Scomposizione Modulare

Criteri Generali, Approccio Object-Oriented

e cenni all'Approccio Funzionale

Testo di Riferimento:

A. Baruzzo, Analisi e Progettazione di Sistemi Software Industriali – Vol. 1, Create Space, 2017.

Quando scompongo in moduli come faccio? quanti me faccio?

Design Strategies

- Object-oriented design
  - The system is viewed as a collection of interacting objects. The system state is decentralized and each object manages its own state. Objects may be instances of an object class and communicate by exchanging methods.
- Functional design (progettazione strutturata)
  - The system is designed from a functional viewpoint. The system state is centralized and shared between the functions operating on that state.

Le 2 caratteristiche/metriche di qualità: COESIONE ED ACCOPPIAMENTO

Le metriche di coesione e di accoppiamento sono fondamentali sia per l'approccio object-oriented che per l'approccio funzionale

Design Quality

- Design quality is an elusive concept. Quality depends on specific organizational priorities.
- A "good" **design** may be the most efficient, the cheapest, the most maintainable, the most reliable, etc.
- The attributes discussed here are concerned with the **maintainability** of the design.
  ↳ mantenibilità nel tempo
- From a general point of view, quality characteristics are equally applicable to function-oriented and object-oriented designs, WITH MORE OR LESS FOCUS ON THE VARIOUS CHARACTERISTICS.

On the Criteria To Be Used in Decomposing Systems into Modules

D.L. Parnas
Carnegie-Mellon University

This paper discusses modularization as a mechanism for improving the flexibility and comprehensibility of a system while allowing the shortening of its development time. The effectiveness of a "modularization" is dependent upon the criteria used in dividing the system into modules. A system design problem is presented and both a conventional and unconventional decomposition

questo paper introduce i due criteri di scomposizione in MODULI:

- COESIONE
- ACCOPPIAMENTO

1. Cohesion

- A measure of how well a component "fits together". La coesione è una misura del livello di correlazione tra diverse funzionalità presenti all'interno di un modulo, ossia del suo livello di "omogeneità funzionale"
- A component should implement a single logical entity or function.
  ↳ UN MODULO svolge UNA FUNZIONALITÀ, non più cose che creano confusione
- La componente dovrebbe contenere 'tutto e solo' ciò che serve per implementare la relativa Funzione (? cfr principi **SOLID**)
  ↳ così è più facile da mantenere e upgradare. Localizzare gli interventi di manutenzione, guardo solo lì e non altrove
- Perchè? Cohesion is a desirable design component attribute as when a change has to be made, it is localized in a single cohesive component.? ↑ MANUTENIBILITà

? Baruzzo

Richiamo ai Principi SOLIDI e al principio di *single responsability*

I principi SOLID sono intesi come linee guida per lo sviluppo di software estendibile e manutenibile, in particolare nel contesto delle tecniche di sviluppo agile.

Tra i cinque principi troviamo:

Principio di singola responsabilità (**SRP**): afferma che ogni classe dovrebbe avere una ed una sola responsabilità, interamente incapsulata al suo interno.

In altri termini, ogni componente/modulo deve implementare una singola funzionalità, e quindi deve contenere tutto e solo il codice relativo a quella funzionalità, e nulla di più.

Coesione Principio di singola responsabilità

Cohesion Levels (general types)

WEAK

- Coincidental cohesion (weak)
  - Parts of a component are simply bundled together.
- Logical association (weak)
  - Components which perform similar functions are grouped.
- Temporal cohesion (weak)
  - Components which are activated at the same time are grouped.

Cohesion Levels

Medium

- Communicational cohesion (medium)
  - All the elements of a component operate on the same input or produce the same output.
- Sequential cohesion (medium)
  - The output for one part of a component is the input to another part.

STRONG

- Functional cohesion (strong)
  - Each part of a component is necessary (e sufficiente!, tutto e solo, …) for the execution of a single function.
- Object cohesion (strong)
  - Each operation provides functionality which allows object attributes to be modified or inspected.

Baruzzo

Cohesion as a Design Attribute

- Not well-defined. Often difficult to classify cohesion.
- Inheriting attributes from super-classes weakens cohesion: to understand a component, the super-classes as well as the component class must be examined.
- Object class browsers assist with this process.

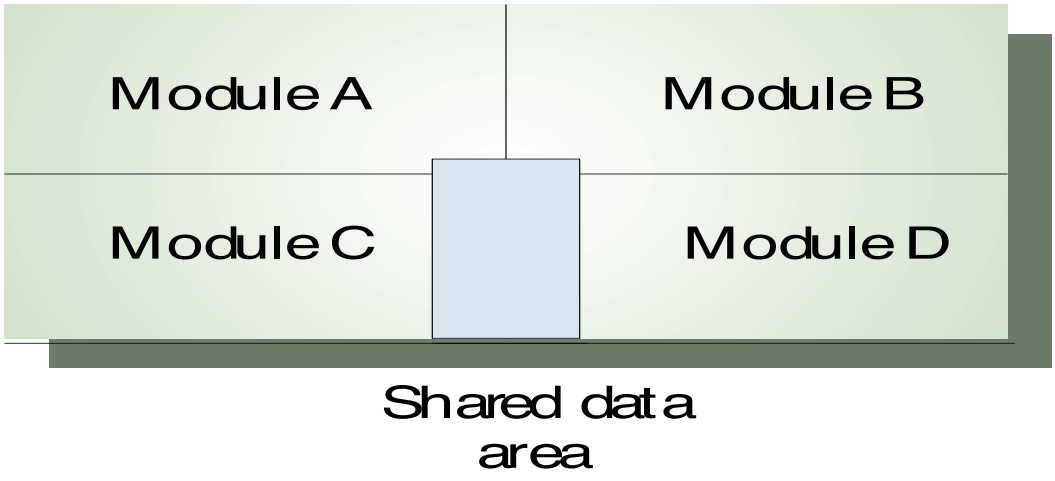
Baruzzo

2. Coupling → intercommessioni MINIME tra moduli (in modo da rendere "indipendenti" i vari moduli)

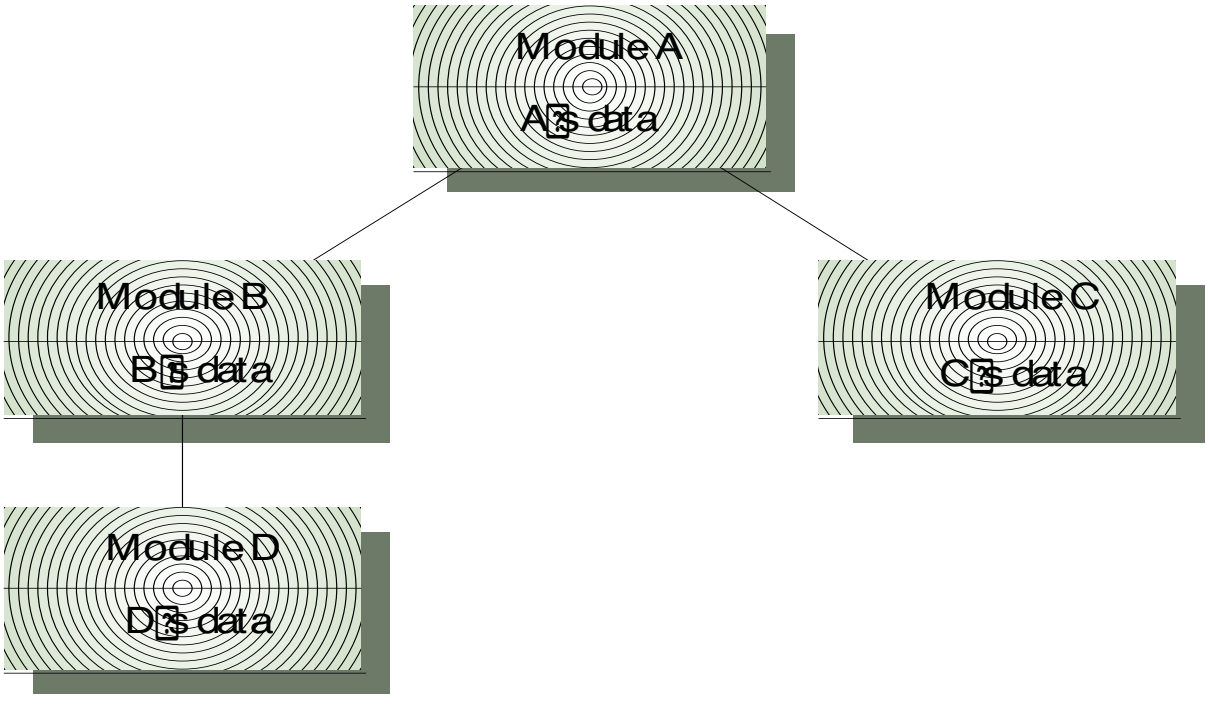
- A measure of the strength of the inter-connections between system components.
- **Loose coupling** (i.e. scarse interrelazioni fra componenti) means component changes are unlikely to affect other components. ? ↑ MANUTENIBILITà
- Shared variables or control information exchange lead to tight coupling.
- Loose coupling can be achieved by state decentralization (as in objects) and component communication via parameters or message passing (i.e. **not** through shared variables).

? Basso Accoppiamento Principio di singola responsabilità

Tight Coupling



Loose Coupling



A. Object-oriented development

Vedi parte del corso del Dr. Baruzzo

APPROCIO FUNZIONALE

B. Function-oriented design

Design with functional units which transform inputs to outputs

IL METODO STRUTTURATO

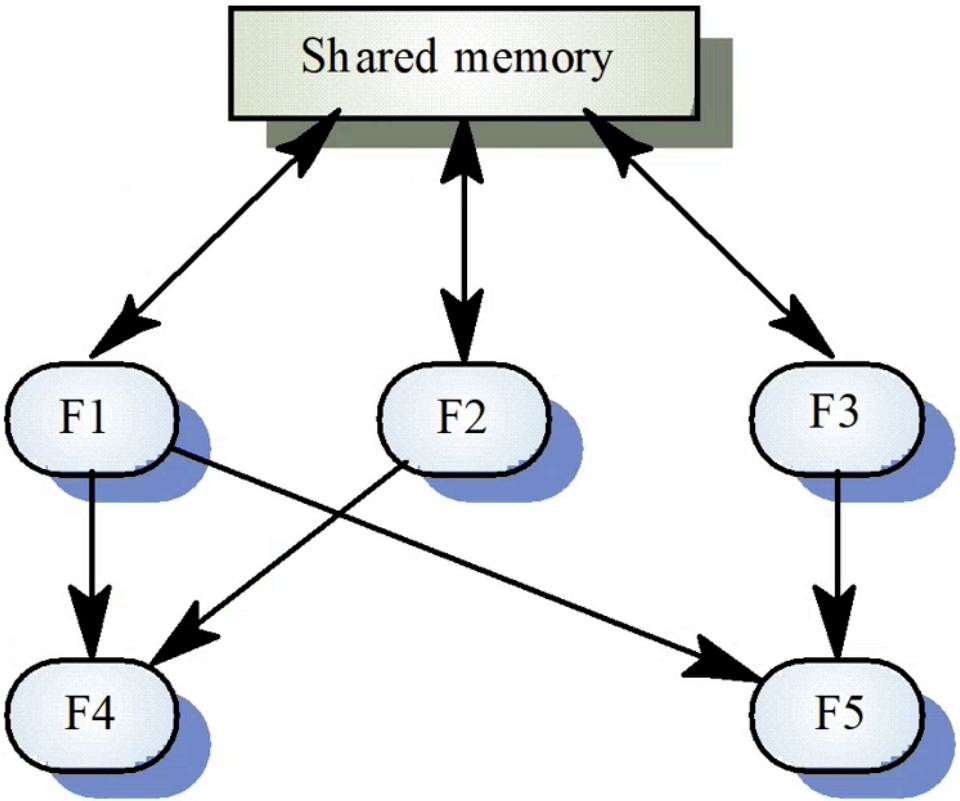
Testo di Riferimento:

A. Baruzzo, Analisi e Progettazione di Sistemi Software Industriali – Vol. 1, Create Space, 2017.

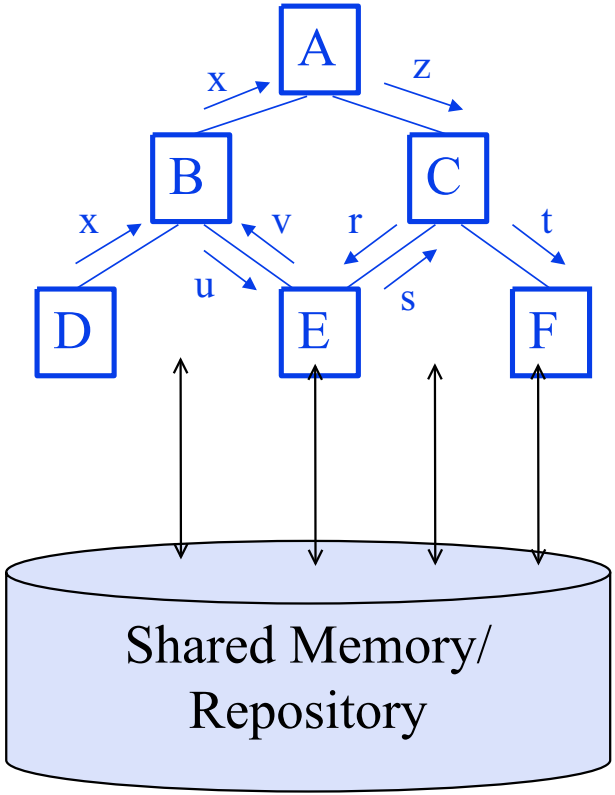
Motivations for function-oriented design

- The **first approach** exploited in SW development: Practised informally since programming began
- **Thousands** of systems have been developed using this approach. Many **legacy** applications were developed using function-oriented design
- **Supported directly** by most programming languages
- Many design methods are functional in their approach
- CASE tools are available for design support

A function-oriented view of design



A function-oriented view of design - 2



Natural functional systems

- Some systems are naturally function-oriented
- Systems which maintain minimal state information i.e. where the system is concerned with processing independent actions whose outcomes are not affected by previous actions (**stateless**)

