Rigorous Management and Assessment of Object Oriented Projects

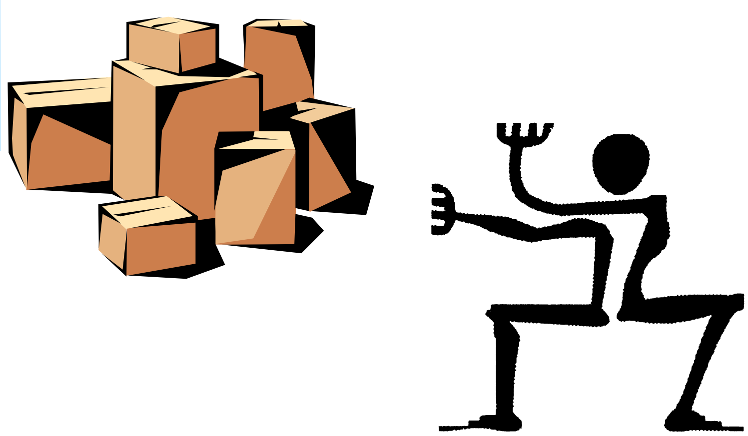
Prof. Paolo Nesi

Corso di Ingegneria del Software

Department of Systems and Informatics
University of Florence
Via S. Marta 3, 50139, Firenze, Italy
tel: +39-055-4796523, fax: +39-055-4796363
cell +39-0335-5917797
email: nesi@ingfi1.ing.unifi.it, nesi@dsi.unifi.it nesi@computer.org
www: http://www.dsi.unifi.it/~nesi

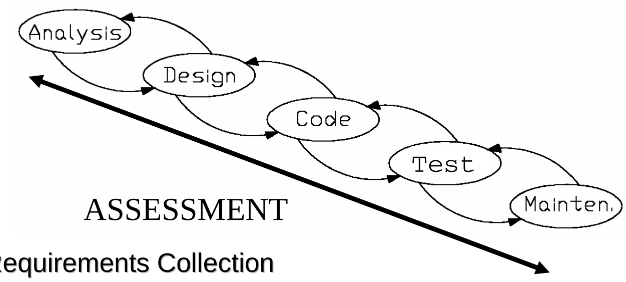
1

Object Oriented Management of Projects



2

Waterfall Life-Cycle



- Requirements Collection
- Requirement analysis
- Analysis, abstract design (composition/decomposition)
- Detailed Design (communication, HW details, behavior)
- Coding
- Testing (formal verification, simulation, etc.)
- Delivering → Maintenance

Classical Models

Structural Division of the project in SubSystems, SS

- Adoption of SubSystem Manages, SSMs.

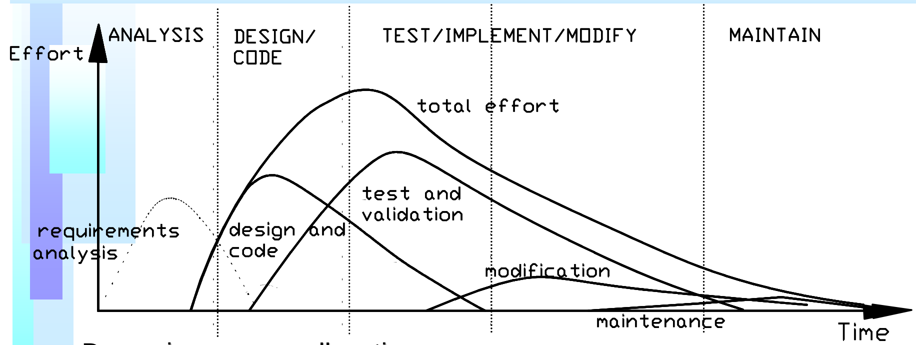
For each Subsystems

- Waterfall life-cycle: Neat Distinction among consequent phases: Analysis, design, code,……
  a sort of no-wired phone

Thus:

- Non-uniform model among phases!
- Different notation and model in different phases
- Who writes and Who reads!, who states and Who does!
- High overhead in communicating and understanding

Waterfall and Putnam’s Res. Alloc. Model

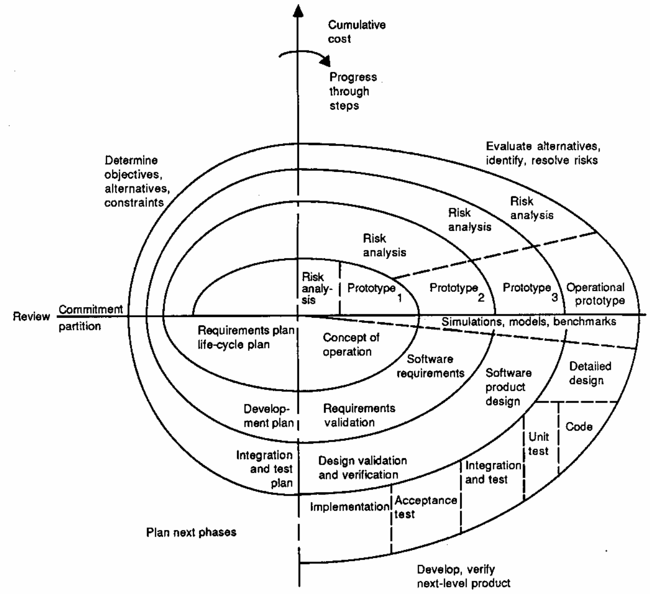


- Dynamic resource allocation
- Vertical Management
- Low flexibility, fixed roles, low gratification too well defined roles, no space for improvement and/or role changing
- Documentation per phases ….
- Low cost predictability

From Traditional to Object Oriented Projects

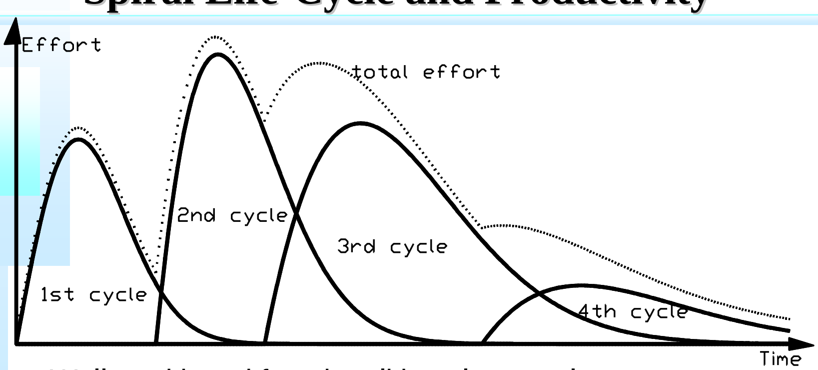
- Uniform Model along the Project Development
- Object Oriented methodologies for Analysis and Design are supported by Evolutionary Life-Cycles: Spiral, Fountain, ..
- Middle out model: both bottom-up and top-down
- Prototype Oriented
- Different roles for PM, SSM, programmers
- Different management methodologies
- Different testing techniques
- Different assessment methodologies

Spiral Life-Cycle (Boehm 1986)



 7

Spiral Life-Cycle and Productivity

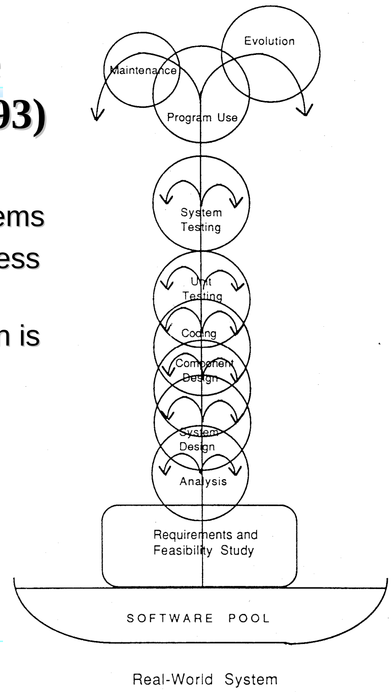


- Well partitioned functionalities along cycles
- Different goals for each cycles
- Suitable for cost control and project planning
- Dynamic allocation/deallocation, discontinuity

 8

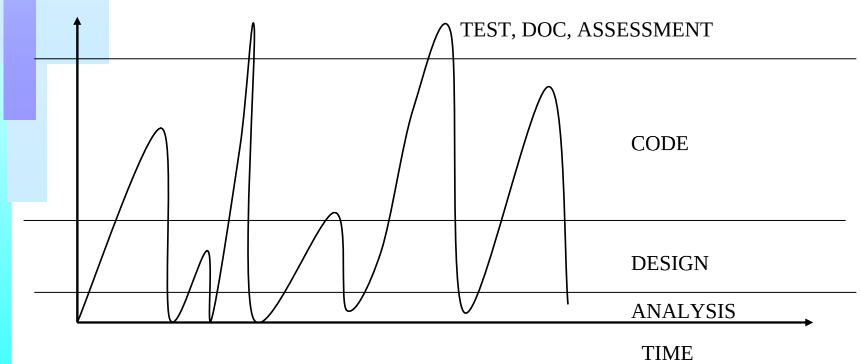
Fountain Life-Cycle (Henderson-Sellers 1993)

- Flexible reaction to problems
- Lack of well-defined process of development
- The deep of each iteration is not previously defined
- ...

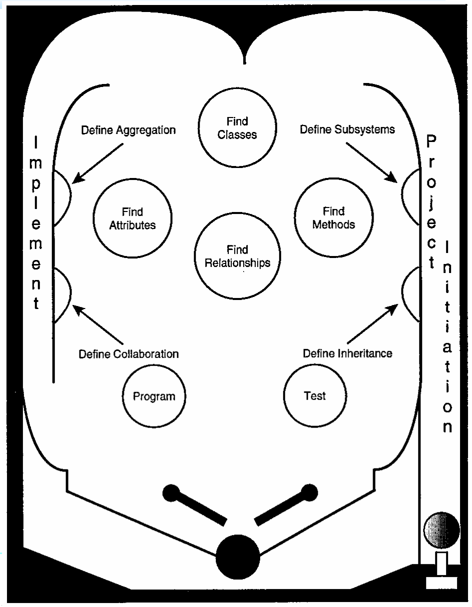


Fountain Life-Cycle and Productivity

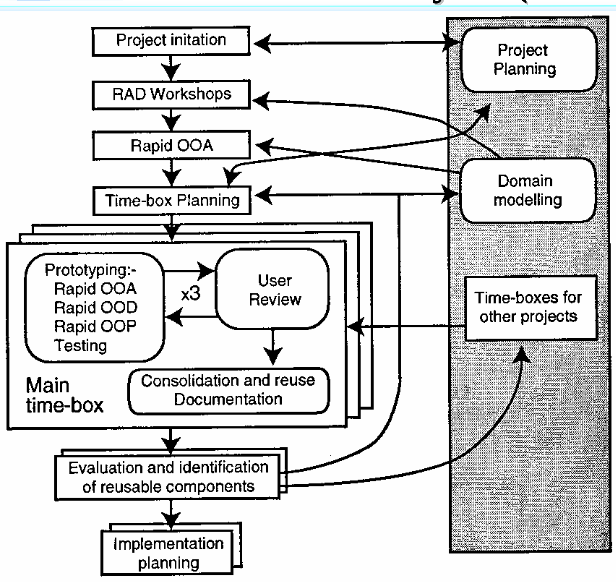
- Unpredictable time for analysis, development, test, etc.
- Unpredictable number of iterations
- Overlapped phases among subtasks
- Hard to control activities



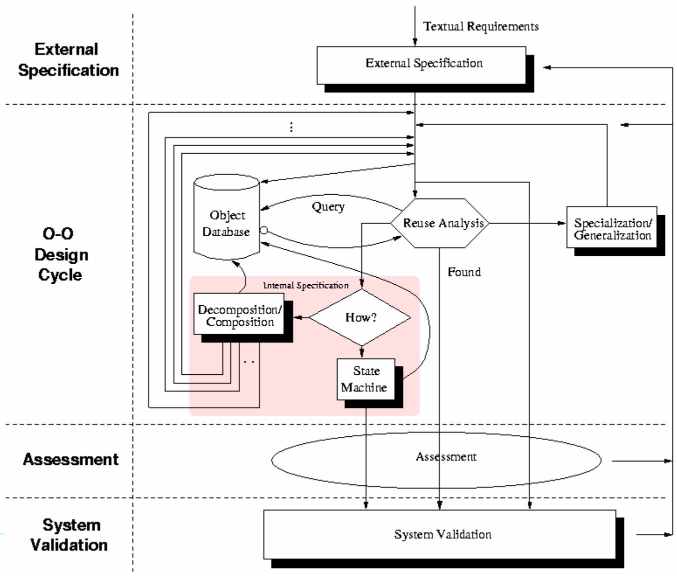
Pinball Life-Cycle (Ambler, 1994)



SOMA OO Life-Cycle (Graham, 1995)



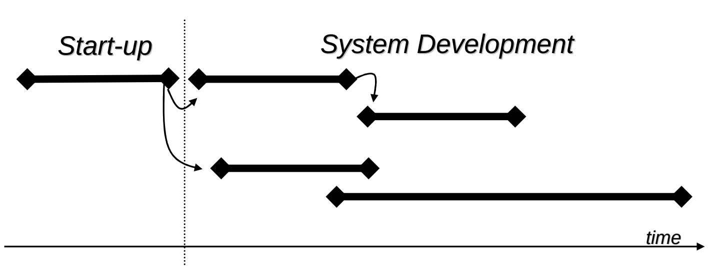
Formal OO Life-Cycle (Bruno, Nesi, 1997)



 13

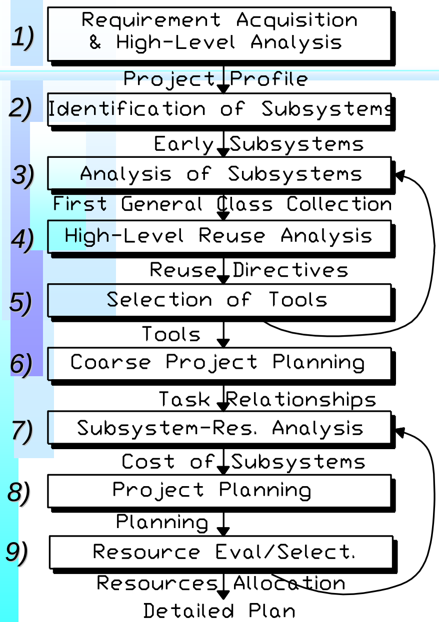
Object Oriented Projects

- Non-well defined phase division: some parts of the system can be under analysis, others under design, etc.
- New concept of SubSystem, Task
- Domain Analysis instead of Problem Analysis
- Design for reuse…
- ...



 14

Project Start-up



Mainly forward with the main products marked

- Few loops
- Two stages for the identification of main classes
- Different concept of sub-system
- Subsystem identification by refinement

 15

1) Acquisizione dei requisiti e analisi di alto livello.

- Valutazione: rischi e benefici economici, rischio tecnologico, etc.
- Requisiti vs direzione strategica dell'azienda analisi dei rischi
- Valutazione ed identificazione delle risorse a livello generale
- Identificazione target finale: end-user, ambiente tipo, piattaforma, etc.
- Definizione del profilo di qualità del prodotto (Hp. di processo di produzione in pieno controllo)
- Identificazione dei tempi e dei milestone principali

Alcuni punti sono discussi con il committente se vi e', tutti sono svolti da numero ristretto di persone con il PM;

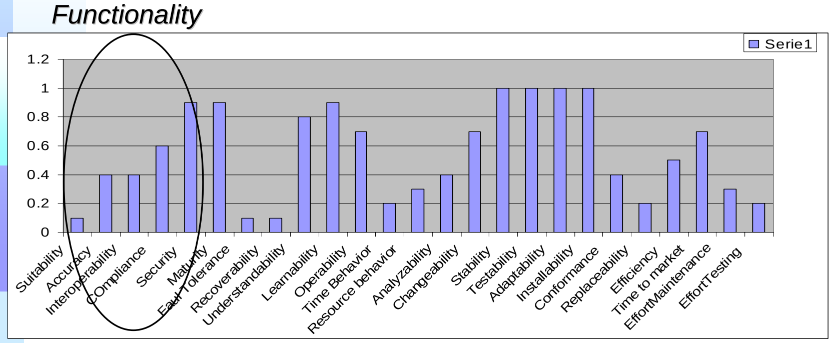
 16

Product Profile

- A set of product features and the level that has to be reached
- for example: reliability, performance, usability, gain for piece, etc.
- The levels to be reached are defined on the basis of company goals

Product Profile

Functionality



- Depending on the development phase, on the product, …
- Features related to quality, factory goals and client
- Features related to the market sector
- Features related to the client satisfactory

2) Identificazione dei sottosistemi.

- Valutazione strutturale piu' che OO del sistema PM con altri che copriranno il ruolo di SSM.
- Sottosistemi hardware e software
- Assegnazione dei sottosistemi ai SSM in base alle precedenti esperienze
- ..

D S I

19

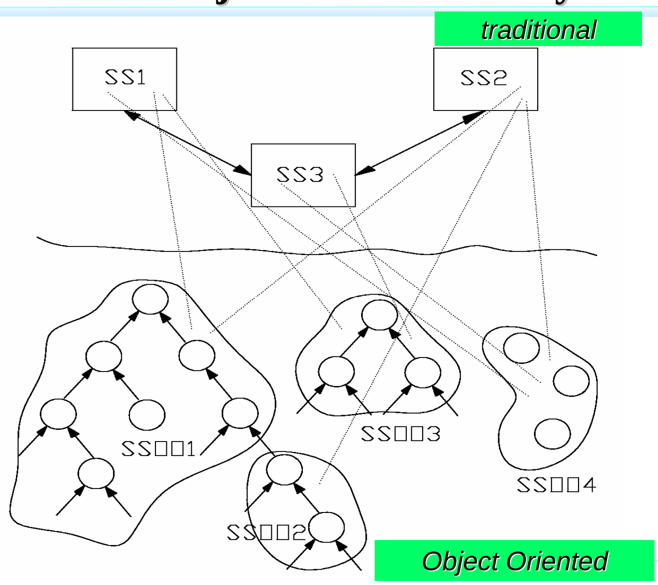
3) Analisi dei sottosistemi.

- Ogni SSM analizza il suo SS per identificare classi nel dominio
- Le gerarchie identificate derivano da viste limitate del problema
- Fusione delle varie gerarchie di classi in un'unica gerarchia
- Riassegnazione dei SS orientati agli oggetti ai SSM (massimo 15 classi per ogni SS, 3+SSM per SS, attenzione alle classi importanti del sistema: key, engine, manager in general)
- Identificazione dei cluster, suttorami, etc.

D S I

20

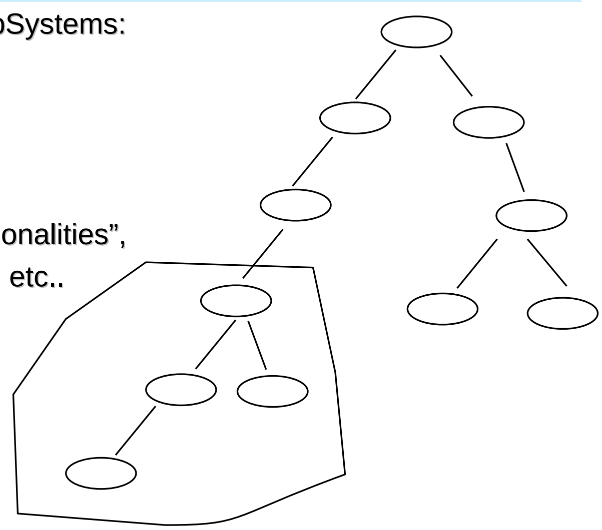
Structural vs Object Oriented Subsystems



SubSystem Projects

New concept for SubSystems:

- sub-trees,
- class clusters,
- packages,
- general libraries,
- “transversal functionalities”,
- template libraries, etc..



Key Classes and Subsystems

- Two phase process for extracting main classes
- Among these, there are the *Key classes*, typically
  - only one instance
  - complex template
  - Among them there exist *engine classes*
    - mainly active and independent ()..
- Each subsystem should have 15-30 classes to be manageable depending on their role
- A larger number leads the teams members to learn to much
- Engine and Basic Classes have to be treated with care

Engine and Basic Classes

- Have to be treated with care !
  Well implemented and tested
- Engine classes: they typically produce only one instance and this is responsible for several objects in the systems. They enforce the most important aspect of system behavior.
- Basic classes: they are very frequently instantiated. At run time most of the objects are effectively instances of these classes. A defect in that classes can lead the system in trouble.

