified**
  - structural similarities and duplications should drawn the reengineering process
  - a manual reengineering with a greater precision was impossible with only 6 MM of effort

TREND Pre-Processing

- **Pre-processing of old MWB and VM Files**

Algorithm	*SI*
No Preprocessing	6.88
SPA	8.38
SFA	10.63
SPA+SFA	13.50
SPA+SFA+VSA	17.25

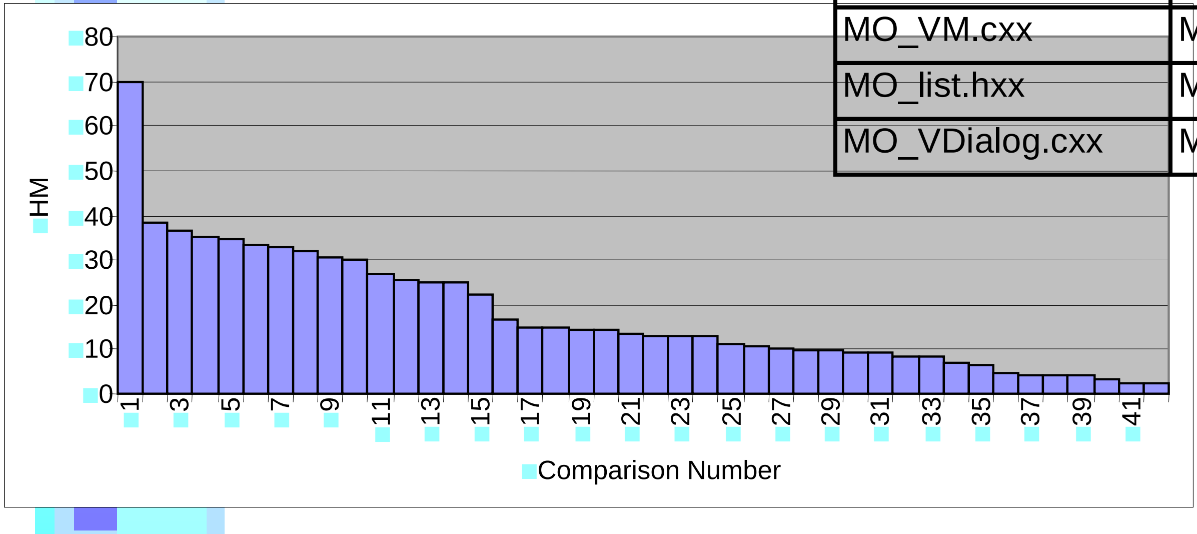
- ***SI*** is the System Duplication Index (mean of IID matrices)
- data obtained for the estimation at file level

- **SPA+SFA+VSA** was the best solution for preparing the duplication analysis as confirmed by the experts considering the values estimated by the single modules

File Duplication Analysis, Old Versions

- Only files with HM>27% have been reported
- Cxx and Hxx files were considered

FILE1	FILE2	IIDXY	IIDXY_S	HM
MO_list.hxx	MO_VM.cxx 🔵	85	83	70
MO_VDialog.cxx	MO_gpro.cxx 🔴	59	39	38
MO_list.hxx	MO_MWB.cxx 🔵	61	37	37
MO_gpro.cxx	MO_VDialog.cxx 🔴	56	36	35
MO_list.hxx	MO_VDialog.cxx 🔵	60	35	35
MO_MWB.cxx	MO_VM.cxx 🔵	56	31	33
MO_MWB.hxx	MO_VM.hxx 🔵	54	34	33
MO_VM.cxx	MO_MWB.cxx 🔵	51	24	32
MO_list.hxx	MO_gpro.cxx 🔵	57	28	30
MO_VDialog.cxx	MO_VM.cxx 🔴	52	29	30



main duplications

- **This analysis confirmed the work performed by the experts**
  - deletion of MO_list.hxx
  - moving part of MO_MWB.cxx and MO_VM.cxx into LIB

File vs Class Level Analysis

- **File level analysis is too coarse for reasoning on classes and class hierarchy**
- **In many cases,**
  - ♣ Hxx files contain more than one class (negative aspect)
  - ♣ Cxx files contain a large number of methods
- **Class level analysis**
  - ♣ structural analysis of classes
  - ♣ duplication analysis of class definition (non useful)
  - ♣ duplication analysis on class methods
- **These analysis produce complementary information**
  - ♣ In terms of attributes' types two classes may similar or even identical structure,   but
  - ♣ their functional part, methods, may confirm or not the similarity.