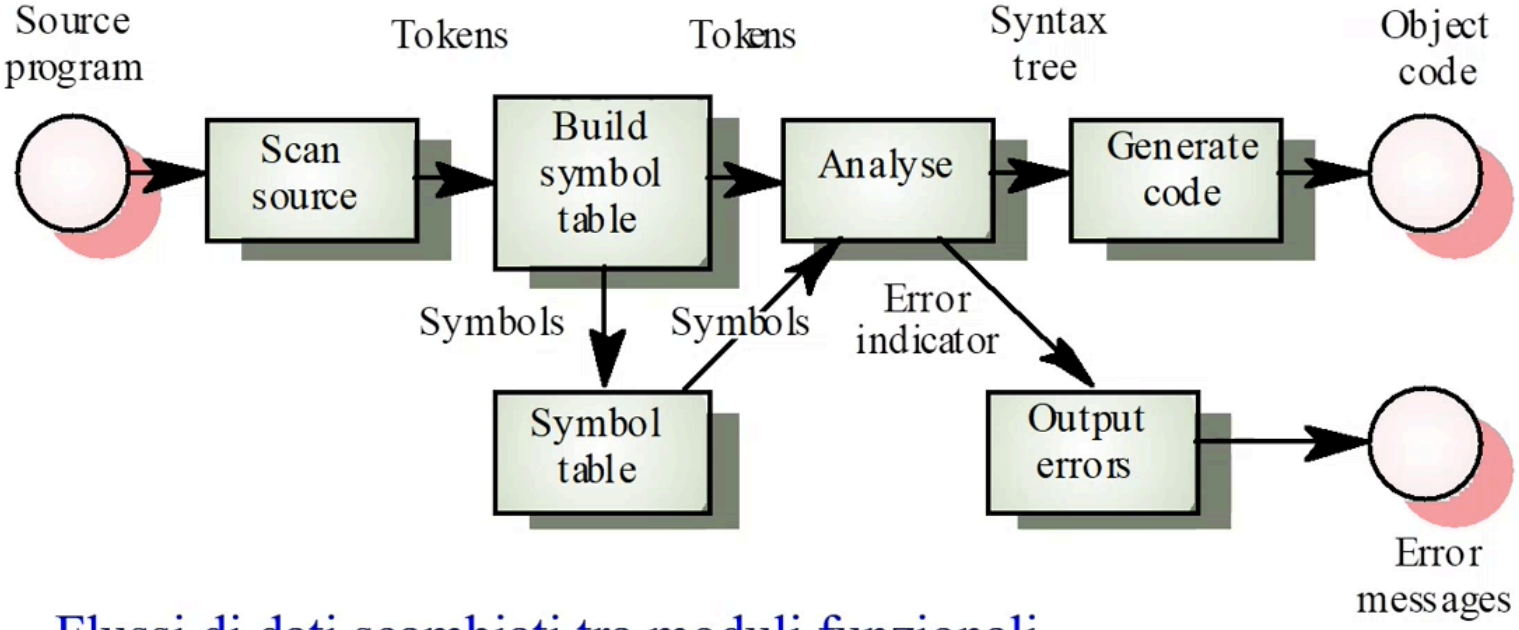
la mia funzione (indipendente) ha un INPUT e fornisce un OUTPUT

- Information sharing through parameter lists and shared repository
- Functions with NO temporal aspect, *i.e. the result of a function invocation is not dependent upon the function's earlier invocations (is not history sensitive, is stateless)*
- **Transaction processing systems** fall into this category. Each transaction is independent. Es. ATM.

TRANSAZIONI BANCARIE

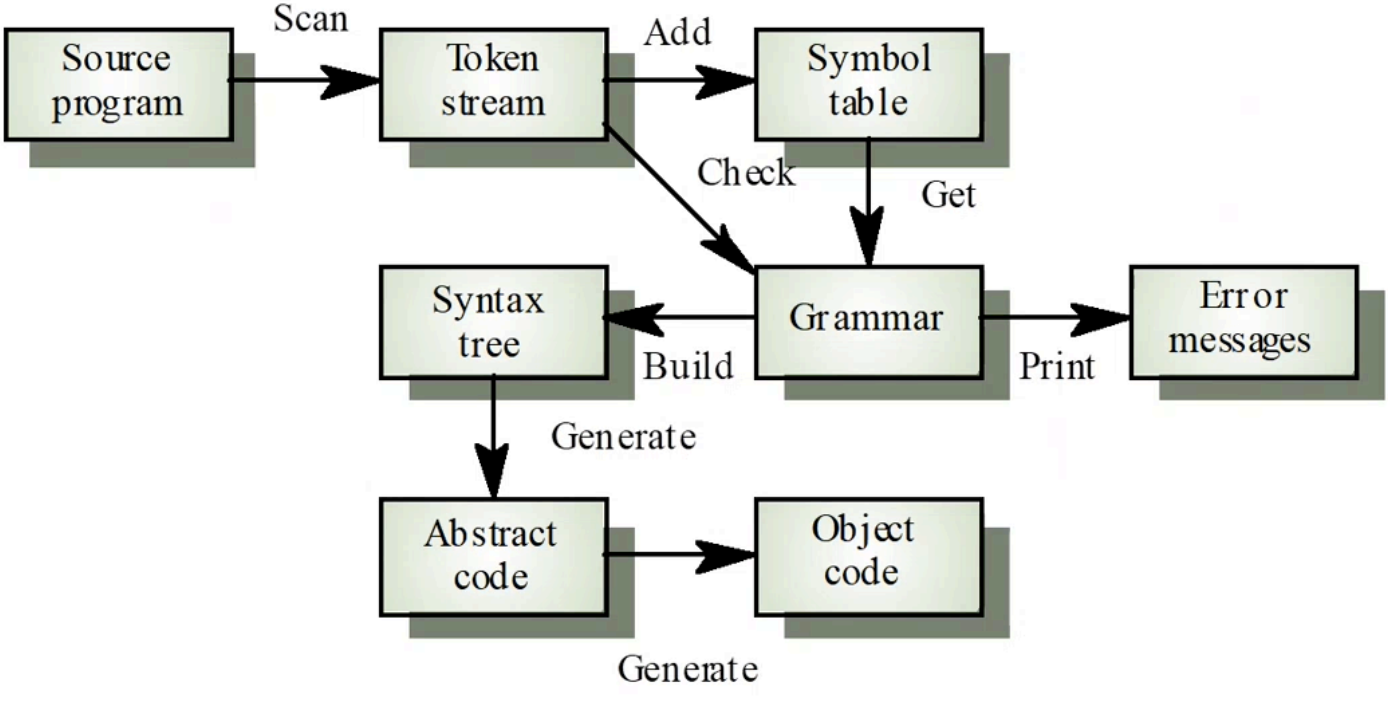
Function-oriented Design is based on a **top- down functional decomposition** style of design

Functional View of a Compiler



Flussi di dati scambiati tra moduli funzionali

Object-oriented View of a Compiler



Richiesta di servizi (invocazione di metodi) tra classi e oggetti

Functional design process

Il metodo strutturato – LE FASI

1. Data-flow design
   - Model the data processing in the system using **DFD data- flow diagrams**, che tipicamente hanno una struttura a grafo.
2. Structural decomposition of modules/code (design strutturato) e **Transfrom Analysis**
   - Transform the DFD into functions and Model how functions are decomposed into sub-functions and represent them by using graphical **structure charts**
3. Detailed design (progettazione esecutiva) → passo i dettagli al programmatore
   - The entities in the design and their interfaces are described in detail (in a **minispec**). These may be recorded in a **data dictionary** and the design expressed using a PDL

segue la codifica della fase di IMPLEMENTAZIONE

The overall development process

[da libro Baruzzo]

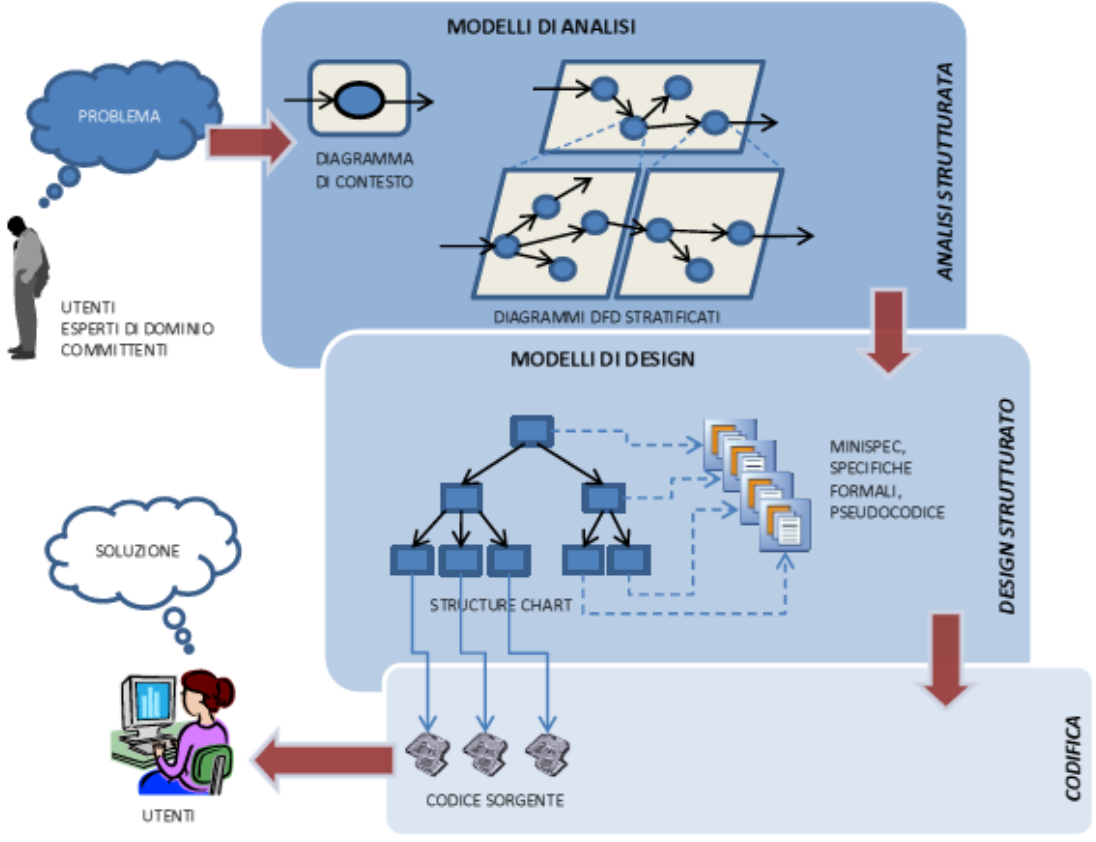


FIGURA 4.18
I diversi modelli creati nel Metodo Strutturato nel passaggio dall'analisi alla codifica

Un esempio dell’intero processo

Example: Regression Testing

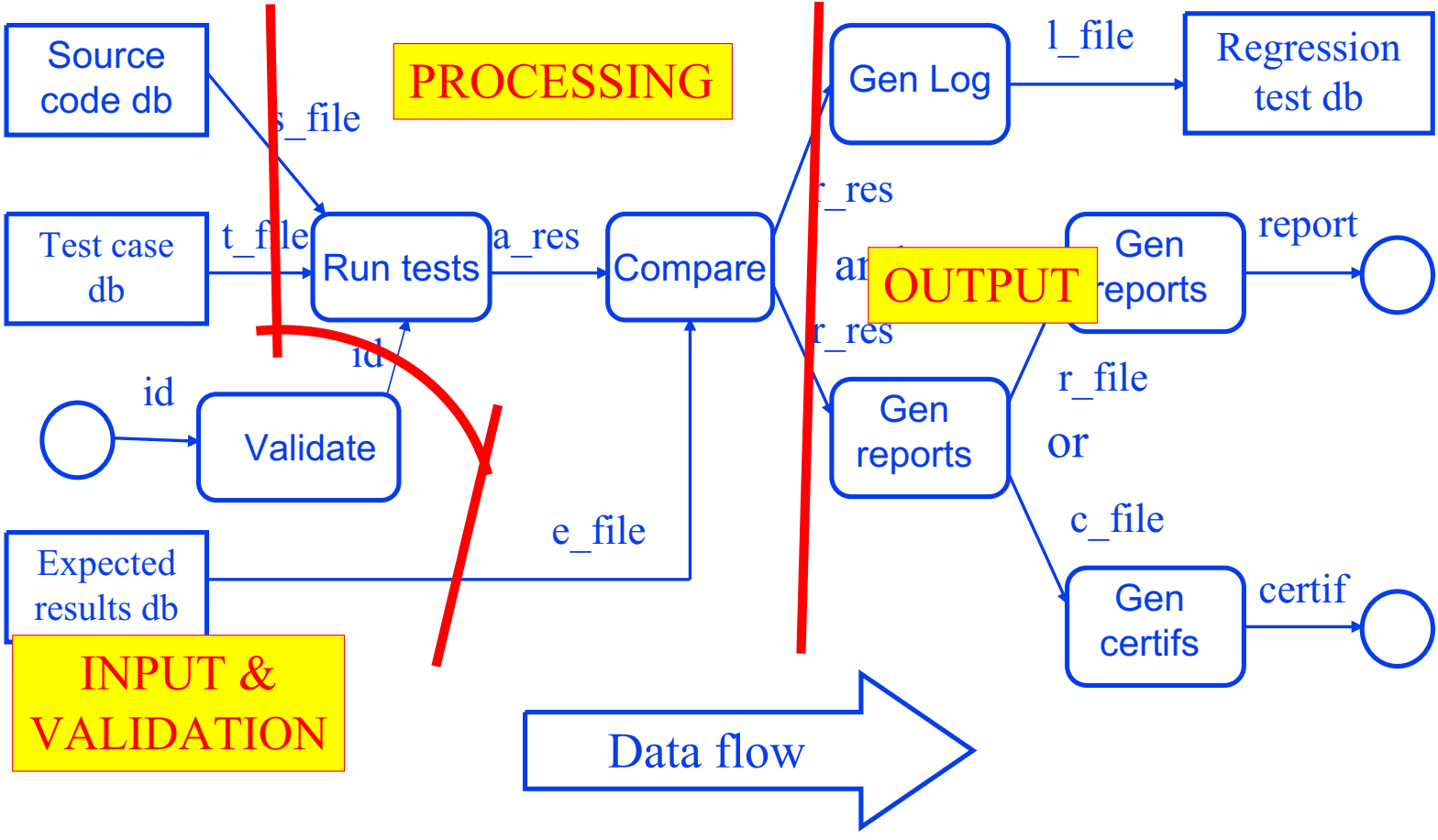
(from A. Ireland, HWU)