4) Analisi di alto livello per il riuso

- Valutazione del costo di realizzazione della versione necessaria
- Identificazione delle parti da riutilizzare: librerie, classi e sottosistemi gia' acquisiti e/o realizzati
- Valutazione del costo di adattamento del riusato
- Decisione: fare/riusare, questo puo' implicare una ridefinizione dei SS

5) Selezione ed identificazione degli strumenti

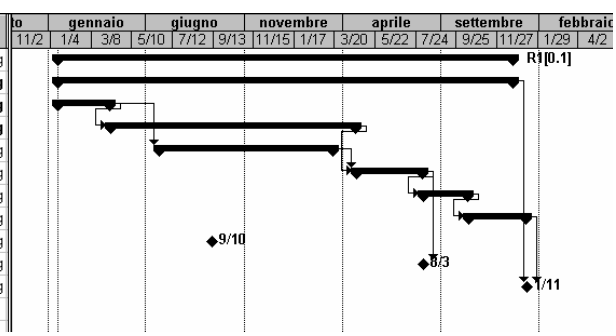
- Linguaggi, CASE tool, development tool, librerie di mercato, lex/yacc, etc. (se non imposti per contratto o specifica, o per competenze acquisite)
- Rivalutazione del rischio tecnologico in funzione delle scelte effettuate.
- Se necessario perche' il rishio e' troppo elevato rispetto alle previsioni di vendita si puo' fare restart dal punto (1) modificando alcune richieste: per esempio diminuendo i requisiti del sistema

6) Definizione del project plan grezzo, considerando:

- l'analisi generale di sistema,
- i SS, le dipendenze strutturali e funzionali SS relative,
- le deadline ed i milestone prefissati in precedenza
- la deadline finale (time to market)

Coarse Gantt

Nome attività	Durata
Management	507g
Dissemination	**507g**
Requirement analysis	**58g**
Software Development	**273g**
Hardware Development	195g
Integration	75g
Optimization	50g
Final Validation and Demonstration	65g
Milestone 1	0g
Milestone 2	0g
Milestone 3	0g



7) Analisi delle risorse per i sottosistemi

- Valutazione del costo di realizzazione di ogni SS, metriche predittive
- Eventuale ribilanciamento dei SS ai SSM e quindi del carico dei Team

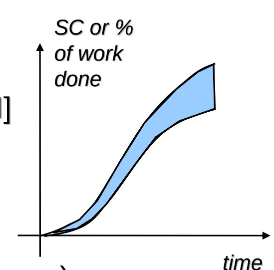
29

Predicting Effort Needed

Early Effort Prediction

Effort = #KC *k* HPC

where HPC = [15, 40] Hours Per Class
k = [2,4] (scale factor) [non GUI, GUI]



Typically we got

Efficiency = [2.2,4] SC_{LOC} (points per hours)

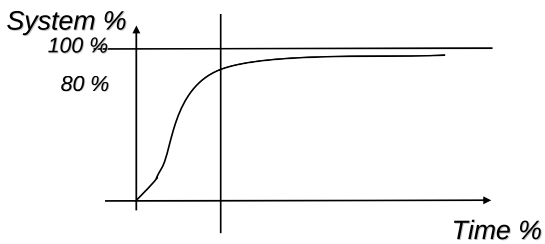
including:

Analysis, design, coding, documentation, assessment, integration, test, meetings.

30

La Legge dell'80-20

- For producing the 80 % of the final system is needed only the 20 % of the total effort.
- Thus, the final 20% for completing the system will cost about 80 % of the total cost, 4 times the costs covered for arriving at the 80 %. In the figure, a constant number of people involved in the development has been supposed.
- Presently this ration 80/20 is going towards 65/35.



Predicting Effort

- A total of 480 hours to produce from 12 to 32 classes
- 12 in the case in which complex key classes are included.
  - Key classes have to be considered 2 times more complex of typical classes
  - Engine classes have to be considered 5 times more complex of key classes
  - This does not means that the K and E have a correspondly high number of LOC (the CC takes into account several factors, as it will be show later)
- Teams with 2-3 people with the SSM full time or at a given percentage 30<X<100

Effort and Efficiency/Productivity

- Effort = [months] | [days]
- Productivity = (Size | Volume | Complexity) / Effort
- Efficiency as productivity
- Quality = <feature> / (Size | Volume | Complexity)
- Unit Cost = Euro per (Size | Volume | Complexity)
- …
- ..
- .

DSI

33

8) Realizzazione del project plan

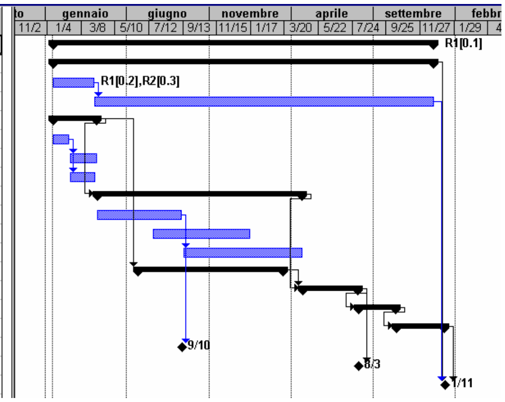
- Utilizzo del Gantt grezzo con i dati del punto (7)
- Identificazione delle sotto attivita'
- Identificazione dei costi per: consulenza, training, licencenze (tools), viaggi, beni di consumo, strumenti/apparecchiature (con piano di ammortamento), etc.
- Il training, puo' dare luogo a task separati

DSI

34

Gantt Diagram

	Nome attività	Durata	Lavoro	Nomi risorse
1	Management	507g	405.6h	R1[0.1]
2	**Dissemination**	**507g**	**220h**	
3	Preparation of dissemination	55g	220h	R1[0.2],R2[0.3]
4	real Dissemination	452g	0h	
5	**Requirement analysis**	**58g**	**0h**	
6	General	22g	0h	
7	Software Requirements	36g	0h	
8	hardware Requirements	34g	0h	
9	**Software Development**	**273g**	**0h**	
10	SS1	114g	0h	
11	SS2	130g	0h	
12	SS3	159g	0h	
13	Hardware Development	195g	0h	
14	Integration	75g	0h	
15	Optimization	50g	0h	
16	Final Validation and Demonstration	65g	0h	
17	Milestone 1	0g	0h	
18	Milestone 2	0g	0h	
19	Milestone 3	0g	0h	



 35

9) Valutazione e selezione delle risorse

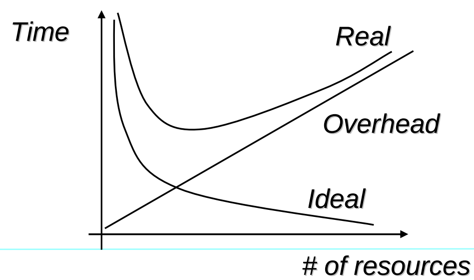
- Per ogni SS: composizione del team in base alle competenze delle persone e alla loro compatibilita'
- Realizzazione di team integrati: analisi, design, code, valut., etc., competenze trasversali.
- Allocazione delle risorse in base alle deadline

 36

Team's Dimension

- Team dimension has to be maintained low
- Overhead costs for
  - knowing the system, sharing code
  - exchanging information, and

  are high even in Object Oriented systems
- The minimum changes depending on the analysis and on the system on the PM, on the Management, etc.



System Development

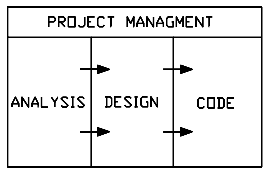
- Uniform Object Oriented model along the Life-Cycle
- Specific OOA and OOD Methodologies

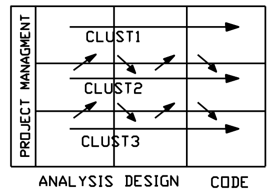
➔Complexity shift towards analysis

High Efficiency if:

- Horizontal Management
- Constant allocation model
- ..

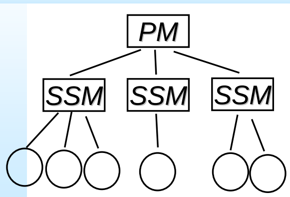
From Vertical to Horizontal Management

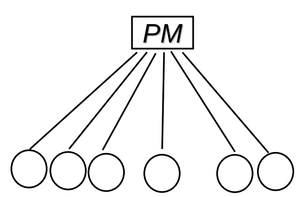




- Different role/skill of PM and SSMs
- Global view of development for all team members
- Flexible roles, fast reaction to personal problems
- Higher responsibility and frequent gratification
- Easier integration among SubSystems
- Easier control, predictability
- Static resource allocation, lower overhead
- ..

Hierarchical vs Flat Structure/Organization





- Delegation
- More controllable development
- Higher costs
- More motivations
- Well defined roles
- Less competition
- ..

- Too weight leaf classes
- Code repeated
- Poor Hierarchical organization
- Poor Design
- Non well-defined roles
- More competitive society
- ..

Macro Life-Cycle

Model:

- Spiral Macro-Cycle and Spiral Micro-Cycle,
- Spiral Macro-Cycle and Fountain Micro-Cycle,
- Spiral Macro-Cycle and micro-optimized-cycle.

- Macro Cycle of 5-8 months
- Well defined goals for each Macro Cycle;
- Last cycles of the Macro-Cycle should include more attention to:
  - Integration, Optimization, demonstration and validation;

Spiral and Fountain

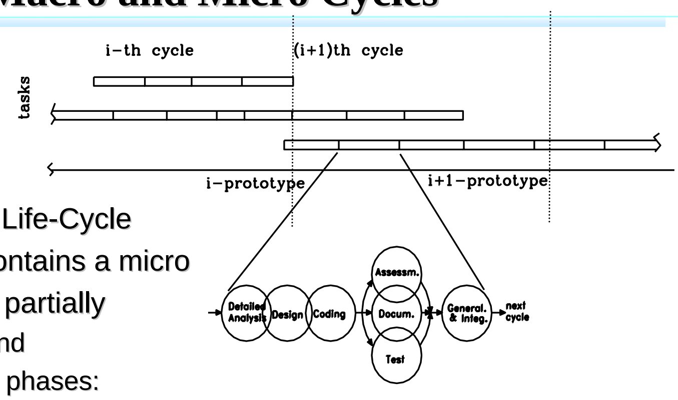
Spiral

- Too complex and complete for the Micro level
- Too expensive for the micro level
- ….
- ..

Fountain

- Too few formalized and controllable for Macro level
- Not enough controllable to be used in 3 people team
- ….
- ..

Macro and Micro Cycles



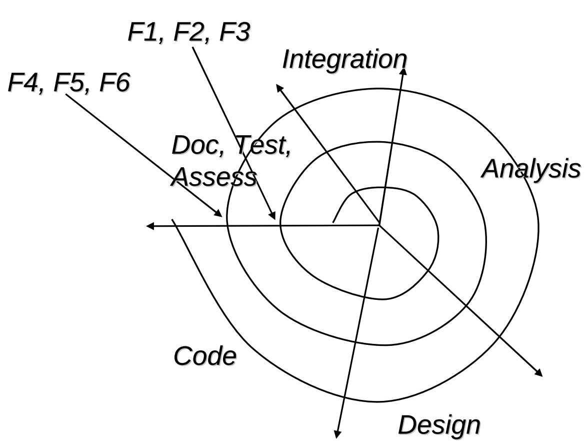
- Macro: Spiral Life-Cycle
- Each Cycle contains a micro
- Each micro is partially
  - Fountain, and
  - has parallel phases:
    - Assessment
    - Test
    - Documentation

© Paolo Nesi 1995-2000

43

Spiral Macro Cycle

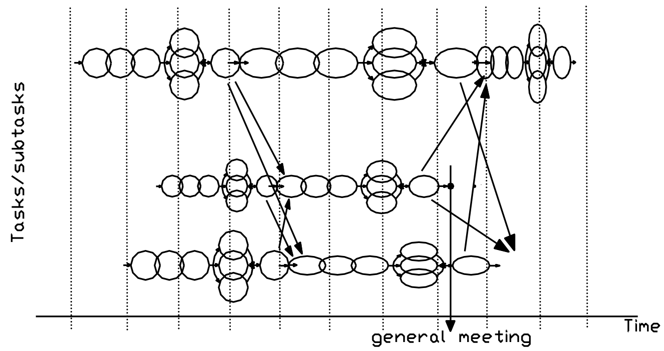
- Well defined Functionalities for each spiral, early defined



© Paolo Nesi 1995-2000

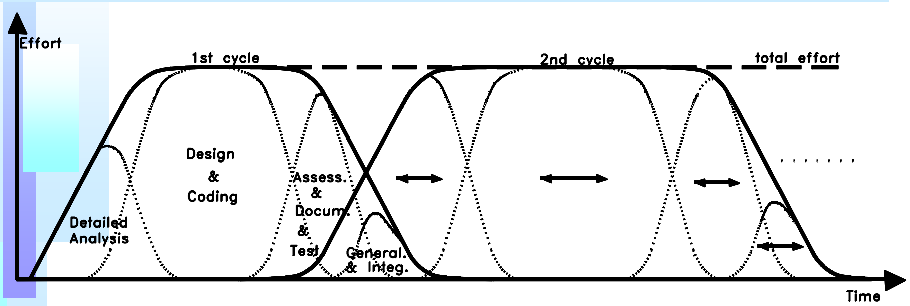
44

Task Relationships



- 3-4 Cycles (spiral) (micro-cycle relationships)
- bi-Weekly meetings
- Monthly general meetings or when needed

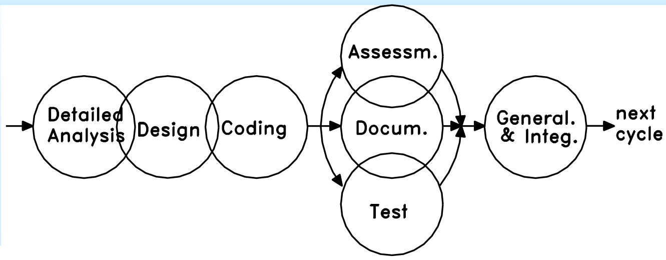
Effort Planning



Constant number of people, no allocation/deallocation

- Low training costs during the life-cycle
- Small Sub-tasks/sub-systems, 3-4 members
- A SubSystem Manager (SSM)
- A Project Manager (PM)

Micro Optimized Cycle (Nesi 98)



- Modified Spiral-Fountain model
- Partially overlapped phases
- Integrated competencies in team
- Parallel: assessment, documentation and test
- Generalization and Integration phase for redistribution of code along classes and sharing information with other teams

Fase di Test

- Utilizzo di Test script, tool Capture & PlayBack, regression testing
- Definizione dei test anticipata in base ai requisiti
- Il test a questo livello dovrebbe essere fatto tipicamente da chi ha fatto il software visto che e' il solo che può verificare se le funzionalità che voleva introdurre sono state effettivamente introdotte. Alcune funzionalità generali se raggiunte devono essere controllate dal SSM.
- Questo test non deve essere confuso con la fase di test vera e propria durante la quale si effettua una validazione del prodotto. In tale caso il test deve essere eseguito da persone esterne. Queste utilizzeranno i vettori di test messi a punto durante i test alla fine di ogni micro-ciclo più quelli che derivano dai requisiti generali del sistema.

Fase di Documentazione

- Generazione incrementale, struttura definita, persone diverse, controllo del SSM e del PM.
- Decisioni operate, cose rimaste in sospeso
- Descrizione generale da parte del SSM, documento contrattuale e del PM

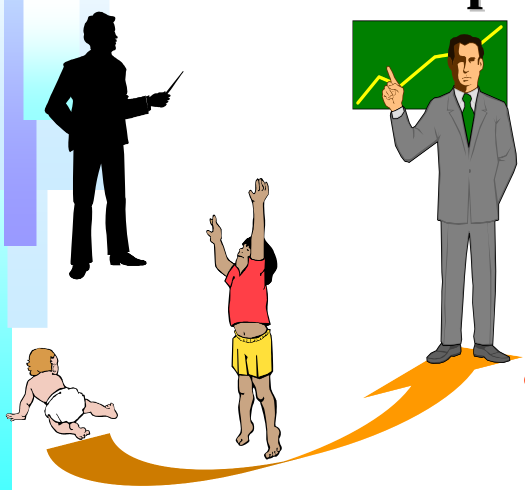
- Questo tipo di attività deve essere svolta dai vari componenti del team. La parte di architettura, di relazioni con altri task/team e di analisi dal SSM. La documentazione relativa ai dettagli realizzativi da chi ha sviluppato il software vero e proprio.

Generalizzazione e Integrazione

- Integrazione e distribuzione dei risultati fra SottoSistemi, altri task, altri team
- General meeting con il PM (ogni 3-4 periodi)
- Identificazione dei problemi e definizione delle azioni

- Questa attività può portare anche ad effettuare meeting specifici con altri team.
- L'attività di comunicazione con altri team e' svolta dal SSM si per acquisire informazioni da altri che per fornirle. Nel primo caso tipicamente si ha che il team che fornisce informazioni o moduli software presenta l'attività svolta a quello che la dovra' utilizzare

Managing with Continuos Improvement



- Learning:
  - Facts
  - Understanding Facts

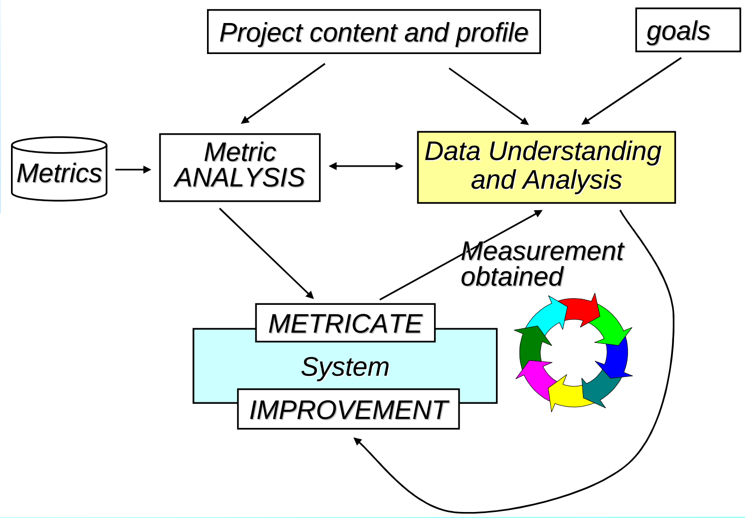
51

Process and Product Assessment

- Process Assessment:
  - Evaluation of the development process
  - Evaluation of the quality and the efficiency of the development process
  - definition of the compliant with the ISO 9000
  - .
- Product Assessment:
  - Process by which some features of the product are evaluated
  - Typical features are those derived from the product profile
  - ..
  - .

52

Continuous Improvement



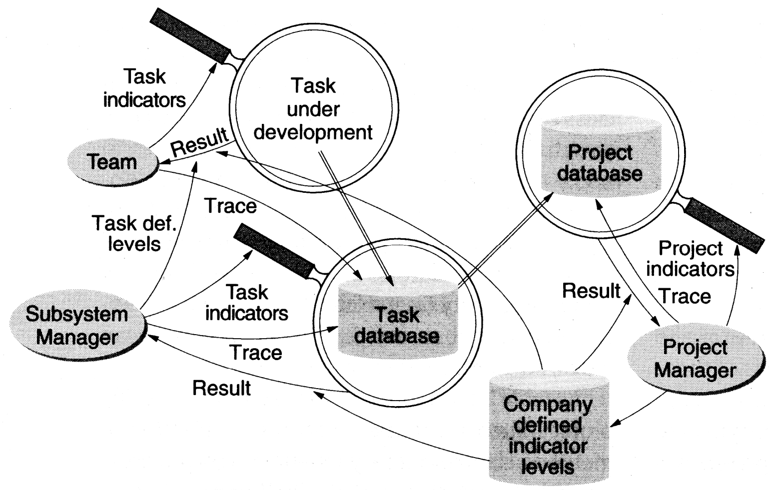
 53

Process Evolution

- A model considering both the development phase and the context:
  - process assessment (questionnaires)
  - define goals and plan to reach them
  - identify the rationales for each success and failure
  - assessing the product during the process improvement
- thus, to reach the process identification

 54

Controlling Development Process



55

Product Assessment Context

- Product Assessment, according to the
  - product context:
    - structure (GUI, embedded, real-time, ...)
    - application field (toy, safety critical, …)
    - development tools and languages
    - development team features
    - libraries
    - OOA, OOD, methodologies
    - assessment tool
    - ...
  - development context
    - development phase

56

Valutazione di Sistemi

Il problema e' equivalente a contare gli elefanti in africa.

- Il matematico prende l'insieme africa e per intersezione estrae da questo uno ad uno tutti gli insiemi non intersecanti: alberi, leoni, serpenti, erba, deserto, ... applica l'operatore conta a quello che rimane.
- L'informatico definisce il tipo elefante e su questo un insieme di assiomi per l'identificazione delle aree in cui possono esserci elefanti. Poi applica l'operatore cerca a tali zone e ne fa l'unione.
- Il fisico prende l'ecosistema africa, dopo certe osservazioni, ne fa un modello, e valuta sulla base di simulazioni del modello quanti elefanti possono viverci.
- L'ingegnere definisce un algoritmo:
  1) si vada in africa con un elefante di riferimento,
  2) si percorra da sinistra a destra dal basso in alto,
  3) ogni volta che incontro un animale lo confronto con quello di riferimento,
  4) se identici allora incrementa di 1.
- Il programmatore prende l'algoritmo dell'ingegnere, inizializza il numero degli elefanti a 0, definisce delle soglie per riconoscere o meno l'elefante, pone come criterio di arresto l'arrivo al Cairo.
- Lo statista effettua un campionamento in diverse aree alle quali applica l'algoritmo dell'ingegnere e quindi estrapola ......
- Il manager valuta i costi e i benifici aziendali dello sviluppo e dell'uso degli algoritmi proposti considerando l'errore di stima, i tempi di stima, ...

