A Repository Based Tool for Re-Engineering towards an Object Oriented Environment

Oreste Signore - Mario Loffredo

ERCIM Database Research Group Workshop
EDRG Workshop 4
"Repositories, methods and tools for systems engineering"

May 3-5, 1993

CONSIGLIO
NAZIONALE
delle RICERCHE

Istituto *CNUCE*
via S. Maria, 36
56126 Pisa (Italy)

Ierapetra, Crete, Greece

CONSIGLIO
NAZIONALE
delle RICERCHE

Istituto *CNUCE*
via S. Maria, 36
56126 Pisa (Italy)

Contents

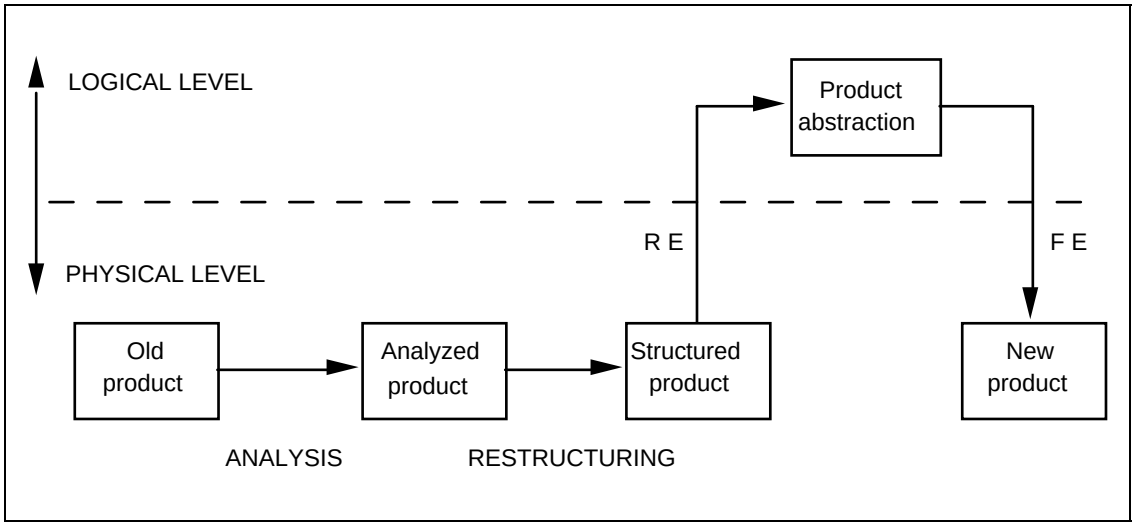
- **Re-engineering, Object orientation, Repository**

- **Re-engineering towards an O-O environment**
  - **Motivation**
  - **Related work**
  - **Problems**

- **The TROOP tool**
  - **The Re-Engineering process**
  - **The architecture**
  - **The Repository**

- **Conclusions**

The Re-Engineering process



Objectives of Re-engineering:

- **Better manage portfolio of existing systems.**
- **Provide automated assistance for maintenance.**
- **Reduce maintenance errors and costs.**
- **Increase productivity of system maintainers.**
- **Make system easier to understand, change, test.**
- **Enable system conversion and migration.**
- **Improve maintenance staff morale.**
- **Enforce adherence to standards.**
- **Improve response to maintenance request.**
- **Protect and extend system life.**
- **Use CASE to support existing systems.**
- **Reuse existing system components.**

Object-Oriented: why?

p O-O design style h many desirable code requirements

- **Modularity**
  - *system naturally decomposed in modules (classes)*

- **Extendibility**
  - *easier reuse of definitions by inheritance mechanism*
  - *addition of new specialisations by type polymorphism*

- **Integrability**
  - *interactions only via well defined interface*
  - *hiding of the implementation details*

- **Robustness**
  - *loose coupling and reduced number of connections between the various classes eliminate the danger of the side effects*

- **Reusability**
  - *consistent support of several kinds of reuse*:
    - at high level, the inheritance supports the modelling of generalisation and specialisation relationships
    - at low level, the inheritance supports the reuse of an existing class as a basis for the definition of a new class.