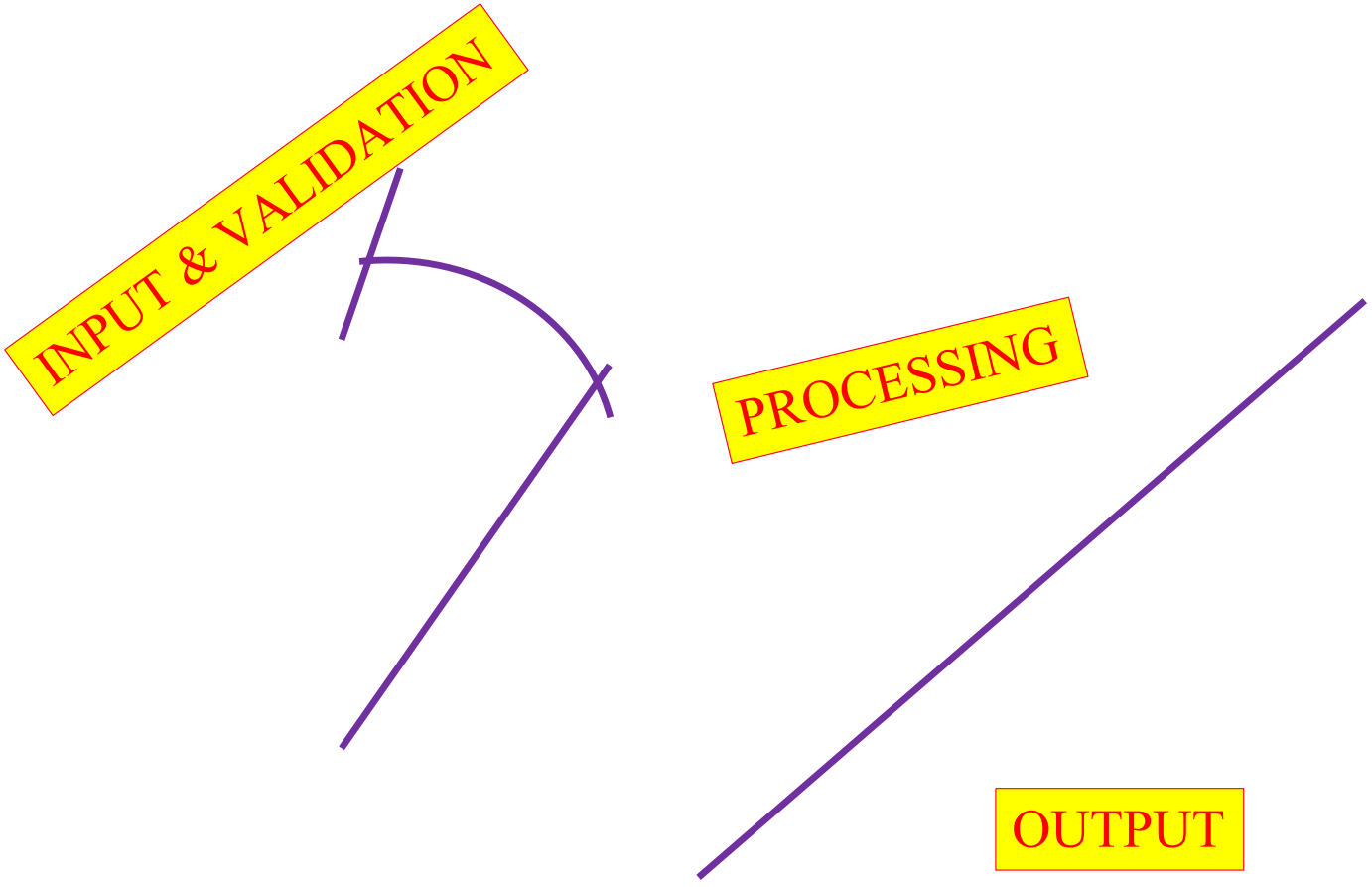
Consider the following system requirements:

1. A user provides a name for a source code base
2. If valid, then the source code base is retrieved from a database along with its associated regression test suit, which is held in a separate database
3. The source code is executed against the regression tests
4. The test results are compared against the expected results, which are held in another database
5. If actual and expected results differ then a report is generated for the user, else a certificate is generated for the user
6. A log file summarizing the regression testing stored in a database

Fase 1- DFD

(from A. Ireland, HWU)

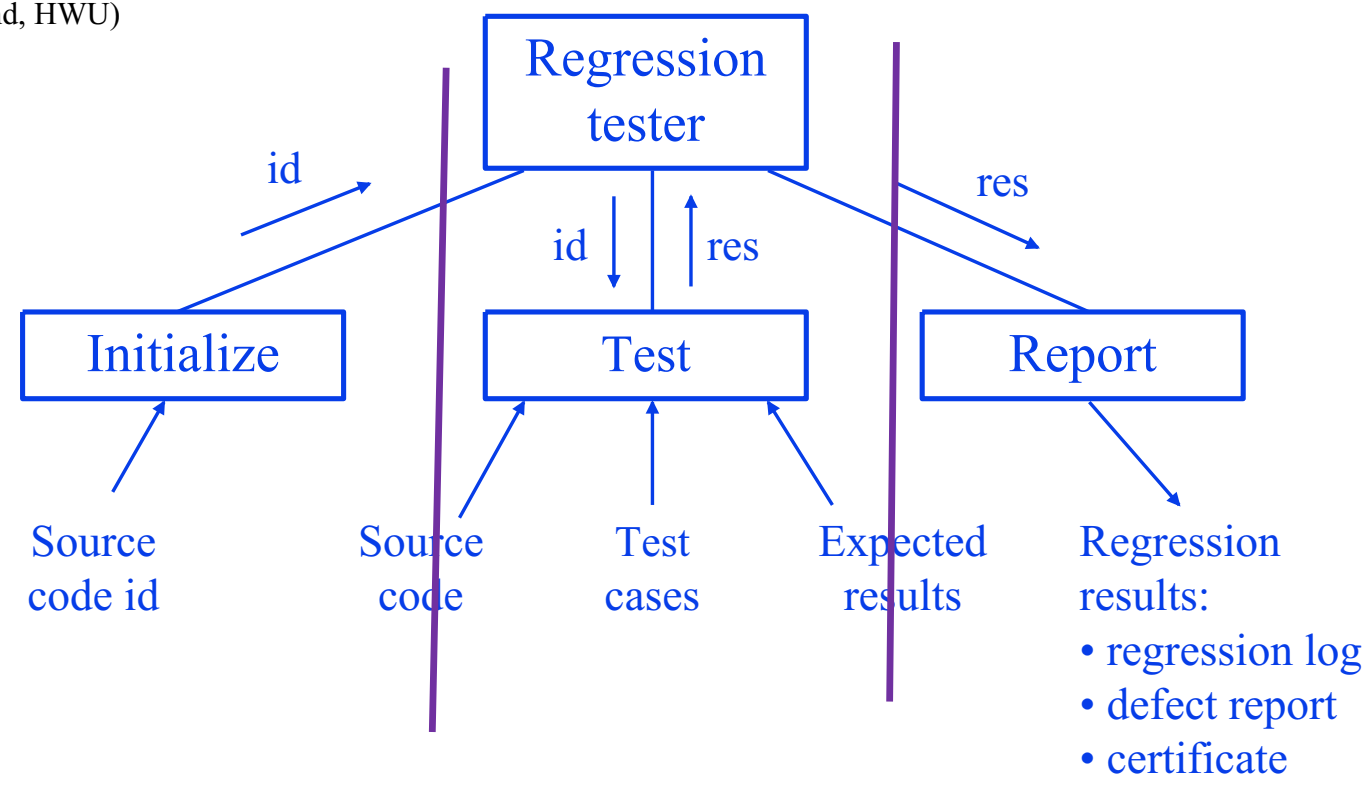



INPUT & VALIDATION
PROCESSING
OUTPUT

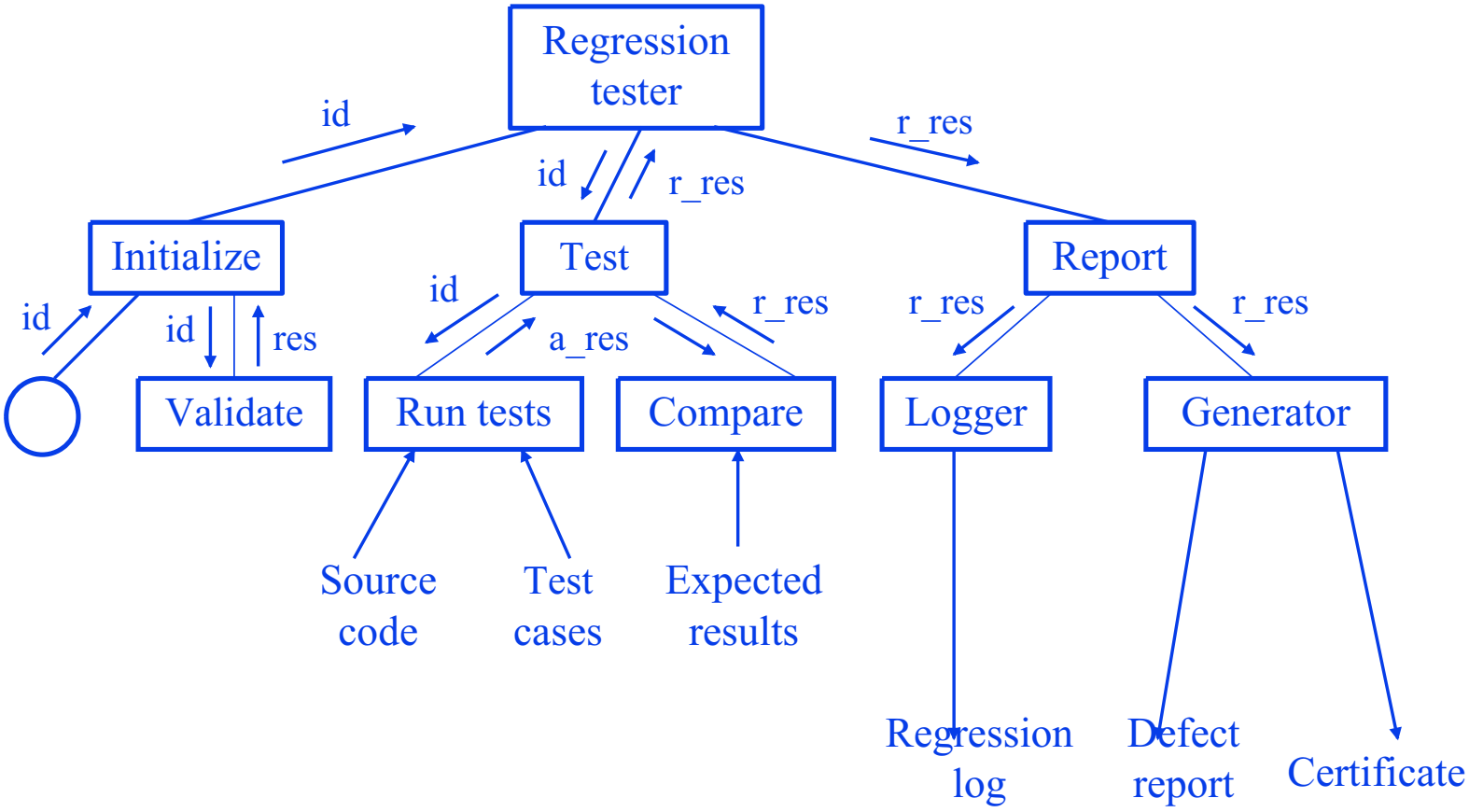
Regression Tester Example

FASE 2 – Trasformazione DFD ? Diagramma della Struttura del sistema (Structure Chart)

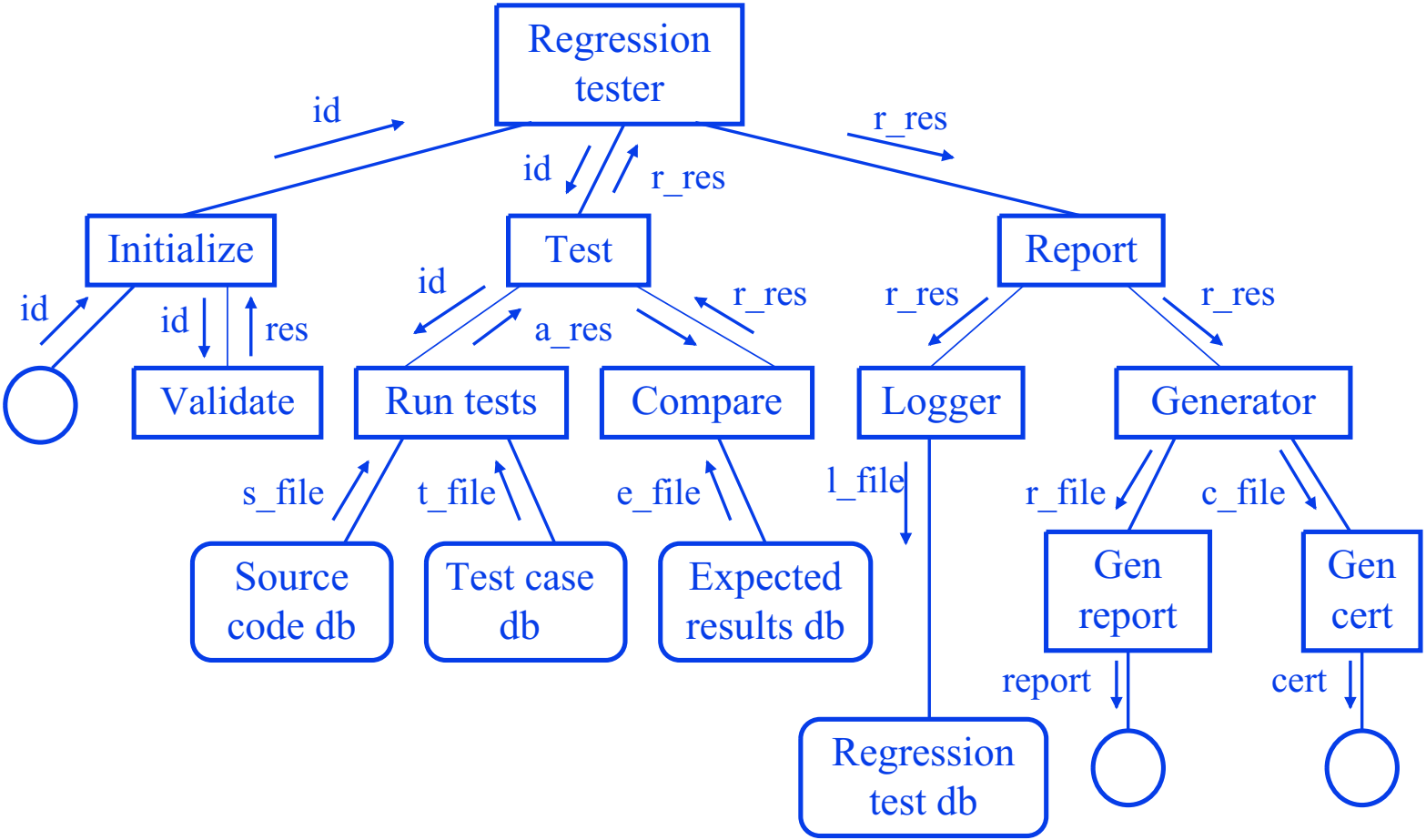
(from A. Ireland, HWU)