59

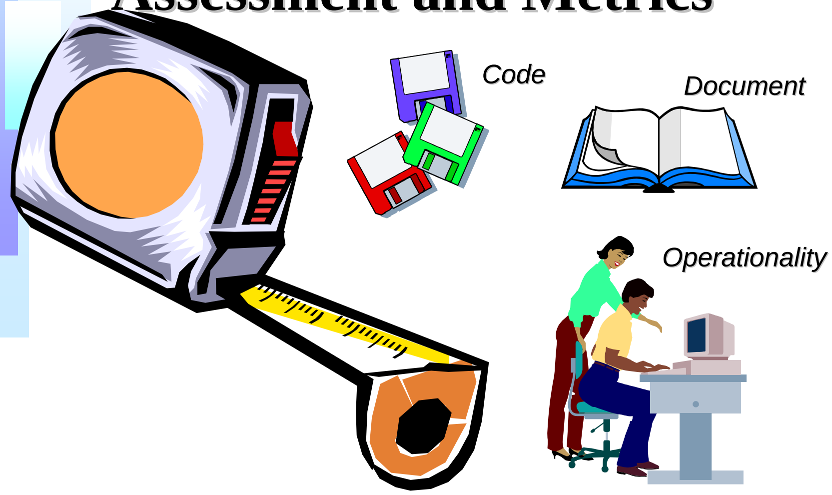
Software Assessment

Prof. Paolo Nesi
Corso di Ingegneria del Software

Department of Systems and Informatics
University of Florence
Via S. Marta 3, 50139, Firenze, Italy
tel: +39-055-4796523, fax: +39-055-4796363
cell +39-0335-5917797
email: nesi@ingfi1.ing.unifi.it, nesi@dsi.unifi.it nesi@computer.org
www: http://www.dsi.unifi.it/~nesi

60

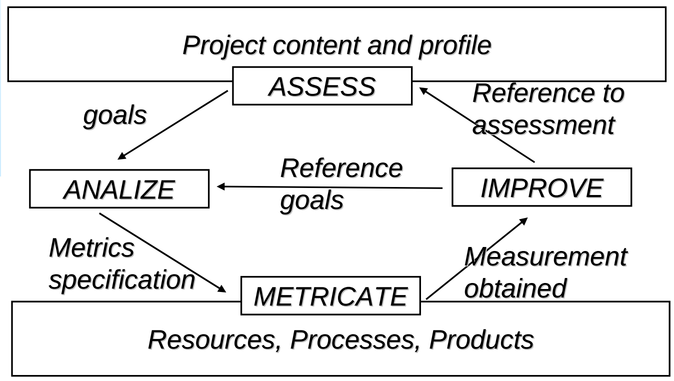
OO Products: Assessment and Metrics



Why to Measure?

- Solo quello che puo’ essere misurato puo’ essere mantenuto sotto controllo
- Prediction
- To maintain under control
- To optimize the process of development avoiding wrong and/or unuseful expenses
- To reduce the costs
- To reduce the time to market
- To evaluate the costs of production

Continuous Improvement



Metrics

- Metrics
- Metric Classification Features
- Direct and Indirect metrics
- Estimation and prediction Metrics
- Quality and Metrics

Metric

- A metric is a measurement method with a defined measurement scale
- Specific conversion between a metrics to another
- Specific scale for each metric focussed on a specific feature

D S I

 65

What can be Measured?

- Quality
  - Reliability, Functionality, testability,
  - portability, compatibility, compliance,
  - ..
- Effort of (human resources)
  - development, Maintenance,
  - Testing, assessment,
  - documenting, porting,
  - ..
- In several cases Quality and Effort are in some way correlated

D S I

 66

Direct/Indirect Metrics

- direct metrics should produce a direct measure of parameters under consideration; for example, the number of the Lines of Code (LOC) for estimating the program length
- Indirect metrics are usually related to high-level characteristics; for example, the number of LOC is typically related to development effort
- the same measure can be considered as a direct and/or an indirect metric depending on its adoption.

Direct Metrics

- Size Metrics:
  - LOC: number of Lines of Code
  - number of language tokens
  - number of functions, operators, statements, modules,
  - number of comments, ...
  - number of ';' or 'CR' or 'LF' or 'begin' .....
- Complexity
  - Data Structure: number of variables, types
  - Logic: number of cycles, etc.
  - Computational: asymptotic, ..
  - Number of Nesting levels
  - Interface
  - Cognitive, psychological

Direct Metrics

- Functionals
  - number of functionalities
  - number of level of procedure call
  - number of use cases
  - number of sections in the manual
  - number of paragraph in the textual requirements
  - number of defects
  - ..
- In/Out
  - number of inputs, outputs
  - number of external variables
  - number of files
  - number of communication channels
  - number of buttons
  - ..

Direct Metrics

- Cohesion/Coupling
  - number of calls for procedure
  - number of external variable of module
  - number of external variable of a function
  - number of calls out of the module
  - ..
- Object Oriented Structure
  - Inheritance hierarchy: balance, ramification, etc.
  - Distribution of metrics on the hierarchy
  - ..
  - .

Indirect metrics

- have to be validated for demonstrating their relationships with the corresponding high-level features.

(i) evaluating parameters of the metrics (e.g. weights and coefficients),

(ii) verifying the robustness of the identified model against several real cases.

- The model can be linear or not, and it must be identified by using both mathematical and statistical techniques

Indirect Metrics

- Indirect metrics have to consider the different scale:

M = w <direct metric>

w   has to correct the metric dimension.

- Metric should be normalized:

$$\mathbf{0 \leq Mi \leq 1}$$

- Thus they can be compared with the values produced for other systems

Composite Metrics

- Most Indirect metrics are composite metrics.
- These are derived as the composition of other direct, indirect metrics with different scales:

$$CM = w1\ M1 + w2\ M2$$

w1, w2 are used to correct the metric dimension and should be defined or estimated during validation.

- In this case a linear dependency has been supposed !
- Composite Metric should be normalized: $\mathbf{0 \leq CM \leq 1}$ or referenced to the features under estimation scale.

 73

Non Linear Composite Metrics

- For example:

$$CMx = w1 \operatorname{Log} M1 + w2 \operatorname{Log} M2 + w3\ M3^{M4}$$

- Logarithmic, exponential and polynomial relationships are typically established on the basis of rationales
- weights w1, w2, w3 have to be
  - estimated during validation process or suitably defined
  - dimensioned in order to produce a corrected scale for CMx metric

 74

Non Linear Metric for Effort Evaluation

- Typically for the development Effort:

  Effort = k size^b

- *k* and *b* are
  - estimated during validation process or suitably defined
  - dimensioned in order to produce a corrected scale for estimating the Effort

Functional, Behavioral and Structural Metrics

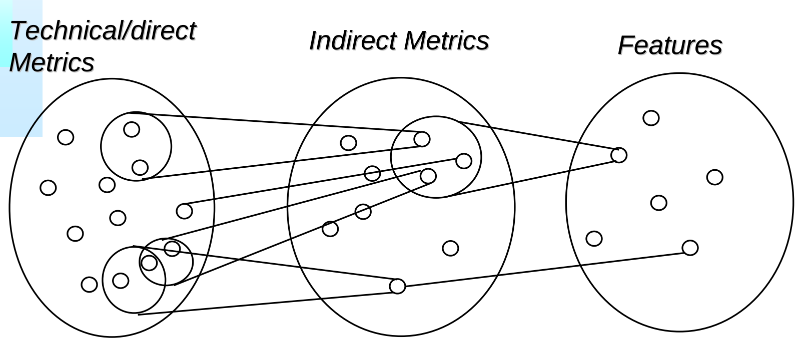
- Functional (data transformation):
  - volume metrics,
  - operands counting,
  - # operators, ...
- Behavioral (system behavior):
  - reactivity, flow diagrams,
  - logic metrics, computational complexity,
  - nesting levels, etc.
- Structural (system structure):
  - data structure complexity
  - #number of variable, variables domains
  - data relationships, ...

Technical/Cognitive/Process Oriented

- **Technical view** refers to the software engineering aspects of system specification (size, complexity, etc.);
- **Cognitive view** takes into account the external understandability and verifiability of the system;
- **Process-Oriented view** refers to the system aspects that are influenced by or can influence the process of system development (productivity, reuse, cost of development, cost of maintenance, etc.).

 77

Metric Relationships



 78

Metrics of each view, different phases of system evolution

- **cognitive metrics** during system development and/or in system maintenance,
- **technical metrics** for the evaluation/certification of some specific characteristics of the system;
- **process-Oriented metrics** for evaluating the impact of technology on the whole development process.

79

Analysis, Design and Coding Metrics

Metrics oriented to maintain under control the specific phases of the development.

- Analysis:
  - complexity (more structure oriented than functional),
  - # of variables, # of functionalities, # of subclasses,
  - # of attributes, # of methods, # of superclasses, # of classes, ….
- Design: all the above metrics + functional metrics,
  - reuse metrics, ...
- Coding: all the above metrics + style metrics
  - (# of goto, # of global variables, ...), …

The same metrics can be used along the whole development life cycle with different references.

80

Development, Maintenance and Reuse Metrics

- Development metrics:
  - specific metrics oriented to the specific phases of the development
- Maintenance metrics:
  - effort for Maintenance, cognitive metrics,
  - reuse metrics, complexity, effort for reuse,
- Reuse Metrics:
  - # of comments,
  - complexity of the class definition with respect to class implementation,
  - ratio between number of comments and the complexity

D S I

81

Consumptive Metrics

For instance:

- Size per function --e.g.--> total size / number of function
- Size per module
- Total size
- Volume per function
- Complexity per module
- Effort per function

Can be estimated by

- summing Σ or
- ratio between other values

Typically reflect averaged values

D S I

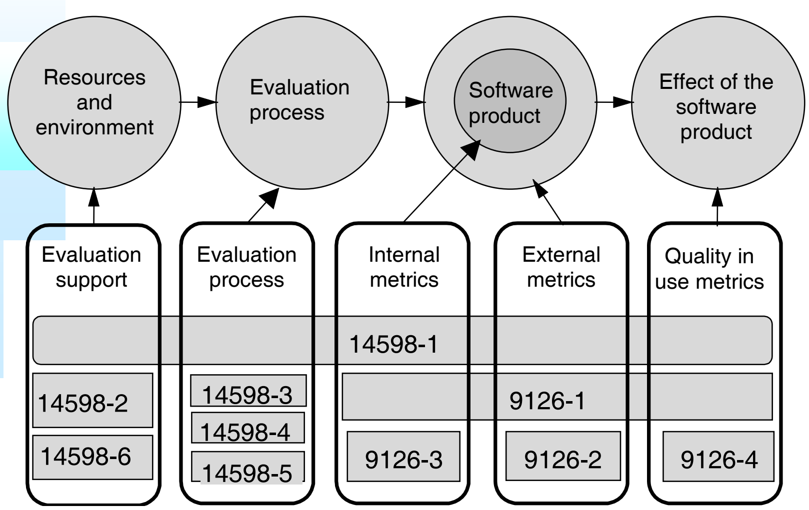
82

Estimation and Prediction Metrics

- **Estimation metrics** try to evaluate the value assumed by a system feature in the time instant and context in which they are applied

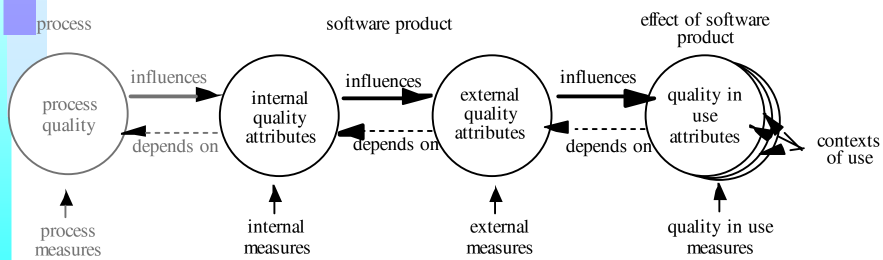
- **Prediction metrics** try to evaluate the value that will be assumed by the system feature in the future. For example, the prediction of number of LOC after 5 months by knowing the absolute values and the variation of LOC measured in the last two weeks.

Quality ?



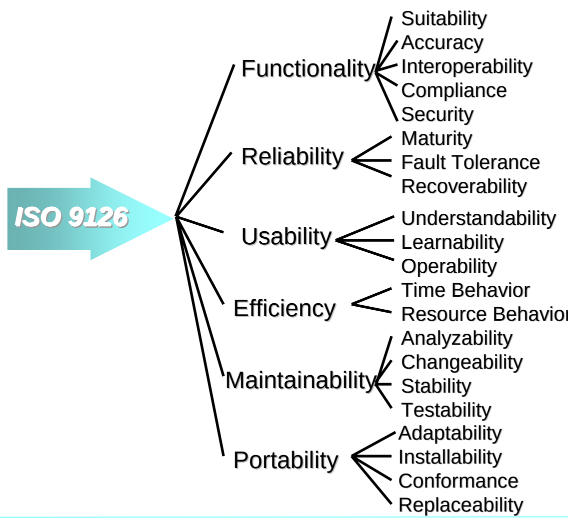
ISO 9126 Relationships and Quality Models

- The adoption of process measures may produce effects on internal parts
- The adoption of internal measures influences the external quality ….
- ….



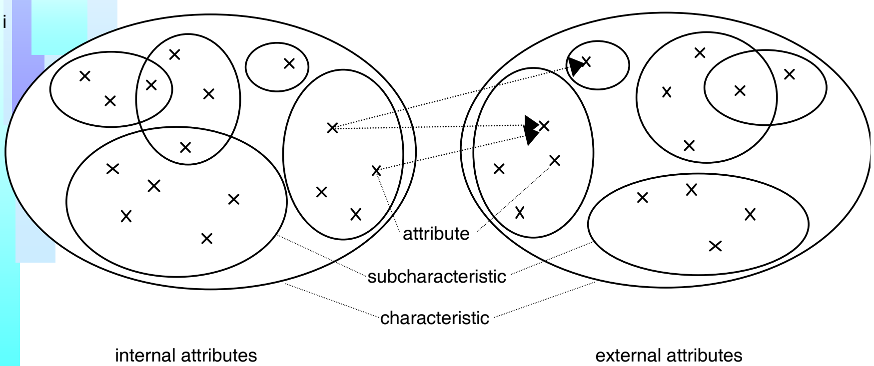
85

ISO 9126 (1991) Internal/External Features



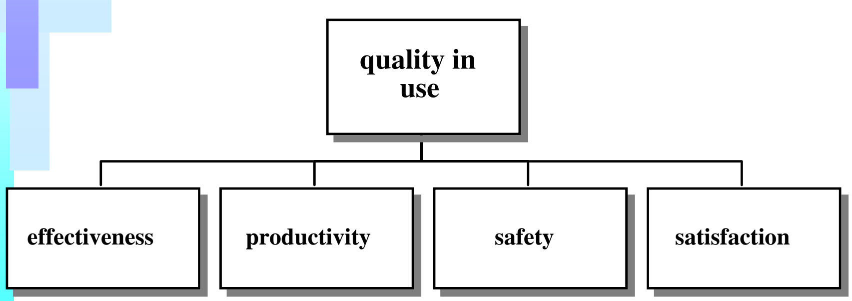
86

Mapping -- Internal/External Features



Quality in Use

- For each Users
- For each role of the product
- ..
- .



Standards

- Do not provide direct solution
- Do not provide metrics to be used
- Give definition for features to be evalutated
- Define a framework for organizing metrics and their selection
- Suggest rating levels

- Product: ISO 9126 series, IEEE 982, ...
- Process: ISO 9000 series, DOD 2167A, …

- They are growing ………..

89

OUTLINE

Product Assessment and Metrics

- Assessment and development process
- System assessment problems, process improvement
- Object Oriented vs quality
- Classification of OO Metrics
- Method Level Metrics
- Class Level Metrics
- System Level Metrics

90

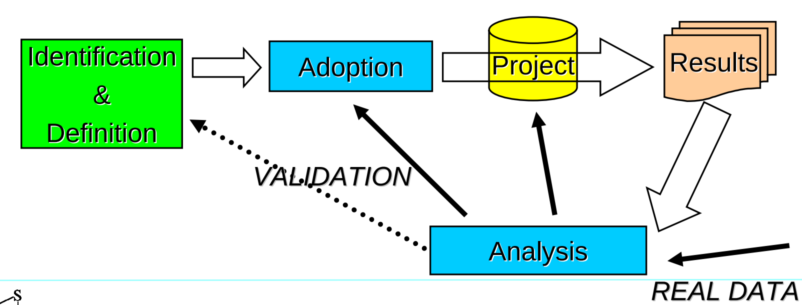
Assessment and development process

- Why?: Effort, Quality, Maintenance, Reuse, ….
- When?: along the development process, after developed
- Where?: in, out, unit, ...
- Who?: SSM, PM, ...

- What has to be done?
- How?

System Assessment Problems

- identification and definition of "result analysis mechanisms"
- adoption of "result analysis mechanisms"
- Result Analysis and understanding
- application of corrections (actions defined)



Metric Adoption

Metric Estimation by using suitable tools

Understanding Results by using :

- Reference Values with respect to confidence values
- Significant Trends with respect to reference trends
- Significant Profiles with respect to attended trends

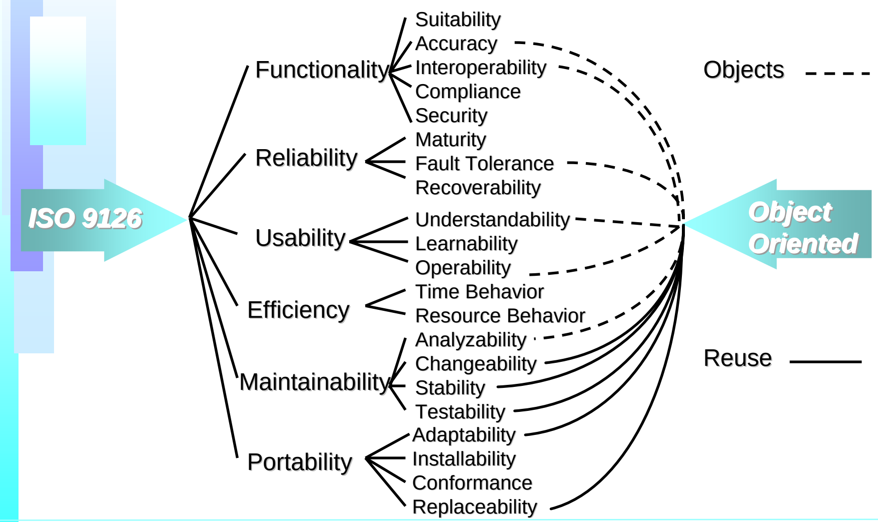
Object Oriented vs quality

In OO Part of Development Complexity is shifted from design and coding towards analysis.

Thus it passes from functional and computational toward structural aspects

- OOP doesn't mean quality, but OOP aids to reach a certain quality compliance
- Some phases are necessary to estimate quality compliance

Quality versus Object Oriented



 95

Metric Classification

Metric	A/D/C	OOP	M/C/S	V/C	C/Q	Rel	P/A	F/B/S	T/C/P	E/M/R
Ha	C	N	M,C,S	V	---	---	A	F	T	E,M
LOC	C	N	M,C,S	V	---	---	A	F	T	E,M
Mc	C	N	M,C,S	C	---	---	A	F,B	T	E,M
MC	C	N	M	V,C	Q	---	A	F,B	T	E,M,R
CC	A,D,C	Y	C	V,C	C,Q	P,I	P.A	F,B,S	T	E,M,R
CM	C	Y/N	C	V,C	Q	---	P.A	F,B	T	E,M,R
HSCC	D,C	Y	C	V,C	C,Q	P,I	P.A	F,B,S	T	E,M,R
NAL	A	Y/N	C	V	Q	P	P.A	S	T	E,M,R
NAM	A	Y	C	V	Q	P,I	P.A	S,B	T	E,M,R
Size2	A	Y	C	V	Q	P	P.A	S,B	T	E,M,R
TJCC	D,C	Y	C	V,C	C,Q	P	P.A	F,B,S	T	E,M
WMC	C	Y	C	C	Q	---	A	F,B	T	E,M,R
CCGI	A,D	Y	C	C	C,Q	P,I	P.A	B	P,C	E,M,R
DIT	D	Y	C	---	C,Q	I	P.A	S	P,T,C	M,R
NOC	A,D	Y	C	---	C	I	P.A	S	P,T,C	M,R
NSUB	D	Y	C	---	C,Q	I	P.A	S	P,T,C	M,R
NSUP	D	Y	C	---	C,Q	I	P.A	S	P,T,C	M,R
CBO	C	Y	C	V	C,Q	P	A	B,F	C,T	M,R
NKC	A	Y	S	V	C	---	P	S	C,T	E
SC	A,D,C	Y	S	V,C	---	P,I	P.A	F,B,S	T	E,M,R
T	C	N	S	V,C	---	---	A	F,B	T	E,M

The metric framework has to be defined and validated against several projects

Where:
- **A/D/C** **A**nalysis, **D**esign and **C**oding; **OOP** object-oriented suitability; **M/C/S** **M**ethod, **C**lass and **S**ystem level; **V/C** **V**olume and/or **C**omplexity metric; **C/Q** **C**onformity to OOP and/or **Q**uality; **Rel**ationships: is-**P**art-of, **I**nheritance; **P/A** **P**redictive and/or **A** Posteriori; **F/B/S** **F**unctional, **B**ehavioral, **S**tructural aspect; **T/C/P** **T**echnical, **C**ognitive, **P**rocess-oriented metric; **E/M/R** **E**ffort, **M**aintenance, **R**euse.