Class Structural Analysis of the Old Versions

- Similar classes
- Self Similarity in VM
- Self Similarity in MWB
- Similarity analysis VM-MWB

On 76 classes of the old systems, 5776 values, these are those with HMm>= 80%

CLASS$_1$ (FILE)	*CLASS$_2$ (FILE)*	*NAL$_1$*	*NAL$_2$*	*NALST*	*HMm*
New_Sys_Metric (MO_VM.hxx)	**New_Class_Metric (MO_VM.hxx)**	74	75	74	99
Function (MO_VM.hxx)	**Class (MO_VM.hxx)**	13	12	11	88
System_Custom_Metric_Parser (MO_VM.hxx)	**Class_Custom_Metric_Parser (MO_VM.hxx)**	23	31	23	85
Function (MO_VM.hxx)	**Method (MO_VM.hxx)**	13	18	13	84
Method (MO_VM.hxx)	**Class (MO_VM.hxx)**	18	12	12	80
Contenitore (MO_MWB.hxx)	**Contenitore_Value (MO_MWB.hxx)**	2	3	2	80
Contenitore_Value (MO_MWB.hxx)	**Global (MO_MWB.hxx)**	3	2	2	80
Contenitore_Value (MO_MWB.hxx)	**Metriche (MO_MWB.hxx)**	3	2	2	80
Contenitore_Value (MO_MWB.hxx)	**VMDialog (MO_MWB.hxx)**	3	2	2	80
PlotDialog (MO_MWB.hxx)	**Variable (MO_VM.hxx)**	5	4	4	89
Attributo (MO_MWB.hxx)	**Variable (MO_VM.hxx)**	3	4	3	86
Method (MO_MWB.hxx)	**View (MO_VM.hxx)**	6	6	5	83
Attributo (MO_MWB.hxx)	**Parent (MO_VM.hxx)**	3	2	2	80
Contenitore_Value (MO_MWB.hxx)	**Global_Variable (MO_VM.hxx)**	3	2	2	80
Info (MO_MWB.hxx)	**Parent (MO_VM.hxx)**	3	2	2	80
InfoDialog (MO_MWB.hxx)	**Variable (MO_VM.hxx)**	6	4	4	80
LISTA (MO_MWB.hxx)	**Container (MO_VM.hxx)**	2	3	2	80
Method (MO_MWB.hxx)	**Variable (MO_VM.hxx)**	6	4	4	80

Class Functional Analysis of Old Versions

- MWB and VM class methods with duplications (IID>60%)

MO_MWB (1)	*MO_VM (2)*	*IID$_{12}$*	*IID$_{21}$*	*IID_S$_{12}$*	*IID_S$_{21}$*
File	**File**	99	95	99	92
Line	**Line**	96	96	96	96
InfoOpen	**InfoOpen**	91	100	91	100
Value	**Selected_Metric**	75	61	65	43
Single_Metric	**Value**	65	75	44	65
ErrorDialog	**ErrorDialog**	61	89	27	81

- VM ***self*** duplication class methods (IID >70%)

Class$_1$	***Class$_2$***	***NL$_1$***	***NL$_2$***	***IID_S$_{12}$***	***IID_S$_{21}$***	***HM***
New_Class_Metric	**New_Sys_Metric**	824	776	91	96	94
Class_Custom_Metric_Parser	**System_Custom_Metric_Parser**	200	147	74	98	84
Function	**Method**	194	228	74	78	75
Attribute	**Variable**	110	60	76	67	74
Class	**Function**	189	194	74	76	73

- MWB ***self*** duplication class methods (IID>50%)

Class$_1$	***Class$_2$***	***NL$_1$***	***NL$_2$***	***IID_S$_{12}$***	***IID_S$_{21}$***	***HM***
UkdmDialog	**VMDialog**	139	114	62	81	71
CustomMetric	**MetricMember**	105	198	49	58	59
Single_Metric	**Value**	34	20	44	65	51

Summary: Class Analysis of the Old Versions

- The selection of classes presenting a structural similarity and/or functional duplication has allowed to identify the most heavy duplicated classes
- This approach allowed to detect and analyse all relevant duplicated classes in the 30% of time needed to “manual” operation (including the results analysis)
- Great part of classes with HM>50% have been also manipulated by the team during manual reengineering
- VM presented a stronger self duplication than MWB
- Common classes have been detected and moved into the LIB.