Second Chart (from A. Ireland, HWU)



Final Chart (from A. Ireland, HWU)

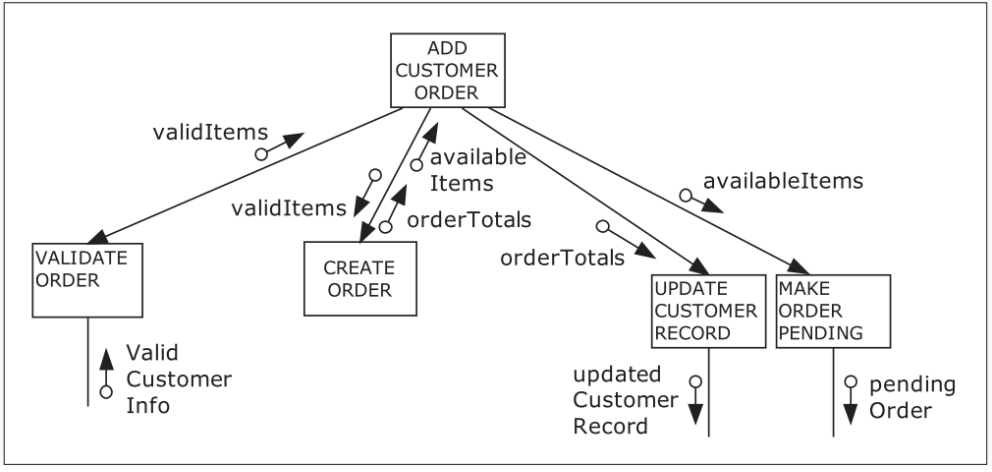


Dettagli sulle modalità di esecuzione – NON svolti lezione

Transform Analysis: da Diagramma DFD a Struttura del Moduli (structure chart)

Nel Metodo Strutturato, la Transfrom Analysis permette di passare dalla fase di analisi al design trasformando i diagrammi DFD che modellano il processi essenziali del sistema in uno **structure chart** che modella invece i moduli software, le loro dipendenze di chiamata e i dati passati.

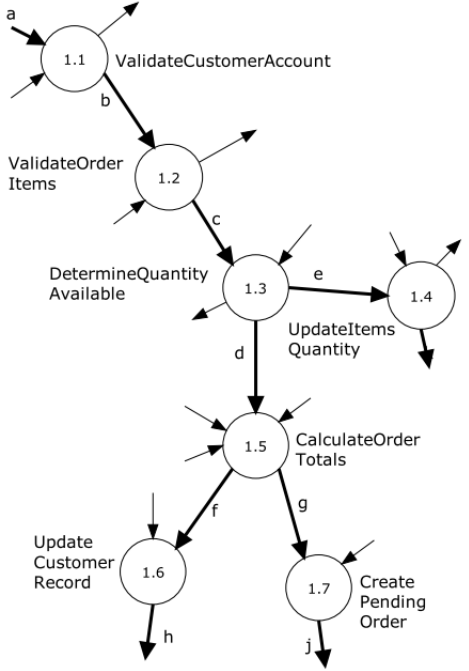
Uno Structure Chart è tipicamente organizzato **gerarchicamente**, come un albero, poiché la scomposizione viene condotta secondo il **Principio di Delega dei Compiti**, che prevede la scomposizione della funzionalità complessa, implementata da un modulo, in sotto-funzionalità la cui esecuzione è demandata ad altri moduli, connessi al modulo principale mediante di dipendenza funzionale.



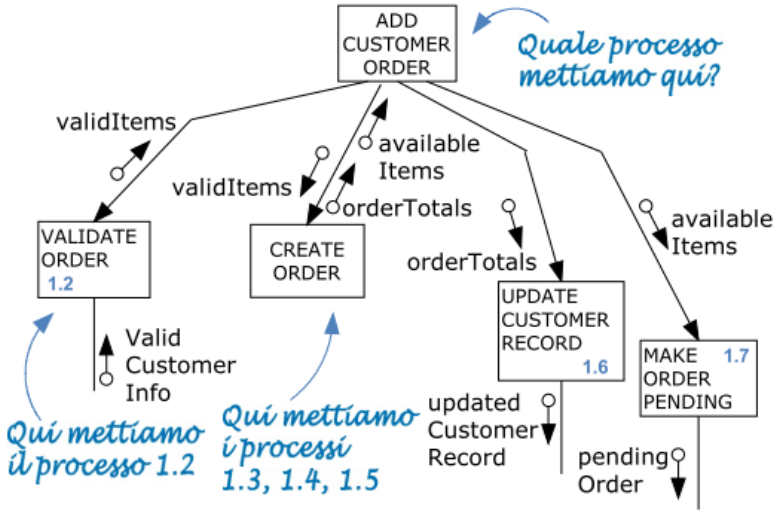
Passaggio da DFD a structure chart

Si passa da una struttura reticolare, a grafo (il DFD) ad una struttura gerarchica ad albero (lo structure chart).

La mancanza di una naturale corrispondenza (isomorfismo) tra queste due rappresentazioni è nota come **impedance mismatch**, e rischia di rendere difficile capire quale elemento di progetto corrisponde a quale elemento di analisi.



(a)*Modello reticolare nell'analisi (DFD)*



(b)*Modello gerarchico nel design (structure chart)*

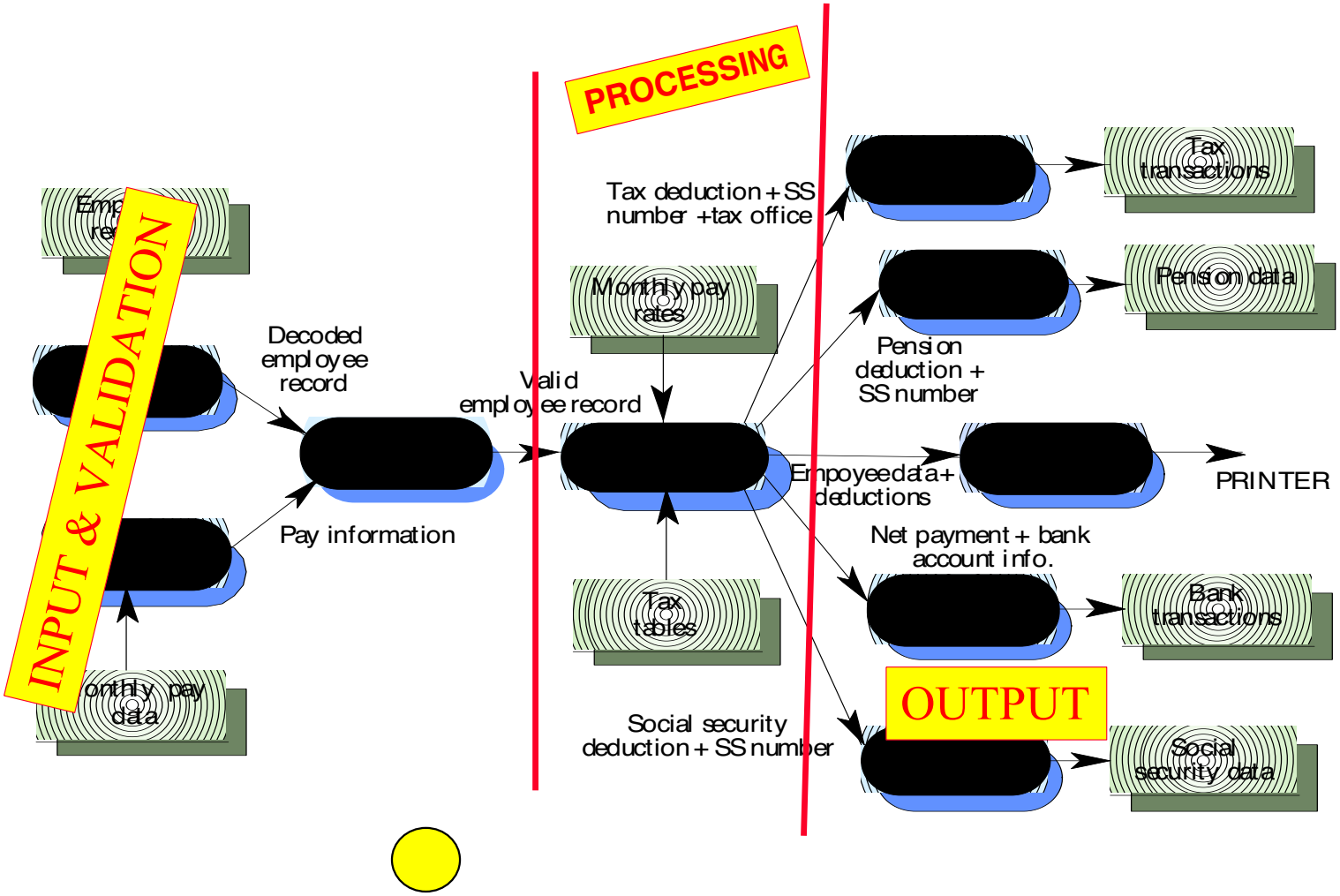
1. Data flow diagrams

- Show how an input data item is functionally transformed by a system into an output data item
- Are an integral part of many design methods and are supported by many CASE systems
- May be translated into either a sequential or parallel design. In a sequential design, processing elements are functions or procedures; in a parallel design, processing elements are tasks or processes

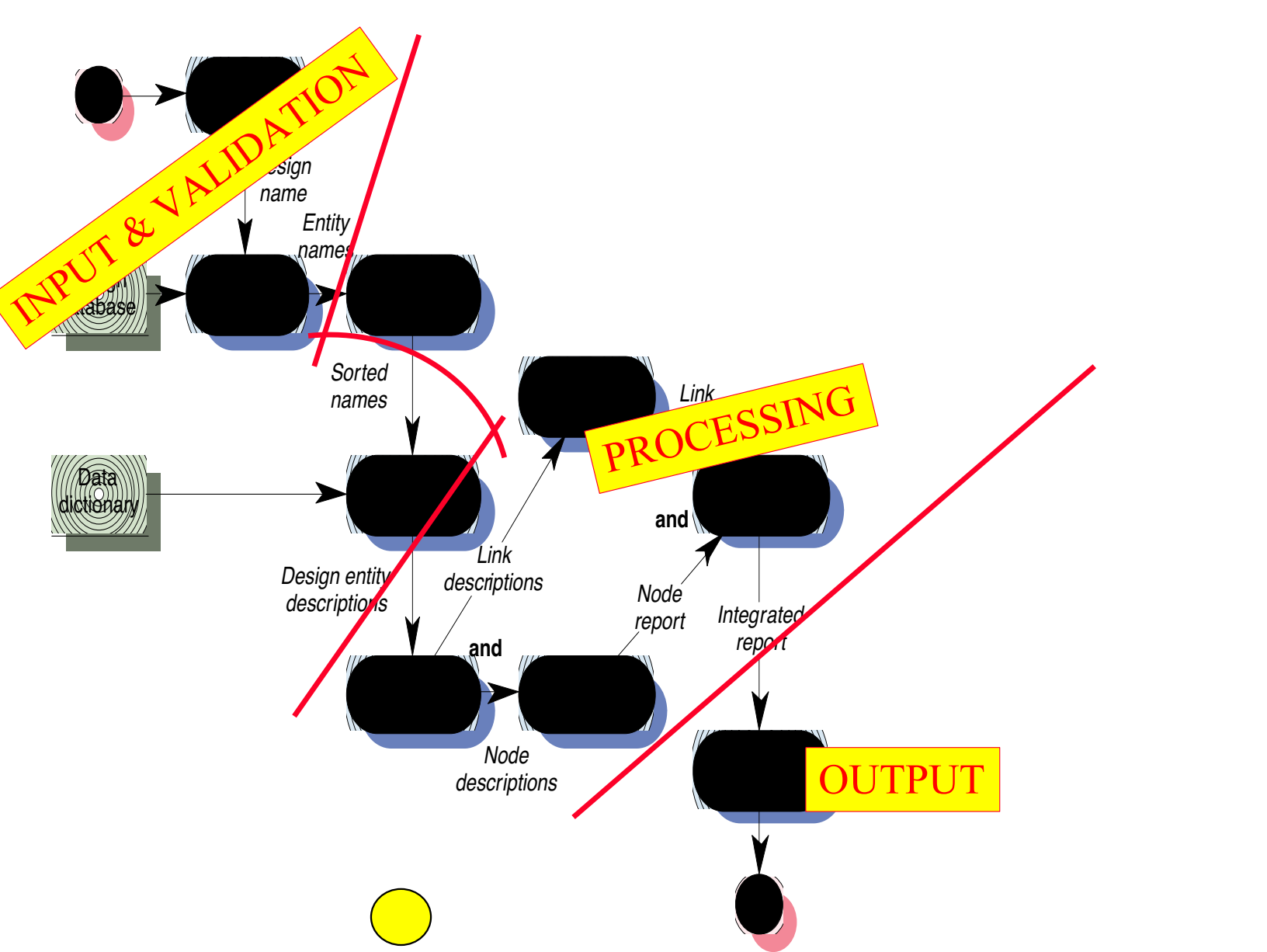
DFD notation

- Rounded rectangle - function or transform
- Rectangle - data store
- Circles - user interactions with the system
- Arrows - show direction of data flow

Payroll system DFD



Design report generator


INPUT & VALIDATION
PROCESSING
OUTPUT
Entity names
Sorted names
Data dictionary
Link
and
Link descriptions
Node report
Integrated report
and
Node descriptions

Exercise: Wanted Persons

(from A. Ireland, HWU)

Consider the following system requirements:

1. A user provides facial features based upon crime reports
2. If valid, then the feature list is used to retrieve a list of relevant feature templates from a database. If the feature list is not valid then an error message is sent to the user
3. The feature template list is used to search for matching offender imagines which are stored within an offender records database
4. Any records for offenders already in prison are then pruned
5. From pruned list, a suspect list is dispatched to the user, while the corresponding records are sent to a “wanted persons” database