 96

Object Oriented Metrics

Method Level Metrics

Method Complexity, Method Size, # local variables, # external var, cohesion with other methods, McCabe, Halstead, etc.

Class Level Metrics

Class Complexity, Class Cohesion, Class Sub, Class Sup, # attributes, # methods, # constructors, WMC, Size2, etc.

System Level Metrics

classes, # roots, system complexity, DIT, # Glob Var, Mean class complexity, mean NAM, etc.

Method Level Metrics

- Classical Functional Metrics Can be used:
  - Mc Cabe Ciclomatic Complexity
  - Halstead Volume Metric
  - LOC
  - Metrics Size based on Token counting
  - etc.
- Functional Metrics result more precise if they are integrated with data structure and reuse metrics:
  - Number of variables
  - Number and complexity of procedure parameters
  - ...

Mc Cabe, Ciclomatic Complexity

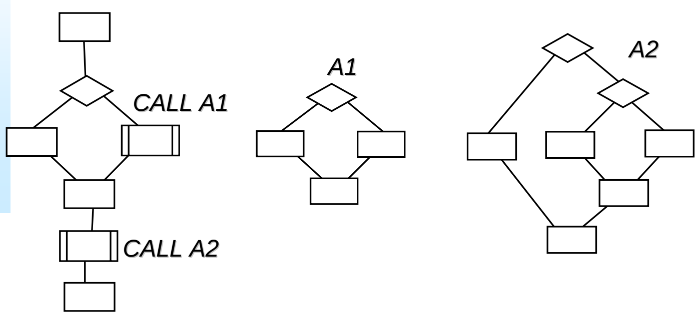
The Ciclomatic complexity (or number) is a measure of the logical complexity defined as (after Henderson-Sellers):

$$V(g) = e - n + p + 1$$

Where given a flow chart:

- e = number of edges (connecting lines)
- n = number of nodes (boxes)
- p = number of procedure calls (connected components)

Mc Cabe, Ciclomatic Complexity



- $V(g) = e - n + p + 1 = 19 - 18 + 3 + 1 = 5$

Halstead Volume Metric

The volume can be used a indirect measure of Effort:

$$Vol = N \, Log_2 \, n$$

where: N = N1 + N2, and n = n1 + n2

- N1 = total number of operators
- N2 = total number of operands
- n1 = number of distinct operators
- n2 = number of distinct operands

D S I

Token Based Size/Volume Metric

...

D S I

© Paolo Nesi 1995-2000

10

Class Level Metrics

- CC = Class Complexity
- CCI = Inherited Class Complexity
- CCL = Local Class Complexity
- WMC = Weighted methods per class (Kidamber and Kemerer)
- NA = Number of attributes, local and inherited
- NAL = number of local attributes
- NAI = number of inherited attributes
- NM = number of methods, local and inherited
- NML = number of local methods
- NMI = number of method inherited
- NSUP = number of superclasses until the root (similar to DIT)
- NSUB = number of subclasses
- NOC = number of children (Kidamber and Kemerer)
- ..
- .

CC Metric

- Class Complexity (*CC*) is a weighted sum

$$
\begin{aligned}
CC = w_{CACL} CACL \; + \; & w_{CACI} CACI + \\
+ w_{CL} CL \; + \; & w_{CI} CI + \\
+ w_{CMICL} CMICL + & w_{CMICI} CMICI
\end{aligned}
$$

- *CACL* ⇒ Local Attributes
- *CACI* ⇒ Inherited Attributes
- *CL* ⇒ Local Methods (on *Vg, Vg', LOC, Ms, Ha*…)
- *CI* ⇒ Inherited Methods (as above)
- *CMICL* ⇒ Local Method Interface
- *CMICI* ⇒ Inherited Method Interface

Class Level Metrics

- Cohesion metrics
  - CBO = Coupling between objects (Kidamber and Kemerer)
  - MPC = message passing coupling (Li & Henry)
  - CMI = Complexity of method interface
- Reuse Metrics
  - ECD = external class description
  - CMIC = Class Method interface Complexity
  - RI = Reuse Index
  - VI = Verifiability Index
  - ..
- ...
- Protected, private and public …..

D S I

10

Cognitive Metrics

- The External Class Description (ECD) measures the absolute external comprehensibility of a class, taking into account the methods interface and the class attributes;

$$ECD = CACL + CACI + CMICL + CMICI$$

- Class CoGnitive Index (CCGI) measures the ratio between ECD and CC (with all weights = 1), giving a relative value for the understandability of the class.

$$CCGI = \frac{ECD}{CC}$$

D S I

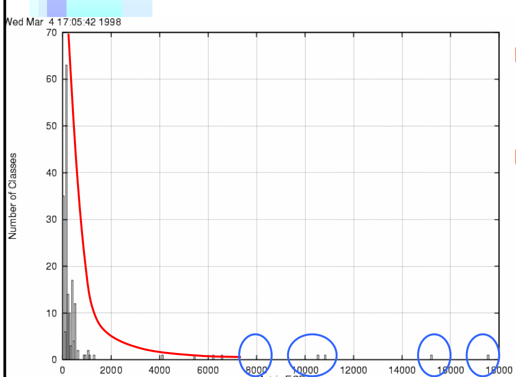
10

Cognitive Metrics

Class Level Metrics:

$$ECD_m = ECDL_m + ECDI_m,$$
$$ECDL_m = w_{CACL_m} CACL_m + w_{CMICL_m} CMICL_m,$$
$$ECDI_m = w_{CACI_m} CACI_m + w_{CMICI_m} CMICI_m,$$
$$ICI_m = w_{CI_m} CI_m + w_{CL_m} CL_m,$$
$$CC_m = ECD_m + ICI_m,$$
$$CCGI_m = \frac{ECD_m}{CC_m}.$$

External Class Description (ECD)

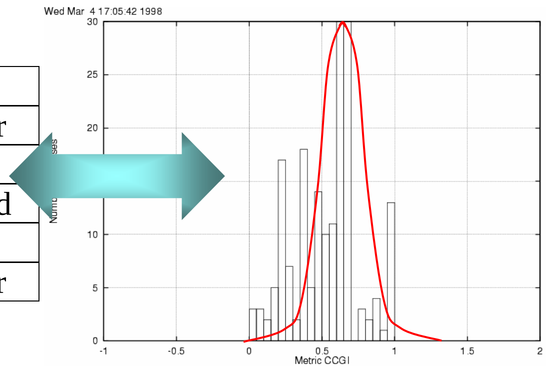


- ECD measures the complexity of the external class interface;
- A too high ECD may show that the class is too complex to be easy understandable;
- ECD alone cannot measure class understandability.

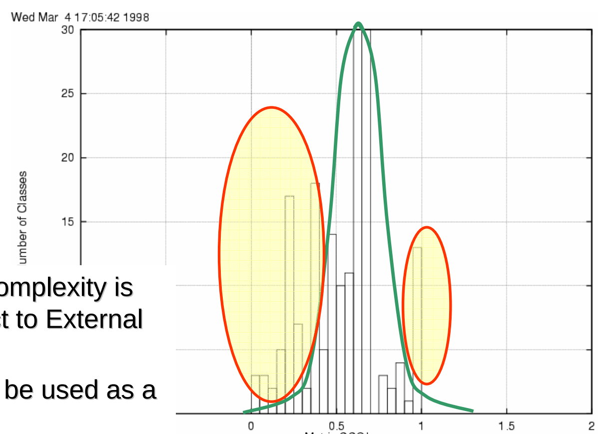
Reference Values and Trends for system assessment

- Metrics Reference Values are necessary for system assessment; BUT they give only a local snapshot of the class or an evaluation of mean values;
- Metrics trends are mandatory for a global project assessment and direct identification.

CCGI Ratings	
less than 0.3	poor
between 0.3 and 0.5	fair
between 0.5 and 0.7	good
between 0.7 and 0.9	fair
greater than 0.9	poor



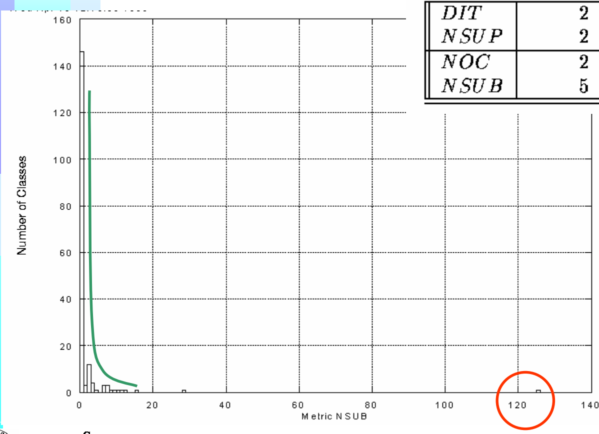
Class CoGnitive Index (CCGI)



- **CCGI<0.3**: Internal complexity is very high with respect to External Interface.
- **CCGI≈0.6**: class can be used as a black box.
- **CCGI=1**: class defined but not yet implemented or C structures.

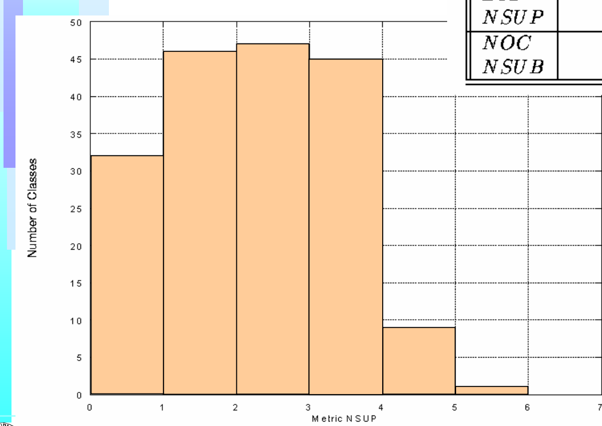
NSUB and its ref. Values and trend

	Mean Values	Maximum Values
DIT	2	4 - 5
NSUP	2	5 - 6
NOC	2	10 - 20
NSUB	5	30 - 90



NSUP and its ref. Values and trend

	Mean Values	Maximum Values
DIT	2	4 - 5
NSUP	2	5 - 6
NOC	2	10 - 20
NSUB	5	30 - 90



System Level Metrics

- Global Behaviour of Class Level Metrics
- Global Behaviour of method Level Metrics
- Mean value of Class Level Metrics
  - MCC = Mean CC
  - MNAM = Mean number of attributes and methods
- Mean value of Method Level Metrics
- Profiles with respect to the Attended values

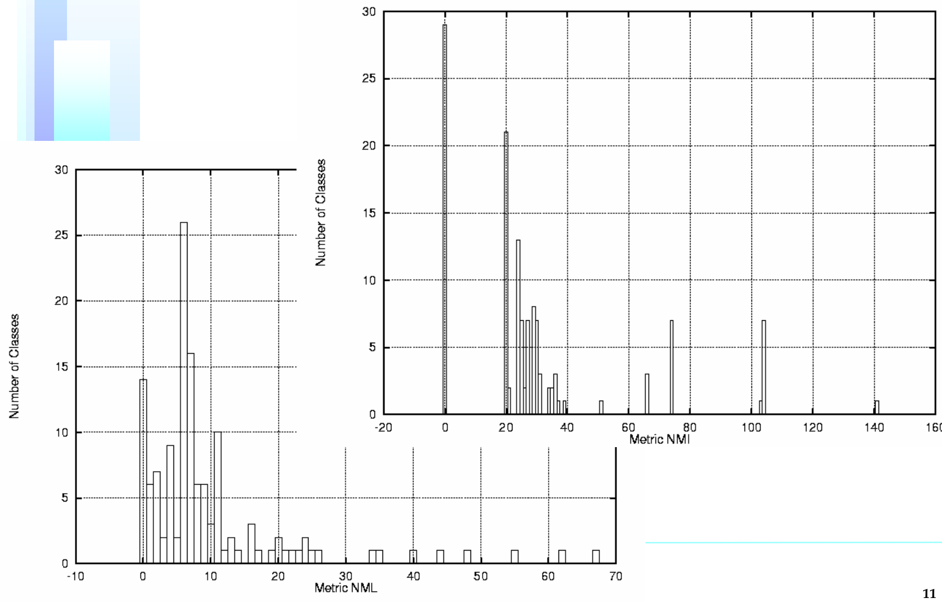
 11

System Level Metrics

- General Consumptive Total Metric:
  - SC = system complexity
  - NC = number of classes
  - Highness of class trees..
  - Number of Global variables
  - Number of Methods
  - .
- Hierarchical Metrics
  - Max Highness of class trees
  - Number of distinct Trees
  - ..
  - .

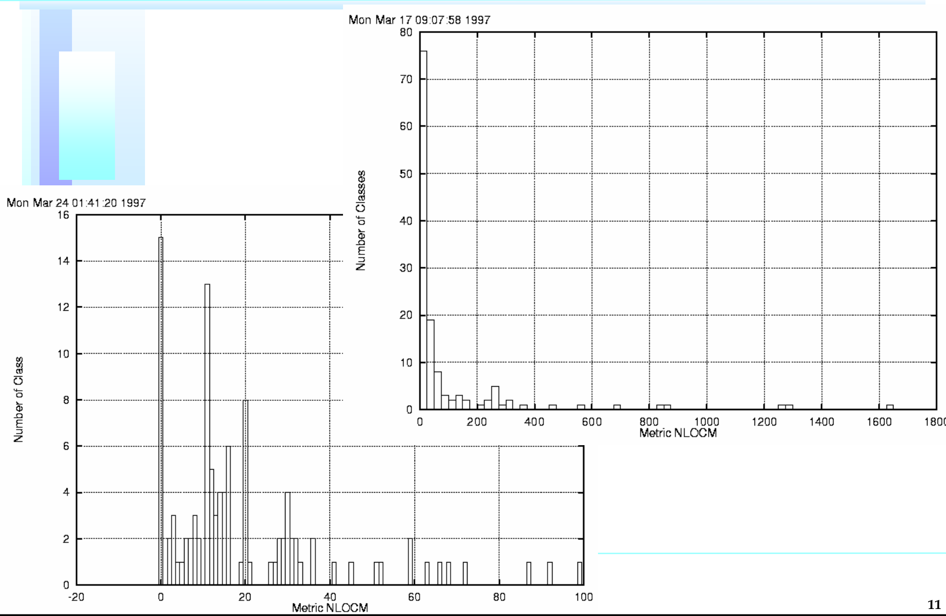
 11

Local and Inherited Number of Methods



11

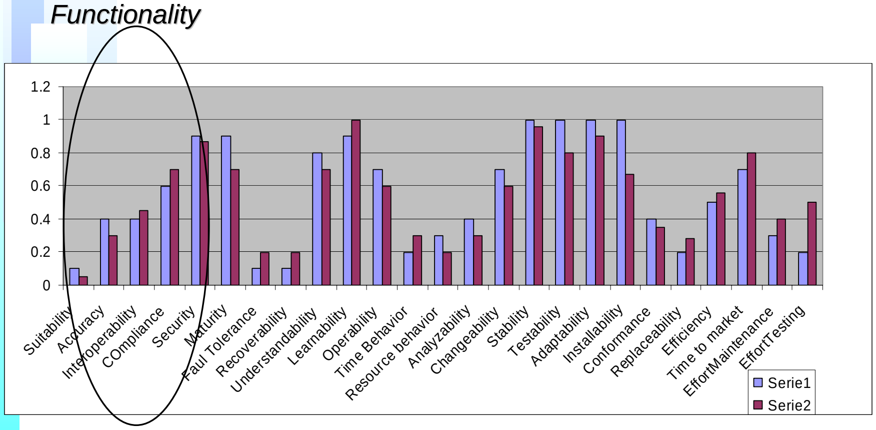
Distribution of Method Complexity



11

Product Profile

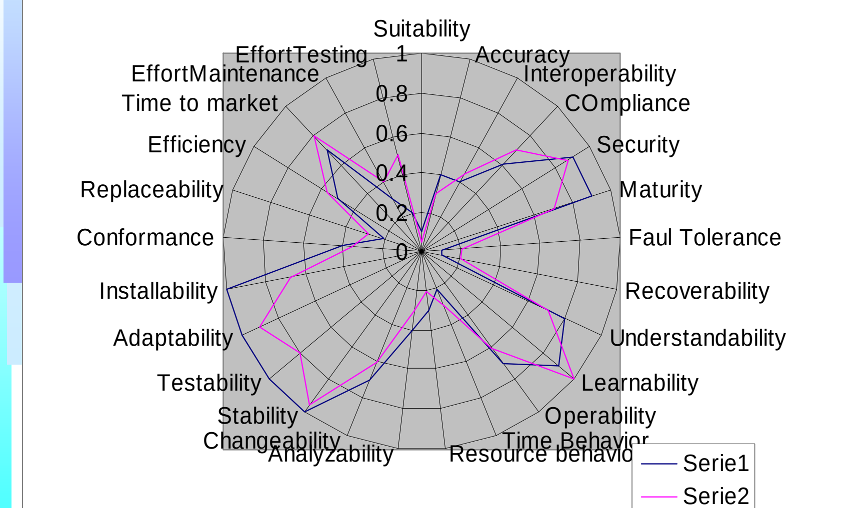
Functionality



© Paolo Nesi 1995-2000

11

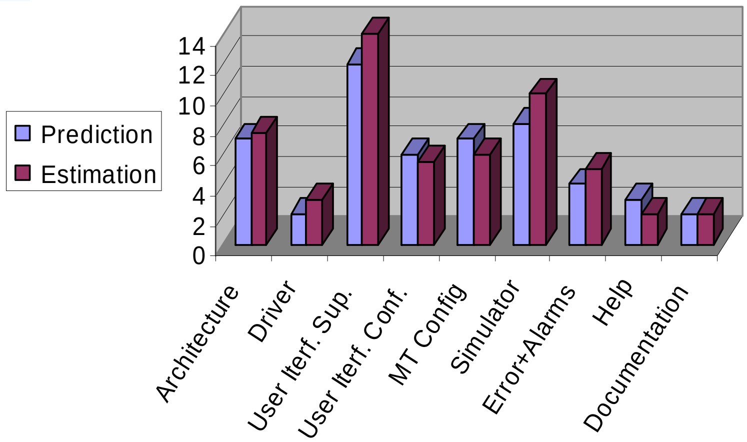
Product Profile



© Paolo Nesi 1995-2000

11

Consumptive Analysis of Effort Profile



© Paolo Nesi 1995-2000

11

Metric Life Cycle

- Metric Identification
- Metric Evaluation
- Metric Validation

© Paolo Nesi 1995-2000

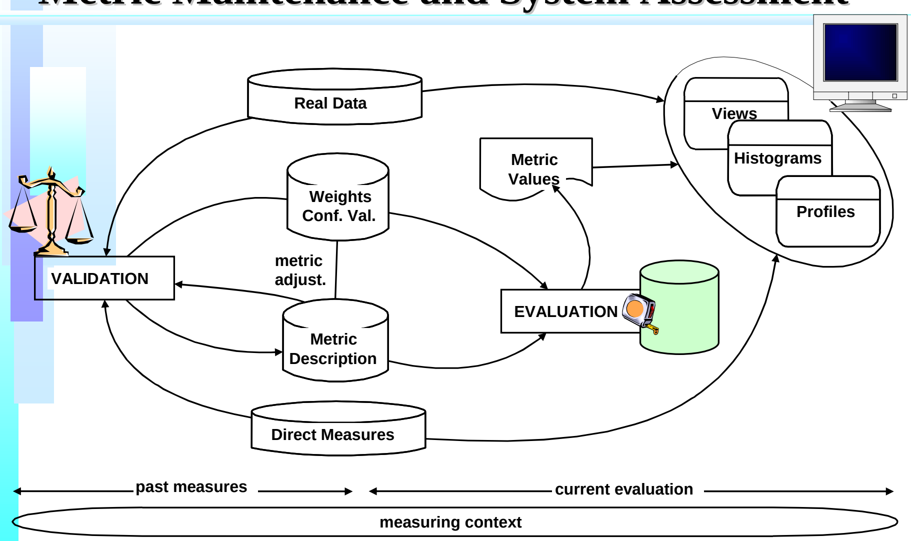
12

Metric Identification and Definition of Result Analysis Mechanisms

Estimation
- Metric Identification
- Metric Evaluation

Understanding Results with Respect
- Reference Values
- Significant Trends
- Significant Profiles

Metric Maintenance and System Assessment



Metric Definition and Identification

Goal Question Metric

- The definition of indirect metrics is a quite complex task
- It is typically based on the generalized experience
- This experience is typically extracted by using questionnaires.