Three kinds of re-engineering

Depending on their target:

- **reverse re-engineering:**

  target: *the system itself*

  possibly a final version implemented in a different imperative language

  benefits: *re-documentation or re-design*

- **reuse re-engineering:**

  target: *a new system, but implemented maintaining the top-down design style*

  benefits *reduction of the implementation costs and time for future applications*

  reuse of knowledge and design elements taken from previous projects

- **object-oriented re-engineering:**

target: *the system itself, but implemented according to the object oriented methodology*

benefits: *object oriented "philosophy"*

Repository: a must?

- **reverse re-engineering:**

  content: *diagrams, annotations, code, etc.*

  problems: *from poor documentation to the formal repository?*

- **reuse re-engineering:**

  content: *potential reusable components, abstractions, election criteria, metrics*

  problems *identification, qualification and selection of the reusable component*

- **object-oriented re-engineering:**

  content: *classes, attributes, methods*

  problems: *mechanisms for retrieval of classes, navigation in the class space, etc.*

may an intermediate repository for the storage of “canditates” be useful?

OORE: related work

p **Jacobson I., Lindström F.:**
***Re-engineering of old systems to an object-oriented architecture*, Proceedings of OOPSLA' 91**

- *A 1st step of RE yelds a more abstract description of the system*
- *In 2nd step, reasoning at a more abstract level about the changes in functionalities*
- *3rd step: forward engineering*

Alabiso M.:
***Transformation of Data Flow Analysis Models to Object Oriented Design*, Proceedings of OOPSLA' 88, September 25-30 1988**

- *Hybrid Software Life Cycle model, with mapping of Data Flow analysis models into O-O design techniques*

Informal documentation must be carefully examined.

High level documentation must be available, complete and consistent.

p **Liu S-S., Wilde N.:**
***Identifying Objects in a Conventional Procedural Language: An Example of Data Design Recovery*, Proceedings of IEEE Conf. on Software Maintenance, San Diego, November 26-29 1990**

- *The approach consider sets of strongly connected data types or data structures as candidate objects and the associate procedures as their methods .*

A further refinement is necessary in order to avoid that potential objects result “too big”.

Software components classification and repositories: related work

p **Biggerstaff T.J.:**
Design Recovery for Maintenance and Reuse, **IEEE Computer, (July 1989)**

- *design recovery is essentially a human task*
- *identification of modules and "software artifacts"*
- *population of the reuse and recovery libraries*
- *analysis of informal documentation*

p **Sedes F.:**
A Hypertext Information System for Reusable Software Component Retrieval**, DEXA'92 - Valencia, September 1992**

- *retrieval on a hypertext containing heterogeneous documents*
- *both faceted and hierarchical classification*
- *associative thesaurus*
- *term weighting*

p **Basili V.R., Caldiera G., Cantone G.:**
A Reference Architecture for the Component Factory**, ACM TOSEM, Vol. 1, N. 1 (January 1992)**

- *the component factory*
- *heterogeneous reusable products and reusable experiences are stored in a repository and made accessible*

p **Burton B.A., Wienk Aragon R., Bailey S.A., Koehler K.D., Mayes L.A.:**
The Reusable Software Library; **IEEE Software, (July 1987)**

- *attributes of every reusable software component stored in a repository*
- *classification of components: hierarchical + keywords*

p **Helm R., Maarek Y.S.:**

Integrating Information Retrieval and Domain Specific Approaches for Browsing and Retrieval in Object-Oriented Class Libraries **, Proceedings of OOPSLA' 91**

- *combination of information retrieval and domain specific approaches to retrieve classes in an O-O library*
- *browsing based on class functionalities*

The TROOP tool

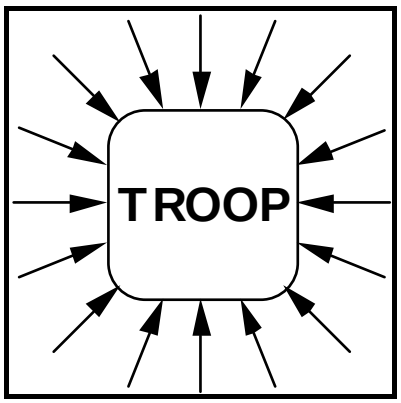
Tool for

Re-engineering towards

Object

Oriented

Paradigm



- **emphasis on data**
  - *data reverse engineering is easier*
  - *existing objects must correspond to some data structures*

- **a fundamental aspect: capture the semantics**
  - *too difficult in a purely automatic way*
  - *access to informal documentation*
  - *"tricky code" obscures design issues*
  - *human intervention is required*

The Re-Engineering process

p *Identification of objects and fields*

1. **Identification of the data structures used by different modules of the existing programs**
   (global variables, record description structures, data structures most used as actual parameters).