Exercise: construct a DFD for the requirements above

2. Transform Analysis and Structural decomposition

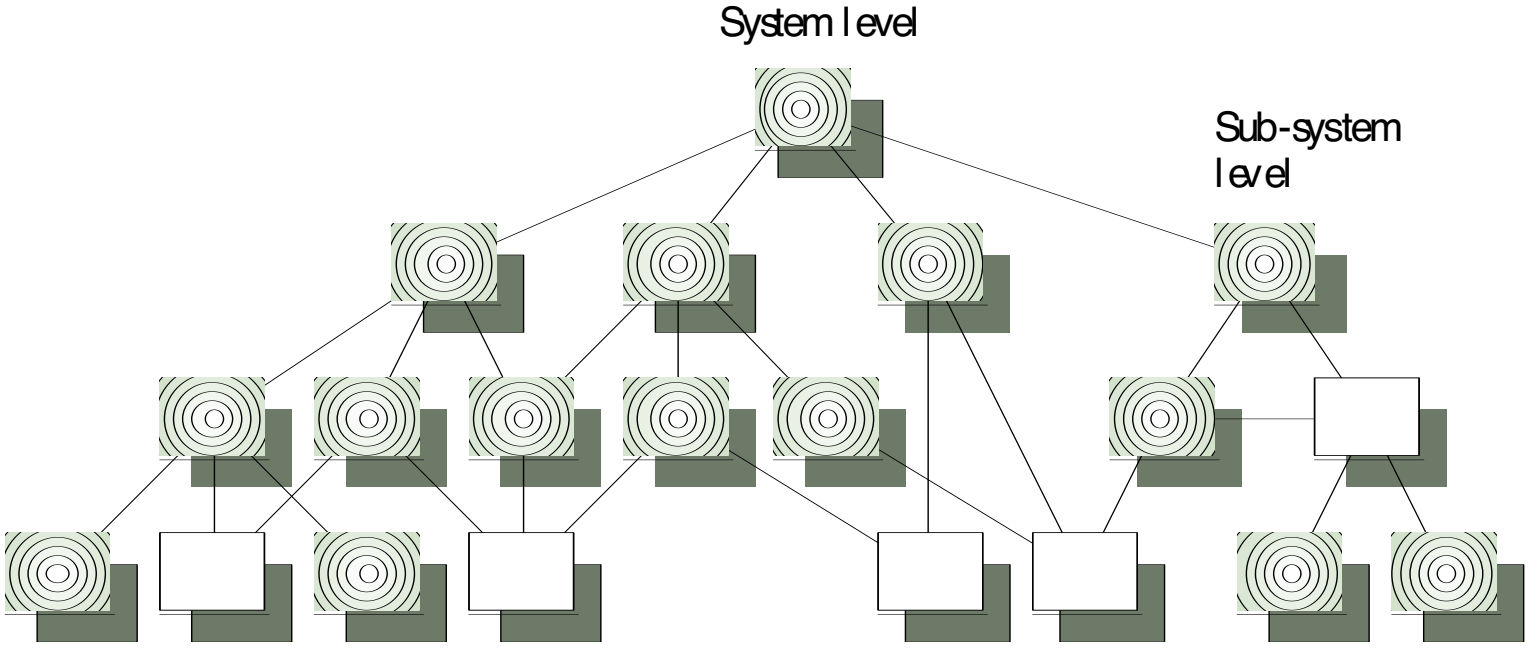
- Structural decomposition is concerned with developing a **model of the design** which shows the **dynamic*** structure (i.e. function calls) of the design units
- This is **not** the same as the static* composition structure
- The general aim of the designer should be to derive design units which are **highly cohesive** and **loosely coupled**
- In essence, transform analysis progressivelly converts a data flow diagram to a structure chart

(*) DINAMICO nel senso delle chiamate (chi chiama chi) e STATICO nel senso della struttura del codice

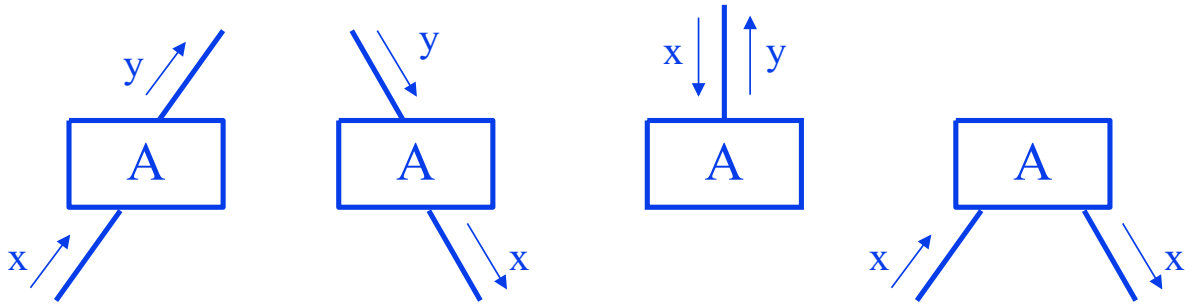
Structure Charts

- For a given DFD there will exist choices as to how it is implemented
- The structure chart notation provides a means of adding more structure to the design
- Each function (or procedure) is represented as a rectangle
- A high-level function is represented as a hierarchy of sub-functions
- Links between sub-functions are labelled with input/output data, *i.e. parameters or shared data*

Tipicamente si ottiene una struttura del design di tipo gerarchico



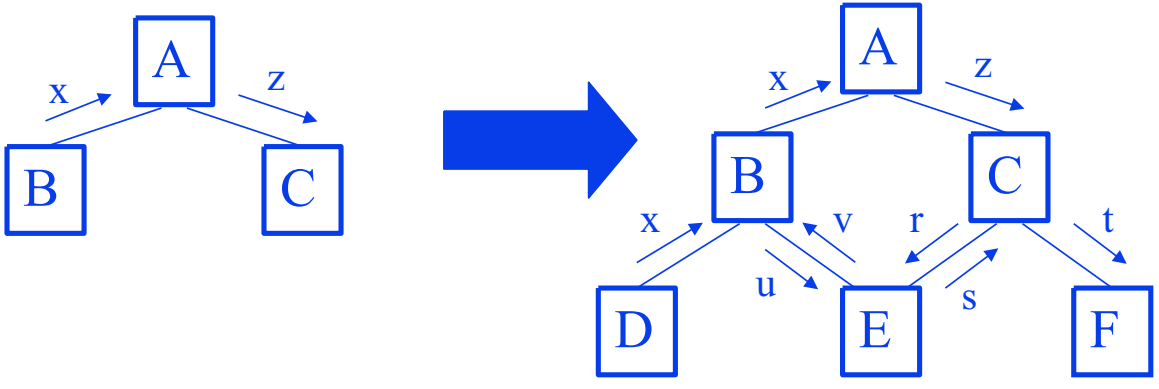
Structural Decomposition (from A. Ireland, HWU)



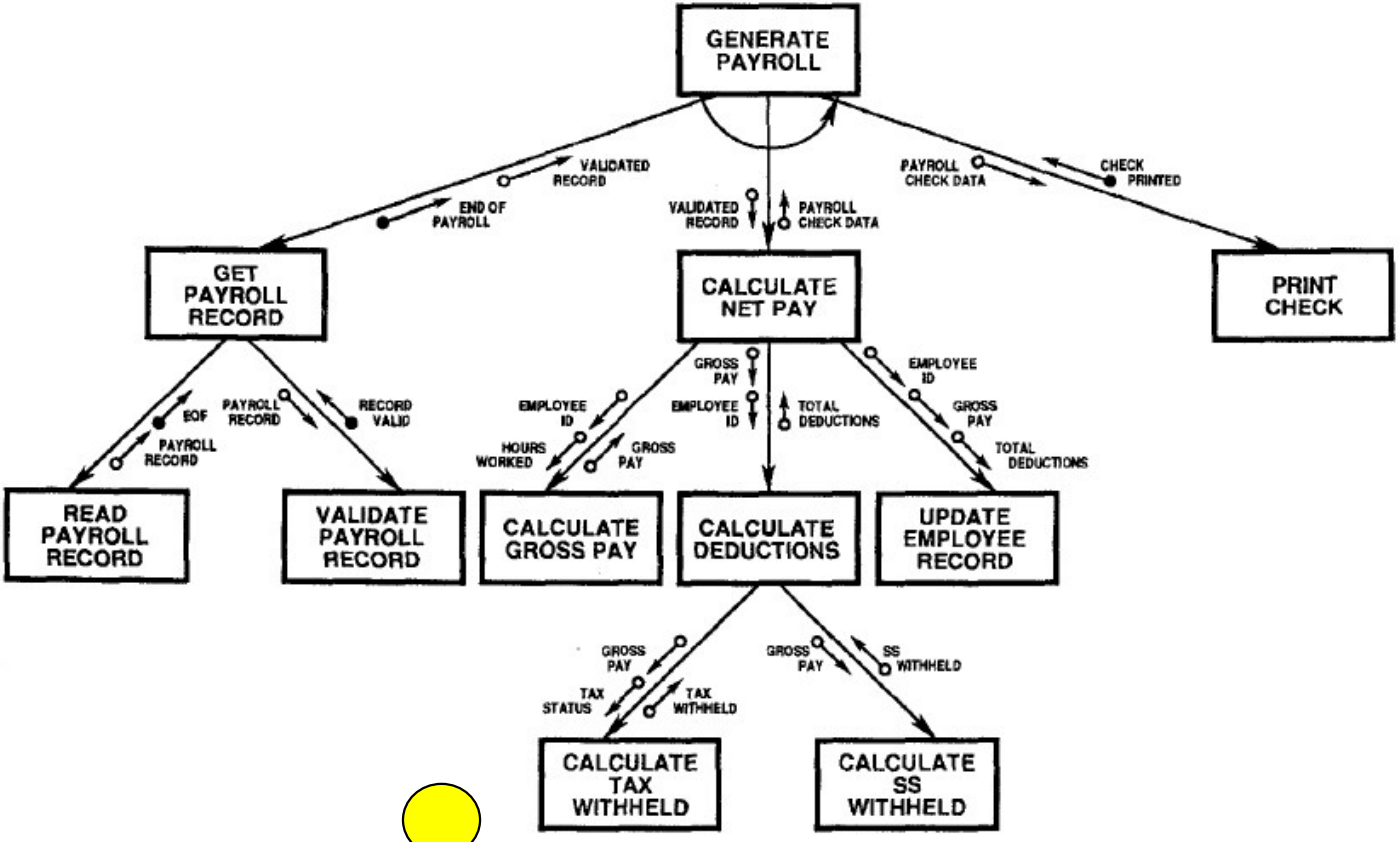
input output transform Co-ordinate

Structural Decomposition (from A. Ireland, HWU)

- Note that a structure chart shows the **functional relationship** between sub-functions, but **not the order** in which they are invoked
- Typically a series of structure charts are developed for a given DFD – *incrementally expanding level by level, e.g.*

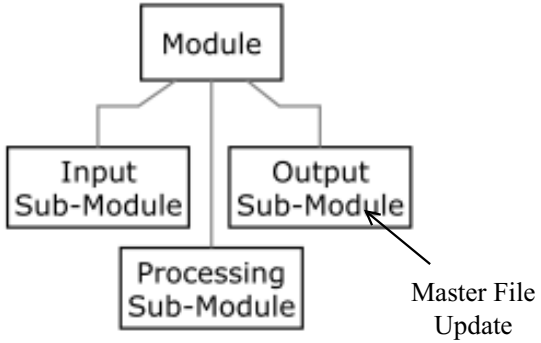


Example of a structure chart



Decomposition guidelines

- For business applications, the top-level structure chart may have 4 functions namely
  - **1.** input,
  - **2.** process,
  - **3.** master-file-update
  - **4.** output



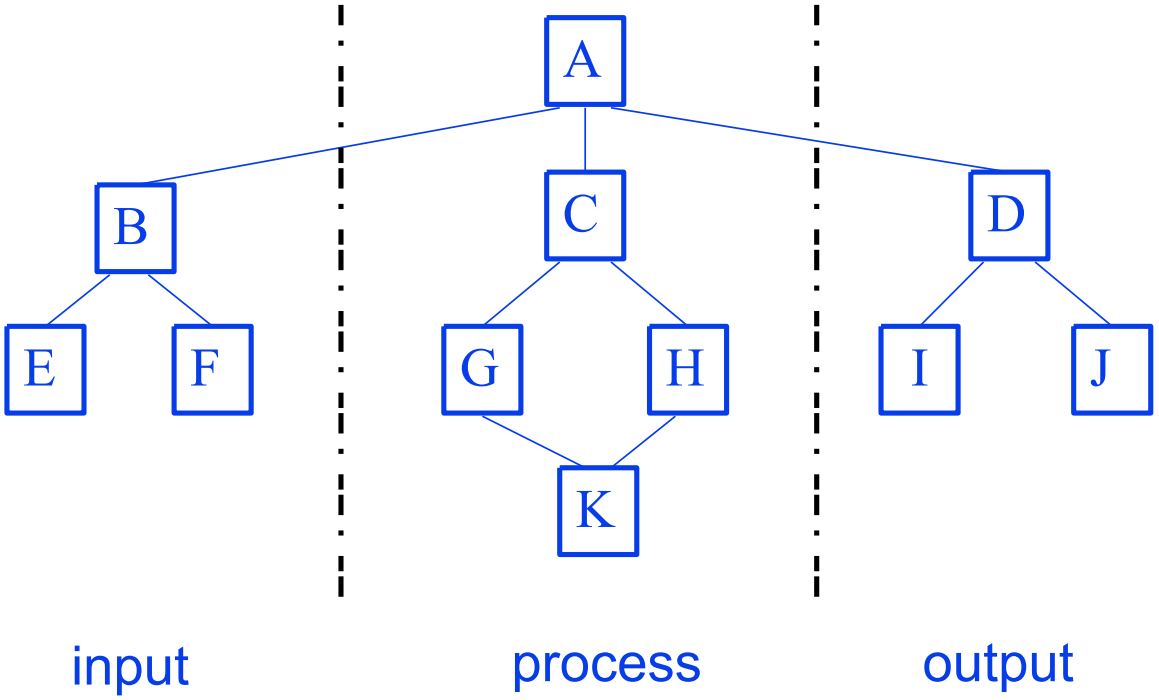
- Data validation functions should be subordinate to an input function
- Coordination and control should be the responsibility of functions near the top of the hierarchy