Good Sense Metric

- On the basis of the simple experience
- ..

Metrics vs Validation

- Most software characteristics are not directly valuable;
- Indirect measures must be performed;
- Validation by the means of statistical instruments must be performed for indirect measures;
- Needs of well designed object oriented reference projects for validation, other metrics can be used for verifying the project goodness
- Metrics are validated if quite stable results are obtained: proving or disproving their goals.
- E.g., Statistic and logistic analyses

Product Assessment Context

- **Development Context --** system/subsystem structure (GUI, non GUI; embedded; Real-Time, etc.), application field (toy, safety critical, etc.); tools and languages for system development (C, C++, Visual C++, GNU, VisualAge, etc.); development team (expert, young, mixture, small, large, to be trained, etc.); adoption of libraries; development methodology; assessment tools; etc.;
- **Life-Cycle Context --** requirements collection, requirements analysis, general structure analysis, detailed analysis, system design, subsystem design, coding, testing, maintenance (e.g., adaptation, porting), documentation, demonstration, testing, regression testing, number of cycle in the spiral life-cycle, etc.

 12

Object Oriented Metrics for Effort Control

The Quality and Productivity of Object-Oriented Development: Measurement and Empirical Results

Paolo Nesi

Department of Systems and Informatics, University of Florence
&
CQ-ware (Center for Software Quality), CESVIT, High Tech Agency
Via S. Marta 3, 50139, Firenze, Italy
tel: +39-055-4796523, fax: +39-055-4796363, cell +39-(0)335-5917797
nesi@ingfi1.ing.unifi.it, nesi@dsi.unifi.it, p.nesi@computer.org
http://www.dsi.unifi.it/~nesi

 12

Why measuring complexity ?

- Complexity as an indirect measure of high level features for estimating and predicting:
  - reusability, reliability, maintainability, testability, etc.
  - testing process, test cases, error proneness, etc.
  - effort for …. development, reuse, maintain, etc.
- Complexity is considered an internal feature, typically estimated on code but could be estimated on: requirements, analysis, design, documents, etc.
- How to trace and convert complexity measures on requirements, analysis, code, design, testing, documentation, etc. ?
  Problems of scale unit ………..? Which scale type ….….?

12

You can't measure what you can't recognize

The lack of agreement about the meaning of complexity

- psychological (understandability, reusability): subjective
- computational/time (nesting levels): asymptotical complexity, *O()*
- structural/space (data relationships): #var, #types
- interface (composability, reusability, coupling): #flows, fan-in, fan-out
- flow graph (control statements): cyclomatic complexity, execution paths
- functional (requirements, data transform): Halstead

Cognitive, behavioral, functional, structural aspects.

12

Complexity Views

- Several different flavors of complexity exits !!
- Most of proposed metrics address more than one aspect together with some subjective element
- Initially complexity was confused with Size. Recently, complexity is considered a design aspect related to internal relationships of the system. **Both these definitions are too restrictive.**
- ***The following relationships are assumed !***

Complexity of the problem ←→ Complexity of the solution

Complexity for building a system ←→ Complexity of the system

- Are they true ? If so…… It is a linear relationship ?
- ***Context factors have to be also considered !***

The Holy Grail of Measuring Complexity

- formal frameworks defining properties for complexity metrics have been proposed.
- Which of the proposed frameworks is the most general, the most powerful, and the most correct for characterizing complexity metrics ?
- Even taking only one single aspect, it has to be demonstrated that

Weak Order

$$C_1 \le \bullet C_2 \le \bullet C_3 \le \bullet C_4 \le \bullet ...... \le \bullet C_n$$

Homomorphism

$$\forall C_1, C_2 \quad C_1 \le \bullet C_2 \Leftrightarrow |C_1| \le |C_2|$$

- It is a common opinion that flow graphs cannot be ordered ….
- The Homomorphism cannot be demonstrated for high level feature and complexity measure,……
- No repeatability, no comparability, no unified concept, no uniform unit, ….

The definition of complexity metrics is impossible ?

- Presently the empirical evidence is the only solution.

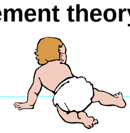
Scale Type vs Complexity

- Effort is meaningful for the ratio scale with positive and negative values.
- Size and complexity are only positive concepts: ratio scale, ordinal scale, interval scale
- for complexity should be ratio scale and non negative values
- for effort should be ratio scale but also negative values
- for size should be interval or ratio scale and non negative values
- for flow graph depends on the metrics used

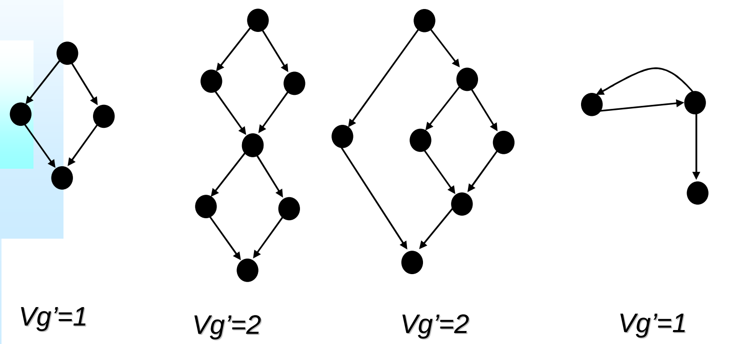
Which Future for Measuring Complexity ?

- Physical phenomena can be ordered based on physical observations.
- In the past, the measuring of length of objects was in the same trouble as software measurement today: no repeatability, no comparability, no uniform unit, ….
- **Confusion about the phenomenon and its effects.** In some cases, the measures of the effects are used for indirectly measuring the phenomenon. *The measure of temperature by measuring the increment of length of a tread.*
- In 600 years the problems were solved demonstrating the dependency on temperature, humidity, type of material, subjective factors (parallax, practices), etc., **the measuring model depended on measuring context.** Today the model is known and the measures can be repeated, compared, converted for scale, etc., with a certain precision.

In the next years, a strong evolution on software measurement theory and technique is needed



Ordering ?



Other Complexities

- Algorithmic
- Computational
- Informational
- Data
- Structural
- Logical
- Combinatorial
- Cyclomatic
- Essential
- Topological
- Harmonic
- Syntactic
- Semantic
- Mnemonic
- Perceptional
- Flow
- Entropic
- Functional
- Organizational
- Diagnostic
- Risk
- Technological

No agreement base

- Whitmire
- Henderson Sellers
- Fenton
- Shepperd
- Zuse
- Pressman
- Jones
- Basili
- Briand
- Tian
- Weyuker

13

System Assessment Problems

- How to assess Object Oriented systems?
- How to select metrics?
- How to understand results?
- How to adjust metrics parameters for each project?
- How to get measures with low effort?
- How to collect metrics
- How to take advantage from the results?
- When to use them?
- Who have to use them?
- Who have to adjust them?
- ……

© Paolo Nesi 1995-2000

13

Outline

- Experience foundation
- Development Effort Estimation
- Development Effort Prediction
- Maintenance Effort Estimation and Prediction
- Weight Evolution for System Analysis and control
- Metric Life Cycle
- Fault Prediction Model
- Discussion and Conclusions

Focus on Effort estimation and prediction by using complexity/size metrics. This implies also to take under control the project evolution according to management, analysis and design guidelines/thumb rules drawn by metrics

Some Reference Projects: 1994-1999.

project	OS	language	tools \ libs	R/D	Prt	NC	SC_LOC	MM	people	teams
TOOMS	UNIX, Linux	C++	Lex/Yacc, CommonView	R&D	1	204	16568	41.5	16	6
ICOOMM	Windows NT	C++	MFC	D	2	193	10870	20	6	3
QV/MOOVI	UNIX, Linux	C++	XLIB, Motif	D	1	65	3900	7	4	2
LIOO	LINUX	C++	Lex/Yacc, XVGAlib	R&D	1	165	16020	30	11	5
MOODS	LINUX	C++	Lex/Yacc, XVGAlib, Moovi	D&R	5	225	24356	44	14	8
TAC++	UNIX, Linux	C;C++	Lex/Yacc, QV, XLIB	R	2	62	2300;4340	13.5	5	2
							LOC			
MUPAAC	WinNT, CE	C;C++	MFC	R&D	3	40	19460	46	5	2
ICCOC	WinNT	Java	JDK	R	3	15	1200	9	3	1
Running P.							**LOC**			
SAMOPROS	Windows NT	Java, C++	JDK, Db-Symantec	D&R	3	157	23500	15	3	1
TOTS	Unix/WinNT	C++	XW, X	R	1	35	15400	9	2	1
OMR	Unix, Linux	C;C++	Slib, X	R	2	40	23240	31	5	2
TecnoTEXT	Windows	C++	MFC, VideoForWindows	D	2	20	14300	9	2	1

- NC: Number of system Classes;
- SC_LOC: object-oriented System Complexity based on LOC (number of lines of code);
- effort in person-months;
- the number of people involved (without including task and project managers);
- number of different teams.
- Some ESPRIT Projects: MUPAAC, ICCOC, MOODS
- Some Technology Transfer projects: TecnoTEXT, ICOOMM, SAMOPROS
- Some Reengineering Projects: MUPAAC, MOODS, MOOVI

Typical problems

- Heavy leaf classes: poor design degeneration
- poor hierarchy: poor design
- Abstract classes with only a super and a sub
- many classes few attributes
- Too cohesion among classes via method calls
- too much friends
- massive use of partial inheritance
- few huge classes managing many small classes
- confusion between inheritance and decomposition specialization with specification
- poor reusability ...
- Public attributes...

Some Reference Values

Several OO conformance projects. Several estimations along their life-cycle, etc..

	Mean Values	Max Values	Min Values
$CC*$	200	1500	–
ECD	350	1500	20
$ECDI$	200	2000	80
$ECDL$	150	1600	20
ICI	200	1500	–
CI	150	1200	10
CL	50	700	–

	Mean Value	Minimum Value
$CCGI$	0.6	0.33
$CCGII$	0.6	0.35
$CCGIL$	0.6	0.30

	Mean Values	Maximum Values
NAM	45 - 117	69 - 189
$NAMI$	30 - 78	46 - 126
$NAML$	15 - 39	23 - 63
NA	9 - 27	15 - 45
NAI	6 - 18	10 - 30
NAL	3 - 9	5 - 15
NM	36 - 90	44 - 144
NMI	24 - 60	36 - 96
NML	12 - 30	18 - 48

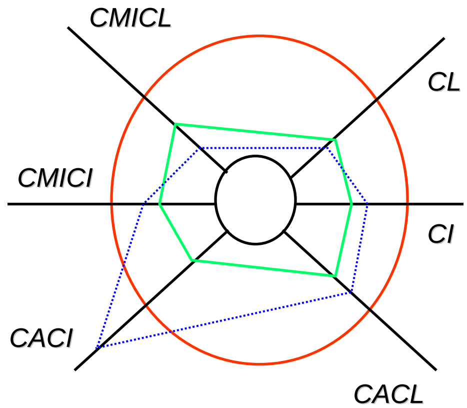
	Mean Values	Maximum Values
DIT	2	4 - 5
$NSUP$	2	5 - 6
NOC	2	10 - 20
$NSUB$	5	30 - 90

Reference Values vs Class Assessment

Class per Class ?

The Kiviat diagram for the CC components is evaluated and the major problem is discovered to be in CACI (Complexity of inherited attributes)

Dep. of the Life-Cycle



 14

Management Thumb Rules………

- A well defined management model
- A light Methodology for Management, Analysis and Design
- A start-up from huge experience/expert people
- A good prediction process: key classes, engine classes, metrics
- Stable Development tools
- Flexible assessment tools, with tunable metrics on context
- Defined Macro-cycle: 4 months
- Moderately Flexible Micro-cycle (be careful with overhead), 2-3 weeks
- Small team, 1-3 + SSM
- Constant resource allocation
- Horizontal management and roles
- Continue metric re-validation and tuning

 14

Controlling Project Evolution

- Early Analysis metrics have to be used for planning the project and evaluating resources: NA, NK.
- Design Metrics have to be used: DIT, NOC, NSUB, NSUP, SI, AID, etc.
  for limiting/detecting project degenerative conditions
- Complexity/size metrics have to be used during the whole life-cycle for controlling project evolution and detecting degenerative conditions.
- Metrics with weights

We reviewed 224 Object-Oriented metrics extracting principal components

D S I

PCA on 224 Object Oriented Metrics

- PC1: (general lines) LOC, GLOC, CL, NGCL, NAIV, NROV, ..
- PC2: (local data) CACL, CC1, CC1', ECD, ECDL, CS, ..
- PC3: (inherited data) NAIPROO, NAIPRO, NAI, ECDI, CACI, .
- PC4: (local methods) LCOM1-4, NMA(B), NMIMP, NUMPARA, NML, NMLPUB, ...
- PC5: (inherited methods) NMIPUB, NMI, NAMI, CI, CIMICI, ..
- PC6: (ancestors) IH-ICP_L, AMMIC_L, NOA, ..
- PC7: (number of local attributes) NAL, NA, ..
- PC8: (data coupling) DAC'_L, DAC'_L, ..
- PC9:….
- PC10:…...

D S I

Outline

- Experience foundation
- Development Effort Estimation
- Development Effort Prediction
- Maintenance Effort Estimation and Prediction
- Weight Evolution for System Analysis and control
- Metric Life Cycle
- Fault Prediction Model
- Discussion and Conclusions

Analysis of Components, no weighted metrics with scale factor, weighted metrics, complexity/size metrics.

Consumptive Estimation Metrics

- try to evaluate the value assumed by a system feature in the time instant and context in which they are applied.
- for
  - verifying the consistency of results with predicted project evolution
  - controlling the project evolution against planned values
  - estimating general weights that can be useful for monitoring other projects

Metrics for effort estimation and prediction for Object Oriented projects

- Traditional functional metrics (*Vg, Vg', Ha, LOC,* token-based,...) are not satisfactory for effort estimation and prediction of OO systems; as demonstrated several times in the literature and by our experience .....
- Compromises are: *WMC, Size2, NML, NAL, NOML, NOAL, etc.*
- More complete but more complex metrics have been proposed by us in the past (1996/97): *CC, CC', NAM*
- These metrics have been validated for effort estimation and prediction, for development and maintenance.

 14

Principal Components

- **Algorithmic, Computational, Structural, Logical, Psychological, Cyclomatic, Syntactic, Semantic, Flow, Entropic, Functional, Technological, etc.**
- **Only some components are relevant and can estimated**
- **In order of relevance:**
  - **Inherited Functionality**
  - **Local Functionality**
  - **Local Data Structure**
  - **Inherited Data Structure**
- **in the 5 PCs of the above analysis**

	1	2	3	4
CACI	0.1412	0.0521	0.0463	0.9823
CACI'	0.4127	0.0457	0.0292	0.8984
CACL	-0.045	0.2773	0.9376	0.0029
CACL'	-0.0438	0.234	0.9385	0.0723
CI	0.9272	0.0808	-0.0396	0.1434
CL	0.1748	0.8284	0.3335	-0.0115
CMICI	0.8815	0.0261	-0.0108	0.2999
CMICI'	0.8815	0.0261	-0.0108	0.2999
CMICL	0.0778	0.9404	0.0645	0.0692
CMICL'	0.0778	0.9404	0.0645	0.0692
NAI	0.9583	0.0462	-0.0503	-0.0219
NAL	-0.1868	0.6388	0.4455	-0.0003
NMI	0.9737	0.0548	-0.0512	0.0615
NML	0.0312	0.9081	0.1774	0.0045

 14

CC Metric

- Class Complexity (*CC*) is a weighted sum

$$CC = w_{CACL} CACL + w_{CACI} CACI + w_{CL} CL + w_{CI} CI + w_{CMICL} CMICL + w_{CMICI} CMICI$$

- *CACL* ⇒ Local Attributes
- *CACI* ⇒ Inherited Attributes
- *CL* ⇒ Local Methods (on *Vg, Vg', LOC, Ms, Ha*…)
- *CI* ⇒ Inherited Methods (as above)
- *CMICL* ⇒ Local Method Interface
- *CMICI* ⇒ Inherited Method Interface

D S I

14

Estimation of CC Terms

$$CACL(c) = \sum_{i=1}^{NAL(c)} CC1(ai) \qquad CACI(c) = \sum_{i=1}^{NAI(c)} CC1(ai)$$

$$CMICL(c) = \sum_{j=1}^{NML(c)} \sum_{i=1}^{|p(mj)|} CC1(pi) \qquad CMICI(c) = \sum_{j=1}^{NMI(c)} \sum_{i=1}^{|p(mj)|} CC1(pi)$$

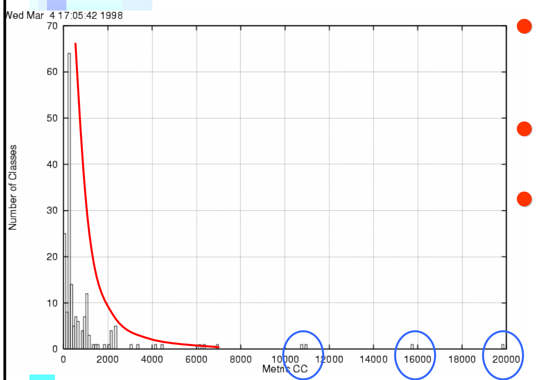
$$CL(c) = \sum_{i=1}^{NML(c)} FM(mi) \qquad CI(c) = \sum_{i=1}^{NMI(c)} FM(mi)$$

- *FM(mi)* can be: *Vg, Vg', Ms, LOC, |N|, |E|, NL, etc.*.
- *CC1* is *CC* with all weights equal 1
- Basic Types are constant value or *0* depending on *FM* used
- *ai* class of a class attribute
- *pi* class of a method parameter

D S I

15

Class Complexity (CC)



- The optimal profile is quite independent by the used OO language;
- Optimal profile respected by 98% percent of classes;
- Some classes present too high complexity.

© Paolo Nesi 1995-2000 15

CC : a generalisation of HS, T&J, C&K ideas ?

- *HSCC* and *TJCC structurally* proposed by Henderson-Sellers ('91) and Thomas and Jacobson (1989):

$$HSCC = w'_{CACL}\, CACL' + w'_{CL}\, CL' + w'_{CI}\, CI'$$

$$TJCC = w''_{CACL}\, CACL'' + w''_{CL}\, CL''$$

- no method interface, method inherited
- *WMC* proposed by Chidamber and Kemerer (1994):

$$WMC = CL_{Vg} \qquad (WMC = NML)$$

© Paolo Nesi 1995-2000 15

Statistic Analysis

One example of statistical analysis is given by the study about CC and Effort relationships. Our assumption is that Effort and CC are proportional for each of the n *developed classes;*

$$Effort_i \approx w_{CACL} CACL_i + w_{CACI} CACI_i + w_{CL} CL_i + w_{CI} CI_i + w_{CMICL} CMICL_i + w_{CMICI} CMICI_i,$$

Estimation of Weights in the metric tuning on the basis of the measuring/develop. context

Class Complexity

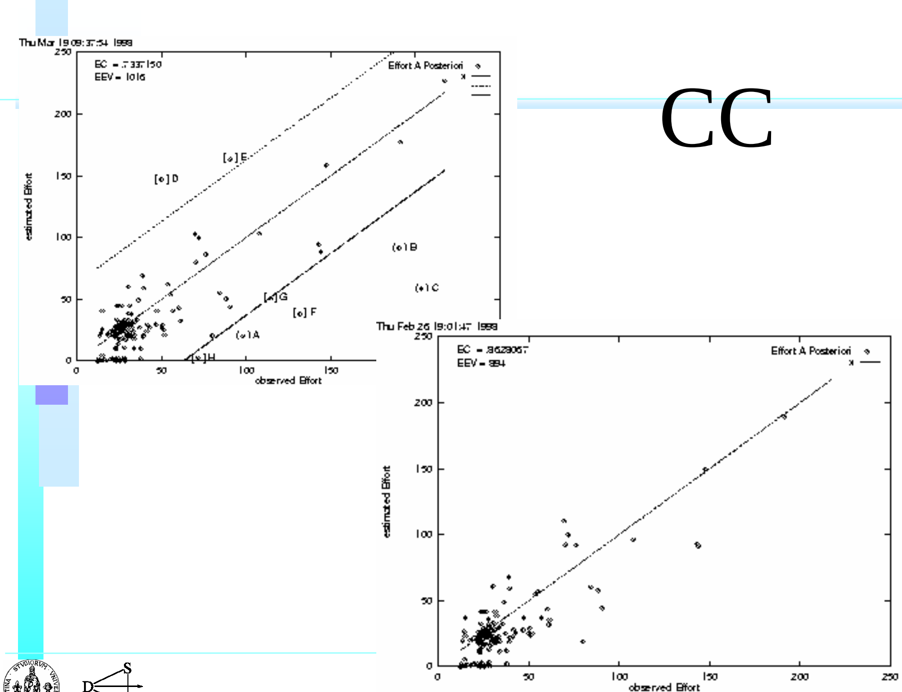
CC_m	$m = LOC$			$m = Ha$		
	w	*t-value*	*p-value*	*w*	*t-value*	*p-value*
$CACL_m$	.001	4.82	.000	.001	4.19	.000
CL_m	.016	8.63	.000	.006	8.37	.000
$CMICL_m$	.053	4.10	.000	.087	8.18	.000
$CACI_m$	-.023	-2.49	.014	-.014	-1.95	.054
CI_m	-.001	-.59	.554	.001	.53	.598
$CMICI_m$	.010	1.49	.140	.002	.28	.776
Eff-Corr.		0.935			0.937	
Variance		145.68			216.63	
LS scale		3.166			3.120	
R-squared		0.896			0.899	
F-stat		134.00			138.47	
p-value		0.000			0.000	

Total development effort including documentation, test, integration, assessment.