   May database structures correspond to objects?

2. **The inheritance hierarchies can be established on the basis of a type classification based on characteristics like access method, scanning, storage ([Meyer]).**

p *Identification of methods*

1. **Arranging the modules by taking into account:**
   - *their size, expressed as Lines Of Code (LOC);*
   - *their depth in the Structure Chart;*
   - *their reuse frequency*

2. **Slicing the modules, starting from the modules derived from the previous step and on the basis of the variables identified in the object identification phase.**

3. **Identifying the code chunks as potential methods and assigning to them a name, a set of keywords, the name of the potential "objects" it is operating on**

(keywords are extracted from a faceted classification a browser can graphically display and navigate through).

The Re-Engineering process
(cont.)

p *Identification of inheritance hierarchies*

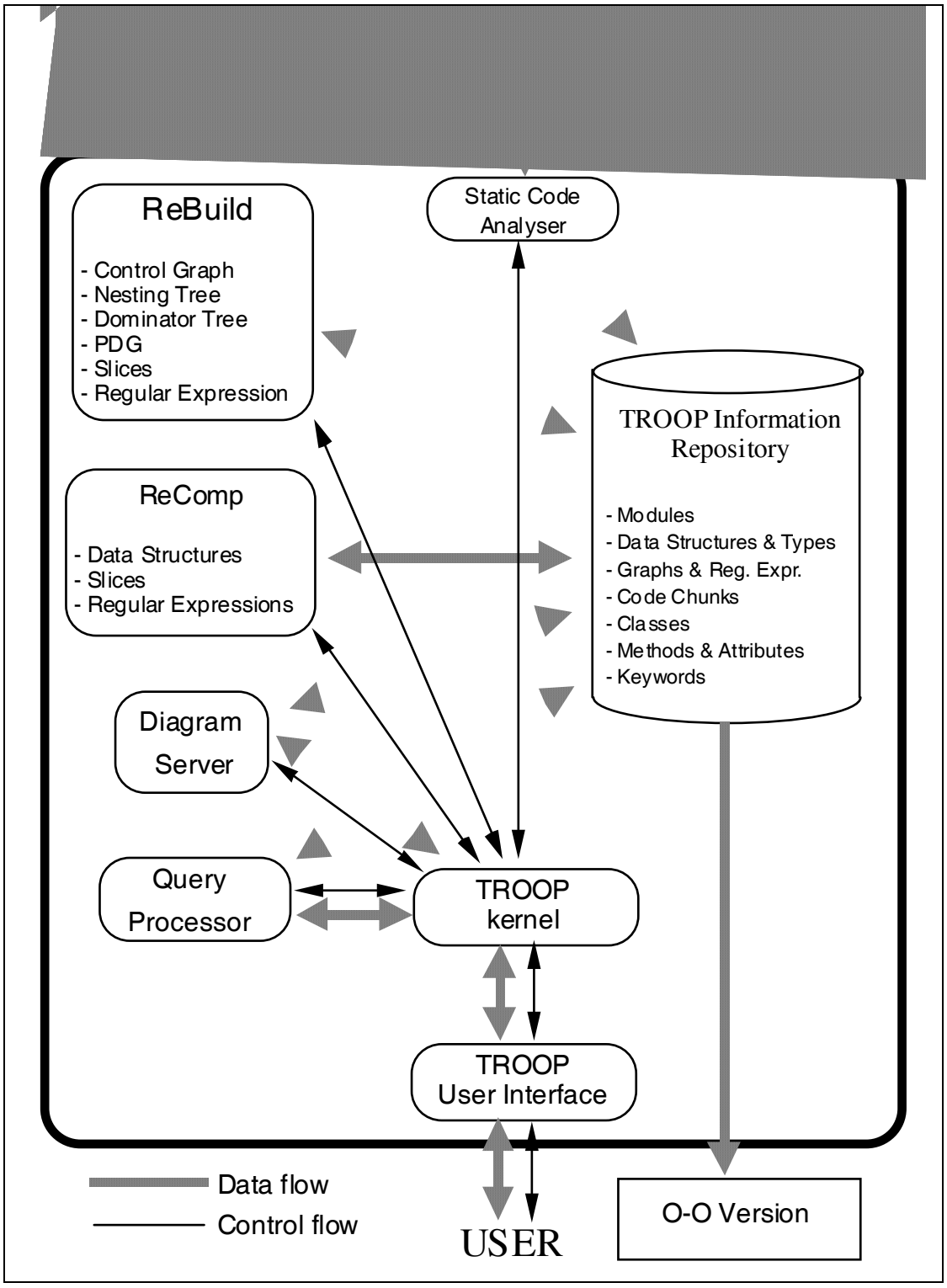
1. **Considering the similarities between the potential methods by making use of:**
   - **types and data structures;**
   - **PDG slices;**
   - **regular expressions at different abstraction levels** *(particular substrings are identified by a single label that gives information about its functionalities)***;**
   - **"formal specifications"**