OPERATION TO PERFORM	NUMBER OF CLASSES
Deletion	20
Moving in the library	20

System	Nfile	NCL	TNML	TNAL
MWB/VM old	7	76	589	674
MWB/VM new	10	62	530	640
MWB/VM TREND	112	56	505	623

Manual

Method Analysis of the Old Versions

- **Identification of similar methods**
  - ♣ in the same classes
  - ♣ in different classes
  - ♣ structural reasoning about the method parameters
  - ♣ flow chart analysis
  - ♣ analysis based on flow chart metrics
- **reasoning about the distribution of similar methods along the class hierarchy**
- **Definition of guidelines for manipulating methods**

Reengineering Process Analysis, *File Level*

- **To Assess the performed manual Reengineering**
- **Analysis of code movements**
- Old systems have been compared with the new versions, main files
- E.g.:
  - ♣ MN_Metric_Dialog.cxx/hxx are quite completely new.
  - ♣ MN_sys_dati.cxx derives its code from MO_MWB.cxx and MO_VM.cxx

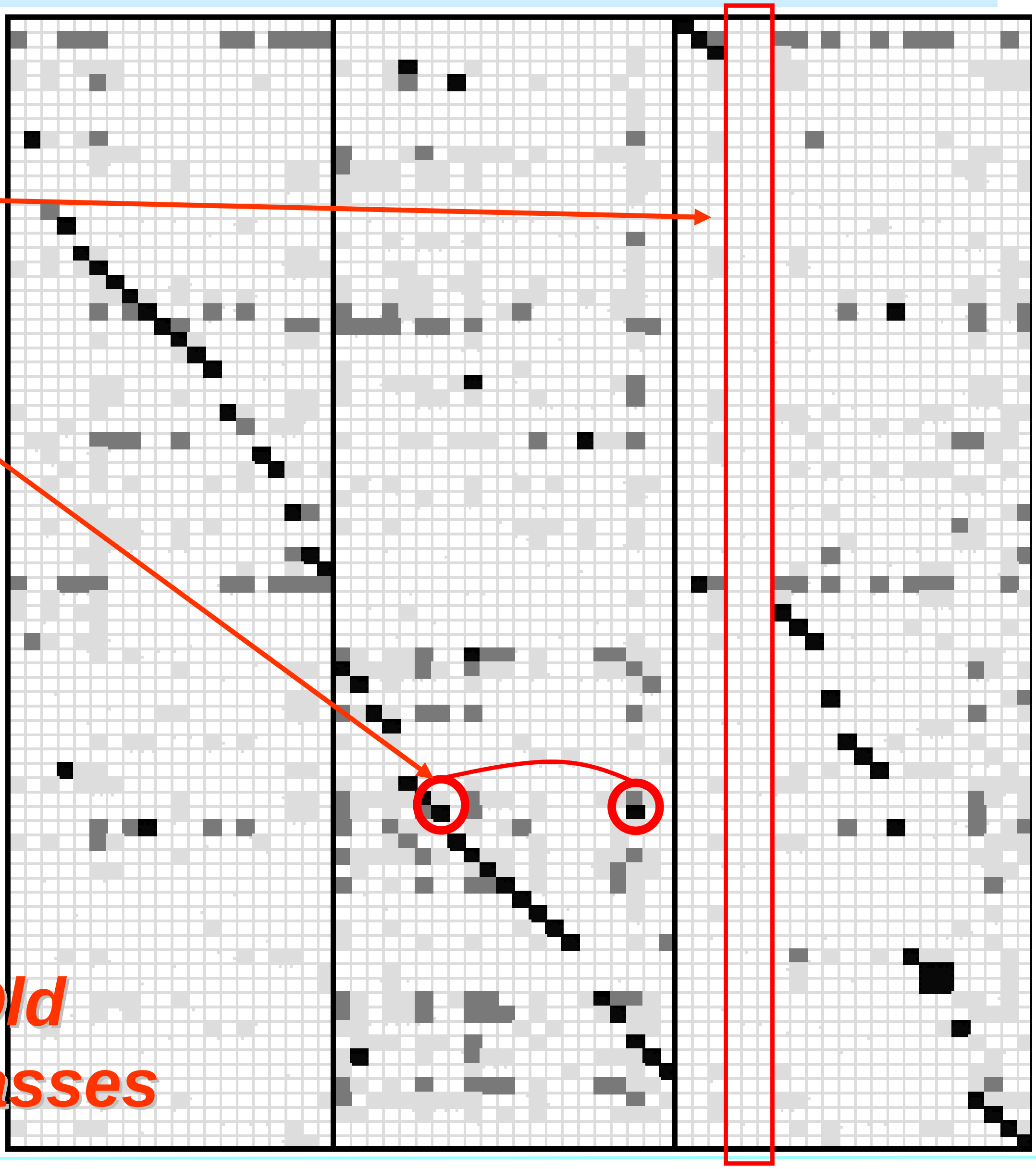
File Level, main files

	MO_gpro.cxx	MO_list.hxx	MO_MWB.cxx	MO_MWB.hxx	MO_VDialog.cxx	MO_VM.cxx	MO_VM.hxx
MN_gpro.cxx	█	░			░		
MN_MetricDialog.cxx					░		
MN_MetricDialog.hxx							
MN_MWB.cxx	░		▓		░	░	
MN_MWB.hxx				▓			░
MN_sys_dati.cxx		░	░			▓	
MN_sys_dati.hxx		░		░			▓
MN_Vdialog.cxx	░	░	░		█	░	
MN_VM.cxx			░			█	
MN_VM.hxx				░			▓