CC_m	$m = MS$			$m = Mc$		
	w	*t-value*	*p-value*	*w*	*t-value*	*p-value*
$CACL_m$	.001	4.97	.000	.001	4.39	.000
CL_m	.022	10.01	.000	.092	7.46	.000
$CMICL_m$	.042	3.39	.001	.042	2.66	.009
$CACI_m$	-.026	-3.48	.001	-.070	-3.57	.001
CI_m	-.002	-.72	.471	.014	1.28	.202
$CMICI_m$	.013	1.75	.083	.014	1.39	.166
Eff-Corr.		0.945			0.926	
Variance		192.98			149.28	
LS scale		2.928			3.379	
R-squared		0.911			0.881	
F-stat		159.24			115.79	
p-value		0.000			0.000	

	Corr	var.
CLvg	0.90	245
CLloc	0.91	186

CC



 15

Comparison Results

Vg' Based

Effort-Mvg

Vg'-Based	k*CC1	CC	CC (without CI)	TJCC	HSCC	k'*CL
Correlation	0.617	0.724	0.725	0.661	0.660	0.623
Std. Dev.	14.97	18.57	18.51	17.26	17.09	16.41
K	0.162	--	--	--	--	0.575

LOC Based

Effort-Mloc

LOC-Based	k*CC1	CC	TJCC	HSCC	k'*CL
Correlation	0.569	0.776	0.670	0.670	0.648
Std. Dev.	13.63	18.24	19.97	16.89	21.47
K	0.008	--	--	--	0.092

Effort-M *Design Metrics*

	k'*NSUB	k''*NSUP
Correlation	0.005	0,110
Std Dev	1.89	3.44
K	0.211	2.952

Only code development effort

 15

Metric Relationships -- CC -- LOC-based

LOC	CACI	CACI'	CACL	CACL'	CC1	CC1'	CI	CL	CMCI	CMCI'	CMCL	CMCL'	Effort	prod	NA1	NAI	NAL	NAM1	NAM1	NAML1	LOC	NM1	NM	NML	NSUB	NSUP
CACI	1.00																									
CACI'	0.95	1.00																								
CACL	0.05	0.02	1.00																							
CACL'	0.11	0.07	0.94	1.00																						
CC1	0.17	0.19	0.96	0.90	1.00																					
CC1'	0.19	0.18	0.93	0.99	0.93	1.00																				
CI	0.29	0.53	-0.07	-0.06	0.18	0.05	1.00																			
CL	0.09	0.11	0.52	0.46	0.66	0.50	0.22	1.00																		
CMCI	0.38	0.58	-0.05	-0.03	0.15	0.11	0.77	0.11	1.00																	
CMCI'	0.38	0.58	-0.05	-0.03	0.15	0.11	0.77	0.11	1.00	1.00																
CMCL	0.10	0.11	0.35	0.33	0.45	0.38	0.11	0.74	0.11	0.11	1.00															
CMCL'	0.10	0.11	0.35	0.33	0.45	0.38	0.11	0.74	0.11	0.11	1.00	1.00														
Effort	0.33	0.30	0.48	0.50	0.57	0.55	0.07	0.65	0.14	0.14	0.65	0.65	1.00													
prod	0.08	0.07	-0.03	-0.01	-0.03	0.00	-0.04	-0.08	0.11	0.11	-0.05	-0.05	0.20	1.00												
NA1	0.13	0.37	0.16	0.14	0.39	0.23	0.83	0.48	0.66	0.66	0.30	0.30	0.22	-0.07	1.00											
NAI	0.12	0.39	-0.09	-0.10	0.13	0.01	0.92	0.18	0.75	0.75	0.07	0.07	-0.02	-0.05	0.89	1.00										
NAL	0.04	-0.01	0.55	0.51	0.57	0.50	-0.11	0.66	-0.12	-0.12	0.53	0.53	0.53	-0.06	0.33	-0.13	1.00									
NAM1	0.19	0.43	0.09	0.07	0.33	0.18	0.87	0.48	0.74	0.74	0.35	0.35	0.23	-0.07	0.95	0.91	0.20	1.00								
NAM1	0.18	0.45	-0.09	-0.10	0.14	0.02	0.93	0.19	0.81	0.81	0.07	0.07	0.00	-0.04	0.87	0.98	-0.14	0.93	1.00							
NAML1	0.06	0.03	0.46	0.41	0.54	0.43	0.02	0.82	-0.04	-0.04	0.75	0.75	0.61	-0.10	0.38	-0.01	0.85	0.37	-0.01	1.00						
LOC	0.09	0.11	0.52	0.46	0.66	0.50	0.22	1.00	0.11	0.11	0.74	0.74	0.65	-0.08	0.48	0.18	0.66	0.48	0.19	0.82	1.00					
NM1	0.20	0.44	0.07	0.05	0.32	0.16	0.87	0.48	0.75	0.75	0.36	0.36	0.23	-0.07	0.93	0.90	0.16	1.00	0.93	0.36	0.48	1.00				
NM	0.20	0.46	-0.09	-0.09	0.14	0.02	0.93	0.19	0.82	0.82	0.07	0.07	0.01	-0.03	0.86	0.97	-0.14	0.92	1.00	-0.01	0.19	0.93	1.00			
NML	0.06	0.04	0.40	0.36	0.50	0.38	0.06	0.82	-0.01	-0.01	0.78	0.78	0.60	-0.11	0.37	0.03	0.76	0.40	0.03	0.99	0.82	0.41	0.03	1.00		
NSUB	-0.04	-0.06	-0.04	-0.04	-0.05	-0.04	-0.10	0.03	-0.10	-0.10	0.14	0.14	0.00	0.00	-0.05	-0.10	0.09	-0.06	-0.13	0.15	0.03	-0.06	-0.14	0.16	1.00	
NSUP	0.14	0.30	-0.17	-0.18	-0.05	-0.12	0.51	-0.03	0.51	0.51	-0.13	-0.13	-0.16	-0.07	0.43	0.59	-0.29	0.55	0.67	-0.20	-0.03	0.57	0.69	-0.16	-0.23	1.00

- *The development effort is not strongly correlated with metric terms*

15

Outline

- Experience foundation
- Development Effort Estimation
- Development Effort Prediction
- Maintenance Effort Estimation and Prediction
- Weight Evolution for System Analysis and control
- Metric Life Cycle
- Fault Prediction Model
- Discussion and Conclusions

© Paolo Nesi 1995-2000

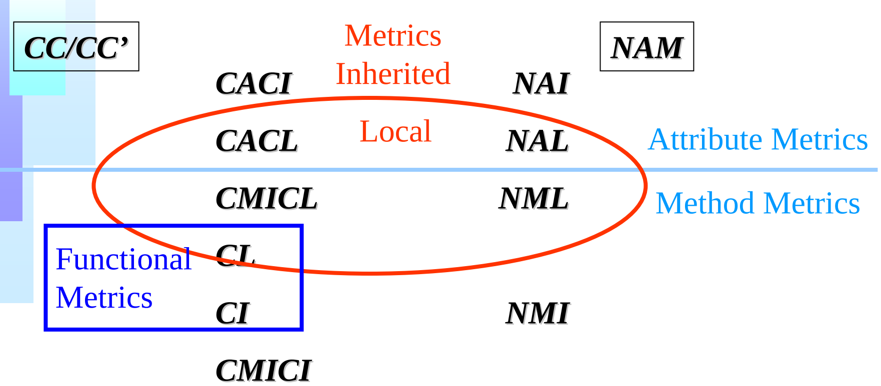
15

Predictive Metrics

- **Prediction metrics** try to evaluate the value that will be assumed by the system feature in the future. For example, the prediction of LOC after 5 months by knowing the absolute values and the variation of LOC measured in the last two check points, CP.
- **Predictive metrics** for
  - predicting project evolution
  - controlling in advance the project evolution against planned values
  - when used in consumptive manner are used for estimating general weights that can be useful for other projects

Considering: CC', NAM, Size2, NML, NAL, etc.

Low Level Metrics Used



CC' a prediction Metric

- Class Complexity (*CC'*) is a weighted sum

$$CC' = w_{CACL'}CACL' + w_{CACI'}CACI' + w_{CMICL'}CMICL' + w_{CMICI'}CMICI'$$

- Weights and terms are different with respect to those of *CC* since are estimated without considering the implementation part (functional) *CL* and *CI*
- Results since the availability of class definition
- Lighter to be estimated than *CC*, but less complete and precise

D S I

16

NAM Metric

- Number of Attributes and Methods of a Class:

$$NAM = w_{NAL}NAL + w_{NAI}NAI + w_{NML}NML + w_{NMI}NMI$$

- where:
  - *NMI* number of inherited methods
  - *NML* number of local methods (i.e., *WMC*)
  - *NAL* number of local attributes
  - *NAI* number of inherited attributes

D S I

16

NAM: a generalisation of Size2

- Size2 proposed by Li and Henry (1993)

$$Size2 = NAL + NML$$

- or of other metrics such as:
  - *NMI* number of inherited methods
  - *NML* number of local methods (i.e., *WMC*)
  - *NAL* number of local attributes
  - *NAI* number of inherited attributes
  - *NAMI* number of inherited members (w*NAI*+w*NMI*) even without weights: NAMI1

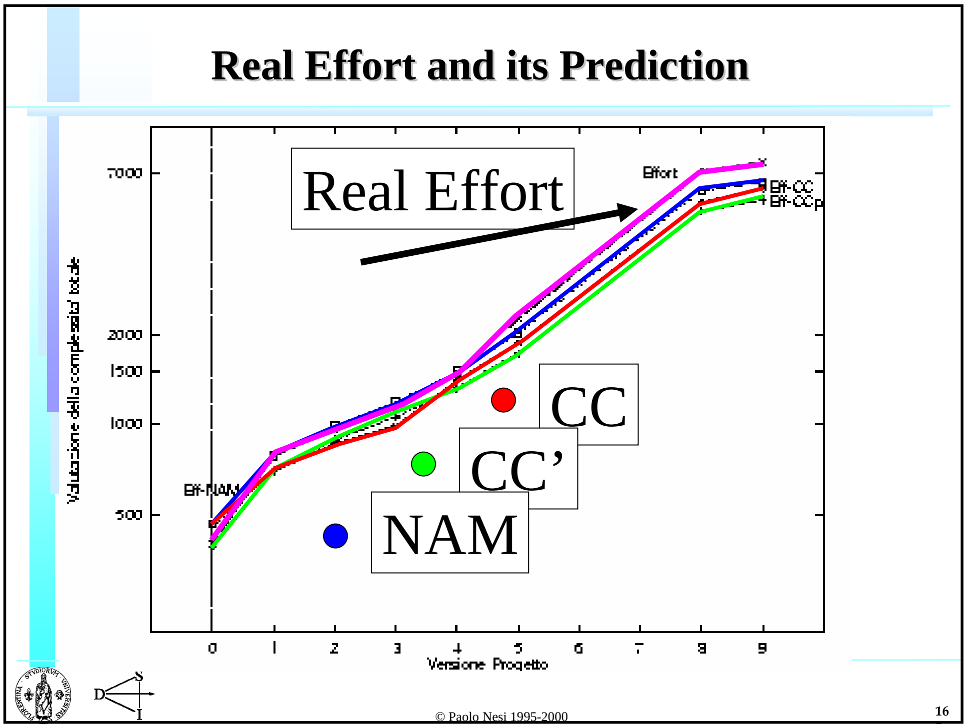
Metric Validation: CC' and NAM

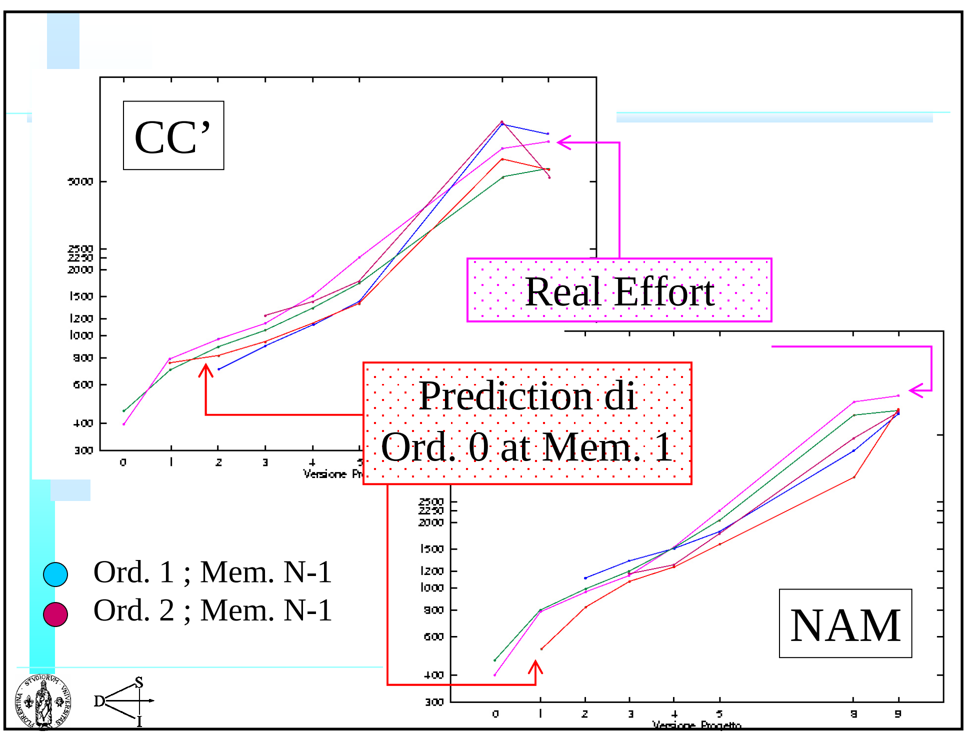
CC'

	LSR			RLSR		
	w	*t-value*	*p-value*	*w*	*t-value*	*p-value*
CACL	0,00420	7,34462	0,00000	0,00415	9,06716	0,00000
CACI	0,00140	1,28266	0,20119	0,00100	1,26245	0,20841
CMICL	0,26821	11,88954	0,00000	0,28858	16,24648	0,00000
CMICI	0,06433	8,52261	0,00000	0,05879	10,63053	0,00000
Effort corr.	0,72782			0,84017		
Effort var.	1188			1077		

NAM

	LSR			RLSR		
	w	*t-value*	*p-value*	*w*	*t-value*	*p-value*
NML	1.89164	11.03266	0.00000	1.72640	13.81481	0.00000
NMI	1.10285	5.89666	0.00000	0.79956	6.47982	0.00000
NAL	1.07325	2.06166	0.04062	1.49450	4.61429	0.00000
NAI	-3.66425	-4.69642	0.00000	-2.33691	-4.60500	0.00000
Effort corr	0.71929			0.83255		
Effort var.	1004			697		


Real Effort and its Prediction
Real Effort
Effort
Eff-CC
Eff-CCp
Eff-NAM
CC
CC'
NAM
Valutazione della complessita' totale
7000
2000
1500
1000
500
0 1 2 3 4 5 6 7 8 9
Versione Progetto
© Paolo Nesi 1995-2000
16


CC'
Real Effort
Prediction di
Ord. 0 at Mem. 1
NAM
Ord. 1 ; Mem. N-1
Ord. 2 ; Mem. N-1
Versione Progetto

Comparison Results

Only code development effort

Scale Factors allow to estimate coherent precise values

Counting Class Members

	NAM	k*NAM1	k'*Size2	k''*NML	k'''*NAL
Correlation	0.619	0.248	0.616	0.605	0.507
Std Dev	13.57	6.73	13.26	12.88	13.26
K	--	0.173	0.929	1.123	3.629

LOC Based

LOC-Based	k*CC1'	CC'
Correlation	0.547	0.751
Std Dev	14.57	17.41
K	0.026	--

Predicting System Effort

- Early estimation on the basis of Class Definition
- Prediction of Total Effort with NAM
- Metrics with weights
- Prediction considering only NAL and guessing NML
- Low errors in predicting system effort

	Actual	NAM	NAML	NAML (NML_{prev})
Hours	406,3	465,564	483,693	462,343
Error Percentage	N.A.	12,7%	16,1%	12,1%
Mean value of class effort	3,94	4,520	4,696	4,489
Mean Error of class effort	N.A.	-0,575	-0,751	-0,544
Standard Deviation	N.A.	5,931	5,918	6,681
Confidence Interval	N.A.	4,520 ± 17,793	4,696 ± 17,754	4,489 ± 20,043

Discussion

- Guidelines on System analysis and design allow to build a more repeatable and stable development process
- Class and system effort can be estimated and predicted since the early phases of the development life-cycle:
  - Key, engine classes, scale factors
  - NAL, NAML, CC', CC in this order
- Cohesion metrics are not so relevant for system effort
- Design metrics are not so relevant for system effort
- The context plays a relevant role in the assessment and a suitable methodology for identifying eventual problems has to be used

Outline

- Experience foundation
- Development Effort Estimation
- Development Effort Prediction
- Maintenance Effort Estimation and Prediction
- Weight Evolution for System Analysis and control
- Metric Life Cycle
- Fault Prediction Model
- Conclusions

Maintenance

- It is typically evaluated to be close to 50 % of the application

Considering 100% of the Maintenance costs:

- **Corrective**: 25%, to correct problems remained in the code
- **Adaptive**: 25 %, to modify the application in order to maintain it operative in the up-dated environment
- **Perfective**: 50 %, to improve the functionalities of he application, the GUI, the manuals, etc.

The last two are actions for improving the application

Maintenance Effort Estimation and Prediction

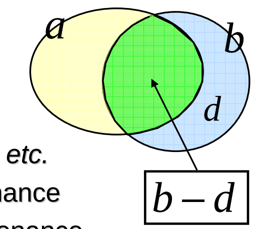
	Before (LIOO1)	After (LIOO5)
NCL	113	133
NRC	19	25
TNM	1087	1341
TLOC	12150	13891
MCC	876	877
MNA	7	7
MNM	38	39

- Project: MOODS ESPRIT IV
- 44 Man Months, 9 for the adaptive maintenance
- Porting from DOS to UNIX, updating the stand-alone to distributed
- The same team of the early development
- 15% of increment

Metric model for Adaptive Maintenance

- General Measure of the activity of Adaptive Maintenance at code level:

$$Mam = Ma - (Mb - md)$$



- where
  - *M* can be: *CC, CC', Size2, WMC, NAM, etc.*
  - *Ma* Measure after the adaptive maintenance
  - *Mb* Measure before the adaptive maintenance
  - *md* Measure of deleted code for the adaptive maintenance
- *md* can be hard to be measured for complexity metric. For *LOC*, it could be the number of deleted lines of code.
- Changes are seen as deletions and writing of new code

Metric model for Adaptive Maintenance

- Hp. of changes uniformly distributed on classes
- Generally *md* can be approximated with a small percentage of class Measure:

$$md = wMb$$

$$Mam = Ma - w_{Mb} Mb$$

- where $w_{Mb} = 1 + w$
- typically:
  - code deletion is limited to parts of methods
  - deletion of attributes or method is rare, members are left ins the class (deprecated class members).
  - This a well know problem of Object Oriented systems

Metric model for Estimation of the Adaptive Maintenance Effort

- General Measure of the activity of Adaptive Maintenance at level of code:

$$Effam \approx Mam + U = Ma - (Mb - md) + U$$

- where *U* is the effort for code understanding during the adaptive maintenance (for adding and deleting code)
- *U=0* when the team performing the Adaptive Mainten. Is the same that have performed the early development
- In first approximation:

$$U \approx w' Mb \qquad md \approx wMb$$

$$Effam \approx Mam + U = Ma - w_{Mb} Mb$$

DSI

© Paolo Nesi 1995-2000 17

Adaptive Maintenance vs OO, some issues

- *Adding classes* for addressing the new functionality. These are totally new classes, *Mb=0*.
- *Deleting Classes* (deleted code, ɯq≠0)
- *Deleting, adding code in classes* ....

More complex and rare cases:

- *Moving methods*, generalisation process.
- *Fusing classes* into one (quite rare).
- *Dividing classes* distributing, delegating functionality (quite frequent operation when a class is too large)

→ *Non Uniform changes* on the system

- *Grouping of classes* with similar history is a solution (different weights can be defined for different evolutions)

DSI

© Paolo Nesi 1995-2000 17

Model Application to CC, CC', NAM

- For example:

12 Weights

$$Effam \approx CCam = CCa - wcc_b CCb$$

8 Weights

$$Effam \approx CC'am = CC'a - wcc'_b CC'b$$

8 Weights

$$Effam \approx NAMam = NAMa - w_{NAMb} NAMb$$

.....for the other metrics.....