Decomposition guidelines

- The aim of the deisgn process is to identify loosely coupled, highly cohesive functions. Each function should therefore do one thing and one thing only (**Principio di singola responsabilità)**
- Each node in the structure chart should have between **two and seven** subordinates

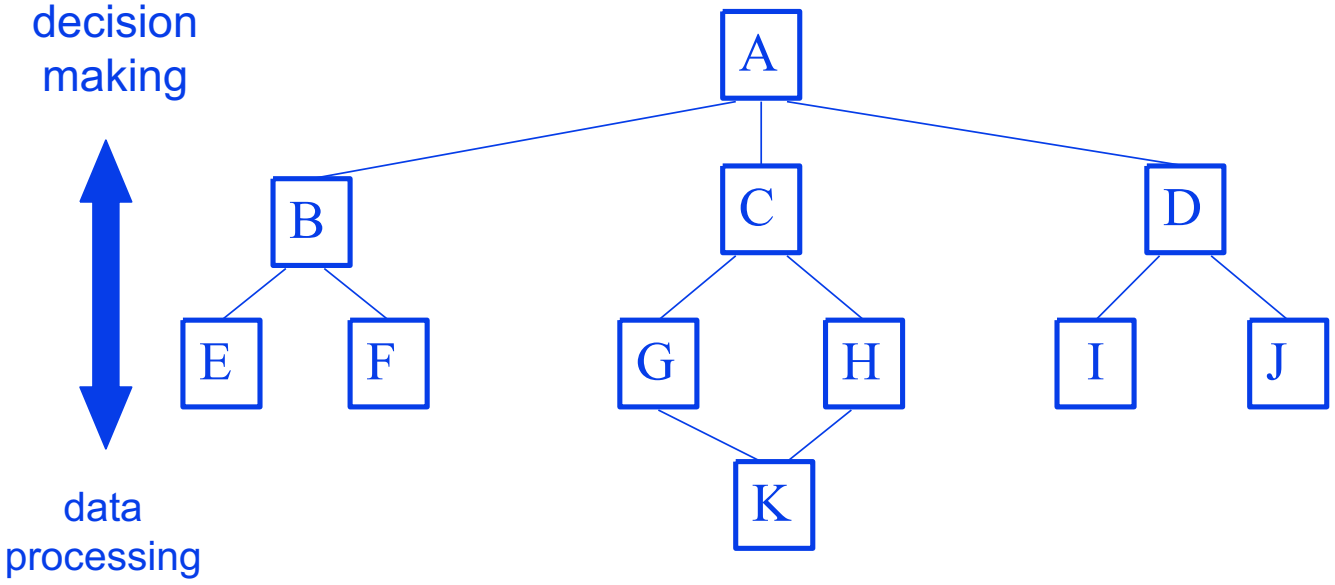
General Structure of a structure chart

(from A. Ireland, HWU)



General Structure of a structure chart

(from A. Ireland, HWU)



Process steps: Transformation Analysis

Processo che trasforma incrementalmente e top down la rappresentazione DFD nello structure chart

1. Identify system processing transformations

   Transformations in the DFD which are concerned with processing rather than input/output activities. Group under a single function in the structure chart

2. Identify input transformations

   Transformations concerned with reading, validating and formatting inputs. Group under the input function

3. Identify output transformations

   Transformations concerned with formatting and writing output, possibly with master file update. Group under the output function

Il passo fondamentale: identifcare la trasformazine centrale

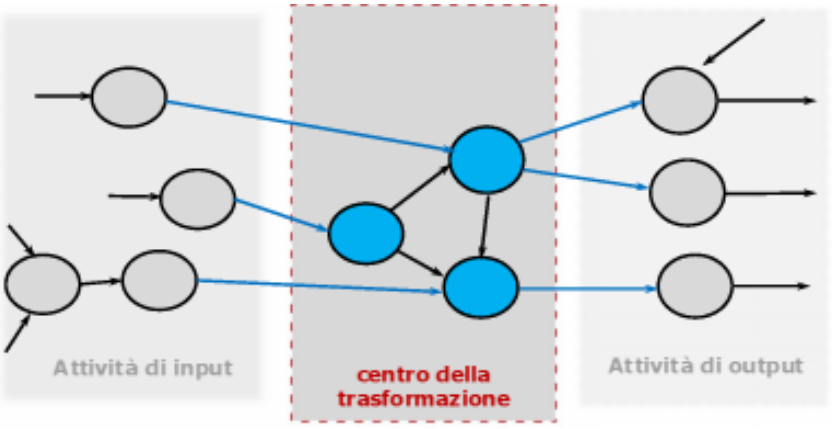
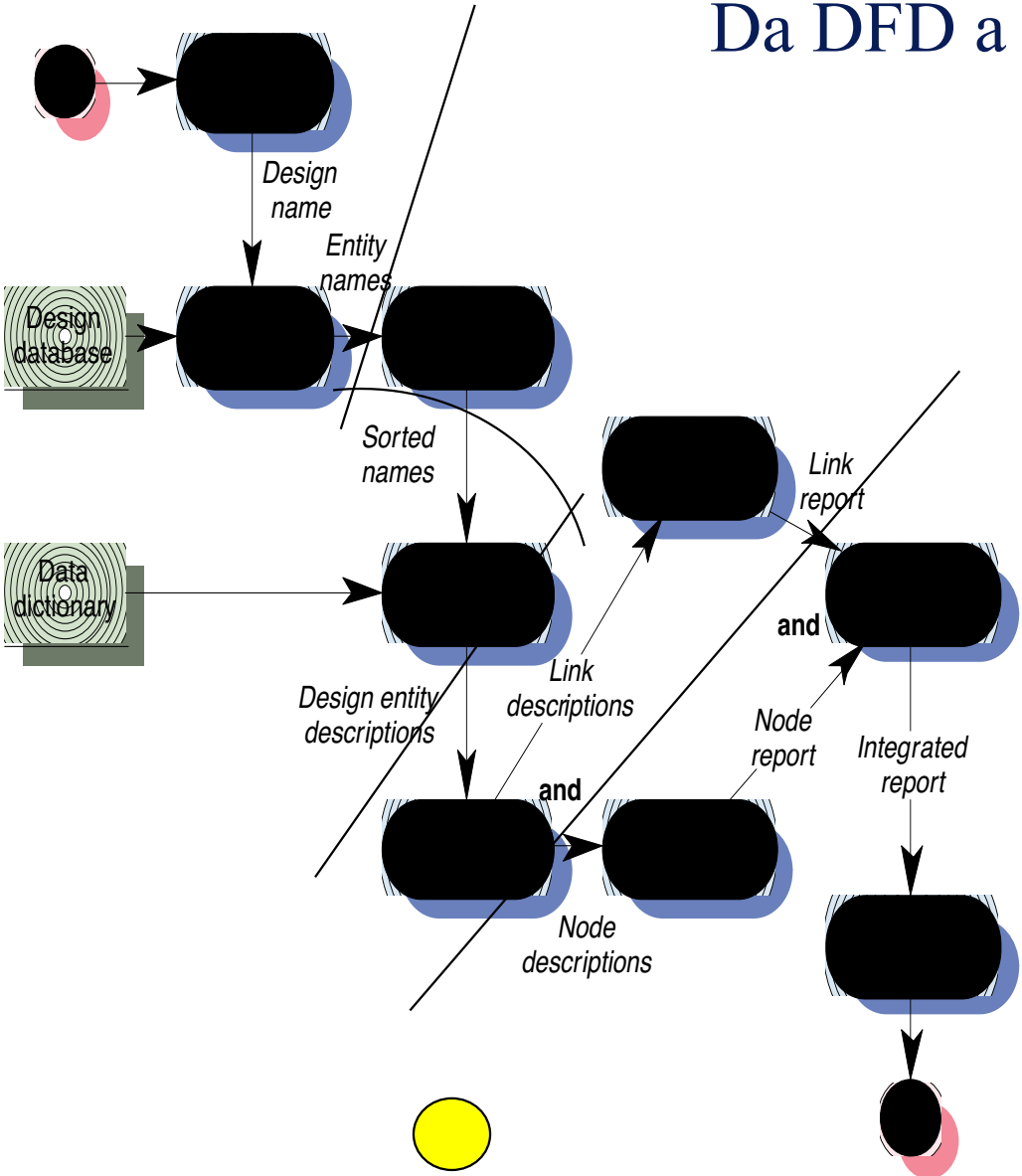
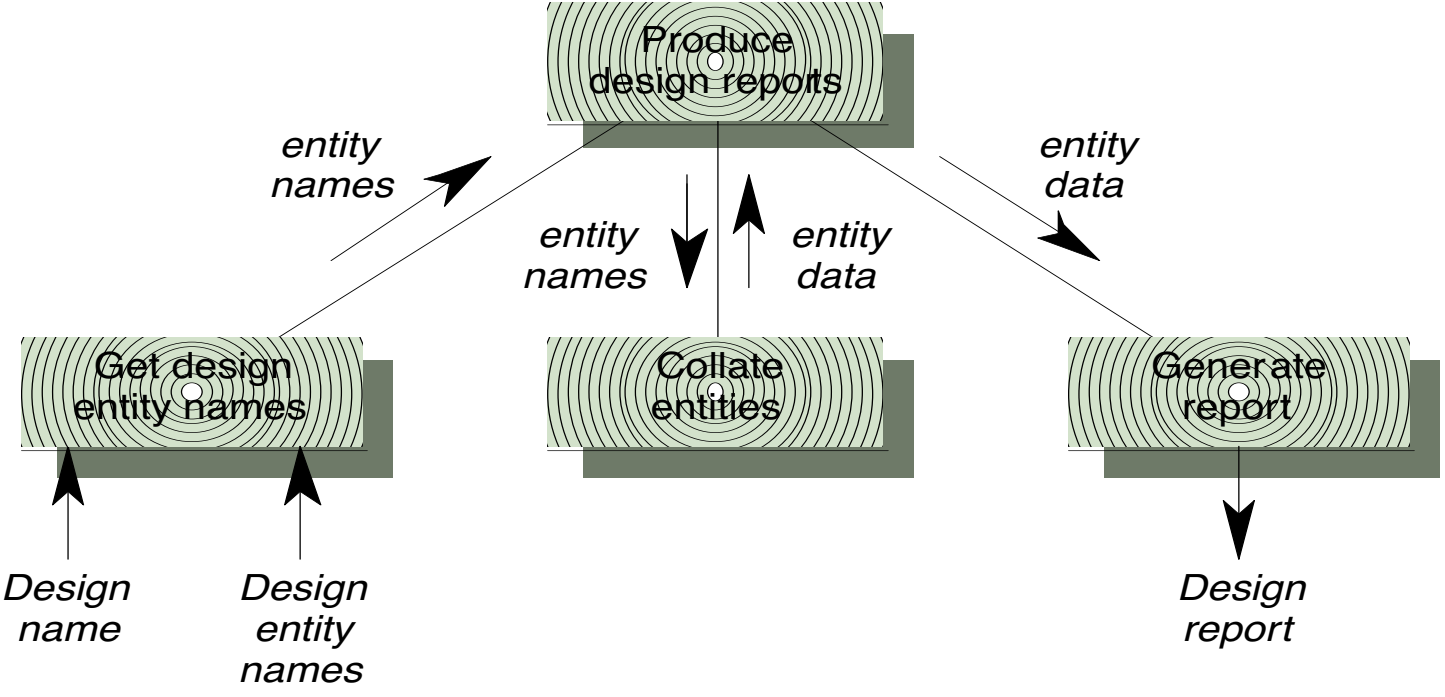


FIGURA 4.19
Partizionamento dei processi in un DFD per identificare la trasformazione centrale