□	HM between 1% and 20%
░	HM between 21% and 50%
▓	HM between 51% and 80%
█	HM between 81 % and 100%

Reengineering Process Analysis, class level

- New Classes having light columns have been created
- Old classes having more than one dark box highlight still present duplications
- the general spreading of colour gives an idea of the work performed by the reengineering team



Some additional data

- Support for the reengineering process
- Support for assessing the reengineering process

- Strong reduction of time analysis
  Manual to semiautomatic:
  - 1 MM instead of 8days
  - A lot of manually non detected duplications

MWB/VM	Files	Classes	Methods
Number	7	76	589
#comparisons	21	2.850	173.166
Time CASH alg	3 s	140 s	530 s
Number of Lines	11.149	10.380	10.380
Total Byes	389.921	248.331	248.331

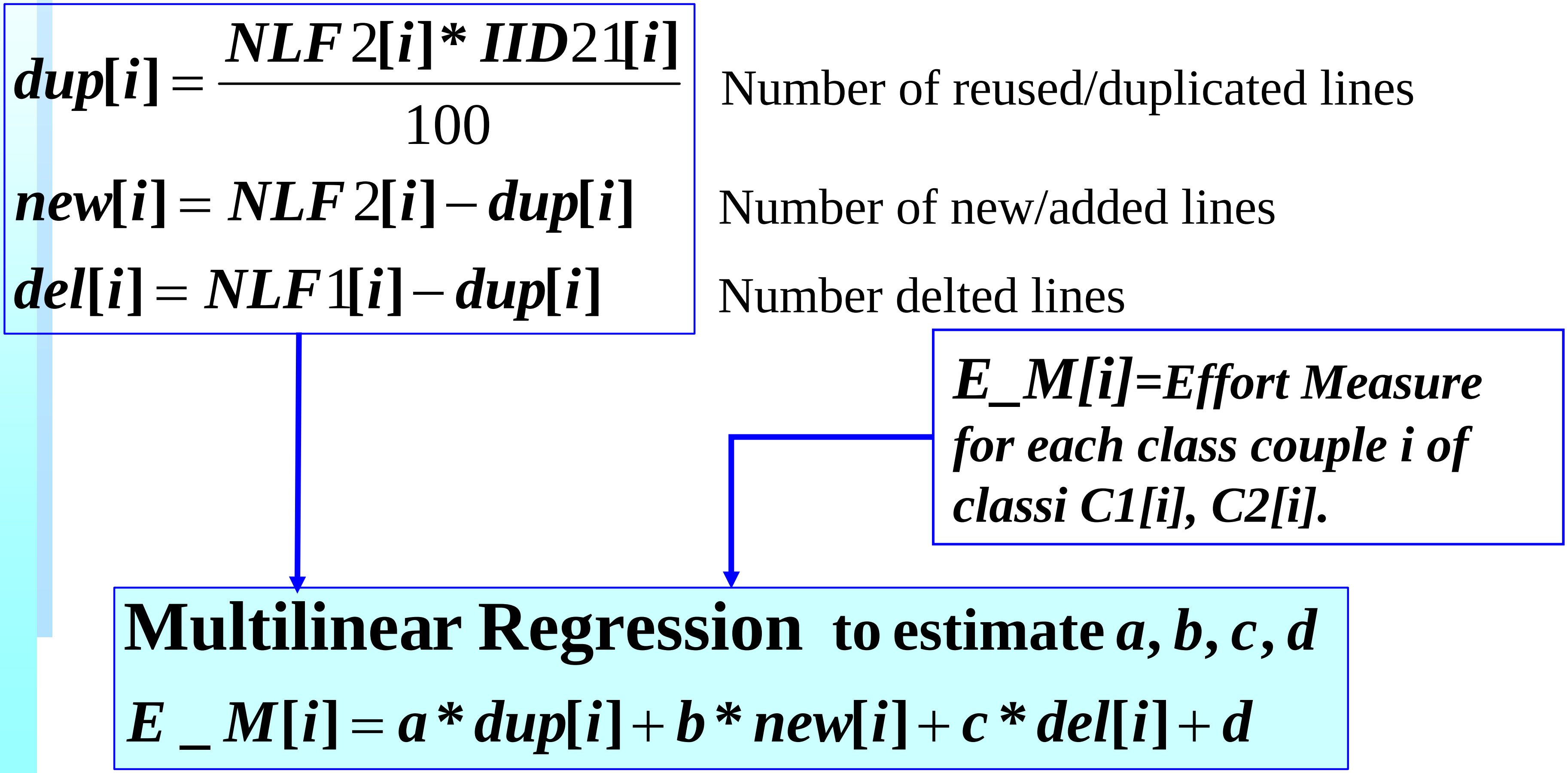
Conclusions

- **A Method and its adoption for**
  - ***duplication detection and analysis***
  - ***similarity detection and analysis***
  - ***it can be used on different languages and cases***
- **Duplication analysis of object-oriented systems**

  ➔***file, class, and method levels***
  - file level analysis is not enough to get conclusions
  - simple duplication metrics and tools are not effective, HM
- ***The tools defined includes***