- The sign of the *Before* part is only a *Hyp.* since *W* is included into the weights that have to be estimated according to a validation phase.
  For example by using a regression analysis.

Coefficients Analysis for *Effam--CCam*

CC_{am}	w	$\mid t-value \mid$	$p-value$
CL_b	-0.012	2.32	0.022
CI_b	-0.024	1.15	0.250
$CMICL_b$	0.522	9.02	0.000
$CMICI_b$	0.009	0.18	0.860
$CACL_b$	-0.009	1.95	0.053
$CACI_b$	0.124	1.11	0.267
CL_a	-0.022	3.38	0.001
CI_a	-0.032	1.94	0.055
$CMICL_a$	0.545	11.12	0.000
$CMICI_a$	0.037	0.94	0.348
$CACL_a$	-0.008	1.79	0.076
$CACI_a$	0.211	2.13	0.035
Correlation	0.84 with all components		
	0.87 by removing $CMICI$		
Variance	116		
R-squared	0.825		
F-stat (p-value)	47.11 (0.000)		

Terms Analysis for *CC'am* -- *Effam* Prediction

CC'_{am}	w	$\mid t-value \mid$	$p-value$
$CMICL'_b$	0.476	8.70	0.000
$CMICI'_b$	0.122	1.84	0.068
$CACL'_b$	-0.038	4.16	0.000
$CACI'_b$	-0.124	0.76	0.449
$CMICL'_a$	0.492	11.83	0.000
$CMICI'_a$	0.102	1.77	0.079
$CACL'_a$	-0.033	3.94	0.000
$CACI'_a$	-0.044	0,27	0.791
Correlation	0.79 with all coefficients		
	0.81 by eliminating $CACI'$ component		
Variance	126		
R-squared	0.708		
F-stat (p-value)	44.35 (0.000)		

 17

Terms Analysis for *NAMam* -- *Effam* Prediction

NAM_{am}	w	$\mid t-value \mid$	$p-value$
NAL_b	1.162	0.77	0.440
NAI_b	2.194	1.34	0.182
NML_b	1.399	3.73	0.000
NMI_b	-0.587	1.83	0.069
NAL_a	3.485	2.46	0.015
NAI_a	1.280	0.85	0.396
NML_a	1.501	4.62	0.000
NMI_a	-0.264	0.88	0.380
Correlation	0.75		
Variance	191		
R-squared	0.729		
F-stat (p-value)	41.813 (0.000)		

No improvement with less terms

 18

Metric Comparison and Results

	A-Posteriori Metrics		
	CC_{am}	$TJCC_{am}$	$HSCC_{am}$
Max Correlation	0.87	0.42	0.43
Variance	166	494	2043
Number of weights	10	2	6
	Predictive Metrics		
	CC'_{am}	NAM_{am}	$Size2_{am}$
Max Correlation	0.81	0.75	0.73
Variance	118	191	59
Number of weights	6	8	1

Simple Model vs Proposed Model

- Simple Model:

$$Effam \approx Ma$$

	CC_a	CC'_a	NAM_a	$Size2_a$
Correlation	0.64	0.56	0.67	0.52
Variance	90	87	166	210
Number of weights	6	4	4	0

- Weights of *Ma* equal to weights of *Mb*, if any.

$$Effam \approx \Delta Mab_{w1,6}$$

Outline

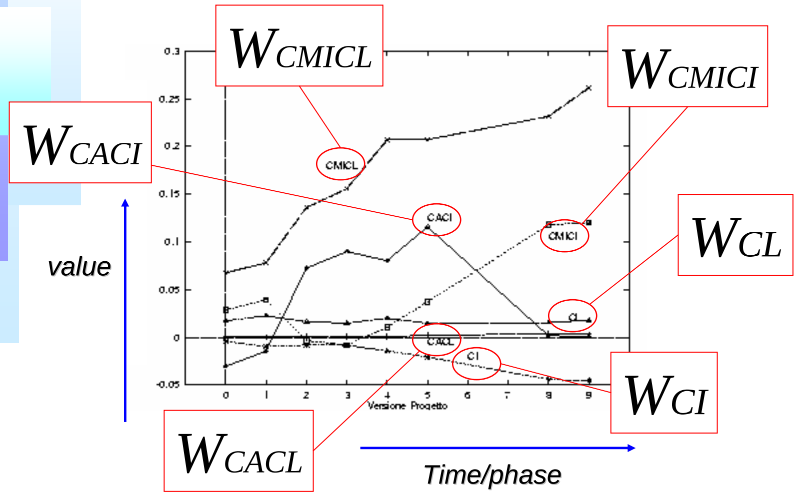
- Experience foundation
- Development Effort Estimation
- Development Effort Prediction
- Maintenance Effort Estimation and Prediction
- Weight Evolution for System Analysis and control
- Metric Life Cycle
- Fault Prediction Model
- Discussion and Conclusions

Weight Evolution for System Analysis and control

The weight estimation allows:

- Tuning metrics for getting more precise estimations
- Modeling/encoding/learning the measuring context
- tracking the project evolution,
- studying/detecting eventual degenerative conditions
- using weights in future projects getting more precise estimating of evaluation and prediction

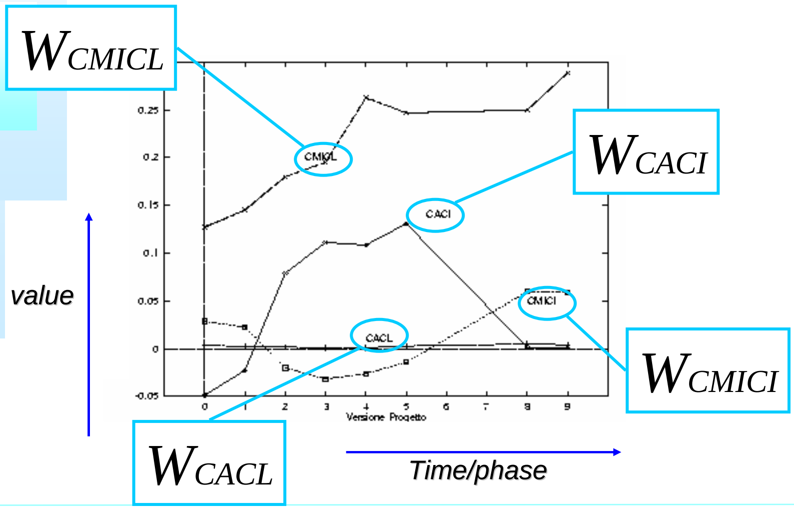
Trend of CC weights during the project life-cycle



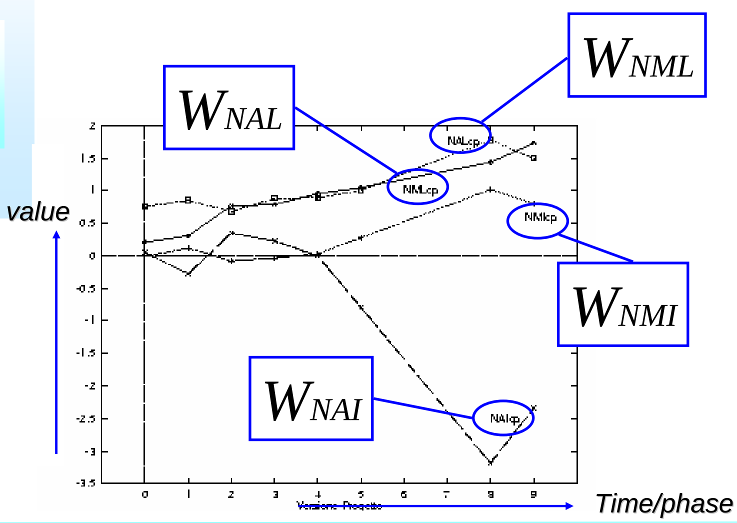
Trend of Metric Parameters

- *WCMICL* increases with the project phase. The metric is decreasing with respect to the other terms;
- *WCL*, *WCACL* are quite constant. The relevance of local functional and structural aspects is constant along the process even during the porting and adaptation;
- *WCI* is negative and decreases (more after the adaptation), the term is a gain. The inheritance is a saving in for development costs.
- *WCMICI* has a decreasing trend from 1 to 3. During the adaptation, the metric relevance is comparable to metrics related to local factors. During the adaptation the interfaces for inherited methods were an advantage.
- *WCACI* presents a positive trend from 1 to 5. During the adaptation the metric has been less relevant than local terms. Adaptation has been implemented by restructuring functional parts, the inspection of the attributes of the super-classes has

Weights Trend for CC'



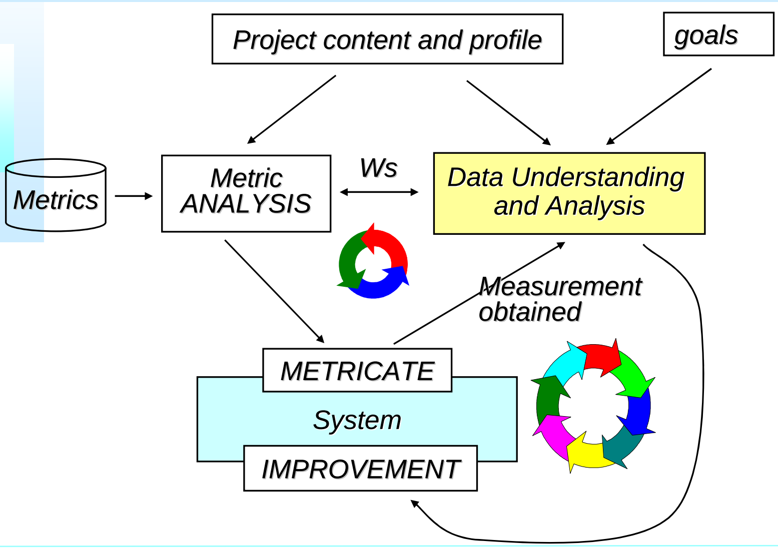
Weights Trend for *NAM*



Outline

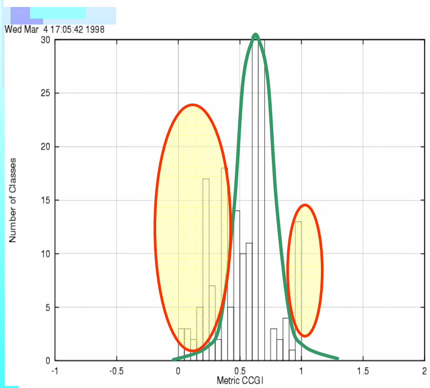
- Experience foundation
- Development Effort Estimation
- Development Effort Prediction
- Maintenance Effort Estimation and Prediction
- Weight Evolution for System Analysis and control
- Metric Life Cycle
- Fault Prediction Model
- Discussion and Conclusions

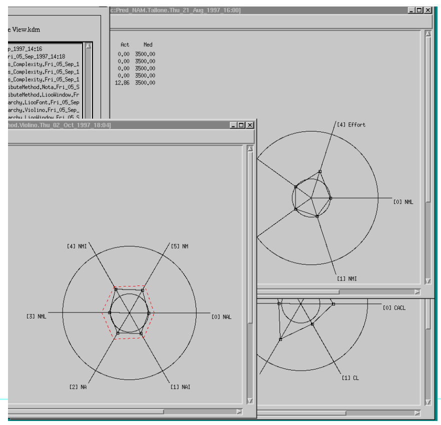
Continuous System and Metric Improvement



Histograms, Views and Bounds

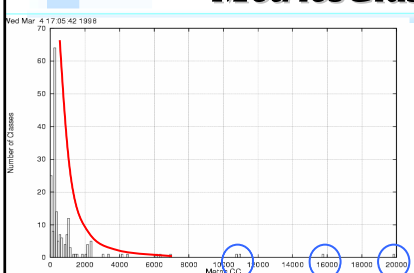
- **CCGI<0.3**: Internal complexity is very high with respect to External Interface.
- **CCGI≈0.6**: class can be used as a black box.
- **CCGI=1**: class defined but not yet implemented or C structures.



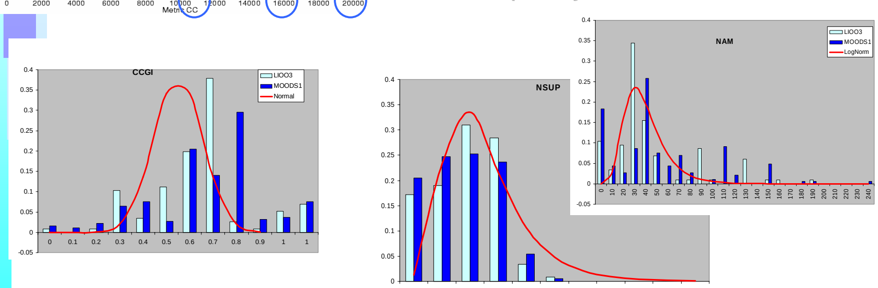


 19

MetricsClass Complexity (CC)



- The optimal profile is quite independent by the used OO language;
- Optimal profile respected by 98% percent of classes;
- Some classes present too high complexity.



 19

Outline

- Experience foundation
- Development Effort Estimation
- Development Effort Prediction
- Maintenance Effort Estimation and Prediction
- Weight Evolution for System Analysis and control
- Metric Life Cycle
- Fault Prediction Model
- Discussion and Conclusions

19

Fault Prediction

- **Model tuned on UMD projects.**
- **Analysis of 224 Object Oriented metrics.**
- **A General Model with detection close to 97 %**
- **A reduced model with 9 metrics close to 87 %**
- **Principal Component Analysis ….**
- **Logistic Regression ….**

Overall 85.84 %		Predicted		
		No fault	With fault	
Observed	No fault	53	7	88.33% of correct no faults
	With fault	9	44	83.02% of correct faults

19

Fault Prediction Model

- **TCC:** Tight Class Cohesion (Bieman and kang, 1995)
- **NMLPRO:** NML protected (Nesi)
- **NLOCM:** method size metric local (Nesi)
- **CMSL:** method size metric local (Nesi)
- **CMS:** method size (local plus inherited) (Nesi)
- **ICC:** CL + CI (Nesi)
- **NDR:** # of data references (Sneed)
- **NDSTT:** # of diff. Statements (Sneed)
- **RFC_inf:** response of a class (Chidamber and Kemerer 1991)

	PC1	PC2	PC3
TCC	0.133	0.387	**0.896**
NMLPRO	-0.163	**0.877**	-0.392
NLOCM	**0.956**	-0.174	0.055
CMSL	**0.965**	-0.110	0.003
CMS	**0.966**	-0.119	-0.002
ICC	**0.949**	-0.189	0.044
NDR	**0.775**	0.431	0.008
NDSTT	**0.809**	0.445	-0.005
RFC_OO	**0.850**	-0.042	-0.331

Metric Model Relationships

	TCC	NMLPRO	NLOCM	CMSL	CMS	ICC	NDR	NDSTT	RFC_OO
TCC	**1.000**	-0.018	0.103	0.089	0.082	0.088	0.242	0.256	-0.140
NMLPRO	-0.018	**1.000**	-0.306	-0.237	-0.242	-0.312	0.184	0.240	-0.057
NLOCM	0.103	-0.306	**1.000**	**0.919**	**0.918**	**0.990**	0.602	0.741	0.768
CMSL	0.089	-0.237	**0.919**	**1.000**	**0.998**	**0.905**	0.728	0.673	0.798
CMS	0.082	-0.242	**0.918**	**0.998**	**1.000**	**0.912**	0.725	0.667	**0.801**
ICC	0.088	-0.312	**0.990**	**0.905**	**0.912**	**1.000**	0.590	0.726	0.770
NDR	0.242	0.184	0.602	0.728	0.725	0.590	**1.000**	0.728	0.605
NDSTT	0.256	0.240	0.741	0.673	0.667	0.726	0.728	**1.000**	0.649
RFC_OO	-0.140	-0.057	0.768	0.798	**0.801**	0.770	0.605	0.649	**1.000**

- This model is a compromise
- Low estimation cost, low metric number
- Any change leads to decrease the model predictability
- non critical cut-off level of 0.5

Outline

- Experience foundation
- Development Effort Estimation
- Development Effort Prediction
- Maintenance Effort Estimation and Prediction
- Weight Evolution for System Analysis and control
- Metric Life Cycle
- Fault Prediction Model
- Discussion and Conclusions

Discussion and Conclusions

- Lessons learned for managing projects have been shortly reported
- Guidelines on System analysis and design allow to build a more repeatable and stable development process
- Class and system effort can be estimated and predicted since the early phases of the development life-cycle:
  - Key, engine classes, scale factors
  - NAL, NAML, CC', CC in this order
- The same metrics can be used during maintenance
- Cohesion metrics are not so relevant for system effort
- Design metrics are not so relevant for system effort
- The assessment methodology plays a relevant role

Conclusions

- A general model for estimation and prediction of the Adaptive Maintenance has been presented with a corresponding validation based on multilinear regression.
- The model works quite well with: *CC, CC', NAM*
- Less satisfactory results have been obtained with *WMC, HSCC, TJCC* and *Size2*
- The model is general enough to be used with other metrics

Lesson Learned in Pills

- Simple metrics can be used: NAM, Size2, NML, NAL they have also to be tuned: scale factor
- Complex metrics are better, and Tuned metrics are strongly better and more expensive
- System design has to be controlled
- Consumptive metrics based on mean value are not significant: max, min bounds can be not enough and too expensive; Combined with Histograms and profiles become good tools
- Good tools for System Assessment are hard to find out
- Metric tuning and revalidation is needed
- The measuring context has to be absolutely considered: changes in the bonds, absolute values and weights

Maintenance Effort

- A general model for estimation and prediction of the Adaptive Maintenance has been identified with a corresponding validation based on multilinear regression.
- The model works quite well with: *CC, CC', NAM*
- Less satisfactory results have been obtained with *WMC, HSCC, TJCC, Size2, etc.*
- The model is general enough to be used with other OO metrics

Ma in pratica..............

- Un modello di gestione ben definito
- Una metodologia leggera ma ripetibile
- Uno start-up eseguito da persone di grande esperienza
- Un buon processo di predizione
- Strumenti di sviluppo stabili nel tempo
- Strumenti di valutazione flessibili e con metriche semplici e validate sul profilo utilizzato
- Un macro-ciclo stabile e definito, 4 mesi/ciclo
- Un micro-ciclo flessibile ma no troppo altrimenti costa troppo in overhead, 2-3 settimane/ciclo
- piccoli gruppi di sviluppatori, 1-3 + SSM
- continua rivalidazione e correzione del processo di stima

Recent Bibliography

- P. Nesi, ``Managing OO Projects Better'', IEEE Software, July/August, pp.50-60, 1998.
- F. Fioravanti, P. Nesi, S. Perlini, "A Tool for Process and Product Assessment of C++ Applications'', Proc. of the 2nd Euromicro Conference on Software Maintenance and Reengineering, IEEE Press, Florence, Italy, pp.89-95, 8-11 March 1998.
- F. Fioravanti, P. Nesi, S. Perlini, "Assessment of System Evolution Through Characterization'', Proc. of the IEEE International Conference on Software Engineering, Kyoto, Japan, pp.456-459, 19-25 April 1998.
- G. Bucci, F. Fioravanti, P. Nesi, S. Perlini, "Metrics and Tool for System Assessment'', Proc. of the IEEE International Conference on the Engineering of Complex Computer Systems, Monterrey, California, USA, pp.36-46, 10-14 August 1998.
- P. Nesi and T. Querci, "Effort Estimation and Prediction of Object-Oriented Systems'', The Journal of Systems and Software, 42, pp.89-102, 1998.
- F. Fioravanti, P. Nesi, F. Stortoni, "Metrics for Controlling Effort During Adaptive Maintenance of Object Oriented Systems", IEEE Conference on Software Maintenance, Oxford, Agosto, 1999

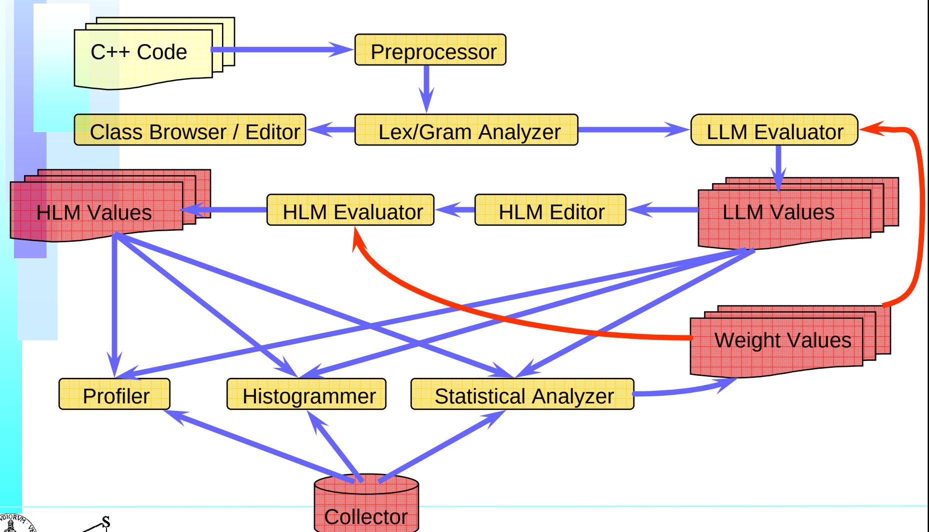
 20

 20

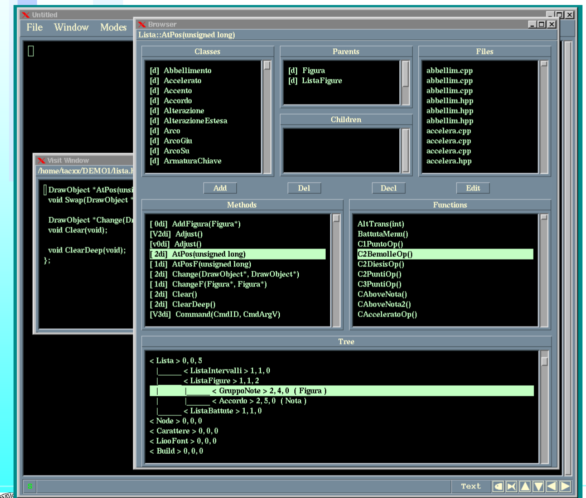
TAC++
Tool for Analysis of C++ Code

- An integrated environment for analyzing C++ Code;
- A tool to evaluate Low Level Metrics;
- A tool to define and evaluate High Level Metrics;
- A tool for collecting real data: effort, fault, changes, effort for maintenance, etc.
- A tool for graphical representation of results;
- A statistical analyzer for metrics validation.