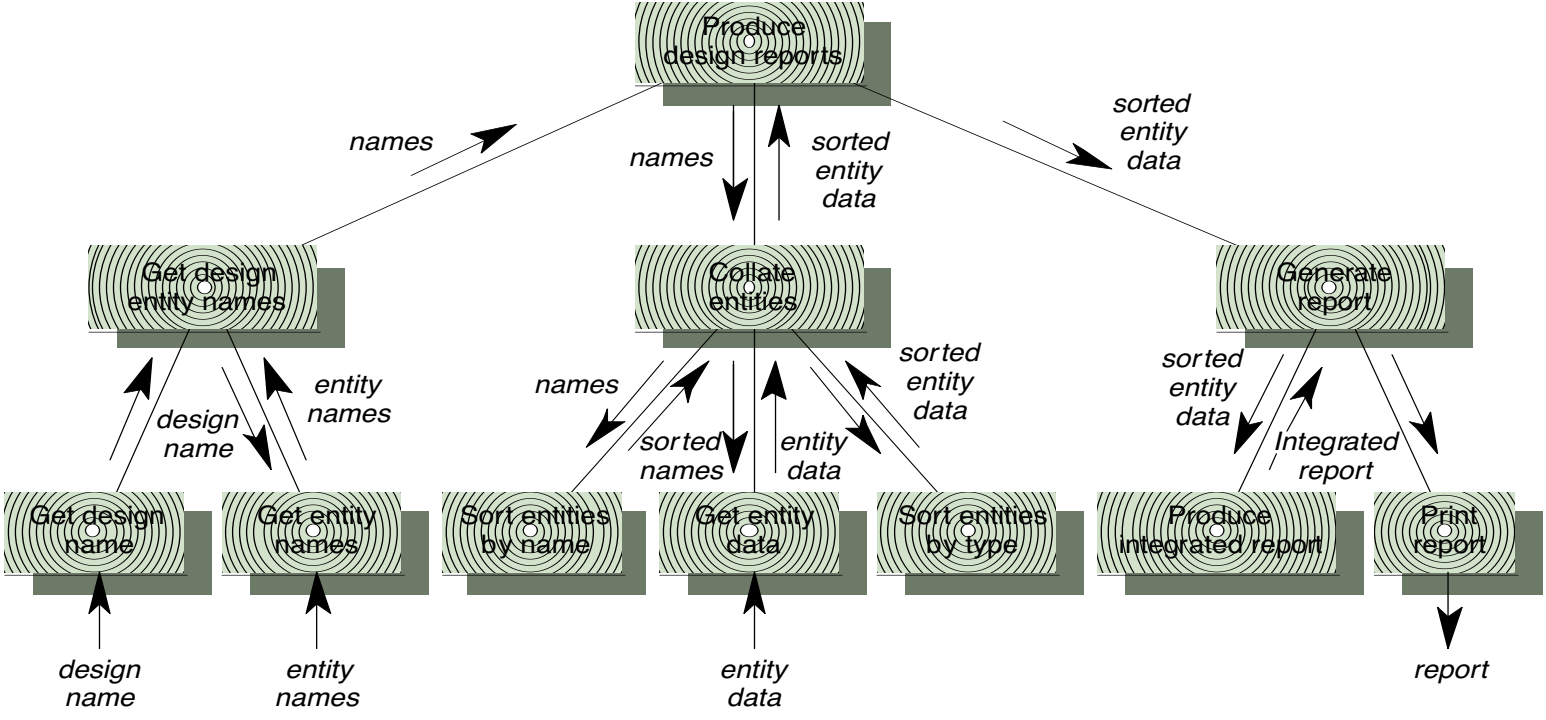
Da DFD a S.Chart: Esempio



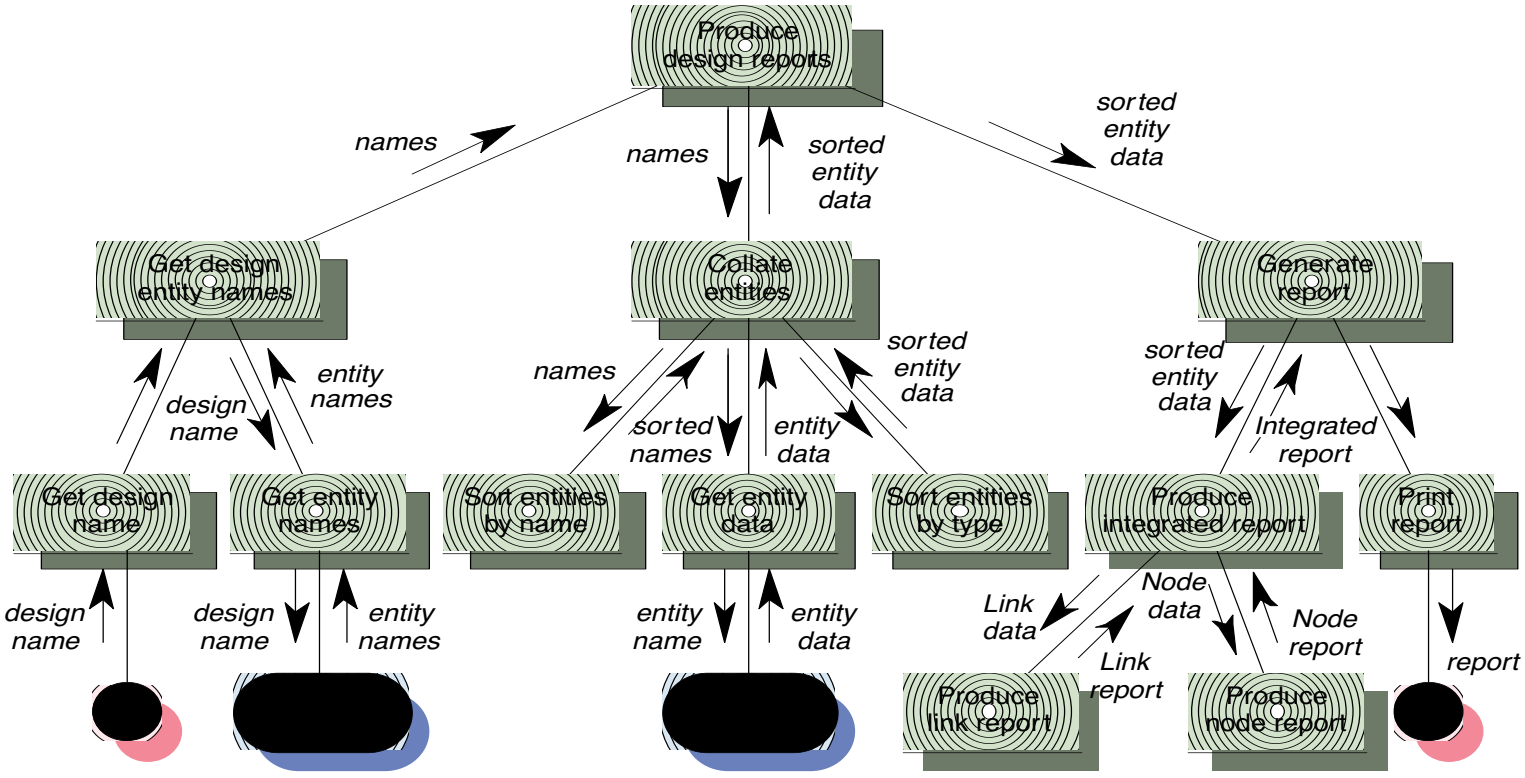
Initial structure chart



Expanded structure chart



Final structure chart



Altro esempio [da Libro Dr. Baruzzo]

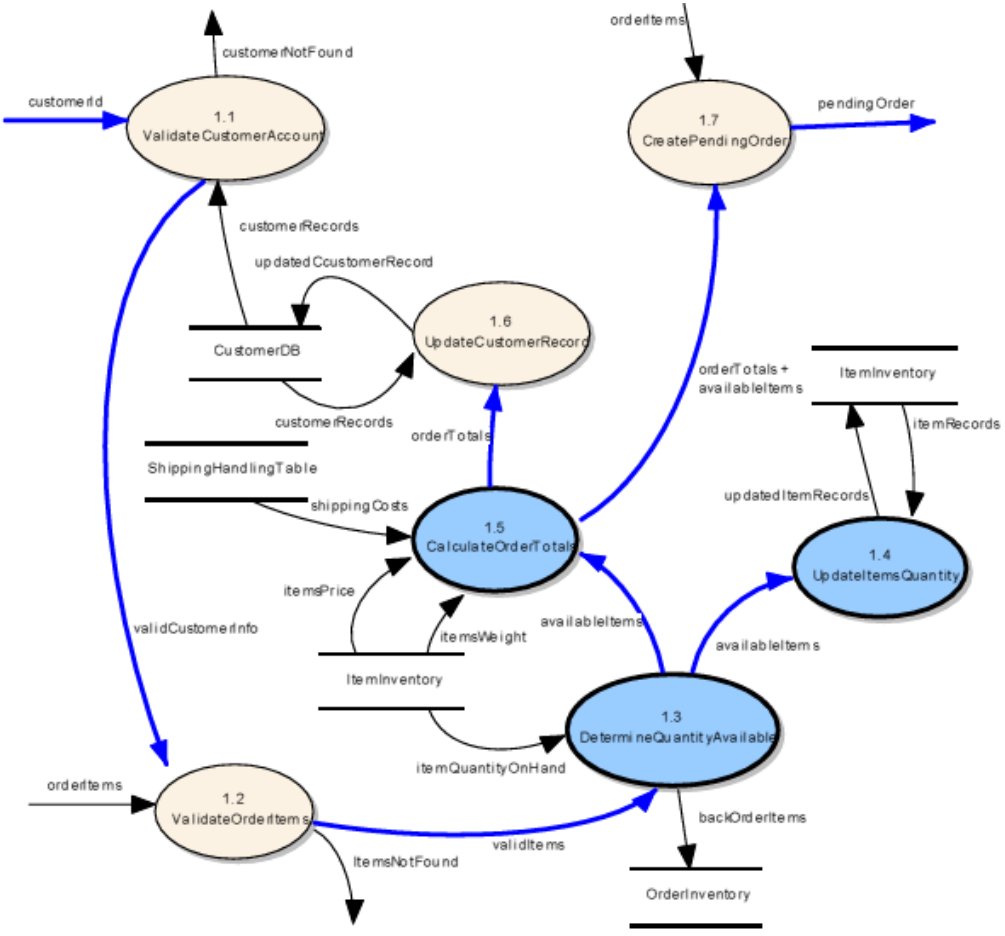
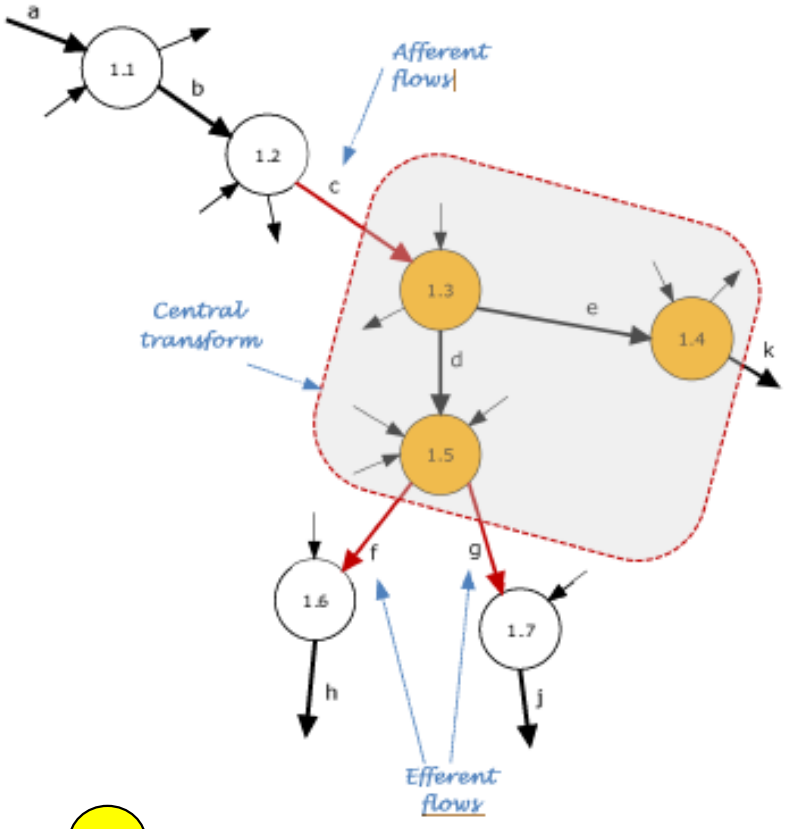
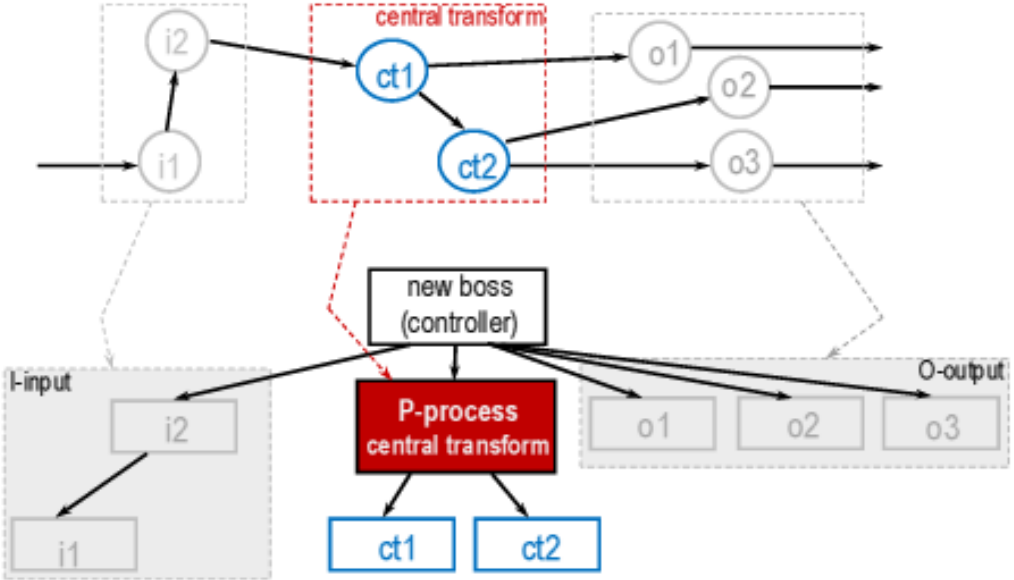


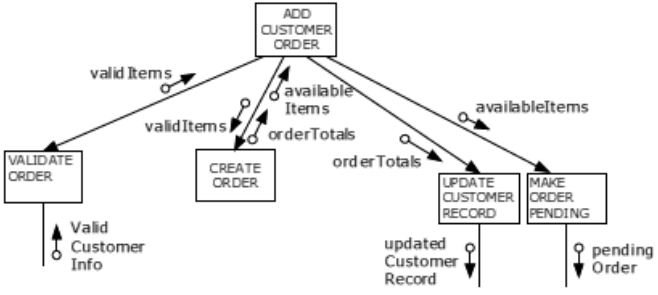
FIGURA 4.20
Diagramma DFD per la registrazione di un ordine in un negozio virtuale online

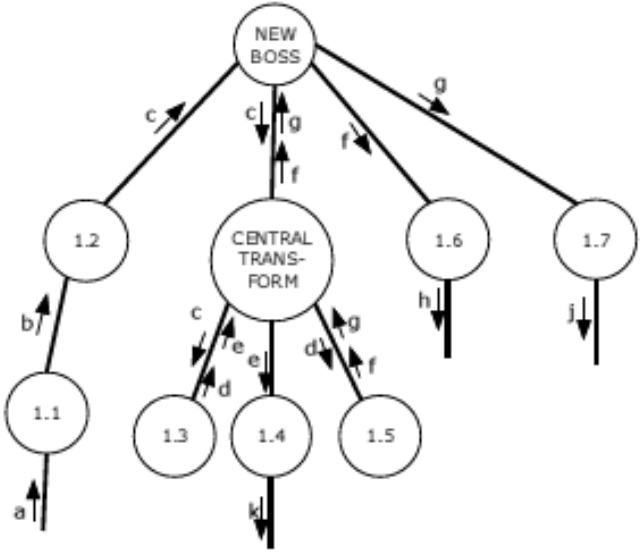
Identificazione della trasformazione centrale