  - ***algorithms for duplication and similarity***
  - ***specific metrics for the process***
  - ***suitable visualisation tools***
  - ***code reorganisation tool***

Detailed Estimation Costs

By knowing the effort of maintenance it is possible to estimate the **costs of: duplication, deletion and addition.**



$$dup[i] = \frac{NLF2[i] * IID21[i]}{100}$$

Number of reused/duplicated lines

$$new[i] = NLF2[i] - dup[i]$$

Number of new/added lines

$$del[i] = NLF1[i] - dup[i]$$

Number delted lines

E_M[i]=Effort Measure for each class couple i of classi C1[i], C2[i].

Multilinear Regression to estimate a, b, c, d

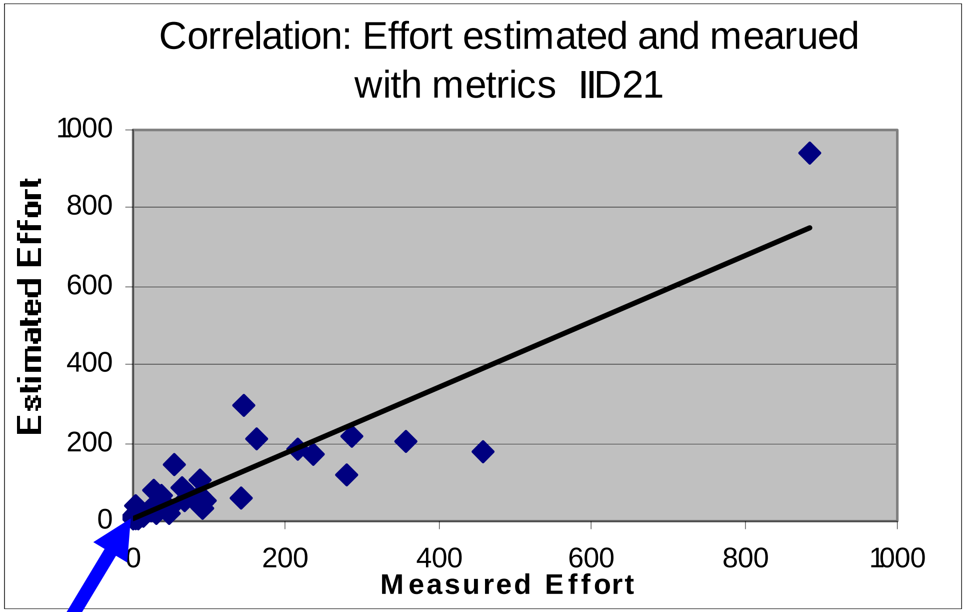
$$E\_M[i] = a * dup[i] + b * new[i] + c * del[i] + d$$

Detailed Estimation Costs a real case

1) multilinear regression analysis

R	0,858
R-squared	0,736
stderr	47,9

	Coeffi	*p-val*
d	**d = 10,68**	**0,0086**
dup	**a = 0,245**	**3.86E-18**
new	**b = 0,197**	**4.34E-15**
del	**c = 0,463**	**2.01E-16**


Correlation: Effort estimated and mearued with metrics IID21

3) Correlation Analysis

2)

Estimated Effort = a* DUP + b*NEW + c*DEL + d *N

DUP = a*Σ dup[i]	:effort of code reuse
NEW = b*Σ new[i]	:effort of code addition
DEL = c*Σ del[i]	:effort of code deletion
D = d*N	:fixed cost of code analysis