TAC++ Architecture



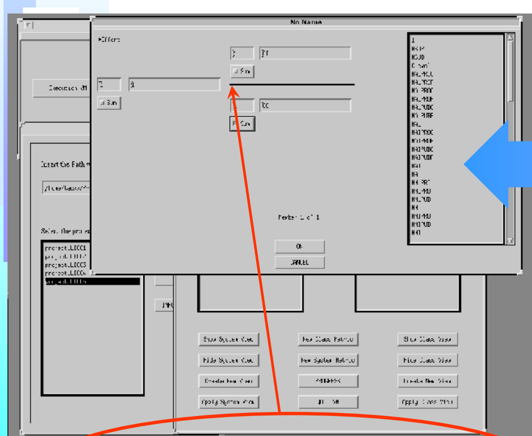
TAC++ Class Browser



- Class tree structure visualization;
- Class tree navigator;
- Source code editor.

20

TAC++ High Level Metric Editor

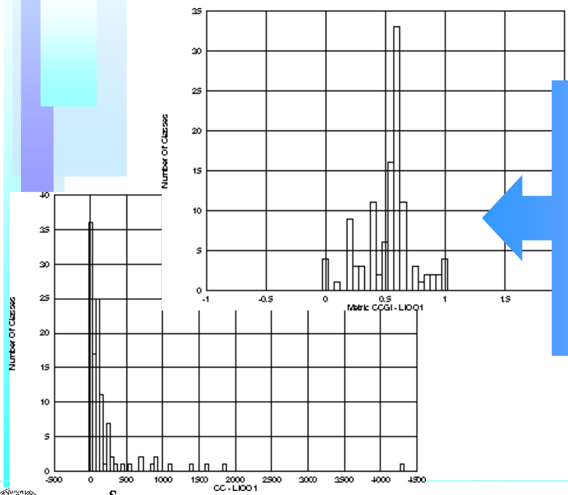


- High Level Metric Definition;
- High Level Metric Editing;
- System, Class, Method Level Metric Utilization;
- Complex Metric Definition.

$$NewMetric = \sum w_{M1} M1 \frac{w_{M2} M2}{w_{M3} M3}$$

20

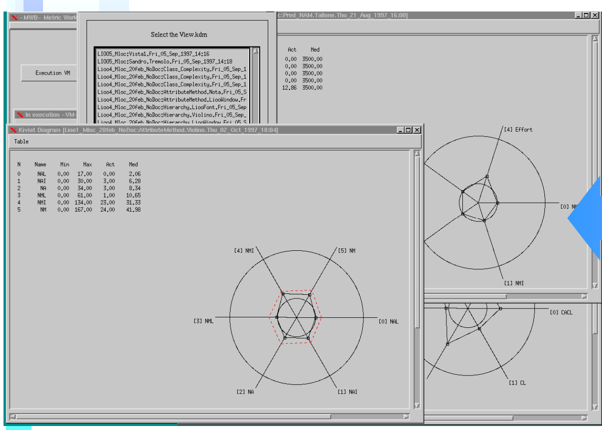
TAC++ Histogrammer



- Low Level Metrics Histograms;
- High Level Metric Histograms;
- Method Level Histograms.

20

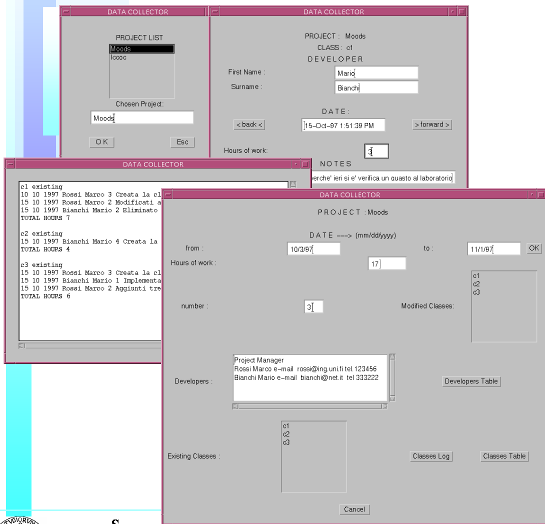
TAC++ Profiler



- Class Profiling;
- System Level Profiling;
- Kiviat;
- Tables.

21

TAC++ Data Collector

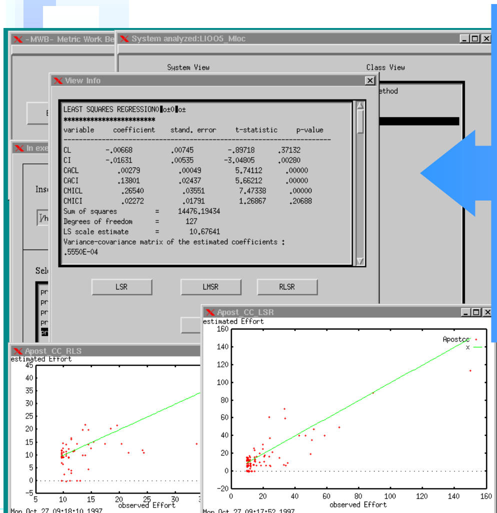


Distributed Database (JAVA):

- Collects Real Effort;
- Collects Number of Fault;
- etc.

 21

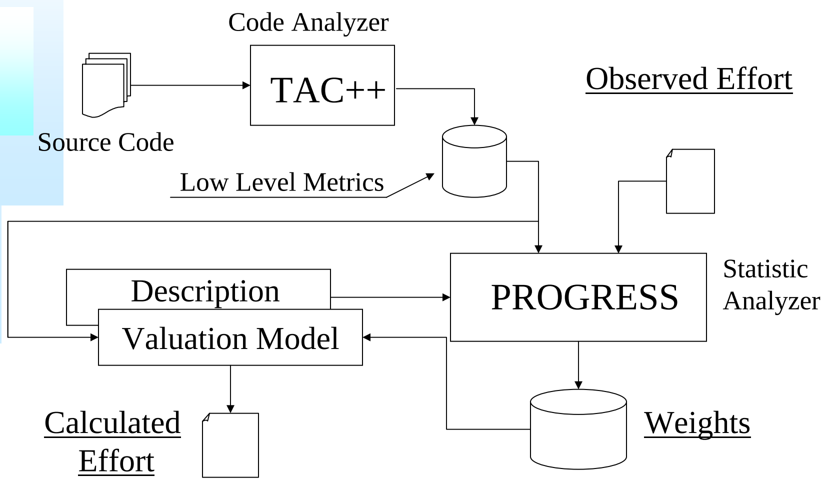
TAC++ Statistical Analyzer



- Correlation analysis among real data and user defined metrics;
- Multi linear regression analysis with LRS optimization;
- Evaluation of weights during the various phases of system development.

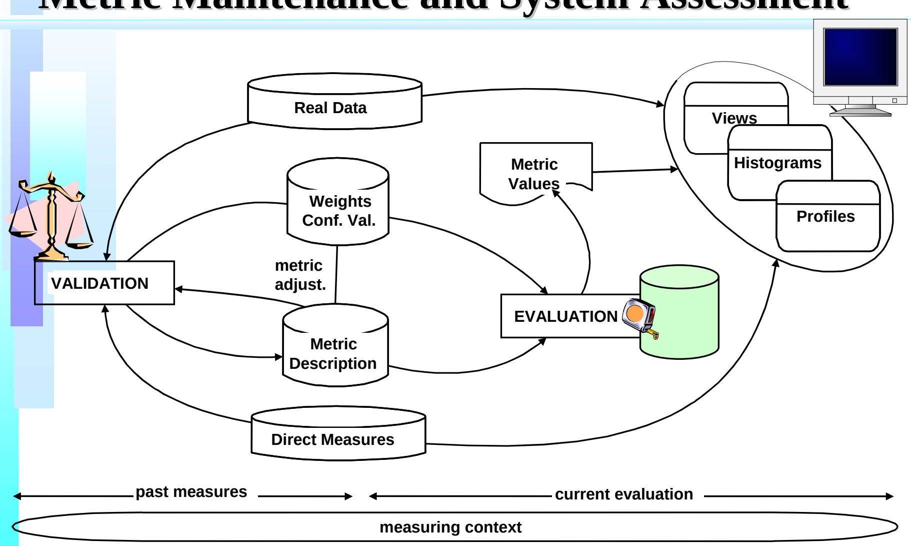
 21

Metric Calibration



 21

Metric Maintenance and System Assessment



 21

Statistical Analysis (2)

The general multiregression problem can be summarized by the following formulas:

Given the Real Effort y_i

$$y_i = w_1 x_{i1} + w_2 x_{i2} + ... + w_p x_{ip}, \quad i = 1,2,\ldots,n$$

we want to estimate the weights,

$$\hat{y}_i = \hat{w}_1 x_{i1} + \hat{w}_2 x_{i2} + ... + \hat{w}_p x_{ip}, \quad i = 1,2,\ldots,n$$

in order to:

$$\underset{\hat{w}_1,\ldots,\hat{w}_j,\ldots,\hat{w}_p}{Minimize} \sum_1^n \left\{ y_i - \sum_1^p \hat{w}_j x_{ij} \right\}^2$$

Statistical Analysis (3)

A more compact expression is:

$$\left(\underline{y} - X\underline{\hat{w}}\right)^T \underline{X_k} = 0 \quad \forall k, \quad \Rightarrow \quad \underline{\hat{w}} = \left(X^T X\right)^{-1} X^T \underline{y},$$

At this point we can estimate expected value and variance of the weights:

$$E\left[\underline{\hat{w}}\right] = \underline{\overline{w}}, \quad Var\left[\underline{\hat{w}}\right] = \sigma^2 \left(X^T X\right)^{-1},$$

Where σ^2 is the variance of y; correlation analysis is now easy.

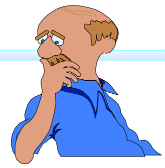
Statistical Analysis (4)

σ^2 can be estimated by $s^2 = \frac{1}{n-p} \sum_{i=1}^{n} r_i^2$, $r_i = y_i - \hat{y}_i$, $i = 1, \ldots, n$

The variable t is a student t, $t = \frac{\hat{w}_j - w_j}{\sqrt{s^2\left(\left(X^T X\right)^{-1}\right)_{jj}}}$, $j = 1, 2, \ldots\ldots, p$

A confidence interval can be defined for the weights

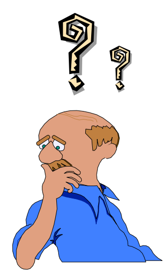
$$\hat{w}_j - t_{(n-p)\left(1-\frac{\alpha}{2}\right)} \sqrt{s^2\left(\left(X^T X\right)^{-1}\right)_{jj}} < w_j < \hat{w}_j + t_{(n-p)\left(1-\frac{\alpha}{2}\right)} \sqrt{s^2\left(\left(X^T X\right)^{-1}\right)_{jj}}, \quad j = 1, 2, \ldots, p$$



- Sarà più economico sviluppare ad oggetti?
- Quanto costerà il processo di inserimento?
- Potremmo realizzare sistemi ibridi?
- Potremmo convertire il vecchio a costi ragionevoli?
- ....
- ...
- Di chi possiamo fare a meno per mandarlo ad un corso sull'OOP?
- Su quale progetto possiamo provare l'Object Oriented?
- Quale linguaggio? Java, C++?
- CORBA, COM, JAVA ?

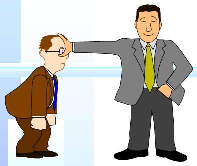
21

Ma ……..



- Dovremmo adottare una metodologia OO?
  - Quale?, perche'? ….
- Dovremmo acquisire nuovi strumenti CASE?
  - Quali?, perche'? ….
- La Qualità?
  - Si misura?, si ha gratis?,
  - come la si misura?, strumenti ?, …..
- E' vero che vi è un risparmio a causa del riuso?
  - lo pago in qualche modo?
- Dovremmo modificare anche la fase di testing?
  - Come?
- ...

22

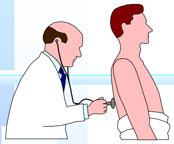


C'è anche chi pensa che:

- Si potranno utilizzare i vecchi strumenti
- L'OO è solo un'etichetta, basta dire che lo stiamo utilizzando.
- Dal mio punto di vista non cambia niente è solo un problema di programmazione.....
- Figurati se per utilizzare l'OO sara' necessario cambiare il modo di pianificare lo sviluppo....
- Fra qualche mese si sgonfiera' ......

Ma in realtà:

- Ins. dal Basso ...... programmatori (corso OO), vecchie notazioni (metodologie), senza gestione (gestione tradizionale), "*lo abbiamo adottato ma non ha dato risultati*", ***dice il capo.***
- dall'alto....manager, corso OO, adattamento strumenti e gestione orizzontale tradizionale, "*l'OO è troppo costoso e complicato non conviene inserirlo*", ***dice il capo.***
- dall'alto....manager, ..impone l'OO, uso di compilatori, senza metodologia, disaccordo fra management e programmatori, "*l'importante è che noi utilizziamo il ...++",* ***dice il capo.***
- i piccoli si difendono a spese della metodologia, della documentazione, del testing, del controllo di processo di produzione, "*sono cose inutili e hanno costi inaccettabili*", ***dice il capo.***



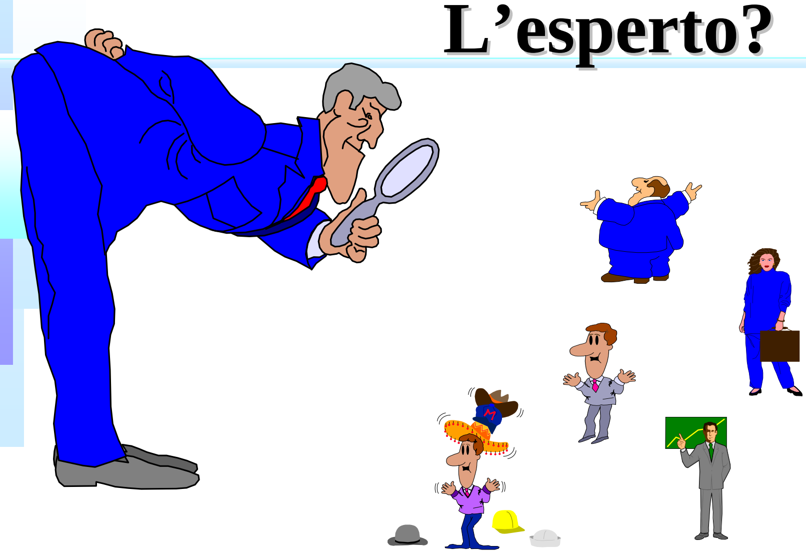
Dice il saggio:

Ogni caso deve essere valutato

- Riformazione di manager, med-manager e programmatori
- Ridefinizione del modello manageriale per i progetti
- uso di una metodologia di sviluppo
- realizzazione del manuale di sviluppo/qualita'
- inserimento controllato con metriche oggettive per: costi, concetti dell'OOP, rendimenti, etc.
- realizzazione di un gruppo per il riuso
- controllo della qualita e del processo di sviluppo

controllo di un "esperto" su un progetto pilota

L'esperto?



2) Identificazione dei sottosistemi.

- Valutazione strutturale piu' che OO del sistema PM con altri che copriranno il ruolo di SSM.
- Sottosistemi hardware e software
- Assegnazione dei sottosistemi ai SSM in base alle precedenti esperienze

3) Analisi dei sottosistemi.

- Ogni SSM analizza il suo SS per identificare classi nel dominio
- Le gerarchie identificate derivano da viste limitate del problema
- Fusione delle varie gerarchie di classi in un'unica gerarchia
- Riassegnazione dei SS orientati agli oggetti ai SSM (massimo 15 classi per ogni SS, 3+SSM per SS, attenzione alle classi importanti del sistema: key, engine, manager in general)
- Identificazione dei cluster, suttorami, etc.

4) Analisi di alto livello per il riuso

- Valutazione del costo di realizzazione della versione necessaria
- Identificazione delle parti da riutilizzare: librerie, classi e sottosistemi gia' acquisiti e/o realizzati
- Valutazione del costo di adattamento del riusato
- Decisione: fare/riusare, questo puo' implicare una ridefinizione dei SS

5) Selezione ed identificazione degli strumenti

- Linguaggi, CASE tool, development tool, librerie di mercato, lex/yacc, etc. (se non imposti per contratto o specifica, o per competenze acquisite)
- Rivalutazione del rischio tecnologico in funzione delle scelte effettuate.
- Se necessario perche' il rishio e' troppo elevato rispetto alle previsioni di vendita si puo' fare restart dal punto (1) modificando alcune richieste: per esempio diminuendo i requisiti del sistema

6) Definizione del project plan grezzo, considerando:

- l'analisi generale di sistema,
- i SS, le dipendenze strutturali e funzionali SS relative,
- le deadline ed i milestone prefissati in precedenza
- la deadline finale (time to market)

7) Analisi delle risorse per i sottosistemi

- Valutazione del costo di realizzazione di ogni SS, metriche predittive
- Eventuale ribilanciamento dei SS ai SSM e quindi del carico dei Team

8) Realizzazione del project plan

- Utilizzo del Gantt grezzo con i dati del punto (7)
- Identificazione delle sotto attivita'
- Identificazione dei costi per: consulenza, training, licencenze (tools), viaggi, beni di consumo, strumenti/apparecchiature (con piano di ammortamento), etc.
- Il training, puo' dare luogo a task separati

9) Valutazione e selezione delle risorse

- Per ogni SS: composizione del team in base alle competenze delle persone e alla loro compatibilita'
- Realizzazione di team integrati: analisi, design, code, valut., etc., competenze trasversali (team piccoli).
- Allocazione delle risorze in base alle deadline

Macro Life-Cycle

Model:

- Spiral Macro-Cycle and Spiral Micro-Cycle,
- Spiral Macro-Cycle and Fountain Micro-Cycle,
- Spiral Macro-Cycle and micro-optimized-cycle.

- Macro Cycle of 5-8 months
- Well defined goals for each Macro Cycle;

Spiral

- Too complex and complete for the Micro level
- Too expensive for the micro level

Fountain

- Too few formalized and controllable for Macro level
- Not enough controllable to be used in 3 people team

Paolo Nesi's Biography

He is Associate Professor at the University of Florence. He has been a visiting researcher at the IBM Almaden Research Center, USA. He received his doctoral degree in Computer Science from University of Florence, and his Ph.D. from University of Padoa.

He has been Chair of international conferences (CSMR'98, Euromicro Conference on Software maintenance and Reengineering (IEEE, Euromicro, REF); and CSMR'99; Objective Quality 1995, LNCS Springer). He is, and has been, program committee member of several conferences -- among them: IEEE ICECCS, IEEE METRICS, IEEE ICSM, AQUIS, REF, etc. **Paolo Nesi will be the General Chair of IEEE ICSM'2001 in Florence, ITALY.**

He is an editorial board member of international journal and series of books, and guest editor in special issues of international journals.

He holds the scientific responsibility at the CESVIT (High-Tech Agency for technology transfer) for object-oriented technologies and HPCN TETRApc TTN (Technology Transfer Node) of ESPRIT. Paolo Nesi has been involved in several international projects, among them he has been the project manager of MOODS HPCN ESPRIT IV, and the project promoter and responsible for ESPRIT multipartner projects: ICCOC, MUPAAC, OFCOMP/MEPI and IMEASY. He is the project co-ordinator of a multi-University Project on software maintenance.

He has published more than 100 papers on journals and conference proceedings.

He has several collaborations with universities, research centers and industries. He is a member of IEEE,IAPR, AIIA, and founding member of TABOO (Assoc. on Object-Oriented Technologies)