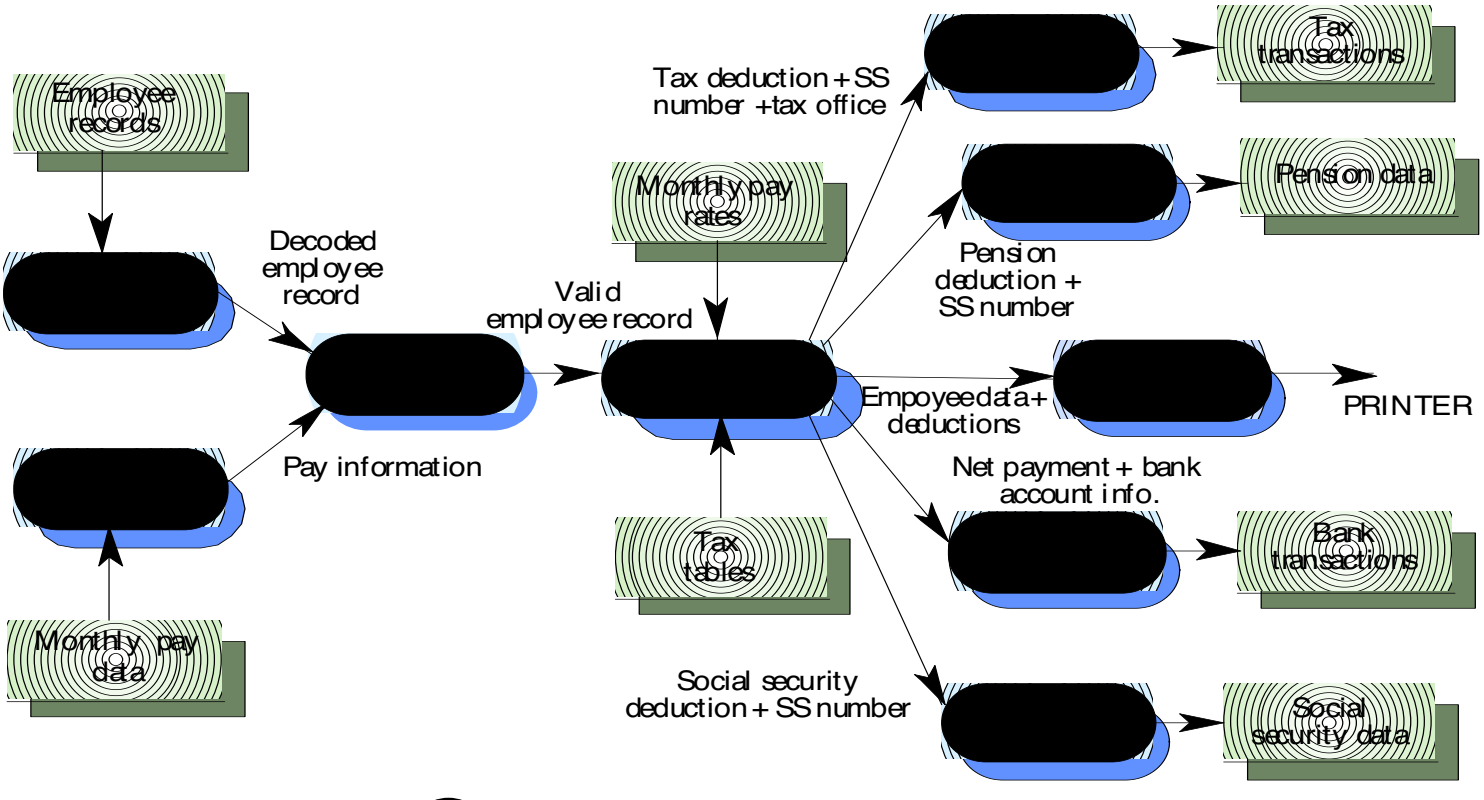
Da DFD a Structure Chart







Payroll system DFD: trasformare in Structure Chart



3. Detailed design (progetto esecutivo)

↳ preparare le informazioni per il programmatore

- The final step
- Concerned with producing a short design specification (**minispec**) of each function. This should describe the processing, inputs and outputs

↳ specifiche per il programmatore

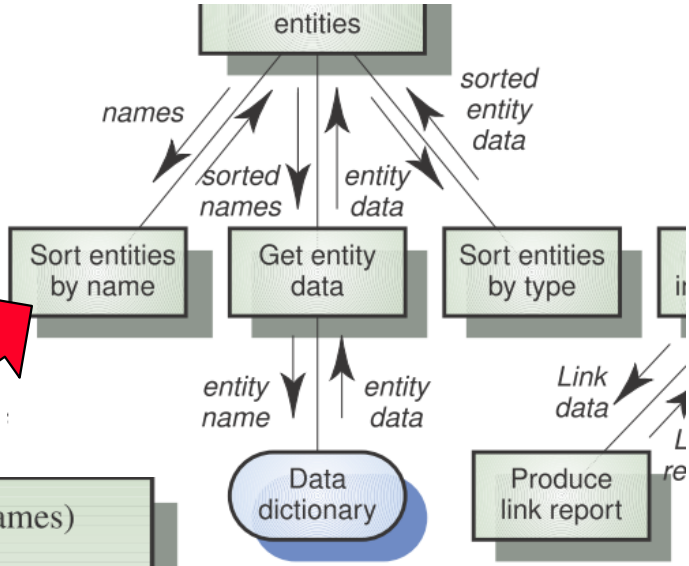
- MINISPEC = progetto esecutivo per ciascun modulo/risultato della progettazione che viene passato ai programmatori per la fase successiva
- These descriptions should be managed in a **data dictionary**
- From these descriptions, detailed design descriptions, expressed in a PDL or programming language, can be produced ([?] delvered to PROGRAMMERS)

Data dictionary entries with mini-spec

Entity name	Type	Description
Design name	STRING	The name of the design assigned by the design engineer.
Get design name	FUNCTION	*Input*: Design name *Function* This function communicates with the user to get the name of a design that has been entered in the design database. *Output*: Design name
Get entity names	FUNCTION	*Input*: Design name *Function* Given a design name, this function accesses the design database to find the names of the entities (nodes and links) in that design. *Output* Entity names
Sorted names	ARRAY of STRING	A list of the names of the entities in a design held in ascending alphabetical order.

MINISPEC

Design entity information



MINISPEC

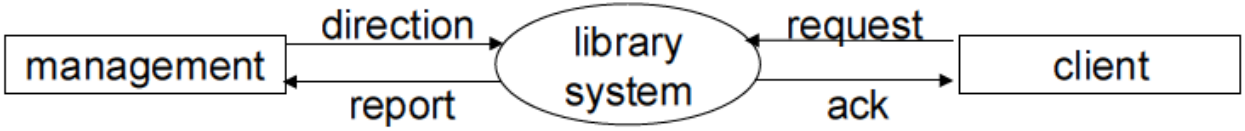
Transform name: Sort entity names (Namelist: in out Names)

Description: This transform takes a list of entity names and sorts them into ascending alphabetical order.
Duplicates are removed from the list.

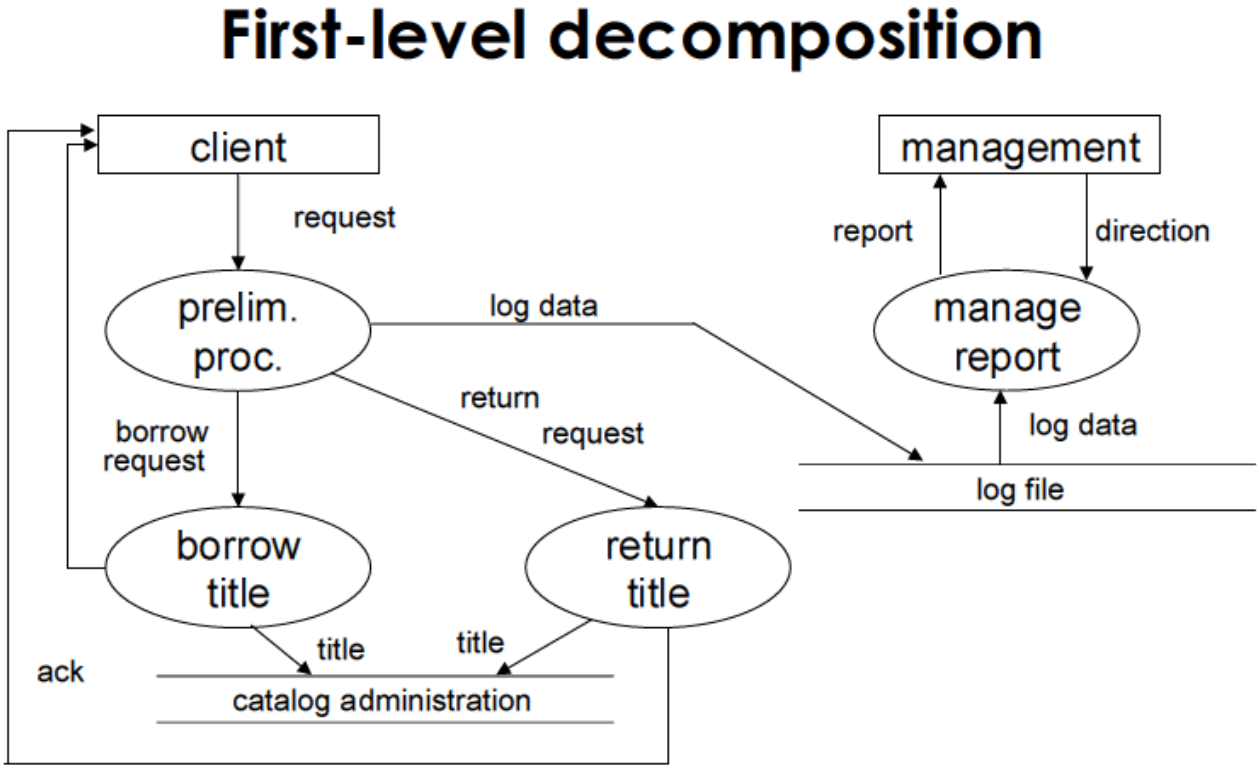
It is anticipated that the names will be randomly ordered and that a maximum of 200 names need be sorted at one time.
A quicksort algorithm is recommended.

Example2: library system

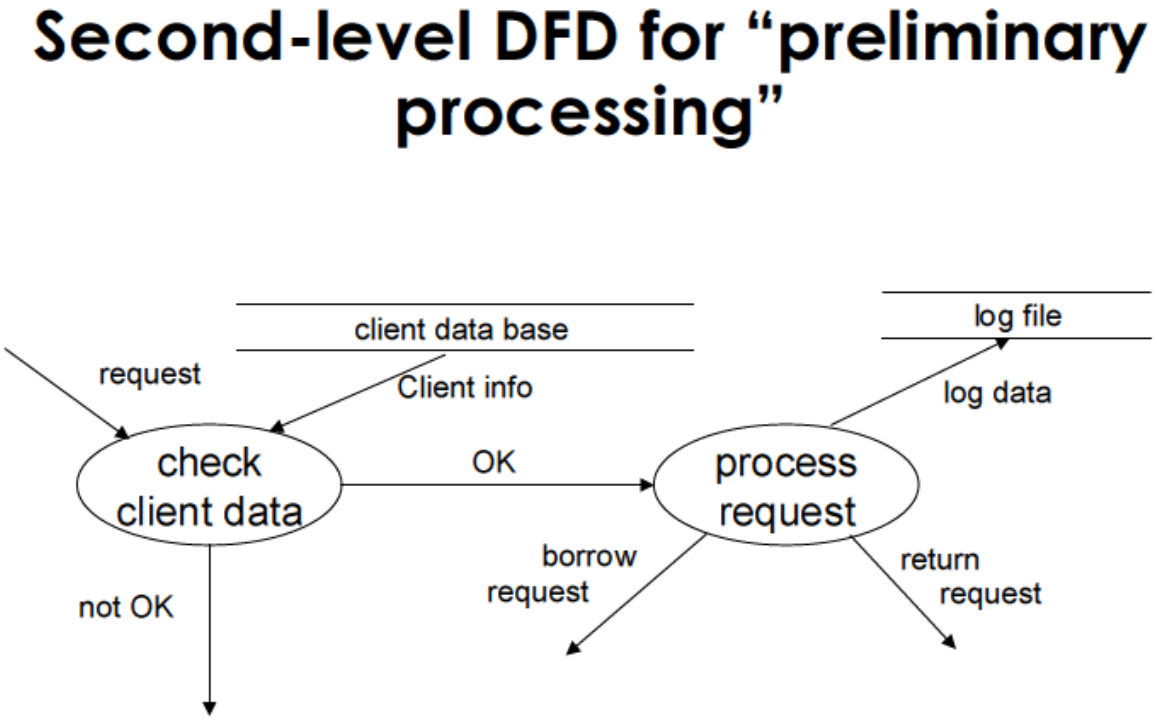
Top-level DFD: context diagram



Example2 (cont.d-1)



Example2 (cont.d-2)



Example2 (cont.d-3)

Minispec of process: «Process Request» **in PDL**

Example minispec

Identification: process request
Description:
1. Enter type of request
1.1. If invalid, issue warning and repeat step 1
1.2. If step 1 repeated 5 times, terminate transaction
2. Enter book identification
2.1. If invalid, issue warning and repeat step 2
2.2. If step 2 repeated 5 times, terminate transaction
3. Log client identifier, request type and book identification
4. ...

Miniscpec in Pseudocodice

Listato 4-2
Specifica funzionale in pseudocodice della procedura di un modulo

```
module NotifyCustomersForUnpaidBills
  /*** scans the unpaid bills file, hence issues notices to */
  /*** customers who are slow in forking out their payments */
     open unpaid bill file, customer details file
     get current date
     repeat
       call get next unpaid bill
         getting unpaid bill, end of unpaid bill file
     until end of unpaid bill file
     set days overdue to current date - bill date
       case days overdue
    > 90:  call generate legal threat using unpaid bill
    61-90: call generate stern warning using unpaid bill
    31-60: call generate gentle hint using unpaid bill
    0-30:  no action
       endcase
     endrepeat
     close unpaid bill file, customer details file
endmodule
```

[vedi Dispensa Dr. Baruzzo]

Data Dictionary & Process Specs

Source: Adapted from Shelbatat, 1990, p262-4

Example Data Dictionary

Mailing Label =
customer_name +
customer_address

customer_name =
customer_last_name +
customer_first_name +
customer_middle_initial

customer_address =
local_address +
community_address + zip_code

local_address =
house_number + street_name +
(apt_number)

community address =
city_name + [state_name |
province_name]

Example Mini-Spec

```
FOR EACH Shipped-order-detail
GET customer-name + customer-address
FOR EACH part-shipped
    GET retail-price
    MULTIPLY retail-price by quantity-shipped
    TO OBTAIN total-this-order
CALCULATE shipping-and-handling
ADD shipping-and-handling TO total-this-order
TO OBTAIN total-this-invoice
PRINT invoice
```

Set 12 - Cosa ricordare: concetti, motivazioni, conseguenze, relazioni fra concetti, ecc.

- Qualità della progettazione; parametri di qualità, progettazione per la manutenibilità; COESIONE: def. e motivazione, Principio SOLIDO di Singola Responsabilità; ACCOPPIAMENTO: def, motivazioni, esempi.
- Approccio Function-Oriented (F-O), inquadramento generale, motivazioni, , caratteristiche, funzioni con 'minimo' stato e non 'history sensitive', modalità di comunicazione: parametri o memoria condivisa, complementarietà dei due approcci O-O e F-O. Processo di progettazione function-oriented, fasi del metodo strutturato, esempio, progettazione esecutiva, data dictionary e minispec, esempi.