   Techniques developed in the context of the Information Retrieval area may help in this process

2. **Paying attention to the identification of possible cases of generalisation when considering modules at higher levels in the Structure Chart.**

3. **Rebuilding of the program, eventually restructuring it, expressing it by means of the identified components.**

The architecture of TROOP



Ref.:	**15th International Conference on Software Engineering, Baltimore, May 1993**

T.I.R.
TROOP Information Repository

- **two main sets of entities:**
  - **classical environment**
    (Programs, Modules, Data structures, Code chunks, etc.)
  - **object oriented perspective**
    (Classes, Attributes, Methods).
  - **keywords can characterise both the Code chunks and the Methods**

- **some entities:**

- ***Code chunks***
  pieces of code resulting from the slicing process on the modules

- ***Representations***
  representations of the structure of a Code chunk or of a Module, both as a graph (Program Dependence Graph, Nesting Tree, Control Graph, Dominators Tree) and as regular expression

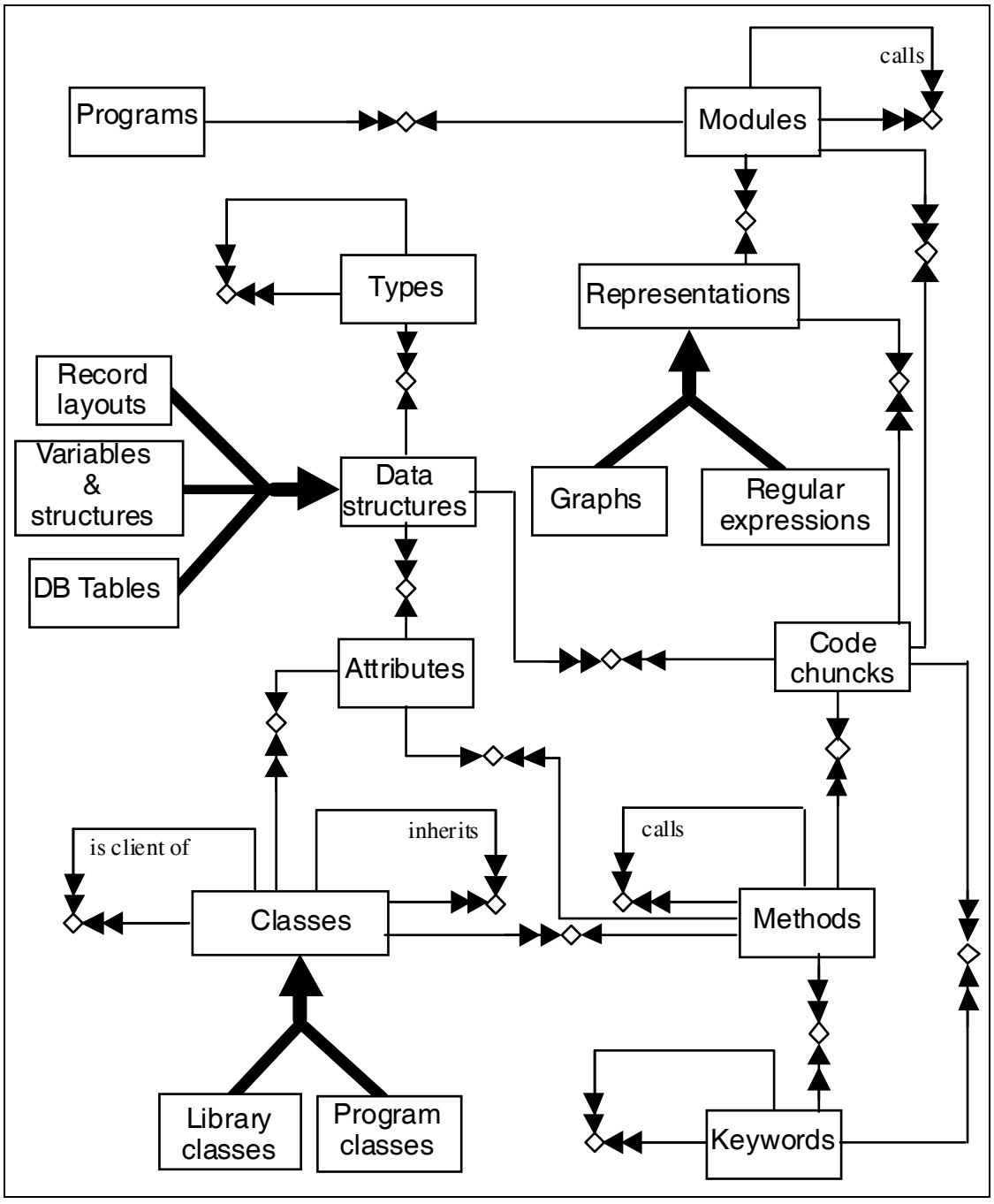
- ***Classes* and *Attributes***
  O-O classes and attributes which have a representation in terms of Data structures in the conventional programs.

- ***Types***
  model the conventional types (int, char, struct, array, etc.).

The unary relationship involving Types model the subtype relationship.

(We conform to the terminology adopted by Eiffel).

T.I.R. Conceptual Schema



Possible implementation environments:

1. **Relational: Sybase**
2. **Logic DB: ConceptBase**
3. **O-O: ObjectStore**

Source Code Analyzer

Task:

1. **Extract information needed to "candidate" objects and methods.**

2. **Insert in the Repository.**

What we extract:

1. **Global variables**

2. **Local variables**

3. **Data types**

4. **Formal parameters**

Re-Build Module

Task:

1. **Building the program representations using the particular description formalism adopted by Diagram Server.**

2. **Slicing algorithm implementation.**

Representations:

1. **Structure Chart**

2. **Control Flow Graphs**

3. **Nesting Trees**

4. **Dominator Trees**

5. **DAGs**

6. **PDGs (Horwitz)**

7. **Slices**

8. **Regular Expressions**

Re-Comp Module

Task:

1. **Comparison between potential objects.**
2. **Comparison between potential methods.**
3. **Developing a similarity measure.**

Evaluation parameters:

1. *Similarity between data structure* (Meyer):

    a) **Access method**

    b) **Scanning method**

    c) **Storage method**

2. *Similarity between potential methods*:

    a) **"Formal specifications"**

    b) **PDG Slices**

    c) **Regular expressions at different abstraction levels**

    d) **Informal description**

Conclusions

- ***Re-engineering could improve the maintainability of the existing software applications, possibly by their reconfiguration.***

- ***Re-engineering towards an object-oriented environment seems to be an interesting and fruitful activity.***

- ***To accomplish this task, we have sketched the architecture of a re-engineering tool (TROOP).***

- ***TROOP is based on a central repository (T.I.R.) that contains information pertaining to both the traditional as well as the object oriented target environment .***

- ***We take into account both formal and informal documents.***